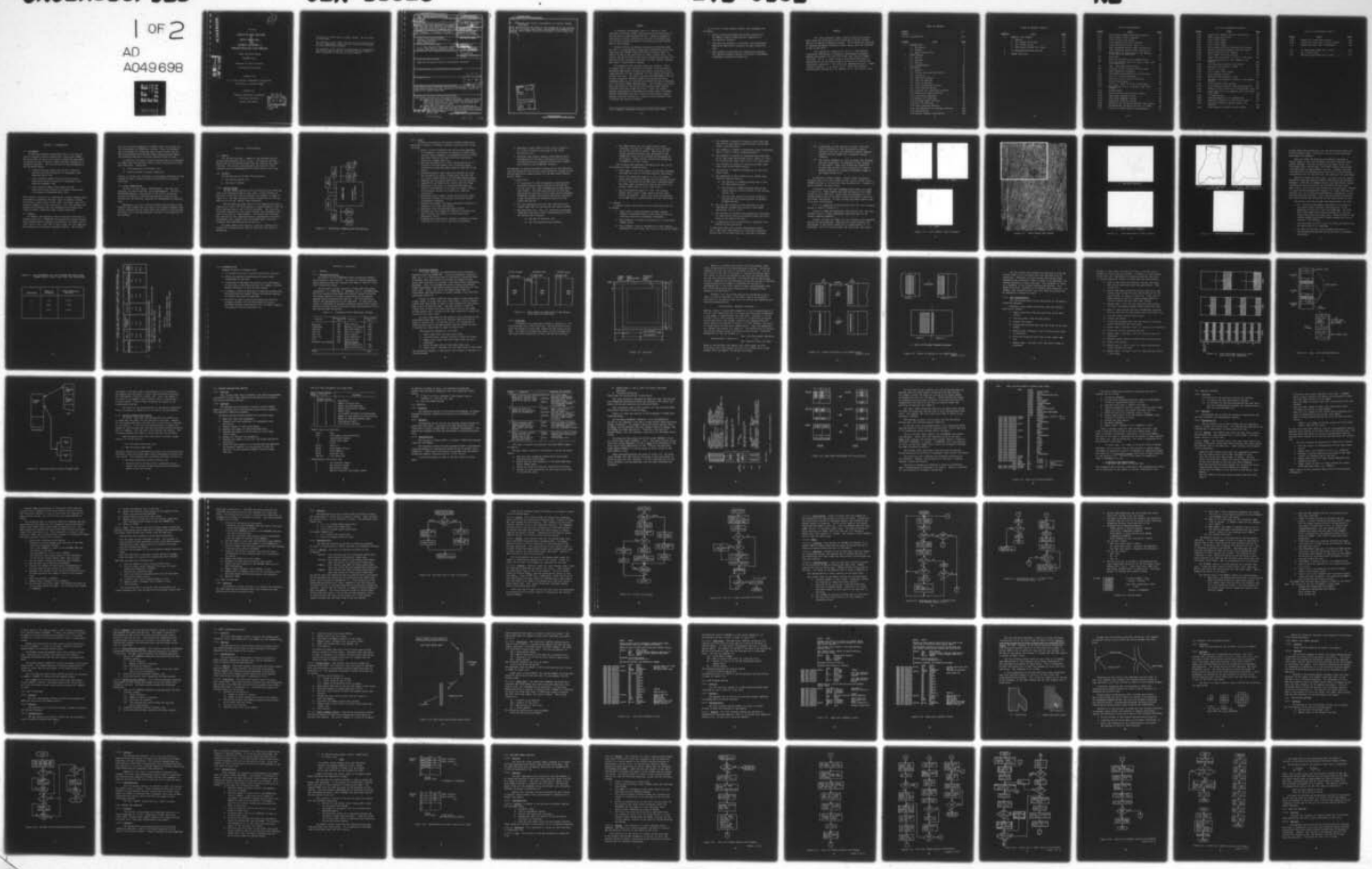
AD-A049 698 GOODYEAR AEROSPACE CORP AKRON OHIO F/G 9/5
ASSOCIATIVE ARRAY PROCESSING OF RASTER SCANNED DATA FOR AUTOMAT--ETC(U)
NOV 77 N J ADAMS, J M VOCAR, K LOSCH DAAG53-76-C-0146
UNCLASSIFIED GER-16523 ETL-0132 NL

1 OF 2
AD
A049698



(12)

ETL-0132

ASSOCIATIVE ARRAY PROCESSING OF RASTER SCANNED DATA FOR AUTOMATED CARTOGRAPHY II (IMPROVED RESOLUTION & DATA HANDLING)

FINAL TECHNICAL REPORT

NOVEMBER 1977

Approved for Public Release:

Distribution Unlimited

Prepared For

U. S. Army Engineer Topographic Laboratories

Fort Belvoir, Virginia 22060

Prepared By

Goodyear Aerospace Corporation

1210 Massillon Road

Akron, Ohio 44315

Destroy this report when no longer needed. Do not return to originator.

The findings in this report are not to be construed as an official Department of Army position unless so designated by other authorized documents.

The citation in this report of trade names of commercially available products does not constitute official endorsement or approval of the use of such products.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ETL-0132

2. GOVT ACCESSION NO.:

3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): ASSOCIATIVE ARRAY PROCESSING OF RASTER SCANNED DATA FOR AUTOMATED CARTOGRAPHY II (IMPROVED RESOLUTION and DATA HANDLING)

5. TYPE OF REPORT & PERIOD COVERED: Final technical rept. May 1976 - Oct 1977

6. PERFORMING ORG. REPORT NUMBER: GER-16523

7. AUTHOR(s): N. J. Adams, J. M. Vocar, K. Losch

8. CONTRACT OR GRANT NUMBER(s): DAAG53-76-C-0146

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Goodyear Aerospace Corporation
1210 Massillon Road
Akron, Ohio 44315

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:

11. CONTROLLING OFFICE NAME AND ADDRESS:
U. S. Army Engineer Topographic Laboratories, Fort Belvoir, Virginia 22060

12. REPORT DATE: Nov 1977

13. NUMBER OF PAGES: 150 (63 p.)

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

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

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Associative Array Processing, automated cartography, line thinning, line symbol generation, line separation by thickness, raster to vector, run length coded data

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

The primary objective of this effort was to:

(a) expand the existing STARAN cartographic raster processing capability from 4-mil in/out processing to: 4-mil input with 4-mil output; 2-mil input with 2-mil output; and 1-mil input with 1-mil output;

(b) expand the I/O capabilities to include processing of ETL run-length-coded data and DMA run-length-coded data; and

→ next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

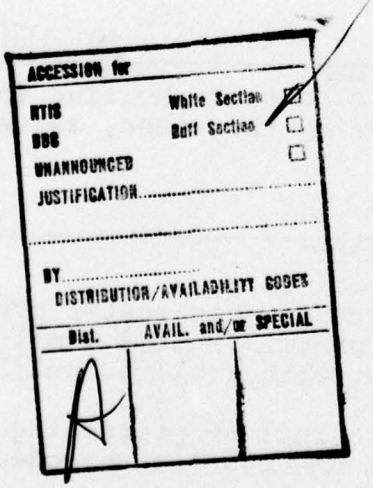
UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

156 800 JOB

20. c) perform some initial investigation of contour tagging techniques.

These modifications were based on the processing of map overlays up to 19 x 22 inches (data area), and include the capability to generate those symbols previously provided at 4-mil (only) processing.



SUMMARY

In automated cartography, there is a desire to take advantage of the much faster digitizing and plotting speeds of the new raster scanner/plotter devices. However, sequential processing of raster scanned data has shown an extremely poor fit in terms of both the speed of processing and the development of the software.

Previous efforts by Goodyear Aerospace Corporation (GAC) for the U. S. Army Engineer Topographic Laboratories (ETL) have shown that the STARAN* Associative Array Processor (AAP) is ideally suited to the processing of raster scanned data. As a result, the AAP provides a much faster processing rate than that obtained using a sequential approach, and the software development time is significantly reduced.

The purpose of this present effort was twofold: first, to remove some of the limitations resident in the previously developed experimental software and second, to perform initial investigations into AAP techniques for the tagging by height of contour line data. The result has been the development of a set of semi-production software capable of producing the basic symbology types at all required resolutions and easily adaptable to all existing STARAN configurations. This software can accept data from either of the two existing raster/scanner devices currently installed at DMA facilities. Also, the contour tagging investigations indicated an extremely good fit between the AAP architecture and the solution to the contour tagging problem, and have resulted in a contract being awarded to GAC to develop the required software.

---

*T.M., Goodyear Aerospace Corporation, Akron, Ohio 44315

As a result of these present efforts, GAC recommends the following:

1. That all existing software be quickly installed in the DMA/ETL Digital Image Analysis Laboratory at Fort Belvoir, Virginia.
2. That once the software is installed, the Input/Output modules be optimized to maximize the data processing rate.
3. That software be developed to automatically generate the remaining symbols required to provide an automated cartographic production capability.

PREFACE

This final technical report records efforts and achievements under the Associative Array Processing of Raster Scanned Data for Automated Cartography program. This program was conducted by Goodyear Aerospace Corporation (GAC), Akron, Ohio and submitted by GAC as GER-16327.

The program was conducted for the U. S. Army Engineer Topographic Laboratories, Topographic Developments Laboratory, Mapping Developments Division (ETD-TD-M) under contract DAAG53-76-C-0146. Mr. R. A. Clark served as the Contracting Officer's Representative and provided valuable assistance in reaching the contractual objectives. This effort was implemented by N. J. Adams, (project engineer), J. M. Vocar, and K. Losch with technical contributions provided by R. W. Messner, R. W. Lott, and R. Faiss.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT'D)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (CONT'D)

LIST OF ILLUSTRATIONS (CONT'D)

SECTION I - INTRODUCTION

1.1 BACKGROUND

In previous contracts performed by GAC, in the area of raster processing, results showed that a four-array STARAN AAP could offer at least a two order of magnitude improvement in processing time of raster scanned data when compared to a sequential approach. The work and results were documented in the following reports.

1. Interim Technical Report (ETL-CR-74-1) entitled Associative Array Processing for Topographic Data Reduction March 1974.
2. Final Technical Report (ETL-CR-74-20) entitled Associative Array Processing for Topographic Data Reduction November 1974.
3. Final Technical Report (ETL-0046) entitled Associative Array Processing of Raster Scanned Data for Automated Cartography March 1976.

These efforts resulted in an experimental software package which was capable of producing the basic types of symbols, namely, solid lines, double lines, broken lines, railroads, area fills, and point symbols, at 4-mil resolution. Also, the software was capable of performing both line separation by thickness and raster-to-vector conversion under limited conditions.

1.2 PURPOSE

In general, the purpose of this contractual effort was to produce a base for a STARAN AAP semi-production software system. This software would be capable of processing all normal resolutions (4-mil, 2-mil, and 1-mil), sheet sizes, and line densities and producing all previously generated symbols. The software

was to be directly adaptable to STARAN stand-alone systems and easily adaptable to STARAN-host systems. Also, the software was to be capable of processing data scanned on both the ETL-IBM scanner/plotter and the DMA raster scanner/plotter (RAPS), and producing data which could be plotted on both of these.

In addition, two other software programs were to be expanded to handle high line densities under similar constraints to those described above. These were:

a) Line separation by thickness, and

b) raster-to-vector (linear) conversion.

Finally, an effort was initiated to investigate techniques for the tagging of raster scanned contour data with their associated elevations.

1.3 REPORT ORGANIZATION

Section 2 of this report, INVESTIGATION - describes the approach, the results derived, and recommendations. Section 3, DISCUSSION-contains a description of the overall processing sequence followed by a detailed description of each routine divided into its function, approach and implementation technique used.

Appendices A, B, and C contain detailed descriptions of the magnetic tape formats that are used in this system, together with utility routines used to either check the results or provide some additional aid to the GAC personnel involved in the generation of the software.

SECTION 2 - INVESTIGATION

2.1 GENERAL

This section provides a summary of the approach used to perform the previously defined goals. The approach includes a description of the testing system that was used and the tasks involved. This section also documents the results of this contract and the recommendations for future work in this area.

2.2 APPROACH

The approach may be divided into two parts:

1. The testing system, and
2. the tasks involved.

2.2.1 Testing System

The GAC STARAN Evaluation and Training Facility (SETF) was used as the testing system for this work. A block diagram of that part of the SETF used to generate the software is shown in Figure 2-1. The tape transports used to read/write the overlay data are 37-1/2 ips, 800 bpi tape transports.

The two RK05 cartridge disks are 1.45 megabits/sec with a capacity of 1.228×10^6 16-bit words, a rotation speed of 1500 rpm and an average access time of 50 milliseconds. The first disk drive DK0 contained all the system software and cartography software in the form of linked modules. The second disk drive DK1 was reserved for the reformatted Input (unprocessed) and Output (processed) data, together with off-loaded tables, and semi-processed data generated during processing.

The Comtal 8000 series display is used as a debug aid to display four adjacent array loads of data after any stage of processing.

Figure 2-1. Stand-Alone STARAN System Configuration

2.2.2 Tasks

In order to accomplish the overall software capabilities described in Section I, Goodyear Aerospace performed the following tasks.

- Wrote a Fortran preprocessing routine which generates all parameters required by the executive (controlling) routine; these parameters are generated from an initial card deck input by the user containing resolution, sheet size, symbol type, etc.
- Modified the executive to allow the generation of all currently required symbols while retaining enough flexibility to add new symbol types as required in the future.
- Developed magnetic tape read/write software to allow the processing of raster scanned data from both the ETL raster scanner/plotter and the DMA scanner/plotter.
- Expanded the line symbol generation, and line editing capabilities to handle 1-mil, 2-mil, and 4-mil data.
- Generated a new area fill technique which does not require the production of an area fill symbol library. The resulting routines will handle all required resolutions.
- Developed a point symbol library and new point symbol generation routines to minimize I/O and yet satisfy cartographic standards.
- Expanded the broken line symbol generation software to operate at all three resolutions.
- Generated a new railroad symbology routine.
- Expanded the line separation routines to operate with user selected line widths, variable within predetermined limits.
- Expanded the vectorization software to operate on dense line data at 1-mil, 2-mil, and 4-mil resolutions.

- Developed a Comtal debug routine used to inspect a 2 x 2 set of arrays at any stage of the raster processing cycle.
- Performed some contour tagging investigations which provided part of the basic information for writing a proposal which was accepted called Contour Tagging and Gridding using STARAN/CDC software.

The approach to testing the above software was to use digitized line data generated from various source materials scanned on both the ETL-IBM raster scanner/plotter and the DMA raster scanner/plotter (RAPS).

In particular;

- ETL used their ETL-IBM cartographic scanner/plotter to perform further raster scanning at Fort Belvoir of 2-mil and 1-mil Lake Istokpoga source data (binary plot format). This data was minimized to that data that would be required to fully check out the expanded STARAN AAP software and satisfy GAC and ETL personnel to the correctness and resolution quality of the symbolized results.
- ETL also generated 2-mil and 1-mil resolution run-length coded data for testing the new Input routines.
- DMA generated 2-mil, and 1-mil resolution run-length-coded data from their raster scanner/plotter suitable for testing the following:
    1) The new I/O routines, and
    2) the line separation by thickness.

- GAC demonstrated the line symbolization, point symbology, and area fill symbology to ETL personnel during the contractual effort. While most of the raster processing software was developed for the STARAN AAP, some utility software was developed for the Sigma 9. These utility routines are documented in Appendix B of this report.

The following AAP processed information was sent to ETL for plotting on their scanner/plotter.

- Three tapes of all point symbols on the Lake Istokpoga map sheet for each of the three possible resolutions.
- Three line separation tapes of a 10" x 10" graphic which contains three different line thicknesses scanned at 1-mil resolution on the DMA raster scanner/plotter. The three tapes correspond to each of the three line widths, namely 4-mils, 7-mils, and 16-mils.
- Two line separation tapes of a 10" x 10" contour graphic which contain index contour and non-index contour information. Again, this data was generated on the DMA raster scanner/plotter at 2-mil resolution.

2.3 RESULTS

In general terms, the results of this contractual work are as follows.

1. There exists a semi-production software package capable of performing many of the raster processing functions required in automated cartography.
2. This software is capable of operating on a stand-alone STARAN system.
3. This software is easily adaptable to a host computer connected to a one array, two array, or four-array STARAN.

4. The software is capable of handling Input data from both the ETL-IBM raster scanner/plotter and the DMA raster scanner/plotter.*

5. The software is capable of generating Output (processed) data for both of these scanners/plotters.*

6. The software is capable of processing data digitized at all the normal resolutions (4-mil, 2-mil, or 1-mil) for standard map sheet sizes of 19" x 22" data areas.

7. This software is capable of processing larger sheet sizes with minimal modifications.

8. The software is capable of generating all the basic symbol types.

9. The generation of the different basic symbol types have been demonstrated by:

   a) the plotting of STARAN processed data on the ETL-IBM scanner/plotter,

   b) the display of STARAN processed data on the Comtal graphics screen existing in the testing system, and

   c) the hard copy output of polaroid pictures of sub-sections of processed sheets using GAC's DICOMED Recorder.

10. The fast data processing rate obtained with the experimental software also exists at the higher resolutions (2-mil, 1-mil).

11. The software can perform line separation by thickness of lines within certain pre-defined limits and selectable at run-time by the user.

12. A capability exists to perform basic automatic line editing during processing.

* It should be noted that crossed input/output formats (i.e., IBM input, DMA output) will have some distortion due to 1-mil vs 25μ SPOT SIZE (or 2 mil/50μ,4 mil/100μ).

13. A technique for the tagging of contour lines with their elevations has been devised which should provide a base for a cost-effective contour tagging system using the STARAN and a sequential (host) computer.

14. The current software has built-in debug aids allowing the easy maintenance and upgrading of the software in the future. These aids include the capability to pass data currently being processed to a Comtal graphics display for viewing at different stages of the symbolization cycle.

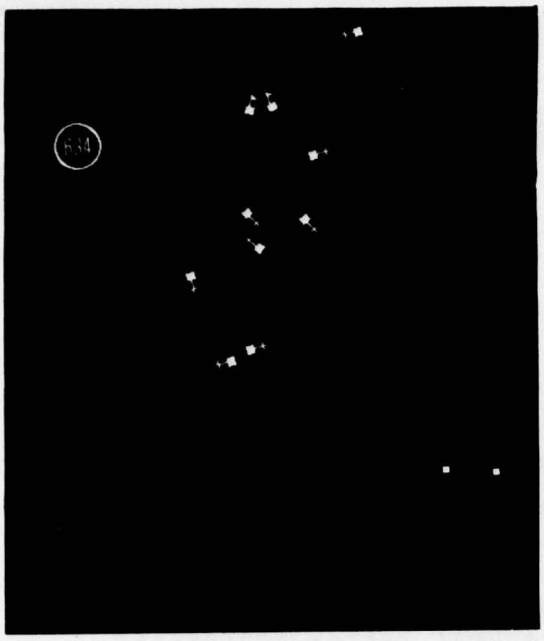
In addition to these results, several raster processing functions were run on the STARAN and the results plotted using a DICOMED recorder to provide hard copy documentation. Examples of these processes are given below.

A subsection of the Lake Istokpoga point symbols are shown in Figure 2-2. These symbols have been produced on the DICOMED plotter at twice map size. Notice that the deformities in the rotated symbols such as churches, schools are greatly reduced at the higher resolutions. In fact, at map size these symbols seem to be quite satisfactory.

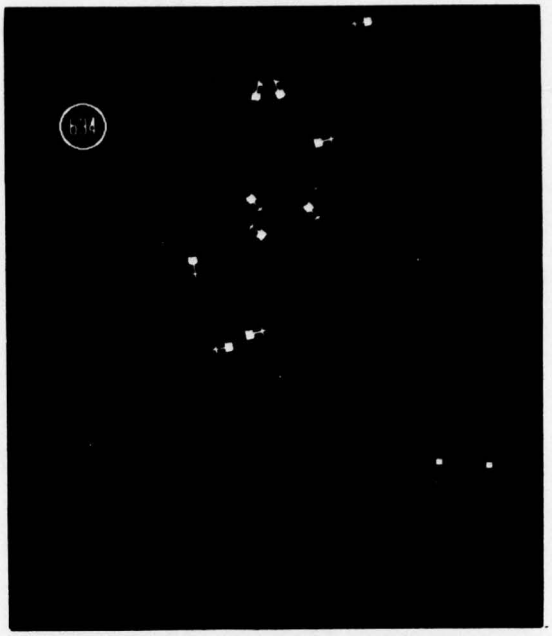
A 4" x 4" subsection of the original 10" x 10" 2-mil contours source is shown in Figure 2-3.

Figure 2-4 shows the separated index (10 mil) and, non-index contours (4-mil). These separations constitute part of the contour tagging procedure.

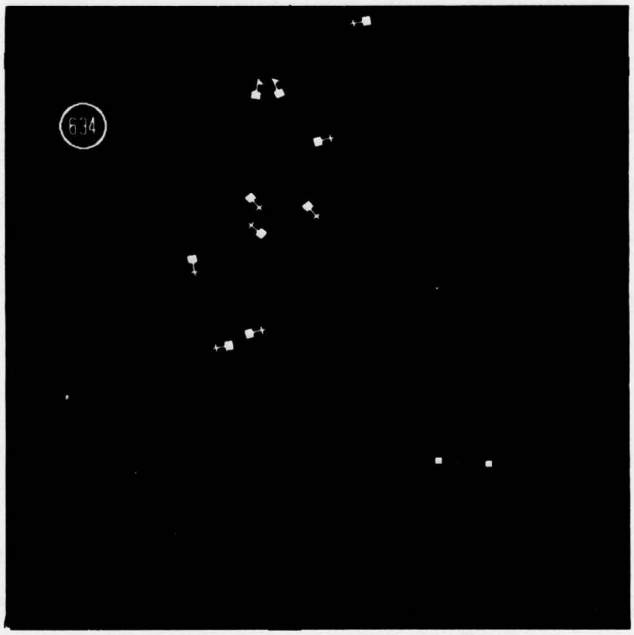
Figures 2-5, 2-6, 2-7 show a 2" x 2" section of the woodlands overlay during processing. These pictures were produced from photographs of the Comtal screen and are approximately 3:1 in magnitude. The first picture shows the input data, the second



A. 4-MIL



B. 2-MIL



C. 1-MIL

Figure 2-2. Point Symbols (Lake Istokpoga)

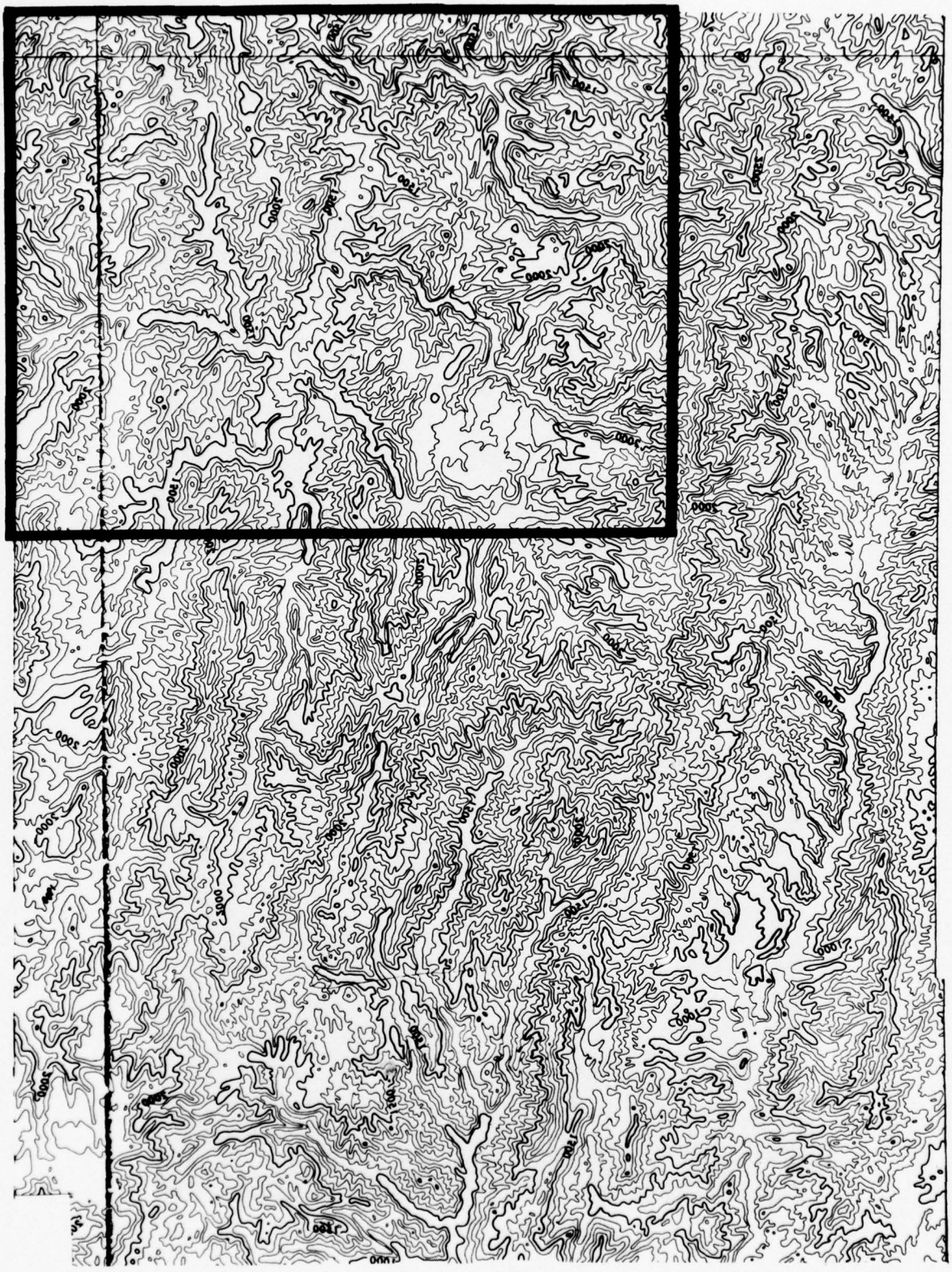
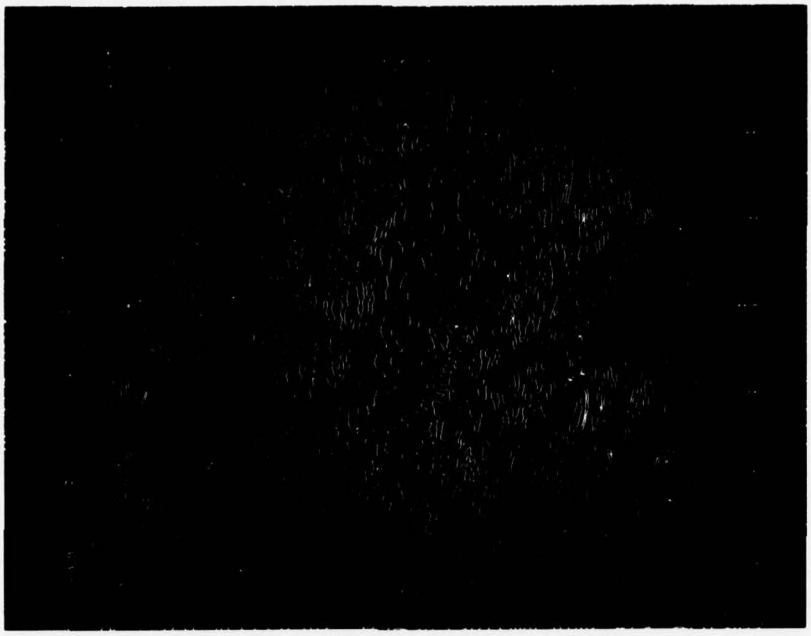
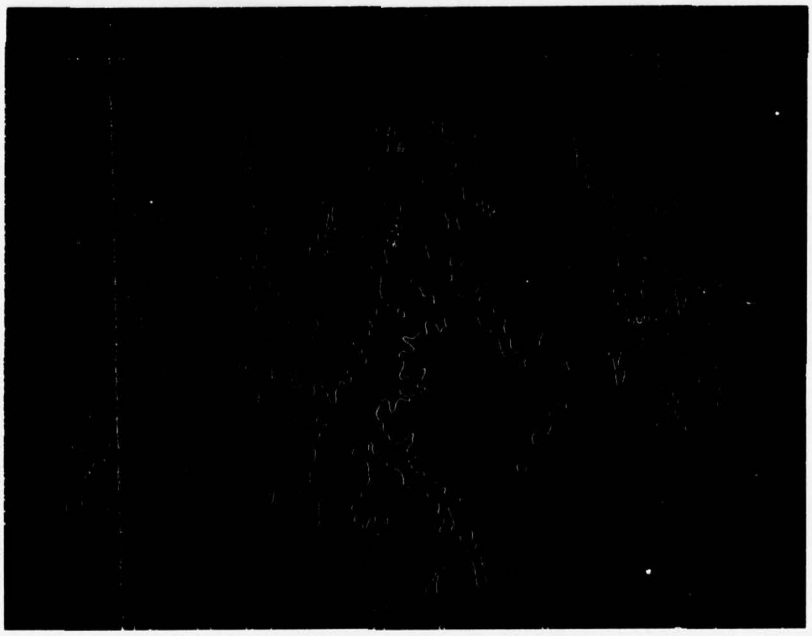


Figure 2-3. 2-Mil Contour Data Source



a. NON-INDEX CONTOURS



b. INDEX CONTOURS & NUMBERS

Figure 2-4. Line Separation of 2-Mil Contours

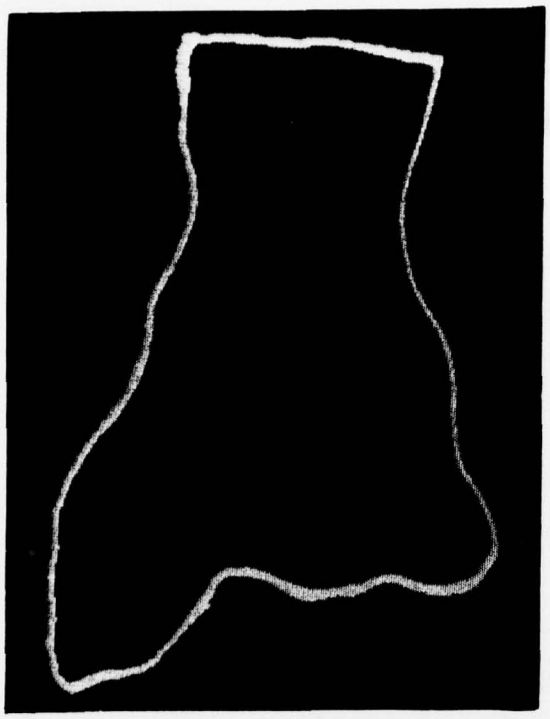


Figure 2-5. 4-Mil Woodlands/ Outline Input 3:1

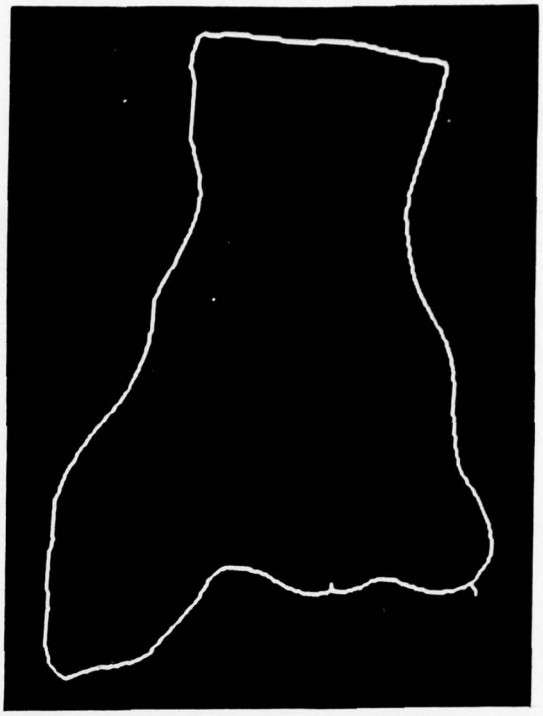


Figure 2-6. 4-Mil Woodlands Outline After Thinning 3:1

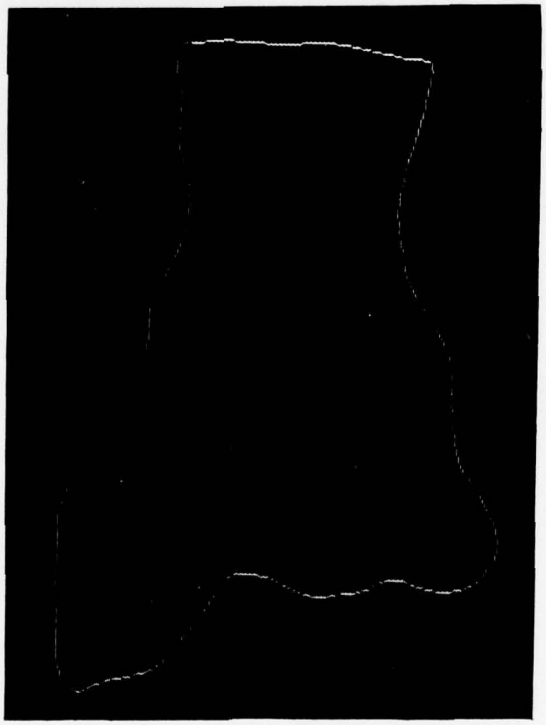


Figure 2-7. 4-Mil Woodlands Outline After Editing 3:1

picture shows the thinned data, and the third picture shows the edited (clipped) data. This process works on both closed and open-ended line segments.

Table 2-I shows the measured performance to generate a typical symbol on the Lake Istokpoga map sheet using the cartographic package. The time to generate any symbol (area fill, line symbol using vectorizing, line symbol without vectorizing, line separation) is approximately the same. However, the time to perform the array processing routines such as line thinning, line editing, etc. do vary with the line density of the map sheet.

Table 2-II shows the total times, including I/O, for symbol generation using the existing STARAN S-500 stand-alone system at Goodyear. Since the I/O depends greatly upon the overall system configuration and the speed of the peripheral devices (disk, tapes, etc.), this area of the software was not optimized. However, in a semi-production/production system it is highly desirable to minimize this I/O by using fast peripherals, overlapping I/O and reformatting where possible. Also, the use of double buffering will help minimize the I/O portion of the total processing time.

Conclusions that can be drawn from these results are that:

- Particularly at the higher resolutions it is necessary to produce clean, "thinned" line data prior to the generation of area fill, and line symbology. The automatic open-ended line clipping routines go a long way towards providing such clean data. Future efforts in the contour tagging area should further enhance the area of automatic line joining.
- The resolutions of 2-mil, and 1-mil seem satisfactory for the generation of all symbology.
- The tape I/O and disk I/O are significant parts of a semi-production/production system and must be addressed accordingly.

