AD-A061 821 NEW MEXICO UNIV ALBUQUERQUE ERIC H WANG CIVIL ENGINE--ETC F/G 13/2
AIR FORCE REFUSE-COLLECTION SCHEDULING PROGRAM DESCRIPTION. VOL--ETC(U)
MAY 78 H J IUZZOLINO, E P DUNPHY F29601-76-C-0015
UNCLASSIFIED CERF-EE-20 CEEDO-TR-78-23-VOL-2 NL

1 OF 2
AD
A061821

CEEDO-TR-78-23

AD A061821

DDC FILE COPY

A061369

LEVEL III

②

AIR FORCE REFUSE-COLLECTION SCHEDULING PROGRAM DESCRIPTION VOLUME II: PROGRAM PHASE 2

HAROLD J. IUZZOLINO

ERIC H. WANG CIVIL ENGINEERING RESEARCH FACILITY
UNIVERSITY OF NEW MEXICO
BOX 25, UNIVERSITY STATION
ALBUQUERQUE, NEW MEXICO 87131

MAY 1978

FINAL REPORT FOR PERIOD JANUARY 1976 – APRIL 1977

CEEDO

CIVIL AND ENVIRONMENTAL ENGINEERING DEVELOPMENT OFFICE
(AIR FORCE SYSTEMS COMMAND)
TYNDALL AIR FORCE BASE
FLORIDA 32403

78 11 13 153

Vol 3 A060 986

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
(18) 1. REPORT NUMBER CEEDO-TR-78-23-~~Volume II~~-2	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER (9)
4. TITLE (and Subtitle) (6) AIR FORCE REFUSE-COLLECTION SCHEDULING PROGRAM DESCRIPTION, Volume II, Program PHASE2.		5. TYPE OF REPORT & PERIOD COVERED Final Report, January 1976 - April 1977
		6. PERFORMING ORG. REPORT NUMBER (14) CERF-EE-20
7. AUTHOR(s) (10) Harold J. Iuzzolino, Edward P. Dunphy		8. CONTRACT OR GRANT NUMBER(s) (15) F29601-76-C-0015
9. PERFORMING ORGANIZATION NAME AND ADDRESS Eric H. Wang Civil Engineering Research Facility, University of New Mexico, Box 25, University Station, Albuquerque, NM 87131		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS T.D. 4.03
11. CONTROLLING OFFICE NAME AND ADDRESS DET 1 (CEEDO) HQ ADTC, Air Force Systems Command, Tyndall Air Force Base, FL 32403		12. REPORT DATE (11) May 1978
		13. NUMBER OF PAGES 132
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) (12) 131p.		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Available for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Available in DDC.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Minimum number of trips
Spatial clustering of streets
Shared near neighbors

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes program PHASE2, the second of four programs in the Air Force Refuse-Collection Scheduling Program. Program logic, input, output, and limitations are presented in detail. Some recommendations for changes, a program listing, and sample output are included.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

400 976 1B

PREFACE

This report documents work performed during the period January 1976 through April 1977 by the University of New Mexico under Contract F29601-76-C-0015 with DET 1 (CEEDO) ADTC, Air Force Systems Command, Tyndall Air Force Base, Florida 32403. Captain Robert F. Olfenbuttel managed the program.

This volume, which documents program PHASE2, is the second of four volumes constituting the Air Force refuse-collection-scheduling program description. The sectioning algorithm for program PHASE2 was developed and coded by Edward P. Dunphy. The map-plotting algorithm was developed and coded by Harold J. Iuzzolino.

The report has been reviewed by the Information Officer and is releasable to the National Technical Information Service (NTIS). At NTIS it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

[illegible]
ROBERT F. OLFENBUTTEL, Capt, USAF, BSC
Chief, Resources Conservation Branch

Peter A. Crowley
PETER A. CROWLEY, Maj, USAF, BSC
Director of Environics

[illegible]
EMIL C. FREIN, Maj, USAF
Chief, Envmtl Engrg & Energy Research Division

[illegible]
JOSEPH S. PIZZUTO, Col, USAF, BSC
Commander

LEVEL II

(The reverse of this page is blank.)

TABLE OF CONTENTS

TABLE OF CONTENTS (Concl'd.)

LIST OF FIGURES

LIST OF TABLES

(The reverse of this page is blank.)

SECTION I
INTRODUCTION

1. OBJECTIVES

In designing the Air Force Refuse-Collection Scheduling Program (RCSP), the fundamental objective was to reduce collection costs. The most significant cost reduction is effected by a reduction in the number of collection trips used to service a given region. If a collection crew can be dropped from the fleet, the cost of manpower will be cut. In addition, fuel and maintenance costs will be lessened if the total mileage traveled by the collection fleet can be reduced. The first objective, then, is to generate a collection schedule that calls for the theoretical minimum number of trips. This objective is accomplished in program PHASE2 of the RCSP.

A secondary objective, good spatial clustering of all streets serviced by a vehicle during one trip, is also achieved by PHASE2, except possibly for the last trip. In addition PHASE2 plots maps that show the section (trip) to which each street segment is assigned.

2. SCOPE

This section (Volume II) of the report describes the workings of the second program, PHASE2. A program overview is given, followed by a thorough description of the logic involved in map processing. A skeleton of the logic flow is provided. Input and output files are described. Program requirements and restrictions, error messages and error handling techniques, definitions of import variables, and an estimate of running time are also presented.

SECTION II
PROGRAM OVERVIEW

Determining the minimum number of collection vehicles needed to service a base is fairly simple; the task of assigning collection schedules in such a way that each vehicle is used to capacity, but not overfilled, while travel time and distance are kept close to the minimum, is more difficult.

Program PHASE2 serves two purposes: it assigns street segments to sections (a section is a set of streets to be serviced by one collection vehicle), and it plots the results of the sectioning. The sectioning groups the street segments into closely connected, reasonably convex sets. The size of each section is determined by the capacity of the refuse-collection vehicle that will service it. Therefore, choosing the streets in a section so that each section is compact is the main effort in PHASE2.

Two types of data are used as input to PHASE2. Data describing the nodes and segments are read from files TAPE11 and TAPE9. Card input is used to specify the problem title, the vehicle capacities and numbers, the time limits, the base segment for the first section, and the map bounds.

The program consists of a main program, PHASE2, and 11 subroutines. PHASE2 reads the data cards, the segment data from file TAPE9, and the node data from file TAPE11. Refuse-quantity information included with the segment data and vehicle-capacity data from the data cards are used to determine the number of vehicles required to collect all of the refuse.

PHASE2 calls subroutine BUILD to build a near-neighbor table. The table indicates, for each street segment, the 60 other segments closest to it. To build the table, subroutine BUILD computes the distances from each segment to each other segment. The 60 shortest distances and the corresponding segment numbers are found using an in-core tree-sort algorithm in subroutines SORTK and SIFTUP.

PHASE2 calls subroutine SECTION to assign the segments to sections (corresponding to collection trips). Segments are selected for addition to a section on the basis of the number of near neighbors they share with another segment, called a base segment, already in the section. The first base segment is specified by the user. Subsequent base segments are selected as the sections are built. Segments are added to a section as long as the vehicle capacity is not exceeded.

After each section is filled, subroutine SECTION checks to see whether all remaining unassigned segments will fit into a single, last section. If not, selection continues on the basis of the shared near-neighbor criterion. As sections are completed, the segment numbers are written to file TAPE4, and statistics on vehicle time and capacity are accumulated.

After the sectioning has been completed, PHASE2 calls subroutine PLOTS to initialize the plotting package. Subroutine MAPPLT is called to plot maps indicating the section assignment of each segment. MAPPLT uses subroutine SHAPCOM to set shape parameters for the segments and subroutine COORD to generate coordinates of points on each segment. Subroutine NUMBER is called to append section numbers to the segments. Program PHASE2 terminates after a trip data summary is written to file TAPE1.

The flow of control from one subprogram to another is shown in Figure 1. Within each subprogram, only the first call to each other subprogram is shown. (Three of the subroutines shown in Figure 1--PLOTS, PLOT, and SYMBOL-- are subroutines from the basic Calcomp software package and are not included in the description of program PHASE2.)

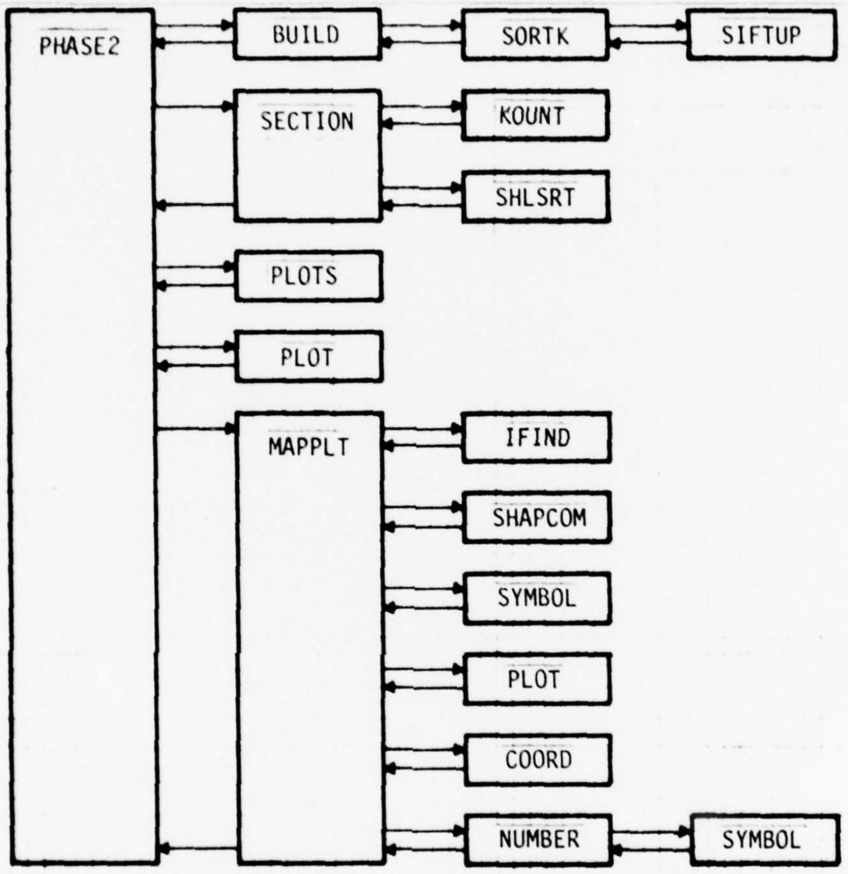


Figure 1. Control Relationships Among Subprograms

(The reverse of this page is blank.)

SECTION III
PROGRAM LOGIC

The logic for program PHASE2 is described from three viewpoints. The first description is task oriented. The second is data-storage oriented and includes discussions of the preparation of data for use by subsequent programs, the use of input data, and the data structures used in PHASE2. The third view describes each subroutine in terms of its purpose and the manipulations performed within it.

1. PROGRAM TASKS

Three tasks are accomplished by program PHASE2: (1) the number of trips is determined on the basis of one of two options available to the user; (2) street segments are assigned (in accordance with vehicle-capacity restrictions) to sections, each section corresponding to a collection-vehicle trip, on the basis of the number of near neighbors each segment shares with some segment already in the section; (3) finally, one or more maps are plotted showing the section assignments of the segments.

Program execution begins in the main program, PHASE2. The problem title is read from the first data card. The number of vehicles and their capacities, the time limits, and the number of the segment that is to be the first base segment are read from the next two data cards. The number of vehicles specified on the data cards determines the method used to generate the number of trips. If enough vehicles are specified to collect all the refuse in the collection region, then that number of vehicles is used, even if it is not the minimum. If fewer vehicles are specified than are needed to service the entire region, the program will assign vehicles in the order given on the data cards until the minimum number needed to collect all the refuse is obtained.

Segment data are read from file TAPE9, and refuse-quantity and node data are read from file TAPE11. The input data and the number of vehicles that will be needed are printed. Subroutine BUILD is called to construct a near-neighbor table. For each segment, BUILD computes the distances to each other segment. The numbers of the other segments are masked into the low-order 12 bits of the distance. The distances are tree-sorted by subroutines SORTK and SIFTUP. The tree sort orders only the specified number of items. The 60 shortest distances are obtained from subroutine SORTK, and the segment numbers are retrieved from these distances. Individual bits corresponding to the segment numbers are set to 1 in an array called a near-neighbor list. The near-neighbor list and information describing the original segment are written to disk. This procedure is repeated for each segment in the map. PHASE2 then calls subroutine SECTION to assign segments to sections.

Subroutine SECTION chooses segments to be added to a section by determining which segments share the most near neighbors with a base segment in the section. Segments are added to the section as long as the vehicle's capacity and time limit are not exceeded. A section is complete when a certain minimum load has been achieved or when each remaining segment would cause the vehicle's capacity or time limit to be exceeded. The minimum-load criterion tries to make the cumulative load at the completion of a section equal to that fraction of the total refuse corresponding to the ratio of vehicle capacity used to total vehicle capacity available. If one or more sections are closed out because each of the remaining segments would cause the vehicle capacity to be exceeded, a situation may occur wherein the determined minimum number of vehicles will be inadequate to collect all of the refuse. In this case additional vehicles are assigned in the order in which the vehicles have been specified, and a message is printed. As each section is completed, the remaining refuse quantity is determined; if the remaining segments can be assigned to one vehicle, the shared near-neighbor testing is discontinued and all of the remaining segments are assigned to the last section.

After subroutine SECTION has completed the sectioning, PHASE2 prints a summary of the loads and times required by the vehicles. A list of the numbers of the segments in each section is also printed.

Subroutine PLOTS is called to initialize the plotting package. Subroutine PLOT is called to place a 3-inch border at the bottom of the plot.

PHASE2 then reads map bounds from the remaining data cards. If no map-bounds cards are found, defaults are set up for a 30- by 30-inch map.

Subroutine MAPPLT is called once per output map. MAPPLT examines sequentially the original segment data, skipping segments that are outside the map bounds and drawing segments that lie at least partially within bounds. Before each segment is drawn, subroutine SHAPCOM is called to set up parameters used to determine the position on the segment of points at a given distance from the start of that segment. The actual coordinates of the points on the segment are returned by subroutine COORD.

Four different computations are used by subroutines SHAPCOM and COORD to produce the coordinates of points on a segment. The simplest computation is performed for straight segments and involves a linear interpolation between the initial and final nodes. Another calculation processes both circular-arc and S-curve segments; an S-curve is treated as two consecutive circular arcs. In calculating the coordinates of a point on a rectangular segment, the slope components of the first side are determined; appropriate multiples of these components are then added to the starting or ending node's coordinates. The coordinates of a point on an angle segment are found by linear interpolation between one end of the angle and the vertex. (A full description of the geometry, as well as relevant calculations, are given in Section III of Volume I of this report. The scale ratio SCR is replaced by 1.0 in program PHASE2.)

2. DATA STORAGE

Program PHASE2 obtains data from three sources: card input, file TAPE9, and file TAPE11. Two files, TAPE1 and TAPE4, are generated by program PHASE2 and are saved on disk for use by program PHASE3. Files TAPE2, TAPE3, TAPE7, and TAPE10 are used as scratch files.

The card data, the data from TAPE9, and the data from TAPE11 are read at the beginning of PHASE2. The segment data read from TAPE9 are stored in arrays in blank COMMON. The street number and the number of ways of travel on TAPE9 are not retained in core. The node data from TAPE11 are stored in arrays in labeled COMMON block NODATA. If the number of vehicles from the second data card is zero for any vehicle, it is reset to 1 in the loop on statement 25. The numbers of vehicles, their capacities, and their time limits are moved to the TRUCKS array in the loop through statement 50. The amount of total refuse is also accumulated in this loop. When subroutine BUILD is called to build the near-neighbor table, segment numbers in array ISTPR and street segment midpoints in arrays X, Y, XT, and YT are sent through the argument list. Subroutine BUILD writes to TAPE7 the segment number, refuse quantity, travel and collection time, number of houses, and 26 words of near-neighbor information for each segment.

When PHASE2 calls subroutine SECTION, the vehicle data in array TRUCKS, which are grouped by vehicle capacity, are expanded into array TRUCK so that each line of array TRUCK corresponds to a single vehicle. As the first base segment is sought in the loop through statement 17, segments other than the base segment are written to TAPE1. TAPE1 will contain unassigned segments and their near-neighbor lists. As each segment is considered for addition to a section, its segment and neighbor data are written to TAPE2 if it is not added to the section. As a segment is assigned to a section, its segment and neighbor data are written to file TAPE3. When a section is completed, the segment numbers are read from file TAPE3 and written to file TAPE4. The unassigned segments on file TAPE1 are recopied to file TAPE2. When all of the segments have been assigned to sections, control returns to program PHASE2. At the end of program PHASE2, the number of segments and the number of sections are written to TAPE1, as are pointers to the first and last segment numbers on file TAPE4 and the vehicle capacity for each section.

3. PURPOSE AND PERFORMANCE

In this section the simpler subroutines are described first so their workings will be clear when they are mentioned again in the descriptions of the more complicated subroutines and, finally, of the main program. Logic flowcharts are given in Appendix A. Complete program listings are provided in Appendix B. In Appendix C, the more important variables mentioned in the following descriptions are defined in terms of their specific meaning for each subroutine.

a. Function KOUNT

The purpose of function KOUNT is to count the 1 bits in a 60-bit word. The function is written in the COMPASS assembler language for the CDC 6600.

Function KOUNT has one argument. The argument is a bit pattern with bits set to 1 where two segments share a near neighbor. The value returned is the number of 1 bits in the argument. The sequence of instructions generated by the compiler where KOUNT is called includes setting register A1 to the address of the argument list. In KOUNT, the first SA1 instruction causes the X1 register to receive the address of the first argument. The second SA1 instruction causes the value of the argument to be placed in register X1. The CX6 instruction counts the 1 bits in register X1 and places the result in register X6. Control then returns to the calling program.

b. Subroutine SHLSRT

Subroutine SHLSRT sorts one array into decreasing order and carries a second array along during the sorting. The algorithm used is Shell's sorting algorithm.

Subroutine SHLSRT has three arguments. The first argument is the array to be sorted. The second argument is an array that is paired with the array to be sorted and is rearranged as the first array is sorted. The third argument is the number of words to be sorted.

The statements up to statement 60 arrange array X in increasing order. The numbers are sorted by a procedure in which pairs of numbers are compared and interchanged if necessary to put the smaller number closer to the beginning of the array. The separation of the numbers compared is approximately one-half the number of entries in the array; this spacing is halved in subsequent passes through the array. When two numbers are interchanged, the pointers are moved up so that the smaller number is compared to a number farther up in the array. The spacing (N) is set initially to one-half the number of words. The number of comparisons (K) to be performed in the loop through statement 50 is computed as the total number of words less N.

The loop through statement 50 uses index I as one of the pointers. This pointer is sorted in variable J. The other pointer, L, is set equal to I+N. The values of the array to be sorted and the array to be carried along are saved as XT and AT. The values of X at the locations indicated by the pointers are compared; if they are in order, control transfers to statement 40. If not, the larger value is stored closer to the end of the array. The pointers are both moved up by N; if the smaller valued pointer is a valid subscript, control transfers to statement 20, where another comparison is performed.

At statement 40 the saved values are stored in the appropriate place in the arrays. When the loop through statement 50 is completed, if the spacing is equal to 1, the sort is complete and control transfers to statement 60. Otherwise, the spacing is halved and control transfers to statement 10.

The loop through statement 70 rearranges the arrays so that the X-array is in decreasing order. Control returns to the calling program.

c. Subroutine SIFTUP

Subroutine SIFTUP orders each subtree in a binary tree from a given subroot up to the root so that each subroot is smaller than either of its branches. Subroutine SIFTUP has four arguments. The first is the subscript of the subroot at which sorting starts. The second is the number of items in the array to be sorted. The third is the array to be sorted. The fourth is the dimension of the array to be sorted.

Variable I is set equal to the subscript of the subroot where sorting will start. Variable I will continue to point to a subroot throughout the sorting. The value at TREE(I) is saved in variable COPY.

At statement 10, pointer J is set equal to the subscript of the left branch. If J points to or beyond the last item in the tree, control transfers to statement 6. Otherwise, the left and right branches are compared. If the right branch is smaller, J will be incremented so that it points to the right branch.

At statement 4 the smaller branch is compared to the root. If the root is smaller, control transfers to statement 6. Otherwise, control resumes at statement 5, and the branch value is stored in the root position. The branch from which the smaller number came now becomes the root in another iteration. Control transfers back to statement 10.

At statement 6, the value of the root saved in variable COPY is stored in the appropriate place in array TREE. Control returns to the calling program.

d. Subroutine SORTK

Subroutine SORTK returns the KN smallest numbers in array TREE. A tree-sort algorithm is used. The array to be sorted is treated as a binary tree and is partially ordered, in such a manner that each subroot is smaller than its branches, by calls to subroutine SIFTUP.

Subroutine SORTK has four arguments. The first (N) is the number of items in the array. The second (KN) gives the number of items to be returned. The third (TREE) is the array to be sorted. The fourth is the dimension of the array.

Subroutine SORTK begins by comparing the first and last numbers in the array to be sorted. If the last item is smaller than the first, the two are interchanged to prevent the smaller number from being trapped as the last entry in the array when the number of entries is even and the last entry is the smallest. Variable K is set to one-half the number of items in the tree.

The loop through statement 10 calls SIFTUP; the first argument starts at the middle of the tree and works back to the second element in the array. When this loop is complete, the branches in the tree are smaller than any subroots except the root of the entire tree.

The loop through statement 11 causes SIFTUP to move the smallest number to the root of the tree. This number is then exchanged for the last item in the tree. The loop through statement 11 is used once for each number to be returned. When this loop is complete, the KN smallest numbers will be at the end of array TREE, and the smallest number will be last. Control then returns to the calling program.

e. Function IFIND

Function IFIND uses a binary search to locate a given number in an array; the subscript corresponding to the location of the number is assigned as the value of IFIND. If the number is not found, the function sets the value of IFIND equal to the negative of the subscript at which the number, to be in numerical order, should be inserted. (The array is assumed to be in increasing order.)

The comment cards at the beginning of function IFIND list the latest changes to the function and state the function's purpose.

Argument NUM is the number that is sought in array IARRAY. The length of array IARRAY is given by argument LEN. Function IFIND begins by checking that LEN $>$ 0. If LEN $\leq$ 0, the function assigns a value of -1 to IFIND. This value indicates that the number sought is not in the array and would be stored as the first entry in the array. The binary search uses variables II, IP, and IF as pointers. II is the subscript of the front of the region being searched, IP is the subscript of the item being compared to the number sought, and IF is the subscript of the last item in the region being searched. Variable II is initially set to 1 at statement 5, and variable IF is set to the end of the array in the next statement. The pointer, IP, is the subscript about midway between II and IF.

The computation of IP occurs at statement 10. The statement following statement 10 compares the number being sought, NUM, to the data at IARRAY(IP). If NUM $<$ IARRAY(IP), control transfers to statement 20, indicating that the number is in the front one-half of the region being searched; at statement 20 the final pointer is moved to the subscript preceding the point just searched. If NUM $>$ IARRAY(IP), control transfers to statement 30, indicating that the number being sought follows the subscript just inspected. At statement 30 the initial pointer, II, is set to the present pointer, IP, plus 1.

If the number sought is found at IARRAY(IP), control transfers to statement 50, where IFIND is set equal to the current pointer and control returns to the calling program. Where NUM is unequal to IARRAY(IP), control resumes at statement 40 after the initial or final pointers are moved. At statement 40 the final pointer is compared to the initial pointer; if IF $\geq$ II, control is transferred to statement 10.

At statement 10 the search is resumed on the appropriate one-half of the region examined previously. If the final pointer becomes less than the initial pointer, the number sought is not in the array. In this case, control resumes following statement 40, and the value of IFIND is set to the negative of the current pointer. If the number at the current pointer is less than the number being sought, IFIND is set to -(IP + 1) so the number can be inserted in the appropriate place. Control then returns to the calling program.

f. Subroutine NUMBER

Subroutine NUMBER appends numbers to plotted output. Its purpose is almost identical to that of the standard Calcomp number routine, the primary difference being that the last argument in subroutine NUMBER gives an alphanumeric format rather than an integer format code.

Subroutine NUMBER has six arguments. The first two give the coordinates, in plotter inches, of the lower left corner of the field. The third gives the height, in inches, of the digits. The fourth is the number to be plotted. The fifth is the angle at which the number is to be plotted, measured in degrees counterclockwise from the horizontal. The last argument is an alphanumeric format up to 10 characters long, which describes the appearance of the plotted number.

Array TEXT is used to hold the character representation of the number. Up to 30 characters are allowed. The first executable FORTRAN statement sets this array to three words of blanks. The second statement moves the format into the second word of array FORM. The first and third words of this array have been preset to a left and a right parenthesis by a DATA statement. The ENCODE statement converts the number from binary form in variable NUM to character form in array TEXT, according to format FORM.

A character count, variable NC, is set to 30. The loop through statement 10 searches for the last non-blank character in array TEXT. Each time a blank is found, starting at the end of the TEXT array, the character count (NC) is decremented by 1. When a non-blank character is encountered, control transfers to statement 20. Statement 20 calls the standard SYMBOL subroutine to plot the character representation of the number. Control then returns to the calling program.

g. Subroutine SHAPCOM

Subroutine SHAPCOM sets up parameters in COMMON block COPARM that describe the geometrical properties of a segment. These parameters are used by subroutine COORD to produce the coordinates of points on a segment.

Subroutine SHAPCOM has two arguments. Argument TOTLEN gives the total length of the segment, in miles. Argument AVMD gives the number of miles per map coordinate unit (MCU) on the first map input to program RCINPT. The values of the arguments are sent to subroutine SHAPCOM, and all output values from SHAPCOM are placed in COMMON block COPARM.

In COMMON block COPARM, variable SF indicates the shape of the segment. XNI and XNF are the x-coordinates of the initial and final nodes of the segment. YNI and YNF are the y-coordinates of these nodes. SX and SY are the slope, in MCU per mile, in the x and y directions. RPR is the reciprocal of the radius of curvature for circular segments and the circular portions of S-curves. C11 and C12 are the position differences, in MCU, of the starting point and center of a circular arc or of the first one-half of an S-curve. XCTR and YCTR are the center coordinates, in MCU, for a circular arc or one-half of an S-curve. BR1 is the distance in miles from the start of a segment to some particular point on that segment. It is not used for straight segments. For circular segments, BR1 is the total perimeter. For an S-curve, BR1 is the perimeter to the midpoint of the S-curve. For a rectangular segment, BR1 is the distance to the first bend in the rectangle. For an angle, BR1 is the distance to the vertex. BR2 is defined only for rectangular segments and angles. For a rectangular segment, it is the perimeter in miles from the start of the segment to the second bend. For an angle, BR2 is the length of the second side. SGN is -1 for shapes involving the L (left) prefix, and +1 otherwise.

Subroutine SHAPCOM begins execution by assuming that the shape code indicates a straight line. Break indicators BR1 and BR2 are set to 0. DX and DY, the x- and y-components of the vector from the initial to the final node on the segment, are computed. The x- and y-components of the slope of the vector, measured in MCU per mile, are computed and stored in SX and SY. The shape code is tested; if the segment proves to be a straight line or is not to be plotted, the subroutine returns control to the calling program. For any other shape code, execution continues. The angle of the vector from the starting to the stopping node is computed as variable THETA. The distance from the starting to the stopping point, D, is computed in miles. If the shape code indicates a shape other than circular or S-curve, control transfers to statement 60.

At statement 45 the coordinates of the final node are stored in variables XE and YE. The first break, BR1, is set to the total length of the segment. Variable DD is set to the straight-line distance from the starting to the stopping node. If the shape code indicates a circular segment, control transfers to statement 50. If not, variables XE and YE are reset to the coordinates of the midpoint of the S-curve. Break indicator BR1 is reset to the perimeter length from the starting point to the center of the S-curve. Variable DD is set to one-half the distance from the starting to the stopping point.

At statement 50, SGN is set to 1. If the shape code indicates a left circle or left S-curve, SGN is reset to -1. Variable V is set equal to 1-D/TOTLEN. VS is the square of V. The reciprocal of the radius of curvature of the circle or the circular portion of the S-curve is evaluated using a polynominal approximation to the solution from a transcendental equation containing the reciprocal of the radius of curvature. The approximate radius of curvature, RPR, is improved by a series of linear interpolations if the value for RPR causes an error greater than 0.00001 in the transcendental equation

$$\sin \frac{BR1*RPR}{2} = \frac{DD*RPR}{2}$$

When RPR is within the desired accuracy, control resumes at statement 51. The radius of curvature, R, is computed. A temporary variable, ARG, is evaluated. The height of the center of the circle from the line connecting the starting and stopping points, H, is set to 0. If variable ARG is greater than 0, H is recomputed. The distance to the first break, BR1, is tested to determine whether the circular arc is greater than one-half a circle. If so, the sign of the height is changed. The x- and y-coordinates of the center of the circle are computed. The components of the vector from the center to the starting point, C11 and C12, are computed. All variables needed to compute points on the S-curve or circle are now available, so control returns to the calling program.

Processing continues at statement 60 for the remaining shape codes. At statement 60, the shape code is tested; if neither a right nor a left rectangle is indicated, control transfers to statement 80. Otherwise, for a rectangular segment, the distance from the start to the first bend, BR1, is computed. If this distance is greater than 0.05 of the total length, control transfers to statement 70. Otherwise, the rectangle is assumed to be so shallow that a straight-line approximation is adequate, and the shape code is set to 0. Control then returns to the calling program.

At statement 70 the perimeter to the second bend in the rectangle, BR2, is computed. SX and SY, the x- and y-components of the slope of the vector from starting point to stopping point, are computed, and control returns to the calling program.

The only segments that reach statement 80 are the angles. The sign of the shape code is retrieved in variable SGN, and the distance from the starting node to the vertex of the angle is retrieved as the magnitude of the shape code and is stored in variable BR1. The length of the second leg of the angle is computed and saved in variable BR2. If the angle is incorrectly specified so that it is actually a straight segment, a round-off error may occur in the computation of ARG, the square of the distance from the vertex to the line connecting the endpoints. If ARG is zero or negative, control transfers to statement 100. Otherwise, the x- and y-coordinates of the vertex are computed, and control returns to the calling program.

At statement 100, the shape code and break indicators are set to 0, indicating a straight segment. Control returns to the calling program.

h. Subroutine COORD

Subroutine COORD is given a distance, in miles, from the beginning of a segment and returns the coordinates in MCU. Parameters describing the segment to be processed have been sorted in COMMON block COPARM by subroutine SHAPCOM before COORD is called. Argument CUMLEN is the cumulative length along the string, in miles; arguments XX and YY are the coordinates returned for a point CUMLEN miles from the start of the segment.

The first statement of COORD sets S equal to the cumulative length. If the shape code is nonzero, control transfers to statement 10. The coordinates of the point on a straight-line segment are computed and returned in variables XX and YY. Control returns to the calling program.

At statement 10 control transfers to statement 30 if the shape code indicates other than a circular or S-curve segment. For circular and S-curve segments, the reciprocal of the radius of curvature is stored in RIP. The coordinates of the center of the circular portion are stored in XC and YC. The components of the vector from the center of the circle to the initial node are stored in C1 and C2. If the point on the segment is less than or equal to 0.999 of the first break distance, or if the shape code indicates a circular segment, control transfers to statement 20. The statements following this test change parameters to generate coordinates for the second circular portion of an S-curve. The sign of the reciprocal of the radius of curvature is reversed. The cumulative distance, S, is set to the distance from the midpoint of the S-curve. The coordinates of the center of the second circular portion, XC and YC, are computed. Variables C1 and C2 are recomputed for the new center.

At statement 20 the sine and cosine of the angle subtended by the perimeter corresponding to S are computed. The coordinates of the point, XX and YY, are computed, and control returns to the calling program.

At statement 30, control transfers to statement 60 if the shape code indicates that the segment is not a rectangle. Otherwise, variable SGN is set to 1. If the shape code indicates a left rectangle, SGN is reset to -1. If S, the distance along the rectangle, is greater than 1.05 times the first side's

(The reverse of this page is blank.)

length, control transfers to statement 40. If S is greater than 0.95 times the length of the first leg, S is set to the length of the first leg. The x- and y-coordinates of the point on the first leg are computed by linear interpolation, and control returns to the calling program.

At statement 40, S is tested to see whether it falls on the second leg of the rectangle. If S is greater than 1.05 times BR2, the length of the second leg of the rectangle, control transfers to statement 50. If S is greater than 0.95 times BR2, S is set equal to BR2. The x- and y-coordinates of the point on the second leg are computed by linear interpolation and control returns to the calling program.

At statement 50 the x- and y-coordinates of a point on the third [illegible] of the rectangle are computed by linear interpolation. Control returns [illegible]e calling program.

At statement 60 the distance, S, is compared to the length of the first side of an angle segment. If S is greater than this length, control transfers to statement 70. If not, the x- and y-coordinates are computed by interpolation for a point on the first leg. Control returns to the calling program.

At statement 70 the distance along the angle is decreased by the length of the first leg of the angle. The coordinates of the point on the second leg are computed by linear interpolation, and control returns to the calling program.

i. *Subroutine MAPPLT*

Subroutine MAPPLT draws a map of the street segments, one line per segment, with the section number appended to each segment. Up to 10 maps can be drawn.

Subroutine MAPPLT has two arguments. Argument II indicates the sequence number of the map. Argument KF is the number of segments.

The coordinates of the region bounding the map are contained in arrays in COMMON block MPDATA. In this COMMON block, arrays XMIN and XMAX are the minimum and maximum x-coordinates for the map. XLEN is the length, in inches, of the map in the x-direction. YMIN, YMAX, and YLEN are the corresponding arrays in the y-direction. Array YHCUT contains the height, in plotter inches, at which the map must be sliced into strips. Variable AVMD contains the miles per MCU conversion factor for each map.

MAPPLT begins by retrieving or computing the map bounds, the height of a strip of the map (PHGT), the maximum length, the number of map strips (MX), the map scale factors, and the intervals in MCU at which the strips are to be cut. These parameters are printed according to format 90.

The loop through statement 200 will test each segment to see whether it falls within the frame of the map; if it does, the segment will be plotted. Variables NI and NF are set equal to the numbers of the nodes bounding the segment. The midpoint coordinates of the segment are saved in variables XMD and YMD. The lines in the node-number array at which the initial and final nodes occur are saved in variables NS1 and NS2. The initial and final coordinates of each node are retrieved.