TABLE 2-I - AAP PERFORMANCE FOR LAKE ISTOKPOGA MAP SHEET USING STARAN S-500 AT 4-MIL, 2-MIL, AND 1-MIL RESOLUTIONS

RESOLUTION	ARRAY I/O (SECONDS)	ARRAY PROCESSING (MINUTES)
4	4.5	0.42
2	18.0	1.30
1	72.0	4.23

TABLE 2-II - TIMINGS FOR LAKE ISTOKPOGA USING A STARAN S-500 STAND-ALONE FOR 19-x22-INCH MAP SHEET, SCANNED AT 4-MIL, 2-MIL, AND 1-MIL RESOLUTIONS

RESOLUTION	TAPE I/O *	BUFFER-DISK REFORMAT **	DISK I/O **	MISCELLANEOUS	ARRAY (I/O & PROCESSING	TOTAL (MIN)
4	3.8	2.1	1.7	0.4	0.5	8.5
2	14.5	9.5	6.9	0.5	1.6	33.0
1	54.0	42.0	28.0	0.6	5.4	130.0

STARAN S-500 STAND-ALONE PERIPHERAL SPECIFICATIONS

* TAPE SPEED OF 37.5 I.P.S. 800 B.P.I., 24 M-BIT/SEC RATE
(ASSUMES BINARY PLOT FORMAT DATA

** DISK 1.45 M-BIT/SEC TRANSFER RATE

1500 R.P.M.
11.1 μSEC/16-BITS TRANSFER RATE
TRACK POSITIONING (AVERAGE) 50 MSECS

2.4 RECOMMENDATIONS

Goodyear Aerospace recommends that:

- All software be quickly installed at the ETL facility.
- Efforts be applied to developing efficient (fast) Input/Output of the data.
- A technique be developed to perform an on-line merging of such data as the digitized area fill symbols, and the merged information of each colored overlay.
- A software routine be developed to generate automatically the remaining point symbol library information required to produce the remaining symbology.
- An effort be made to continually update the existing software routines with new techniques that may be developed through investigations such as contour tagging and enhanced editing techniques, etc.

SECTION 3 - DISCUSSION

3.1 GENERAL

3.1.1 Section Description

The details of the STARAN-AAP raster processing software are discussed in this Section. The specific routines developed and/or modified under this contract are shown in Table 3-I with their instruction counts.

The discussion of each routine is broken down into three subsections: 1) FUNCTION, 2) APPROACH, and 3) IMPLEMENTATION. The first subsection describes the routine's specific function. The second subsection includes a description of the general approach taken to perform the function. The last subsection details the implementation of that approach. In some cases the reader is referenced to previous technical reports which contain material relevant to the discussions herein and which may clarify the routine under discussion.

Table 3-I. STARAN-AAP RASTER PROCESSING SOFTWARE

Routine	Instructions (No.)	Routine	Instructions (No.)
Executive	300	Line Editing	300
Preprocess	225	Broken Line Symbols	1100
Tape/Disk	275	Railroad Symbols	820
Tape I/O	550	Area Fill Symbols	135
Disk I/O	40	Point Symbols	1300
Array I/O	235	Line Separation	225
Thin	225	Vectorization	1200
Variable Line Symbols	175	Miscellaneous	
		Comtal Debug Aid	400
		HSDB (Definition)	150
TOTAL			7655

3.1.2 Processing Sequence

It is very important to understand the data processing sequence within the STARAN AAP. Since data movement (read/write tape, etc.) is a significant part of the overall processing time, it becomes clear that efforts to minimize the amount of data during data transfers will provide a desirable overall performance advantage. Also, the intermediate buffer storage requirements become unreasonably large when processing high resolution (2- and 1-mil) data. Hence, the use of run-length-coded data for Input/Output and vectorization as output becomes a very desirable processing capability.

Figure 3-1 helps to define, with respect to the image area, a tape record, an array load, and a data block. The ETL scanner/plotter plot tape format is used (in this example) for the source tape as well as the plot tape. The tape record, as shown in Figure 3-1A, is eight scan lines wide and as long as one image. In other words, the first byte of the tape record will contain the first cell of 8 scan lines.

An array load refers to the amount of data processed by one associative array and is 224 cells wide and 248 cells long (as shown in Figure 3-1B). To load one array, it is necessary to read 224 scan lines or 28 tape records, and this amount of data is defined as a data block as shown in Figure 3-1C.

The processing sequence is as follows:

1. Read a data block from the source tape,
2. Load the arrays, process the data, unload the arrays,
3. Repeat step 2 until the entire data block has been processed,
4. Write the data block to the output tape, and,
5. Repeat steps 1 through 4 for all data blocks of the image.

The processing sequence is described in more detail in Section 2.3, Testing system.

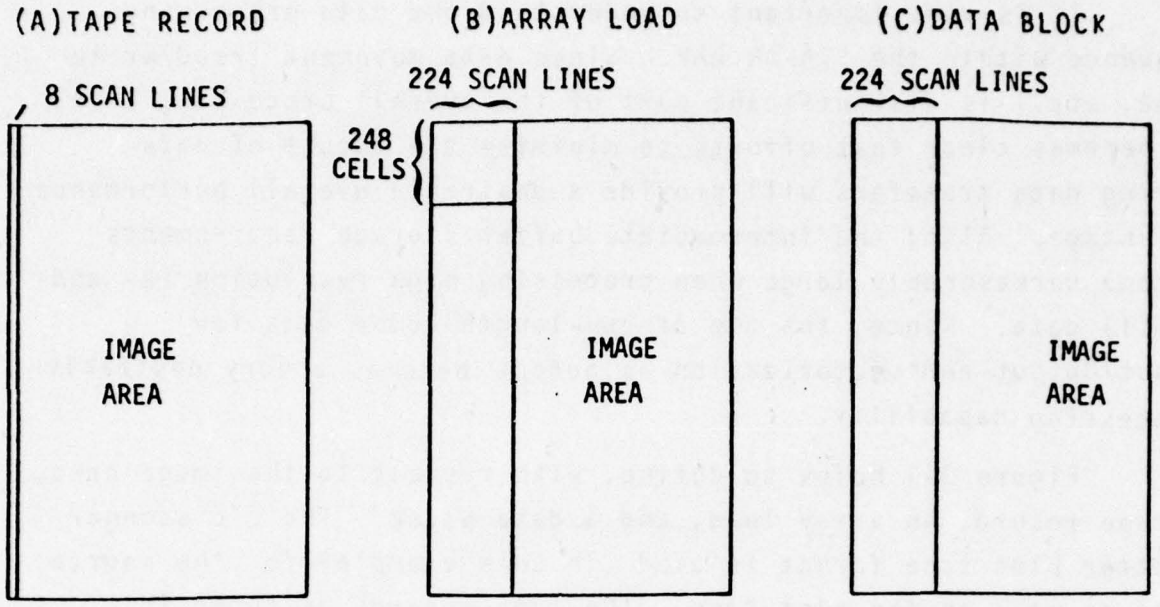


Figure 3-1. Relationship of Image Area to Tape Record, Array Load, and Data Block

3.1.3 Array Map

A diagram of the array map is shown in Figure 3-2. The solid outside line is the array size. The solid inside line shows the amount of source data loaded into the arrays at one time. The dashed line shows the amount of data unloaded from the arrays after processing (in this case 32 cell overlap).

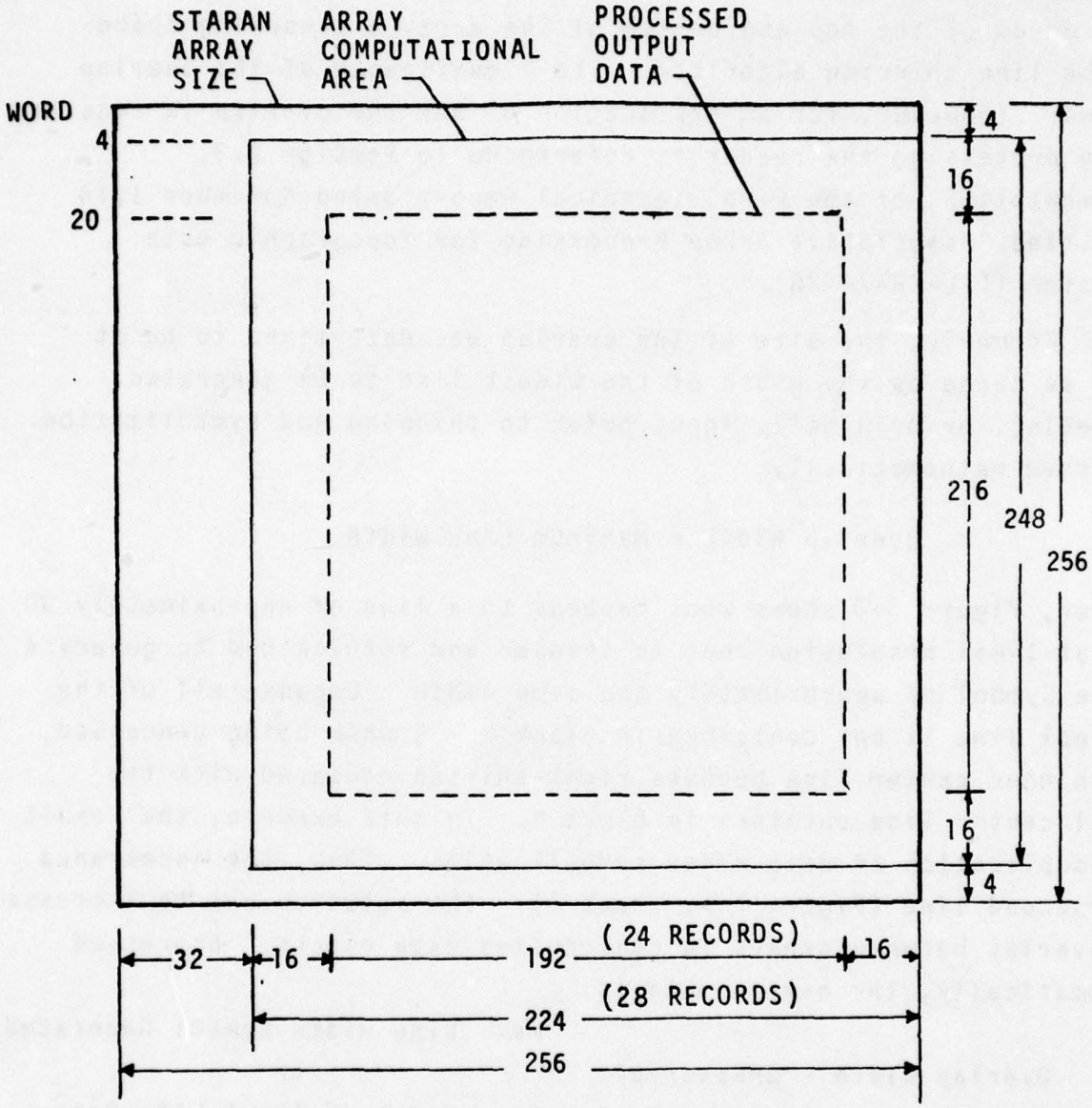


Figure 3-2. Array Map

There is a variable size overlap (N cells) between array loads and data blocks to eliminate boundary problems. Each array discards the outer N/2 cells as indicated in Figure 3-2. The 32-cell field on the left of the array is reserved as working space. The 4 words at the top and bottom of the array are working space for the line thinning algorithm. The significance of the overlap follows. (However, for an explanation of why the overlap is required during processing the reader is referenced to Section 2.3, Implementation, of the Final Technical Report dated November 1974 and titled "Associative Array Processing for Topographic Data Reduction (ETL-CR-74-20).")

Formerly, the size of the overlap was determined to be at least as large as the width of the widest line to be generated (32 cells), or originally input prior to thinning and symbolization. Expressed mathematically:

Overlap Width = Maximum Line Width.

However, Figure 3-3 shows what happens to a line of approximately 30 mils at 1-mil resolution that is thinned and rethickened to generate a Line Symbol of approximately the same width. Because all of the original line is not contained in block N + 1 when being processed, the thinned center line becomes right-shifted compared with the actual center line obtained in block N. In this example, the result is a duplication of data after symbolization. Thus, the appearance of a second line (Figure 3-3, sheet 2). The solution was to increase the overlap between arrays in consecutive data blocks. Expressed mathematically, the new formula is:

Overlap Width = Greater of: Max. Line Width Symbol Generated or Max. Width of Input Line Data.

Based on the possible line symbols that could appear at 1-mil resolution, this width becomes 72 cells for a symbol such as Dual Highway #820 (see Symbols for Medium Scale Maps).

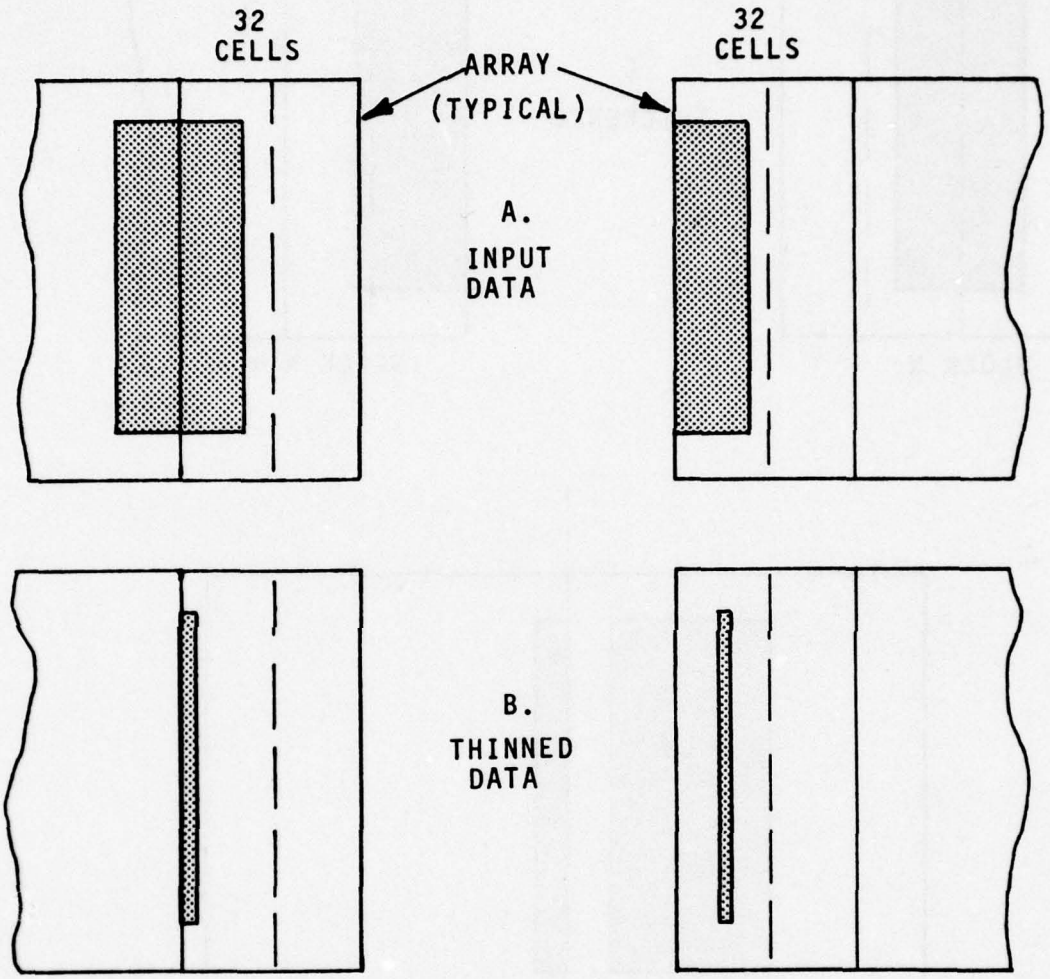
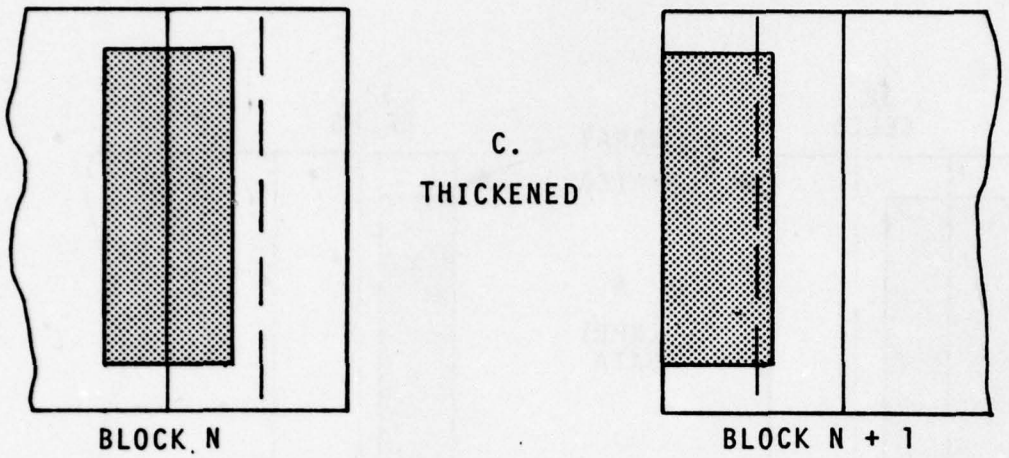
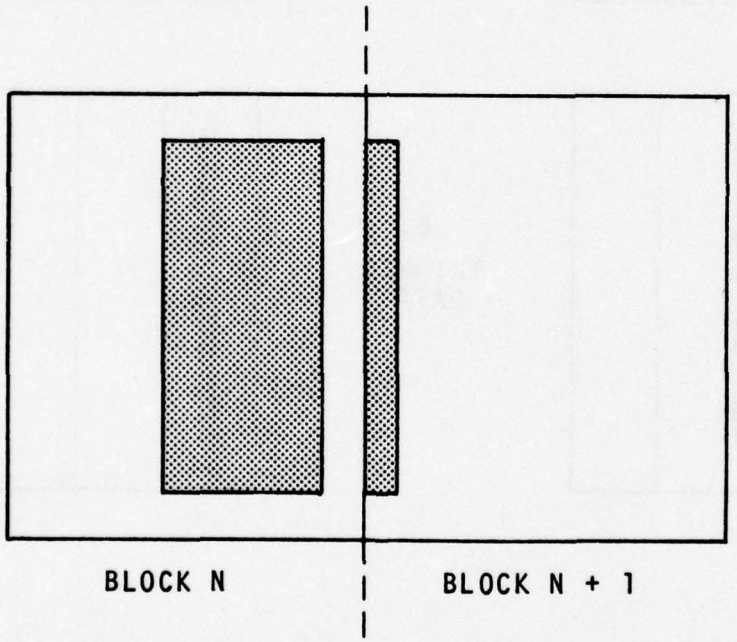


Figure 3-3. Effect of Overlap on Line Symbolization
(sheet 1 of 2)





D. RESULT AFTER ARRAY BOUNDARY MATCHING

Figure 3-3. Effect of Overlap on Line Symbolization
(sheet 2 of 2)

For this reason, the software was modified to allow the overlap width to be a variable and to be selected prior to processing. Therefore, the user selects the overlap width depending upon resolution, width of symbol to be generated, and maximum thickness of input line data. This parameter is input as a parameter in the Fortran Preprocessing Routine. This added flexibility (variable) will minimize data processing. For example, when processing 4-mil and 2-mil data the overlap can be reduced by a factor of 2 and 4.

3.1.4 Disk Implementation

The last section gave a brief description of the general processing sequence.

Assuming a large enough data buffer area the sequence would be as follows:

1. Read a data block from the input tape to the data buffer,
2. Load the arrays from the data buffer,
3. Process the arrays,
4. Unload the processed data from the arrays to the data buffer,
5. Repeat steps 2 through 4 until the entire data block is processed,
6. Write the processed data block to the output tape, and
7. Repeat steps 1 through 6 until the entire image is processed.

However, in most cases, the amount of data in the data block exceeds the size of the data buffer. Following is a description of the basic disk implementation technique when the data block (in binary coded plot format) is larger than the data buffers.

1. Read as many data block records from the input tape that can be contained within the data buffer. (See Figure 3-4.)
2. Divide each tape record into N-array load parts, and create a disk record from the same section of all the tape records, as shown in Figure 3-5. Store each of these records on the disk leaving enough space for similar records from the other tape records.
3. Repeat steps 1 and 2 for the remaining tape records, filling in the spaces between each N-array disk load.
4. When all tape records have been reformatted, load one N-array load into the data buffer area (see Figure 3-6).
5. Load the arrays from the data buffer.
6. Process the arrays.
7. Unload the processed data from the arrays to the data buffer overwriting the input data.
8. Repeat steps 5 through 7 until all N-arrays are processed.
9. Unload the N-array load onto the disk.
10. Repeat steps 4 through 9 until all N-array loads are processed.
11. Generate output records in data buffer from disk data.
12. Output tape records.
13. Repeat steps 11 and 12 until all output records/data blocks written onto tape.
14. Repeat steps 1 through 14 for all remaining data blocks of the image.

Figure 3-4. 16-K Data Buffer Layout for 4-Mil, 2-Mil, and 1-Mil Resolution

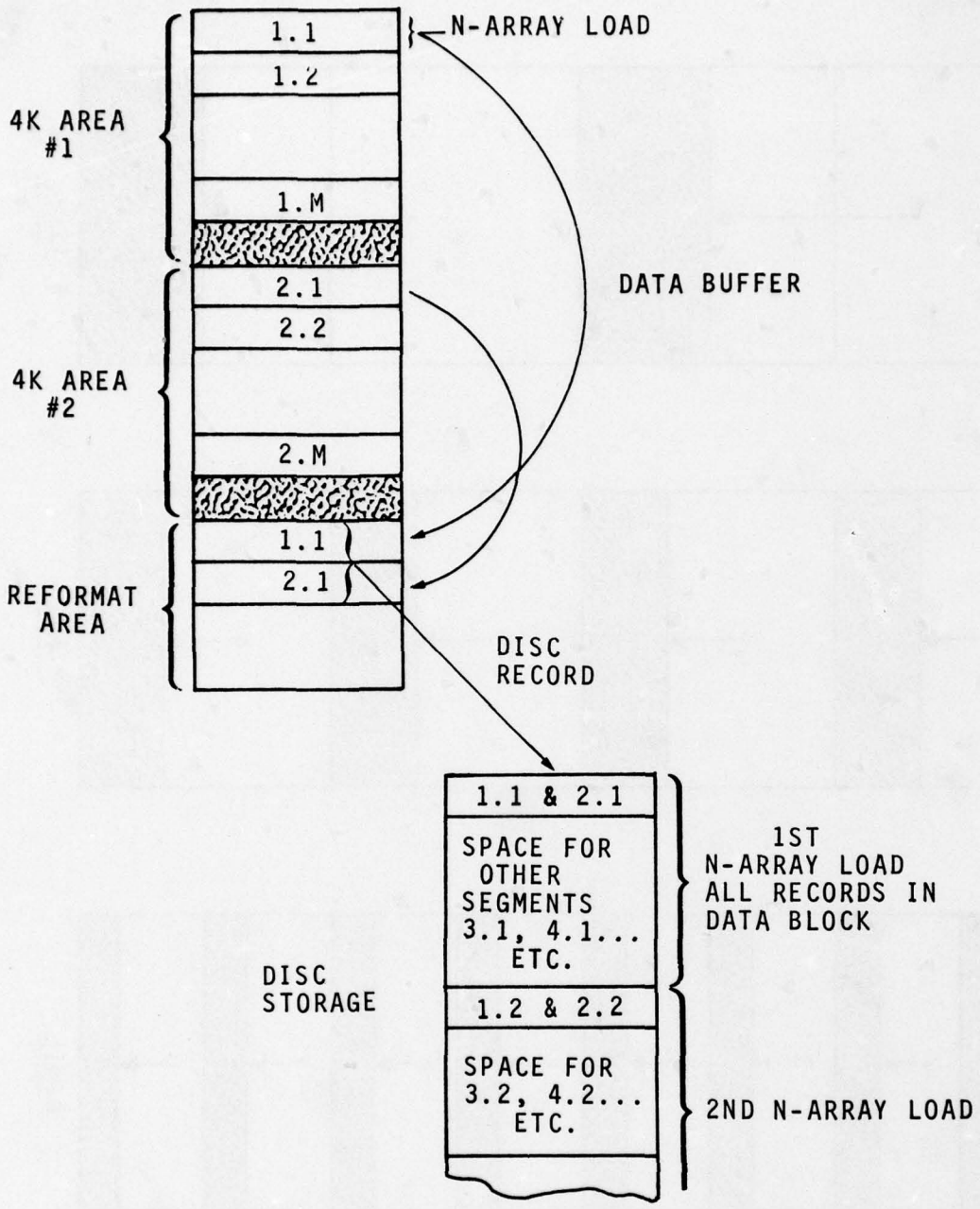


Figure 3-5. Tape - Disk Data Reformatting

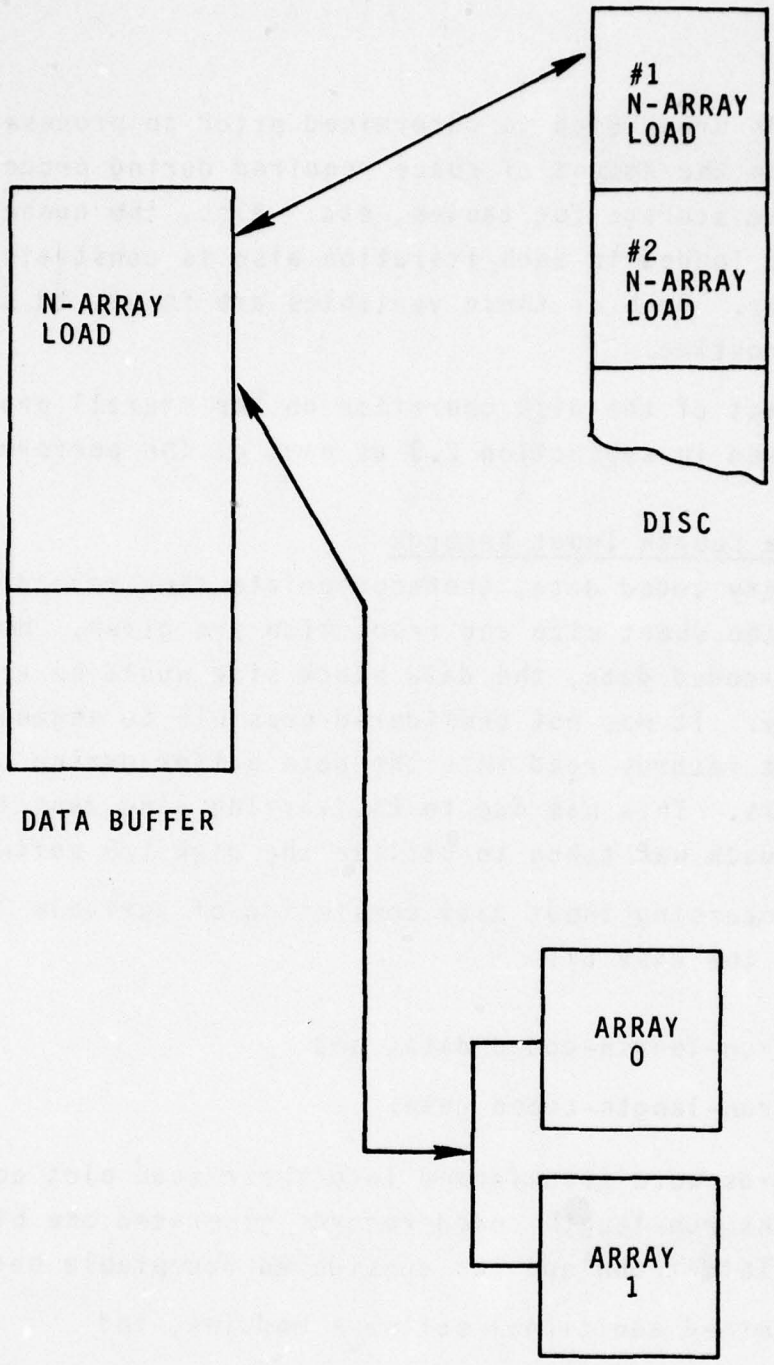


Figure 3-6. Disk-Data Buffer Loading (N-ARRAY Load)

The number of 4K areas used is determined prior to processing and depends upon the amount of space required during processing for intermediate storage for tables, etc. Also, the number N of arrays to be loaded in each iteration also is constrained in a similar manner. Both of these variables are inputs to the preprocessing routine.

The effect of the disk operation on the overall processing time is discussed in subsection 2.3 as part of the performance.

3.1.5 Variable Length Input Records

For binary coded data, the appropriate disk record size is known when the sheet size and resolution are given. However, for run-length-coded data, the data block size would be a function of line density. It was not considered possible to segment the number of input records read into the data buffer during processing into fix numbers. This was due to the varying line density. The following approach was taken to utilize the disk I/O software.

When processing input data consisting of variable length records, as in the case of:

- ETL run-length-coded data, and
- DMA run-length-coded data.

The input records were transformed into their scan plot equivalents. Therefore, eight run-length-coded records generated one binary scan plot record. This technique was considered acceptable because it:

- Minimized additional software modules, and
- Could be used to overlap certain procedures such as reading the variable length records while reformatting others to the plot format.

3.2 FORTRAN PREPROCESSING ROUTINE

3.2.1 Function

This routine reads input parameters from cards and generates a collection of data items required by the Executive to control the raster processing software.

3.2.2 Approach

The approach was to generate a Fortran routine (PRPRC) which was executed prior to processing and which would expand as the procedures themselves expanded.

The preprocessing routine performs the following steps:

1. Reads the input card parameters.
2. Computes the array parameters for subsequent array processing.
3. Computes number of data blocks/sheet.
4. Computes the number of iterations/data block.
5. Computes the number of 32-bit words in a plot tape record format.
6. Compute all the disk I/O parameters.
7. Computes the number of cells for line symbol generation processing.
8. Identifies what process is to be performed and generates the keyboard requests for the operator to load the specific I/O and Processing modules from the system disk DKO.

The card input parameters are shown below.

Card Number	Card Columns 1 thru 8	Parameter
1	Integer Value Right Justified	Resolution
2		Number of Arrays Used
3		Number Tape Records/Sheet
4		Number of Bytes/Plot Record
5		Symbol Type*
6		Line Symbol Thickness to be Generated
7		Number of Plot Records in Array Overlap
8		Number of 4096 Word Buffers Used
9		Input/Output Type**
10		Line Editing Thickness

*Symbol Type to be generated

Value	Type
0-19	Line Symbol Using Vectorizing
0	Intermittent Streams
1	Second Class Roads
2	Trails
20-39	Area Fill
40-59	Line Symbol (Solid)
60-79	Point Symbols
80-99	Railroad Ticks
100-119	Line Separation

**Input/Output Tape Format Type

0	ETL Plot/Plot Format
1	ETL RLC/Plot Format
2	DMA RLC/RLC Format
3	DMA RLC Input/ETL Plot Output Format

In addition to these 10 cards, Line Separation processing requires two additional parameters (see Line Separation) which define:

a) A flag (1 or 0) to indicate if the fattest line is required (1 = yes, Ø = no), and

b) The line weight desired.

3.3 EXECUTIVE

3.3.1 Function

The executive routine is the controlling program, all Raster Processing routines including I/O routines are initiated by this program.

3.3.2 Approach

The approach was to utilize the existing routine written for processing 4-mil data. (See Report ETL-0046 Associative Array Processing of Raster Scanned Data for Automated Cartography.) This routine was expanded to allow the processing of varying resolution data.

3.3.3 Implementation

The executive routine (EXEC) is written in APPLE and executes from Bulk Core Memory.

The Executive routine has been designed around a hierarchical structure whereby,all routines are placed on one of several level categories. These levels are directly associated with the general processing sequence discussed earlier in this section.

A brief description of each of the four levels is given below.

Level	Function	Examples of routines
0	These routines are executed once prior to/after performing an overall procedure on an overlay sheet	CELLSET: sets Line Symbol Generation parameters PREPRO: Performs Point Symbol pre-processing MVECT: Generates master-vectors for subsequent Broken Line Symbology
1	These routines are executed for each data block of information	TPRDF: Read a data block of Tape I/O data SETPTBT: Set the point bits in a data block of vectors. BLKPRO: Do point symbol processing for a data block of points
2	These routines input/output reformatted data between the disk and a Bulk Core data buffer	D2BI: Send I/P data to buffer for subsequent processing B2DO: O/P processed data to disk for subsequent output
3	These routines perform array I/O and all processing of data within the arrays on an iteration basis	LDA1: Load arrays with I/P data THIN1: Perform Line Thinning on array data AFIL1: Perform Area Fill.

All level 3 routines are executed from page memory (page 1 or page 2).

The main steps involved in the Executive routine are given below:

1. Read the input parameters generated by the Fortran preprocessing routine PRPRC.
2. Modify and save the parameters in the High Speed Data Buffer (HSDB) areas.
3. Select all Level 0-3 routines required for this pass.
4. Perform all data processing for this pass for all data on the overlay sheet.

5. Repeat steps 3. and 4. until all passes have been performed.
6. Terminate processing.

These steps are now described in more detail.

The input parameters generated by PRPRC are read from the disk file FOR002.DAT generated by PRPRC and located on the system disk DK0.ky calling and executing the subroutine RDISK.

This subroutine reads a multi-record file into the High Speed Data Buffer starting at location 608_{16}.

Figure 3-7 shows the location of the parameters in HSDB after executing RDISK.

The variables BFADRFAD, TRECAD(I), I=1,8 and ARECLOC(J) J=1,28 are all computed in the PDP-11 using Fortran single precision integer arithmetic. For example, where the buffer address should be $B000_{16}$ and the Reformat area base address should be $F000_{16}$, they are 3000_{16} and 7000 respectively. Therefore, the Executive unconditionally sets bit locations 0 and 16 of all the above 32-bit values.

The executive then stores the 16-bit values TRECAD(I) I=1,8 in a 32-bit word table RECADR(I), I=1,8. The executive also stores the 16-bit values ARECLDC(I) I=1,28 in <u>both</u> 16-bit halfwords of the array record table RESETLOC(I) I=1,28. Finally, the Base Disk Block numbers DADDR(I) I=1,36 are saved as 32-bit values in the table B2D(I) I=1,36.

These move/modifications are shown in Figure 3-8. The three labels RECADR, LOC, B2D identify the names of the three tables by which the data are later accessed. All the remaining disk I/O and array parameters are now generated from the other preprocessing parameters.