Initially the segment is assumed to be entirely within the bounds, and indicators INBI, INBM, and INBF are set to 1. If the coordinates of the initial node lie outside the frame of the map, INBI is set to 0. Similar tests are made on the coordinates of the midpoint of the segment and the coordinates of the final node of the segment. If all three points are outside the frame of the map, control transfers to statement 200 and the segment is not plotted. For segments that are at least partially within the frame of the map, the section number and total length of the segment, in miles, are saved in variables NUMS and TOTLEN. The number of points to be used in plotting one-half of the segment (NPMID) is computed. The number will be restricted to a maximum of 10 points. The total number of points per segment, NPPSEG, is set to twice (NPMID).

Subroutine SHAPCOM is called to set up the parameters needed to generate coordinates of points on the segment. The cumulative length along the segment is initially set to 0. A step size, DS, is computed as the total length divided by the number of points to be plotted on the segment. The coordinates of the initial node are stored in variables XX and YY. The number of the strip of the map into which the node falls is computed. Both a current value of the strip number, NMAP, and a value for the previous point, NMAPO, will be used. The pen position, up or down, is determined by whether the initial point was in bounds. Variable IPEN will be 3 if the point is out of bounds and 2 if the point is in bounds. If the point is out of bounds, control transfers to statement 130. If not, the coordinates of the point are converted to plotter inches and stored in variables XP and YP. If the current node has already been plotted as the last node on the previous segment, control transfers to statement 120. If not, a small square marking its position is appended to the map. At statement 120 the pen is moved to the position of the current point on the segment.

Statement 130 starts a loop through statement 170 that will advance the pen through the remaining points on the segment. The cumulative length is incremented by DS. Subroutine COORD is called to obtain the coordinates of the point in MCU.

At statement 140 the coordinates are converted to plotter inches. The point is assumed to be in bounds, and variable INB is set to 1. If the coordinates of the point are out of bounds, INB is reset to 0. If the pen has been up and the current point is out of bounds, or if the strip number is greater than the number of the final strip, control transfers to statement 60. Otherwise, the pen is moved to the position of the current point. If the pen is up, it is lowered. Variable IPEN is recomputed to reflect whether the point is in bounds.

At statement 150, if the loop index is not equal to the number of the midpoint of the segment, control transfers to statement 160. Otherwise, the section number is appended to the map near the segment midpoint, and the pen is repositioned at the midpoint.

At statement 160 the number of the current strip is computed. If the current strip number is equal to the previous strip number, control transfers to statement 170. If not, the old strip number (NMAPO) is set equal to the current strip number; IPEN is set to 3, indicating that the pen is up; and control transfers to statement 140. In this case, the pen is positioned at the current point on the new strip.

Statement 170 is the end of the loop that causes the segment to be drawn. If the last point drawn is out of bounds, control transfers to statement 200. Otherwise, a small square marking the node's position is appended to the map. The pen is repositioned at the last node. The number of the node is saved in variable LASTNN. Statement 200 is the end of the loop that draws the various segments. At statement 300 the plotter pen is positioned 2 inches beyond the end of the last strip. Control returns to the calling program.

j. Subroutine BUILD

Subroutine BUILD creates a near-neighbor table for the street segments in the map description. This subroutine has 13 arguments. The first, N, is the total number of segments in the map description. The second, KN, is the number of near neighbors that will be found for each segment. KN is set to 60 in the main program. The next two arguments, X and Y, are arrays containing the x- and y-coordinates of the segment midpoints. The fifth argument, MINFR, is an array containing refuse quantity, servicing time, and number of houses for each segment. The sixth argument, TREE, is an array used in the construction of the near-neighbor table. The seventh argument, ISTPR, is an array of segment numbers. The eighth and ninth arguments, NNT and NNTEMP, are temporary storage arrays. The next two arguments, XT and YT, are arrays of the x- and y-coordinates of the segment midpoints. The twelfth argument, KP, is the number of words used to store near-neighbor information for each segment. The last argument, IUNX, gives the number of the unit on which the near-neighbor table will be written. If IUNX is zero, the near-neighbor table will be written on file TAPE7.

Subroutine BUILD begins by setting variable IUNIT equal to UNIT(5). If argument IUNX is positive, IUNIT is reset to IUNX. The loop on statement 5 creates sixty 1-bit masks in array BDATA. The next statement begins a loop through statement 1111 that will examine each segment. The segment number is stored in NNT(1). The loop through statement 1000 transfers the segment numbers to array NNTEMP and the segment midpoint coordinates to arrays XT and YT. The next nine statements interchange the segment stored in the first location of arrays XT, YT, and NNTEMP with the segment stored at location II. Variable MM is set to one less than the number of segments. The loop through statement 20 computes the distances from the segment whose midpoint coordinates are in the first location of arrays XT and YT to each other segment. The distances are stored in array TREE with the low-order 12 bits replaced by the segment number. Subroutine SORTK is called to return the smallest KN distances of the MM distances in array TREE. The loop through statement 30 retrieves the segment numbers from the low-order 12 bits of the KN smallest distances. These segment numbers are stored in array NNT. The loop on statement 94 sets the COMP array equal to 0. This array will be used to store in turn each segment's near-neighbor list. The loop through statement 95 generates the near-neighbor list for the segment currently in location 1 of array NNTEMP. A unique word pointer, IW1, and a unique bit pointer, IP1, are generated from the neighboring segment number in array NNT. The appropriate bit in the appropriate word of array COMP is set to 1 by means of the appropriate mask in array BDATA. The segment number is stored in STRING(1). The load, time, and number of houses on the segment are transferred to array MINFO, which is equivalent to STRING(2). Array STRING is written to file IUNIT. Note that the COMP array is equivalenced to the STRING array starting at STRING(5). When all near-neighbor table information has been written to file IUNIT, the file is rewound. Control returns to the calling program.

k. Subroutine SECTION

Subroutine SECTION assigns each segment in the map description to a section (a section corresponds to a collection trip or vehicle load). Subroutine SECTION has 15 arguments. The first, NN, is the number of segments. The second, K, is the number of near neighbors found for each segment. The next two arguments, MODE and IFLAG, are provided to facilitate modifications that will allow for user-specified base segments. The fifth argument, KCUTOF, gives the minimum number of segments to be considered for inclusion in the same section as a given base segment. The sixth and seventh arguments, X and Y, are the coordinates of the segment midpoints. The eighth, NNTS, is an array of segment numbers. The next four arguments, IST0, IST1, IST2, and IST4, are temporary storage arrays of 30 words each. The thirteenth, KP, is the number of words used for near-neighbor table data for each segment. The fourteenth, KPB, is the number of words in use per segment in array STRING. The fifteenth argument, MA, is the dimension of arrays IST0, IST1, IST2, and IST4.

Subroutine SECTION begins by initializing parameters. The cumulative time and load are set to 0. The maximum number of base segments to be saved, NSTO, is set to 30. A count of the number of passes through near-neighbor histogram generation (LPASS) and a count of the number of completed sections (KPASS) are set to 0. The number of segments is stored in variable NP. Variable SMLD, the smallest cumulative load required to complete the current section, is set to 0. The number of trucks, NTRUCK, is set to 0. The number of nodes is saved in variable N. Symbolic names for disk files are assigned values. Variable OLDUNT is set to 1, IUNIT to 2, IO3 to 3, IO4 to 4, IO5 to 7, and IO10 to 10.

The loop on statement 5 clears array TRUCK, while the loop through statement 6 sets the cumulative time to the unloading time, TDUMP. After statement 6, the pointer to the first segment in section 1 is set to 1. The loop through statement 15 selects vehicles of differing capacities, while the loop thorugh statement 10 generates an entry in the TRUCK array for each vehicle requested. Following statement 15, the total number of vehicles is stored in variable NTO.

Since MODE is set to 0 in the calling program, control continues to the loop through statement 17. (If user-specified base segments were to be added to the program, SECTION would be called with a nonzero MODE.) The loop through statement 17 scans the near-neighbor data on unit 7 (symbolically IO5) searching for the first base segment. When the segment is found, the segment data and the neighbor list for the segment are transferred to array BASE in the loop on statement 172. The refuse quantity, servicing time, and number of houses for the segment are also transferred to variables INF1, INF2, and INF3. Segments other than the base segment are written to file OLDUNT at statement 171. After the loop through statement 17 has been completed, files OLDUNT and IO5 are rewound. Control transfers to statement 1301, bypassing the coding that selects a base segment on the basis of its distance from the previous base segment.

At statement 100 the coordinates of the current base segment are transferred to variables XR and YR. A distance variable, ODIS, is set to 1000000. The loop through statement 1101 scans the segments on file OLDUNT, computing the distance from each segment to the current base segment. As distances shorter than ODIS are encountered, the new distance and segment numbers are saved. The neighbor data are saved in array BASE. When the loop through statement 1101 is complete, the segment closest to the old base segment will be saved as the new base segment. Files OLDUNT and IUNIT are rewound.

The loop through statement 1401 scans file IUNIT for the current base segment. All other segments are rewritten to file OLDUNT. When the loop through statement 1401 is complete, both files are rewound.

Following statement 1301, the smallest cumulative load necessary to complete the section is computed. The loop on statement 90 transfers base segment information to array NNTS. Array BASE contains the segment number, refuse quantity, servicing time, number of houses, and near-neighbor list. This information is written to file IO3. The count of the number of base segments stored in the NNTS array (NSTD) is set to 1. The base segment refuse quantity, servicing time, and number of houses are transferred to the TRUCK array. Variable TRUCK(6, SECTN) is set to 1, indicating that one segment is serviced by the vehicle in this section. A count of segments is also kept in variable PC, which is also set to 1.

At statement 1021 the count of segments left to be assigned (KL) is computed. At statement 440, counters JON and ION are set to 0. The loop through statement 1020 sets array IST2 equal to consecutive integers and clears arrays IST0, IST1, and IST4. The loop on statement 1019 sets to 0 the 60 words in array HISTO.

The segment number of the current base segment is stored in variable L1. The loop through statement 2929 scans all segments on file OLDUNT. The segment and its neighbor data are read, and the segment number is stored in variable L2. Word pointer IW1 and bit pointer IP1, which indicate the position of segment L2 in the near-neighbor data for the base segment, are computed. This bit is examined in the neighbor table of the segment read from OLDUNT; if it is nonzero, control transfers to statement 1022, indicating that the base segment is a near neighbor of the segment from file OLDUNT. Otherwise, control transfers to statement 3030.

At statement 1022, word and bit pointers are computed for the segment read from file OLDUNT. The near-neighbor table for the base segment is examined. If the segment from file OLDUNT is a near neighbor of the base segment, control transfers to statement 1023. Otherwise, control transfers to statement 3030.

At statement 1023, the shared-neighbor count, IC, is set to 0. The loop on statement 1024 counts the number of neighbors shared by the base segment and the segment read from file OLDUNT. The count is used as a subscript on array HISTO, and the appropriate location is incremented by 1. If the count is negative, which should be impossible, or if variable JON, the number of segments sharing neighbors with the base segment, is greater than or equal to 30, control transfers to statement 3030. Otherwise, JON is incremented by 1, and the segment number and shared-neighbor count are saved in arrays IST0, IST1, and IST4. The STRING array for this segment is written to file IO10, and control transfers to the end of the loop.

At statement 3030, the STRING array for segments that are not near neighbors of the base segment is written to file IUNIT. The number of segments on file IUNIT is computed and stored in variable NUT. File IUNIT is rewound.

The loop through statement 4040 forms a running count of the entries in the HISTO array, beginning with the end of the array. When the count passes KCUTOF, control transfers to statement 4450. KCUTOF is set to 5 in the main program. At least five segments sharing the greatest number of near neighbors with the base segment will be examined for inclusion in the current section. Variable KI is set equal to the smallest number of near neighbors that any of these five segments shares with the base segment.

Following statement 4450, file OLDUNT is rewound. If JON, the number of segments sharing neighbors with the base segment, is 0, control transfers to statement 4990. Otherwise, the shared-neighbor counts in array IST1 are sorted into decreasing order; the segment line numbers in IST2 are carried along during the sort. File IO10 is rewound.

A segment counter, variable ION, is set to 1. At statement 1001, if ICODE is equal to 1, indicating that the section is complete, control transfers to statement 700. Otherwise, at statement 101 variable IP is set equal to the line number of the unassigned segment sharing the most neighbors with the base segment. Unit IO10 is rewound. The segment number is stored in variable LNX, and the count of shared neighbors is stored in LNY. If the segment counter (ION) is greater than the total number of segments (JON), control transfers to statement 4991. Otherwise, the loop on statement 102 reads segments from unit IO10 until a segment is found with a segment number equal to LNX and a shared-neighbor count greater than or equal to variable KI. If the segment is found, control transfers to statement 500. If not, following the loop through statement 102, if the current load plus the load from all previous sections exceeds the smallest load necessary to complete the current section, control transfers to statement 600. Otherwise, files IO10 and OLDUNT are rewound.

At statement 4991, counter LPASS is incremented by 1. The loop through statement 3436 reads the segment and neighbor data from file IO10. If segments have not been assigned to the current section, they are written to file OLDUNT. The loop through statement 3437 transfers the remaining segments from file IUNIT to file OLDUNT. Files IUNIT, OLDUNT, and IO10 are rewound.

At statement 4990, if more base segments are needed than have been saved in array NNTS, control transfers to statement 800. Otherwise, the loop on statement 475 transfers segment and neighbor data from array NNTS to array BASE. Variable NEXTN is incremented by 1, and control transfers to statement 1021.

At statement 500 the refuse quantity for the segment is stored in variable CURL. The servicing time on the segment is stored in variable CURT. The segment counter, ION, is incremented by 1. If adding the segment to the current section keeps the section within the vehicle capacity and time limit, control transfers to statement 550. If ION is greater than the number of segments sharing neighbors with the base segment (JON), control transfers to statement 600. Otherwise, file IO10 is rewound. Variable IP is set to the line number of the next segment sharing neighbors with the base segment. The segment number is stored in variable LNX, and the count of shared neighbors is stored in variable LNY.

The loop through statement 510 reads the segment and neighbor data from file IO10 into array STRING. When the segment being sought is found, control transfers to statement 500. If the segment is not located before the loop is completed, control transfers to statement 600.

At statement 550 the count of shared neighbors in array IST1 is made negative, indicating that the segment has been assigned to a section. The next three statements add the refuse quantity, the servicing time, and the number of houses on the segment to the corresponding quantities for the section in progress. If the number of segments saved for use as base segments, NSTD, is greater than or equal to the maximum number allowable, NSTO, control transfers to statement 598. Otherwise, NSTD is incremented by 1.

The loop on statement 599 moves the current segment information in array STRING to the base segment array NSTS. Following statement 598, the STRING array for the current segment is written to file IO3. A segment count, PC, is incremented by 1. The number of segments in the section is incremented by 1. Control transfers to statement 1001.

Statement 600 is reached when a section is full or when no other segments can be added to the section. When the section is complete, variable ICODE is set to 1. File IO10 is rewound.

The loop through statement 698 reads the segment and neighbor data from file IO10 into array STRING. If any segment has not been assigned to a section, IST1 will be positive for that segment and array STRING will be written to file OLDUNT. After the loop has been completed, file IO10 is rewound.

The loop through statement 699 transfers the remaining segments and neighbor lists from file IUNIT to file OLDUNT. Both files are rewound when the loop has been completed.

At statement 700, file I03 is rewound. The count of total unassigned segments, N, is decremented by PC, the count of segments in the section just completed. Variable NP is set to the number of currently unassigned segments. Variable LC is set equal to PC. The loop through statement 705 reads from file I03 the segments assigned to the section just completed and adds the refuse quantity and servicing time to the cumulative totals. The segment numbers are written to file I04.

After the loop through statement 705 has been completed, the refuse quantity, TESTL, and the servicing time, TESTT, for all remaining unassigned segments are computed. The number of the next section, IST, is computed. If IST is greater than the total number of vehicles, NTRUCK, control transfers to statement 801.

At statement 818, if the unassigned refuse quantity exceeds the capacity of the next vehicle, control transfers to statement 710. If not, and if the servicing time for all remaining unassigned segments exceeds the time limit for the next vehicle, control transfers to statement 710. Otherwise, the count of total segments assigned, PK, is incremented by PC. The sequence number of the next segment to be assigned is stored in the TRUCK array. File IUNIT is rewound. Variable LN is set to the number of segments yet to be assigned. The loop through statement 706 reads the remaining unassigned segments from file OLDUNT and writes the segment numbers on file I04. When the loop is complete, the section count, SECTN, is incremented. Pointers to the first and last segment on file I04 in the current section are stored in array TRUCK. File I04 is rewound, and control returns to the calling program.

Statement 801 is reached when additional trucks must be defined to complete the sectioning. At statement 801 the number of vehicles, NTRUCK, is incremented. A message is printed indicating that the vehicle configuration has been extended. Cyclical counter TPR is reset to 1 if it exceeds the original number of vehicles. A new vehicle is defined in the TRUCK array according to the current value of TPR. The loop on statement 811 clears all except capacity and time-limit items in the TRUCK array on the line for the new vehicle. TPR is incremented by 1. Variable IST is set equal to the number of the new vehicle. Control transfers to statement 818.

Statement 710 is reached when a section is completed. At statement 710, files OLDUNT, IUNIT, and I03 are rewound. The count of assigned segments, PK, is incremented by PC. The section number is incremented by 1. A pointer to the next segment to be written to file I04 is computed and stored in the TRUCK array. Various parameters are reset, and control transfers to statement 100.

Statement 800 is reached when base segments can no longer be obtained from the NNTS array. The coordinates of the last base segment midpoint are stored in variables XR and YR. A distance variable, ODIS, is initially set to 1000000. The number of remaining unassigned segments is computed and stored in variable NLK. The count of base segments is reset to 0.

The loop through statement 2101 searches for a new base segment. Segments are read from file OLDUNT and are written to file IUNIT. The segment number is saved in variable NS. The segment midpoint coordinates are stored in variables XS and YS. The distance between the current segment and the previous base segment is computed and stored in variable DIS. If DIS is greater than or equal to ODIS, control transfers to the end of the loop. If the refuse quantity for the current segment causes the vehicle capacity to be exceeded, control transfers to statement 2101. If not, and if the servicing time for the current segment causes the vehicle capacity to be exceeded, control transfers to statement 2101. Otherwise, the current distance is saved in ODIS, and the current segment number is saved in variable NBASE.

The loop on statement 2101 transfers the STRING array for the segment to the BASE array. Statement 2101 ends the loop that seeks a new base segment. Files OLD and IUNIT are rewound. If no base segment was found, NBASE will be equal to 0, and control will transfer to statement 600.

The loop through statement 2401 reads segments and their neighbor lists from file IUNIT into array STRING. If the segment number is not equal to the base segment number, control transfers to statement 2400. Otherwise, the segment refuse quantity, servicing time, and number of houses are transferred to variables INF1, INF2, and INF3. Control transfers to statement 2401.

At statement 2400 the STRING array is written to file OLDUNT. Following the loop, files OLDUNT and IUNIT are rewound. The base segment and its neighbor data are written to file IO3. The count of segments assigned to the current section, PC, is incremented by 1. Section data in array TRUCK are updated for the refuse quantity, servicing time, number of segments, and number of houses. The number of base segments, NSTD, is reset to 1 and variable NEXTN is set to 2. If the total amount of refuse assigned so far is less than the smallest load required to complete the current section, control transfers to statement 1021. Otherwise, control transfers to statement 600.

1. Program PHASE2

Main program PHASE2 drives the sectioning and the plotting of section assignment maps. It uses 11 files: INPUT, OUTPUT, TAPE1, TAPE2, TAPE3, TAPE4, TAPE7, TAPE8, TAPE9, TAPE10, and TAPE11. File TAPE5 is equivalenced to INPUT in order to test for end-of-record cards in the card input data. File TAPE1 is used for section-description information. Files TAPE2 and TAPE3 are used for temporary storage of segment and near-neighbor information. File TAPE4 is used for a list of segment numbers in the order they are assigned to sections. File TAPE7 is used for temporary storage of segment and neighbor data. File TAPE8 is the Calcomp plot tape. File TAPE9 holds input segment data from program RCINPT. File TAPE10 is used for temporary storage of segment and neighbor data. File TAPE11 holds node data from program RCINPT.

Blank COMMON holds arrays for the problem title and the segment data. Array ISEG contains the section numbers assigned to segments by subroutine SECTION. Arrays NN1 and NN2 are the starting and ending node numbers for the segments. Array FLEN holds the lengths of the segments. Array NH is the number of houses on the segments. Array FMPH is the speed limits on the segments. Array RQF gives the refuse quantity adjustment factor for the segments. Arrays

X and Y are the midpoint coordinates of the segments. Array SF holds the shape codes for the segments.

COMMON block MPDATA holds arrays describing the map bounds and sizes for up to ten maps. Array YHCUT contains the height of a strip of the map, and variable AVMD is the number of miles per MCU.

COMMON block NODATA holds the node data. KNODES is the number of nodes. Array NBS holds the segment numbers of up to six segments bounding each node. Array NODNUM contains the node numbers. Arrays XNOD and YNOD are the coordinates of the nodes.

COMMON block DISKIO holds array UNOT, which is used for symbolic references to disk and tape file numbers.

COMMON block ROUTE holds vehicle and section-description data. Array TRUCK has seven words of data for each of up to 50 vehicles. Array TRUCKS holds descriptive information for the vehicle fleet available in the program. Variable NTRUCK is the total number of vehicles or sections. Variable NBASE is the segment number for the segment to be used as the first base segment.

COMMON block STATS holds additional information about the sections. Variable FRACT is the ratio of total refuse to total vehicle capacity. Variables TOTL and TOTT are the total refuse quantity and total servicing time for all segments in the map description. Variables CUML and CUMT are the cumulative load and servicing time. Variable ISECTN is set to the total number of sections by subroutine SECTION. Variable TDUMP is the unloading time at the landfill.

Program PHASE2 begins execution by reading three data cards: the title card, the vehicle-description card, and the time-limits card. Segment data are read from file TAPE9. Node data are read from file TAPE11.

The loop on statement 15 retrieves the absolute value of the number of houses on each segment. A negative number for variable NH indicates that collection is from the right side of the street only. The number of segments is stored in variable NA. If the number of the first base segemnt, NBASE, was left blank on the third data card or was improperly specified, it is set to 1. Variable KN, the number of near neighbors to be found for each segment, is set to 60. Variables MODE, OPTION, and NSEED, which are provided to facilitate implementation of user-specified base segments, are set to 0. The total time and load are set to 0.

The loop through statement 20 calculates and saves the total refuse quantity, servicing time, and number of houses for each segment. On each segment with houses, the servicing time is recomputed to include travel at 5 miles per hour and stopping time at the houses. The total time and load in the network is accumulated in variables TOTT and TOTL. Following statement 20, if the maximum trip time is unspecified or negative, it is reset to 24 hours. The following statement converts the maximum trip time to minutes. For a vehicle with a nonzero capacity, the loop on statement 25 sets the number of vehicles in the NT array to 1 if no vehicles were specified. Variable REF is initially set to 0. It will be used to accumulate the total capacity available.

The loop through statement 50 moves to the TRUCKS array the number of vehicles, the capacity, and the maximum trip time for each of the four types of vehicles allowed. The total vehicle capacity available, REF, is also accumulated in this loop. The integer part of the ratio of total refuse to total vehicle capacity is computed and stored in variable NTEA. If NTEA is 0, which indicates that enough vehicles were specified in the card input to completely service the region, control transfers to statement 80. Otherwise, REF is reset to 0.

The loop through statement 60 multiplies the number of vehicles specified on input cards by NTEA and stores the result in the TRUCKS array. The vehicle capacity now available is accumulated at statement 60. The loop through statement 70 adds additional vehicles where necessary to bring the total fleet capacity above the total amount of refuse to be collected. An increment, NINC, is preset to 0. For each vehicle capacity specified, NINC is set to the smaller of the number of vehicles specified on the data card or the number of vehicles required to provide enough capacity to completely service the region. The vehicle capacity available, REF, is increased by NINC times the vehicle capacity. The number of vehicles is increased by NINC in the TRUCKS array. If enough capacity is available to service the region, control transfers to statement 80.

At statement 80 the ratio of total refuse to total vehicle capacity is computed and stored in variable FRACT. The input card data are printed, along with FRACT, the minimum fraction that each vehicle must be filled. The starting base segment, NBASE, is also printed. Variable IUNX is set to 0.

The loop on statement 96 generates an array of segment numbers (ISTPR). The segments are numbered sequentially starting at 1. Variable MA, the maximum number of base segments to be saved while building a section, is set to 30. Variable KP, the number of words needed to store the near-neighbor data, is computed. Variable KPB, the number of words used in array STRING, is set to KP plus 4. Subroutine BUILD is called to generate the near-neighbor table. Variable IFLAG is set to 0. Variable KCUTOF is set to 5, causing at least five segments sharing the most neighbors with a base segment to be examined for addition to a section.

Subroutine SECTION is called to assign the segments to sections. If IFLAG is set to 1, control transfers to statement 333. Otherwise, a summary is printed for the various trips. The summary includes trip number, vehicle capacity, vehicle time limit, vehicle load, servicing time for the section, number of segments in the section, and number of houses in the section. The loop through statement 40 prints the data in this tabulation. If OPTION is equal to 1, the number of segments is incremented by the number of user-specified base segments. (Note that user-specified base segments are not implemented at this time.) Variable IUNIT is set to UNOT(4), which has value 4. Variable J is set to 1.

The loop through statement 33 reads the segments from file IUNIT and groups and prints them according to their section. When the loop index is not equal to the sequence number of a segment starting a section, control transfers to statement 88. Otherwise, a heading for the section is printed. The section number J is incremented by 1. A carriage control, variable CC, is set to a blank. The number of segments on the current line of printed output, NN, is set to 0. At statement 88, NN is incremented by 1. The next statement prints the segment number preceded by the number of blanks required to put the segment number in its appropriate position on the line. The carriage control is set to a plus sign. If fewer than 30 segment numbers have been written, control transfers to statement 33. Otherwise, the carriage control is reset to a blank, and the count of segment numbers on the line is reset to 0. Statement 33 ends the loop that prints the segments in each section. File IUNIT is rewound.

The plot package is initialized by a call to PLOTS. Subroutine PLOT is called twice, moving the pen down and then up 3 inches in order to create a 3-inch border on the plot.

The loop through statement 1000 controls the reading of segments by section. Variable IL is set to the sequence number of the last segment in the Ith section. The loop through statement 2000 reads each segment from file IUNIT and assigns its section number to the appropriate location in array ISEG. The number of maps, MAPS, is initially set to 0. The loop through statement 1030 controls the reading of up to 10 output-map-bounds cards. The minimum and maximum coordinates and lengths in the x and y direction are read from a card. Also read is the height at which the map should be cut into strips. If an end-of-file card is encountered during the read, control transfers to statement 1040. Otherwise, execution continues at statement 1020. At statement 1020, if the map-strip height is less than or equal to zero, or is unspecified on the card, the strip height will be set to 30 inches. At statement 1030 the number of maps is set equal to the loop index, I. At statement 1040, if the number of maps is greater than zero--that is, if any map-bounds cards have been found--control transfers to 1070. Otherwise, default values are set.

The default values are set in the following manner. MAPS is set to 1, and the lengths in the x and y directions are set to 30 inches. The height of a map strip is set to 30 inches. Bounds on the coordinates are set initially so that the minimums are large positive numbers and the maximums are large negative numbers. The loop through statement 1050 scans the midpoint coordinates of the segments and resets the minimum and maximum coordinates appropriately. The loop through statement 1060 scans the node coordinates and resets the minimum and maximum coordinates appropriately.

The loop on statement 1080 calls subroutine MAPPLT to plot each output map. Subroutine PLOT is called to terminate the plot file. Following statement 333, file TAPE1 is rewound. The number of segments and the number of sections are written to TAPE1, followed by the sequence numbers of the first and last segments in each section and the vehicle capacity. Files TAPE1 and TAPE4 are end-filed, and program execution stops.

(The reverse of this page is blank.)

SECTION IV
INPUT AND OUTPUT

1. INPUT

Input to program PHASE2 consists of card input, segment data from file TAPE9, and node data from file TAPE11.

a. Card Input

Table 1 presents the form and contents of the four types of data cards. The first card contains a title, which is printed on the first line of the output. The second card contains the number and capacity of up to four kinds of vehicles. The third card contains time restrictions and the number of the segment to be used as a base segment for the first section. The fourth card specifies coordinate bounds and sizes for the output map. If the output map is to be plotted in strips less than 30 inches high, the height of the strip must be indicated on the fourth card. The fourth card may be omitted, or it may be repeated up to 10 times. If it is omitted, one 30- by 30-inch map of the entire collection region will be plotted. If it is repeated, any map-bounds cards after the tenth card will be ignored.

b. Segment Data

Disk file TAPE9 contains segment data and the map distance conversion factor (miles per MCU) for the overall map. All of the data are read by one binary READ statement. The first word is the count of the segments. The segment data follow, 11 words per segment for each segment. After the segment data comes the overall distance conversion factor.

The list used in the READ statement is NSEG,(DUMMY,NN1(I),NN2(I), FLEN(I),NH(I),FMPH(I),DUMMY,RQF(I),X(I),Y(I),SF(I),I=1,NSEG),AVMD. In the list, variable NSEG is the count of segments. Since the street numbers of the segments are not needed, they are read into variable DUMMY. Arrays NN1 and NN2 hold the starting and ending node numbers for the segments. Array FLEN holds

TABLE 1. PHASE2 DATA CARDS

Card	Columns	Format	Contents
1	1-80	8A10	Title
2	1-10	I10	Number of vehicles of the first kind
	11-20	F10.0	Capacity of vehicles of the first kind
	21-30	I10	Number of vehicles of the second kind
	31-40	F10.0	Capacity of vehicles of the second kind
	41-50	I10	Number of vehicles of the third kind
	51-60	F10.0	Capacity of vehicles of the third kind
	61-70	I10	Number of vehicles of the fourth kind
	71-80	F10.0	Capacity of vehicles of the fourth kind
3	1-10	F10.0	Stop time per household, in minutes
	11-20	F10.0	Stop time per unit refuse, in minutes
	21-30	F10.0	Unloading time, in minutes
	31-40	F10.0	Maximum trip time, in hours
	41-50	I10	First section starting segment number
	The following card is optional and may be repeated up to 10 times.		
4	1-10	F10.0	Minimum x-coordinate of map
	11-20	F10.0	Maximum x-coordinate of map
	21-30	F10.0	Length of map in x (horizontal) direction, in inches
	31-40	F10.0	Minimum y-coordinate of map
	41-50	F10.0	Maximum y-coordinate of map
	51-60	F10.0	Length of map in y (vertical) direction, in inches
	61-70	F10.0	Height of map strips, in inches

the length of the segment, in miles. Array NH holds the number of houses. Array FMPH holds the speed limits for the segments. The one-way-street indicators are not needed and are read into variable DUMMY. The refuse quantity adjustment factors are read into array RQF. The midpoint coordinates of the segments are read into arrays X and Y. The shape codes for the segments are read into array SF. Variable AVMD is the map-distance conversion factor.

c. Node data

Disk file TAPE11 contains refuse-quantity information and node data. All of the data are read by one binary READ statement. The list used in the READ statement is NHTOT,TOTREF,KNODES,(NODNUM(I),NBS(I),XNOD(I),YNOD(I),I=1, KNODES). The first three words, NHTOT, TOTREF, and KNODES, are the total number of houses, the total refuse quantity, and a count of the nodes. Array NODNUM holds the node numbers. Array NBS holds up to six segment numbers of segments bounding the node. Arrays XNOD and YNOD are the coordinates of the nodes.

2. SCRATCH FILES

Disk files TAPE1, TAPE2, TAPE3, TAPE7, and TAPE10 are used by subroutine SECTION as scratch files, and all contain the same type of data. The data consist of 40 words of segment and near-neighbor information for each segment. The first word is the segment number. The second word is the refuse quantity. The third word is the servicing time for the segment. The fourth word is the number of houses on the segment. The next 25 words are the near-neighbor list for the segment. The last 11 words are not used at present. The first word of the near-neighbor list indicates which of the first 60 segments are near neighbors to the segment. The low-order bit corresponds to segment number 1. The bits are set to 1 if the corresponding segment is one of the 60 nearest segments; otherwise, they remain 0. The second word of the neighbor list contains the bits corresponding to segments 61 through 120, and so forth.

The segment data and the neighbor list are written initially to TAPE7 by subroutine BUILD. TAPE7 is read in subroutine SECTION, and all segment data except those for the first base segment are rewritten to file TAPE1, symbolically referred to as OLDUNT. The first base segment is written to file TAPE3. TAPE7 is not used again. The data on OLDUNT are read, and segments sharing near neighbors with the base segment are written to file TAPE10, symbolically IO10. Those segments not sharing neighbors with the base segment are written to file TAPE2, symbolically IUNIT. The data on TAPE10 are examined, and those which can be added to the current section are written on TAPE3. Those which cannot be added are written to OLDUNT. The segments on IUNIT are copied to OLDUNT, and the process repeats until a section is completed. When a section is completed, the segment numbers are transferred from TAPE3 to TAPE4. When only one section remains to be completed, all segment numbers from OLDUNT are copied directly to TAPE4.

In summary, TAPE1 holds the segment and neighbor data for segments not yet assigned to a section. TAPE2 holds segment and neighbor data for all segments not sharing enough neighbors with the base segment. TAPE3 holds segment and neighbor data for segments added to the current section. TAPE7 holds segment and neighbor data for all of the segments. TAPE10 holds segment and neighbor data for those segments sharing enough neighbors with the base segment to be considered for addition to the section.

3. OUTPUT

a. Disk and Plot Files

File TAPE1 is reused for output in main program PHASE2 after the map plotting is completed. The list used in the binary WRITE statement is NA, ISECTN,(TRUCK(5,I),TRUCK(6,I)+TRUCK(5,I)-1., TRUCK(1,I),I=1,ISECTN). Variable NA is the number of segments; ISECTN is the number of sections. The next three items written for each section are the sequence number on TAPE4 of the first segment in the section, the sequence number of the last segment in the section, and the vehicle capacity. The file should be cataloged for later use by program PHASE3.

Disk file TAPE4 is used to hold the segment numbers in the order in which they are assigned to sections. The segment numbers are written with a formatted WRITE statement, one segment number at a time. The format used is 1X,I5. The file should be cataloged for later use by program PHASE3.

File TAPE8, the plot file, will be disk or tape depending on the procedure used by the local installation to produce plots. Each output map requested occupies one file. PHASE2 generates an empty file before terminating TAPE8. The structure of the file depends on the local installation. Figure 2 shows a section map for Kirtland Air Force Base.

b. Printed Output

There are four parts to the printed output: a list of the input card data, a table of trip information, a list of segments in each section, and a list of the parameters used in each output map.

Appendix D contains sample printed output. The first page summarizes the vehicle input data. The title is printed following the heading INPUT VEHICLE DATA. The capacity and number of vehicles available for four kinds of vehicles are listed. Zeros are printed where no vehicle was specified. If the number of vehicles on the input data was inadequate to service the region, the number of trips required will be listed in the column headed TRIPS. The ratio of total refuse to total vehicle capacity is printed as MINIMUM FILL FRACTION. The sectioning algorithm endeavors to fill each vehicle to the indicated fraction before completing a section. The next four lines give the times specified on the third data card. The segment to be used as the first base segment is also printed. If this item has been left blank on the third data card, the printed message indicates that sectioning starts with segment 1.

The second page of printed output starts with a table of information about each section. The capacity, time limit, and load for the vehicle is given, followed by the collection-time estimate, the number of segments, and the number of houses in the section. At this stage, the collection time is approximate because some streets not requiring collection may be dropped from

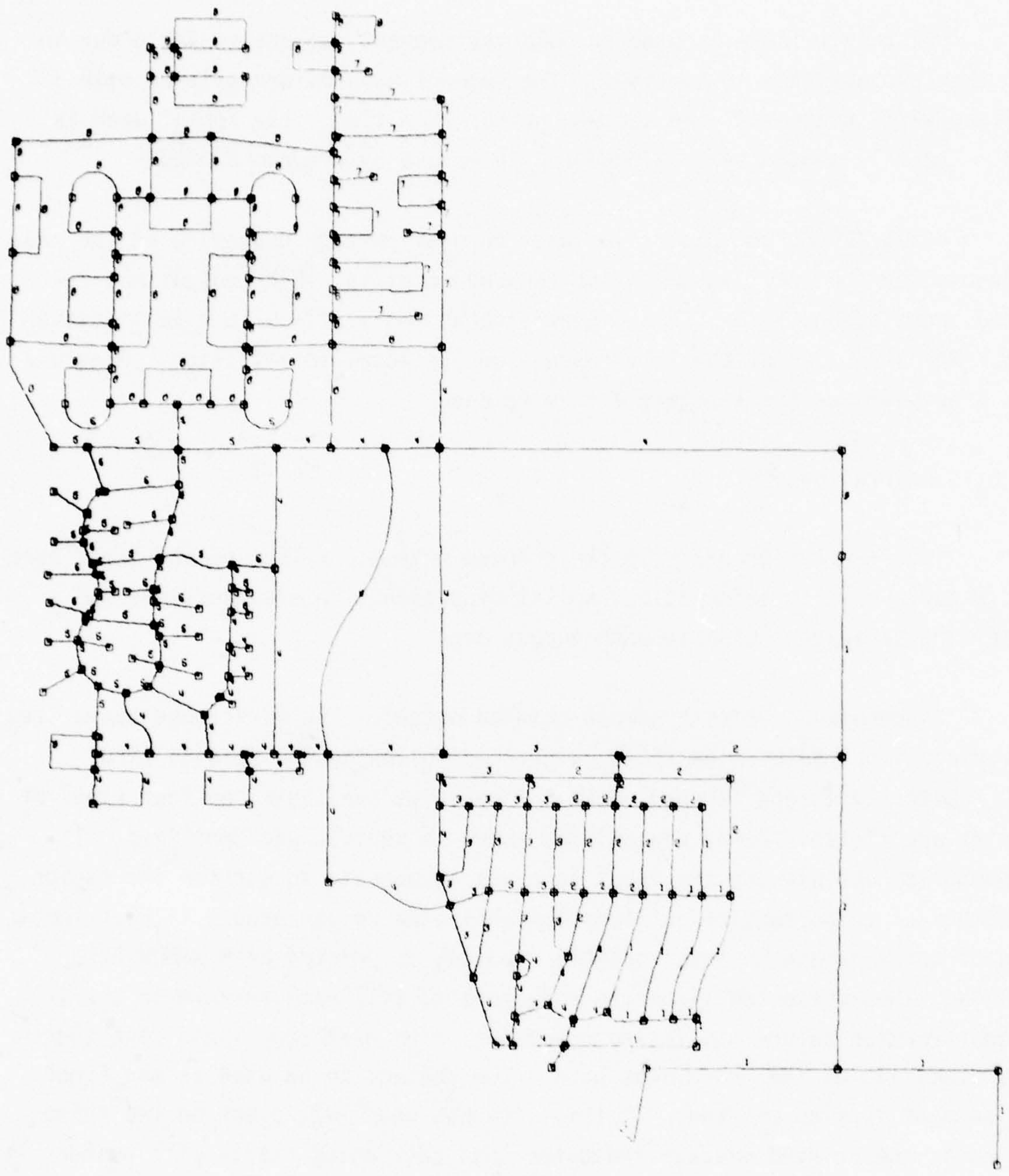


Figure 2. Section Assignment Map for Kirtland Air Force Base

the section and others may be added. Break times and lunch time are not included in the collection-time estimate.

The printed output continues with a list of the segments in each section. The segment numbers are listed in the order in which they were selected for inclusion in the section and will not usually be in numerical order. UP to 30 segment numbers are printed on each line.

Fourteen parameters are printed for each output map, following the heading "MAPPLT PARAMETERS FOR MAP n," where n is the sequence number of the map. AVMD is the map distance conversion factor in miles per MCU. Each of the next two lines gives the minimum and maximum x- and y-coordinates in MCU. On the final line, XSC and YSC identify the x and y map scales in inches per MCU. The plot strip height, PHGT, and the overall plot length, PLEN, are printed in inches. YCUT is the height of a map strip in MCU.

(The reverse of this page is blank.)

SECTION V
PROGRAM REQUIREMENTS

1. SYSTEM

Except for function KOUNT, program PHASE2 is written entirely in FORTRAN IV. Function KOUNT is written in the COMPASS assembler language. The program runs on a CDC 6600 using a SCOPE 3.4.4 operating system.

Eleven obvious types of computer-dependent coding are used in program PHASE2 and its subroutines. A 60-bit word is assumed in the main program and in subroutines BUILD, SECTION, and NUMBER. System function SHIFT is used in subroutine BUILD. An R format is used in subroutines SHAPCOM and COORD. Six-bit characters are assumed in subroutine NUMBER. An ENCODE statement is used in subroutine NUMBER. In-line function MINO is used in PHASE2. Asterisk-bounded character strings are used in formats in PHASE2 and in subroutine SECTION. A computation is used as an item in an output list in PHASE2. The AND operation is used for masking in subroutine SECTION. Function KOUNT is written in COMPASS. A dollar sign is used between FORTRAN statements in program PHASE2 and subroutines SORTK, SECTION, MAPPLT, SHAPCOM, and COORD. More subtle types of machine dependencies may exist, according to the machine used.

2. STORAGE

The core requirement is slightly less than $114,000_8$ words, but may vary slightly depending on the plotting system used by the local installation. The peripheral storage for output disk files should not exceed 28,000 words for TAPE1, TAPE2, TAPE3, TAPE7, and TAPE10. When file TAPE1 is cataloged, it should not contain more than 152 words. Disk file TAPE4 should not exceed 700 words. The plot file, TAPE8, should not exceed 1.5 million words, although more typically the file will contain roughly 20,000 words per output map.

3. TIME

The execution CP time for program PHASE2 is approximately 0.005 S_{TOT}^2 + 0.004 $\sum_{maps} S_i$ seconds, where S_{TOT} is the total number of segments and S_i is the number of segments on the ith output map. The maximum possible CP time using 10 full maps of a 700-segment network would be 273 seconds. Rough estimates of the IO and PP times are 0.2 S_{TOT} seconds for IO time and 0.5 S_{TOT} seconds for PP time.

SECTION VI
PROGRAM LIMITATIONS

1. PLOTTER

The plotter used by PHASE2 can be a drum plotter with at least a 10-inch drum or a flatbed plotter with a 30-inch-square or larger bed. If the user does not indicate a plot-strip height on the fourth data card, the program assumes that a 30-inch height is available. Maps may not be drawn in strips on a flatbed plotter; therefore, the height and width of the output map must not be specified larger than the plot-strip height. Some systems limit plot lengths; for example, Calcomp plots may not exceed 120 inches in length at the Air Force Weapons Laboratory at Kirtland Air Force Base. Each strip of an output map has a 1-inch space after it, with an additional inch after the last strip. Thus a 30- by 30-inch map plotted as three 30- by 10-inch strips generates a plot 94 inches long. Any limitations on plot lengths imposed by the local system must be considered when the output-map description cards are punched.

2. NODE AND SEGMENT

Array dimensions in program PHASE2 limit the number of nodes to 500 and the number of segments to 700. Since these data are passed to PHASE2 by program RCINPT as disk files TAPE9 and TAPE11, the limits should not be exceeded.

3. VEHICLE FLEET

The vehicle fleet is limited to vehicles of at most four different capacities. The total number of trips is limited to 50. If more than 50 trips are required, data will be overwritten and improper and unpredictable functioning will result, but no error message is given.

4. OUTPUT MAP

From 0 through 10 output maps may be specified on map-bounds cards. If no map-bounds cards are present, one 30- by 30-inch map of the entire region is plotted. If more than 10 map-bounds cards are included in the data deck, the cards after the tenth card will be ignored. No message is printed, and the program functioning is not otherwise affected.

SECTION VII
WARNING MESSAGE AND CORRECTIVE ACTION

One warning message is printed by subroutine SECTION. The message TRUCK CONFIGURATION HAS BEEN EXTENDED is printed as the last line of the first page of printed output when more trips are needed than either the minimum number computed by PHASE2 or the number specified by the user. The problem has two possible causes. First, the time limit on a trip may have caused a section to be completed before the vehicle was filled. In this case, the user must decide whether the time limit is important enough to necessitate an extension of the number of trips. Second, the number of vehicles needed may have been extended because all segments considered for addition to a section contained enough refuse to cause the vehicle capacity to be exceeded.

The problem can be solved in one of two ways. The simplest requires re-running PHASE2 with a different choice of starting base segment. It may be necessary to do this several times before the program runs without extending the number of trips. An alternative approach is to cause the segments that could not be added to the section to be collected one side at a time. The section involved can be found by looking at the summary of vehicle loads; the section (other than the last) which has a vehicle load considerably below the vehicle capacity will be the section in which the problem has occurred. Segments near the last segment added to this section are those which should be modified. The modification will cause one or more of the segments to be serviced by two collection vehicles, one on each side of the street.

Note: program malfunction was observed on one run. Three segments were assigned twice, and three segments were not assigned to any section. The cause of this problem is still unknown, but the problem can be bypassed by choosing a different starting base segment.

(The reverse of this page is blank.)

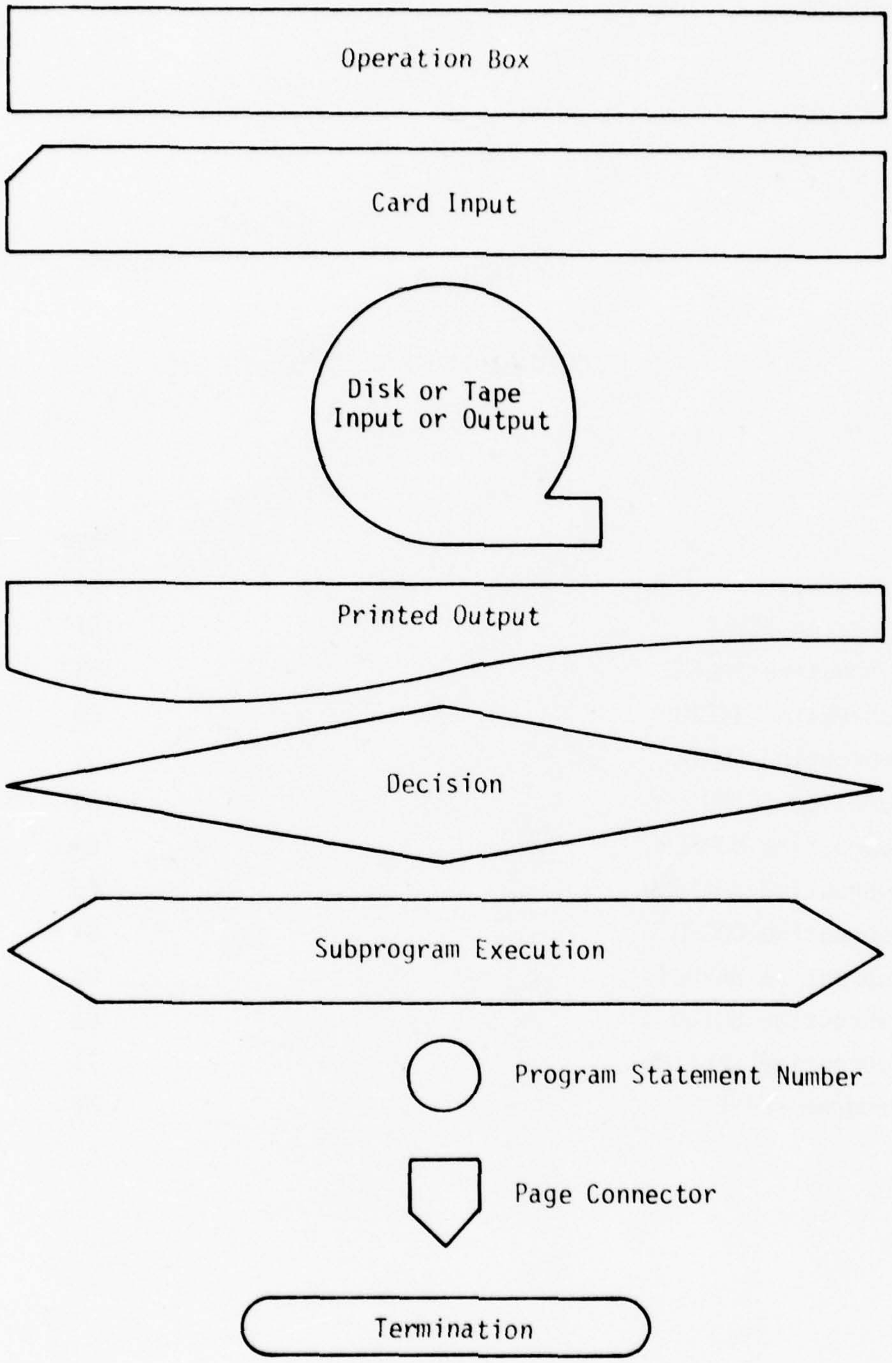
SECTION VIII
RECOMMENDED CHANGES

Three changes in program PHASE2 are recommended. The problem title could be appended to the section maps. If the house count in array NH is made floating-point instead of integer in program RCINPT, the array should be declared type REAL in the main program. It will be necessary to change the absolute value function name from IABS to ABS in statement 15 of the main program. An error message should be added to warn the user if more than 50 trips will be required.

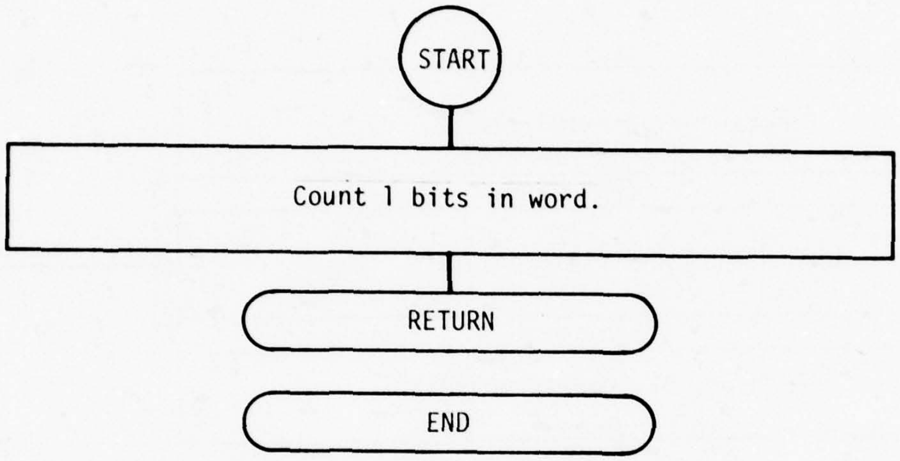
(The reverse of this page is blank.)

APPENDIX A

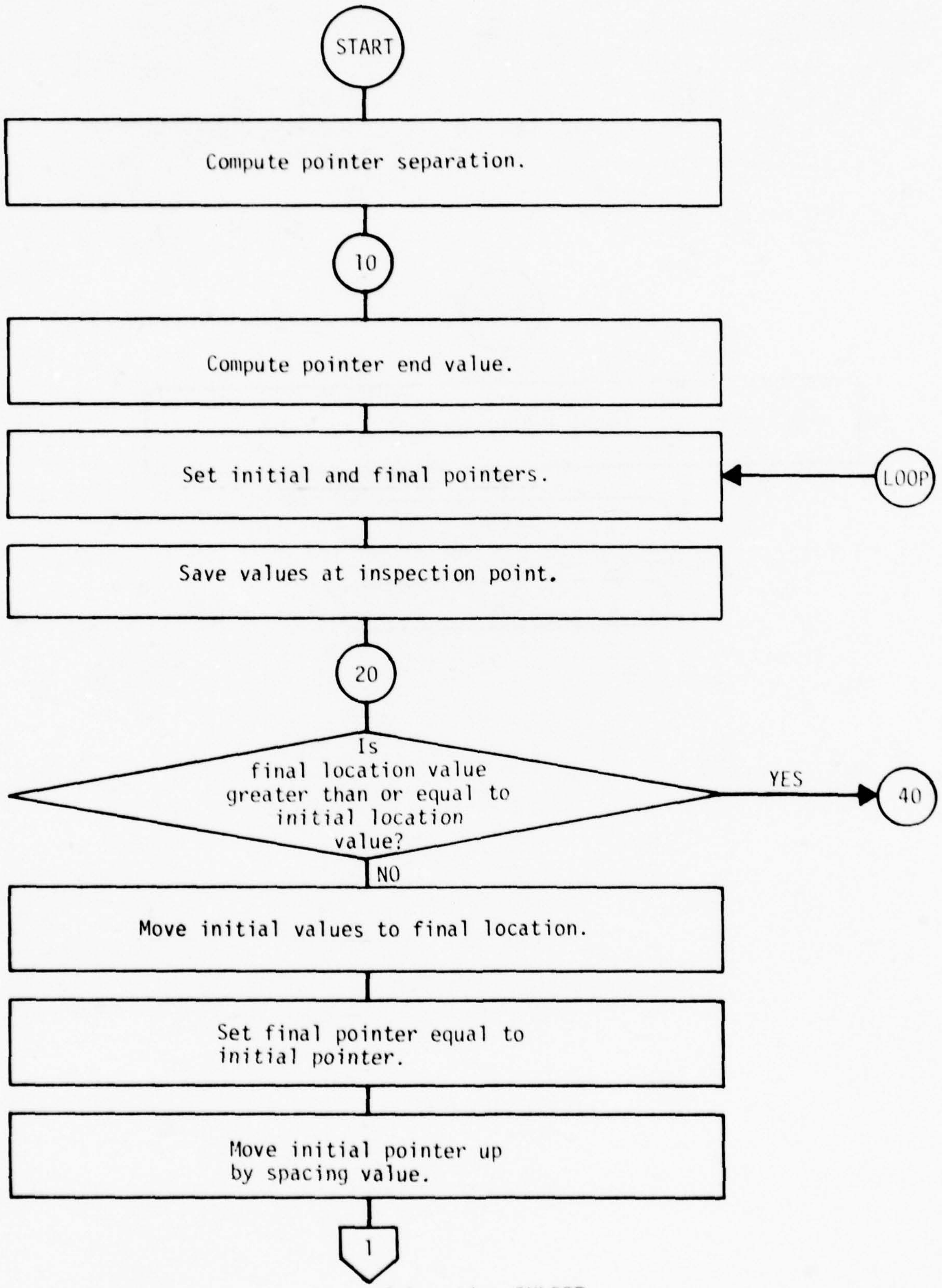
LOGIC FLOWCHARTS

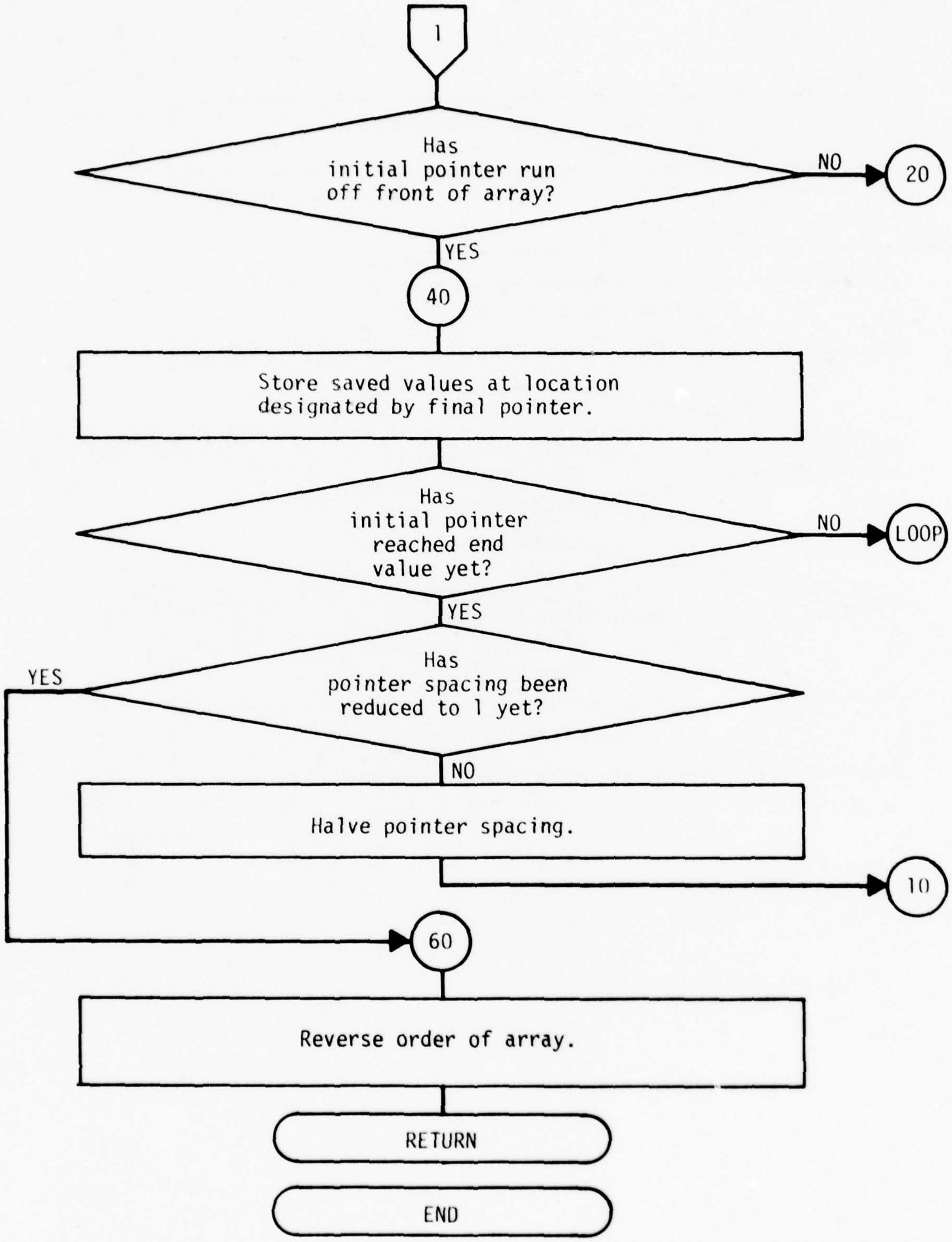


FLOWCHART SYMBOLS

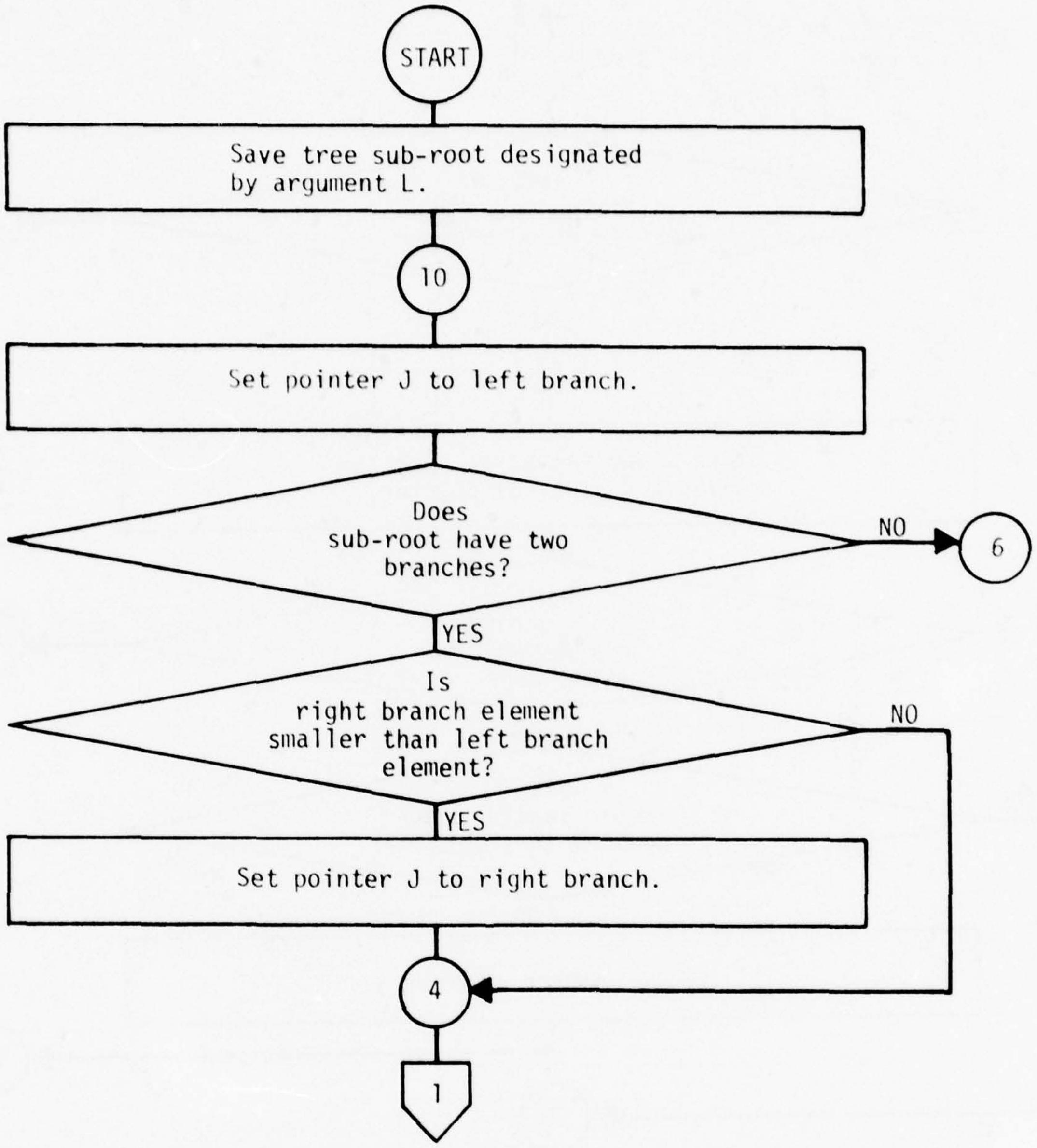


Function KOUNT

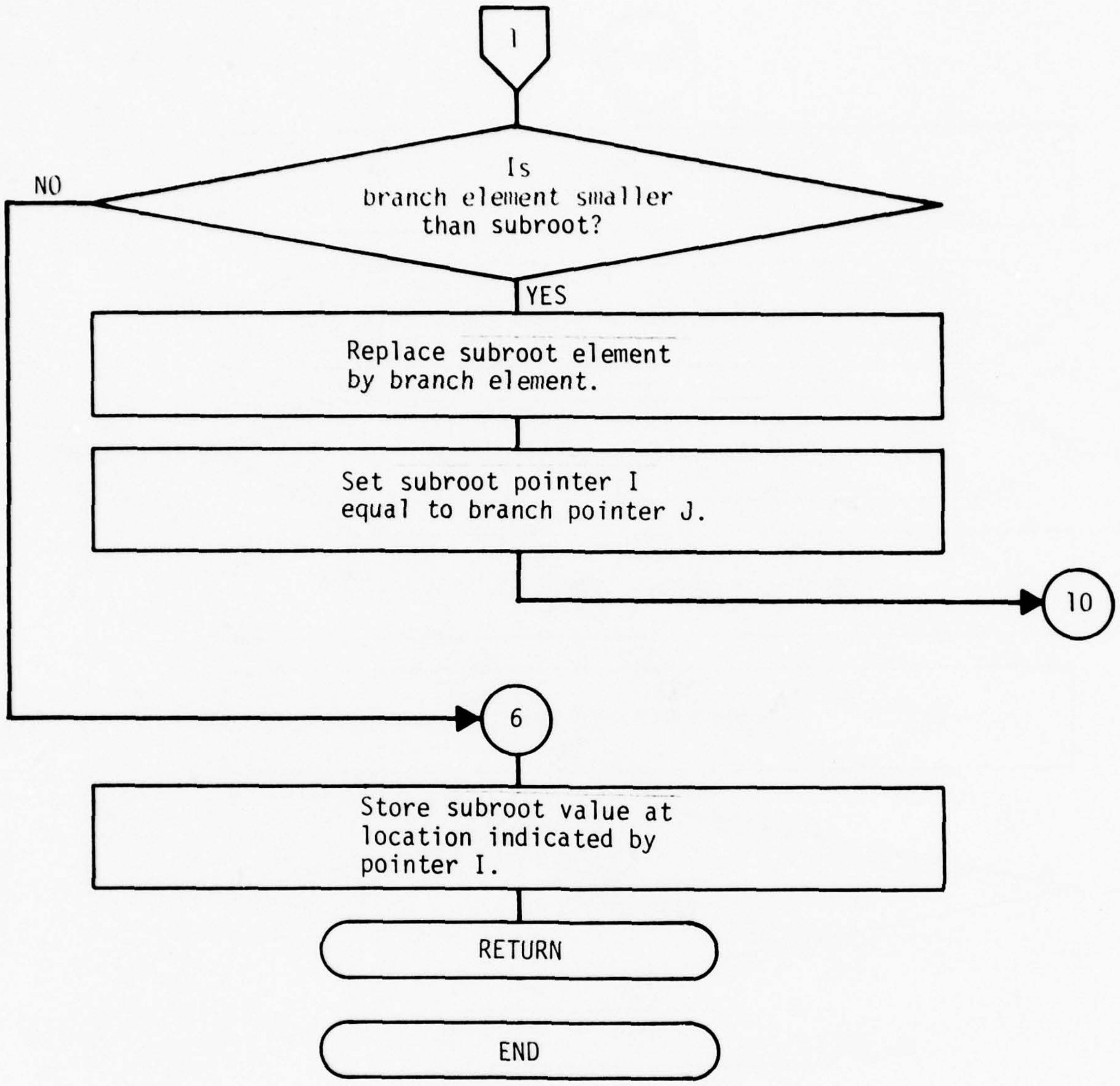

START
Compute pointer separation.
10
Compute pointer end value.
Set initial and final pointers.
LOOP
Save values at inspection point.
20
Is final location value greater than or equal to initial location value?
YES
40
NO
Move initial values to final location.
Set final pointer equal to initial pointer.
Move initial pointer up by spacing value.
1
Subroutine SHLSRT


1
Has initial pointer run off front of array?
NO
20
YES
40
Store saved values at location designated by final pointer.
Has initial pointer reached end value yet?
NO
LOOP
YES
Has pointer spacing been reduced to 1 yet?
YES
NO
Halve pointer spacing.
10
60
Reverse order of array.
RETURN
END

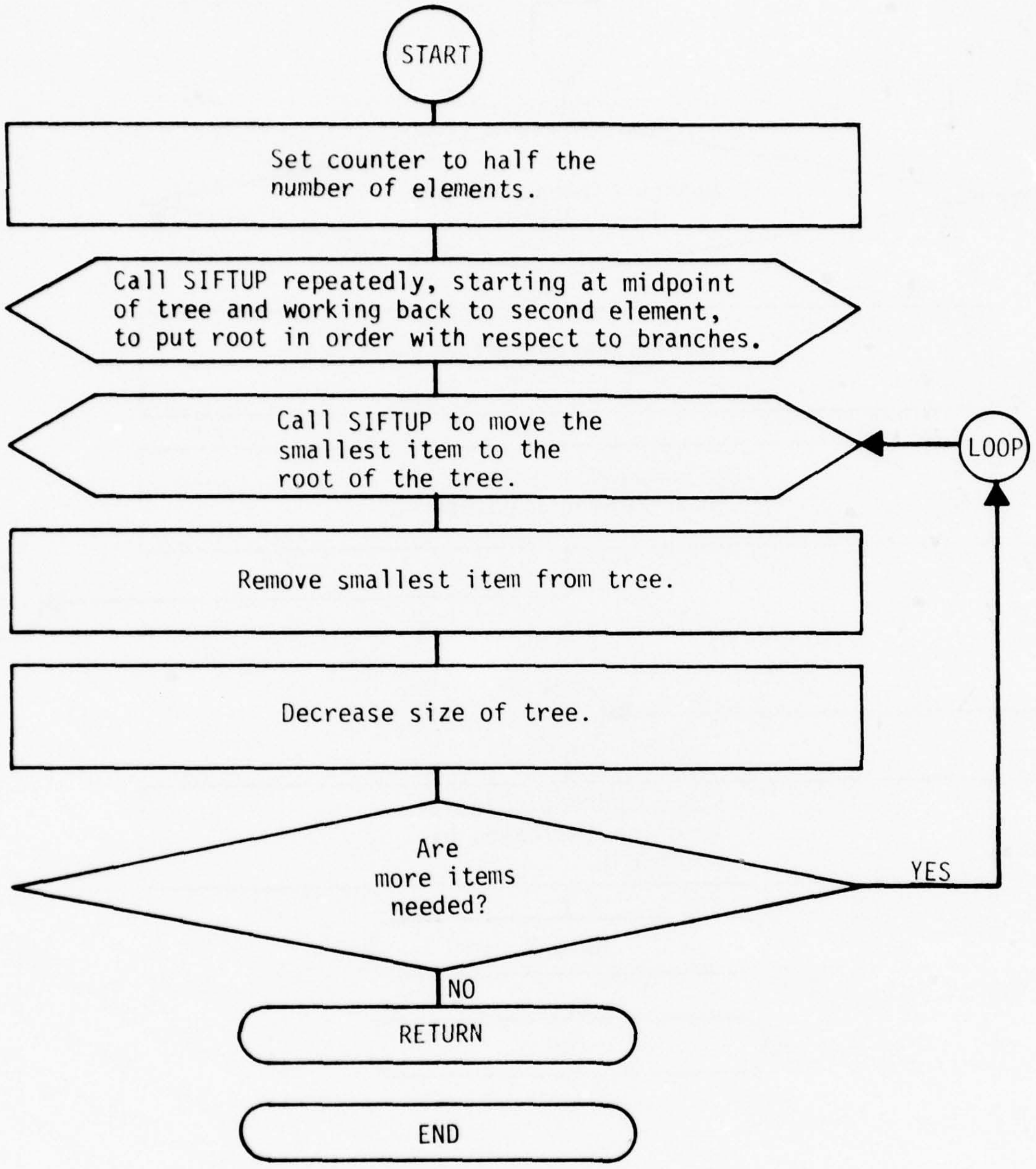
Subroutine SHLSRT


START
Save tree sub-root designated by argument L.
10
Set pointer J to left branch.
Does sub-root have two branches?
NO
6
YES
Is right branch element smaller than left branch element?
NO
YES
Set pointer J to right branch.
4
1