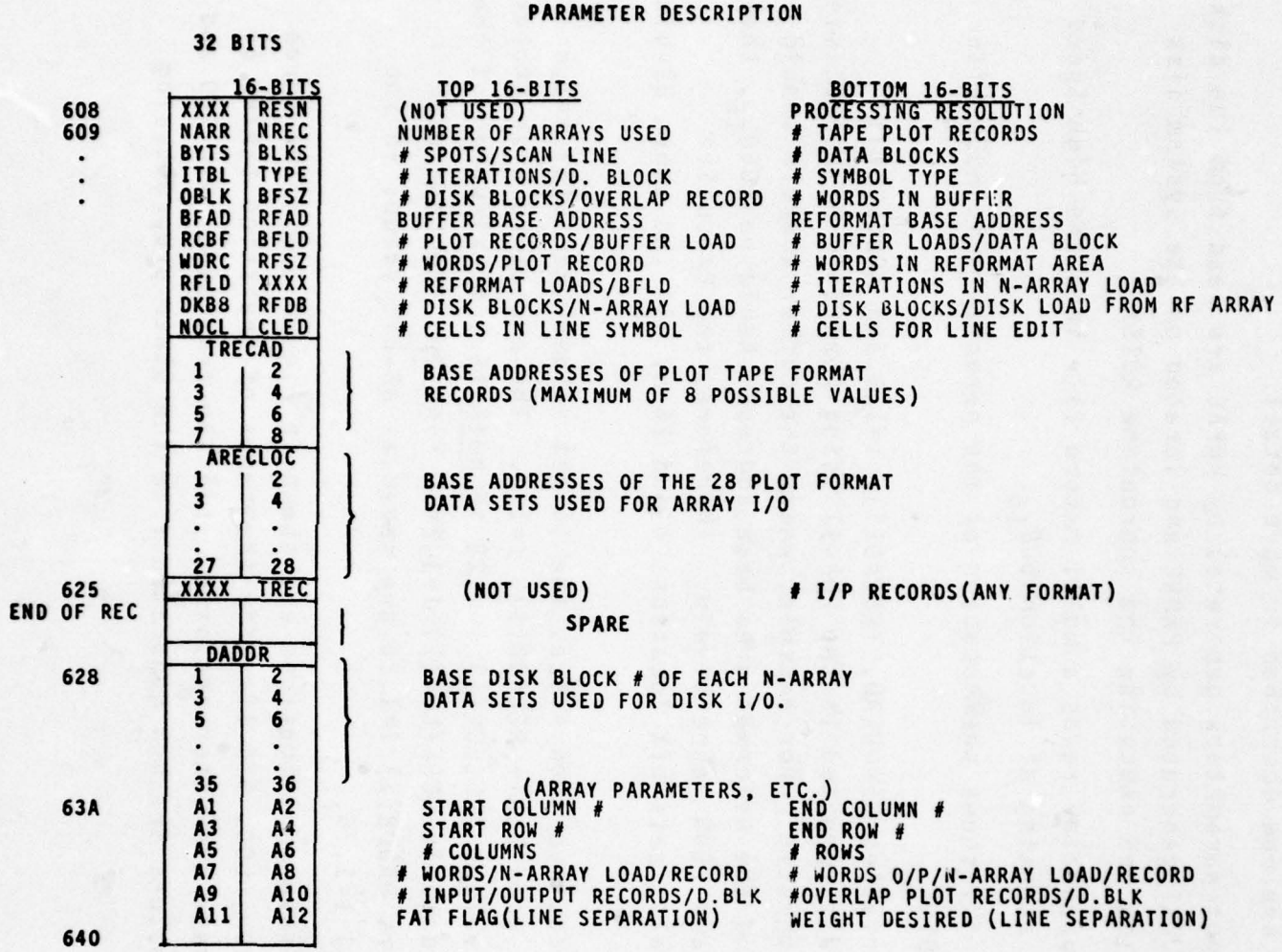


Figure 3-7. HSDB Preprocess Parameters

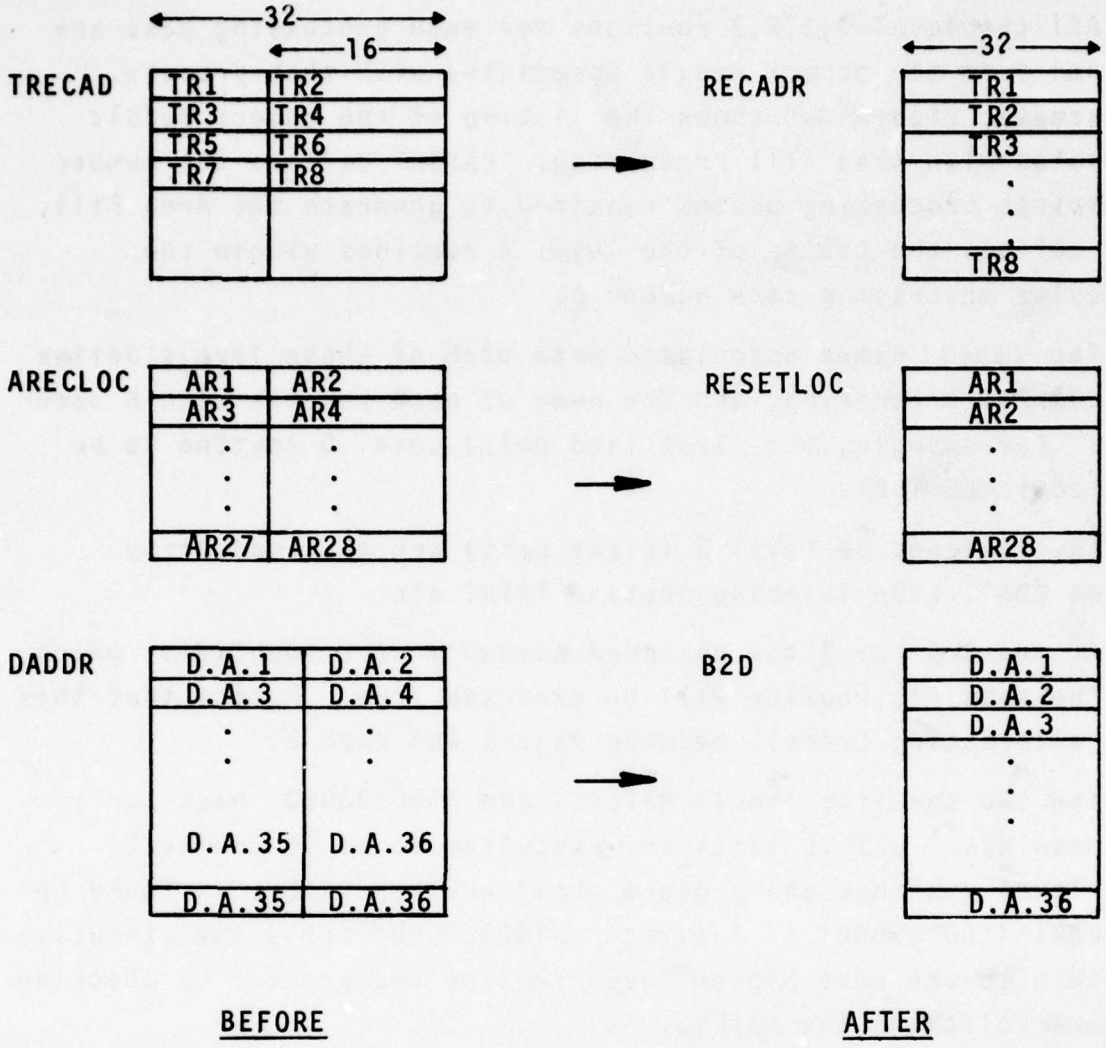


Figure 3-8. Table Modification/Move for Executive Use

All the level 0,1,2,3 routines for each processing pass are obtained from the object module associated with that process. For example, Figure 3-9 shows the listing of the object module associated with Area Fill Processing. PASSCT defines the number of complete processing passes required to generate the Area Fill. LEV*nm* defines the tables of the level *m* routines within the particular processing pass number *n*.

The label names associated with each of these levels define the order of processing, and the name of each routine within each level. For example, the first (and only) Level 0 routine to be executed is EDITSET.

The routines at Level 3 (first pass) are the Load Array Routine LDA1, Line Thinning Routine THIN2 etc.

At the 3rd Level the appended number 1 or 2 identifies which Page the specific routine will be executed from. Notice that this is an alternating process between Page 1 and Page 2.

The two specific labels NXTLVL, and X'80000000' have particular meanings. NXTLVL tells the executive to go to the next *lower* level routines and proceed until another NXTLVL is found or the label X'80000000' is located. X'80000000' tells the executive to return to the next higher level routine and proceed by checking the name following the NXTLVL.

Any working tables and buffer areas that may be required during processing are defined as subsequent labels within the module.

The level 0,1, and 2 routines are processed similarily by the Executive. However, the processing of the Level 3 routines requires further description.

The Level 3 routines are executed in order by alternating their loading/execution from Page 1 and Page 2. Page 0 is reserved for the execution of arithmetic subroutines.

```
                        ARFL        START
                                    EXTRN     THIN1,THIN2
                                    EXTRN     NXTLVL
                                    EXTRN     UDA2,THKED2,LDA1,GTL1
                                    EXTRN     SVL1
                                    EXTRN     AFIL2
                                    EXTRN     CLIP1,CLIP2
                                    ENTRY      GET,SAVE
                                    ENTRY     GETADR,SAVADR
                                    ENTRY      COMBF
                                    ENTRY     NOCELLM1
                                    ENTRY     LEV10,LEV20,LEV30
                                    ENTRY     PASSCT
                                    EXTRN     TPRDF,TPWF,B2DO,D2BI
                                    EXTRN     EDITSET
                                    ENTRY     PIPMAX
0000 0000 00010000  PASSCT          DC        X'00010000' # PASSES (TOP 16
0001 0001 00000000  LEV10           DC        EDITSET                   BITS)
0002 0002 00000000                  DC        NXTLVL
0003 0003 80000000                  DC         X'80000000'
0004 0004 00000000  LEV11           DC        TPRDF
0005 0005 00000000                  DC        NXTLVL
0006 0006 00000000                  DC        TPWF
0007 0007 80000000                  DC         X'80000000'
0008 0008 00000000  LEV12           DC        D2BI
0009 0009 00000000                  DC        NXTLVL
000A 000A 00000000                  DC        B2DO
000B 000B 80000000                  DC         X'80000000'
000C 000C 00000000  LEV13           DC        LDA1
000D 000D 00000000                  DC        THIN2
000E 000E 00000000                  DC        CLIP1
000F 000F 00000000                  DC        THKED2
0010 0010 00000000                  DC        THIN1
0011 0011 00000000                  DC        CLIP2
0012 0012 00000000                  DC        GTL1
0013 0013 00000000                  DC        AFIL2
0014 0014 00000000                  DC        SVL1
0015 0015 00000000                  DC        UDA2
0016 0016 80000000                  DC         X'80000000'
0017 0017 00000000  LEV20           DC        0
0018 0018 00000000  LEV30           DC        0
0019 0019 0000A400  GET             DC        X'A400'
001A 001A 0000A400  SAVE            DC        X'A400'
                9200  COMBF         EQU       X'9200'
                001B  NOCELLM1      DS,1
001C 001C 0000A400  GETADR          DC        X'A400'
001D 001D 0000A400  SAVADR          DC        X'A400'
                FFFF                END
```

DEFINITIONS OF TABLES, BUFFER AREAS, ETC.

Figure 3-9. Area Fill Processing Module

The actual operations involved in executing the Level 3 routines are as follows:

1. Initialize SUBPASS3.
2. Set the Array Input/Output buffer addresses RESETLOC(I), I=1,28 in a spare table LOC(I), I=1,28.
3. Load the first Level 3 routine into Page 1.
4. Initiate loading of the next routine into the second Page (Page 2 initially), and execute the other routine.
5. After the routine is executed, repeat step 4.
6. Repeat steps 4 and 5 until the marker X'80000000' (end of Level 3 routines) is found.
7. Execute the last routine.
8. Decrement SUBPASS3.
9. Repeat steps 2 through 8 until SUBPASS3 is zero.

There is another condition that prevents the Level 3 and Level 2 routines from exiting to the next higher level. The Level 2 routines Load/Unload a data buffer in Bulk Core Memory. The size of this area is determined by the amount of space required for tables etc., during processing. This area is equivalent to N array loads of data where:

N array loads = 2 x number of 4K buffer areas allocated

Therefore, for an N array load of data residing in buffer memory the number of subpasses at level 3 (SUBPASS3) to completely process this data will be $\frac{N}{\#\text{ arrays in system}}$, where both of these variables is some multiple of 2. Similarily the number of subpasses at level 2 (SUBPASS2) will be:

$$= \frac{\#\text{ vertical scan spots/record}}{\#\text{ vertical scan spots in N-array load}} .$$

This parameter was previously generated in the Preprocessing routine and is equivalent to the number of reformat loads/buffer load.

3.4 TAPE/DISK ROUTINES

3.4.1 Function

The function of the Tape/Disk routines are twofold.

i) To control the Tape Input/Output routines.

ii) To reformat the unprocessed/processed data and transfer the reformatted data between the reformat area in Bulk Core and the RK05 disk.

3.4.2 Approach

The approach was to write two routines to perform the two way data transfers and Input/Output requests.

3.4.3 Implementation

The two routines are written in APPLE and are executed as Level 1 routines out of Bulk Core. The reformatting subroutines are executed out of page memory to reduce execution time.

3.4.3.1 General - The routine that calls the tape input routine, and performs that input reformatting is called TPRDF. The tape output routine is called TPWF.

3.4.3.2 TPRDF - This routine controls the tape input routine and executes a subroutine called B2DI which reformats the input records and offloads them to the RK05 disk. TPRDF performs the following initial operations:

- Loads the B2DI routine into Page 1 for subsequent execution.
- Set the index pointer used by D2BI DISCINDXI to zero.
- Sets the initial disk block number at 4300_{10}. The disk blocks following will contain the overlap records which will be the first records of the next data block (BSDBLK).
- Sets the starting disk block numbers for the multiple array loads/unloads during the DISK-BUFFER I/O (BLKB2D).
- Initializes the number of plot records in each buffer load (RECBF).
- Sets the number of overlap records between each data block (OVLP).

- Initializes the pointer (SRECAD) to the table - (RECADR) which contains the Bulk Core Memory base addresses of where the Tape Input records will be loaded.
- Sets the number of unique input plot records/data block load (IRCM4).
- Sets the total number of input plot records/data block, including the overlap records (IPRBLK).
- Sets the total number of plot format records expected to be input or generated for the overlay sheet (NREC).
- Sets a variable (EOFCK) during the first and the last data blocks.

    EOFCK = ½ of number of overlap records for first block.
    EOFCK = 300 for last block.

For the first data block EOFCK is used as a decrementable count to generate blank records in the first half of the overlap area. For the last data block EOFCK is used as a decrementable count to generate blank records to fill up the last data block for processing.

The basic sequence of operations in TPRDF are as follows:

1. Call the routine to read/generate one plot tape format record.
2. Repeat step 1 until buffer is full; i.e., RECBUF=0.
3. Reformat a section (multiple array load) of the tape records via reformat area to the disk.
4. Repeat step 3 until all tape records completely off-loaded to the disk.
5. Repeat steps 1-4 until all input records for a data block have been read; i.e., IRCM4=0.

Steps 3 and 4 are the subroutine B2DI which will be discussed subsequently.

Routine TPRDF also executes to subroutines which save the overlap records (SAVREC) and retrieves them at the start of the next data block (GETREC). A set of disk blocks are reserved for these overlap records, the initial disk block being defined as 4300_{16}.

The subroutine B2DI is called by TPROF and reformats the tape input data onto the RK05 disk for subsequent load by the Disk-to-Buffer Input (D2BI) routine described in subsection 3.6. The B2DI subroutine transfers the tape buffer data to disk in N-array loads (N is a multiple of 4). This subroutine is a Level 1 subroutine of TPRDF, but is executed out of Pages to increase efficiency. The steps involved in B2DI are as follows:

1. Perform the following initialization.
   Generate the table RECADRL from the table of the tape record base addresses RECADR.
   Set up the TRANBLOK to point to the REFORMAT AREA and for output to the disk.
   Set the number of N-array loads (N8ARYL).
   Set the disk block index pointer to zero (DISCINDX).
   Initialize the record address table index to zero.
2. Get the number of tape records/buffer load (NRECBL).
3. Get the next record address from RECADRL.
4. Transfer the number of 32-bit words corresponding to N-array loads (of one record) to the reformat area.
5. Decrement the pointer to the last word transferred from the tape record by the number of words in the overlap area.
6. Save this pointer in RECADRL.
7. Repeat steps 3 through 6 until NRECBL=0.
8. Get the base disk block number from BLKB2D (DP) where DP is the index pointer DISCINDX. Set the disk block number in TRANBLOK.

9. Output the Reformat Area to the disk.
10. Update the base disk block number by the number of disk blocks just transferred (NBLKTRN).
11. Update the base disk block table index.
12. Repeat steps 2 through 11 until all N-array loads have been reformatted and output to the disk (N8ARYL=0).
13. Return to TPRDF.

3.4.3.3 TPWF - This routine controls the tape output routine and executes a subroutine called D2BO which reformats the processed data into output (plot format) tape records from the RK05 disk.

TPWF performs the following initialization operations:

- Sets the index pointer DISCINDXO to zero. This pointer is used by B2DO to index the buffer to disk table which contains the base disk block numbers which will contain the output data for reformatting (B2D).
- Initiates the movement of the Disk-to-Buffer Output routine to Pages.
- Sets the table B2D in secondary storage defined as BLKB2D.
- Sets the number of output plot records/data block (RECBLK).

TPWF then performs the following steps:

1. Sets the number of output records/buffer load.
2. Sets the index pointer SRECAD to the table of output record base addresses in RECADR.
3. Performs the subroutine D2BO.
4. Calls the routine to generate/write one the equivalent of one plot format record.
5. Performs step 4 until records/buffer is zero.
6. Repeats steps 1 through 5 until RECBLK is zero.
7. Exits Routine.

The subroutine D2BO is called by TPWF and reformats the output processed data from the Buffer-to-Disk-Output routine B2DO

described in section 3.6. The D2BO subroutine transfers the processed data from the disk to the reformat area in N-array loads. This subroutine as a Level 1 subroutine of TPWF which is executed out of Pages to increase efficiency. The steps involved in D2BO are as follows:

1. Perform the following initialization.
   - Generate the table RECARDR1 from the table of the tape record base addresses RECADR.
   - Set up the TRANBLOK to point to the REFORMAT Area and for input from the disk.
   - Set the disk block index pointer to zero (DISCINDX2).
   - Set the number of N-array loads (N8ARYL).
2. Set the number of plot tape records/buffer load (NRECBL).
3. Set the initial disk block number in the TRANBLOK for the disk output from the disk to buffer block table BLKD2B, using the index pointer DISCINDX2.
4. Transfer from disk to reformat area the disk blocks corresponding to the processed data to be reformatted for output.
5. Update the BLKD2B disk block number.
6. Reformat the data into the partial tape record.
7. Update the start address of the RECADR1 table for the next reformatting.
8. Repeat steps 5 through 7 until NRECBL is zero.
9. Repeat steps 2 through 8 until all of the N-array loads have been output and the tape records are generated.
10. Return to TPWF.

3.5 TAPE I/O ROUTINES

3.5.1 Function

The function of these routines is to perform the transfer of the input data and the processed output data between the tape devices and the Bulk Core buffer area.

3.5.2 Approach

The approach was to write several APPLE routines to allow the input/output of raster data in several formats. These routines are linked directly to the Tape/Disk routines. The different input formats are:

1. E.T.L run-length-coded scanned data.
2. E.T.L binary plot format data.
3. D.M.A run-length-coded scanned data.

The output formats are:

1. E.T.L binary plot format data.
2. D.M.A run-lenght-coded plot data.

3.5.3 Implementation

The Tape I/O routines are written in APPLE and executed at Level 1 as subroutines of TPRDF, and TPWF, the Tape/Disk routines.

3.5.3.1 General - The Tape I/O routines available are the following:

- ETLIO1 - This routine reads ETL binary-plot format-data, and writes ETL binary-plot format data.
- ETLIO2 - This routine reads ETL run-length-coded format data, and writes ETL binary-plot format data.
- DMAIO - This routine reads DMA run-length-coded format data, and writes DMA run-length-coded format data.
- DMAETL - This routine reads DMA run-length-coded format data, and writes ETL binary-plot format data.

Any one of these I/O subroutines may be selected prior to execution of the preprocessing routine by the selection on the IOTYPE parameters card discussed in subsection 3.2. After execution of the preprocessing routines will come a command to load the selection I/O routine prior to load and executing the particuliar processing routines. Each of the I/O routines is loaded at location 8200_{16} in Bulk Core Memory. The first instruction check as switch (Data Pointer Register) which is set by the calling Tape/Disk routine. to select either the input or the output part of the process. Figure 3-10 shows the top-level flow of the I/O routines.

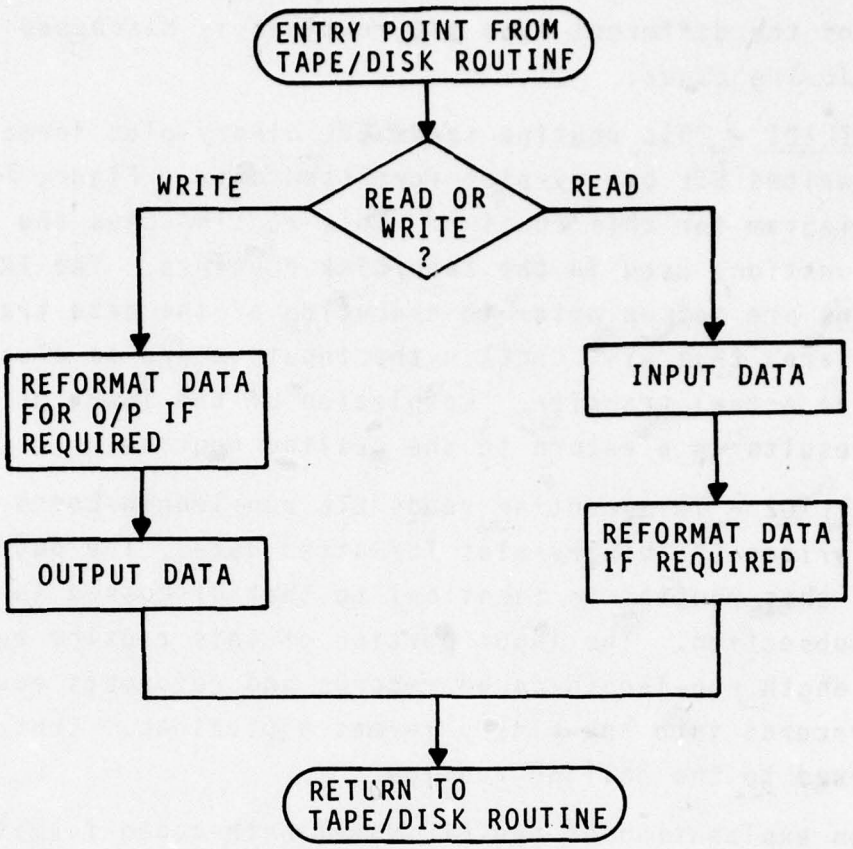


Figure 3-10. Top Level Flow of Tape I/O Routines

Each of the different Tape I/O routines is discussed in detail in the following pages.

3.5.3.2 ETLIO1 - This routine reads ETL binary-plot formatted data, and writes ETL binary-plot formatted data. Figure 3-11 shows the flow diagram for this routine. This routine uses the direct transfer functions used in the Tape/Disk routines. The TRANBLOK instructions are set up prior to execution of the data transfer. The buffer area that will contain the input record is cleared ahead of the actual transfer. Completion of the input or output transfer results in a return to the calling routine.

3.5.3.3 ETLIO2 - This routine reads ETL run-length-coded (RLC) data, and writes ETL binary-plot formatted data. The output portion of this routine is identical to that discussed in the previous subsection. The input portion of this routine reads in variable length run-length-coded records and reformats each group of eight records into the binary format equivalent. Control is then returned to the calling routine.

For an explanation of the ETL run-length-coded format the reader is referenced to Appendix A of this report. Figure 3-12 shows the top-level flow diagram for inputting ETL RLC data.

3.5.3.3.1 General - Basically eight RLC input records (scan lines) are required to generate one binary plot record. These records are alternately input to one of two buffers. The first buffer starts at location $F000_{16}$, and the second buffer starts at location $F800_{16}$. Each buffer area is 2048 words. After the first of the eight records has been input the reformatting of each record into the binary plot format is performed while the next record is being transferred. This 'double buffering' procedure absorbs much of the reformatting time.

After each set of eight records has been input and reformatted to generate one plot record, control is returned to the calling routine (TPRDF).

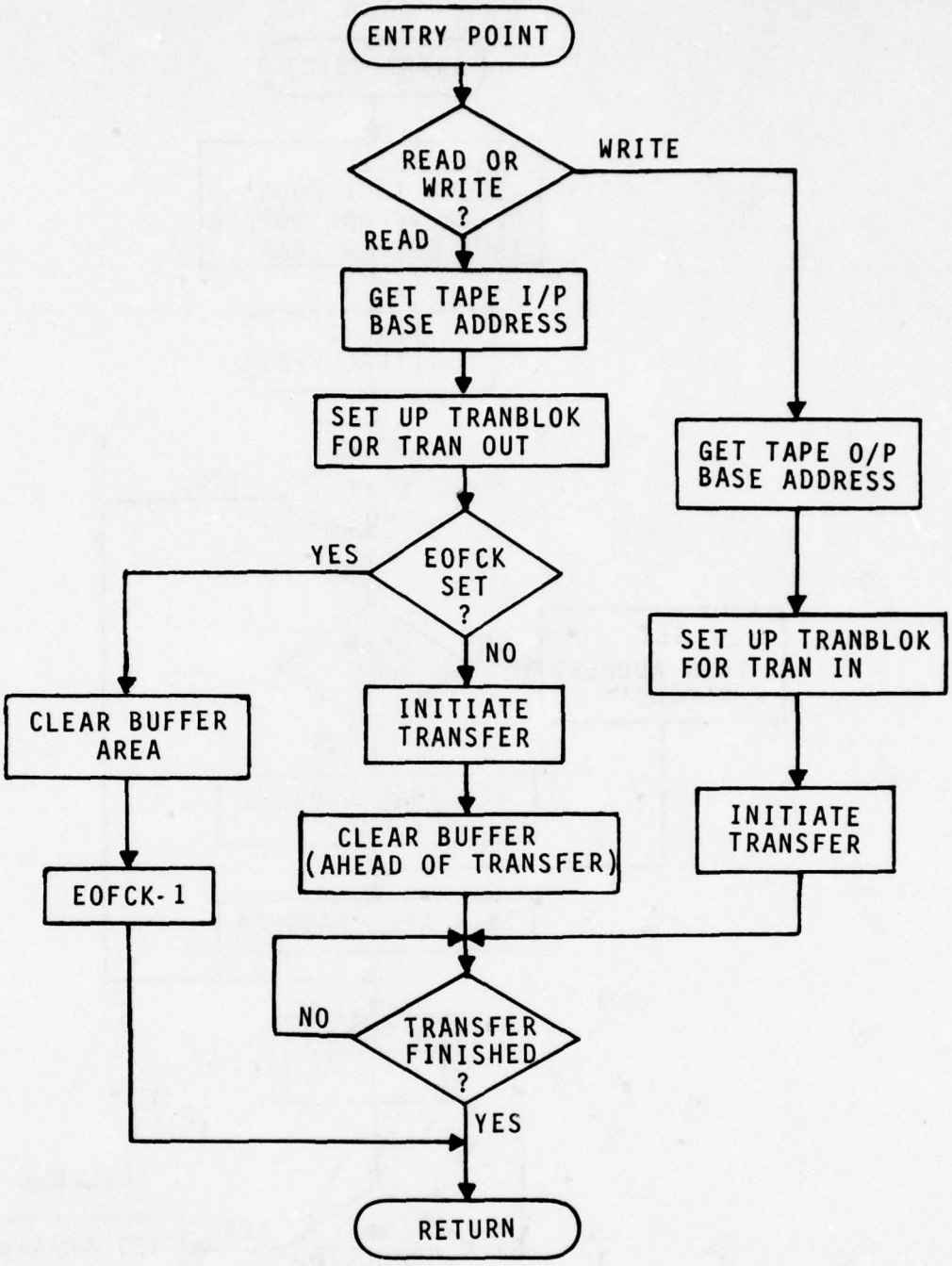


Figure 3-11. ETLI01 Flow Diagram

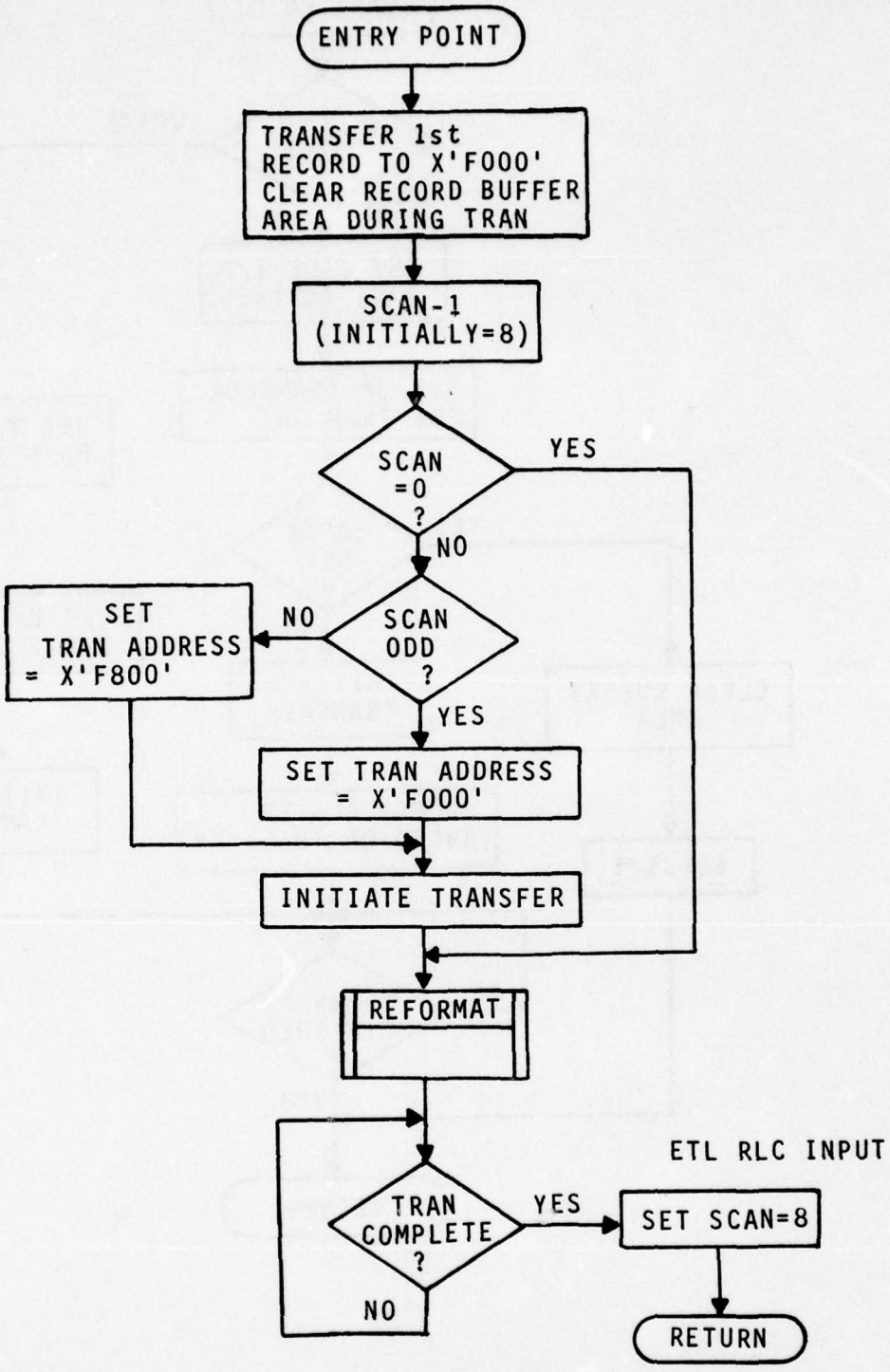


Figure 3-12. ETL R.L.C. Input Top-Level Flow Diagram

3.5.3.3.2 Reformatting - Figure 3-13 shows the flow diagram for the reformatting of the RLC data to binary plot data. Essentially, the transitions from OFF to ON (Word1, Byte1), and from ON to OFF (Word2, Byte2) are computed. The bit within Byte2 of the second transition word WORD2 is then set. This bit depends upon the scan number (0,1,2...7) currently being input. The same bit within all bytes starting with WORD1, BYTE1 are then set in a sequential manner until this same bit is located. The process is then repeated for all pairs of transitions.

3.5.3.4 DMAIO

3.5.3.4.1 Function - The function of the DMA I/O routines is to convert DMA run-length-coded data to binary for processing and to convert processed binary data back to run-length.

3.5.3.4.2 Approach - Because I/O is an area most likely to change in a production system, the approach was to convert the data to and from binary plot tape format and then utilize existing tape and disk I/O routines.

3.5.3.4.3 Implementation - There are two basic routines involved in the conversion of DMA run length coded (RLC) data. DMARLI converts RLC data to binary data. DMARLO converts binary data to RLC data. The format of the DMA-RLC data is given in Appendix A.

The implementation of these routines are discussed next.

The salient tasks of the SMARLI routine are outlined below:

1. Transfer (TRAN) in 1 RLC record. (Eight RLC records are required to create one binary record.)
2. Using the HI WC, LO WC pair, load the field, RLC, with the run-length values. Load until all values have been loaded, or until there is no room left in the array.
3. Call VLSUM$ to calculate running sums for each array.
4. Add last sum of each array to all sum values in succeeding arrays.

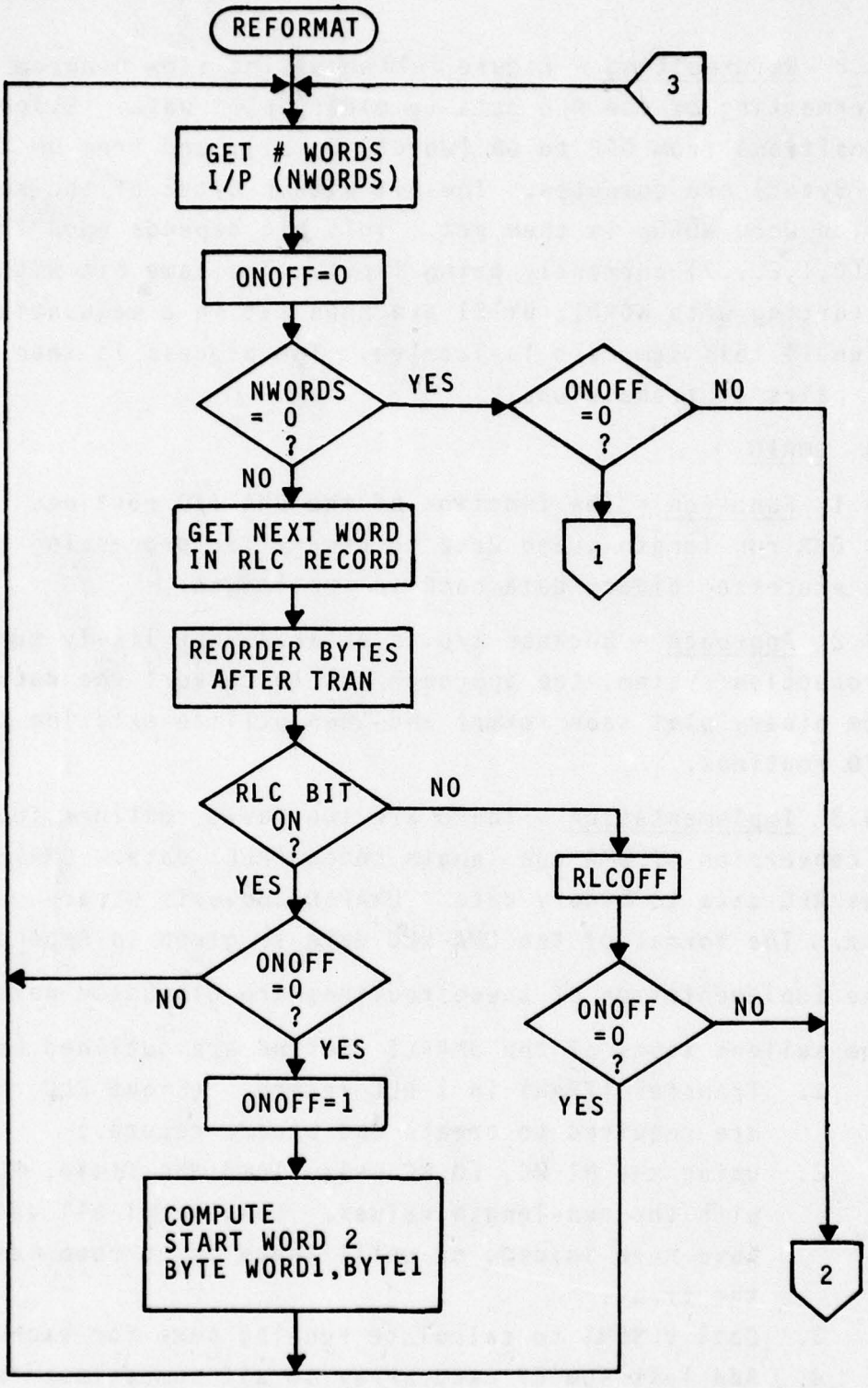


Figure 3-13. Reformatting (R.L.C. to Binary Plot) Flow Diagram (Sheet 1 of 2)

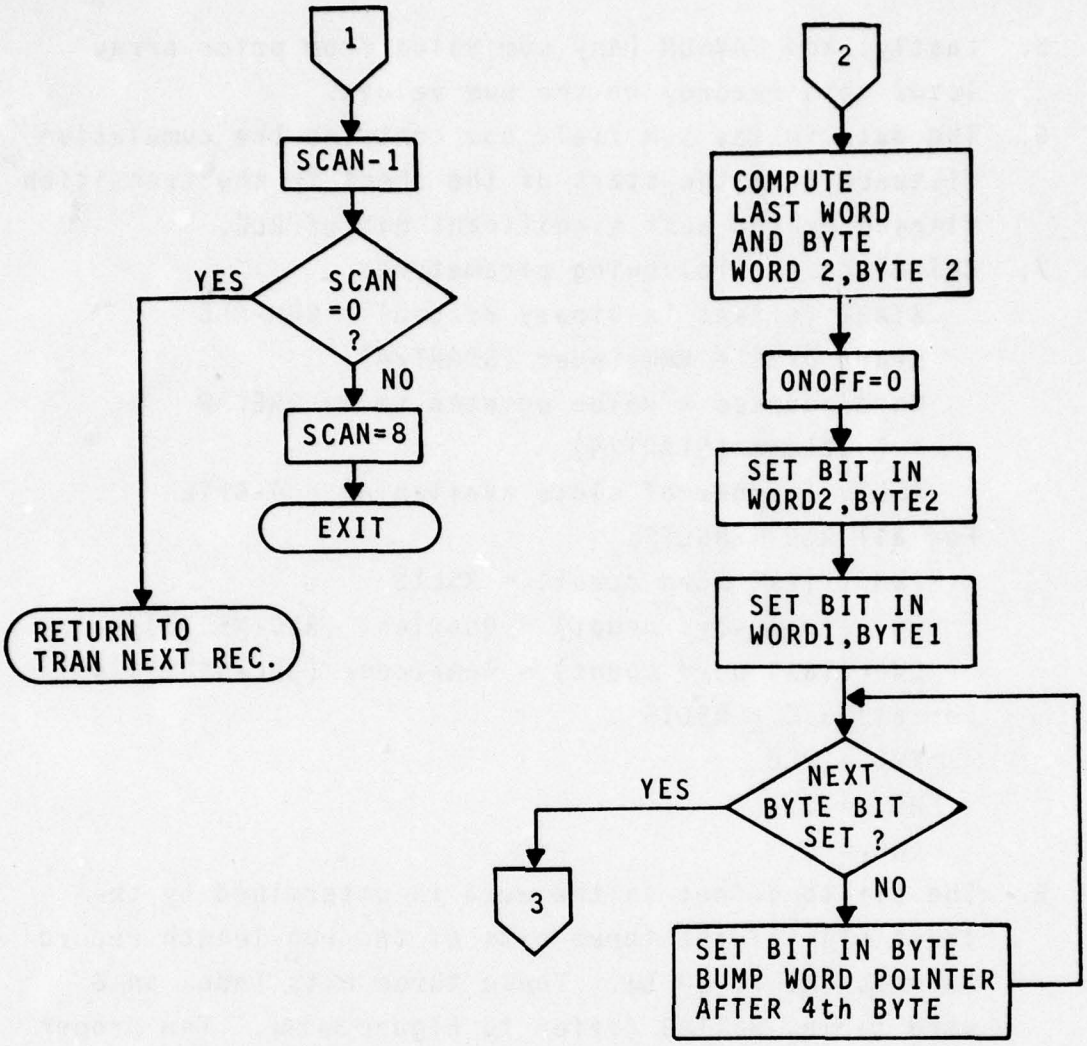


Figure 3-13. Reformatting (R.L.C. to Binary Plot) Flow Diagram (Sheet 2 of 2)

5. Lastly, add SAVSUM (any sum value from prior array loads this record) to the sum values.
6. The data in the SUM field now contains the cumulative distance from the start of the sheet to the transition flagged by the most significant bit of RLC.
7. Calculate the following parameters:

   START (*offset in Binary record*) = SUM-RLC

   start BYTE = Remainder (START/4)

   Word address = value pointed to by SRECAD + Quotient (START/4)

   NSLTS (number of slots available) = 4-BYTE

   For all RLC > NSLTS:

   FWC (first word count) = NSLTS

   WC (full word count) = Quotient (RLC-NSLTS)/4

   LWC (last word count) = Remainder (RLC-NSLTS)/4

   For all RLC $\leq$ NSLTS

   FWC = RLC

   WC = 0

   LWC = 0
8. The bit to be set in the word is *determined by* the least significant three bits of the run-length record address (HI Y, LO Y). These three bits index an 8 word table, BITTAB (refer to Figure 3-14). The proper pattern is selected and stored in BITPAT.

```
BITTAB   X'80808080'
         X'40404040'
         X'20202020'
         X'10101010'
         X'08080808'
         X'04040404'
         X'02020202'
         X'01010101'
```

If record number = 26_{10}
(Binary = 011010)

Then least significant 3 bits = 010

BITPAT = X'20202020'

Figure 3-14. BITTAB format

9. Once steps 7 and 8 have been completed, the proper bits are set in the binary record by calling SETNBITS and SETWORDS (described below).
10. When all bits have been set this arrayload, DONE flag is checked. If DONE≠1, steps 2-9 are repeated until DONE=1.
11. When DONE=1, the COUNT is checked to see if all 8 RLC records have been incorporated into the binary record. Steps 1-11 are repeated until COUNT=0.
12. The program then exits.

The SETNBITS routine is called when FWC or LWC are greater than zero. It sets the number of bits in a word as specified in the BL (FWC or LWC), starting at the byte pointed to by the DP (BYTE). The bytes of the F-register are initially all set. This is achieved by loading F with BITPAT. Depending upon the start byte and the number of bytes to be set, the corresponding 8-bit registers of the F-register are cleared. The contents of the F-register is then .OR.ed with the word from the binary record table (pointed to by ADR) and the resultant word stored back into the binary record table. ADR is incremented prior to the return.

The SETWORDS routine is called whenever WC is greater than zero. It gets WC words from the binary table, .OR.s them with BITPAT, and stores the results back into the binary table. ADR is bumped by WC prior to the return.

The salient tasks of the DMARLO routine are outlined below:

1. Upon entry, the program performs certain initialization tasks that must be done for each binary record. It also creates some flag bits which are required later on in the conversion process. The start address of the binary record and its length are stored in the location LNGADR.

2. Data are then loaded from bulk core bytewise into the array field BIN.
3. Eight run-length records are created from one binary. To avoid reloading the binary data 8 times, the run-length records are built and maintained in the array in fields RECn, n=1-8. To simplify the programming however, all run-length records are created and output from the same field, BLDAOPT. Prior to processing, the appropriate RECn field is moved to BLDAOPT. This field is moved back after processing.
4. Transition points are found by shifting the binary slice (Bits 0-7 of BIN) up one and exclusive .OR.ing the data with itself.
5. The black/white flag, RLC, is created by AND.ing the transition flag with the binary source slice. The run-length value (RLVAL) is then equal to the address of the current transition minus the address of the previous transition value.
6. The BLDREC is called to build a run-length record.
7. Steps 5 and 6 are repeated for all transition points in a slice.
8. Steps 3 thru 7 are repeated for each run-length record in the binary record.
9. Steps 2 thru 8 are repeated until the entire binary record has been converted. The subroutine CLEANUP is then called and the program returns.

The BLDREC adds the data being created in RLVAL to the record data. The set-up for the call to BLDREC is shown below.

CH=Quotient (RLVAL/128) with MSB=RLC
CL=Remainder (RLVAL/128) with MSB=RLC
BL=CH without RLC
DP=CL without RLC

If BL=0, RLVAL is less than or equal to 127 so only the contents of CL need be added to BLDAOPT. If BL≠0, the contents of CH must be added to the record along with BL-number of 7F or FF values. If DP=0 (i.e., RLVAL is an exact multiple of 128), CL need not be stored.

BLDREC must check prior to each store to insure that there is room in BLDAOPT. If there is no room, OPTREC is called to output BLDAOPT to the proper record buffer.

The OPTREC unloads the BLDAOPT field into a buffer in bulk core. Using BIN as a pointer, the address of the record is retrieved from RECPTR and the half word count is retrieved from BYTCT.

The OPTREC outputs BLDAOPT and counts the number of half words being stored. The common register is originally padded with 7F values to insure that an even number of run length values are stored away.

Once the data have been stored, RECPTR and BYTCT are returned to *memory*. BLDAOPT is cleared and the program returns.

The CLEANUP first insures that all data have been output, if not, OPTREC is called. The header data (HI WC, LO WC) (HI Y, LO Y) are then added to each record and the records are .TRAN.ed to tape.

3.6 DISK I/O ROUTINES

3.6.1 Function

The disk input/output routines transfer data between the RK05 disk and a Bulk Core Memory buffer.

3.6.2 Approach

The approach was to write two routines in APPLE to perform the two way transfers.

3.6.3 Implementation

The two routines are written in APPLE and are executed as Level 2 routines from Buld Core Memory.

3.6.3.1 General - The two routines transfer a block of arrays of data between the buffer and the disk. The number of arrays of data depends on the size of the buffer area being used, and is normally a multiple of four. The routine that transfers the unprocessed data from the disk to the buffer is called D2BI. The routine that transfers the processed data from the buffer to the disk is called B2DO.

3.6.3.2 Disk-to-Buffer Transfer - This routine transfers reformatted unprocessed data generated by the Tape/Disk Routines to a buffer in Bulk Core. The D2BI routine performs the following operation:

1. Sets up the TRANBLOK parameters associated with the data transfer, namely,
   i) Buffer Location,
   ii) Number of words to transfer,
   iii) Direction of transfer,
   iv) The starting disk sector number of the data block that is to be transferred
2. Performs the disk-to-buffer transfer, and
3. Exits the routine after completing the data transfer.

3.6.3.3 Buffer-to-Disk Transfer - This routine transfers processed data output from the arrays and located in the data buffer used by D2BI to the disk. The B2DO routine performs the following operations:

1. Sets up the TRANBLOK parameters associated with the data transfer, namely,
   i) Buffer location,
   ii) Number of words to transfer,
   iii) Direction of transfer,
   iv) The starting disk sector number for the data block being output,
2. Performs the buffer-to-disk transfer, and
3. Exits the routine after completing the data transfer.

3.7 ARRAY LOAD/UNLOAD ROUTINES

3.7.1 Function

The array load/unload routines transfers the preprocessed/processed raster data between Bulk Core Memory and the STARAN arrays.

3.7.2 Approach

The approach was to reunite those routines already written for Loading/unloading the arrays to take advantage of mix-mode operations and thus speed up this process.

3.7.3 Implementation

The modified array input/output routines are written in APPLE assembly language and machine code, and are executed as Level 3 routines from the Page memories.

3.7.3.1 General - These two routines load/unload the arrays from/to Bulk Core Memory. These routines utilize three subroutines, LDBYT$, ORDER$, and STBYT$ which were developed to efficiently transfer byte values between control memory and array modules. These subroutines are described in subsection 3.7.3.4.

3.7.3.2 Load Arrays - The load array routine loads the arrays with raster data. This data has been extracted from the reformatted data stored on the RK05 disk using the Disk-to-Buffer-Input (D2BI) Routine described in subsection 3.6. The operations required to load the arrays are as follows:

1. Perform initialization of:
   i) Number of arrays in system,
   ii) Number of 8-bit (byte) array fields to be loaded,
2. Get pointer to Bulk Core Memory containing the 64 words to be stored into the arrays,
3. Get array field pointer.
4. Load the array field using LDBTY$*

4. Load the array field using LDBYT$*,
5. Repeat step 4 for all arrays,
6. Save Bulk Core Memory pointer for next time,
7. Repeat steps 2 through 6 for all array fields,
8. Order the flipped data using ORDER$, and
9. Exit Routine.

*It should be noted that each 32-bit (4 byte) word in Bulk Core is stored in reverse order. Therefore, each four bytes of data is flipped to its correct location within each 32-bit section of the X-response store prior to loading into the arrays as shown in Figure 3-15.

3.7.3.3 Unload Arrays - The unload array routine unloads the processed array data. This data will later be stored on the RK05 disk using the Buffer-to-Disk-Output Routine (B2DO) described in subsection 3.6. The operations required to unload the arrays are as follows:

1. Perform initialization of;
   i) Number of arrays in system,
   ii) start column of data to O/P,
   iii) Number of words/8-bit field to output,
2. Move output data area to the top left edges of the arrays,
3. Determine number of output words/8-bit field,
4. Get next field pointer and Bulk Core Memory output area pointer,
5. Unload one array field of data from all arrays to Bulk Core Memory,
6. Save Bulk Core Memory output area pointer,
7. Repeat steps 4 thru 6 until all 8-bit fields have been output, and
8. Exit Routine.

3.7.3.4 LDBYT$, ORDER$, STBYT$ - These three subroutines perform efficient transfers of byte values between Bulk Core memory ad and the array modules. The area in memory is a block of 64 words,

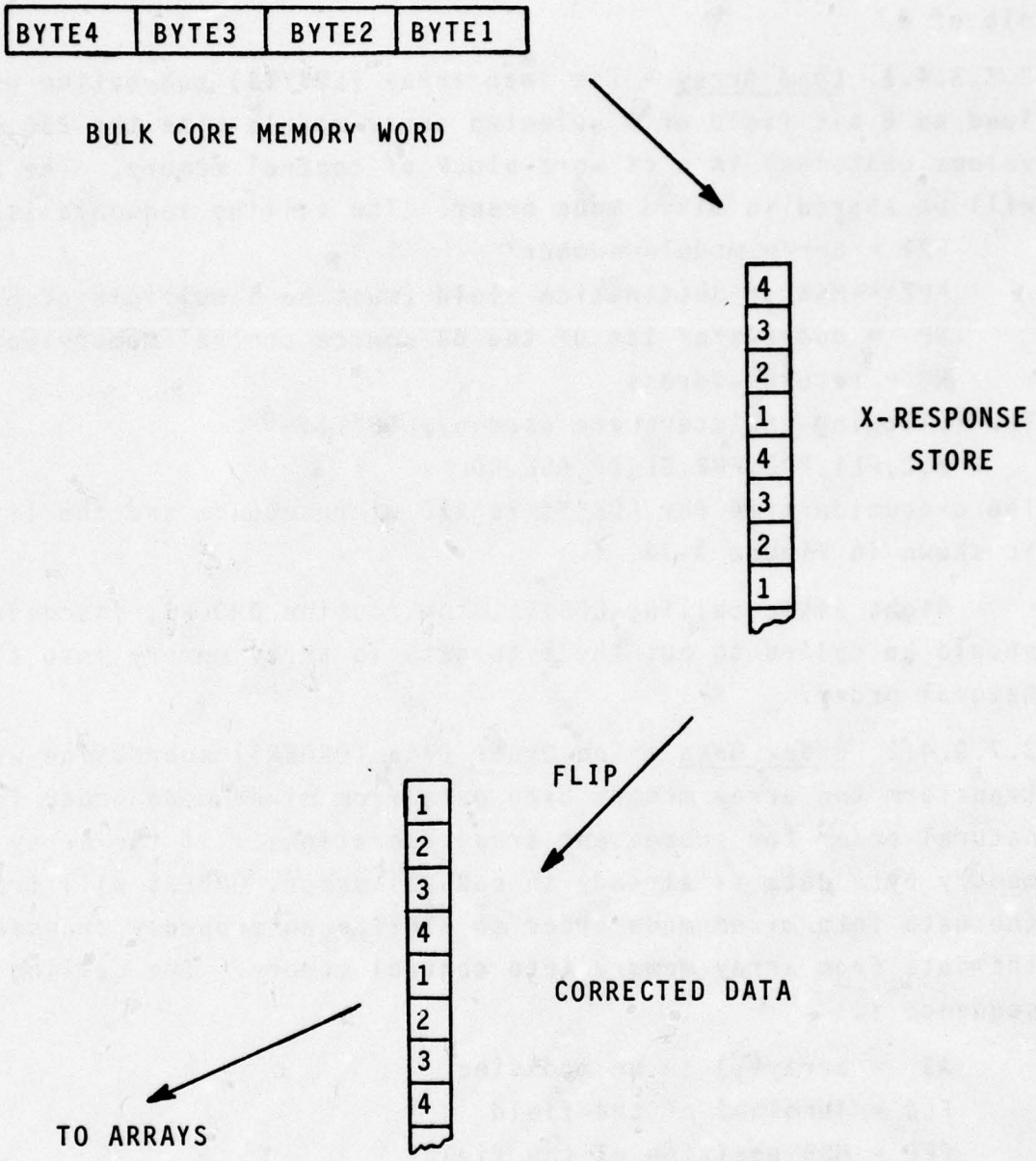


Figure 3-15. Data Correction During Array Loading

each containing four bytes or a total of 256 byte values. The most significant bit address of the array field must be a multiple of 8.

3.7.3.4.1 Load Array - The load array (LDBYT$) subroutine will load an 8 bit field of a selected array module with the 256 byte values contained in a 64 word block of control memory. The bytes will be stored in mixed mode order. The calling sequence is:

FP1 = array module number
FP2 = MSB of destination field (must be a multiple of 8)
DP = address of 1st of the 64 source control memory words
RO = return address

The following registers are used by LDBYT$:

X,C,FL1,FP1,FP2,BL,DP,ASL,RO

The execution time for LDBYT$ is 120 microseconds and the listing is shown in Figure 3-16.

Right after calling LDBYT$, the routine ORDER$, discussed next, should be called to put the byte data in array memory into the natural order.

3.7.3.4.2 Order Data - The Order Data (ORDER$) subroutine will transform the array memory byte data from mixed mode order into natural order for subsequent array operations. If the array memory byte data is already in natural order, ORDER$ will transform the data into mixed mode order so STBYT$ can properly transfer the data from array memory into control memory. The calling sequence is:

AS = array(s) to be modified
FL2 = length-1 of the field
FPE = MSB position of the field
RO = return address

The following registers are used by ORDER$:

X,M,C,FL1,FPE,FL1,FPE,FP1,RO,R1

```
                        LDBYT$    START
                        ;
                        ;LOADS 256 BYTES FROM 64 SUCCESSIVE CONTROL MEMORY WORDS
                        ;INTO AN 8-BIT FIELD OF ALL WORDS IN ONE ARRAY.
                        ;
                        ;ORDER OF BYTES IN ARRAY CAN BE FIXED WITH 'ORDER$' ROUTINE
                        ;
                        ;ENTRY:   FP1=      ARRAY NUMBER
                        ;         FP2=      M'(8-BIT FIELD) (MUST BE A MULTIPLE OF 8
                        ;         DP=       ADDRESS OF FIRST CONTROL MEMORY WORD
                        ;         BAL,R0    LDBYT$
                        ;
                        ;DISTURBS X,C,FL1,ASL,BL,FP2
                        ;         FP2 IS INCREMENTED BY 8
                        ;
                        ;TIMING:120 MICROSEC (470 NANOSEC/BYTE LOADED)
                        ;
                                  ENTRY     LDBYT$
0000 0000 34A00008 LDBYT$ LI        BL,8              GO THRU OUTER LOOP TWICE
0001 0001 30800009        LRR       ASL,DP            SAVE DP IN ASL
0002 0002 38014807        GEN,32    X'38014807'       PUT AMR1 IN BYTE MODE
0003 0003 3F030014 LDBYTS1 LOOP,4   LDBYTS2
0004 0004 360200V0        LR        C,0(DP)
0005 0005 01C088A0        SCW,X'88A0' X(0)
0006 0006 36020008        LR        C,8(DP)
0007 0007 01C199A0        SCW       X(1)
0008 0008 36020010        LR        C,16(DP)
0009 0009 01C299A0        SCW       X(2)
000A 000A 36020018        LR        C,24(DP)
000B 000B 01C399A0        SCW       X(3)
000C 000C 36020020        LR        C,32(DP)
000D 000D 01C499A0        SCW       X(4)
000E 000E 36020028        LR        C,40(DP)
000F 000F 01C599A0        SCW       X(5)
0010 0010 36040030        LR        C,48(DP),2        INCR DP
0011 0011 01C699A0        SCW       X(6)
0012 0012 36050037        LR        C,55(DP),3        INCR DP,DECR BL
0013 0013 41C799A0        SCW       X(7)
0014 0014 1FE00003 LDBYTS2 S,W      X,FP12+           STORE X;INCE FP2
0015 0015 3040000A        LRR       DP,ASL            RESTORE DP
0016 0016 29010019        BZ,BL     LDBYTS3           EXIT IF DONE
0017 0017 28140001        INCR      DP                ELSE INCR DP
0018 0018 28010003        B         LDBYTS1           AND REPEAT OUTER LOOP
0019 0019 380148FF LDBYTS3 GEN,32   X'380148FF'       RESTORE AMR1 TO WORD
001A 001A 28080000        B         0(R0)             RETURN TO CALLER
```

Figure 3-16. Load Array (LDBYT$) Listing

The execution time for ORDER$ is 1.25* (field length-1) + 13 microseconds and the listing is shown in Figure 3-17.

3.7.3.4.3 Dump Array - The Dump Array (STBYT$) subroutine will transfer an 8 bit field of array data to a block of 64 words of control memory. In other words, the opposite function is performed by LDBYT$. The address of the MSB of the data to be transferred must be a multiple of eight and the data must be in mixed mode order. The calling sequence is:

FP1 = array module number
FP2 = MSB of source field (must be a multiple of 8)
DP = address of the 1st of the 64 destination control memory words
RO = return address

The following registers are used by STBYT$:

X,C,FL1,FP1,FP2,BL,DP,ASL,RO

The execution time for STBYT$ is 134 microseconds and the listing is shown in Figure 3-18.

3.8 LINE THINNING ROUTINE

3.8.1 Function

The line thinning routine is an AAP subroutine which thins line data to a unit (one-all) thick line.

3.8.2 Approach

The approach was to utilize the existing routine, modified to automatically terminate thinning.

3.8.3 Implementation

The line thinning routine (THIN) is a Level 3 routine written in APPLE and executed in Page Memory.

3.8.3.1 General - The line thinning routine was modified to automatically terminate when all lines in the arrays were completely thinned (i.e., no more calls could be removed).

```
                        ORDER$    START
                        ;
                        ;CHANGES WORD ORDER OF A FIELD IN ALL ENABLED ARRAYS
                        ;BETWEEN NATURAL ORDER AND ORDER REQUIRED BY 'LDBYT$'
                        ;AND 'STBYT$' ROUTINES.
                        ;
                        ;USE 'ORDER$' AFTER 'LDBYT$' TO PUT INPUT BYTES IN
                        ;NATURAL ORDER.
                        ;
                        ;USE 'ORDER$' BEFORE 'STBYT$' SO OUTPUT BYTES WILL
                        ;BE IN NATURAL ORDER.
                        ;
                        ;ENTRY:   AS=       ARRAYS TO BE MODIFIED
                        ;         FL2=      L'(FIELD)
                        ;         FPE=      M'(FIELD)
                        ;         BAL,R0    ORDER$
                        ;
                        ;DISTURBS X,M,C,FL1,FL3,FP1,R1
                        ;
                        ;TIMING:1.25(L'(FIELD))+13  MICROSEC
                        ;
                                  ENTRY     ORDER$
0000 0000 742055AA ORDER$ LI        CH,X'55AA'          C=MASK 1
0001 0001 720055AA        LI        CL,X'55AA'
0002 0002 33900009        LI        FP1,X'09'           FLIP CODE (00001001)
0003 0003 2C110009        BAL,R1    ORDER1              CALL INNER ROUTINE
0004 0004 74203333        LI        CH,X'3333'          C=MASK2
0005 0005 7200CCCC        LI        CL,X'CCCC'
0006 0006 33900012        LI        FP1,X'12'           FLIP CODE (00010010)
0007 0007 2C110009        BAL,R1    ORDER1              CALL INNER ROUTINE
0008 0008 28080000        B         0(R0)               RETURN
                        ;
                        ;INNER ROUTINE - SETS UP MASK AND FLIPS FIELD BETWEEN
                        ;MASKED WORDS.
                        ;
0009 0009 420088A0 ORDER1 SC,X'88A0' X(0)             MASK FROM C
000A 000A 40E099B3        GENS,32   X'40E099B3'       REPLICATE INTO ALL X
000B 000B 40C099B3        GENS,32   X'40C099B3'
000C 000C 408099B3        GENS,32   X'408099B3'
000D 000D 08000003        L         M,X               M=MASK
000E 000E 31000017        LRR       (FL1,FP3),(FL2,FPE) FIELD PARAMETERS
000F 000F 3D000013        LOOP      ORDER2
0010 0010 438088A5        L         X,FP3             LOAD X WITH A BIT
0011 0011 430088A3        GENS,32   X'430088A3'       FLIP X USING FP1
0012 0012 4B800001        GENS,32   X'4B800001'       SETUP M
0013 0013 53A00003 ORDER2 GENS,32   X'53A00003'       STORE X MASKED;FP3+
0014 0014 00000001        NOP
0015 0015 28090000        B         0(R1)             RETURN TO OUTER ROUTE
```

Figure 3-17. Order Data (ORDER$) Listing

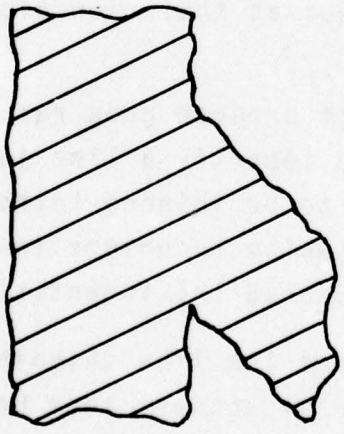
```
                        STBYT$    START
                        ;
                        ;STORES 256 BYTES FROM AN 8-BIT FIELD OF ALL WORDS IN ONE
                        ;ARRAY INTO 64 SUCCESSIVE CONTROL MEMORY WORDS.
                        ;
                        ;TO KEEP ORDER STRAIGHT FIX THE BYTES IN THE ARRAY WITH
                        ;THE 'ORDER$' ROUTINE BEFORE STORING WITH THIS ROUTINE.
                        ;
                        ;ENTRY:   FP1=      ARRAY NUMBER
                        ;         FP2=      M'(8-BIT FIELD) (MUST BE A MULTIPLE OF 8
                        ;         DP=       ADDRESS OF FIRST CONTROL MEMORY WORD
                        ;         BAL,R0    STBYT$
                        ;
                        ;DISTURBS X,C,FL1,ASL,BL,FP2
                        ;         FP2 IS INCREMENTED BY 8
                        ;
                        ;TIMING:134 MICROSEC (526 NANOSEC/BYTE STORED)
                        ;
                                  ENTRY     STBYT$
0000 0000 34A00008 STBYT$ LI        BL,8              GO THRU OUTER LOOP TWICE
0001 0001 30800009        LRR       ASL,DP            SAVE DP
0002 0002 38014807        GEN,32    X'38014807'       PUT AMR1 IN BYTE MODE
0003 0003 3F030014 STBYTS1 LOOP,4   STBYTS2
0004 0004 47E086A5        L,W       X,FP12+           LOAD X:INCR FP2
0005 0005 21C0A0FB        LCW       X(0)
0006 0006 30020C00        SR        C,0(DP)
0007 0007 21C1A0FB        LCW       X(1)
0008 0008 30020008        SR        C,8(DP)
0009 0009 21C2A0FB        LCW       X(2)
000A 000A 30020010        SR        C,16(DP)
000B 000B 21C3A0FB        LCW       X(3)
000C 000C 30020018        SR        C,24(DP)
000D 000D 21C4A0FB        LCW       X(4)
000E 000E 30020020        SR        C,32(DP)
000F 000F 21C5A0FB        LCW       X(5)
0010 0010 30020028        SR        C,40(DP)
0011 0011 21C6A0FB        LCW       X(6)
0012 0012 30040030        SR        C,48(DP),2        INCR DP
0013 0013 21C7A0FB        LCW       X(7)
0014 0014 30050037 STBYTS2 SR       C,55(DP),3        INCR DP,DECR BL
0015 0015 3040000A        LRR       DP,ASL            RESTORE DP
0016 0016 29010019        BZ,BL     STBYTS3           EXIT IF DONE
0017 0017 28140001        INCR      DP                ELSE INCR DP
0018 0018 28010003        B         STBYTS1           AND REPEAT OUTER LOOP
0019 0019 380148FF STBYTS3 GEN,32   X'380148FF'       RESTORE AMR1 TO WORD
001A 001A 28080000        B         0(R0)             RETURN TO CALLER
```

Figure 3-18. Dump Array (STBYT$) Listing

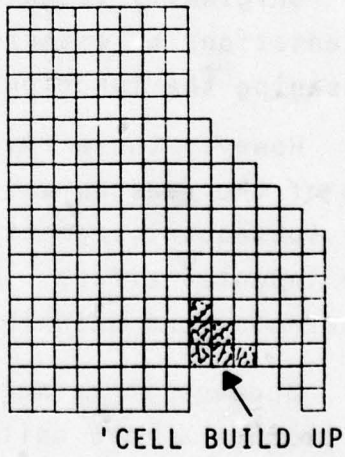
The line thinning algorithm is identical to that described in subsection 3.5 of the Final Technical Report titled, Associative Array Processing of Raster Scanned Data for Automated Cartography, ETL-0046. However, at the end of each pass through the line thinning algorithm (vertical or horizontal array sweep) a pointer CHNG is set to denote at least one cell was removed. At the end of four passes this pointer is checked and the THIN routine is exited if CHNG is zero.

The line thinning process thins in such a way as to produce some deformation at line junctions which causes indentations in the resulting line symbols at these points. This is noticeable at 4-mil resolution, but is much less apparent at 2-mil and 1-mil resolution. The reasoning is as follows.

A close examination of the digitized source data showed that some line junctions produce a 'build up' of cells after the scanning/digitizing process. This 'build up' appears to be related to the acuteness of the angle between the two line segments and is shown below.

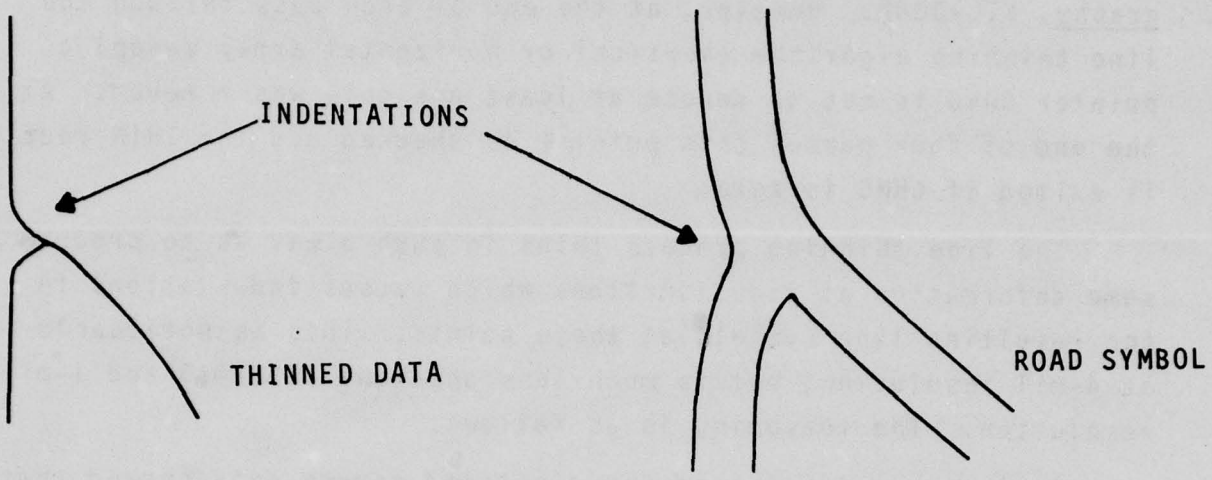


A. SOURCE DATA



B. SCANNED-REPLOTTED SOURCE

Because the line thinning algorithm reduces the line segments to the most central set of points the result becomes that shown below.



Generation of the correct line symbology from this data, in this case a double line, produces the indented effect visible at some junctions on the final results as shown in the following diagram.