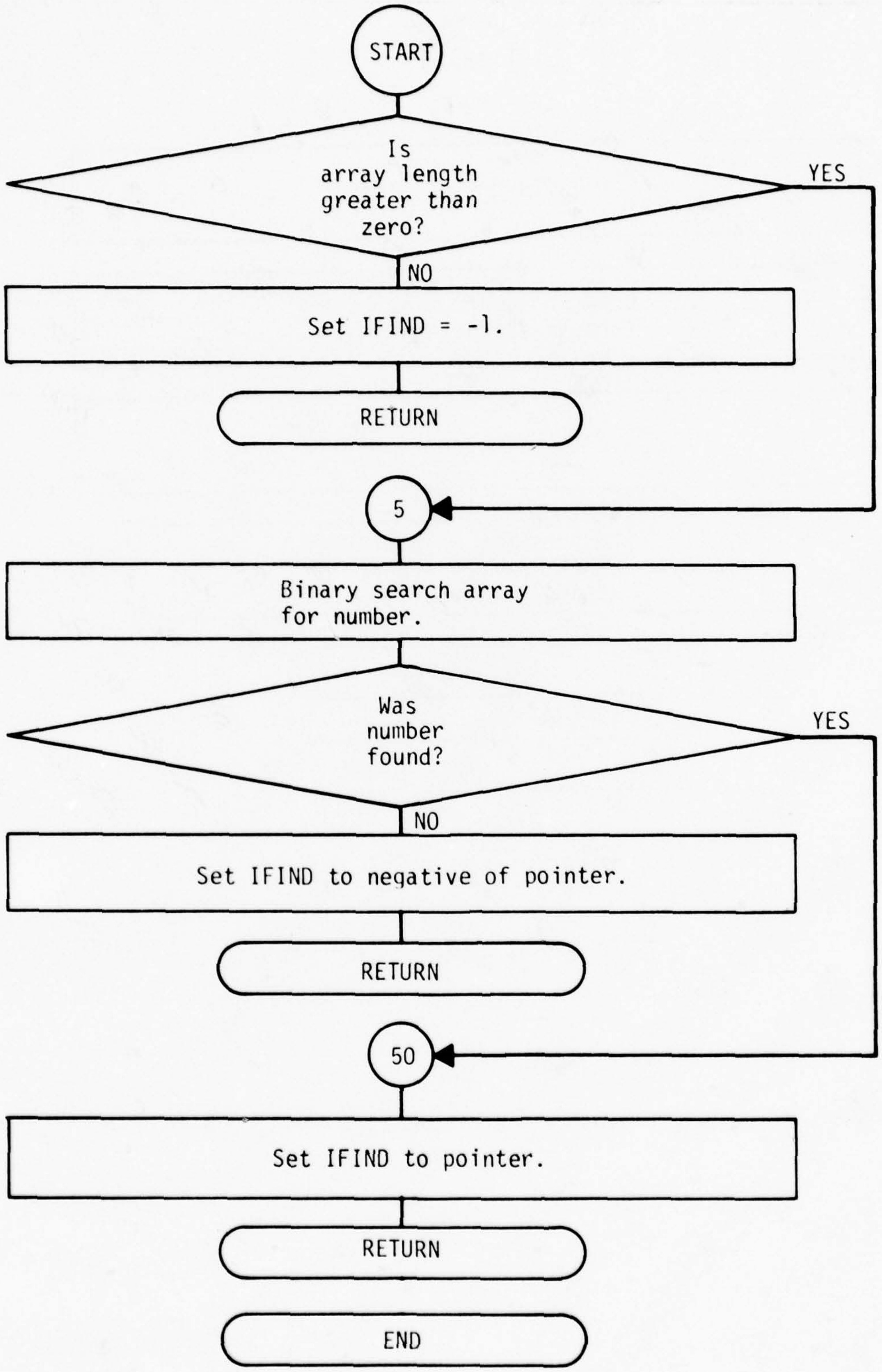
Subroutine SIFTUP


1
Is branch element smaller than subroot?
NO
YES
Replace subroot element by branch element.
Set subroot pointer I equal to branch pointer J.
10
6
Store subroot value at location indicated by pointer I.
RETURN
END

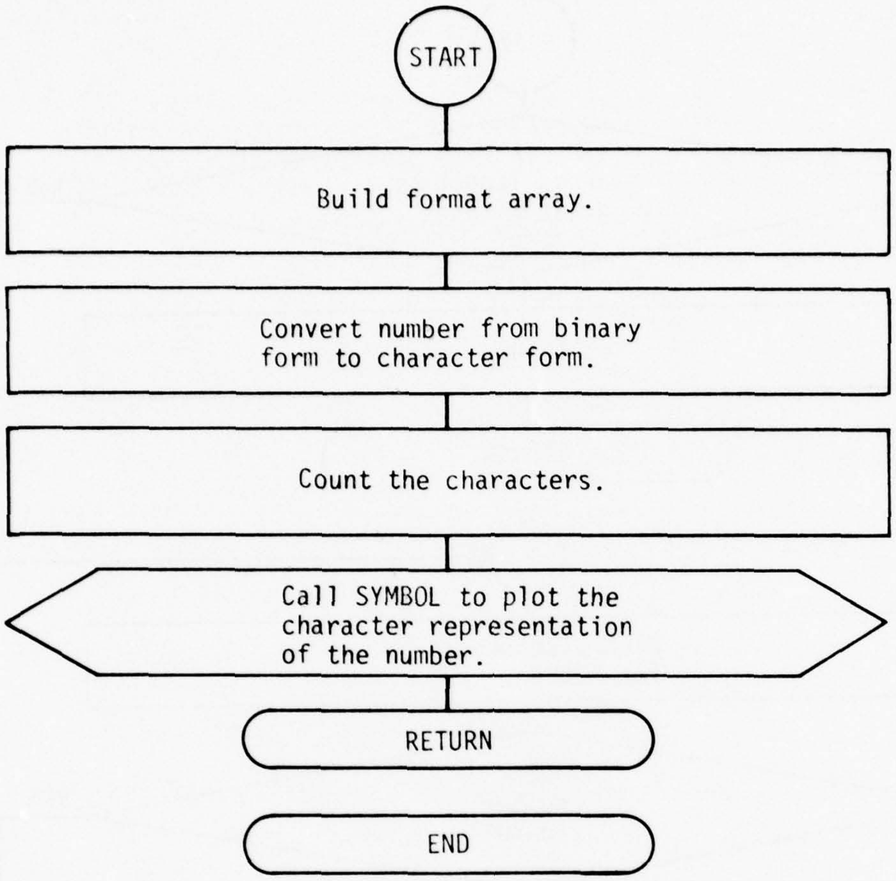
Subroutine SIFTUP


START
Set counter to half the number of elements.
Call SIFTUP repeatedly, starting at midpoint of tree and working back to second element, to put root in order with respect to branches.
Call SIFTUP to move the smallest item to the root of the tree.
LOOP
Remove smallest item from tree.
Decrease size of tree.
Are more items needed?
YES
NO
RETURN
END

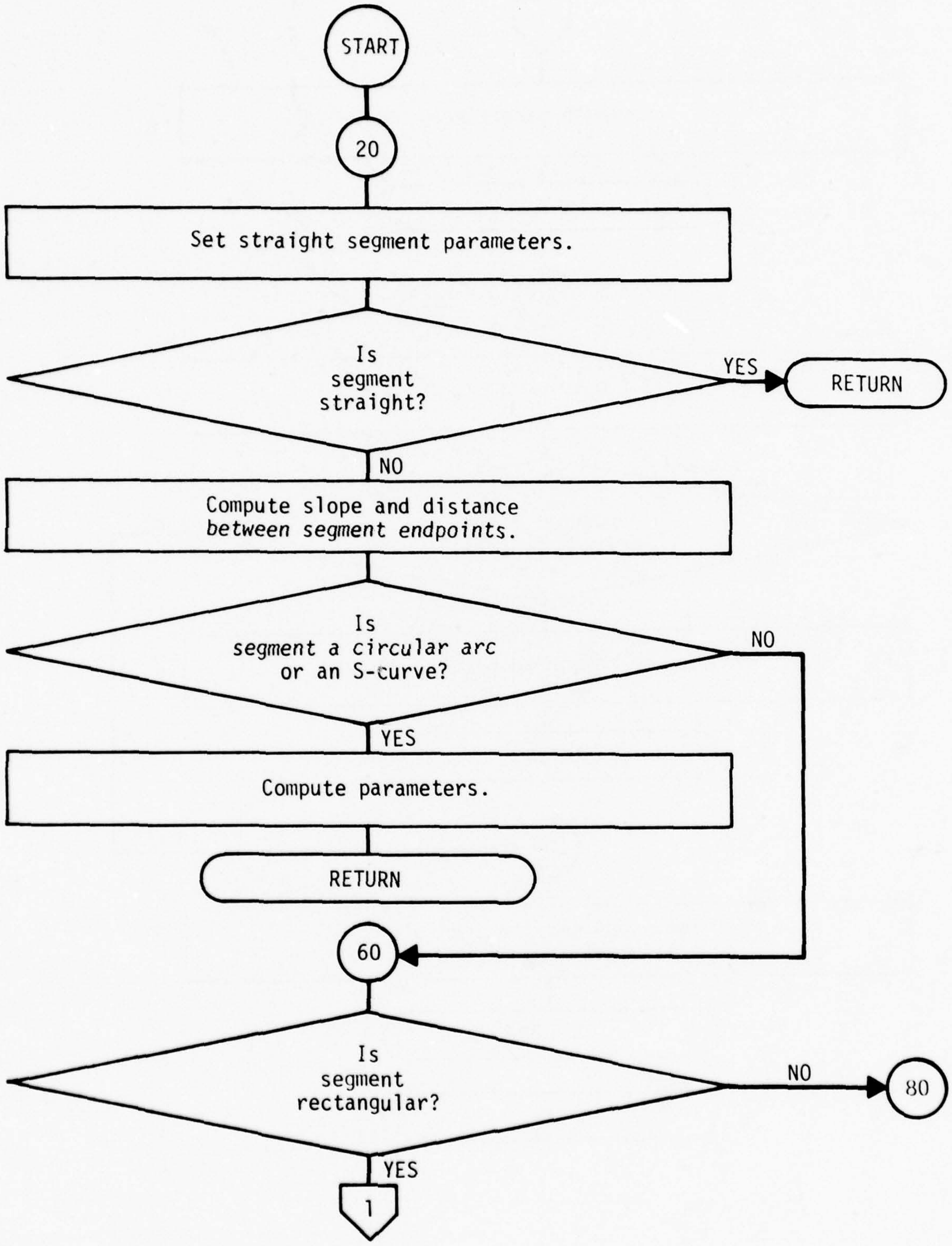
Subroutine SORTK


START
Is array length greater than zero?
YES
NO
Set IFIND = -1.
RETURN
5
Binary search array for number.
Was number found?
YES
NO
Set IFIND to negative of pointer.
RETURN
50
Set IFIND to pointer.
RETURN
END

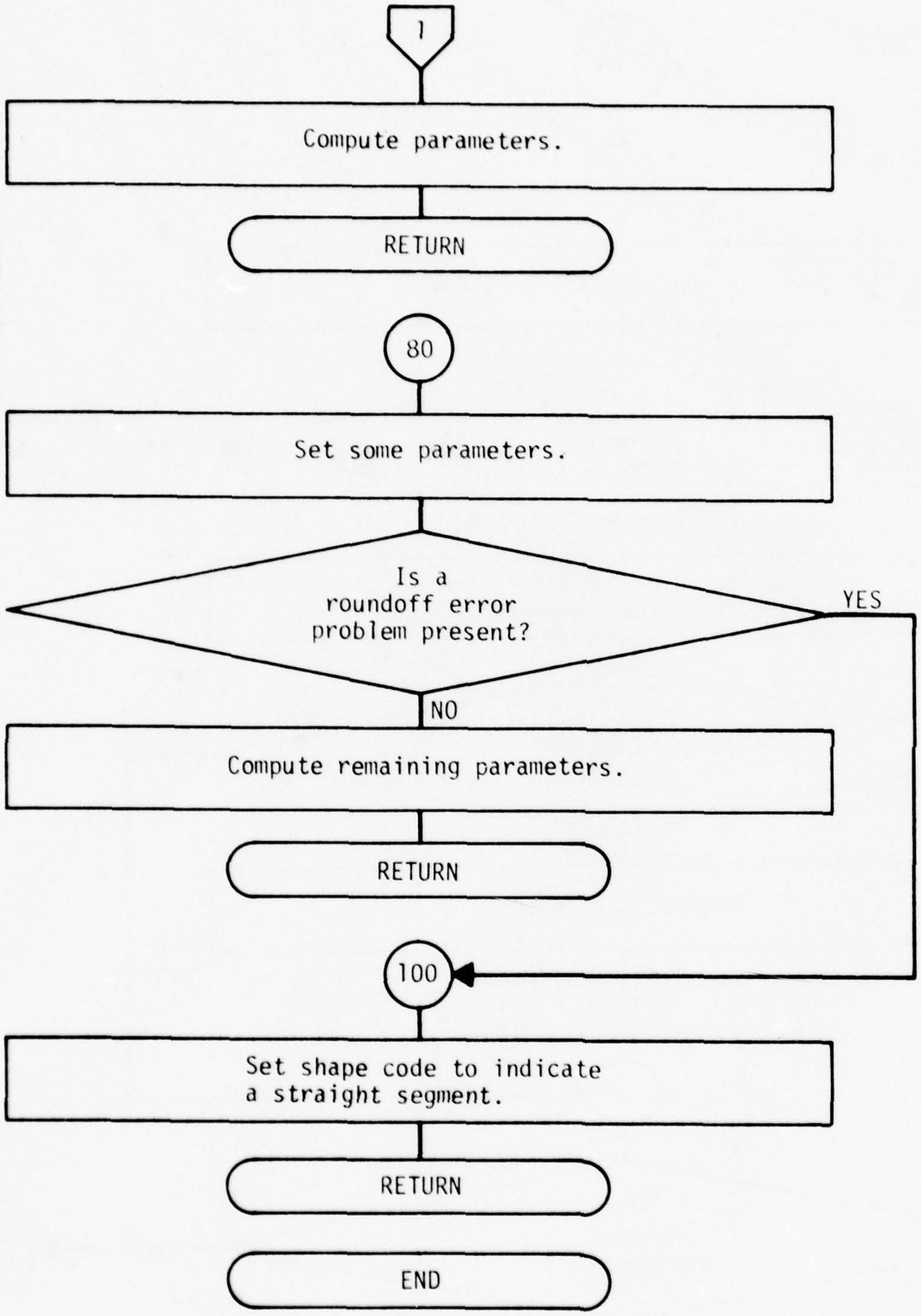
Function IFIND


START
Build format array.
Convert number from binary form to character form.
Count the characters.
Call SYMBOL to plot the character representation of the number.
RETURN
END

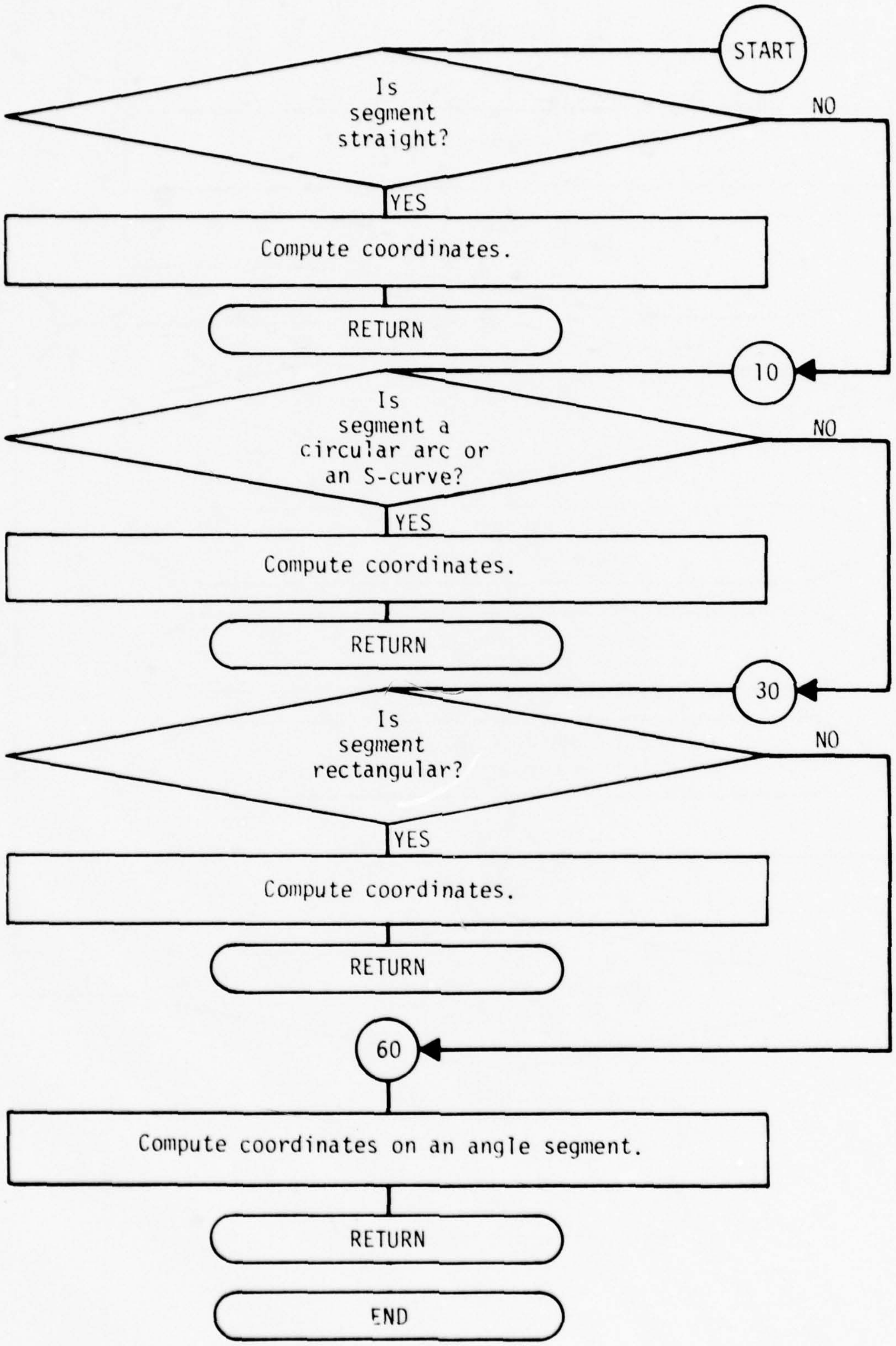
Subroutine NUMBER


START
20
Set straight segment parameters.
Is segment straight?
YES
RETURN
NO
Compute slope and distance between segment endpoints.
Is segment a circular arc or an S-curve?
NO
YES
Compute parameters.
RETURN
60
Is segment rectangular?
NO
80
YES
1

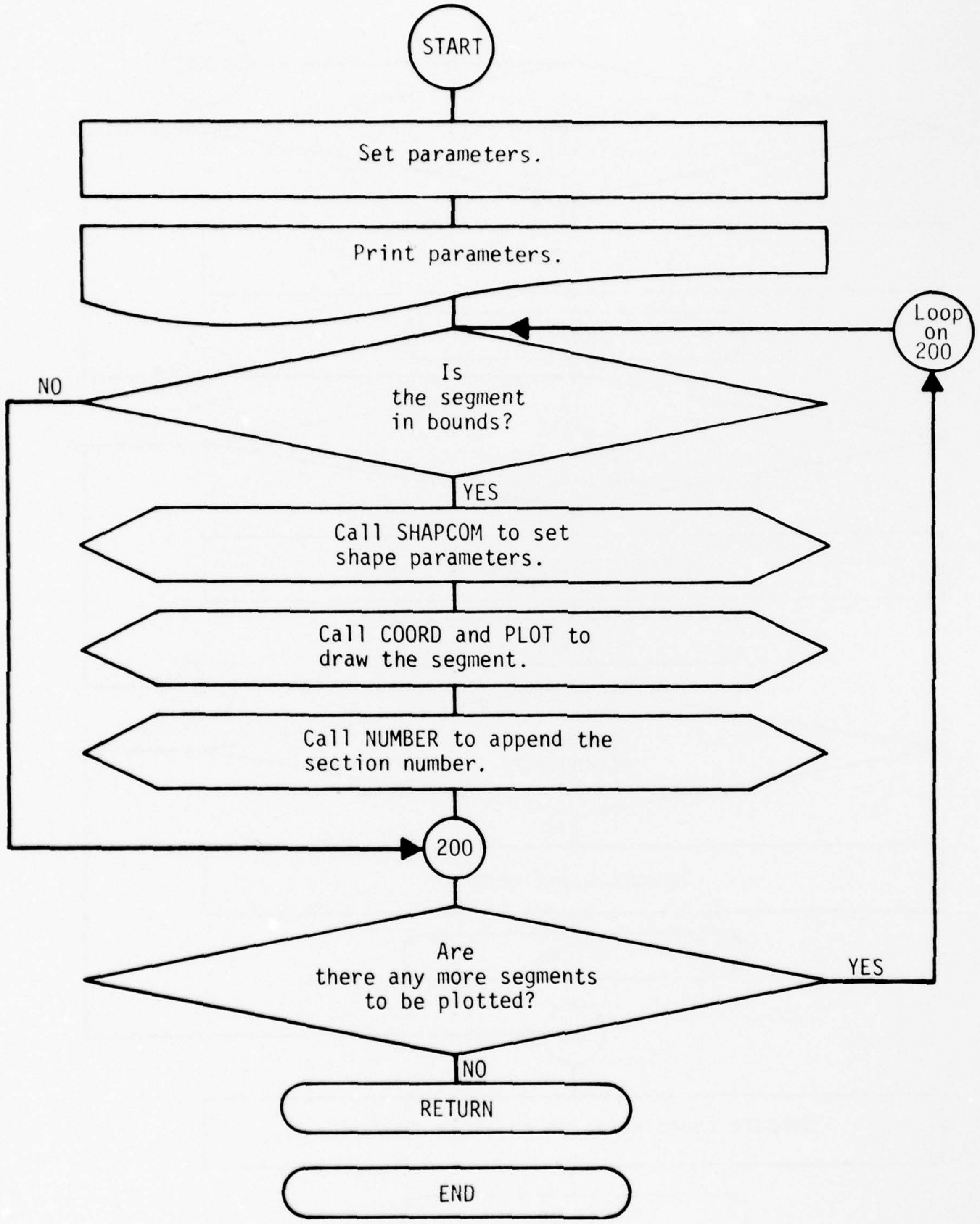
Subroutine SHAPCOM


1
Compute parameters.
RETURN
80
Set some parameters.
Is a roundoff error problem present?
YES
NO
Compute remaining parameters.
RETURN
100
Set shape code to indicate a straight segment.
RETURN
END

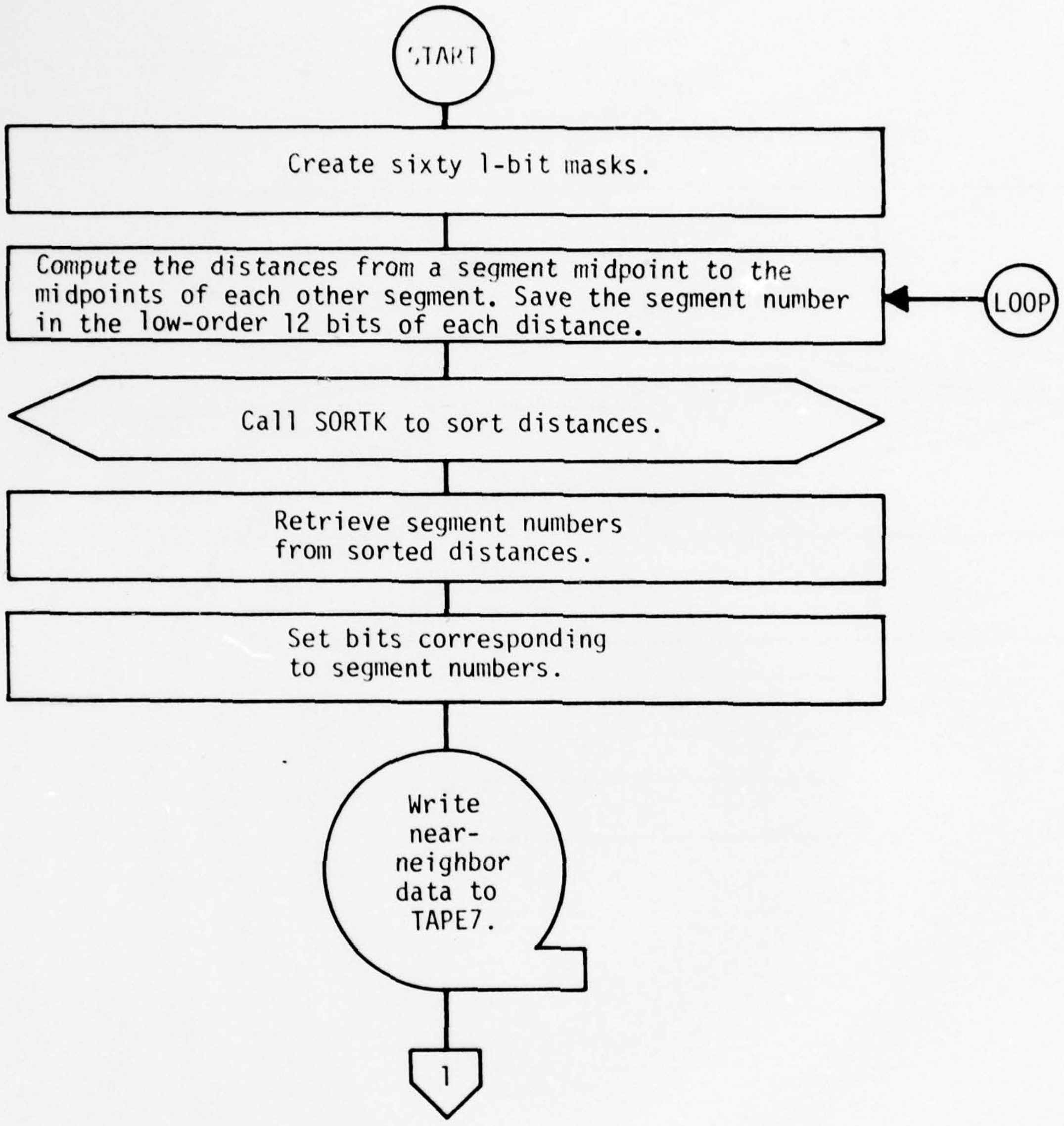
Subroutine SHAPCOM



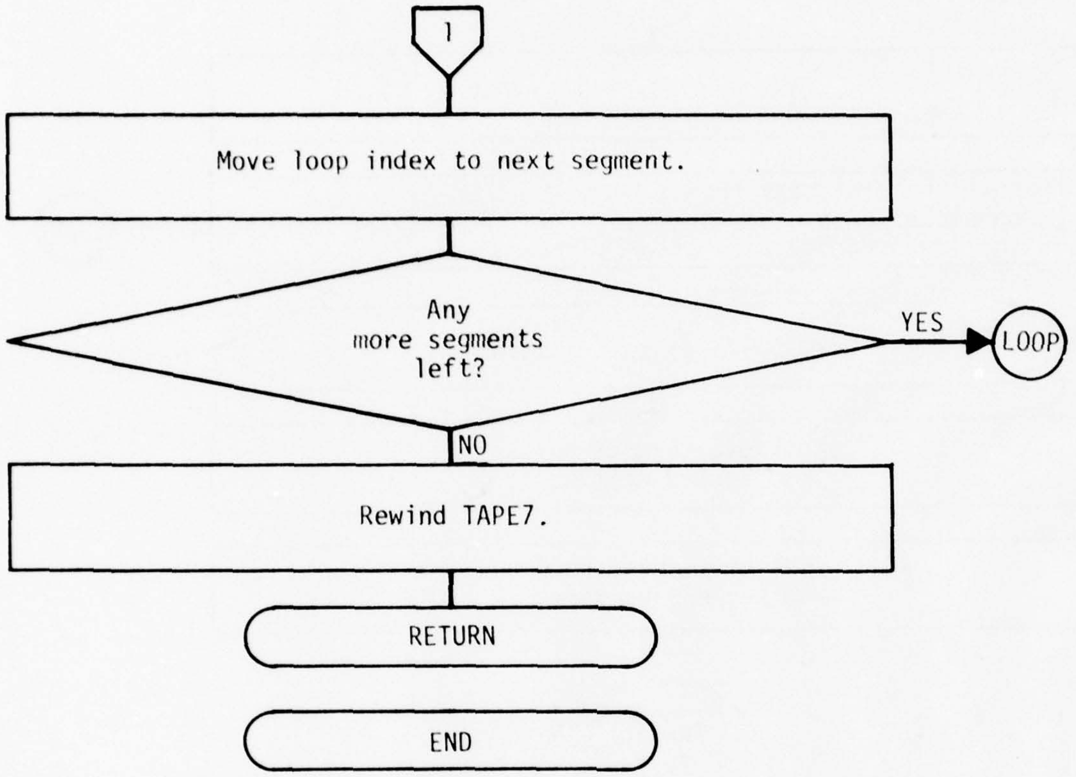
Subroutine COORD


START
Set parameters.
Print parameters.
Is the segment in bounds?
NO
YES
Loop on 200
Call SHAPCOM to set shape parameters.
Call COORD and PLOT to draw the segment.
Call NUMBER to append the section number.
200
Are there any more segments to be plotted?
YES
NO
RETURN
END

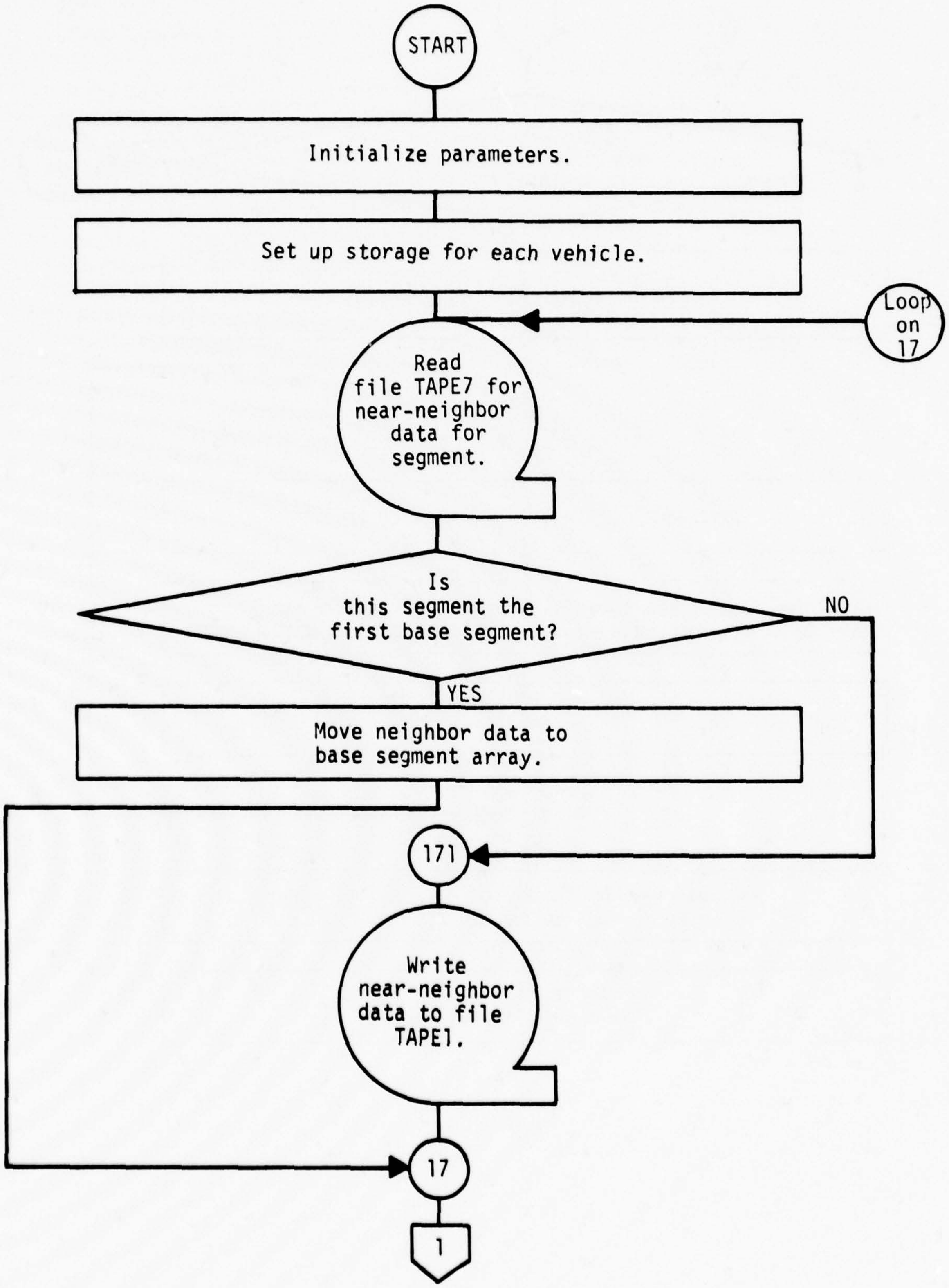
Subroutine MAPPLT


START
Create sixty 1-bit masks.
Compute the distances from a segment midpoint to the midpoints of each other segment. Save the segment number in the low-order 12 bits of each distance.
LOOP
Call SORTK to sort distances.
Retrieve segment numbers from sorted distances.
Set bits corresponding to segment numbers.
Write near-neighbor data to TAPE7.
1