Originally techniques were investigated to remove the indentations by either breaking the lines at their junctions or massaging the junction area.

However, these investigations didn't produce good results. One of the reasons was that in order to identify a line junction for subsequent segmenting the line had to be thinned introducing the indented effect. Secondly, the massaging technique involved thickening and thinning which again produced the indentations.

Because it is not possible to change the line thinning algorithm to produce a more uniform line junction it appears that better results are obtained by using one or more of the following techniques.

- Utilize thinner or more sharply defined source material,
- scan/digitize the source data at the higher resolutions, or
- adjust the threshold on the scanner/plotter to minimize the addition of cells at line junctions.

3.9 VARIABLE LINE THICKENING ROUTINES

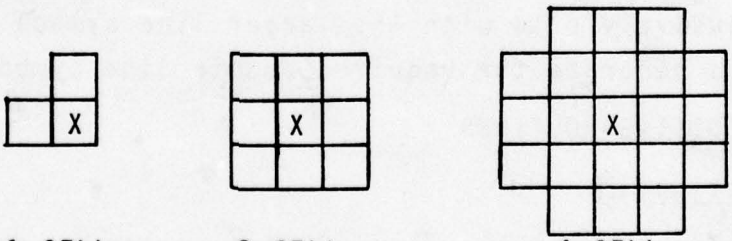
3.9.1 Function

These routines generate the different solid line symbols.

3.9.2 Approach

The approach was to generate two routines. The first routine (THK) performs a series of 4-cell, 2-cell, and 1-cell line thickness similar to the three routines described in subsection 3.3.7 Line Symbol Generation Routines of the Final Technical Report ETL-CR-74-20 Associative Array Processing for Topographic Data Reduction. The second routine (ELLSET) generates the input parameters to THK. The input parameters provide the number of passes required by each of the three thickening subroutines. These parameters are determined from the parameter card input to the Fortran preprocessing routine PRPRC which indicates the thickness in mils of the final line symbol.

The three line symbol patterns used to perform the thickening are shown below.



1-CELL 2-CELL 4-CELL

X 'ON' CELL OF THINNED LINE

RESULTING 'ON' CELLS GENERATED

Figure 3-19 shows the top-level flow diagram for the Variable line thickening Routine.

3.10 DOUBLE LINE SYMBOL ROUTINES

3.10.1 Function

These routines generate the double line symbols.

3.10.2 Approach

The original approach to Double Line Symbol Generation was to generate two lines within the arrays 'on-the-fly'. The larger width line corresponded to the outside diameter of the required line symbol, and the smaller line was the width of the inside edges of the double line symbol. For a detailed explanation the reader is referenced to subsection 3.3.7.3 of the Final Technical Report ETL-CR-74-20 Associative Array Processing for Topographic Data Reduction.

Because of the varying degree of widths (outer and inner diameters of the double line symbols) which may occur, it was decided to use a different technique. The technique will be to make two separate skid line symbols whose widths correspond to the outer and inner diameters of the double line symbol. This will be accomplished by using the Variable Line Thickening routines discussed in subsection 3.9. The smaller line symbol data will then be exclusively or'd with the larger line symbol data at merge time and thus generate the required double line symbol.

3.11 LINE EDITING ROUTINES

3.11.1 Function

The function of the line editing routines are to improve the line quality of data prior to processing by:

1) Joining small line breaks, and
2) Remove 'pips' on the thinned line data.

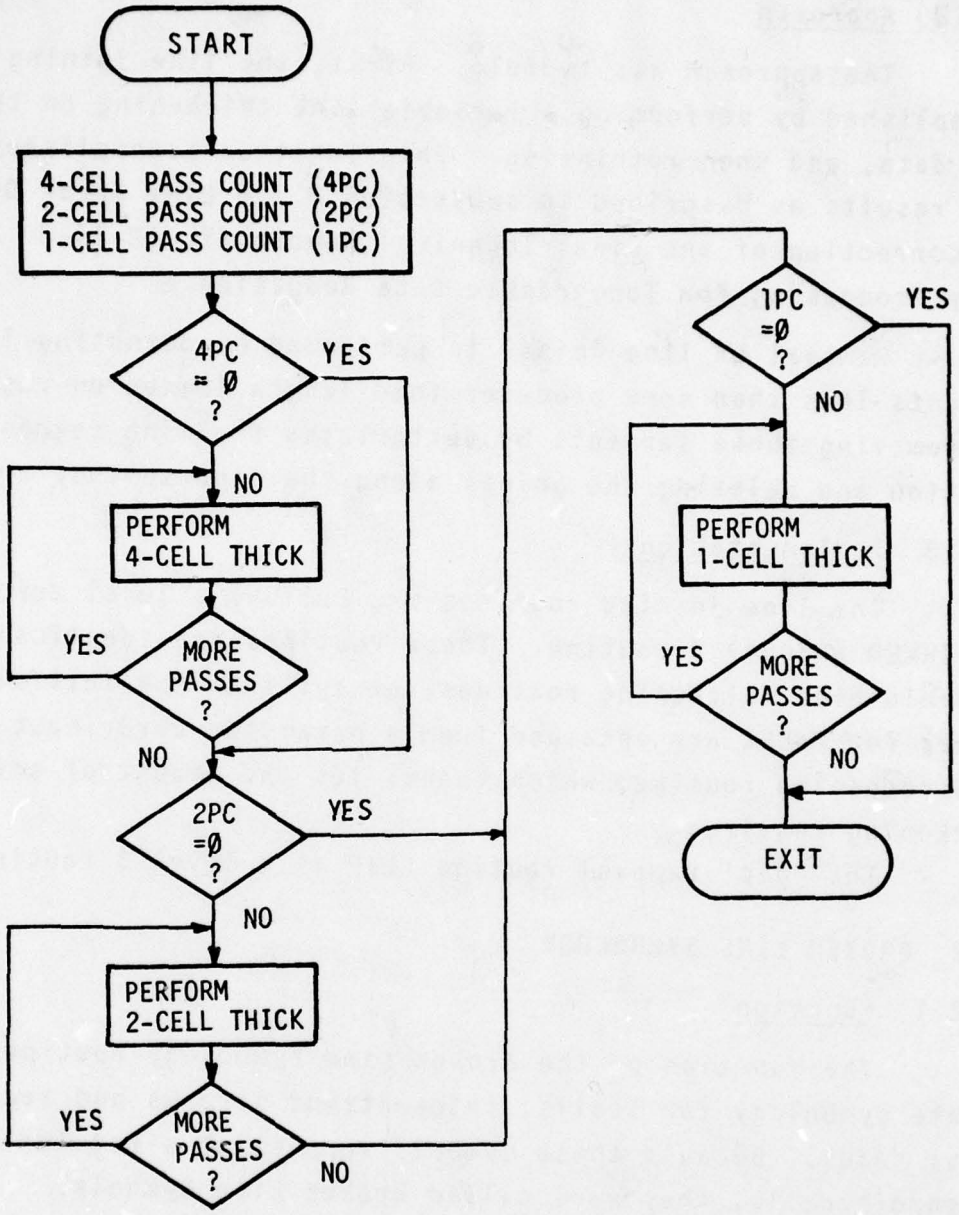


Figure 3-19. Variable line Thickening Routine Flow Diagram

3.11.2 Approach

The approach was twofold. First, the line joining is accomplished by performing a variable line thickening on thinned line data, and then rethinning. This function accomplishes the same results as described in subsection 3.3.6 Line Break Detection and Correction of the Final Technical Report ETL-CR-74-20 Associative Array Processing for Topographic Data Reduction.

Removal of line 'pips' is performed by detecting line segments less than some predetermined length (based on resolution) and removing those segments by vectorizing the line segments in question and deleting the points along the line as they are found.

3.11.3 Implementation

The line joining routines are EDITSET a level zero routine and THKED a level 3 routine. These routines are identical to the Variable Line Thickening routines, except that the initial parameters for THKED are obtained from a parameter card input to PRPRC (preprocessing routine) which identifies the amount of edit thickening required.

The 'pip' removal routine CLIP is a level 3 routine.

3.12 BROKEN LINE SYMBOLOGY

3.12.1 Function

The function of the Broken Line Symbology routines is to create symbology for trails, intermittent streams and second class roads. Because these symbols consist of a regular pattern of on/off cells, they were called Broken Line Symbols.

3.12.2 Approach

The approach was to preserve as much of the existing software as possible. (Refer to ETL-0046 subsection 3.8 for details of existing software.) Because creating railroad symbology

was, in reality, somewhat unrelated, its function was removed and placed in a separate module. To allow for more dense data, the "setting of paint bits" section of the module was modified to operate on a block basis. This modification allows array vectors to be off-loaded to the disk at the end of a block. In the old method, all array vectors were required to be core-resident at the time paint bits were set.

3.12.3 Implementation

Three routines are required to create Broken Line Symbols. They are CALRS, STPTBT and PAINT1. Each module is discussed in detail. The CALRS calculate the master vector lengths and other running sum values required by the set paint bit routine. It is called only once for each sheet, (after vectorizing and prior to STPTBT.) It performs the following functions:

1. Retrieve number of master vectors from MASTAB+1 and store in NOMVEC.
2. Retrieve number of array vectors from MASTAB and store in NOAVEC. Load array vector number of increments from VNITAB in master vector order until NOAVEC=0, along with the number of increments, save the array vector number and the reverse bit, RVBT, in the array.
3. Flag each group by setting the first bit of each group in START flag.
4. Continue steps 2 and 3 until NOMVEC=0 or there is no more array space.
5. Calculate the running sum values for each Master vector by calling the routine VLSUM$. This running sum becomes the E, or end, value for each array vector in the set paint bit routine.
6. Replicate the last value in each group (the master vector length or MVL) by calling the routine REPVAL$.
7. Output the E, MVL, and RVBT to VNITAB using the array vector number to address VNITAB.

8. If there are more master vectors, repeat steps 2-7, else, exit.

NOTE

If a master vector contains more than 256 array vectors, the message ERROR AT X'nnnn' (where 'nnnn' is an address in CALRS) will be printed out on the terminal and the program will halt.

Figure 3-20 shows the vector number of increments table before and after the execution of CALRS.

The set paint bit routine (STPTBT) is used to create the broken line symbology. It is executed on a block basis, setting paint bits for all array vectors in a particular block. The routine CALRS must have been run prior to the execution of this routine. The array vectors (ARVECTAB) for a particular block and the corresponding address table (VADTAB) must have been read off the disk prior to the call.

This routine is similar to the old set paint bit routine with the following exceptions.

1. The end (E) and master vector length (MVL) values are retrieved from VNITAB.
2. The number of increments (VI) are retrieved from ARVECTAB headers.
3. The threshold values for a particular pattern are variables (rather than constants.) They are stored in a 5 word table which is indexed by the resolution of the data being created.

The PAINT1 routine rasterizes the vectorized data back into the arrays for final output. It is identical to the PAINT routine developed under the last contract.

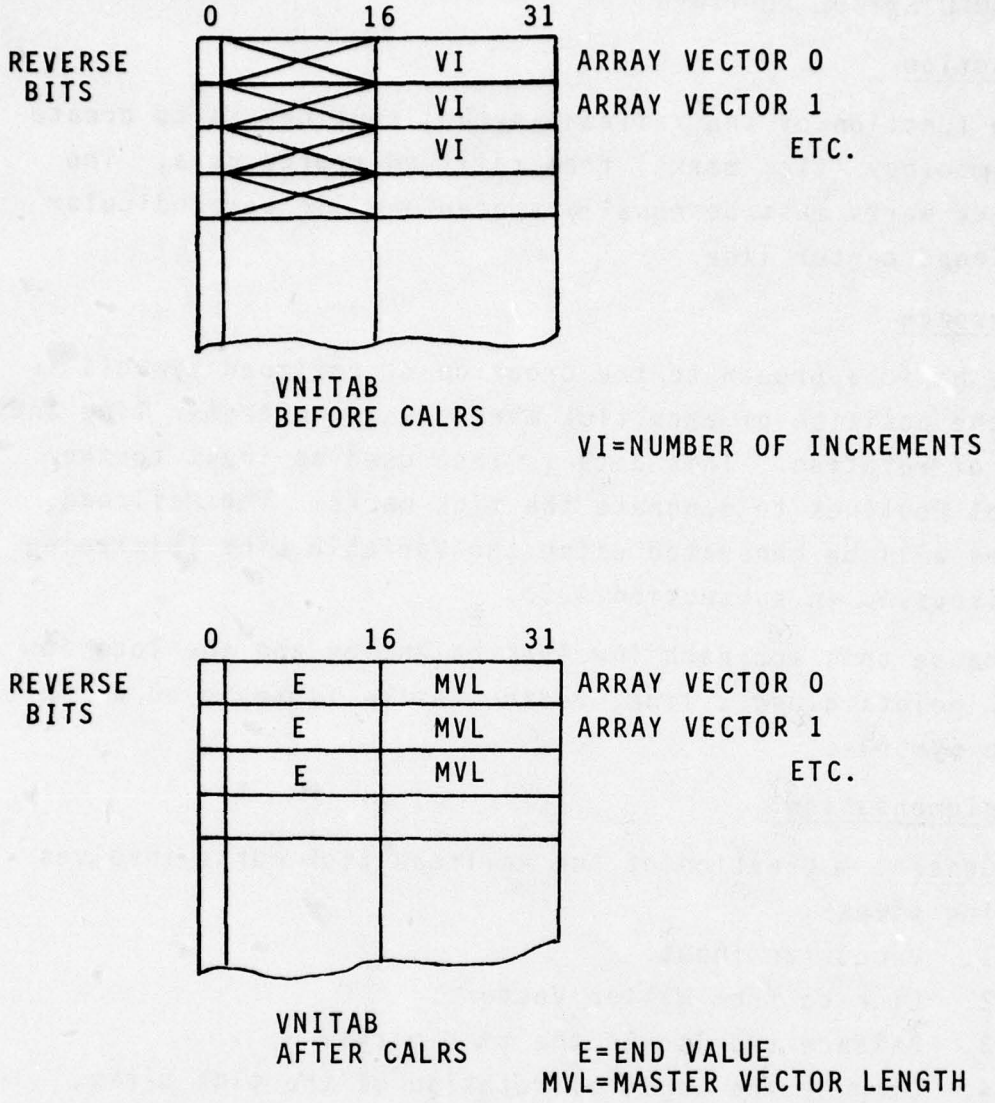


Figure 3-20. VNITAB Before and After Execution of CALRS

3.13 RAILROAD SYMBOL ROUTINES

3.13.1 Function

The function of the railroad symbol routines is to create railroad symbology (tick marks) from railroad source data. The railroad tick marks must be equally spaced and lie perpendicular to the railroad center line.

3.13.2 Approach

The basic approach to the creation of railroad symbols is to locate the position of each tick mark along the center line and it's angle of rotation. This data is then used as input to the Point Symbol Routines to generate the tick marks. The Railroad center lines will be generated using the Variable Line Thickening Routines discussed in subsection 3.9.

Because this approach involves balancing and the location of specific points along a line, vectorization is employed as in broken line symbols.

3.13.3 Implementation

3.13.3.1 General - Creation of the Railroad tick marks involves the following steps:

1. Vectorize input.
2. Link to from Master Vectors.
3. Balance and locate the tick marks.
4. Compute the angle of rotation of the tick marks, and their X,Y locations.
5. Draw the tick marks using the Point Symbol Routines.

These steps are discussed in more detail in the following paragraphs.

3.13.3.2 Vectorize - This technique is similar to that described in subsection 3.16.

3.13.3.3 Link - This routine is the one discussed in subsection 3.16.

3.13.3.4 Balance - This routine is a level 1 routine called TCKED (see Figure 3-21) which finds the locations of the railroad ties (ticks) in a similar manner to the SETPTBT routine discussed in subsections 3.12 and 3.8.3.5 of the previous Technical Report, Associative Array Processing of Raster Scanned Data for Automated Cartography (ETL-0046). TCKFD utilizes the results of the CALRS routine which calculates the master vector lengths and the running sum values. The TCKFD routine is similar to the original SETPTBT routine (as used for Railroad Symbology), except for the following changes:

1. TCKFD is executed on a block basis.
2. The end (E) and master vecotr lengths (MVL) are retrieved from VNITAB.
3. The number of increments in each array vector (VI) are retrieved from the ARVECTAB headers.
4. The railroad symbol threshold values are variable with respect to resolution, as are the ends of the railroad ticks.
5. During the determination of the ticks the end values LE, RE overwrite the array vector increments within the ARVECTAB, and the number of ticks and their start address within ARVECTAB are saved in VADTAB.
6. The final tick values (at the end of each data block) are located as pairs of LE, RE values stored in a tick location table TKLTAB where the number of ticks and the initial address relative to the start of TKLTAB overwrite the VNITAB.

3.13.3.5 Compute - This routine is a Level 0 Routine called (CRE8TK), see Figure 3-22, which generates the angle of rotation and the absolute location of the center of the ticks.

The first part of the routine is similar to the first part of the Create Tick Routine documented in Section 3.9.36 of the previous Technical Report, Associative Array Processing of Raster Scanned Data for Automated Cartography.

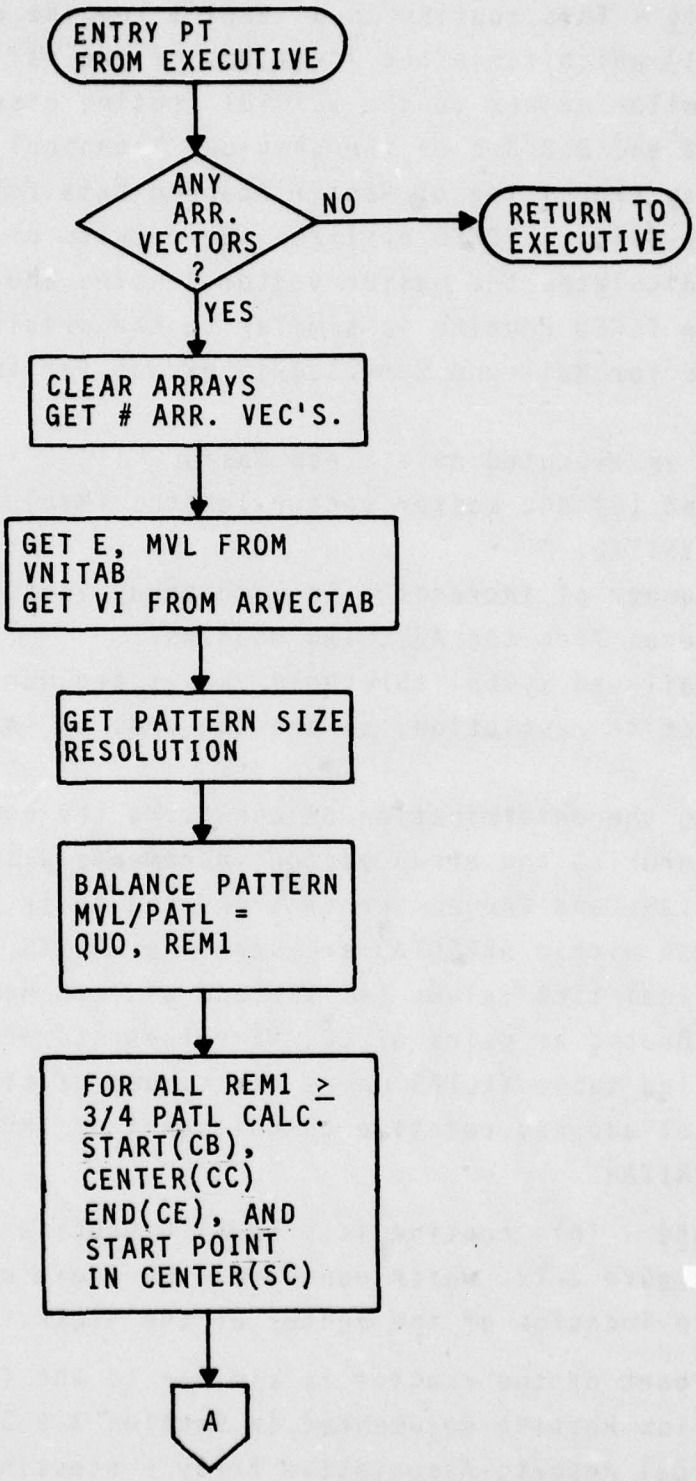


Figure 3-21. Find Tick (TCKFD) Routine Flow Diagram

(Sheet 1 of 3)

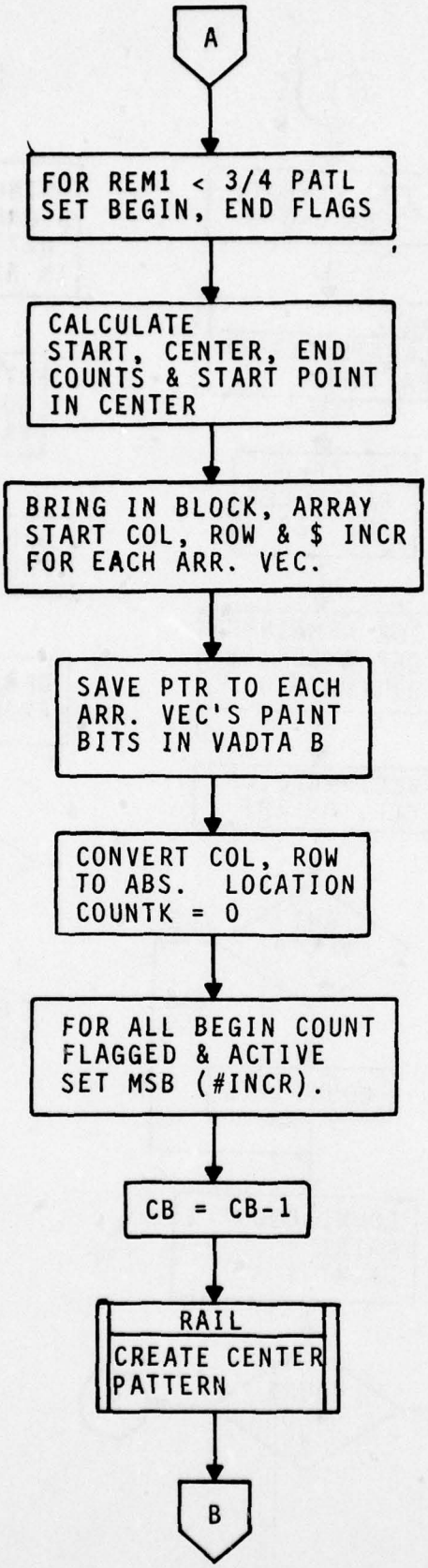


Figure 3-21. Find Tick (TCKFD) Routine Flow Diagram

(Sheet 2 of 3)

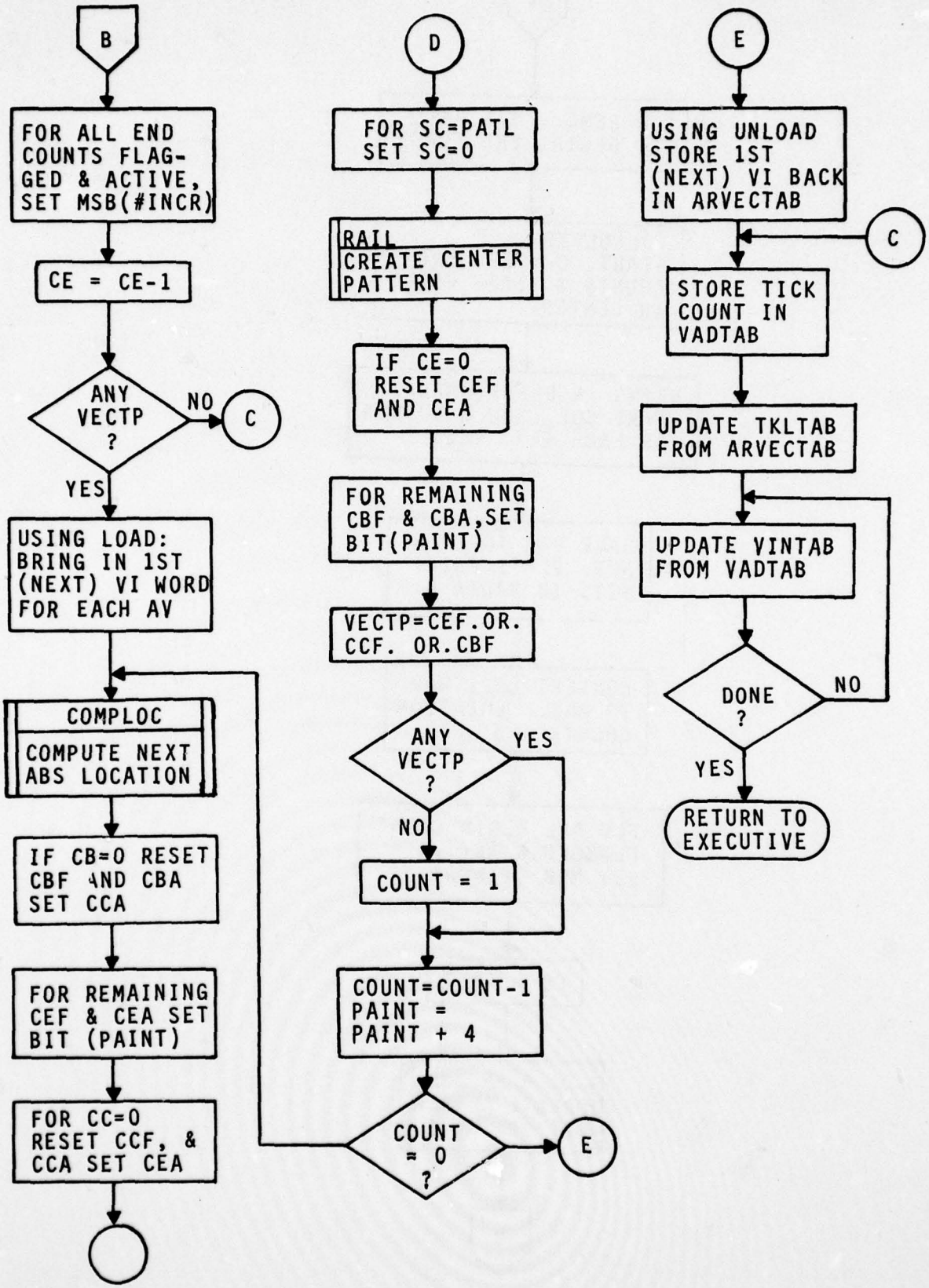


Figure 3-21. Find Tick (TCKFD) Routine Flow Diagram

(Sheet 3 of 3)

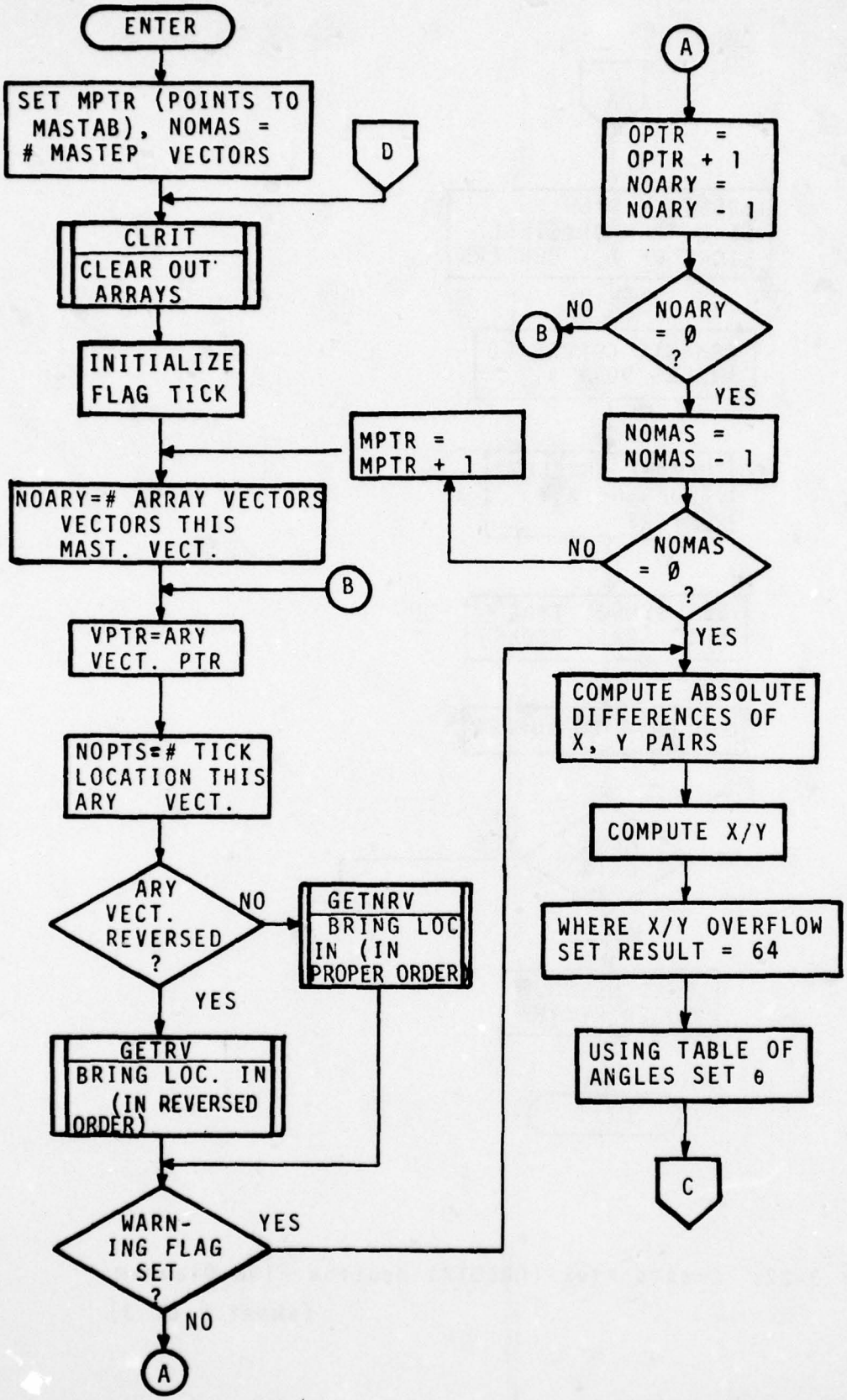


Figure 3-22. Create tick (CRE8TK) Routine Flow Diagram (sheet 1 of 3)

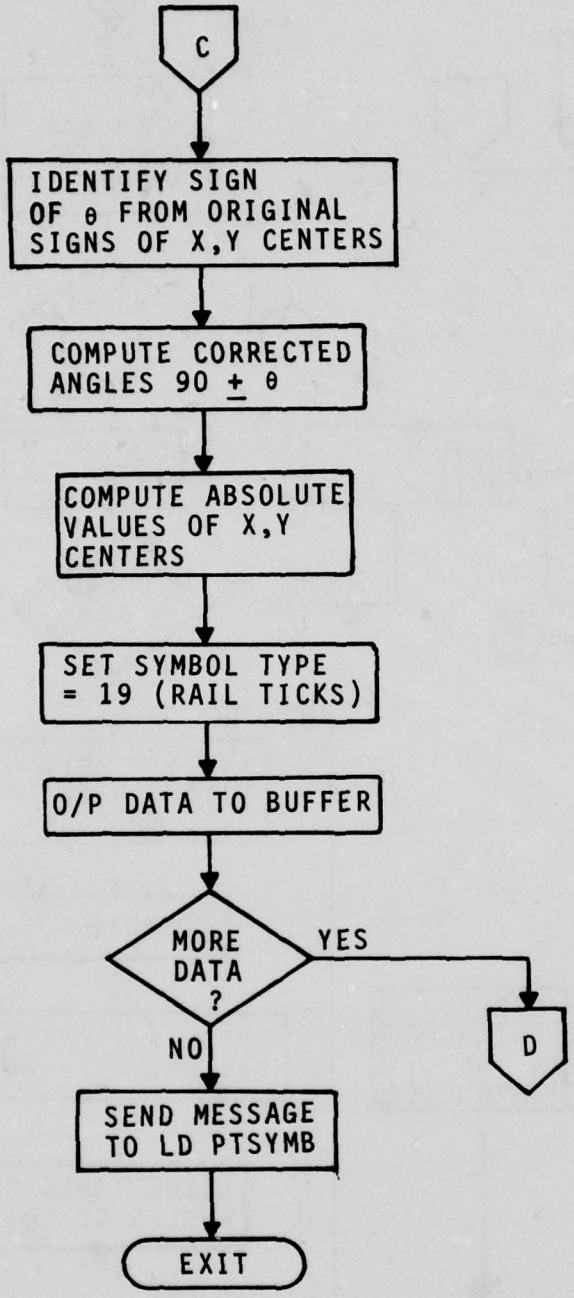
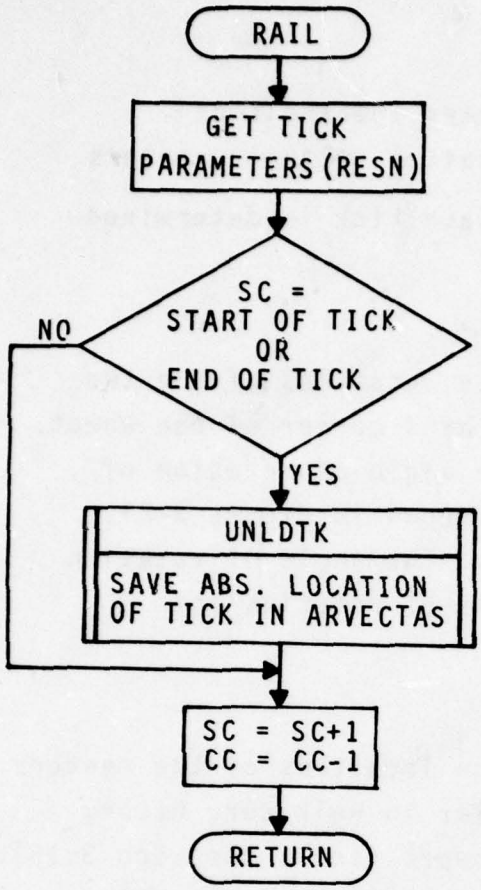
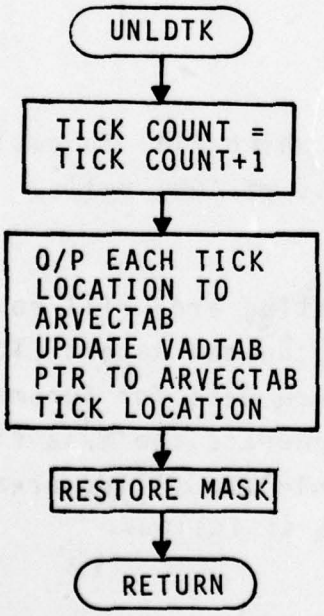


Figure 3-22. Create tick (CRE8TK) Routine Flow Diagram (sheet 2 of 3)





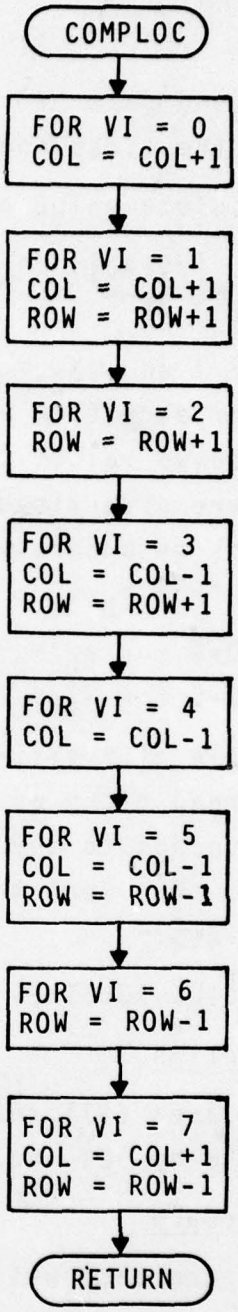


Figure 3-22. Create tick (CRE8TK) Routine Flow Diagram (sheet 3 of 3)

The second part of the routine generates the angle of rotation of the ticks and the absolute location of their centers.

The absolute value of the center of each tick is determined by:

$$X_D = \frac{|X_1 - X_2|}{2} \qquad \frac{|Y_1 - Y_2|}{2} = Y_D$$

where (X_1, Y_1) and (X_2, Y_2) are the absolute locations of the two end points of each tick from the top left hand corner of the sheet. The table lookup values to approximate the angle of rotation of the ticks, are determined from the tables shown in Figure 3-23. This provides an accuracy of $\pm$7.5 degrees. The angle of rotation is computed from $X_D/Y_D \simeq \theta$ and the true angle of rotation is:

$90+\theta$ for X_D/Y_D negative valued
$90-\theta$ for X_D/Y_D positive valued

The angle of rotation and the absolute locations of the centers of the railroad ticks are output to a buffer in Bulk Core Memory which corresponds to the buffer used in Prepro (see subsection 3.15). A message is then sent to the user to load and execute the Point Symbol Routines.

3.14 AREA FILL ROUTINES

3.14.1 Function

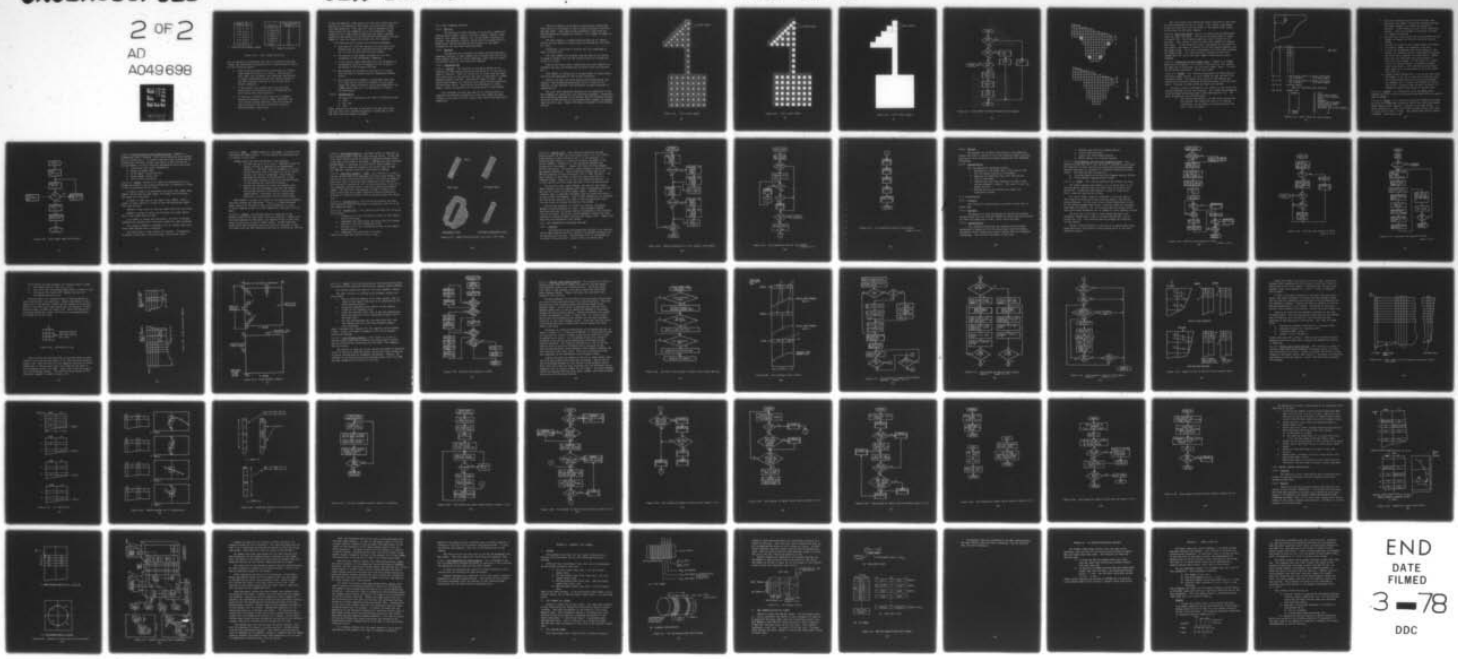
The area fill routines are routines which fill in outlined areas defined by 'ON' cells with a solid mass of 'ON' cells.

3.14.2 Approach

The approach was to utilize the existing area fill routines described in subsection 3.10 of the Final Technical Report ETL-0046 Associative Array Processing of Raster Scanned Data for Automated Cartography. However, the routines which generate the area fill patterns and logically .AND. the patterns into the filled areas have been excluded. The reason for this is as follows.

AD-A049 698 GOODYEAR AEROSPACE CORP AKRON OHIO F/G 9/5
ASSOCIATIVE ARRAY PROCESSING OF RASTER SCANNED DATA FOR AUTOMAT--ETC(U)
NOV 77 N J ADAMS, J M VOCAR, K LOSCH DAAG53-76-C-0146
UNCLASSIFIED GER-16523 ETL-0132 NL

2 OF 2
AD
A049698



END
DATE
FILMED
3-78
DDC

tanθ	θ°
0.0	0.0
0.13	7.5
0.41	22.5
0.77	37.5
1.3	52.5
2.41	67.5
7.6	82.5

A. Relationship between Tanθ/θ

X/Y = N	Resulting Angle Chosen θ
$0.0<N<0.13$	0
$0.13\leq N<0.41$	15
$0.41\leq N<0.77$	30
$0.77\leq N<1.3$	45
$1.3\leq N<2.41$	60
$2.41\leq N<7.6$	75
$7.6\leq N<\infty$	90

B. Range for each θ

Figure 3-23. Table Lookup Definition

The old approach of generating the area fill patterns from some library was considered inappropriate for a semi-production system for the following reasons.

1. Each different pattern had to be designed as a binary coded image and saved in a Library. The design of each symbol was extremely time consuming, particularly at higher resolutions and yet must closely adhere to the specifications required to satisfy the Cartographic community.
2. If the pattern size exceeds the array size then subsections of the pattern had to be generated to match across the array boundaries.
3. In the case of patterns which consist of randomly located sub-patterns such as swamps, marshland, etc. this library storage technique, and the calling of each sub-pattern into the arrays when required, became extremely complicated.

In the new approach, a map sheet sized STIC-PAT of each area fill pattern will be scanned at the proper resolution and a raster plot tape generated. This tape will then be merged with the output tape from the STARAN as an off-line procedure. Where scanning a full sheet requires more than one tape, a smaller segment of the pattern will be scanned and then merged a multiple number of times to produce the correct area fill pattern.

This technique will provide the following advantages.

1. Each area fill STIC-PAT pattern will only have to be scanned once for each required resolution and then the file can be saved and used when needed.
2. The quality of the symbols produced will be more acceptable to the Cartographic community.
3. In the case of complicated patterns, the randomness of sub-patterns (such as in swamps) will be as good as the design of the original STIC-PAT.
4. New patterns can be added to the area fill Library with a minimal of design time and subsequent software effort.
5. As a production environment is approached these area fill tapes may be stored as resident files on some larger and faster peripheral device and merged with the STARAN output tape as it is generated on a record-by-record basis.

3.14.3 Implementation

The area fill routines are all level 3 routines and are:

1. AFIL,
2. GTL, and
3. SVL.

These routines are the same as the area fill, get last slice, and save last slice routines described in subsection 3.10 of the Final Technical Report ETL-0046.

3.15 POINT SYMBOLOGY ROUTINES

3.15.1 Function

The function of these routines is to provide a method for creating point symbols such as, churches, houses, schools, route markers, mine symbols, numbers, etc. at the proper locations on the map. The churches, schools, houses must be rotatable and all symbols must reflect the quality and accuracy of set symbol standards at the resolution being processed.

3.15.2 Approach

The approach was to modify and expand the existing point symbology routines in such a manner as to minimize the size of the point symbols defined in a Point Symbology Library and yet obtain the desired results at the resolution required.

3.15.3 Implementation

3.15.3.1 General - The locations of the point symbols to be produced on a map overlay sheet are defined by an X,Y pair and an angle of rotation. These X,Y values are in inches from the upper left hand corner of the overlay; the angle is specified in integer degrees between 0 and 359. This data is input on cards. The actual point symbols are stored as $\Delta X, \Delta Y$ pairs from an arbitrarily defined center point. These points are contained in a Point Symbol Library and describe only selected points of each symbol.

This technique utilizes the same set of library points to generate the different resolutions of the same symbol and thus minimize, disk space utilized, access time, and Library generation complexity.

The 1-mil symbols are created by rotating and translating these points to correspond to their correct position in an array and data block. These data points contained in the array are then thickened and smoothed to achieve a 1-mil symbol as shown in Figure 3-24.

The 2-mil symbol is created from the same set of library points, but the values of those points are divided by 2 (a shift of 1 bit).

Thickening is utilized to achieve the 2-mil symbology as shown in Figure 3-25.

The 4-mil symbols are created from the same set of points, as shown in Figure 3-26, but the output points are divided by 4 (shift of 2 bits).

Some of the points used in generating the 4-mil symbols are redundant and map into neighboring locations after the division by 4.

Each symbol is defined with a minimum number of points while retaining the appearance of the correct resolution.

Figure 3-27 shows the Point Symbol editing/thickening flow diagram. The thickening subroutines (Thick 1, Thick 3) are identical to the Variable line thickening routines discussed in subsection 3.9.

The 1-mil edit subroutine smoothes out the boundaries of the Point Symbols to create the 1 mil resolution boundary. Essentially, this subroutine generates cells (points) when the boundary changes direction by 90^{o} and both of the perpendicular segments are greater than two cells long. Figure 3-28 shows an example of this editing process.

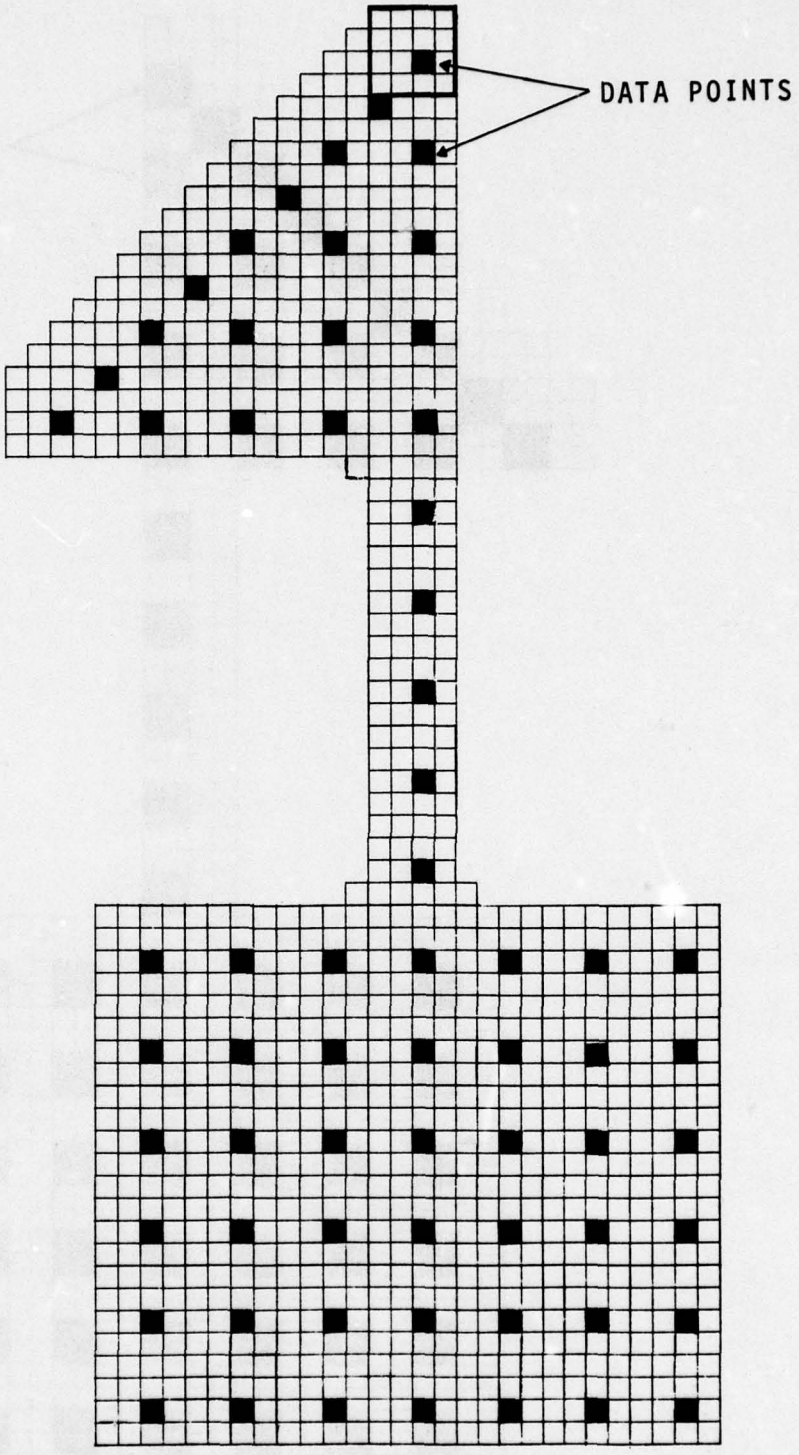


Figure 3-24. 1-Mil School Symbol

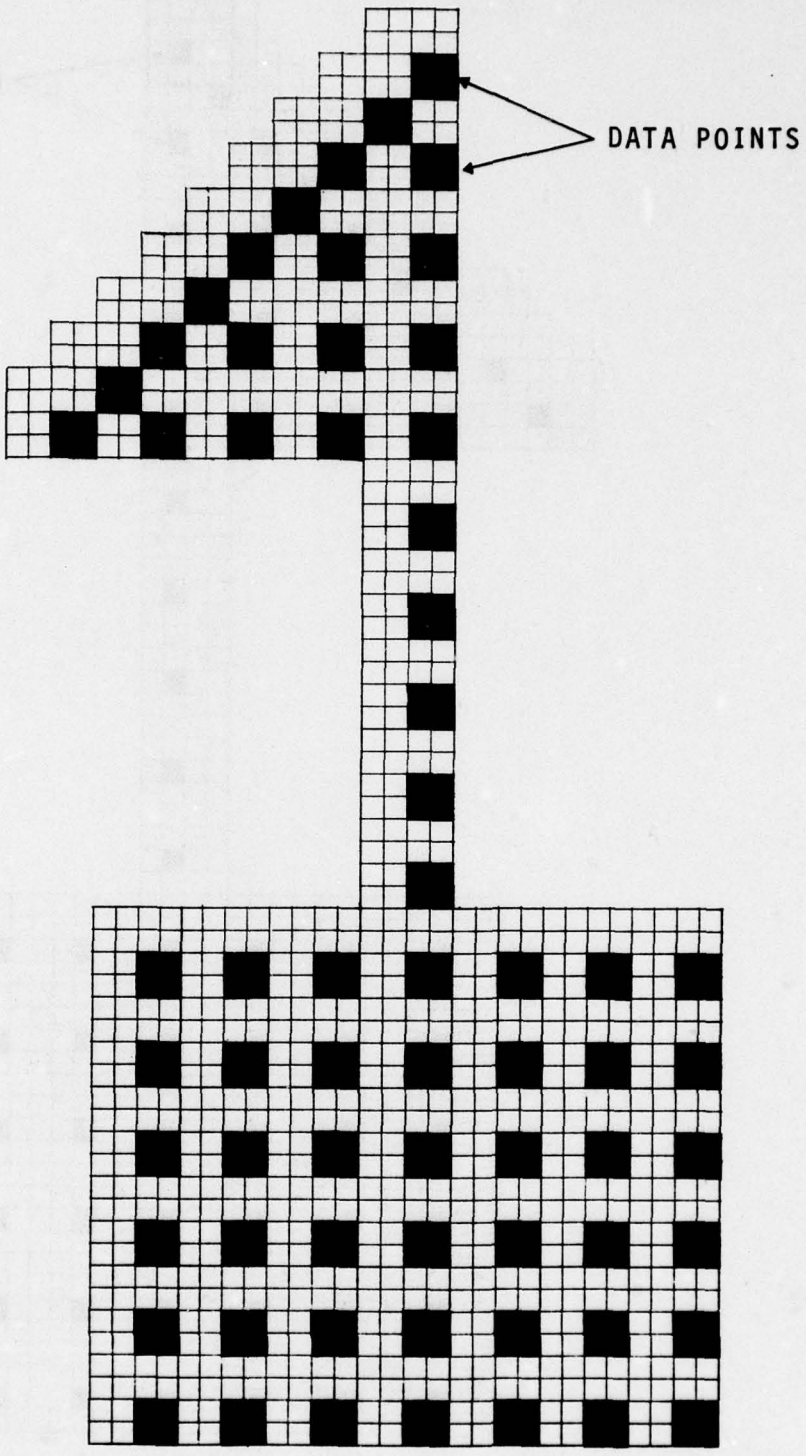


Figure 3-25. 2-Mil School Symbol

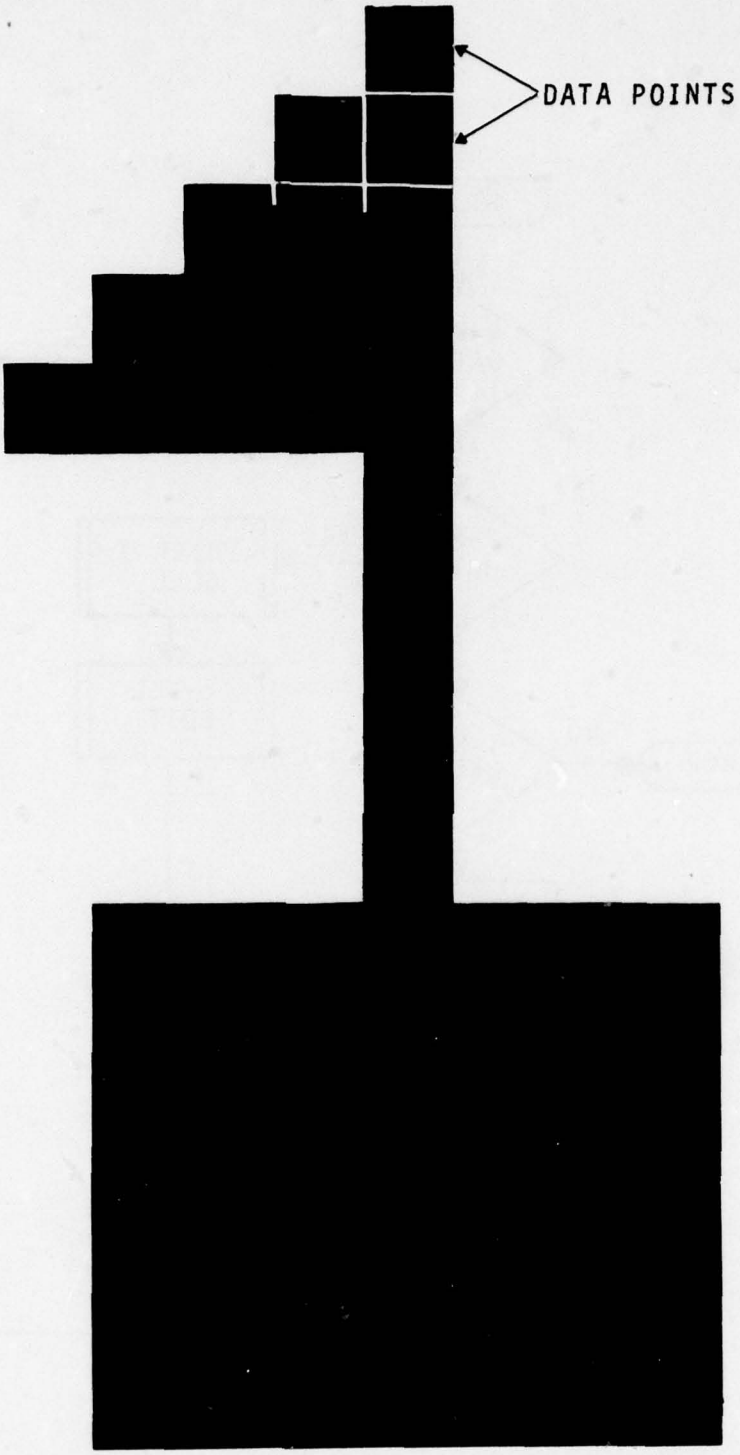


Figure 3-26. 4-Mil School Symbol

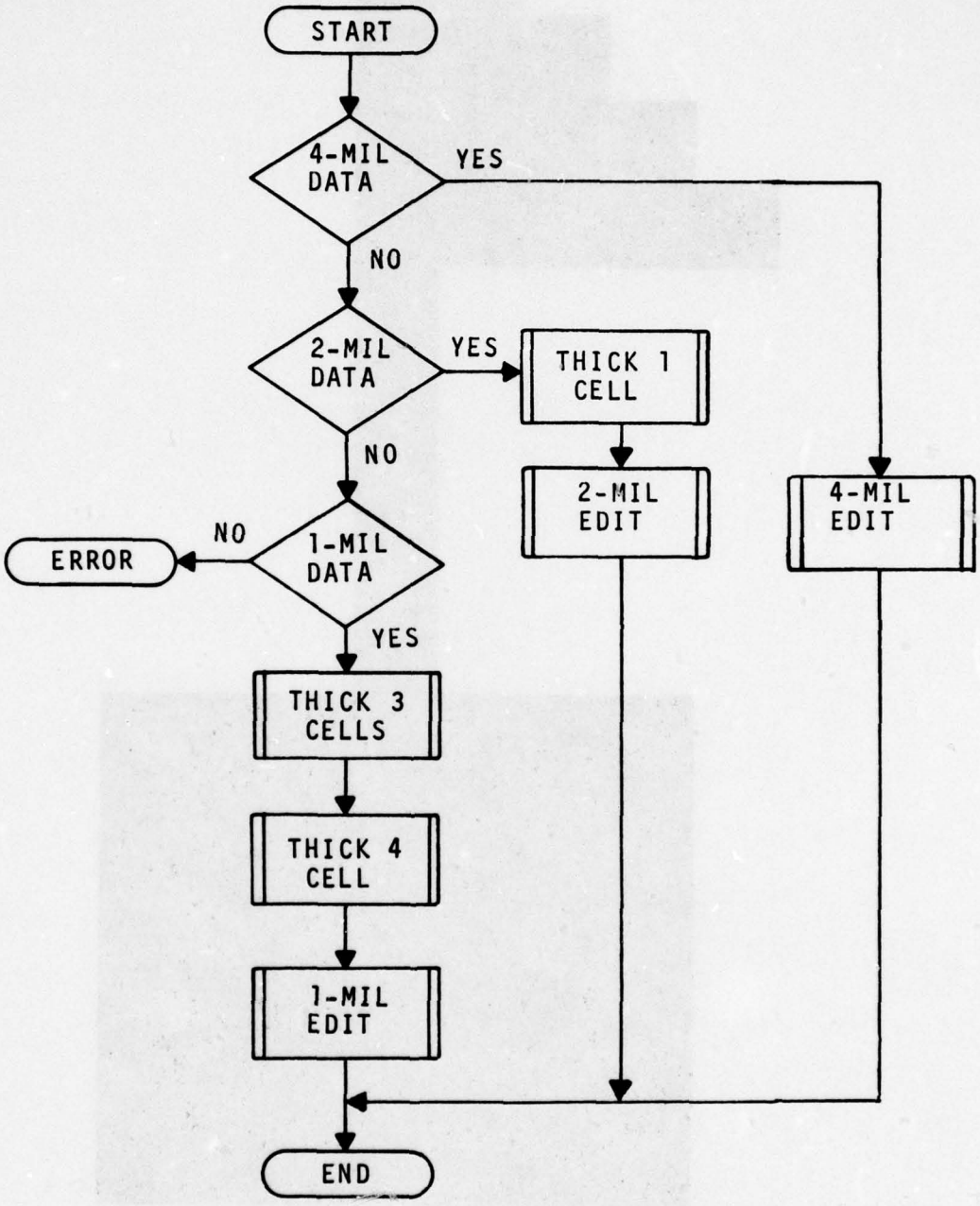


Figure 3-27. Point Symbol Editing/Thickening Flow Diagram

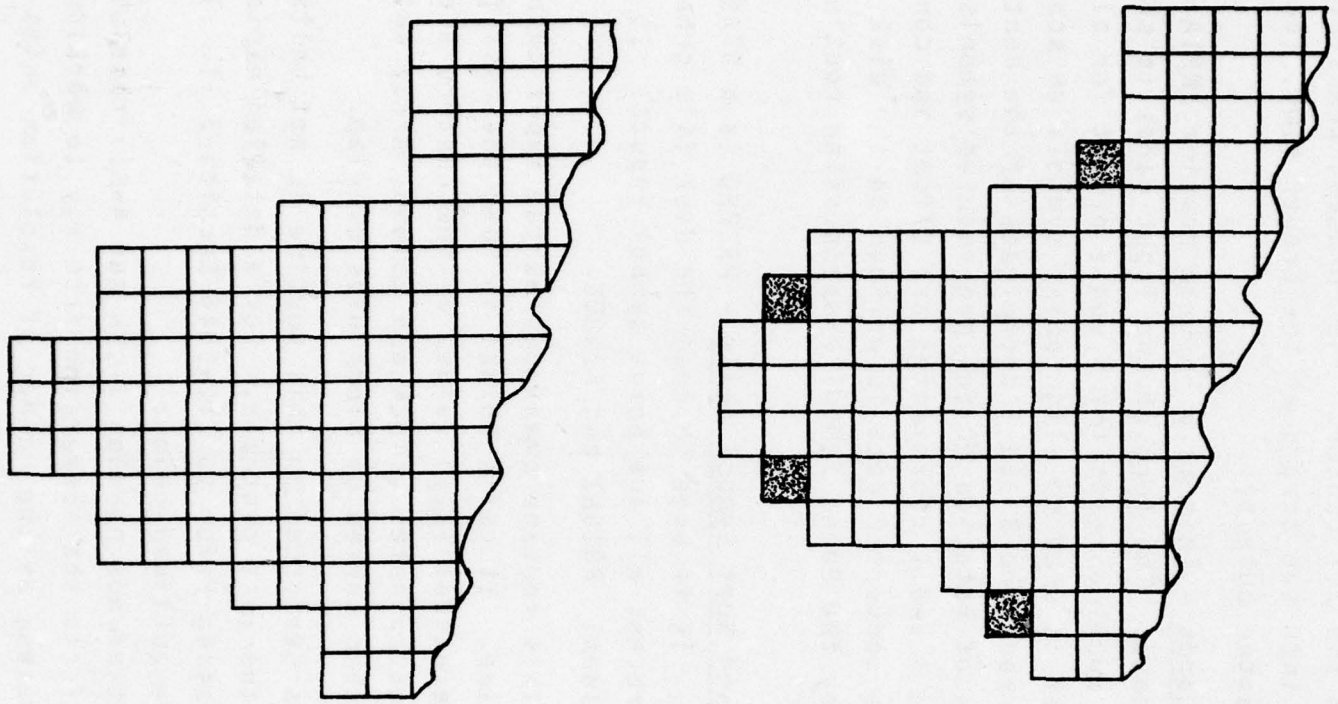


Figure 3-28. One-mil resolution Boundary Editing process

The $\Delta X, \Delta Y$ points are rotated the amount specified on the input card and translated to the X,Y center. The rotated, translated points are then drawn into the arrays at the proper time, (based on the order of the raster output).

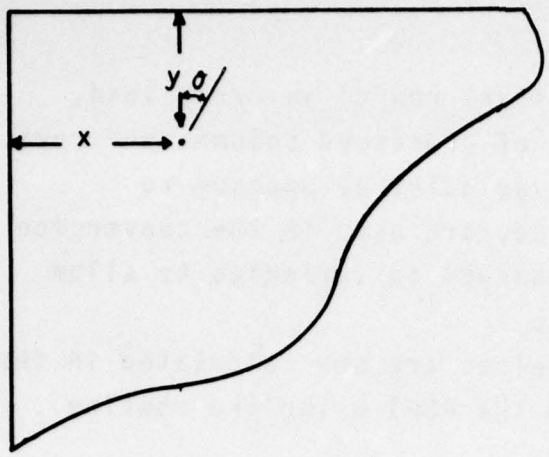
3.15.3.2 Read Input Cards - This is a Fortran routine INPTA which reads the input card data. The form of the input cards is shown in Figure 3-29. This routine reads the X and Y offset for all the data. (This offset is used to align point symbols on scanned road data). It then reads the X and Y coordinate of the center of the symbols, the angle of rotation (0 for non-rotated symbols), and the symbol flags. The X and Y coordinates are offset and converted to STARAN format. The converted data is written on the disk (FOR001.DAT) for use by the Point Symbol Preprocessing routine (PREPRO).

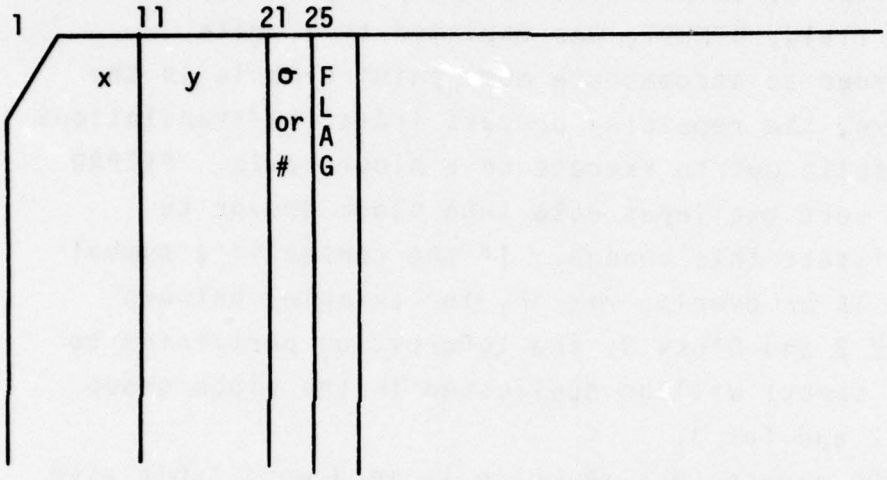
3.15.3.3 Preprocess and Sort Symbol cells - PREPRO is a STARAN-APPLE Level 0 program. It is used to read the disk file created by INPTA and to pre-process all the point symbol inputs. It calls the following subroutines: *RDISK1 and SINCOS*.

3.15.3.3.1 PREPRO - This routine preprocesses the input data for the point symbol routine. It calls RDISK1 to read the disk file created by INPTA. The actual (x,y) values of the centers are converted to working coordinates (Block and column, Array and row) pairs. The data are then sorted by block plus overlap.

The basic process performed in this routine is not unlike the process performed in the last contract. For a detailed explanation of point symbol processing refer to ETL-0046 Section 3.11. The major discepancies are outlined below:

1. The routine now handles 1-,2- and 4-mil resolutions. This affects the conversion from x,y to working coordinates, as the number of resolution units per inch is now a variable.





CARD INPUT

COL 1-10 x POSITION OF CENTER IN INCHES FROM UPPER LEFT HAND CORNER (10.3 FORMAT)

COL 11-20 y POSITION OF CENTER IN INCHES FROM UPPER LEFT HAND CORNER (10.3 FORMAT)

COL 21-24 ROTATION ANGLE:

ANGLE OF ROTATION 0=359 (INTEGER)

COL 25-28 SYMBOL FLAG

0	NUMBER 0	12	MINE
1	1	13	STATE ROUTE MARKER
2	2	14	FEDERAL ROUTE MARKER
3	3	15	HOUSE
4	4	16	LANDMARK
5	5	17	FEDERAL ROUTE BLANKER
6	6	18	STATE ROUTE BLANKER
7	7	19	RAILROAD TIES
8	8	20	(RESERVED FOR FUTURE SYMBOLS)
9	9	21	
10	CHURCH	22	
11	SCHOOL	•	
		•	

Figure 3-29. Input Format for Point Symbols

2. The start column and start row of an array load, along with the number of processed columns and rows have been changed to variables as opposed to constants. These values are used in the conversion process. They were changed to variables to allow for changes in overlap.
3. The sine and cosine values are now calculated in the STARAN (as opposed to the PDP) using the routine SINCOS.
4. In order to allow for more symbol types, the symbol flag field, SYMBPT, was expanded to 12 bits.
5. In order to accommodate more paint symbols in the future, the remaining process (rotation/translation) was split out to execute on a block basis. PREPRO must sort the input data into block groups to facilitate this change. If the center of a symbol lies in an overlap region, for example, between Block 2 and Block 3, the information pertaining to that symbol will be duplicated in the block group for 2 and for 3.
6. PREPRO creates BLKTAB which is an 8 word table used by BLKPRO. A bit set in that table indicates that the corresponding block has data to be processed. For example, Bit 3 in word 0 of BLKTAB set implies that Block 3 contains data and Bit 2 in word 1 of BLKTAB set implies that Block 34 contains data.

3.15.3.3.2 SINCOS - Is a subroutine called by PREPRO which computes the sine, cosine of the angles associated with the symbols read by PREPRO.

3.15.3.3.3 RDISK1 - Is a subroutine called by PREPRO which reads the disk file FOR001.DAT created by INPTA into a Bulk Core Memory buffer whose initial address is defined by the name BUFFER. It reads 124 bytes at a time; 120 of which contain input point data. It reads until an EOF is found. The program then returns control to PREPRO. (see Figure 3-30).