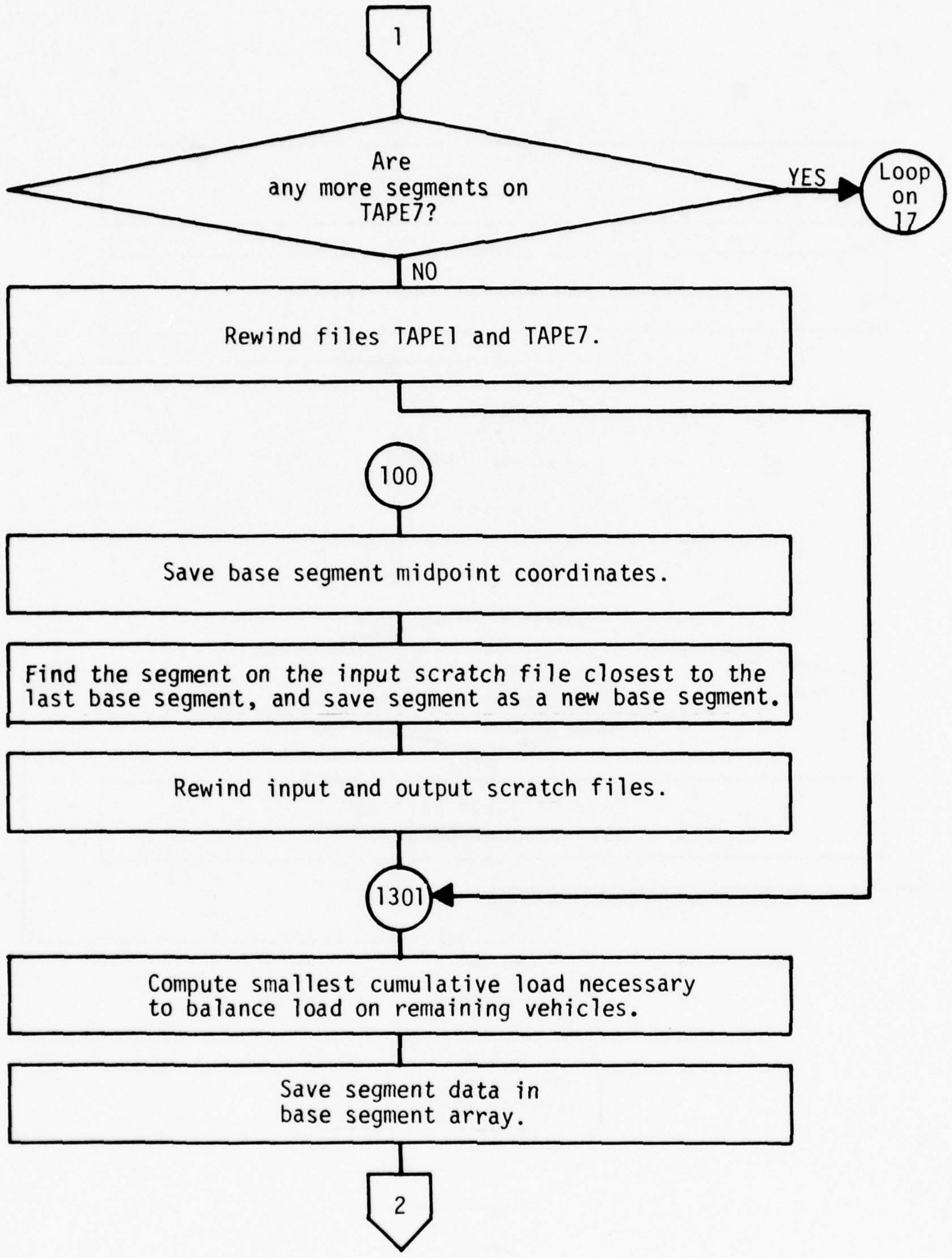
Subroutine BUILD



Subroutine BUILD

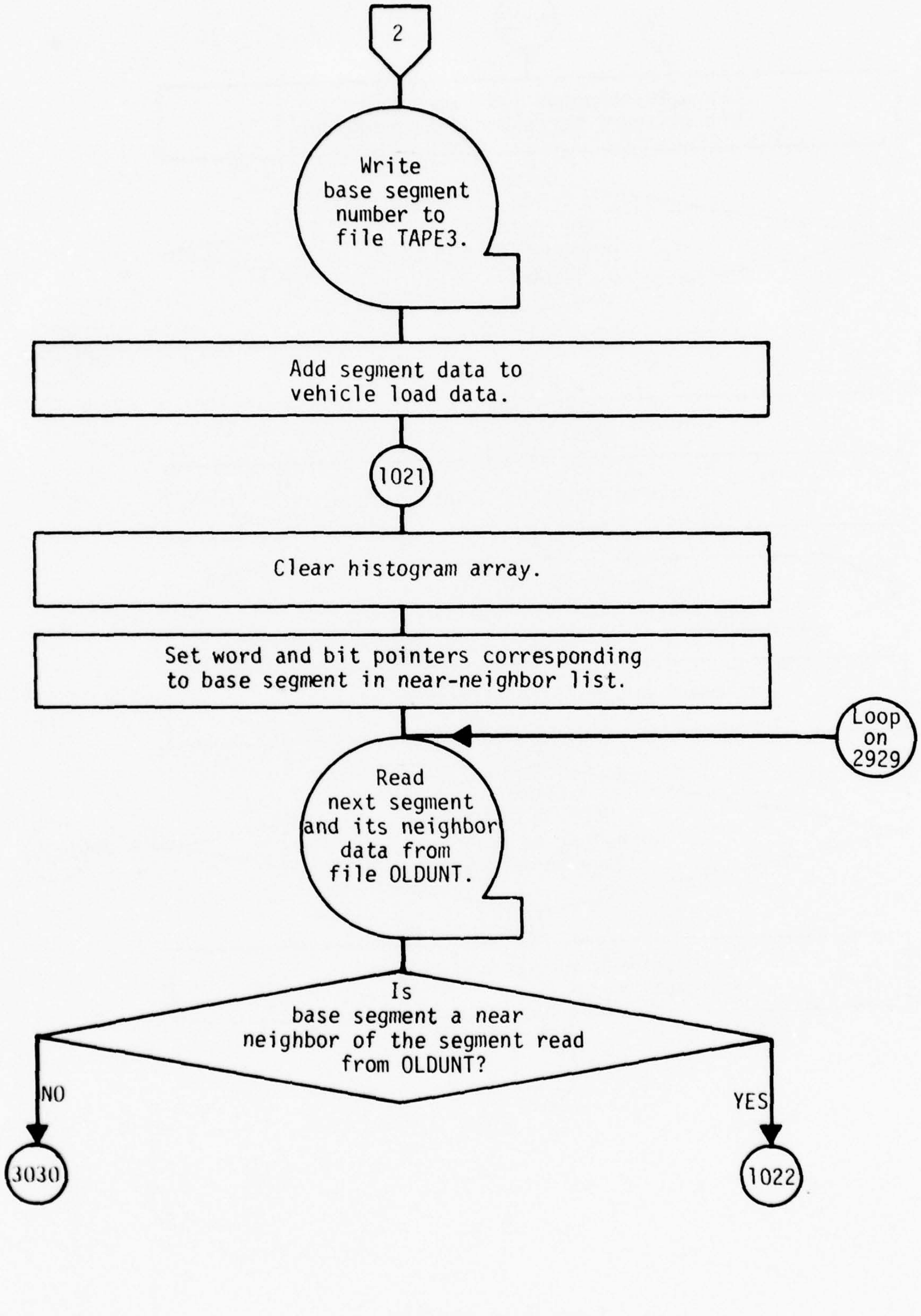

START
Initialize parameters.
Set up storage for each vehicle.
Loop on 17
Read file TAPE7 for near-neighbor data for segment.
Is this segment the first base segment?
NO
YES
Move neighbor data to base segment array.
171
Write near-neighbor data to file TAPE1.
17
1

Subroutine SECTION

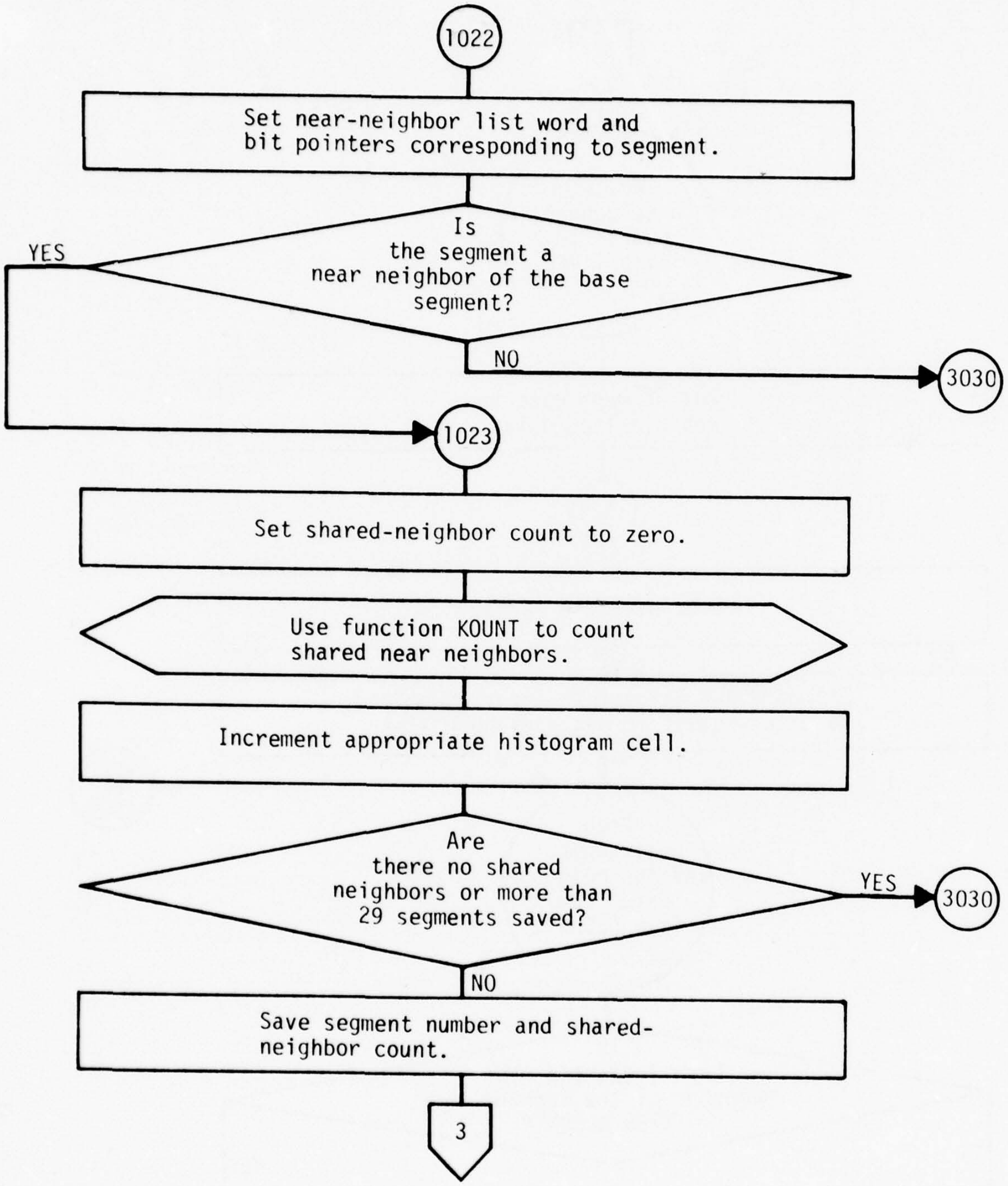


1
Are any more segments on TAPE7?
YES
Loop on 17
NO
Rewind files TAPE1 and TAPE7.
100
Save base segment midpoint coordinates.
Find the segment on the input scratch file closest to the last base segment, and save segment as a new base segment.
Rewind input and output scratch files.
1301
Compute smallest cumulative load necessary to balance load on remaining vehicles.
Save segment data in base segment array.
2

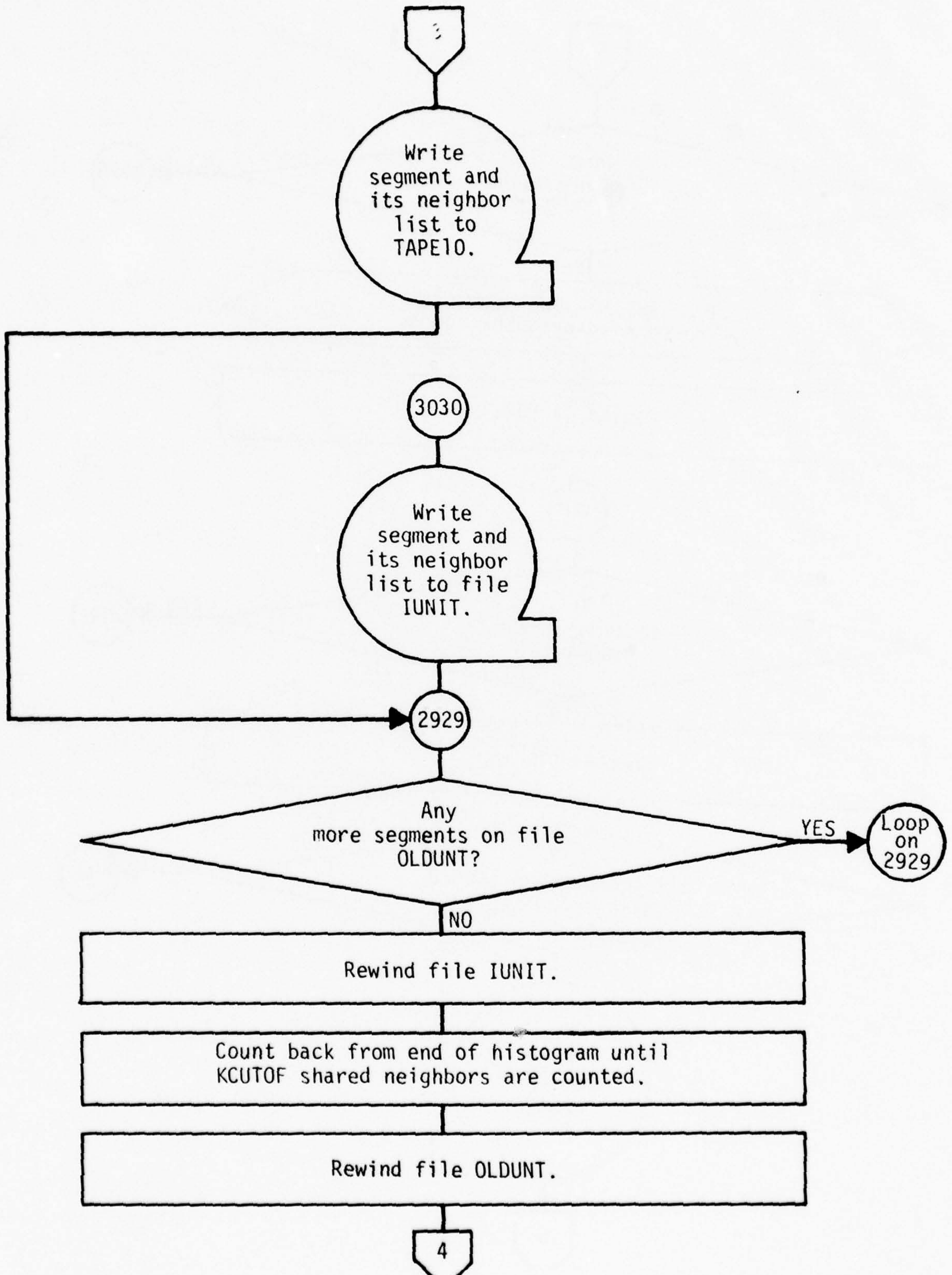
Subroutine SECTION


2
Write base segment number to file TAPE3.
Add segment data to vehicle load data.
1021
Clear histogram array.
Set word and bit pointers corresponding to base segment in near-neighbor list.
Loop on 2929
Read next segment and its neighbor data from file OLDUNT.
Is base segment a near neighbor of the segment read from OLDUNT?
NO
YES
3030
1022

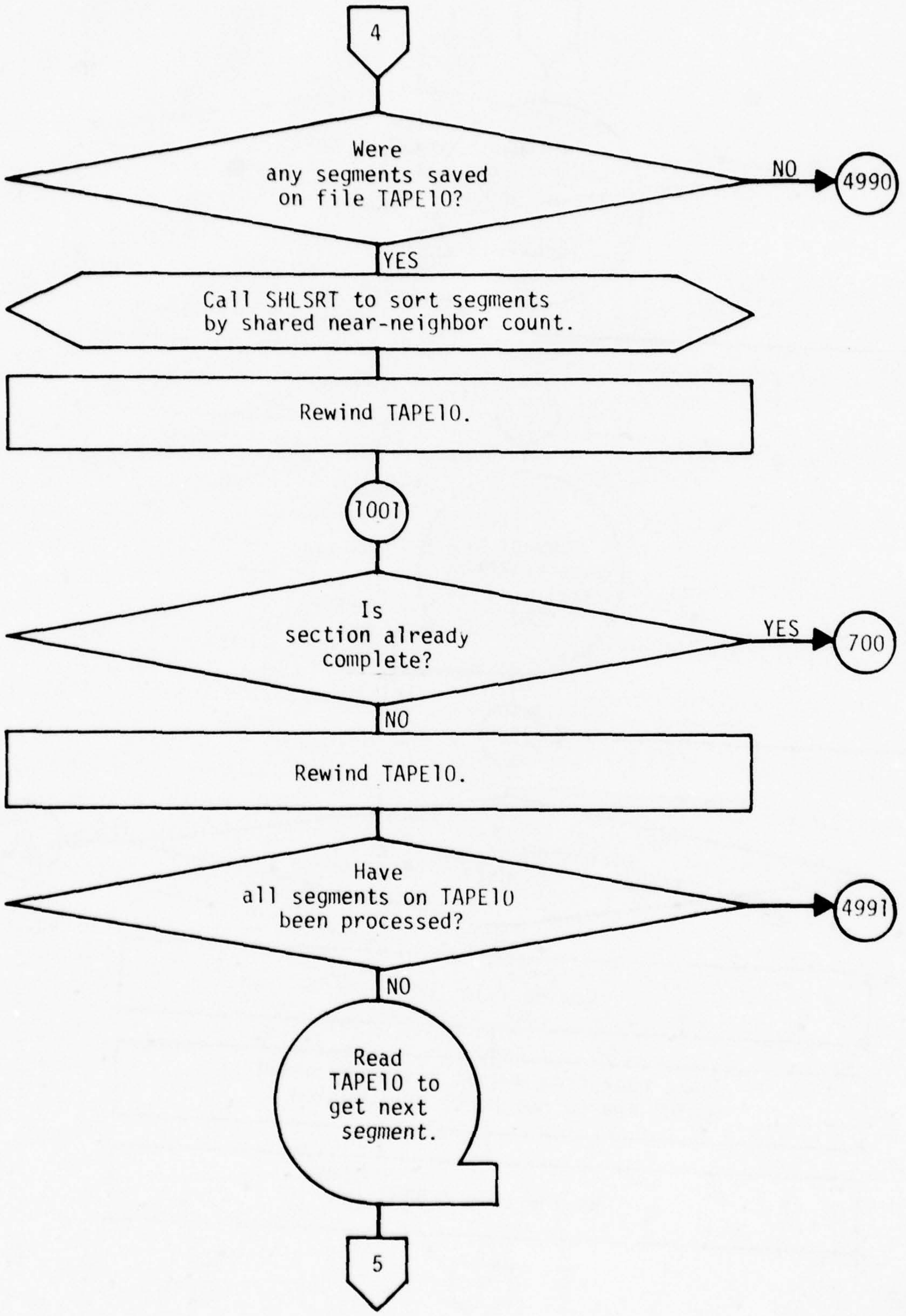
Subroutine SECTION


1022
Set near-neighbor list word and bit pointers corresponding to segment.
Is the segment a near neighbor of the base segment?
YES
NO
3030
1023
Set shared-neighbor count to zero.
Use function KOUNT to count shared near neighbors.
Increment appropriate histogram cell.
Are there no shared neighbors or more than 29 segments saved?
YES
3030
NO
Save segment number and shared-neighbor count.
3

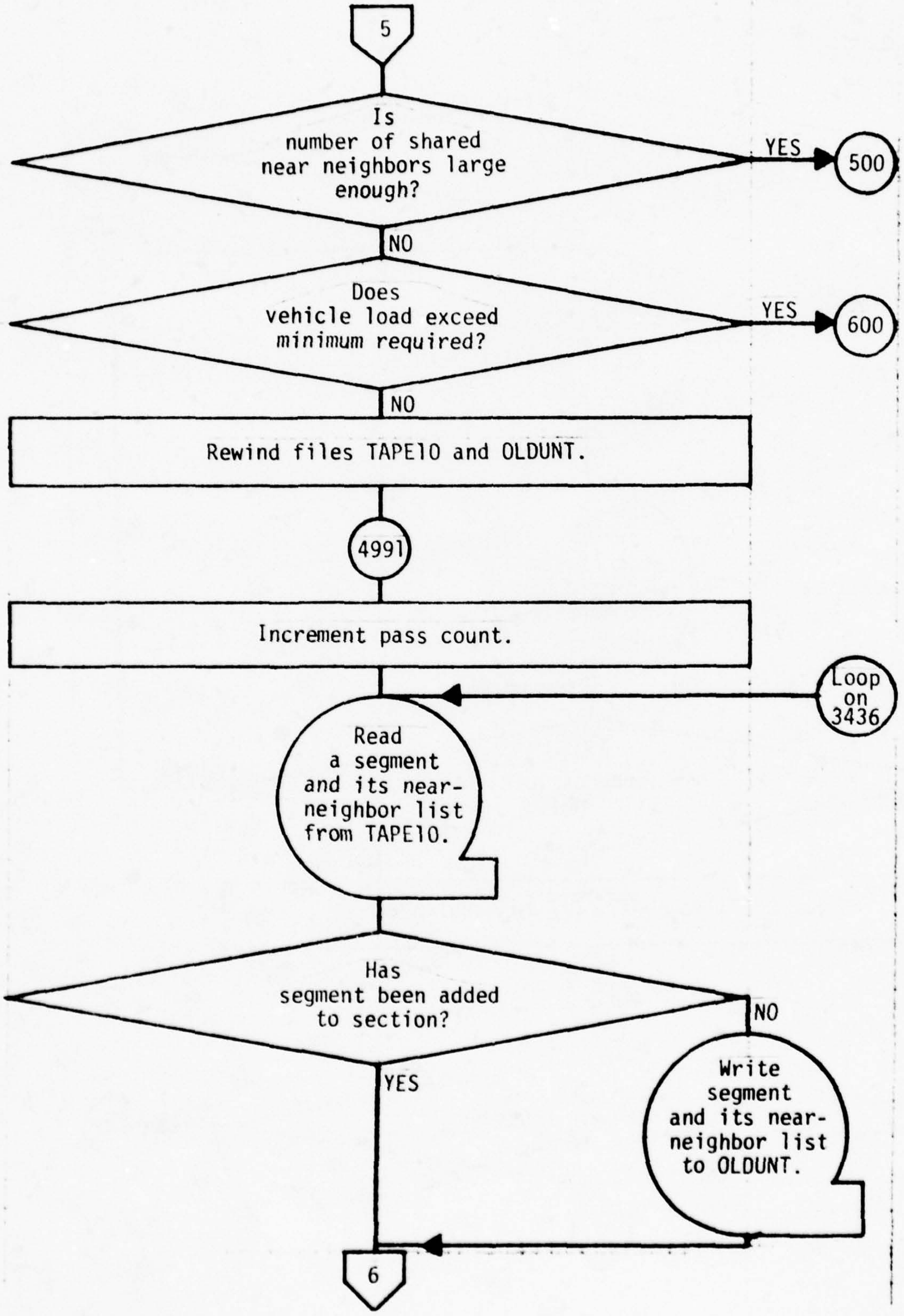
Subroutine SECTION


3
Write segment and its neighbor list to TAPE10.
3030
Write segment and its neighbor list to file IUNIT.
2929
Any more segments on file OLDUNT?
YES
Loop on 2929
NO
Rewind file IUNIT.
Count back from end of histogram until KCUTOF shared neighbors are counted.
Rewind file OLDUNT.
4

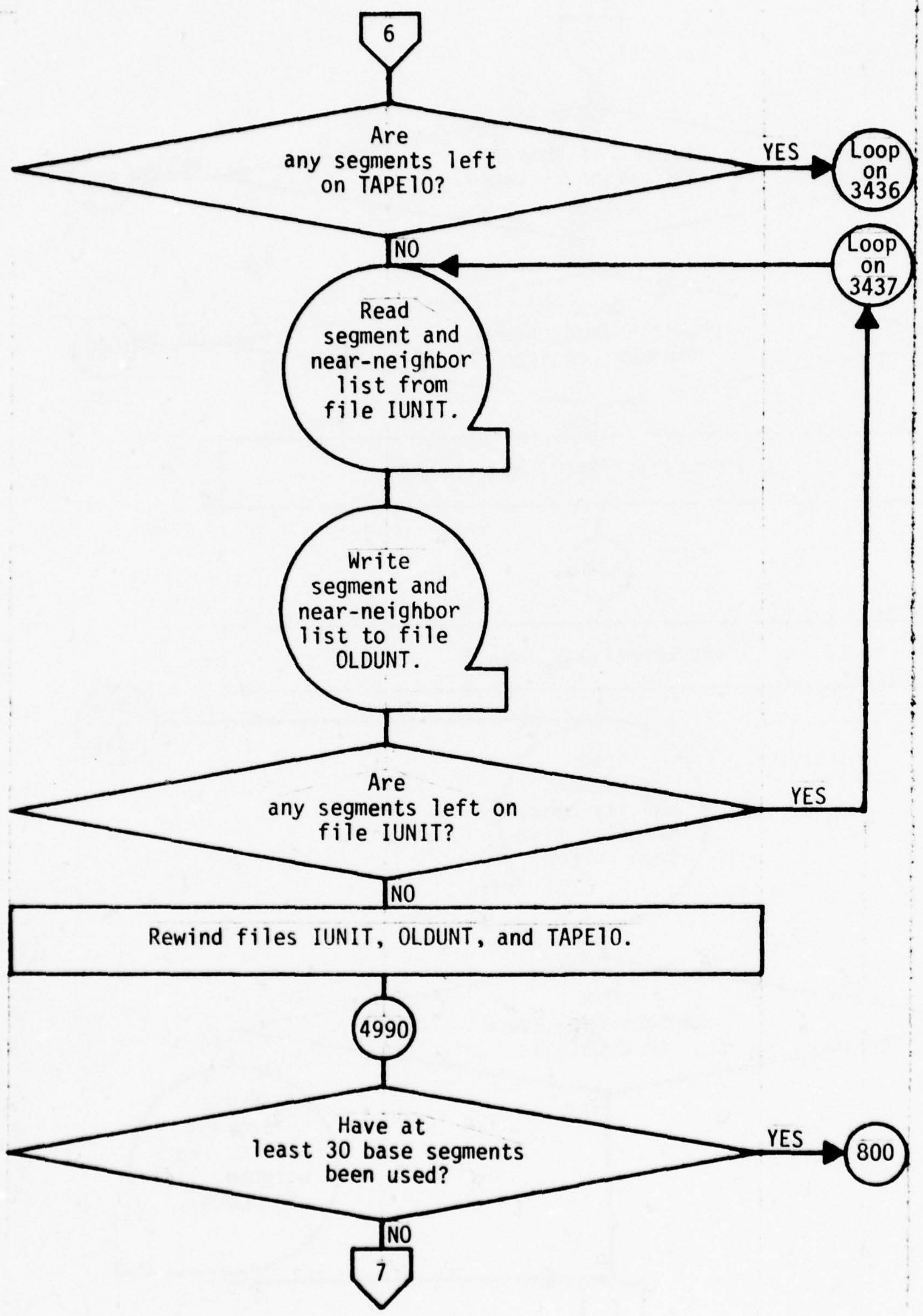
Subroutine SECTION


4
Were any segments saved on file TAPE10?
NO
4990
YES
Call SHLSRT to sort segments by shared near-neighbor count.
Rewind TAPE10.
1001
Is section already complete?
YES
700
NO
Rewind TAPE10.
Have all segments on TAPE10 been processed?
4991
NO
Read TAPE10 to get next segment.
5

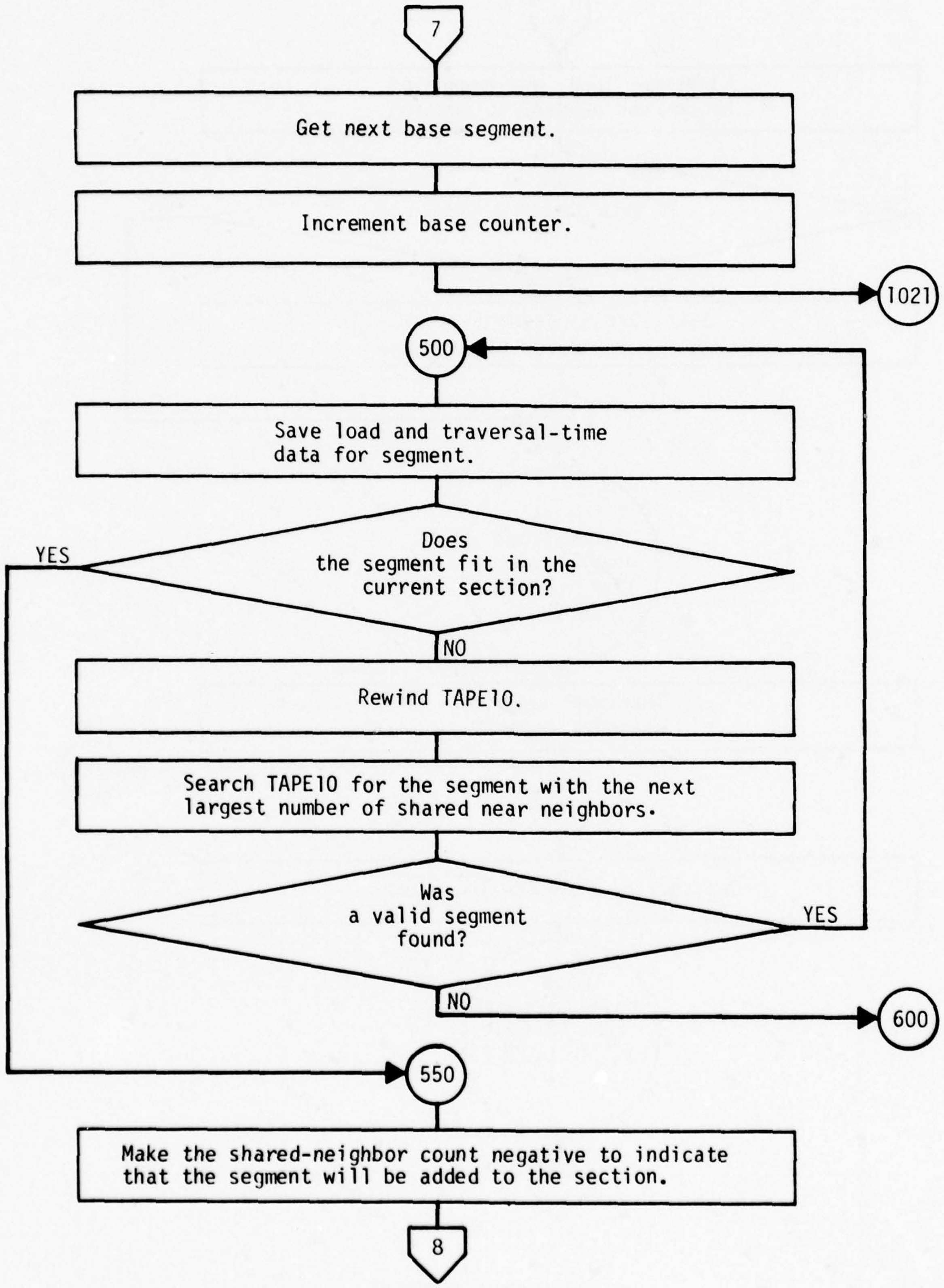
Subroutine SECTION


5
Is number of shared near neighbors large enough?
YES
500
NO
Does vehicle load exceed minimum required?
YES
600
NO
Rewind files TAPE10 and OLDUNT.
4991
Increment pass count.
Loop on 3436
Read a segment and its near-neighbor list from TAPE10.
Has segment been added to section?
NO
Write segment and its near-neighbor list to OLDUNT.
YES
6