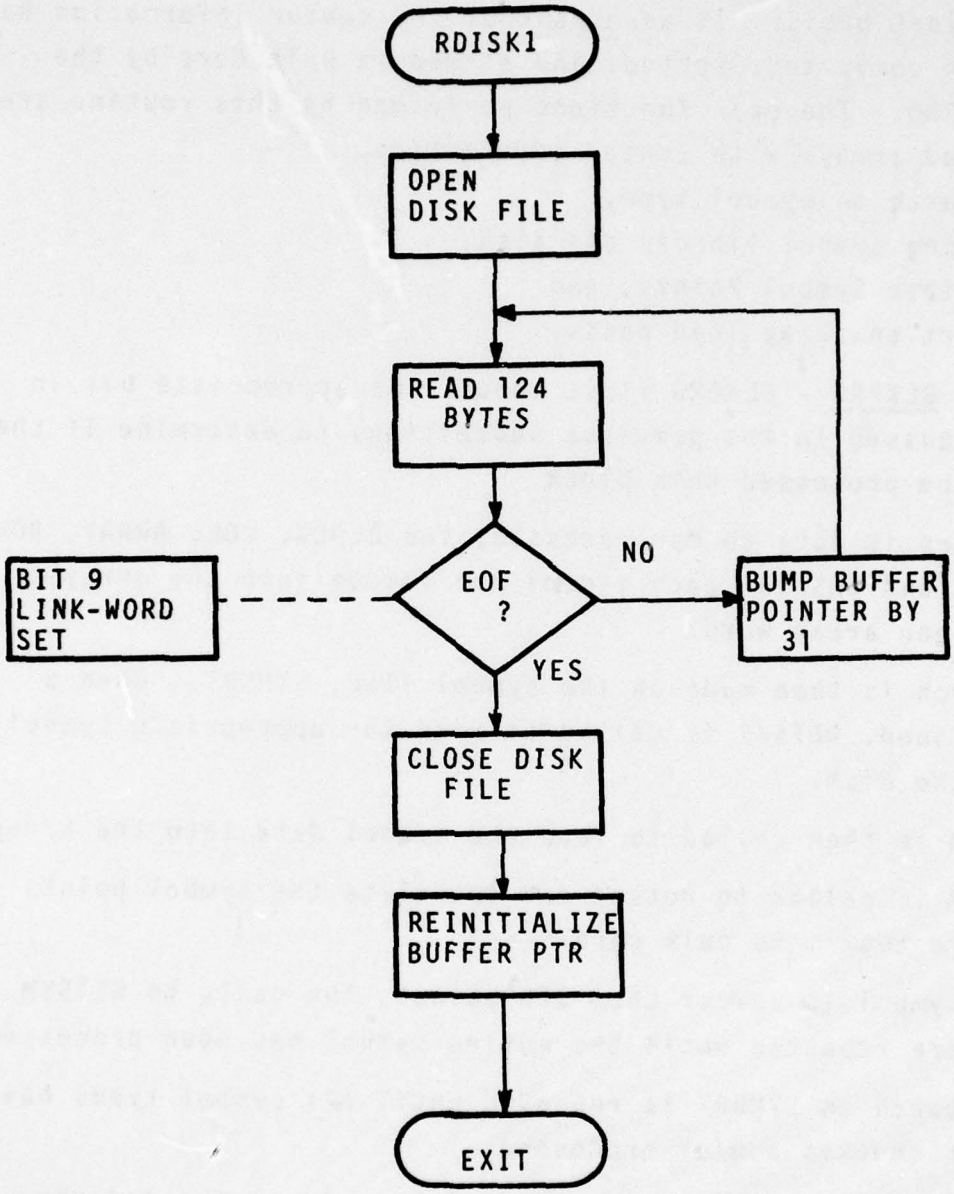


Figure 3-30. Point Symbol Read Disk Routine

3.15.3.4 Block Processing Point Symbol Routine - BLKPRO is a STARAN-APPLE Level 1 program. This routine processes point symbol data on a block basis. It assumes that the center information has already been converted, sorted, and stored in Bulk Core by the routine PREPRO. The main functions performed by this routine are:

1. Load arrays with center information,
2. Search on symbol type,
3. Bring symbol library off disk,
4. Rotate Symbol Points, and
5. Sort on array load basis.

3.15.3.4.1 BLKPRO - BLKPRO first checks the appropriate bit in BLKTHB (discussed in the previous subsection) to determine if there is data to be processed this block.

If there is data to be processed, the BLOCK, COL, ARRAY, ROW, SYMBPT, SIN and COS for each symbol are loaded into the arrays, one symbol per array word.

A search is then made on the symbol flag, SYMBPT. When a symbol is found, *RDISK1 is called to read the appropriate symbol* file from the disk.

GETSYM is then called to load the symbol data into the arrays.

PROSYM is called to rotate and translate the symbol points and to store them into bulk core.

If a symbol is larger than 128 points, the calls to GETSYM andPROSYM are repeated until the entire symbol has been processed.

The search on SYMBPT is repeated until all symbol types have either been checked and/or processed.

The output data is then ordered for drawing. The ordering technique is the same as described under the previous effort.

3.15.3.4.2 PROSYM - PROSYM rotates (if the symbol is rotable) and translates the symbol points. It also outputs the processed data to a buffer in bulk core.

PROSYM varies from the old routine in two respects:

1. The data can be 4-,2- or 1-mil. Therefore, prior to output the data is adjusted to the appropriate resolution. The 4-mil data is divided by 4 (a 2 bit shift) the 2-mil data is divided by 2 (1 bit shift) and the 1 mil data is unchanged. The divide is performed at the end of the processing, so that the fractional portion of the $\Delta X, \Delta Y$ can participate in the rotation and translation process.
2. Because the creation of 2 and 1 mil symbols ultimately requires thickening, data must be drawn in the overlap region of the array to insure proper symbolization. Therefore, the sort routine was modified to include points in the overlap region.

When PROSYM is exited, all symbols of the type being processed have been rotated and translated. (That is, if there are more symbol centers of the type or than can be processed in 1 pass, (1 pass=4 symbols at GAC, 8 at ETL) PROSYM will loop until all are done.)

3.15.3.4.3 GETSYM - The GETSYM routine is identical to the GETSYM routine of the last effort with one exception. Symbol size has been increased to 128 points, (as opposed to 64) so that each array can hold only 2 sets of symbol points. GETSYM loads the symbol point data from bulk core into the first 128 words of each available array and then replicates the data by rotating and .OR.ing.

3.15.3.5 Draw Symbol Routine - The DRAW routine is identical to that routine created under the last contract effort. It was the array table ARYTAB created by BLKPRO to determine whether there is any data to be drawn in the array. (ARYTAB acts just like BLKT BLKTAB except it indicates an array load of data instead of a Block.) Data is drawn by setting a bit in the array for each COL, ROW pair stored in the bulk core buffer by PROSYM.

3.15.3.6 Edit Point Symbols - PTEDT - This routine is a Level 3 routine called PTEDT which generates the final point symbols from the set of $\Delta X, \Delta Y$ values positioned within the arrays by DRAW and edits this data in different ways depending on the resolution. This editing is required to fill in holes resulting in some $\Delta X, \Delta Y$ cells mapping into each other during the rotation and drawing stages. The top-level flow diagram is shown in Figure 3-27. Each of the three point symbol generation editing subroutines performs a slightly different function depending upon the resolution.

3.15.3.6.1 Four-Mil Edit - This subroutine performs the same editing process that was described in subsection 3.14.3.5 of the last Technical Report.

3.15.3.6.2 Two-Mil Edit - This subroutine performs the following edit process.

1. Perform a 3 x 3 cell thickening process on the symbols within the arrays.
2. Complement the data within the arrays (ON cells become OFF cells, OFF cells become ON cells).
3. Perform a 3 x 3 cell thickening process on the complemented symbols.
4. Complement the data within the arrays.

Figure 3-31 shows the results of the 4 steps.

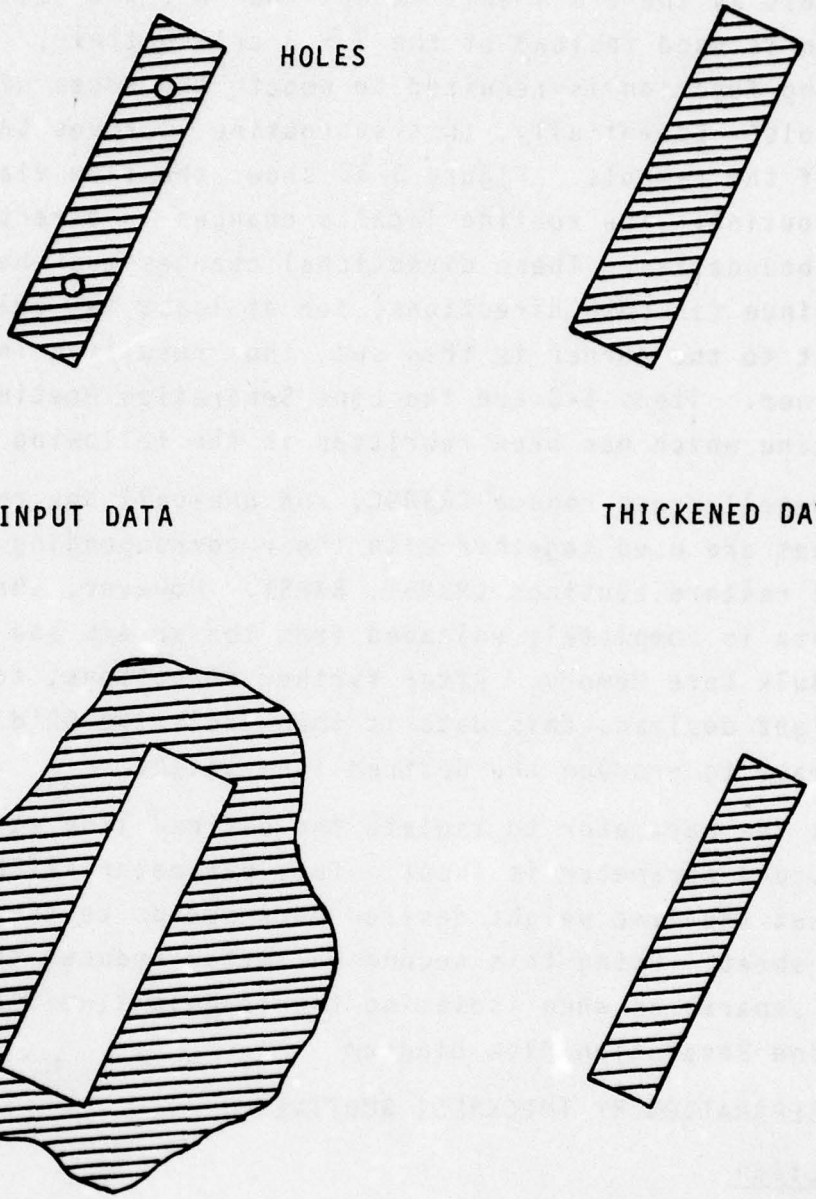


Figure 3-31. Symbol Editing process for 2-mil, 1-mil data.

3.15.3.6.3 One-Mil Edit - This subroutine performs the same editing process as the 2-mil edit except that a 5 x 5 cell thickening pattern is used instead of the 3 x 3 cell pattern. A second editing function is required to smooth the edges of the one-mil symbols. Essentially, this subroutine improves the resolution of the symbols. Figure 3-32 shows the flow diagram of this subroutine. The routine locates changes in direction along the symbols boundaries. These directional changes must be by 90^0 and continue (in both directions) for at least two cells. The point nearest to the corner is then set, thus resulting in a smoothed corner. Steps 1-3 are the Line Separation Routine. It is this routine which has been rewritten in the following manner.

The two-cell cross reduce CRSRDC, and one-cell box reduce BXRDC routines are used together with their corresponding two cell and one cell restore routines CRSRST, BXRST. However, the 'reduced' data is completely unloaded from the arrays and saved in an area of Bulk Core Memory. After further reductions, to eliminate the line weight desired, this data is then exclusive OR'd back into the arrays to produce the desired line weight.

Besides the parameter to isolate the desired line weight WGTDES, a second parameter is input. This parameter FATFLG indicates that the line weight desired corresponds to the thickest line on the sheet. Using this second parameter reduces the steps involved in separation when isolating the largest line. Figure 3-33 shows the Line Separation Flow Diagram.

3.16 LINE SEPARATION BY THICKNESS ROUTINES

3.16.1 Function

The function of the Line Separation routines is to take one raster input tape and extract a desired line weight selected prior to processing by the user. The extracted line is represented by its enter line and stored in raster from on an output tape.

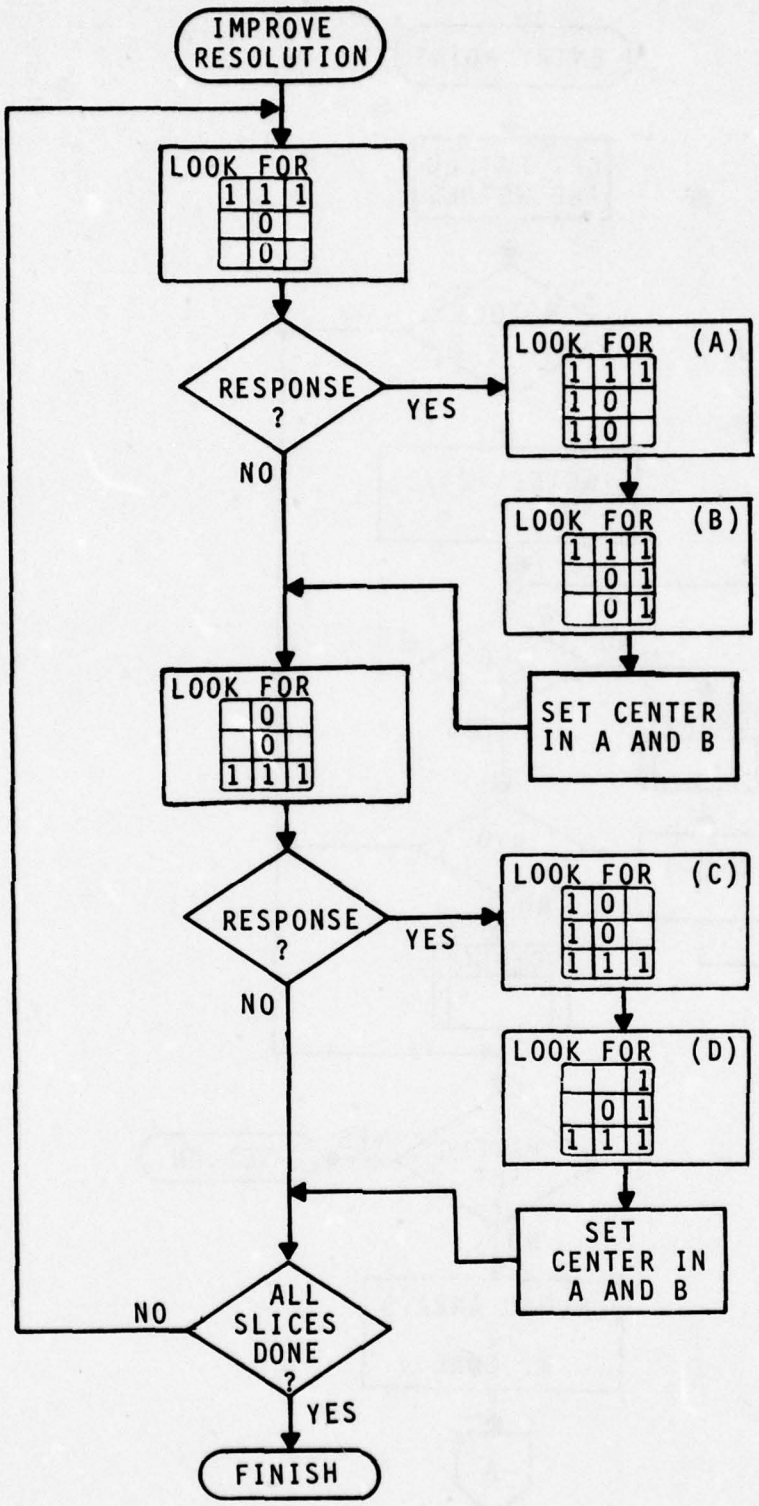


Figure 3-32. Improve Resolution of 1-mil Symbols Flow Diagram

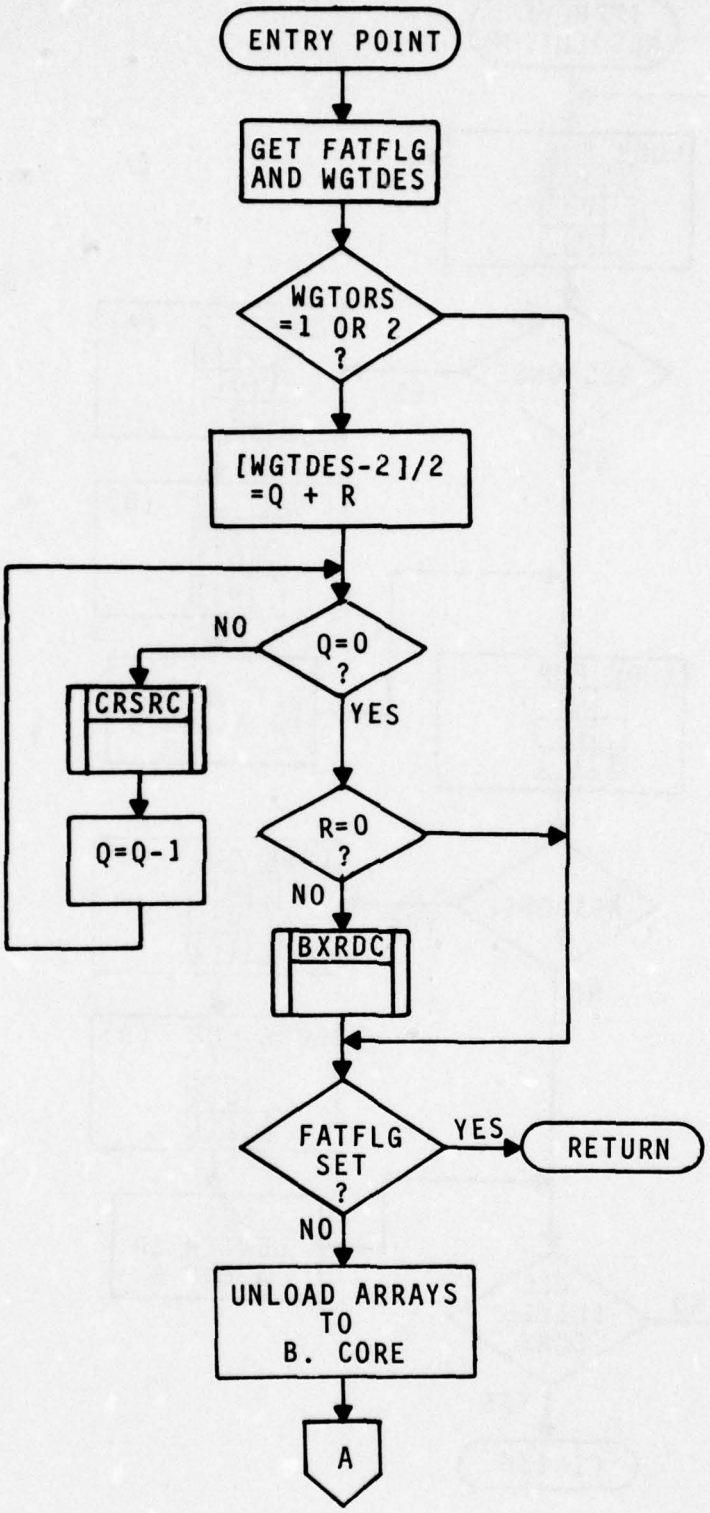


Figure 3-33. Line Separation Routine Flow Diagram
(sheet 1 of 2)

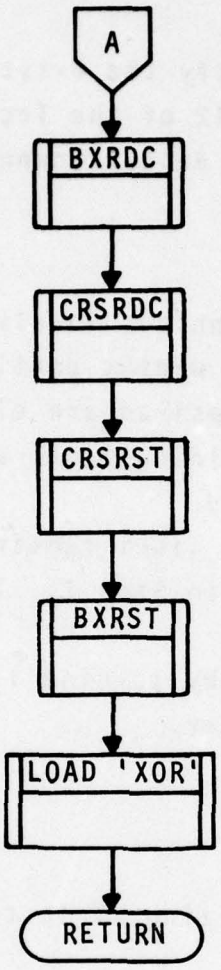


Figure 3-33. Line Separation Routine Flow Diagram
(sheet 2 of 2)

3.16.2 Approach

The approach was to modify the existing Line Separation Process described in Section 3.12 of the Technical Report entitled Associative Array Processing of Raster Scanned Data for Automated Cartography.

3.16.3 Implementation

The line separation technique involves.

1. Reducing all line widths until lines smaller than the line weight desired are eliminated.
2. Reducing the remaining lines until the desired line is eliminated.
3. "Subtracting" the lines remaining in Step 2 from the set produced in Step 1. The result is the line weight desired.
4. Editing the line by joining line breaks and eliminating clutter.

3.17 VECTORIZATION

3.17.1 Function

The function of vectorizing is to convert raster data to vector data.

3.17.2 Approach

The approach to this problem was to utilize the routines written during the last contract DAAK02-75-C-0114 with modifications to allow the processing of 4-mil, 2-mil, and 1-mil data.

3.17.3 Implementation

This subsection describes the changes to the routines documented in subsection 3.13 of the previous Technical Report, Associative Array Processing of Raster Scanned Data for Automated Cartography. The routines previously written were:

1. Boundary and End Point Tagging Routine.
2. Line Following Routine.
3. Vector Link Table Update Routine.
4. Master Vector List Creation Routine.

3.17.3.1 Array Boundary and End Point Tagging Routine - This routine has now been split into two Level 3 routines to facilitate the execution of these routines utilizing the alternating-paging approach discussed in subsection 3.3 (Executive Routine) of this report. The resulting two routines are:

a) Junction/Detection and End Point Tagging Routine (DTCTA)
b) Array Boundary Tagging Routine (DTCB)

3.17.3.1.1 DTCTA - This routine processes the thinned line data.

The routine detects and removes a 3 x 3 set of cells at all junction points and then tags and identifies, by position, all end points. A detailed flow diagram is shown in Figure 3-34; also, a detailed description follows.

An initial check is made to determine if the line data in the array(s) coincides with data along any edge of the overlay being processed. In this case slices of data (equivalent to half of the overlap between data blocks) occurring along the corresponding edge are cleared. This is to ensure that any array vectors that cross these boundaries are not tagged as array boundary vectors.

The next step is to clear a frame around the data to be processed, namely word slices 0,1,2,3,252,253,254,255, and bit slices 0,1,---31. This provides working storage and locations for the results.

Two bit slices (254,255) are now set up to denote those array words which will participate in the junction point or end point detection process.

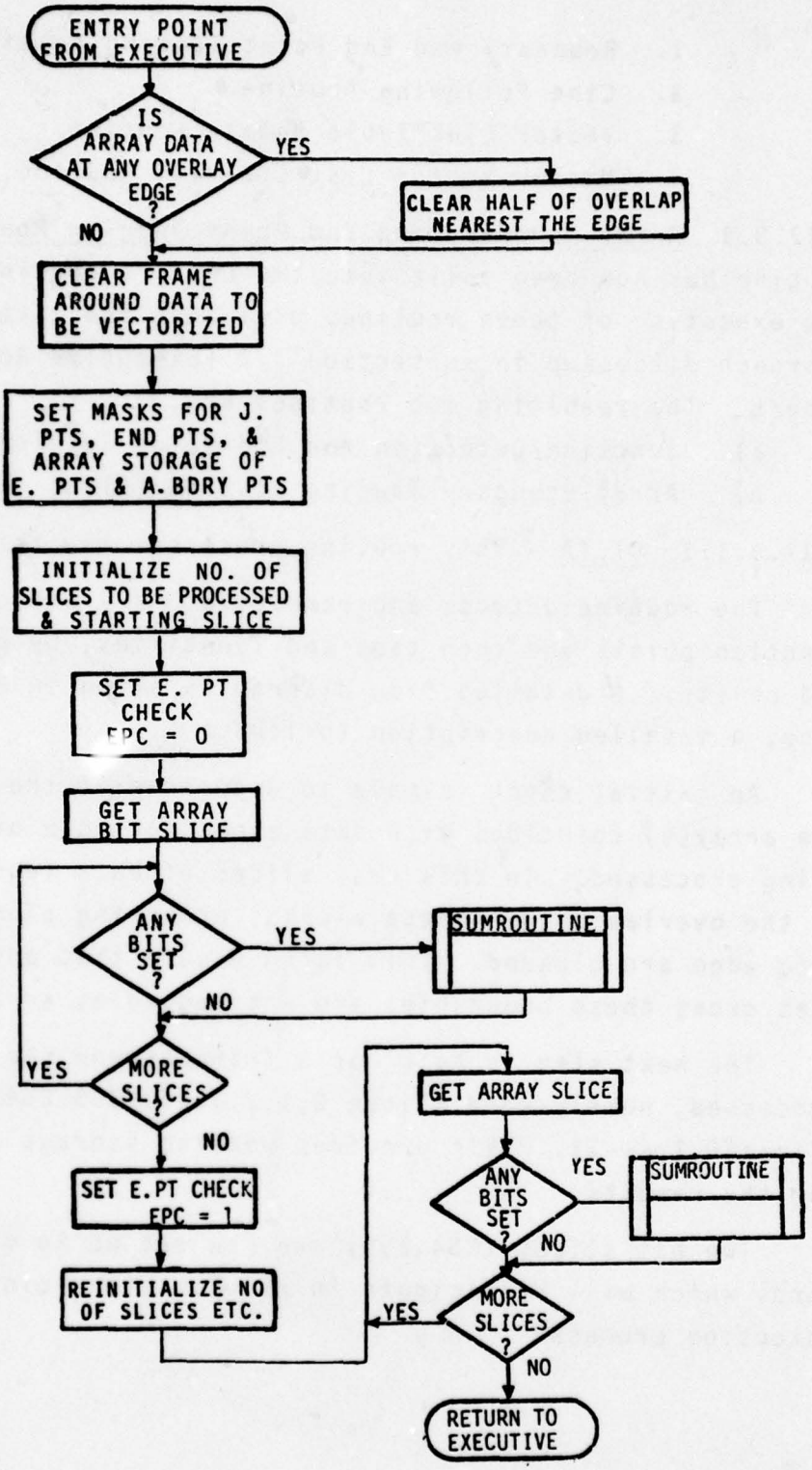


Figure 3-34. Detailed Flow Diagram of DTCTA (sheet 1 of 3)

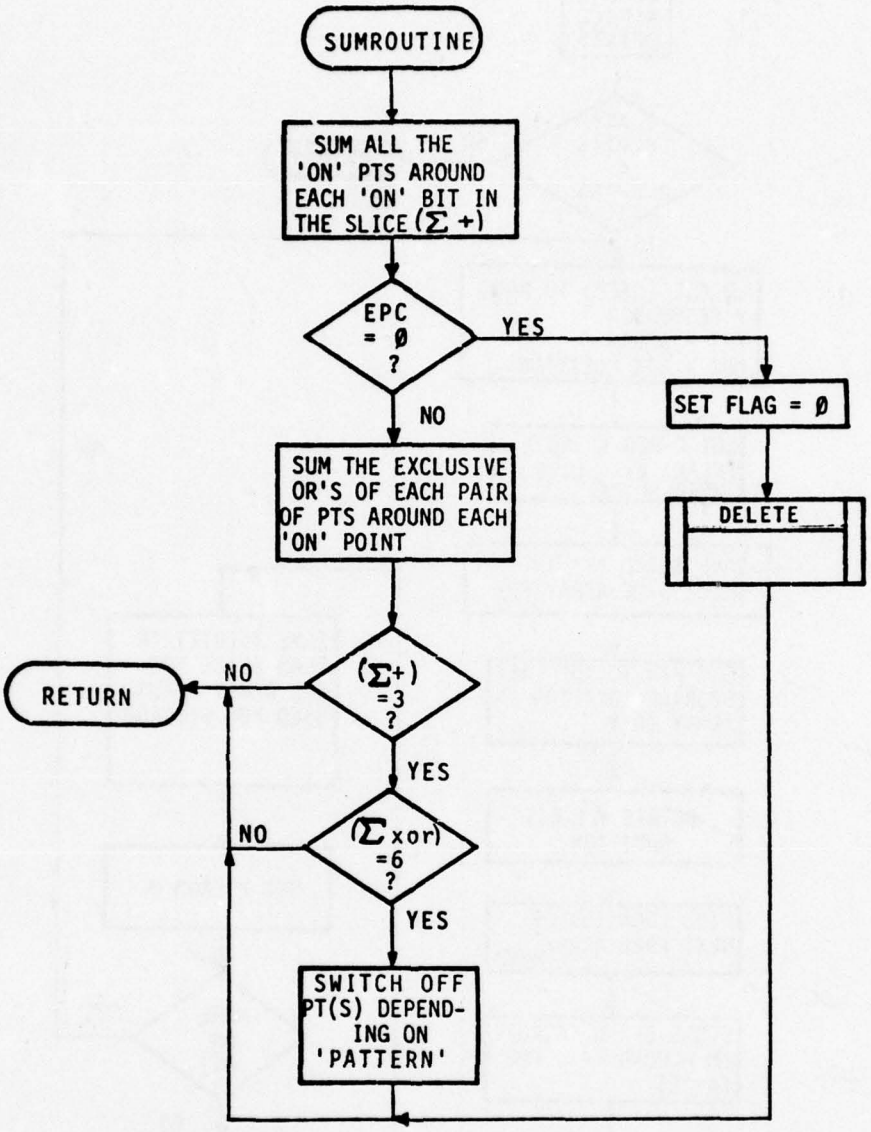


Figure 3-34. Detailed Flow Diagram of DTCTA
(sheet 2 of 3)

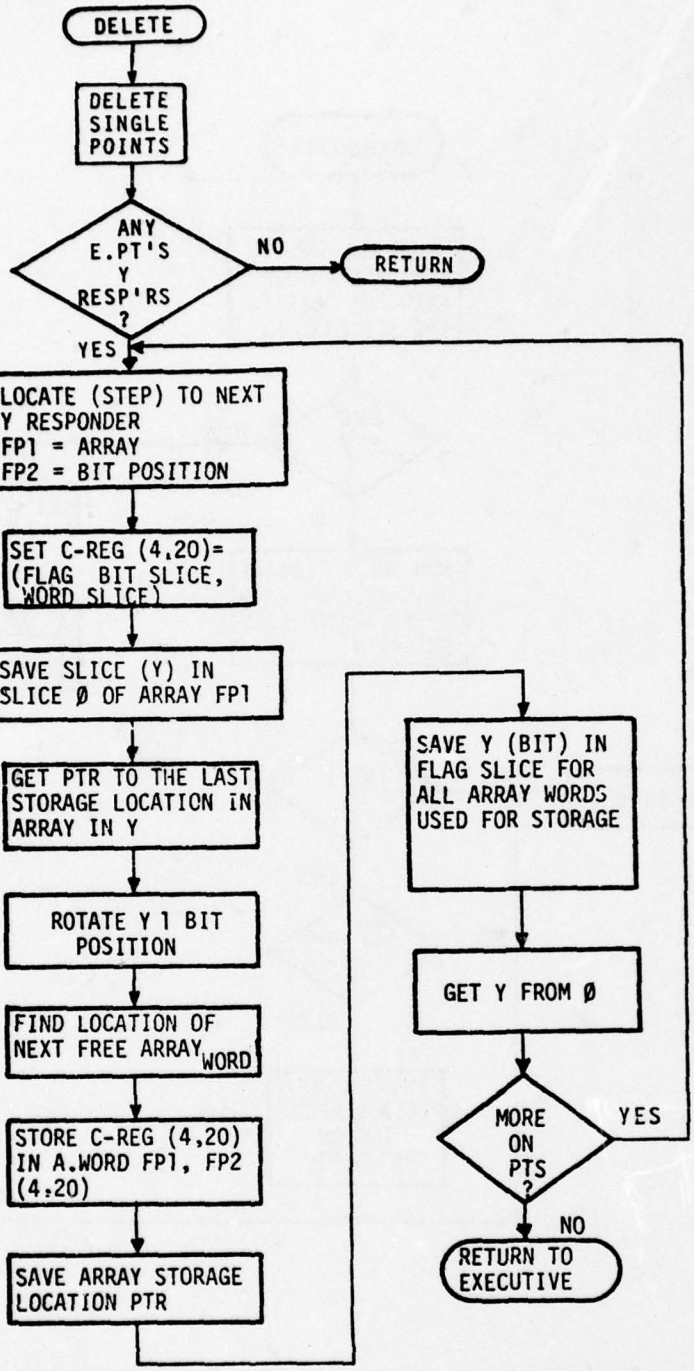


Figure 3-34. Detailed Flow Diagram of DTCTA

(sheet 3 of 3)

The criteria for the existence of a junction point is that:

1. The center point is 'ON' (i.e., all).
2. The sum of the surrounding eight points is three or four.
3. The sum of the 'logical OR' between each pair of surrounding points is six.

The next step is to locate and tag all the end points by their word and bit slice positions. Again, the process is the same as for 'line editing' except that the number of surrounding points is one (the sum of the logical OR's of each pair is automatically two), as shown in Figure 3-35. While the end points are being located any 'spots' which have no surrounding points are edited out of the array.

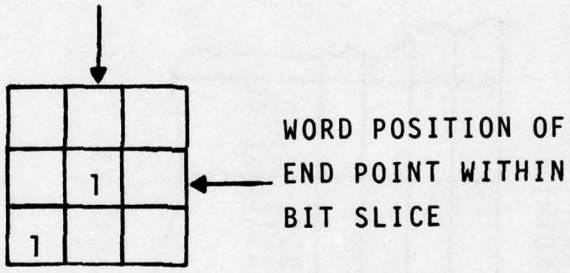


Figure 3-35. End Point of Line

The bit slice and word positions of each end point are then stored in consecutive array words in the field positions shown in Figure 3-36. Each end point is also tagged by setting its corresponding bit in the end point flag slice (EM) and end point or array boundary vector slice (MM). These flag slices are used by the line following routine. The resulting array layout of the data after tagging is shown in Figure 3-37.