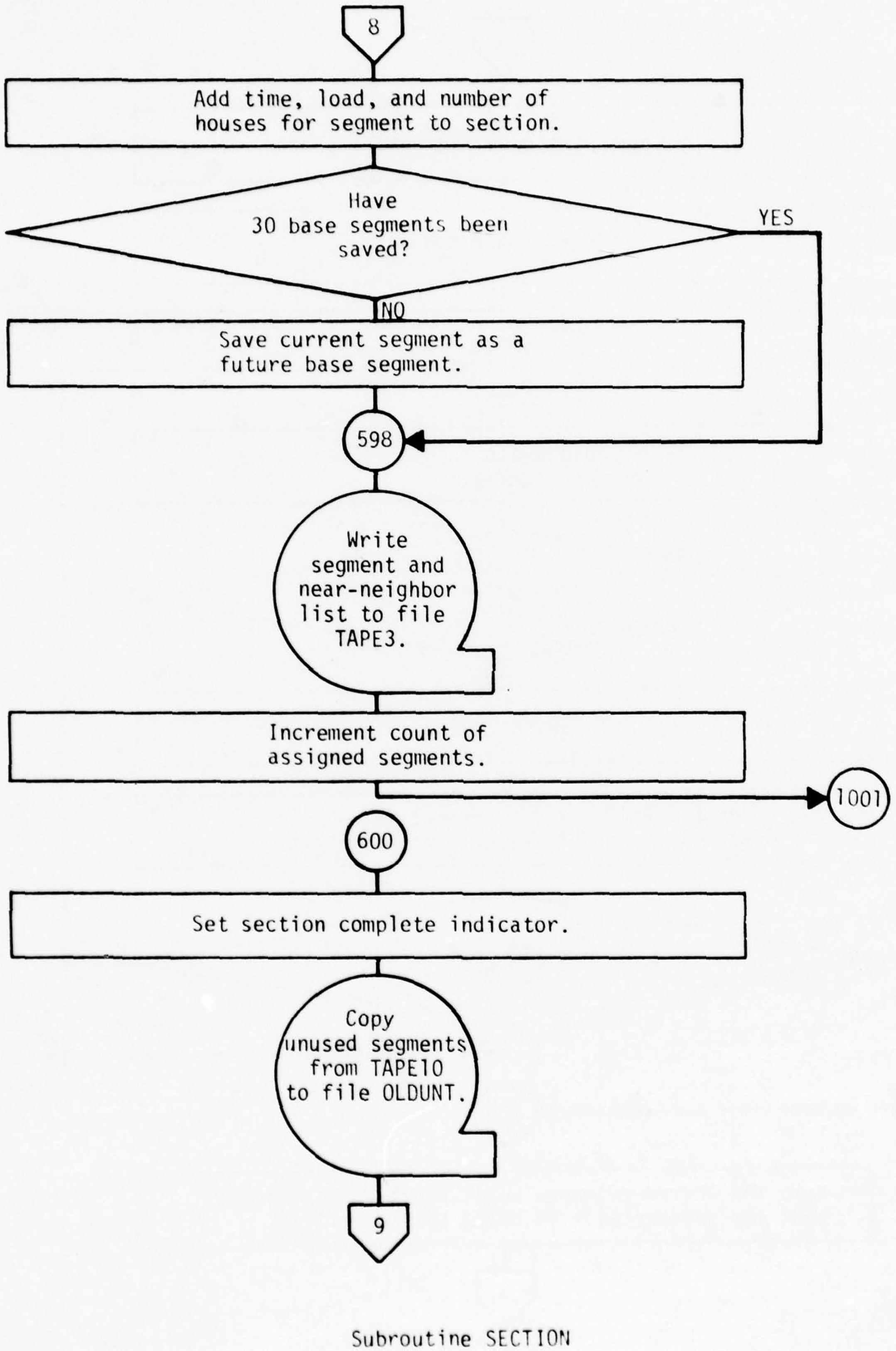
Subroutine SECTION


6
Are any segments left on TAPE10?
YES
Loop on 3436
NO
Loop on 3437
Read segment and near-neighbor list from file IUNIT.
Write segment and near-neighbor list to file OLDUNT.
Are any segments left on file IUNIT?
YES
NO
Rewind files IUNIT, OLDUNT, and TAPE10.
4990
Have at least 30 base segments been used?
YES
800
NO
7

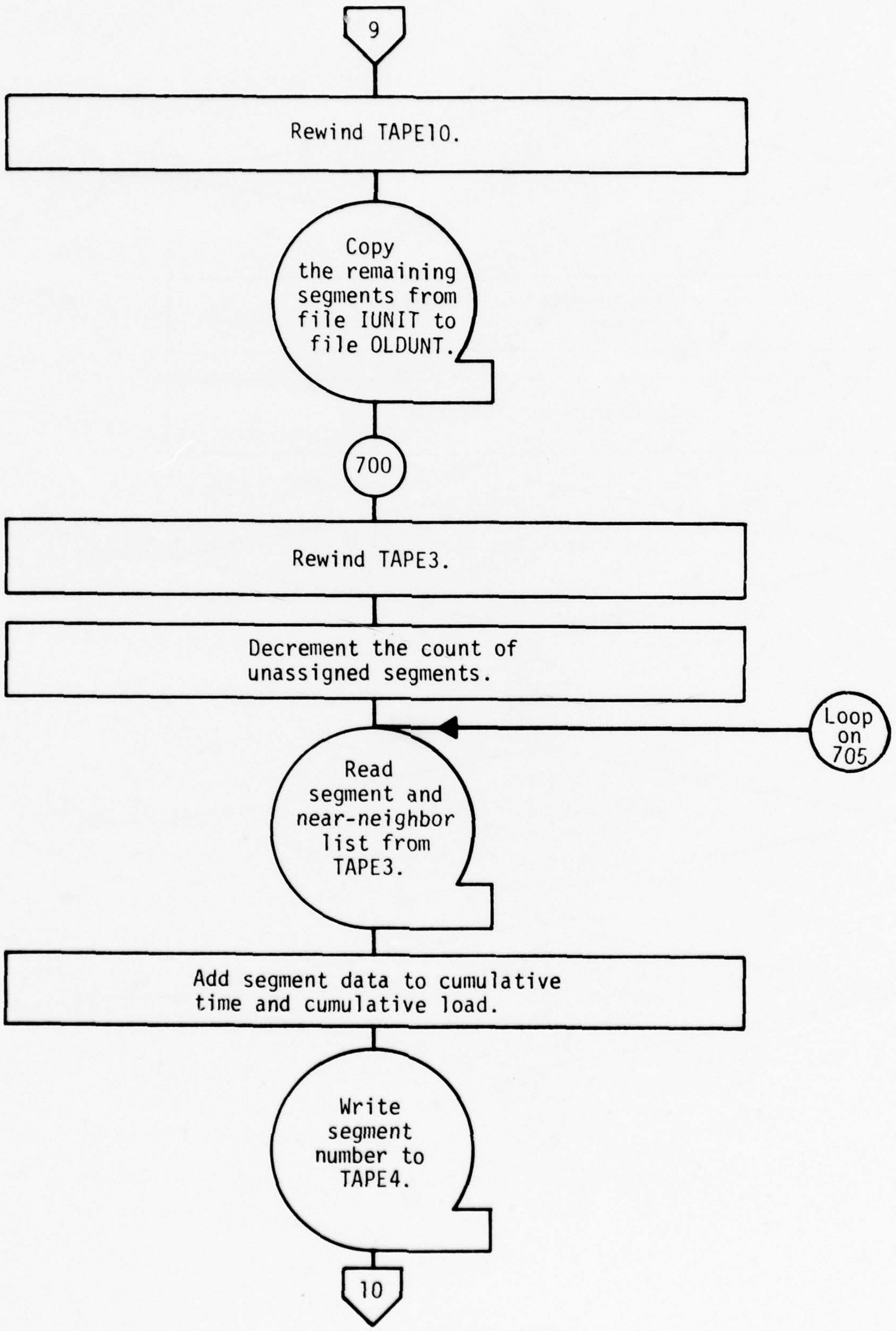
Subroutine SECTION


7
Get next base segment.
Increment base counter.
1021
500
Save load and traversal-time data for segment.
Does the segment fit in the current section?
YES
NO
Rewind TAPE10.
Search TAPE10 for the segment with the next largest number of shared near neighbors.
Was a valid segment found?
YES
NO
600
550
Make the shared-neighbor count negative to indicate that the segment will be added to the section.
8

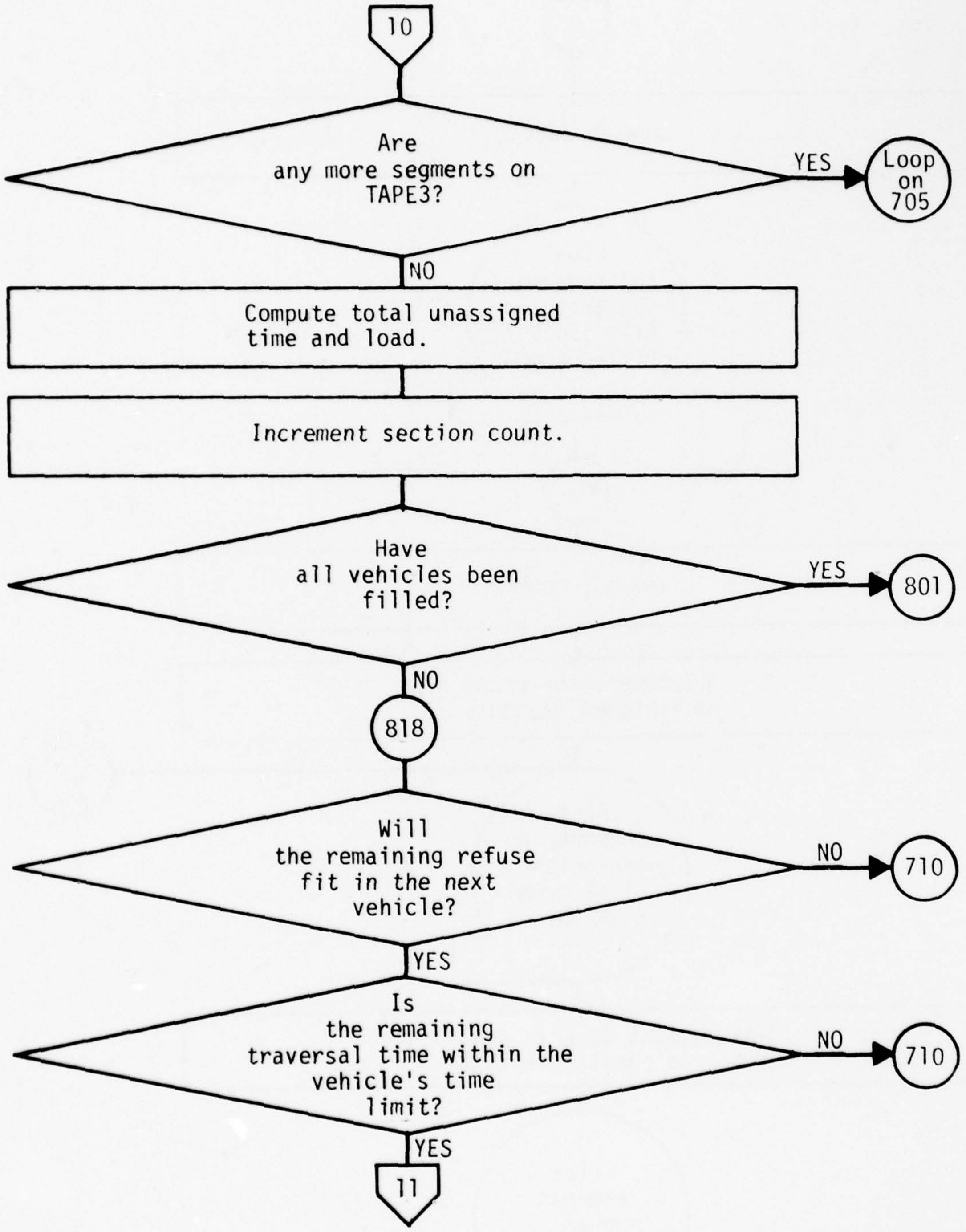
Subroutine SECTION


8
Add time, load, and number of houses for segment to section.
Have 30 base segments been saved?
YES
NO
Save current segment as a future base segment.
598
Write segment and near-neighbor list to file TAPE3.
Increment count of assigned segments.
1001
600
Set section complete indicator.
Copy unused segments from TAPE10 to file OLDUNT.
9

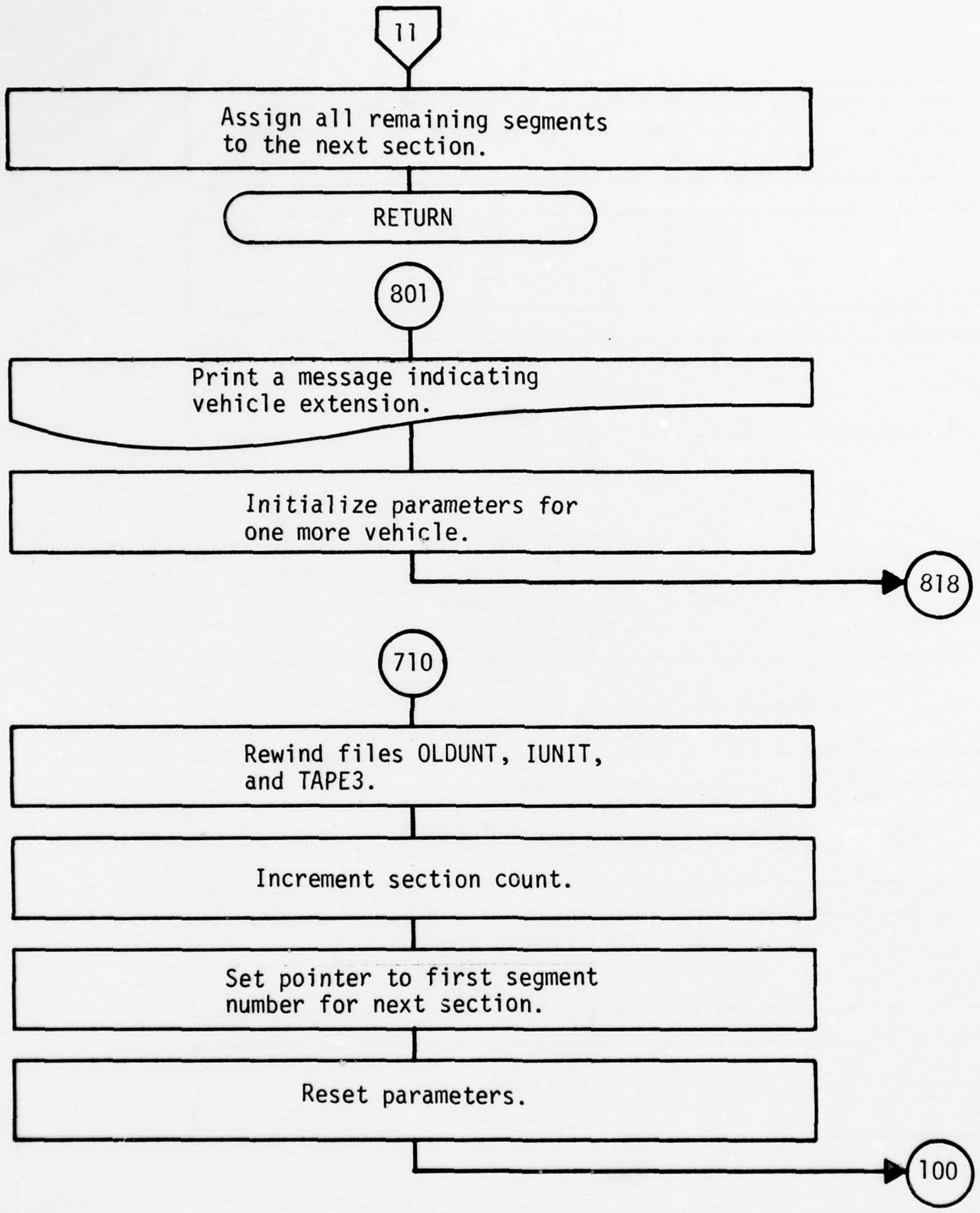
Subroutine SECTION


9
Rewind TAPE10.
Copy the remaining segments from file IUNIT to file OLDUNT.
700
Rewind TAPE3.
Decrement the count of unassigned segments.
Loop on 705
Read segment and near-neighbor list from TAPE3.
Add segment data to cumulative time and cumulative load.
Write segment number to TAPE4.
10

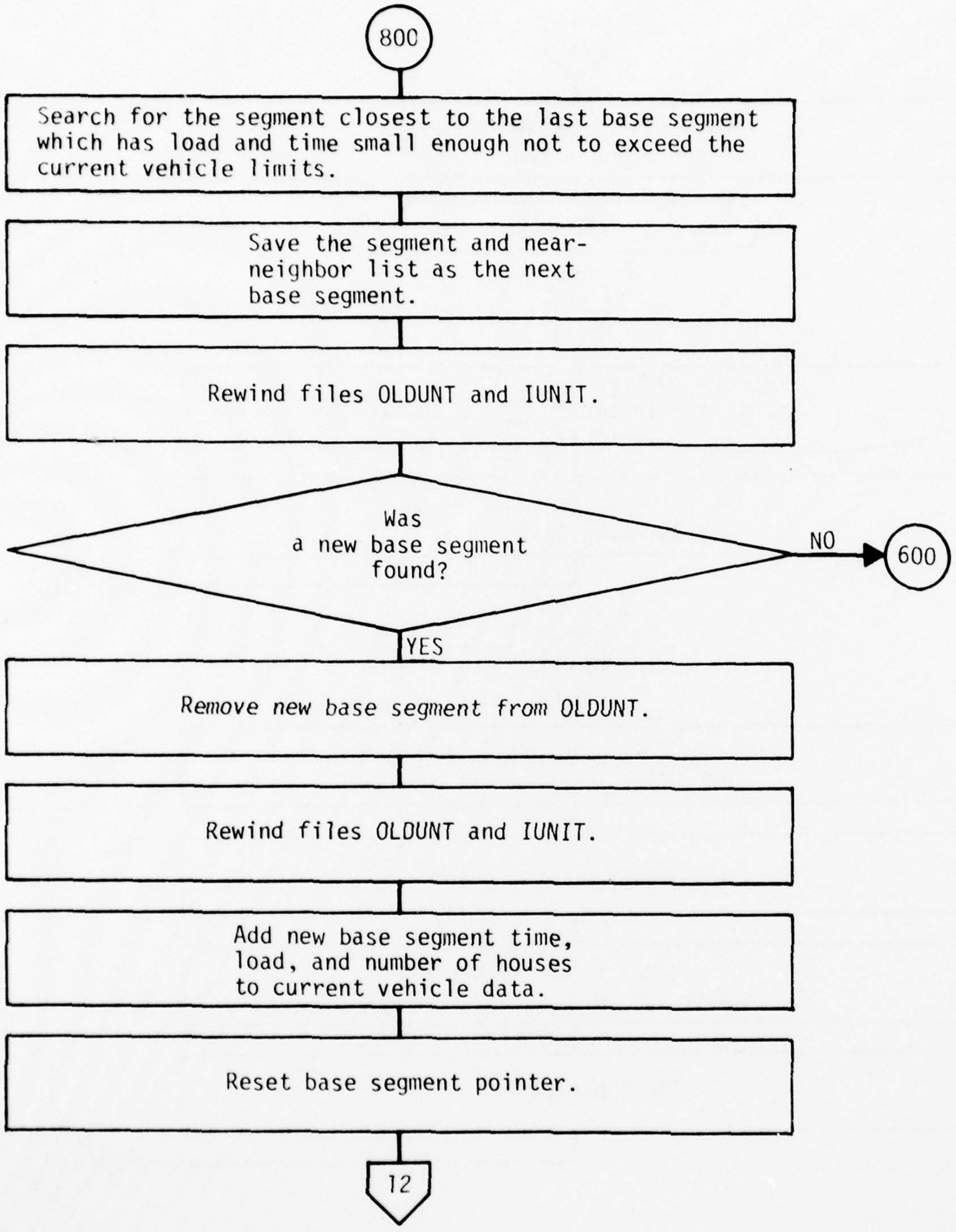
Subroutine SECTION


10
Are any more segments on TAPE3?
YES
Loop on 705
NO
Compute total unassigned time and load.
Increment section count.
Have all vehicles been filled?
YES
801
NO
818
Will the remaining refuse fit in the next vehicle?
NO
710
YES
Is the remaining traversal time within the vehicle's time limit?
NO
710
YES
11

Subroutine SECTION

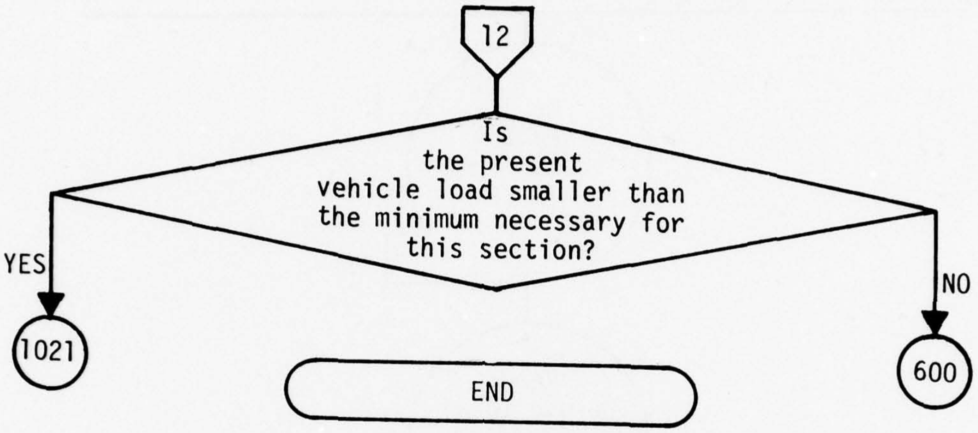

11
Assign all remaining segments to the next section.
RETURN
801
Print a message indicating vehicle extension.
Initialize parameters for one more vehicle.
818
710
Rewind files OLDUNT, IUNIT, and TAPE3.
Increment section count.
Set pointer to first segment number for next section.
Reset parameters.
100

Subroutine SECTION

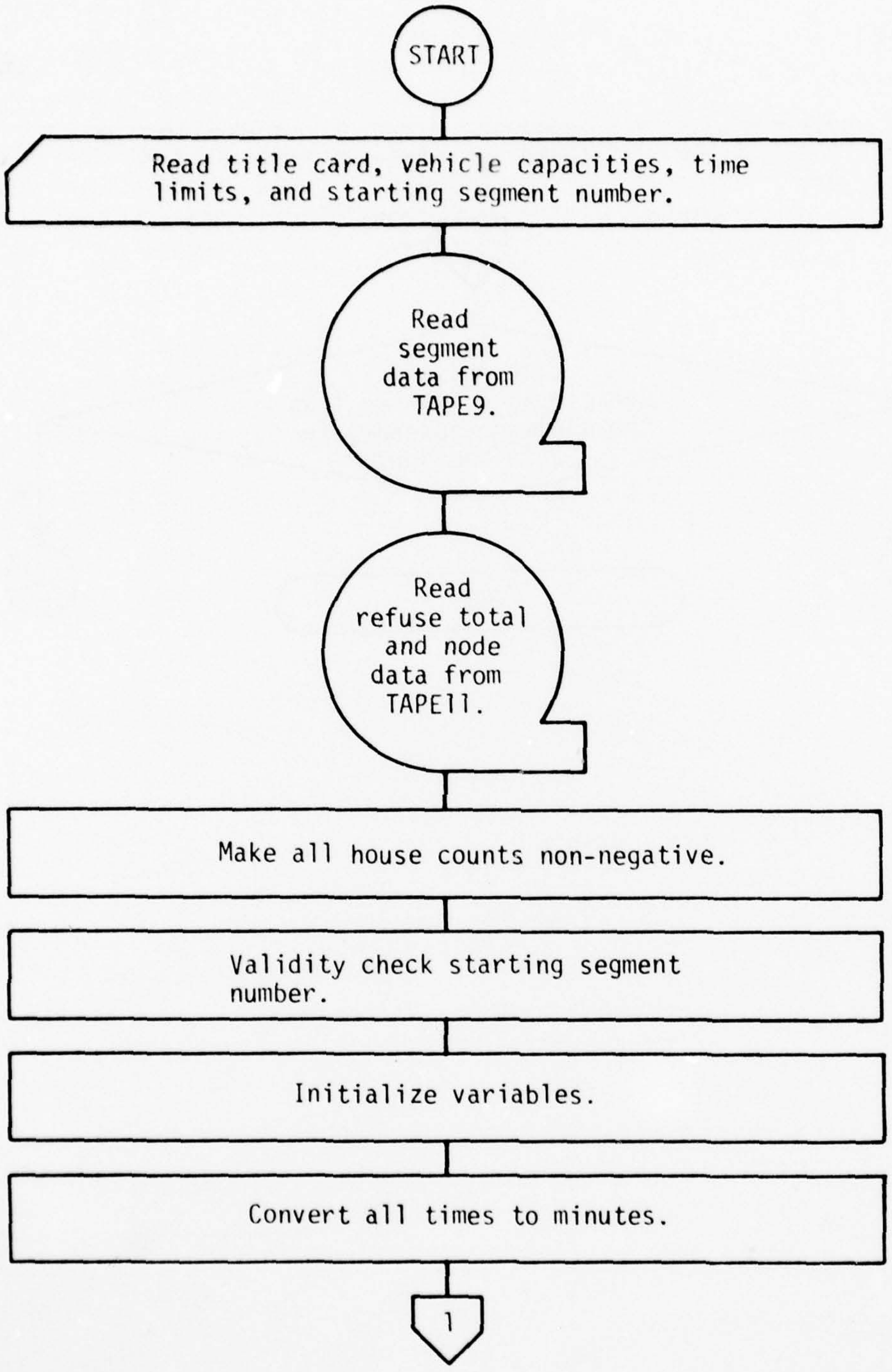


800
Search for the segment closest to the last base segment which has load and time small enough not to exceed the current vehicle limits.
Save the segment and near-neighbor list as the next base segment.
Rewind files OLDUNT and IUNIT.
Was a new base segment found?
NO
600
YES
Remove new base segment from OLDUNT.
Rewind files OLDUNT and IUNIT.
Add new base segment time, load, and number of houses to current vehicle data.
Reset base segment pointer.
12

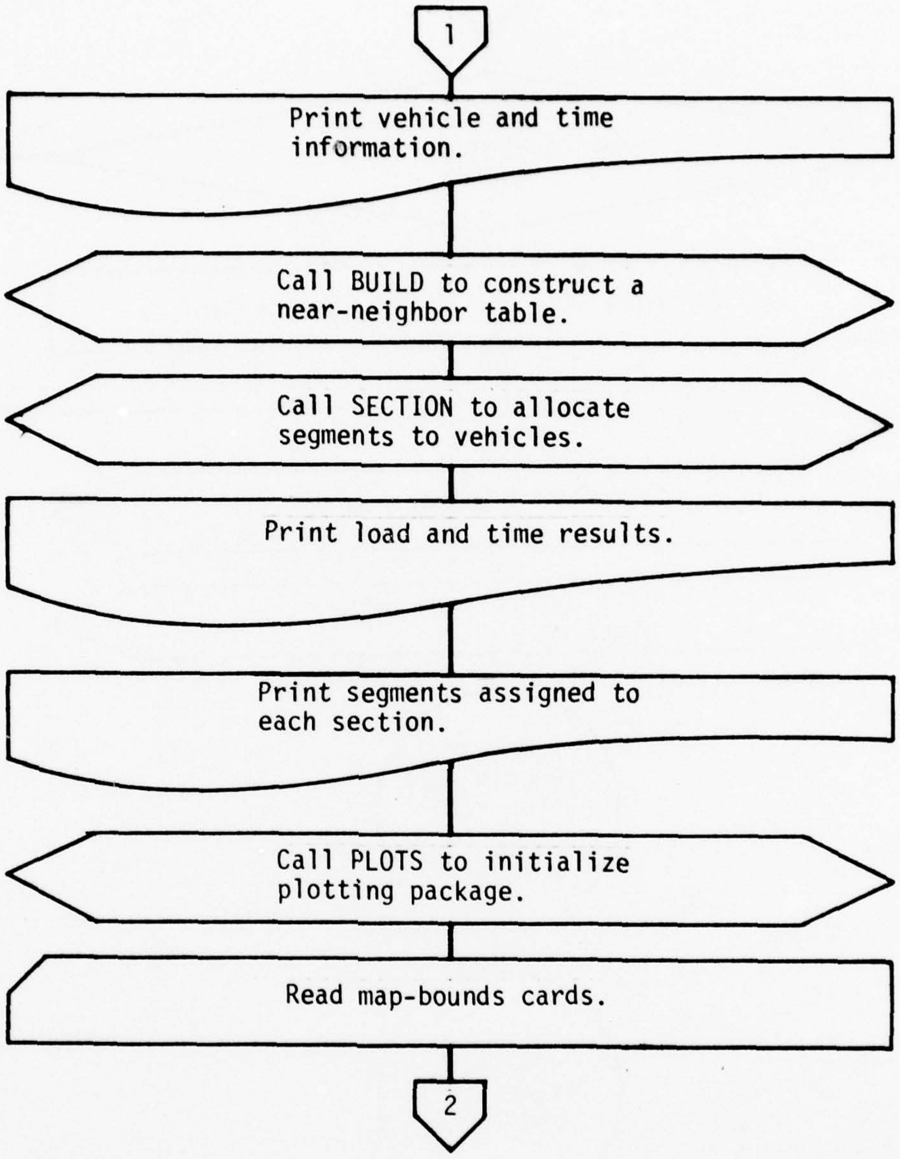
Subroutine SECTION


12
Is the present vehicle load smaller than the minimum necessary for this section?
YES
NO
1021
600
END

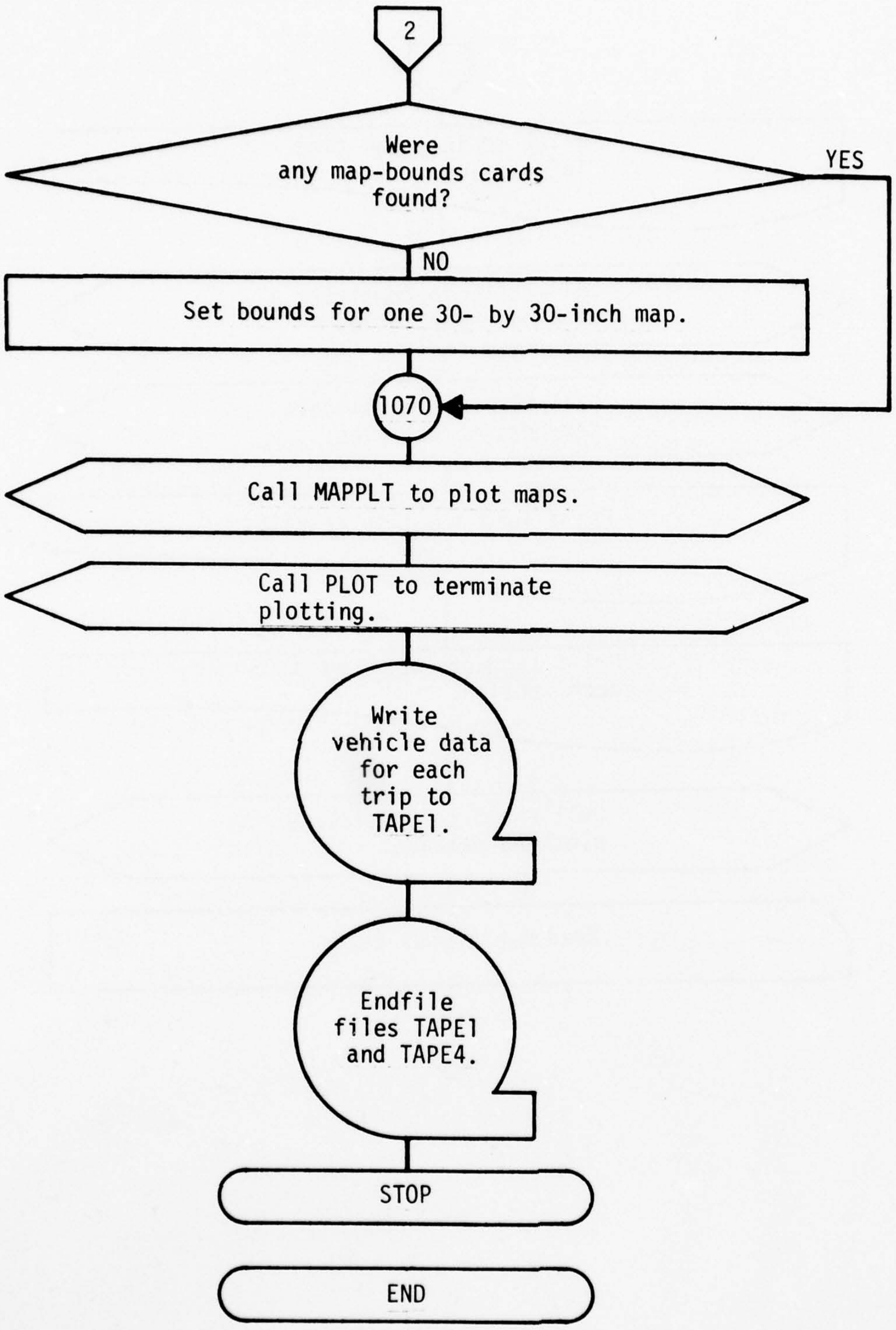
Subroutine SECTION


START
Read title card, vehicle capacities, time limits, and starting segment number.
Read segment data from TAPE9.
Read refuse total and node data from TAPE11.
Make all house counts non-negative.
Validity check starting segment number.
Initialize variables.
Convert all times to minutes.
1

Program PHASE2


1
Print vehicle and time information.
Call BUILD to construct a near-neighbor table.
Call SECTION to allocate segments to vehicles.
Print load and time results.
Print segments assigned to each section.
Call PLOTS to initialize plotting package.
Read map-bounds cards.
2

Program PHASE2


2
Were any map-bounds cards found?
YES
NO
Set bounds for one 30- by 30-inch map.
1070
Call MAPPLT to plot maps.
Call PLOT to terminate plotting.
Write vehicle data for each trip to TAPE1.
Endfile files TAPE1 and TAPE4.
STOP
END

Program PHASE2

APPENDIX B

PROGRAM LISTINGS

```
         IDENT   KOUNT                                                   KOUNT010
         ENTRY   KOUNT                                                   KOUNT020
         VFD     42/0LKOUNT,18/1                                         KOUNT030
KOUNT    DATA    0                                                       KOUNT040
         SA1     A1                                                      KOUNT050
         SA1     X1                                                      KOUNT060
         CX6     X1                                                      KOUNT070
         EQ      KOUNT                                                   KOUNT080
         END                                                             KOUNT090
```

```
      SUBROUTINE SHLSRT(X.A.NW)
      INTEGER A.X
      DIMENSION A(1).X(1)
      N=NW/2
10    K=NW-N
      DO 50 I=1.K
      J=I
      L=I+N
      XT=X(L)
      AT=A(L)
20    IF(XT.GE.X(J)) GO TO 40
      X(L)=X(J)
      A(L)=A(J)
      L=J
      J=J-N
      IF(J.GT.0) GO TO 20
40    X(L)=XT
      A(L)=AT
50    CONTINUE
      IF(N.LE.1) GO TO 60
      N=(N+1)/2
      GO TO 10
60    NW2=NW/2
C      REARRANGE ARRAYS IN DECENDING ORDER
      DO 70 I=1.NW2
      IO=X(I)
      IT=A(I)
      A(I)=A(NW-I+1)
      A(NW-I+1)=IT
      X(I)=X(NW-I+1)
      X(NW-I+1)=IO
70    CONTINUE
      RETURN
      END
      SUBROUTINE SIFTUF(L,N,TREE,NM)                                    SFTUP010
      DIMFNSION TREF(NM)                                                SFTUP020
      I=L                                                               SFTUP030
      COPY=TREE(I)                                                      SFTUP040
   10 J=2*I                                                             SFTUP050
      IF(J-N)1,1,6                                                      SFTUP060
    1 IF(J-N)2,6,6                                                      SFTUP070
    2 IF(TREE(J+1)-TREE(J))3,3,4                                        SFTUP080
    3 J=J+1                                                             SFTUP090
    4 IF(TREE(J)-COPY)5,5,6                                             SFTUP100
    5 TREE(I)=TREE(J)                                                   SFTUP110
      I=J                                                               SFTUP120
      GO TO 10                                                          SFTUP130
    6 TREE(I)=COPY                                                      SFTUP140
      RETURN                                                            SFTUP150
      END                                                               SFTUP160
      SUBROUTINE SORTK(N.KN.TREE.NM)                                    SORTK010
      DIMENSION TREE(NM)                                                SORTK020
      IF (TREE(1) .LE. TREE(N)) GO TO 5                                 SORTK030
      T=TREE(1)          $      TREE(1)=TREE(N)     $      TREE(N)=T    SORTK040
    5 K=N/2                                                             SORTK050
      DO 10 J=2.K                                                       SORTK060
      NN=(N/2)+2-J                                                      SORTK070
 10   CALL SIFTUP(NN.N.TREE.NM)                                         SORTK080
      KNP1=KN+1                                                         SORTK090
      DO 11 J=2.KNP1                                                    SORTK100
      KK=N+2-J                                                          SORTK110
      CALL SIFTUP(1.KK.TREE.NM)                                         SORTK120
      T=TREE(KK)                                                        SORTK130
      TREE(KK)=TREE(1)                                                  SORTK140
      TREE(1)=T                                                         SORTK150
 11   CONTINUE                                                          SORTK160
      RETURN                                                            SORTK170
      END                                                               SORTK180
      FUNCTION IFIND(NUM,IARRAY,LEN)                                     IFIND010
C                                                                        IFIND020
C        LATEST CHANGES --                                               IFIND030
C        APR 2, 1975.  HJI. MODIFIED TO RETURN NEGATIVE OF SUBSCRIPT     IFIND040
C        WHERE NUM SHOULD BE WHEN NUM IS NOT IN IARRAY.                  IFIND050
C        JAN 17, 1975. HJI. ORIGINAL VERSION.                            IFIND060
C                                                                        IFIND070
C        FUNCTION IFIND SEARCHES IARRAY(1)) THROUGH IARRAY(LEN) FOR NUM.IFIND080
C        IF NUM IS NOT FOUND, THE FUNCTION RETURNS WITH IFIND EQUAL      IFIND090
C        TO THE NEGATIVE OF THE SUBSCRIPT WHERE NUM SHOULD BE INSERTED. IFIND100
C        IF NUM=IARRAY(I), THE FUNCTION RETURNS WITH IFIND=I.            IFIND110
C                                                                        IFIND120
      DIMENSION IARRAY(1)                                                IFIND130
C                                                                        IFIND140
      IF (LEN .GT. 0) GO TO 5                                            IFIND150
      IFIND=-1                                                           IFIND160
      RETURN                                                             IFIND170
    5 II=1                                                               IFIND180
      IF=LEN                                                             IFIND190
   10 IP=(II+IF)/2                                                       IFIND200
      IF(NUM-IARRAY(IP)) 20,50,30                                        IFIND210
   20 IF=IP-1                                                            IFIND220
      GO TO 40                                                           IFIND230
   30 II=IP+1                                                            IFIND240
   40 IF (IF .GE. II) GO TO 10                                           IFIND250
      IFIND=-IP                                                          IFIND260
      IF (IARRAY(IP) .LT. NUM) IFIND=-1-IP                               IFIND270
      RETURN                                                             IFIND280
   50 IFIND=IP                                                           IFIND290
      RETURN                                                             IFIND300
      END                                                                IFIND310
      SUBROUTINE   NUMBER(X,Y,HGT,NUM,ANG,FMT)                          NUMBR010
                                                                        NUMBR020
C        LATEST CHANGES   --                                            NUMBR030
C        OCT 24. 1975. HJI.  ORIGINAL VERSION                           NUMBR040
                                                                        NUMBR050
      DIMENSION FORM(3) ,TEXT(3)                                        NUMBR060
      DATA FORM/1H(,0.1H)/                                              NUMBR070
                                                                        NUMBR080
      TEXT(1)=TEXT(2)=TEXT(3)=1H                                        NUMBR090
      FORM(2)=FMT                                                       NUMBR100
      ENCODE (30,FORM,TEXT) NUM                                         NUMBR110
      NC=30                                                             NUMBR120
      DO 10 I=1.3                                                       NUMBR130
      DO 10 J=6.60.6                                                    NUMBR140
      IF ((SHIFT(TEXT(4-I),6-J) .A. 77B) .NE. 55B) GO TO 20             NUMBR150
   10 NC=NC-1                                                           NUMBR160
   20 CALL  SYMBOL(X.Y.HGT,TEXT,ANG,NC)                                 NUMBR170
      RETURN                                                            NUMBR180
      END                                                               NUMBR190
      SUBROUTINE SHAPCOM(TOTLEN,AVMD)
C
C        LATEST CHANGES  --
C        MAY 13, 1976. HJI.  ACAPTEC FOR USE IN SECTIONING PROGRAM.
C        MAR 4, 1976.  HJI.  ADDED VALIDITY TEST FOR LENGTHS CF ANGLE
C        SIDES.
C        OCT 28, 1975. HJI.  REMOVED 2 MORE BUGS IN S CURVE CCDING
C        AND IMPROVED H CALCULATION BY IMPROVING RPR ESTIMATE.
C        OCT 21, 1975. HJI   REMOVED BUG FROM S CURVE CODING
C        SEPT 17, 1975. HJI. IMPROVEC RPR CALCULATICN FOR CIRCLES AND
C        S CURVES.
C
      EQUIVALENCE (ISF,SF)
      COMMON /COPARM/ SF,XNI,XNF,YNI,YNF,SX,SY,RPR,C11,C12,XCTR,YCTR,
     1 BR1,BR2,SGN
      DATA TWOPI/6.283185308/

   20 BR1=BR2=0.
      DX=XNF-XNI                $      DY=YNF-YNI
      SX=DX/TOTLEN              $      SY=DY/TOTLEN
      IF (ISF .EQ. 0  .OR.  ISF .EQ. 778) RETURN

      THETA=ATAN2(DY,DX)
      D=SQRT(DX**2+DY**2)*AVMD
      IF (ISF.NE.2RRC .A. ISF.NE.2RLC .A. ISF.NE.2RRS .A. ISF.NE.2RLS)
     1         GO TO 60

C        PROCESS SHAPE FACTOR FOR CIRCULAR ARC OR S CURVE.
   45 XE=XNF                    $      YE=YNF
      BR1=TOTLEN                $      CD=D
      IF (ISF .EQ. 2RRC .OR.  ISF .EQ. 2RLC) GO TO 50
      XE=0.5*(XNI+XNF)          $      YE=0.5*(YNI+YNF)
      BR1=0.5*BR1               $      CD=0.5*DD
   50 SGN=1.            $      IF (ISF.EQ.2RLC .OR. ISF.EQ.2RLS) SGN=-1.
      V=1.-D/TOTLEN             $      VS=V*V
      ARG=V*(.6332067+VS*(.4303681+VS*(-.2629513+.1998384*VS)))
      IF (ARG .LE. 0.) GO TO 100
      RPR=SGN*TWOPI*SQRT(ARG)/BR1
C        IMPROVE APPROXIMATICN OF RPR
```

AD-A061 821   NEW MEXICO UNIV ALBUQUERQUE ERIC H WANG CIVIL ENGINE--ETC   F/G 13/2
AIR FORCE REFUSE-COLLECTION SCHEDULING PROGRAM DESCRIPTION. VOL--ETC(U)
MAY 78   H J IUZZOLINO, E P DUNPHY   F29601-76-C-0015
UNCLASSIFIED   CERF-EE-20   CEEDO-TR-78-23-VOL-2   NL

2 OF 2
AD
A061821

END
DATE
FILMED
2-79
DDC

```
      EPS1=SIN(.5*BR1*RPR)-.5*DD*RPR
      IF (ABS(EPS1) .LT. 1.E-5) GO TO 51
      DRPR=SIGN(.0002,EPS1)
      EPS2=SIN(.5*BR1*(RPR+DRPR))-.5*DD*(RPR+DRPR)
      RPR=RPR-EPS1*DRPR/(EPS2-EPS1)
   51 CONTINUE
      R=1./RPR                   $     ARG=R*R-0.25*DD*DD
      H=0.                       $     IF (ARG .GT. 0.) H=SQRT(ARG)/AVMD
      IF (BR1 .GT. 3.14159*ABS(R)) H=-H
      XCTR=0.5*(XNI+XE)-SGN*SIN(THETA)*H
      YCTR=0.5*(YNI+YE)+SGN*COS(THETA)*H

C        SET UP ROTATION COEFFICIENTS
      C11=XNI-XCTR               $     C12=YNI-YCTR
      RETURN

   60 IF (ISF .NE. 2RRR .AND. ISF .NE. 2RLR) GO TO 80
      BR1=0.5*(TOTLEN-D)
      IF (BR1 .GT. 0.05*TOTLEN) GO TO 70
      ISF=0                      $     RETURN
   70 BR2=0.5*(TOTLEN+D)
      SX=SIN(THETA)/AVMD         $   SY=COS(THETA)/AVMD
      RETURN

   80 SGN=SIGN(1.,SF)            $     BR1=ABS(SF)
      BR2=TOTLEN-BR1
      F=0.5*(1.-(BR2**2-BR1**2)/D**2)
      ARG=BR1**2-(F*D)**2        $     IF (ARG .LE. 0.) GO TO 100
      H=-SGN*SQRT(ARG)
      XCTR=XNI+(COS(THETA)*F*D-SIN(THETA)*H)/AVMD
      YCTR=YNI+(COS(THETA)*H+SIN(THETA)*F*D)/AVMD
      RETURN

  100 SF=BR1=BR2=0.              $     RETURN
      END
      SUBROUTINE COORD(XX,YY.CUMLEN)
C
C        LATEST CHANGES  --
C        APR 14. 1975. HJI. BUG REMOVED FROM S-CURVE CALCULATION.
C        APR 9. 1975. HJI. ORIGINAL VERSION.

      COMMON /COPARM/ SF.XNI.XNF,YNI,YNF.SX,SY,RPR,C11,C12,XCTR,YCTR,
     1  BR1.BR2.SGN
      INTEGER SF

      S=CUMLEN
      IF (SF .NE. 0) GO TO 10
      XX=XNI+SX*S              $      YY=YNI+SY*S
      RETURN
   10 IF (SF.NE.2RRC .A. SF.NE.2RLC .A. SF.NE.2RRS .A. SF.NE.2RLS)
     1         GO TO 30
      RIP=RPR
      XC=XCTR                  $      YC=YCTR
      C1=C11                   $      C2=C12
      IF (S .LE. .999*BR1 .OR. SF .EQ. 2RRC .OR. SF .EQ. 2RLC) GO TO 20
      RIP=-RPR                 $      S=S-BR1
      XC=XNI+XNF-XCTR          $      YC=YNI+YNF-YCTR
      C1=0.5*(XNI+XNF)-XC      $      C2=0.5*(YNI+YNF)-YC
   20 SN=SIN(S*RIP)            $      CN=COS(S*RIP)
      XX=C1*CN-C2*SN+XC        $      YY=C2*CN+C1*SN+YC
      RETURN
   30 IF (SF .NE. 2RRR .AND. SF .NE. 2RLR) GO TO 60
      SGN=1.                   $      IF (SF .EQ. 2RLR) SGN=-1.
      IF (S .GT. 1.05*BR1) GO TO 40
      IF (S .GT. 0.95*BR1) S=BR1
      XX=XNI+SGN*SX*S          $      YY=YNI-SGN*SY*S
      RETURN
   40 IF (S .GT. 1.05*BR2) GO TO 50
      IF (S .GT. 0.95*BR2) S=BR2
      XX=XNI+SGN*SX*BR1+SY*(S-BR1)      $     YY=YNI-SGN*SY*BR1+SX*(S-BR1)
      RETURN
   50 XX=XNF+SGN*SX*(BR1+BR2-S)         $     YY=YNF-SGN*SY*(BR1+BR2-S)
      RETURN
   60 IF (S .GT. BR1) GO TO 70
```

```
      XX=XNI+(XCTR-XNI)*S/BR1                                          COORD400
      YY=YNI+(YCTR-YNI)*S/BR1                                          COORD410
      RETURN                                                           COORD420
   70 S=S-BR1                                                          COORD430
      XX=XCTR+(XNF-XCTR)*S/BR2     $     YY=YCTR+(YNF-YCTR)*S/BR2     COORD440
      RETURN                                                           COORD450
      END                                                              COORD460
      SUBROUTINE MAPPLT(II,KF)                                          MPPL2010
                                                                        MPPL2020
C         LATEST CHANGES --                                             MPPL2030
C         MAY 13, 1976. HJI.  ADAPTED FOR USE IN SECTIONING PROGRAM.    MPPL2040
C         MAR 4, 1976.  HJI.  CHANGED MAPPLT TO PLOT SEGMENT NUMBERS    MPPL2050
C         INSTEAD OF STREET NUMBERS.                                    MPPL2060
C         NOV 19, 1975. HJI.  ADDED MX TO KEEP PLOTTING WITHIN FRAME.   MPPL2070
C         OCT 29, 1975. HJI.  ADJUSTED NUMBER OF POINTS PER SEGMENT SO  MPPL2080
C         THAT NO LINE CAN EXTEND MORE THAN .5 INCHES PAST THE FRAME.   MPPL2090
C         OCT 6, 1975. HJI.   CORRECTED XL,XR,YB,YT CALCULATIONS.       MPPL2100
C         AUG 13, 1975. HJI.  ORIGINAL VERSION.                         MPPL2110
                                                                        MPPL2120
      INTEGER SF                                                        MPPL2130
      COMMON TITLE(8),ISEG(700),NN1(700),NN2(700),FLEN(700),NH(700),    MPPL2140
     1  FMPH(700),ROF(700),X(700),Y(700),SF(700)                        MPPL2150
      COMMON /NODATA/ KNODES,NBS(500),NODNUM(500),XNOD(500),YNOD(500)   MPPL2160
      COMMON /MPDATA/ XMIN(10),XMAX(10), XLEN(10),YMIN(10),YMAX(10),    MPPL2170
     1  YLEN(10),YHCUT(10),AVMD                                         MPPL2180
      COMMON /COPARM/ISF,XNI,XNF,YNI,YNF,SX,SY,RPR,C11,C12,XCTR,YCTR,   MPPL2190
     1  BR1,BR2,SGN                                                     MPPL2200
      DATA SIZE/.08/                                                    MPPL2210
      XL=XMN=XMIN(II)          $      YB=YMN=YMIN(II)                   MPPL2220
      XR=XMAX(II)              $      YT=YMAX(II)                       MPPL2230
      PHGT=YHCUT(II)           $      XMX=XLEN(II)+1.                   MPPL2240
      MX=YLEN(II)/PHGT+.99     $      PLEN=XMX*MX                       MPPL2250
      XSC=XLEN(II)/(XMAX(II)-XMN)   $     YSC=YLEN(II)/(YMAX(II)-YMN)   MPPL2260
      YCUT=PHGT/YSC            $      LASTNN=0                          MPPL2270
      PRINT 90, II,AVMD,XMN,XMAX(II),YMN,YMAX(II),XL,XR,YB,YT,XSC,      MPPL2280
     1  YSC,PHGT,PLEN,YCUT                                              MPPL2290
   90 FORMAT(*0MAPPLT PARAMETERS FOR MAP*,I2/ * AVMD=*,F10.5/           MPPL2300
     1                              * XMIN=*,F9.5,10X,*XMAX=*,F10.5,10X, MPPL2310
     2  *YMIN=*,F10.5,10X,*YMAX=*,F10.5/ *  XL=*,F10.5,10X,*  XR=*,F10.5MPPL2320
     3,10X,*  YB=*,F10.5,10X,*  YT=*,F10.5/ * XSC=*,F10.5,10X,* YSC=*,  MPPL2330
     4F10.5,10X,*PHGT=*,F10.5,10X,*PLEN=*,F10.5,10X,*YCUT=*,F10.5/)     MPPL2340
      DO 200  K=1,KF                                                    MPPL2350
      NI=NN1(K)                $      NF=NN2(K)                         MPPL2360
      XMD=X(K)                 $      YMD=Y(K)                          MPPL2370
      ISF=SF(K)                                                         MPPL2380
      NS1=IFIND(NI,NODNUM,KNODES)  $     NS2=IFIND(NF,NODNUM,KNODES)    MPPL2390
      XNI=XNOD(NS1)          $     YNI=YNOD(NS1)
      XNF=XNOD(NS2)          $     YNF=YNOD(NS2)
      INBI=INBM=INBF=1
      IF (XNI .LT. XL .OR. XNI .GT. XR .OR. YNI .LT. YB .OR. YNI .GT.
     1  YT) INBI=0
      IF (XMD .LT. XL .OR. XMD .GT. XR .OR. YMD  .LT. YB .OR. YMD .GT.
     1  YT) INBM=0
      IF (XNF .LT. XL .OR. XNF .GT. XR .OR. YNF .LT. YB .OR. YNF .GT.
     1  YT) INBF=0
      IF (INBI .EQ. 0 .AND. INBM .EQ. 0 .AND. INBF .EQ. 0) GO TO 200
      NUMS=ISEG(K)           $     TOTLEN=FLEN(K)
      NPMID=AMAX1(10.,1.+TOTLEN*XSC/AVMD,1.+TOTLEN*YSC/AVMD)
      NPPSEG=2*NPMID
      CALL SHAPCOM(TOTLEN,AVMD)
      CUMLEN=0.              $     DS=TOTLEN/NPPSEG
      XX=XNI                 $     YY=YNI
      NMAP=NMAPO=(YY-YMN-.0001)/YCUT
      IPEN=3-INBI
      IF (IPEN .EQ. 3) GO TO 130
      XP=(XX-XMN)*XSC+NMAP*XMX      $     YP=(YY-YMN-NMAP*YCUT)*YSC
      IF (LASTNN .EQ. NI) GO TO 120
      CALL SYMBOL(XP,YP,SIZE,0,0.,-1)
  120 CALL PLOT(XP,YP,3)
  130 DO 170 I=1,NPPSEG
      CUMLEN=CUMLEN+DS
      CALL COORD(XX,YY,CUMLEN)
  140 XP= (XX-XMN)*XSC+NMAP*XMX
      YP= (YY-YMN-NMAP*YCUT)*YSC
      INB=1
      IF (XX.LT.XL .OR. XX.GT.XR .OR. YY.LT.YB .OR. YY.GT.YT) INB=0
      IF ((IPEN .EQ. 3 .AND. INB .EQ. 0) .OR. NMAP .GE. MX) GO TO 160
      CALL PLOT(XP,YP,IPEN)
      IF (IPEN .EQ. 3) CALL PLOT(XP,YP,2)
      IPEN=3-INB
  150 IF (I .NE. NPMID) GO TO 160
C        APPEND SECTION NUMBERS TO SEGMENT MIDPOINT
      CALL NUMBER(XP-.1,YP+.05,SIZE,NUMS,0.,2HI3)
      CALL PLOT(XP,YP,3)
  160 NMAP= (YY-YMN-.0001)/YCUT
```

```
      IF (NMAP .EQ. NMAPO) GO TO 170                                    MPPL2790
      NMAPO=NMAP           $     IPEN=3          $      GO TO 140       MPPL2800
  170 CONTINUE                                                          MPPL2810
      IF (INB .EQ. 0) GO TO 200                                         MPPL2820
      CALL SYMBOL(XP,YP,SIZE,0,0.,-1)                                   MPPL2830
      CALL PLOT(XP,YP,3)                                                MPPL2840
      LASTNN=NF                                                         MPPL2850
  200 CONTINUE                                                          MPPL2860
  300 CALL PLOT(PLEN+2.,0.,-3)                                          MPPL2870
      RETURN                                                            MPPL2880
      END                                                               MPPL2890
      SUBROUTINE BUILD(N,KN,X,Y,MINFR,TREE,ISTPR,NNT,NNTEMP,XT,YT,KP,
     *  IUNX)
C     THIS SUBROUTINE BUILDS THE NEAREST NEIGHBOR TABLE DURING
C     A CREATION RUN
C      CALCULATE A NEAREST NEIGHBOR TABLE BIT STRING
C
      COMMON /STRINGS/ STRING(40)
      COMMON /BITSTR/ BDATA(60)
      COMMON /DISKIO/ UNOT(6)
      DIMENSION X(N),Y(N),MINFR(3,N),TREE(N)
      DIMENSION ISTPR(N),NNT(N),NNTEMP(N),XT(N),YT(N)
      DIMENSION MINFO(3),COMP(25)
      INTEGER UNOT,COMP,STRING,BDATA
      EQUIVALENCE (STRING(5),COMP(1))
      EQUIVALENCE (STRING(2),MINFO(1))
C
      IUNIT=UNOT(5)
      IF(IUNX.GT.0) IUNIT=IUNX
C
C**** SETUP MASKS
      KP1=KN+1
      DO 5 I=1,60
5     BDATA(I)=SHIFT(1,I-1)
C
C
      DO 1111 II=1,N
      NNT(1)=ISTPR(II)
      DO 1000 I=1,N
      NNTEMP(I)=ISTPR(I)
      XT(I)=X(I)
      YT(I)=Y(I)
1000  CONTINUE
      ISUB=NNTEMP(II)
      XX=XT(II)
      YY=YT(II)
      NNTEMP(II)=NNTEMP(1)
      YT(II)=YT(1)
      XT(II)=XT(1)
      NNTEMP(1)=ISUB
      XT(1)=XX                                                          BUILD400
      YT(1)=YY                                                          BUILD410
      MM=N-1                                                            BUILD420
      DO 20 JJ=2,N                                                      BUILD430
      M=JJ-1                                                            BUILD440
      X1=XT(1)-XT(JJ)                                                   BUILD450
      Y1=YT(1)-YT(JJ)                                                   BUILD460
      D=SQRT(X1*X1+Y1*Y1)                                               BUILD470
C     MASK OUT THE LOW ORDER 12 BITS                                    BUILD480
      TREE(M)=(D.A..NOT.7777B).O.NNTEMP(JJ)                             BUILD490
20    CONTINUE                                                          BUILD500
      CALL SORTK(MM,KN,TREE,N)                                          BUILD510
      KM=KN+1                                                           BUILD520
      DO 30 L=2,KM                                                      BUILD530
      J=MM-L+2                                                          BUILD540
30    NNT(L)=TREE(J).A.7777B                                            BUILD550
C     WRITE(6,12) (NNT(K),K=1,KM),(MINFR(K,II),K=1,MF)                  BUILD560
C                                                                       BUILD570
      DO 94 LL=1,KP                                                     BUILD580
94    COMP(LL)=0                                                        BUILD590
      DO 95 LL=2,KP1                                                    BUILD600
      IW1=(NNT(LL)+59)/60                                               BUILD610
      IP1=MOD(NNT(LL)-1,60)+1                                           BUILD620
      COMP(IW1)=COMP(IW1).OR.BDATA(IP1)                                 BUILD630
95    CONTINUE                                                          BUILD640
      STRING(1)=NNT(1)                                                  BUILD650
      MINFO(1)=MINFR(1,II)                                              BUILD660
      MINFO(2)=MINFR(2,II)                                              BUILD670
      MINFO(3)=MINFR(3,II)                                              BUILD680
      WRITE(IUNIT) STRING                                               BUILD690
1111  CONTINUE                                                          BUILD700
      REWIND IUNIT                                                      BUILD710
      RETURN                                                            BUILD720
      END                                                               BUILD730
      SUBROUTINE SECTION(NN, K, MODE, IFLAG, KCUTOF, X, Y, NNTS, IST0, IST1, IST2,
     * IST4, KP, KPB, MA)
      COMMON /STRINGS/ STRING(40)
      COMMON /BITSTR/ BDATA(60)
      COMMON /DISKIO/ UNOT(6)
      COMMON /ROUTE/ TRUCK(7,50),TRUCKS(3,4),NTRUCK,NBASE
      COMMON /STATS/ FRACT,TOTL,TOTT,CUML,CUMT, SECTN,TDUMP
      DIMENSION X(NN),Y(NN),NNTS(KPB,MA)
      DIMENSION IST0(MA),IST1(MA),IST2(MA),IST4(MA)
      DIMENSION HISTO(60),BCOMP(25),COMP(25),BMINF(3)
      DIMENSION BASE(40),MINFO(3)
      INTEGER BASE,SWITCH,SECTN,PASS,BDATA,OLDUNT,TPR
      INTEGER UNOT,SKIP,PC,STRING
      INTEGER COMP,BCOMP,BMINF,HISTO
      REAL INF1,INF2,INF3,MINFO
      EQUIVALENCE (BASE(5),BCOMP(1))
      EQUIVALENCE (BASE(2),BMINF(1))

      EQUIVALENCE (STRING(2),MINFO(1))
      EQUIVALENCE (STRING(5),COMP(1))
      EXTERNAL KOUNT
C
C     INITIALIZE CODE PARAMETERS
      CUMT=CUML=0
      NJO=0
      NSTO=30
      LPASS=KPASS=0
      NP=NN
      SKIP=0
      SMLD=0.
      NTRUCK=0
      NEXTN=2
      IFLAG=0
      N=NN
      CUML=CUMT=PC=PK=ICODE=0
      PASS=LINE=SECTN=NEXT=OLDUNT=TPR=1
      CLDUNT=UNOT(1)
      SWITCH=2
C
```

```
C       ALLOCATE DISK FILES                                                                  SECT0400
        IUNIT=UNOT(2)                                                                        SECT0410
        IO3=UNOT(3)                                                                          SECT0420
        IO4=UNOT(4)                                                                          SECT0430
        IO5=UNOT(5)                                                                          SECT0440
        IO10=10                                                                              SECT0450
C       EXPAND TRUCK DATA STRUCTURE                                                          SECT0460
        DO 6 I=1,50                                                                          SECT0470
        DO 5 J=1,7                                                                           SECT0480
      5 TRUCK(J,I)=0                                                                         SECT0490
      6 TRUCK(4,I)=TDUMP                                                                     SECT0500
        TRUCK(5,1)=1                                                                         SECT0510
        DO 15 I=1,4                                                                          SECT0520
        ITR=TRUCKS(1,I)          $      IF (ITR .EQ. 0) GO TO 15                             SECT0530
           DO 10 J=1,ITR                                                                     SECT0540
           NTRUCK=NTRUCK+1       $      TRUCK(1,NTRUCK)=TRUCKS(2,I)                          SECT0550
     10    TRUCK(2,NTRUCK)=TRUCKS(3,I)                                                       SECT0560
15      CONTINUE                                                                             SECT0570
        NTO=NTRUCK                                                                           SECT0580
        IF(MODE.NE.0) GO TO 100                                                              SECT0590
C       INITIALIZE OLDUNT                                                                    SECT0600
        DO 17 IZ=1,NN                                                                        SECT0610
        READ(IO5) STRING                                                                     SECT0620
        IF(STRING(1).NE.NBASE) GO TO 171                                                     SECT0630
        DO 172 IJ=1,KPE                                                                      SECT0640
172     BASE(IJ)=STRING(IJ)                                                                  SECT0650
        INF1=MINFO(1)                                                                        SECT0660
        INF2=MINFO(2)                                                                        SECT0670
        INF3=MINFO(3)                                                                        SECT0680
        GO TO 17                                                                             SECT0690
171     WRITE(OLDUNT) STRING                                                                 SECT0700
17      CONTINUE                                                                             SECT0710
        REWIND OLDUNT                                                                        SECT0720
        REWIND IO5                                                                           SECT0730
        GO TO 1301                                                                           SECT0740
C                                                                                            SECT0750
C                                                                                            SECT0760
100     XR=X(NBASE)                                                                          SECT0770
        YR=Y(NBASE)                                                                          SECT0780
      ODIS=1000000                                                      SECT0790
C     FIND THE SEGMENT CLOSEST TO THE LAST BASE                         SECT0800
      DO 1101 I=1,N                                                     SECT0810
      READ(OLDUNT) STRING                                               SECT0820
      WRITE(IUNIT) STRING                                               SECT0830
      NS=STRING(1)                                                      SECT0840
      XS=X(NS)                                                          SECT0850
      YS=Y(NS)                                                          SECT0860
      XSD=XS-XR                                                         SECT0870
      YSD=YS-YR                                                         SECT0880
      DIS=SQRT(XSD*XSD+YSD*YSD)                                         SECT0890
      IF(DIS.GE.ODIS) GO TO 1101                                        SECT0900
      ODIS=DIS                                                          SECT0910
      NBASE=NS                                                          SECT0920
      DO 1201 IJ=1,KPB                                                  SECT0930
1201  BASE(IJ)=STRING(IJ)                                               SECT0940
1101  CONTINUE                                                          SECT0950
      REWIND OLDUNT                                                     SECT0960
      REWIND IUNIT                                                      SECT0970
C     REMOVE BASE POINT FROM OLDUNT                                     SECT0980
      DO 1401 IK=1,N                                                    SECT0990
      READ(IUNIT) STRING                                                SECT1000
      IF(STRING(1) .NE. BASE(1)) GO TO 1400                             SECT1010
      INF1=MINFO(1)                                                     SECT1020
      INF2=MINFO(2)                                                     SECT1030
      INF3=MINFO(3)                                                     SECT1040
      GO TO 1401                                                        SECT1050
 1400 WRITE (OLDUNT) STRING                                             SECT1060
1401  CONTINUE                                                          SECT1070
      REWIND OLDUNT                                                     SECT1080
      REWIND IUNIT                                                      SECT1090
1301  CONTINUE                                                          SECT1100
      SMLD=SMLD+FRACT*TRUCK(1, SECTN)                                   SECT1110
      DO 90 I=1,KPB                                                     SECT1120
90    NNTS(I,1)=BASE(I)                                                 SECT1130
C                                                                       SECT1140
C     ADD BASE TO TRUCK FOR CURRENT SECTION                             SECT1150
      WRITE(IO3) BASE                                                   SECT1160
      NSTD=1                                                            SECT1170
      TRUCK(3,SECTN)=INF1                                                 SECT1180
      TRUCK(4,SECTN)=INF2                                                 SECT1190
      TRUCK(6,SECTN)=1                                                    SECT1200
      TRUCK(7,SECTN)=INF3                                                 SECT1210
      PC=1                                                                SECT1220
1021  KL=NP-PC                                                            SECT1230
440   JON=0                                                               SECT1240
      ION=0                                                               SECT1250
      DO 1020 I=1,30                                                      SECT1260
      IST2(I)=I                                                           SECT1270
      IST0(I)=IST1(I)=IST4(I)=0                                           SECT1280
1020  CONTINUE                                                            SECT1290
      DO 1019 I=1,K                                                       SECT1300
1019  HISTO(I)=0                                                          SECT1310
C     BUILD CONNECTIVE DENSITY FUNCTION                                   SECT1320
      L1=BASE(1)                                                          SECT1330
      DO 2929 K2=1,KL                                                     SECT1340
      READ(OLDUNT) STRING                                                 SECT1350
      L2=STRING(1)                                                        SECT1360
      IW1=(L1+59)/60                                                      SECT1370
      IP1=MOD(L1-1,60)+1                                                  SECT1380
      IF((BDATA(IP1).AND.COMP(IW1)).NE.0) GO TO 1022                      SECT1390
      GO TO 3030                                                          SECT1400
1022  IW1=(L2+59)/60                                                      SECT1410
      IP1=MOD(L2-1,60)+1                                                  SECT1420
      IF((BDATA(IP1).AND.BCOMP(IW1)).NE.0) GO TO 1023                     SECT1430
      GO TO 3030                                                          SECT1440
1023  IC=0                                                                SECT1450
      DO 1024 L=1,KP                                                      SECT1460
1024  IC=IC+KOUNT(COMP(L).AND.BCOMP(L))                                   SECT1470
      HISTO(IC)=HISTO(IC)+1                                               SECT1480
      IF(IC.LE.0.OR.JON.GE.30) GO TO 3030                                 SECT1490
      JON=JON+1                                                           SECT1500
      IST0(JON)=L2                                                        SECT1510
      IST1(JON)=IC                                                        SECT1520
      IST4(JON)=IC                                                        SECT1530
      WRITE(IO10) STRING                                                  SECT1540
      GO TO 2929                                                          SECT1550
3030  WRITE(IUNIT)STRING                                                  SECT1560
```

```
2929  CONTINUE                                                  SECT1570
      NUT=KL-JON                                                SECT1580
      REWIND IUNIT                                              SECT1590
      KSUM=0                                                    SECT1600
      DO 4040 K3=1,K                                            SECT1610
      KDOWN=K-K3+1                                              SECT1620
      KSUM=KSUM+HISTO(KDOWN)                                    SECT1630
      KI=KDOWN                                                  SECT1640
      IF(KSUM.GT.KCUTOF) GO TO 4450                             SECT1650
4040  CONTINUE                                                  SECT1660
4450  CONTINUE                                                  SECT1670
      REWIND OLDUNT                                             SECT1680
      IF(JON.EQ.0) GO TO 4990                                   SECT1690
      CALL SHLSRT(IST1,IST2,JON)                                SECT1700
      REWIND I010                                               SECT1710
C                                                               SECT1720
      ION=1                                                     SECT1730
1001  IF(ICODE.EQ.1) GO TO 700                                  SECT1740
101   IP=IST2(ION)                                              SECT1750
      REWIND I010                                               SECT1760
      LNX=IST0(IP)                                              SECT1770
      LNY=IST4(IP)                                              SECT1780
      IF(ION.GT.JON) GO TO 4991                                 SECT1790
      DO 102 J=1,JON                                            SECT1800
      READ(I010) STRING                                         SECT1810
      IF(STRING(1).EQ.LNX.AND.LNY.GE.KI) GO TO 500              SECT1820
102   CONTINUE                                                  SECT1830
      IF (TRUCK(3,SECTN)+CUML  .GT. SMLD) GO TO 600             SECT1840
      REWIND I010                                               SECT1850
      REWIND OLDUNT                                             SECT1860
4991  LPASS=LPASS+1                                             SECT1870
      DO 3436 I=1,JON                                           SECT1880
      READ(I010) STRING                                         SECT1890
      IF(IST1(I).GT.0) WRITE(OLDUNT) STRING                     SECT1900
3436   CONTINUE                                                 SECT1910
3435  CONTINUE                                                  SECT1920
      DO 3437 I=1,NUT                                           SECT1930
      READ(IUNIT) STRING                                        SECT1940
      WRITE(OLDUNT) STRING                                      SECT1950
```

```
3437  CONTINUE
      REWIND IUNIT
      REWINC OLDUNT
      REWIND I010
4990  IF(NEXTN.GT.NSTO) GO TC 800
C
499   DO 475 I=1,KPB
475   BASE(I)=NNTS(I,NEXTN)
C
      NEXTN=NEXTN+1
      GO TO 1021
C
C
C     INTERSECTION TEST IS PASSED
500   CURL=MINFO(1)
      CURT=MINFO(2)
C       CHECK IF TRUCK HAS TIME AND VOLUME
      ION=ION+1
      IF (TRUCK(3,SECTN)+CURL .LE. TRUCK(1,SECTN) .ANC.
     1    TRUCK(4,SECTN)+CLRT .LE. TRUCK(2,SECTN)) GO TO 550
C        DO NOT GO BEYOND CURRENT BASE POINT TC FILL OUT TRUCK
      IF (ION .GT. JON) GO TO 600
      REWIND I010
      IP=IST2(ION)       $     LNX=IST0(IP)        $     LNY=IST4(IP)
      DO 510 J=1,JON
      READ (I010) STRING
      IF (STRING(1) .EQ. LNX .AND. LNY .GE. KI) GC TO 500
  510 CONTINUE
      GO TO 600

  550 IST1(IP)=-IST1(IP)
C
C     ADD THIS ARC SEGMENT TO CURRENT SECTION
      TRUCK(3,SECTN)=TRUCK(3,SECTN)+CURL
      TRUCK(4,SECTN)=TRUCK(4,SECTN)+CURT
      TRUCK(7,SECTN)=TRUCK(7,SECTN)+MINFO(3)
C
      IF(NSTO.GE.NSTO) GO TO 598
      NSTD=NSTD+1
      DO 599 J=1,KPB
599   NNTS(J,NSTD)=STRING(J)
C
C     OUTPUT TO SCRATCH DISK FILE
598   CONTINUE
      WRITE(IO3) STRING
      PC=PC+1
      TRUCK(6,SECTN)=TRUCK(6,SECTN)+1
      GO TO 1001
C
600   ICODF=1
C     COPY THE REST FROM IO10 AND IUNIT TO OLDUNT
      REWIND IO10
      DO 698 I=1,JON
      READ(IO10) STRING
      IF(IST1(I).GT.0) WRITE(OLDUNT) STRING
698   CONTINUE
      REWIND IO10
      DO 699 I=1,NUT
      READ(IUNIT) STRING
699   WRITE(OLDUNT) STRING
      REWIND IUNIT
      REWIND OLDUNT
C
C     SECTION COMPLETED
700   REWIND IO3
      N=N-PC
      NJO=0
      NP=N
      LC=PC
C       TRANSFER SCRATCH TO OUTPUT FILE
      DO 705 L=1,LC
      READ(IO3) STRING
      CUML=CUML+MINFO(1)
      CUMT=CUMT+MINFO(2)
      WRITE(IO4,51) STRING(1)
705   CONTINUE
      TESTL=TOTL-CUML
      TESTT=TOTT-CUMT
      IST=SECTN+1                                                       SECT2740
      IF(IST.GT.NTRUCK) GO TO 801                                       SECT2750
C      MAKE TESTS ON REMAINDER OF ARCS                                  SECT2760
818   IF(TESTL.GT.TRUCK(1,IST)) GO TO 710                               SECT2770
      IF(TESTT.GT.TRUCK(2,IST)) GO TO 710                               SECT2780
      PK=PK+PC                                                          SECT2790
      TRUCK(5,SECTN+1)=PK+1                                             SECT2800
      REWIND IUNIT                                                      SECT2810
      LN=N                                                              SECT2820
C                                                                       SECT2830
C     COPY REMAINDER OF NEW SECTION                                     SECT2840
701   DO 706 J=1,LN                                                     SECT2850
      READ(OLDUNT) STRING                                               SECT2860
      TRUCK(3,IST)=TRUCK(3,IST)+MINFO(1)                                SECT2870
      TRUCK(4,IST)=TRUCK(4,IST)+MINFO(2)                                SECT2880
      TRUCK(7,IST)=TRUCK(7,IST)+MINFO(3)                                SECT2890
      WRITE(IO4,51) STRING(1)                                           SECT2900
706   CONTINUE                                                          SECT2910
      SECTN=SECTN+1                                                     SECT2920
      TRUCK(5,IST)=NN-N+1                                               SECT2930
      TRUCK(6,IST)=N                                                    SECT2940
      REWIND IO4                                                        SECT2950
      RETURN                                                            SECT2960
C                                                                       SECT2970
C      DEFINE MORE TRUCKS THROUGH RAP AROUND OF EXP TRUCK DATA STRUCTURESECT2980
801   NTRUCK=NTRUCK+1                                                   SECT2990
      PRINT803                                                          SECT3000
      IF(TPR.GT.NTO) TPR=1                                              SECT3010
      TRUCK(1,NTRUCK)=TRUCK(1,TPR)                                      SECT3020
      TRUCK(2,NTRUCK)=TRUCK(2,TPR)                                      SECT3030
      DO 811 J=3,6                                                      SECT3040
811   TRUCK(J,NTRUCK)=0                                                 SECT3050
      TPR=TPR+1                                                         SECT3060
      IST=NTRUCK                                                        SECT3070
      GO TO 818                                                         SECT3080
C                                                                       SECT3090
C     FINISH UP AND PREPARE FOR NEW ITERATION                           SECT3100
710   REWIND OLDUNT                                                     SECT3110
      REWIND IUNIT                                                      SECT3120
      REWIND IO3
      PK=PK+PC
      SECTN=SECTN+1
      TRUCK(5,SECTN)=PK+1
      SKIP=0
      LINE=1
      ICODE=0
      PC=0
      NEXTN=2
      KPASS=KPASS+1
      GO TO 100
800   XR=X(NBASE)
      YR=Y(NBASE)
      ODIS=1000000
C     FIND THE SEGMENT CLOSEST TO THE LAST BASE
      NLK=N-PC
      NBASE=0
      DO 2101 I=1,NLK
      READ(OLDUNT) STRING
      WRITE(IUNIT) STRING
      NS=STRING(1)
      XS=X(NS)
      YS=Y(NS)
      XSD=XS-XR
      YSD=YS-YR
      DIS=SQRT(XSD*XSD+YSD*YSD)
      IF(DIS.GE.ODIS) GO TO 2101
      IF(TRUCK(3,SECTN)+MINFC(1).GT.TRUCK(1,SECTN))GO TO 2101
      IF(TRUCK(4,SECTN)+MINFC(2).GT.TRUCK(2,SECTN))GO TO 2101
      ODIS=DIS
      NBASE=NS
      DO 2201 IJ=1,KPB
2201  BASE(IJ)=STRING(IJ)
2101  CONTINUE
      REWIND OLDUNT
      REWIND IUNIT
      IF(NBASE.EQ.0)GO TO 600
C     REMOVE BASE POINT FROM OLDUNT
      DO 2401 IK=1,NLK
      READ(IUNIT) STRING
      IF(STRING(1) .NE. BASE(1)) GO TO 2400
      INF1=MINFO(1)
      INF2=MINFO(2)
      INF3=MINFO(3)
      GO TO 2401
 2400 WRITE (OLDUNT) STRING
2401  CONTINUE
      REWIND OLDUNT
      REWIND IUNIT
C
C
C     ADD BASE TO TRUCK FOR CURRENT SECTION
      WRITE(IO3) BASE
      PC=PC+1
      TRUCK(3,SECTN)=TRUCK(3,SECTN)+INF1
      TRUCK(4,SECTN)=TRUCK(4,SECTN)+INF2
      TRUCK(6,SECTN)=TRUCK(6,SECTN)+1.
      TRUCK(7,SECTN)=TRUCK(7,SECTN)+INF3
      NSTD=1                    $       NEXTN=2
      IF (TRUCK(3,SECTN)+CUML .LT. SMLD) 1021,600
C
C
C
51    FORMAT(1X,I5)
803   FORMAT(5X,*TRUCK CONFIGURATION HAS BEEN EXTENDED*)
      END
      PROGRAM PHASE2(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT,TAPE1,TAPE2,
     1  TAPE3,TAPE4,TAPE7,TAPE8,TAPE9,TAPE10,TAPE11)
C     DRIVER FOR SECTION SIMULATION
      COMMON TITLE(8),ISEG(700),NN1(700),NN2(700),FLEN(700),NH(700),
     1  FMPH(700),ROF(700),X(700),Y(700),SF(700)
      COMMON /MPDATA/ XMIN(10),XMAX(10),XLEN(10),YMIN(10),YMAX(10),
     1  YLEN(10),YHCUT(10),AVMD
      COMMON /NODATA/ KNODES,NBS(500),NODNUM(500),XNOD(500),YNOD(500)
      COMMON /DISKIO/ UNOT(6)
      COMMON /ROUTE/ TRUCK(7,50),TRUCKS(3,4),NTRUCK,NBASE
      COMMON /STATS/ FRACT,TOTL,TOTT,CUML,CUMT,ISECTN,TDUMP
      DIMENSION XT(700),YT(700),TREE(700),NNT(700)
      DIMENSION ISTPR(700),NNTEMP(700),MINFR(3,700)
      DIMENSION IST0(30),IST1(30),IST2(30),IST4(30)
      DIMENSION NT(4),TC(4)
      INTEGER UNOT
      REAL MINFR
      DATA UNOT/1,2,3,4,7,8/


      READ 10, TITLE
   10 FORMAT(8A10)
      READ 30, (NT(I),TC(I),I=1,4),TSTOPH,TSTOPR,TDUMP,TMXTR,NBASE
   30 FORMAT(4(I10,F10.0)/4F10.0,I10)
      READ (9) NSEG,(DUMMY,NN1(I),NN2(I),FLEN(I),NH(I),FMPH(I),DUMMY,
     1  ROF(I),X(I),Y(I),SF(I),I=1,NSEG),AVMD
      READ(11) NHTOT,TOTREF,KNODES,(NODNUM(I),NBS(I),XNOD(I),YNOD(I),
     1  I=1,KNODES)
      DO 15 I=1,NSEG
   15 NH(I)=IABS(NH(I))
      NA=NSEG
      IF (NBASE .LT. 1 .OR. NBASE .GT. NSEG) NBASE=1
      KN=60
      MODE=0
      OPTION=0
      NSEED=0
      TOTT=TOTL=0
      DO 20 J=1,NSEG
      MINFR(1,J)=NH(J)*ROF(J)      $      MINFR(2,J)=60.*FLEN(J)/FMPH(J)
      MINFR(3,J)=NH(J)                                                  PHS20400
      IF(NH(J).NE.0)MINFR(2,J)=12.*FLEN(J)+MINFR(1,J)*TSTOPR+NH(J)*     PHS20410
     1TSTOPH                                                            PHS20420
      TOTT=TOTT+MINFR(2,J)                                              PHS20430
      TOTL=TOTL+NH(J)*ROF(J)                                            PHS20440
20    CONTINUE                                                          PHS20450
      IF (TMXTR .LE. 0.) TMXTR=24.                                      PHS20460
      TMXTR=60.*TMXTR                                                   PHS20470
C         NOW ALL TIMES ARE IN MINUTES                                  PHS20480
      DO 25 I=1,4                                                       PHS20490
   25 IF (NT(I) .EQ. 0  .AND.  TC(I) .GT. 0.) NT(I)=1                   PHS20500
      REF=0.                                                            PHS20510
      DO 50 I=1,4                                                       PHS20520
      TRUCKS(1,I)=NT(I)                                                 PHS20530
      TRUCKS(2,I)=TC(I)                                                 PHS20540
      TRUCKS(3,I)=TMXTR                                                 PHS20550
   50 REF=REF+NT(I)*TC(I)                                               PHS20560
      NTEA=TOTL/REF                                                     PHS20570
      IF (NTEA .EQ. 0) GO TO 80                                         PHS20580
      REF=0.                                                            PHS20590
      DO 60 I=1,4                                                       PHS20600
      TRUCKS(1,I)=NTEA*NT(I)                                            PHS20610
   60 REF=REF+TRUCKS(1,I)*TRUCKS(2,I)                                   PHS20620
      DO 70 I=1,4                                                       PHS20630
      NINC=0                                                            PHS20640
      IF (TC(I) .GT. 0.) NINC=MIN0(NT(I),INT((TOTL-REF)/TC(I)+.999))    PHS20650
      REF=REF+NINC*TC(I)                                                PHS20660
      TRUCKS(1,I)=TRUCKS(1,I)+NINC                                      PHS20670
      IF (REF .GT. TOTL) GOTO 80                                        PHS20680
   70 CONTINUE                                                          PHS20690
   80 FRACT=TOTL/REF                                                    PHS20700
      PRINT 90,TITLE,(TC(I),TRUCKS(1,I),I=1,4),FRACT,TSTOPH,TSTOPR,     PHS20710
     1 TDUMP,TMXTR                                                      PHS20720
90    FORMAT(*1INPUT VEHICLE DATA*,10X,8A10/*0CAPACITY    TRIPS*/4(2F9.0PHS20730
     1/)/* MINIMUM FILL FRACTION=*,F6.3/*0STOP TIME PER HOUSEHOLD  =*,F1PHS20740
     20.2,* MINUTES*/* STOP TIME PER UNIT REFUSE=*,F10.2,* MINUTES*/* UNPHS20750
     3LOADING TIME*,11X,*=*,F10.2,* MINUTES*/* MAXIMUM TRIP TIME        PHS20760
     4=*,F10.2,* MINUTES*/)                                             PHS20770
      PRINT 91, NBASE                                                   PHS20780
   91 FORMAT(* THE FIRST SECTION WILL START WITH SEGMENT*,I5/)              PHS20790
      IUNX=0                                                                PHS20800
      DO 96 IKB=1,NSEG                                                      PHS20810
   96 ISTPR(IKB)=IKB                                                        PHS20820
C                                                                           PHS20830
C     NA----------------NUMBER OF STREET SEGMENTS                           PHS20840
C     KN----------------NUMBER OF NEAREST NEIGHBORS TO BE COMPUTED          PHS20850
C     STRING------------ARRAY OF ONE NEAREST NEIGHBOR TABLE (KPB WORDS)     PHS20860
C     MINFR-------------TEMPORARY STORAGE (3 WORDS)                         PHS20870
C     NNT,NNTEMP,TREE---TEMPORARY STORAGE (NA WORDS)                        PHS20880
C     X,XT--------------X COORDINATES OF STREET SEGMENT (NA WORDS)          PHS20890
C     Y,YT--------------Y COORDINATES OF STREET SEGMENT (NA WORDS)          PHS20900
C     COMP--------------TEMPORARY STORAGE (KPB WORDS)                       PHS20910
C                                                                           PHS20920
      MA = 30                                                               PHS20930
      KP = (NA+59)/60                                                       PHS20940
      KPB = KP+4                                                            PHS20950
      CALL BUILD(NA,KN,X,Y,MINFR,TREE,ISTPR,NNT,NNTEMP,XT,YT,KP,IUNX)       PHS20960
      IFLAG=0                                                               PHS20970
      KCUTOF=5                                                              PHS20980
C                                                                           PHS20990
C     IST0,IST1,IST2----TEMPORARY STORAGE (MA WORDS)                        PHS21000
C     NNT---------------TEMPORARY STORAGE (KPB*MA WORDS)                    PHS21010
C     BASE--------------TEMPORARY STORAGE (KPB WORDS)                       PHS21020
C                                                                           PHS21030
      CALL SECTION(NA,KN,MODE,IFLAG,KCUTOF,X,Y,NNT,IST0,IST1,IST2,IST4,     PHS21040
     1  KP,KPB,MA)                                                          PHS21050
      IF(IFLAG.EQ.1)GO TO 333                                               PHS21060
      PRINT 99                                                              PHS21070
   99 FORMAT(*1 TRIP  CAPACITY  TIME LIMIT*,4X,*LOAD*,6X,*TIME  SEGMENTSPHS21080
     1  TOTAL NH*/)                                                         PHS21090
      DO 40 I=1,ISECTN                                                      PHS21100
      WRITE(6,41)I,(TRUCK(J,I),J=1,4),(TRUCK(J,I),J=6,7)                    PHS21110
   41 FORMAT(I6,3F10.2,3F10.0)                                              PHS21120
   40 CONTINUE                                                              PHS21130
      IF(OPTION.EQ.1) NA=NA+NSEED                                           PHS21140
      IUNIT=UNOT(4)                                                         PHS21150
      J=1                                                                   PHS21160
      DO 33 I=1,NA                                                          PHS21170
      READ(IUNIT,74) KL                                                 PHS21180
 74   FORMAT(1X,I5)                                                     PHS21190
      IF(I.NE.TRUCK(5,J))GO TO 88                                       PHS21200
      PRINT 87,J                                                        PHS21210
 87   FORMAT(*0SECTION * ,I3, * CONTAINS SEGMENTS*/)                    PHS21220
      J=J+1        $CC=1H     $NN=0                                     PHS21230
 88   NN=NN+1                                                           PHS21240
      PRINT 86,CC,(0,K=1,NN),KL                                         PHS21250
 86   FORMAT(A1,31I4.0)                                                 PHS21260
      CC=1H+                                                            PHS21270
      IF(NN.LT.30)GO TO 33                                              PHS21280
      CC=1H        $NN=0                                                PHS21290
 33   CONTINUE                                                          PHS21300
C    PLOT RESULTS OF SECTIONING ALGORITHM                               PHS21310
      IUNIT=UNOT(4)                                                     PHS21320
      REWIND IUNIT                                                      PHS21330
      CALL PLOTS(0,0,8)  $   CALL PLOT(0.,-3.,3)  $   CALL PLOT(0.,0.,3)PHS21340
      DO 1000 I=1,ISECTN                                                PHS21350
      IL=TRUCK(6,I)                                                     PHS21360
      DO 2000 J=1,IL                                                    PHS21370
      READ(IUNIT,74) IPT                                                PHS21380
      ISEG(IPT)=I                                                       PHS21390
2000  CONTINUE                                                          PHS21400
1000  CONTINUE                                                          PHS21410
                                                                        PHS21420
      MAPS=0                                                            PHS21430
      DO 1030 I=1,10                                                    PHS21440
      READ 1010,XMIN(I),XMAX(I),XLEN(I),YMIN(I),YMAX(I),YLEN(I),YHCUT(I)PHS21450
 1010 FORMAT(7F10.0)                                                    PHS21460
      IF (EOF(5)) 1040,1020                                             PHS21470
 1020 IF (YHCUT(I) .LE. 0.) YHCUT(I)=30.                                PHS21480
 1030 MAPS=I                                                            PHS21490
 1040 IF (MAPS .GT. 0) GO TO 1070                                       PHS21500
                                                                        PHS21510
C        SET UP DEFAULTS --  ONE 30X30 INCH MAP.                        PHS21520
      MAPS=1                                                            PHS21530
      XLEN(1)=YLEN(1)=30.    $      YHCUT(1)=30.                        PHS21540
      XMIN(1)=YMIN(1)=1.E20  $      XMAX(1)=YMAX(1)=-1.E20              PHS21550
C        SCAN NODES AND SEGMENT MIDPOINTS FOR COORDINATE BOUNDS.        PHS21560
```

```
      DO 1050 I=1,NSEG                                                  PHS21570
      IF (X(I) .LT. XMIN(1)) XMIN(1)=X(I)                               PHS21580
      IF (X(I) .GT. XMAX(1)) XMAX(1)=X(I)                               PHS21590
      IF (Y(I) .LT. YMIN(1)) YMIN(1)=Y(I)                               PHS21600
 1050 IF (Y(I) .GT. YMAX(1)) YMAX(1)=Y(I)                               PHS21610
      DO 1060 I=1,KNODES                                                PHS21620
      IF (XNOD(I) .LT. XMIN(1)) XMIN(1)=XNOD(I)                         PHS21630
      IF (XNOD(I) .GT. XMAX(1)) XMAX(1)=XNOD(I)                         PHS21640
      IF (YNOD(I) .LT. YMIN(1)) YMIN(1)=YNOD(I)                         PHS21650
 1060 IF (YNOD(I) .GT. YMAX(1)) YMAX(1)=YNOD(I)                         PHS21660
 1070 DO 1080 I=1,MAPS                                                  PHS21670
                                                                        PHS21680
 1080 CALL MAPPLT(I,NSEG)                                               PHS21690
                                                                        PHS21700
      CALL PLOT(0.,0.,-3)                                               PHS21710
      CALL PLOT(0.,0.,999)                                              PHS21720
C                                                                       PHS21730
333   CONTINUE                                                          PHS21740
      REWIND 1                                                          PHS21750
      WRITE (1) NA,ISECTN,(TRUCK(5,I),TRUCK(6,I)+TRUCK(5,I)-1.,         PHS21760
     1  TRUCK(1,I),I=1,ISECTN)                                          PHS21770
      ENDFILE 1                                                         PHS21780
      ENDFILE 4                                                         PHS21790
      STOP                                                              PHS21800
      END                                                               PHS21810
```

(The reverse of this page is blank.)

APPENDIX C

DEFINITIONS OF IMPORTANT VARIABLES

Note: A single variable symbol may have different meanings in relation to the various subroutines. For this reason, variables are defined below for each subroutine and for program PHASE2.

SUBROUTINE SHLSRT

A	Array reordered as array X is sorted
NW	Number of words to be sorted
X	Array sorted into decreasing order

SUBROUTINE SIFTUP

L	Number of words in tree
N	Pointer to root at which ordering of root with respect to branches begins
TREE	Array of distances to be sorted

SUBROUTINE SORTK

KN	Number of sorted items to be returned
N	Number of words in array TREE
NN	Pointer to root at which subroutine SIFTUP begins ordering root with respect to branches
TREE	Array containing packed distances between segments and segment numbers

FUNCTION IFIND

IARRAY	Array being searched
LEN	Length of IARRAY
NUM	Number being sought

SUBROUTINE NUMBER

FORM	Output format for number
NUM	Number to be plotted
TEXT	Character representation of number

SUBROUTINE SHAPCOM

AVMD	Map distance conversion factor, in miles per map coordinate unit
BR1	Distance to first break in segment shape, in miles
BR2	Distance to second break in segment shape, in miles
ISF	Shape code when in character form
R	Radius of curvature of circular segments, in miles
RPR	Reciprocal of radius of curvature
SF	Shape code when in binary form
THETA	Slope of line from starting to ending node, in radians
TOTLEN	Total length of segment, in miles
XNF	X-coordinate of ending node
XNI	X-coordinate of starting node
YNF	Y-coordinate of ending node
YNI	Y-coordinate of starting node

SUBROUTINE COORD

BR1	Distance to first break in segment shape, in miles
BR2	Distance to second break in segment shape, in miles
CUMLEN	Cumulative length along segment, in miles
RPR	Reciprocal of radius of curvature of a circular segment
S	Distance along segment since previous break
SF	Shape code
XNF	X-coordinate of ending node
XNI	X-coordinate of starting node
YNF	Y-coordinate of ending node
YNI	Y-coordinate of starting node

SUBROUTINE MAPPLT

AVMD	Map-distance conversion, in miles per map coordinate unit
CUMLEN	Cumulative street length, in miles
FLEN	Array of segment lengths, in miles
INB	Point within map-bounds indicator
ISEG	Array of section assignments

SUBROUTINE MAPPLT (Concl'd.)

ISF	Shape code when in character form
KNODES	Count of nodes
NMAP	Map strip number of current point
NMAPO	Map strip number of previous point
NN1	Array of starting node numbers
NN2	Array of ending node numbers
NODNUM	Array of node numbers
NPPSEG	Number of points plotted per segment
PHGT	Height of map strip, in inches
PLEN	Total length of all plot strips, in inches
SF	Shape code when in binary form
SVAV	Array of map distance conversion factors
TOTLEN	Total segment length, in miles
X	Array of node x-coordinates
Y	Array of node y-coordinates
YCUT	Height of map output strips, in map coordinate units

SUBROUTINE BUILD

BDATA	Array of single 1-bit masks
COMP	Array of near-neighbor data
D	Distance in miles between street midpoints
ISTPR	Array of segment numbers
KN	Number of near neighbors to be found for each segment
KP	Number of words required to save near-neighbor indicators for each segment
MINFR	Array of refuse quantity, total traversal time, and number of houses on segments
N	Number of segments
NNT	Array of near-neighbor segment numbers
NNTEMP	Array of segment numbers
STRING	Array of segment number, refuse quantity, total traversal time, number of houses, and near-neighbor indicators for a segment
TREE	Array of packed distances between segments and segment numbers
UNOT	Array of unit (file) numbers

SUBROUTINE BUILD (Concl'd.)

X	Array of segment midpoint x-coordinates
XT	Array of segment midpoint x-coordinates
Y	Array of segment midpoint y-coordinates
YT	Array of segment midpoint y-coordinates

SUBROUTINE SECTION

BASE	Array of segment number, refuse quantity, traversal time, number of of houses, and near-neighbor indicators for the current base segment
BCOMP	Array of base segment near-neighbor indicators
BDATA	Array of single 1-bit masks
BMINF	Array of base segment refuse quantity, traversal time, and number of houses
COMP	Array of segment near-neighbor indicators
CUML	Refuse quantity of all unassigned segments
CUMT	Traversal time of all unassigned segments
CURL	Refuse quantity on current segment
CURT	Traversal time of current segment
FRACT	Ratio of total refuse quantity to total vehicle capacity
HISTO	Array of number of occurrences of number of shared near neighbors equal to HISTO subscript
ICODE	New section indicator: ICODE = 0 if a section is incomplete or ICODE = 1 if a section is complete
ION	Pointer to unassigned segment sharing most near neighbors with base segment
ISTO	Array of segment numbers
IST1	Array of counts of shared near neighbors
IST2	Array of pointers to segment number
IST4	Array of counts of shared near neighbors
ITR	Number of vehicles of a particular capacity
JON	Count of segments which share near neighbors with the base segment
K	Maximum number of near neighbors found for any segment
KCUTOF	Limit on number of segments sharing the most near neighbors with a base segment to be examined for inclusion in a section with the base segment
KL	Number of unassigned segments
KP	Number of words required for near-neighbor indicators

KPB	Count of words in use in array STRING
MA	Maximum number of segments in first section to be saved for use as base segments
MINFO	Array of refuse quantity, traversal time, and number of houses on a segment
N	Count of segments
NBASE	Segment number of first base segment for first section
NEXTN	Pointer to next base segment from NNTS array
NN	Count of unassigned segments
NNTS	Array of data for base segments
NP	Count of unassigned segments
NSTD	Count of base segments currently saved in NNTS array
NSTO	Maximum number of base segments which can be saved in the NNTS array
NTO	Original number of sections required
NTRUCK	Original number of vehicles required
NUT	Count of unassigned segments on file IUNIT
PC	Count of segments assigned to current section
PK	Count of segments assigned to all completed sections
SECTN	Number of current section
SMLD	Minimum total refuse which must be assigned before current section is completed
STRING	Array of segment number followed by MINFO and COMP arrays
TDUMP	Unloading time at the landfill, in minutes
TOTL	Total refuse quantity for all segments
TOTT	Total traversal time for all segments
TRUCK	Array of vehicle capacity, maximum trip time, load, actual trip time, pointer to first segment in section, count of segments section, and count of houses in section, for each section
TRUCKS	Array of quantity, capacity, and maximum trip time for vehicles of each capacity
UNOT	Array of unit (file) numbers
X	Array of segment midpoint x-coordinates
Y	Array of segment midpoint y-coordinates

PROGRAM PHASE2

FLEN	Array of street segment lengths, in miles
FMPH	Array of speed limits, in mph
ISECTN	Count of sections
KCUTOF	Limit on number of segments sharing the most near neighbors with a base segment to be examined for inclusion in a section with the base segment
KN	Count of near neighbors to be found for each segment
KNODES	Count of nodes
KP	Count of words required to save near-neighbor indicators
MA	Maximum number of segments in first section to be saved for use as base segments
MAPS	Count of section maps to be plotted
MINFR	Array of refuse quantities, total traversal times, and number of houses on segments
NA	Count of segments
NBASE	Segment number of first base segment for first section
NH	Array of count of houses on a segment
NSEG	Count of segments
NT	Array of count of vehicles of each capacity
REF	Refuse quantity
RQF	Refuse quantity adjustment factor
TC	Array of vehicle capacities
TMXTR	Maximum trip time
TOTL	Total refuse quantity
TRUCK	Array of vehicle capacity, maximum trip time, load, actual trip time, pointer to first segment in section, count of segments in section, and count of houses in section, for each section
TRUCKS	Array of quantity, capacity, and maximum trip time for vehicles of each capacity
UNOT	Array of unit (file) numbers
X	Array of x-coordinates of segment midpoints
XNOD	Array of node x-coordinates
Y	Array of y-coordinates of segment midpoints
YNOD	Array of node y-coordinates

(The reverse of this page is blank.)

APPENDIX D

SAMPLE PRINTED OUTPUT

```
INPUT VEHICLE DATA              KIRTLAND AFB (EAST) NM

CAPACITY     TRIPS
   220.        8.
     0.        0.
     0.        0.
     0.        0.

MINIMUM FILL FRACTION=  .931

STOP TIME PER HOUSEHOLD  =          .50 MINUTES
STOP TIME PER UNIT REFUSE=         0.00 MINUTES
UNLOADING TIME           =        15.00 MINUTES
MAXIMUM TRIP TIME        =       240.00 MINUTES

THE FIRST SECTION WILL START WITH SEGMENT    1
```

```
TRIP  CAPACITY  TIME LIMIT    LOAD       TIME  SEGMENTS  TOTAL NM

  1    220.00     240.00    208.00       129.      19.      208.
  2    220.00     240.00    205.00       124.      14.      205.
  3    220.00     240.00    207.00       125.      19.      207.
  4    220.00     240.00    211.00       143.      53.      211.
  5    220.00     240.00    213.00       135.      54.      213.
  6    220.00     240.00    211.00       144.      37.      211.
  7    220.00     240.00    192.00       116.      20.      192.
  8    220.00     240.00    191.00       141.      29.      191.

SECTION   1 CONTAINS SEGMENTS

     1   2   3 149  18  61  17   4 150  68  66  60   5  64  59  35  34  36  69

SECTION   2 CONTAINS SEGMENTS

    16  67  65  72  27  26  25  15  47  48  33  32  24  56  23  49  55  31  14

SECTION   3 CONTAINS SEGMENTS

    57  38  22  52  30  37  21  29  39   9  28  58  51  73  20  13  50  19  10

SECTION   4 CONTAINS SEGMENTS

    11  70  12   8  41  40  43   7  42  53  44  54  45  62   6  74  71 168  75  81  76  46  63  82  84 169  83  77 159  78
   144  85 120 118 145 152 160 153 154 155 167  79 119 117 121 146 114 116 110  80 111 122 112

SECTION   5 CONTAINS SEGMENTS

   147 123 102 100  99 124 101 140 141 139 125 103 137  90 127  91  92  89  94 135  93  95 134 142 136  96 113  98 143  97
   133 148 161 104 126 138  88 105 132  87 131  86 130 129 109 108 106 128 157 156 107 233 158 231

SECTION   6 CONTAINS SEGMENTS

   205 206 204 232 203 207 202 234 170 208 209 235 239 201 200 230 244 199 240 229 198 241 197 246 228 196 210 171 226 236
   211 245 172 173 195 174 237

SECTION   7 CONTAINS SEGMENTS

   183 166 184 247 165 185 164 175 248 163 176 162 186 187 177 189 178 180 188 179

SECTION   8 CONTAINS SEGMENTS

   181 182 190 191 212 238 192 213 214 215 225 216 115 151 193 194 217 218 219 220 221 222 223 224 227 242 243 249 250

MAPPLT PARAMETERS FOR MAP 1
AVMO=     .18939
XMIN=  0.00000          XMAX=  10.00000          YMIN=    0.00000          YMAX=  10.50000
 XL=   0.00000            XR=  10.00000            YB=    0.00000            YT=  10.50000
XSC=   1.40000           YSC=   1.40000          PHGT=   30.00000          PLEN=  15.00000          YCUT=  21.42857
```

(The reverse of this page is blank.)

GLOSSARY

Air Force Refuse-Collection Scheduling Program: a set of four computer programs that perform residential refuse-collection scheduling and produce printed schedules and maps of the routes.

base segment: a segment used to limit the addition of other segments to its section on the basis of the number of neighboring segments common to both.

binary search: a procedure for finding one item in an ordered group by repeatedly halving the portion of the group that contains the item.

map coordinate unit (MCU): the length, in inches, between integral divisions on the coordinate system appended to a map.

node: a numbered point on a street at which some characteristic of the street changes.

pointer: a variable that gives the location of some other variable.

segment: a portion of a street between two nodes.

shape code: characters--either two letters or a letter followed by a number--that indicate the shape of a street segment.

spatial clustering of streets: selection of streets traversed by a vehicle on one trip so that the streets are connected by other streets that must be traversed.

(The reverse of this page is blank.)

INITIAL DISTRIBUTION

ADTC/CS	1
DDC/DDA	2
HQ AFSC/DL	2
HQ USAF/RDPS	1
AFIT/Library	1
AFIT/DE	1
AFIT/LSGM	1
National Science Foundation	1
EPA/ORD	1
USA-CERL/EH	1
USA Chief, R&D/EQ	1
USN Chief, R&D/EQ	1
AFETO/DEV	1
Hq AUL/LSE 71-249	1
Det 1 ADTC/TST	1
Det 1 ADTC/ECW	3
Det 1 ADTC/EC	1
USA-CERL/Library	1
USA-CERL	1
UNM-CERF	3

(The reverse of this page is blank)