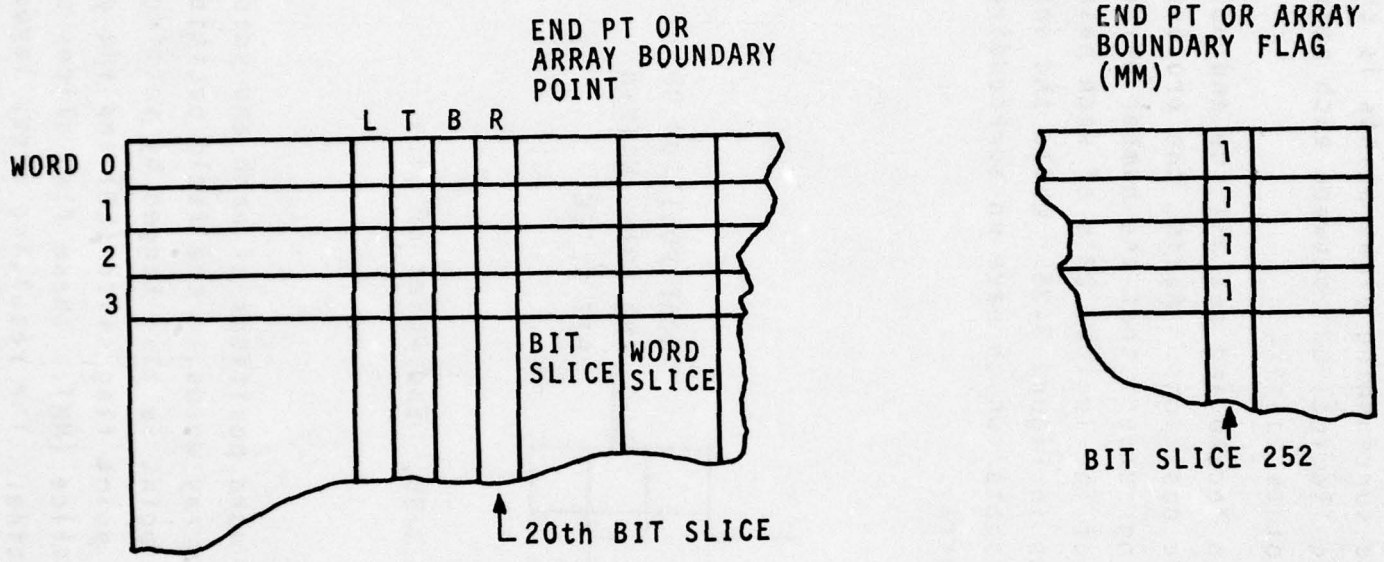


Figure 3-36. Array Layout After Tagging

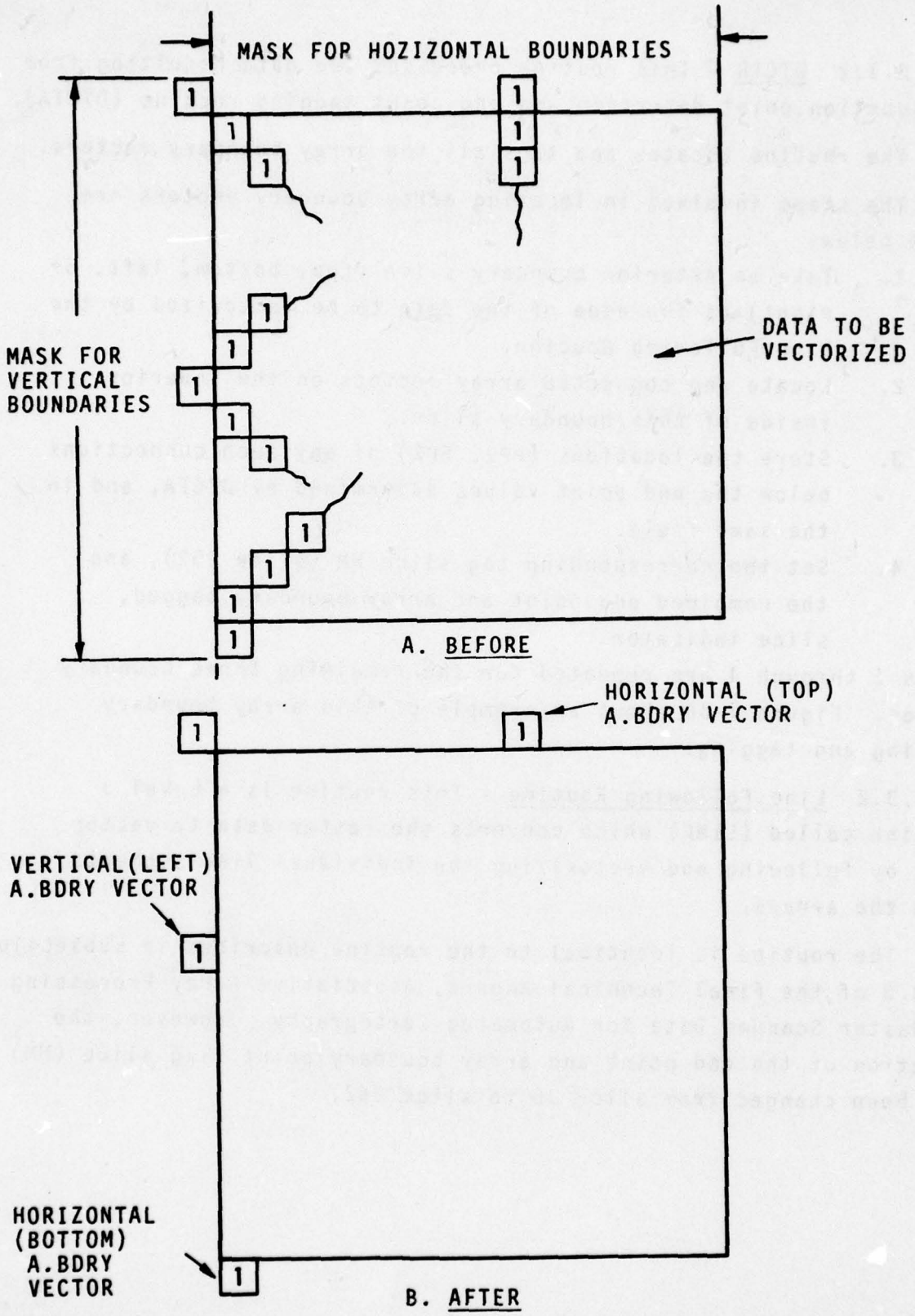


Figure 3-37. Array Boundary Tagging

3.17.3.1.2 DTCTB - This routine processes the data resulting from the junction point detection and end point tagging routine (DTCTA).

The routine locates and tags all the array boundary vectors.

The steps involved in locating array boundary vectors are given below:

1. Take an exterior boundary slice (top, bottom, left, or right) at the edge of the data to be vectorized by the Line Following Routine.
2. Locate any connected array vectors on the interior inside of this boundary slice.
3. Store the locations (FP1, FP2) of any such connections below the end point values determined by DTCTA, and in the same field.
4. Set the corresponding tag slice MM (slice 252), and the combined end point and array boundary tagged, slice indicator.

Steps 1 through 4 are repeated for the remaining three boundary slices. Figure 3-38 shows an example of this array boundary testing and tagging.

3.17.3.2 Line Following Routine - This routine is a Level 3 routine called (LINF) which converts the raster data to vector data by following and vectorizing the individual line segments with the arrays.

The routine is identical to the routine described in subsection 3.13.3 of the Final Technical Report, Associative Array Processing of Raster Scanned Data for Automated Cartography. However, the location of the end point and array boundary point flag slice (MM) has been changed from slice 36 to slice 252.

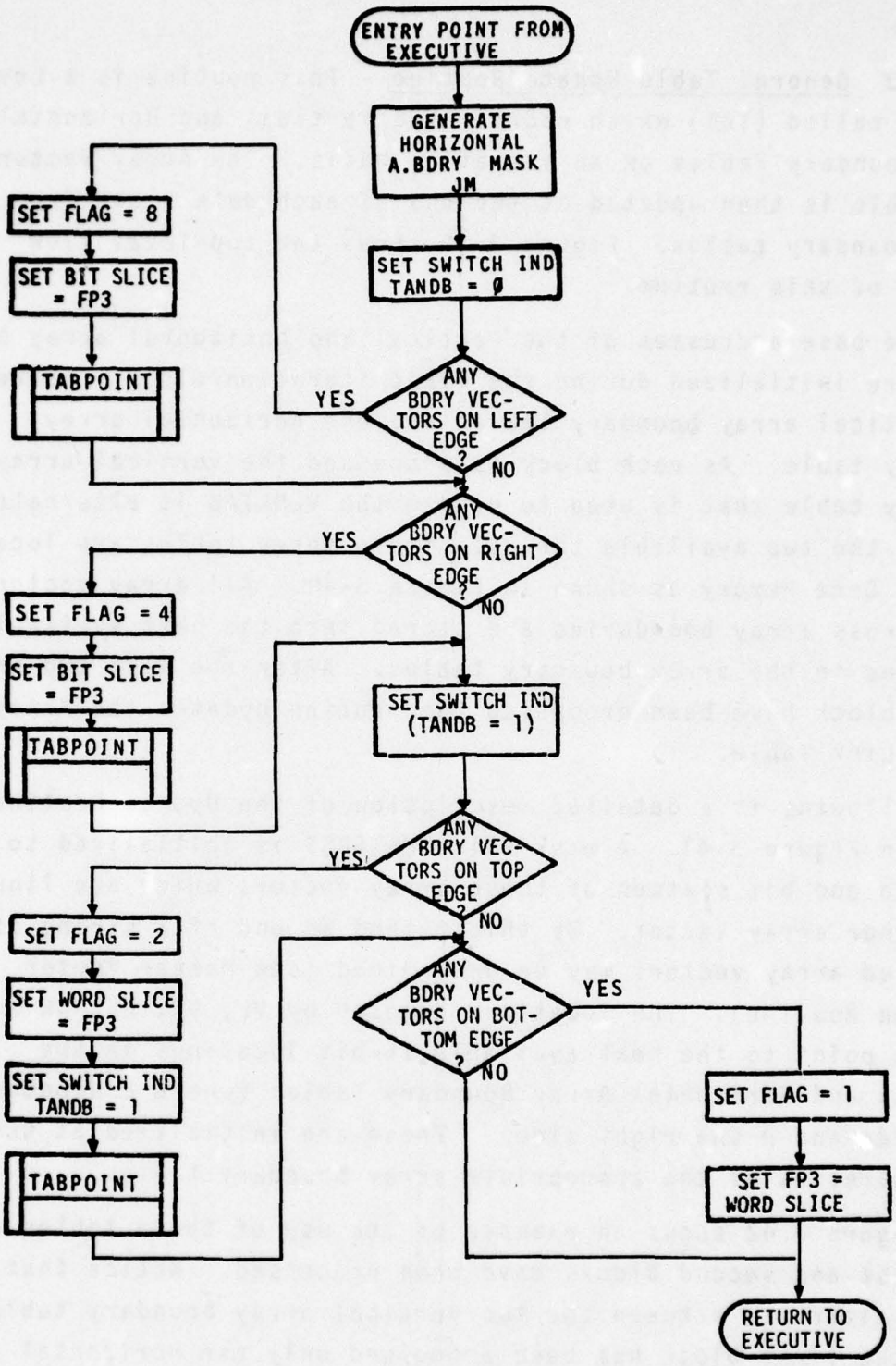


Figure 3-38. Detailed Flow Diagram of DTCTB

3.17.3.3 General Table Update Routine - This routine is a Level 3 routine called (TAB) which updates the Vertical and Horizontal Array Boundary Tables on an iteration basis. The Array Vector Link Table is then updated at the end of each data block from these boundary tables. Figure 3-39 shows the top-level flow diagram of this routine.

The base addresses of the vertical and horizontal array boundaries are initialized during the first iteration/block. There are two vertical array boundary tables and one horizontal array boundary table. As each block is processed the vertical array boundary table that is used to update the VLNKTAB is alternated between the two available tables. These three tables are located in Bulk Core Memory as shown in Figure 3-40. All array vectors which cross array boundaries are stored into the next available locations in the array boundary tables. After the last two arrays in the block have been processed the routine updates the Array Vector Link Table.

Following is a detailed description of the Update Routine, as shown in Figure 3-41. A mask slice MSKERST is initialized to reset bit zero and bit sixteen of those array vectors which are linked to another array vector. By this method an end of a string of connected array vectors may be determined (see Master Vector Creation Routine). The locations denoted by VL, VR, HL, HR are used to point to the next available 16-bit locations in the Vertical and Horizontal Array Boundary Tables (where L denotes the left side and R the right side). These are initialized at the base addresses of the appropriate array boundary tables.

Figure 3.42 shows an example of the use of these tables after the first and second blocks have been processed. Notice that VL and VR alternate between the two vertical array boundary tables. After the first block has been processed only the horizontal array boundary table is used to update the Link table. The array boundary tables are updated from the array vector numbers which were created during the Line Following Routine.

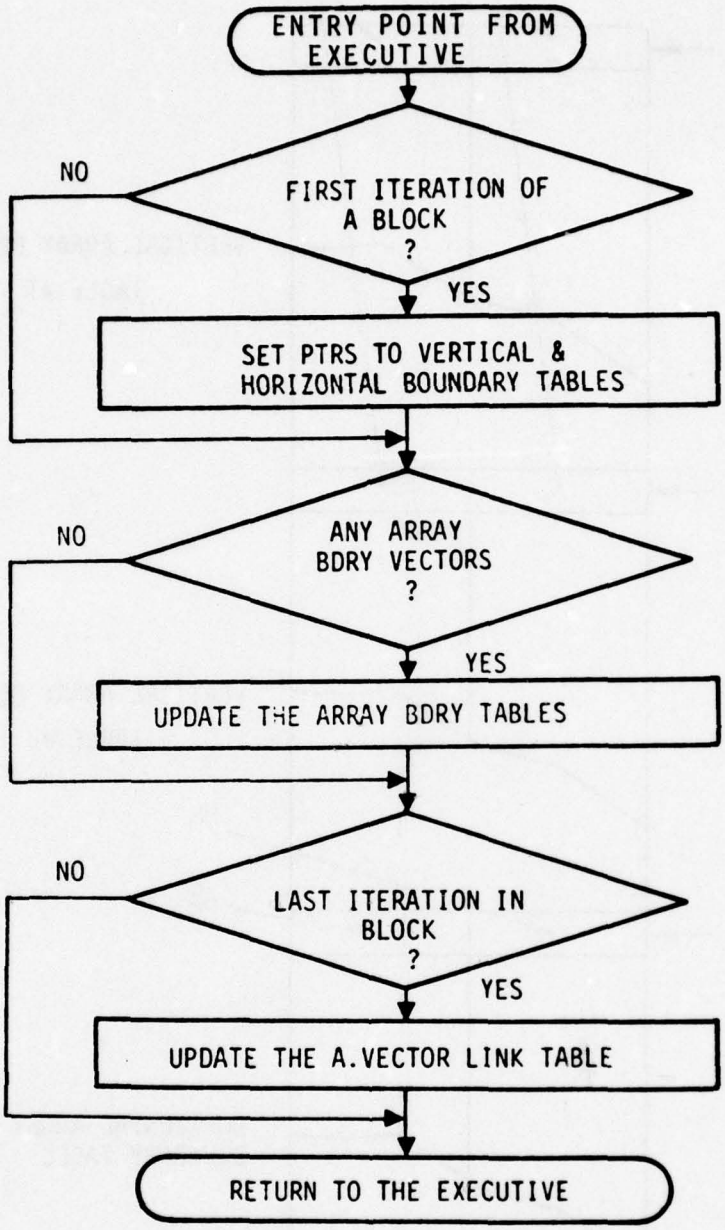


Figure 3-39. Top-Level Flow Diagram of General Table Update Routine

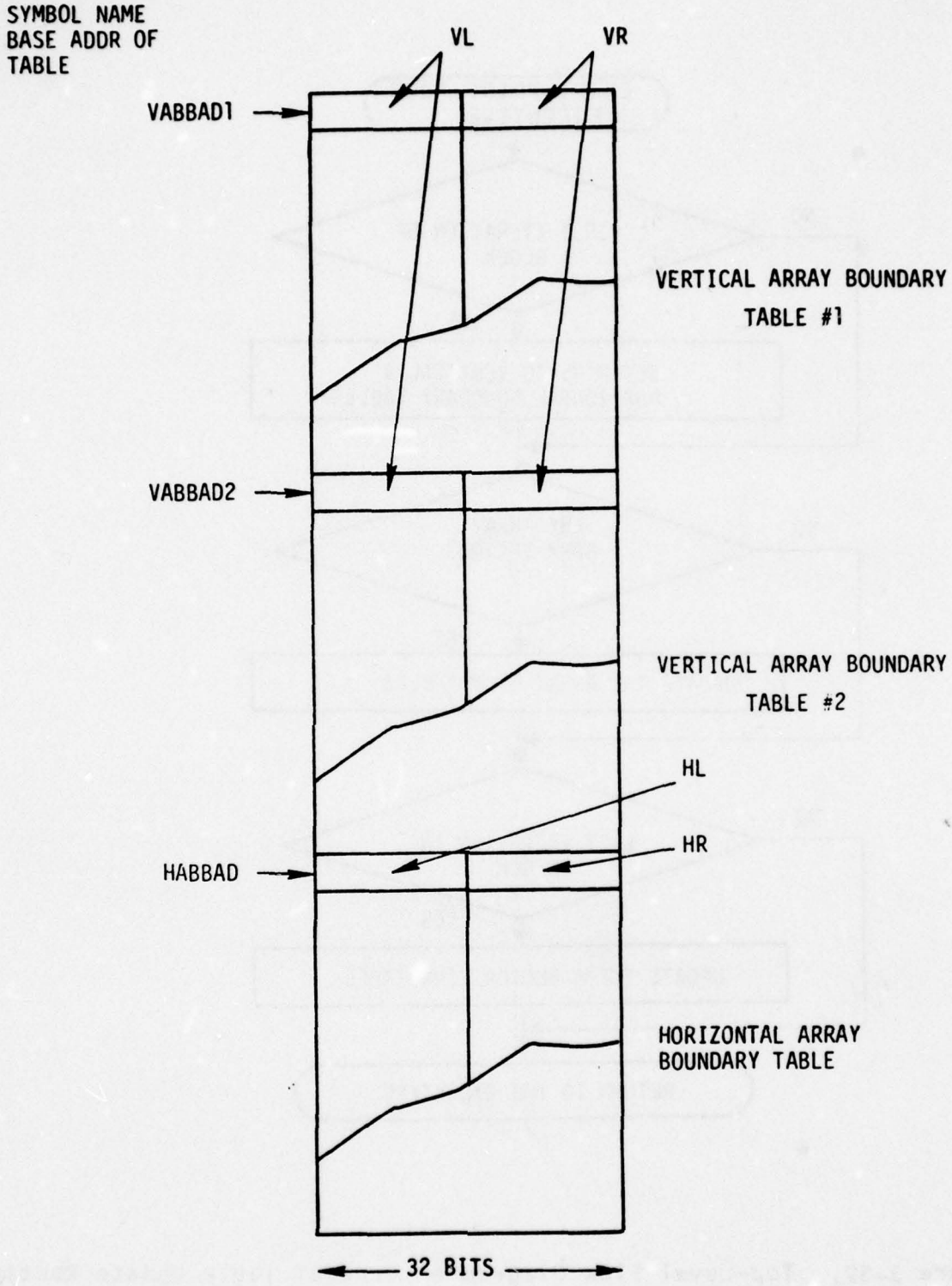


Figure 3-40. Array Boundary Table Layouts

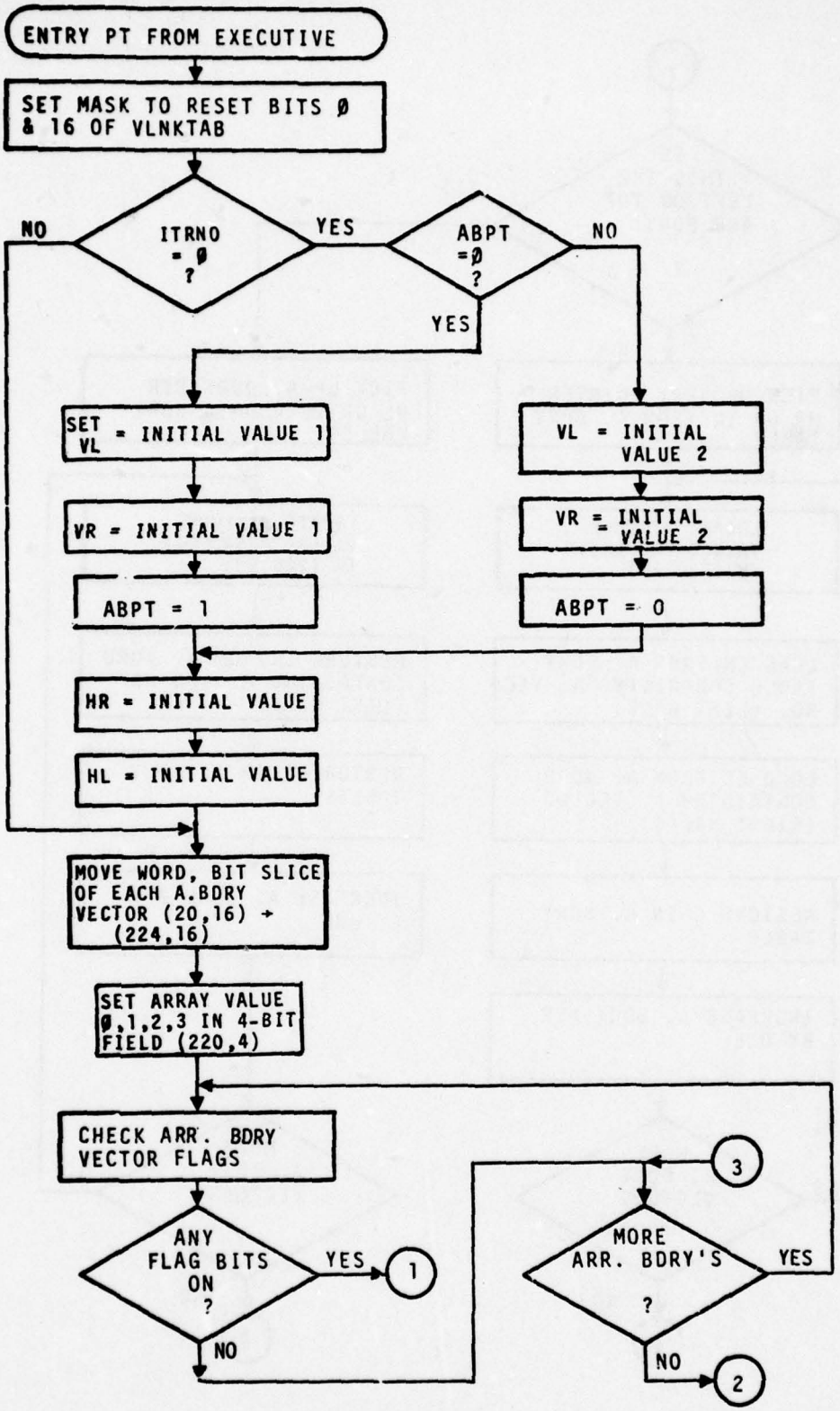


Figure 3-41. Flow Diagram of General Table Update Routine (sheet 1 of 3)

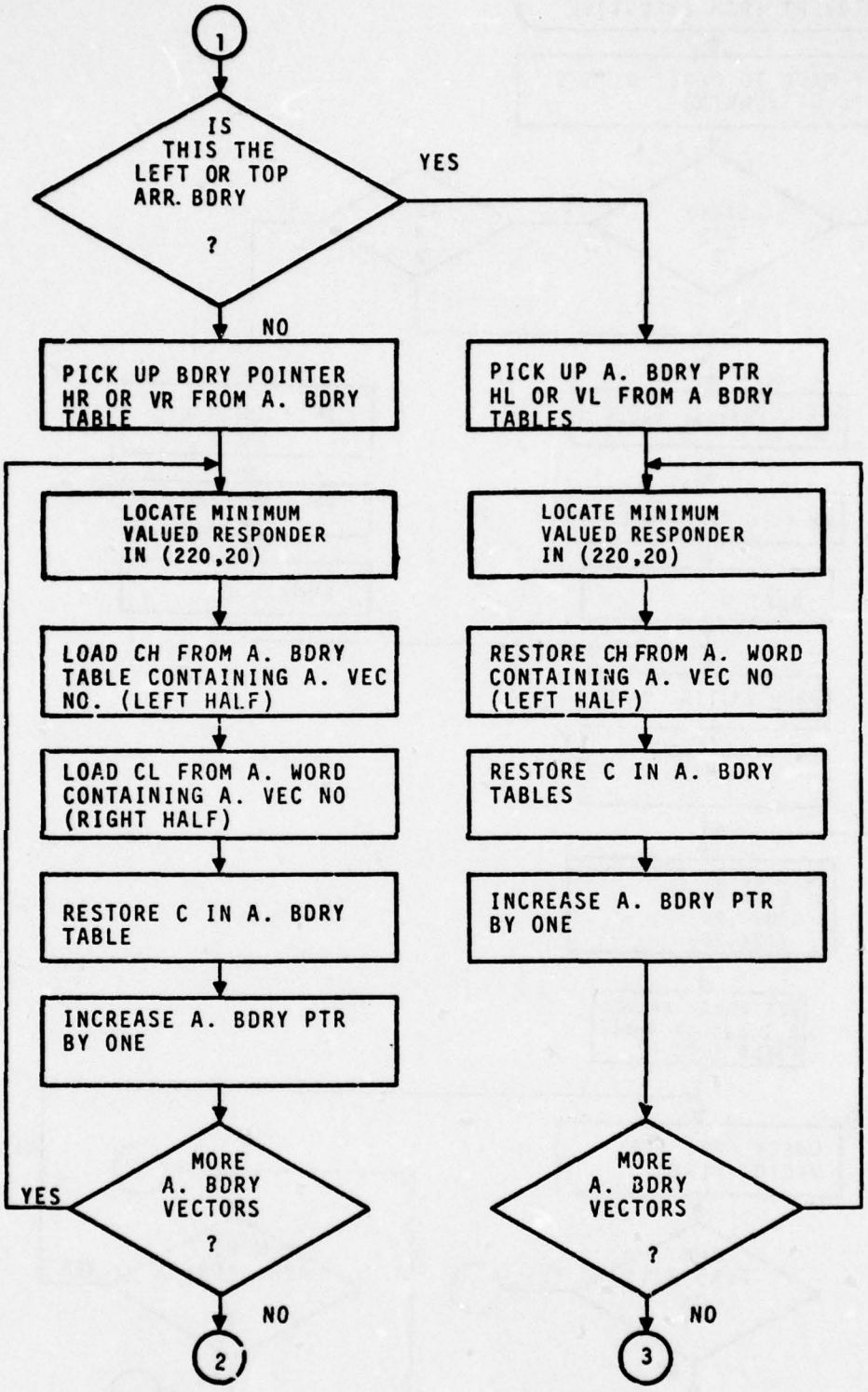


Figure 3-41. Flow Diagram of General Table Update Routine (sheet 2 of 3)

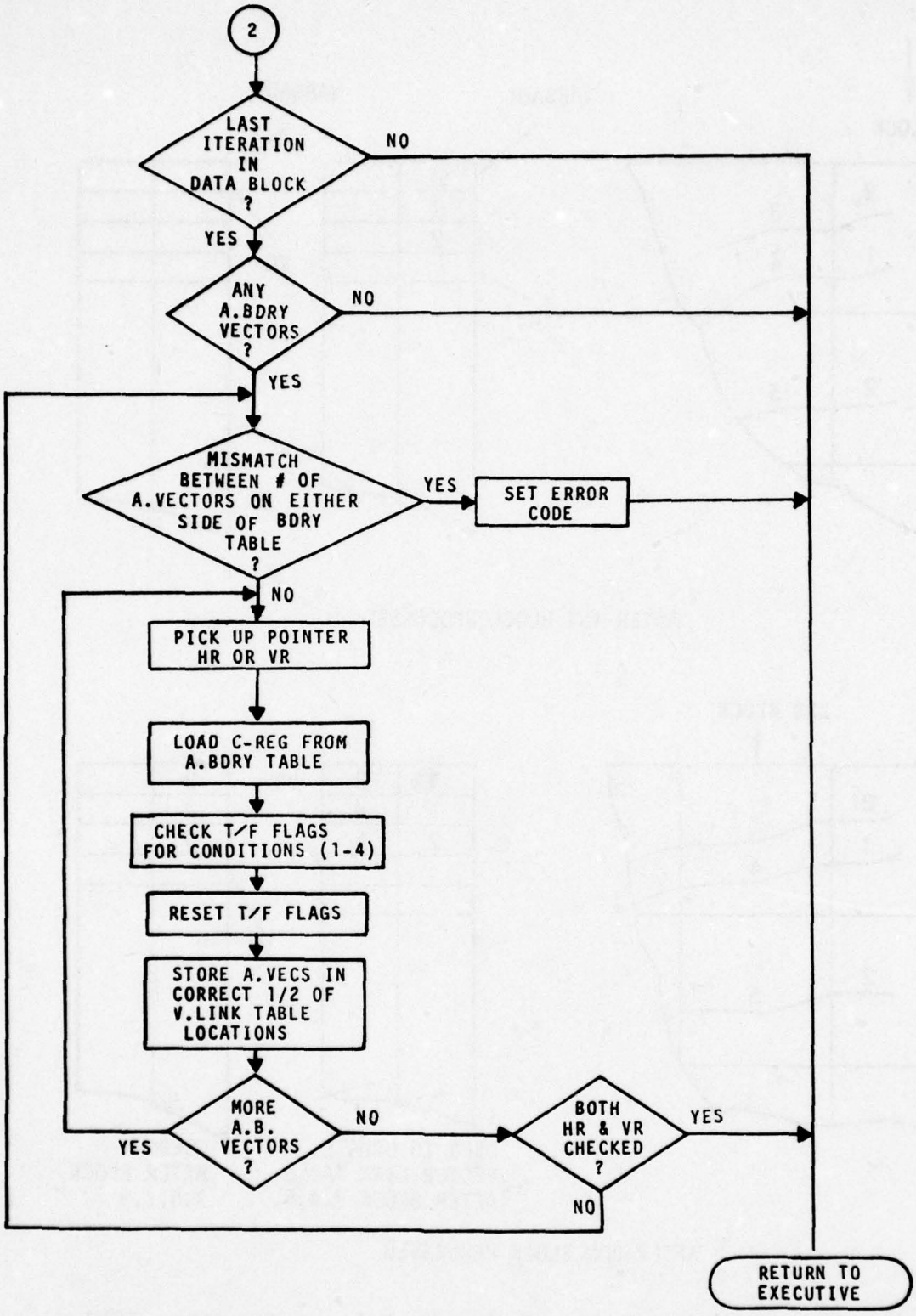
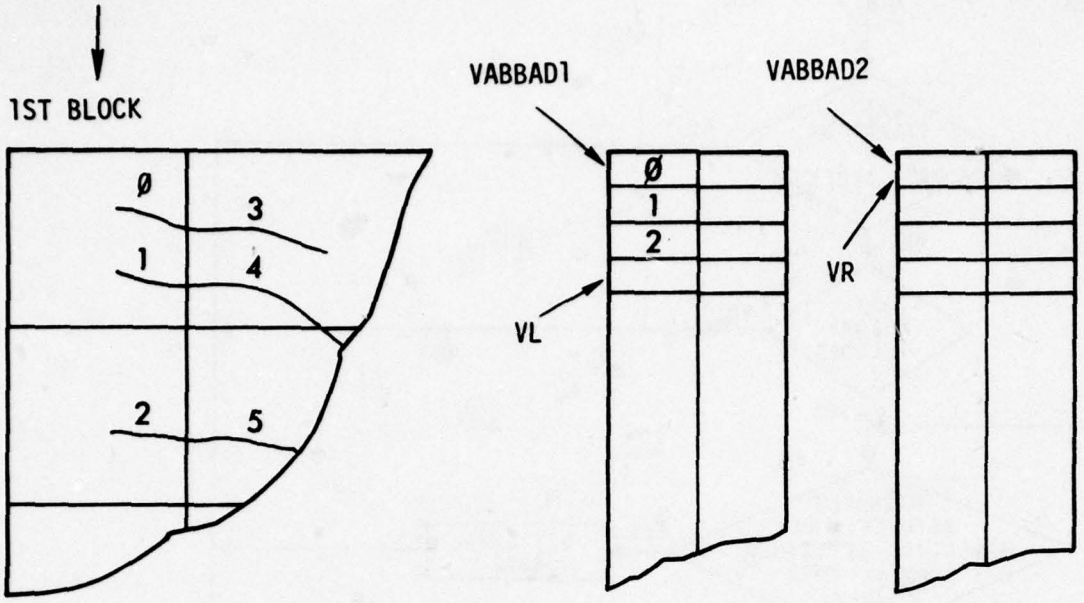
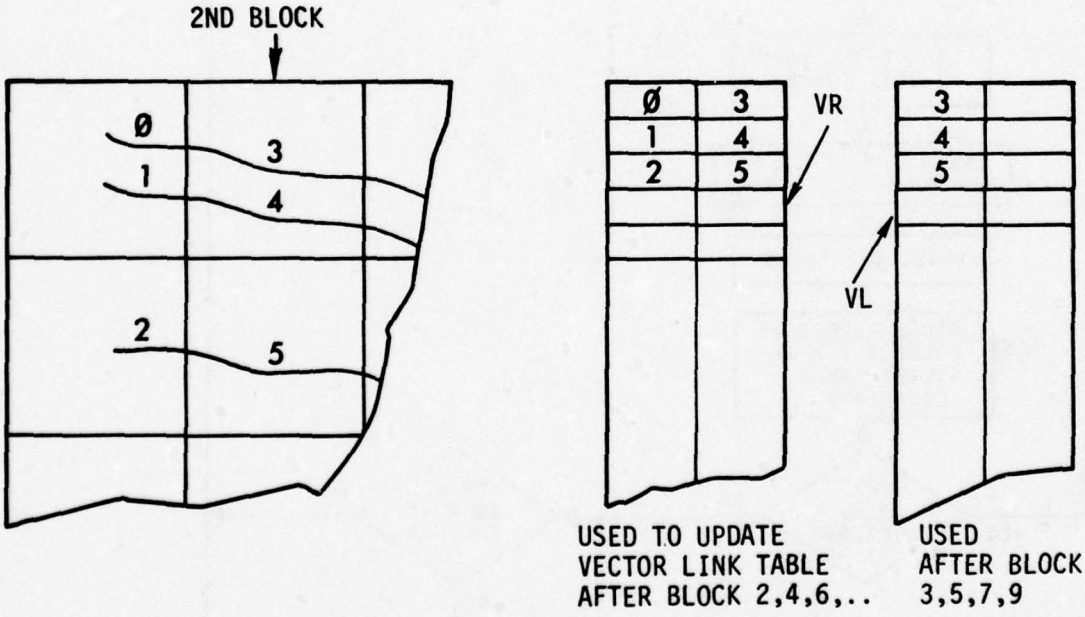


Figure 3-41. Flow Diagram of General Table Update Routine (sheet 3 of 3)



AFTER 1ST BLOCK PROCESSED



AFTER 2ND BLOCK PROCESSED

Figure 3-42. Example of use of Vertical Array Boundary Table

Figure 3-43 shows an array prior to this step. The array boundary table together with the sixteen bit half-word that will contain the array vector number is determined by the four array boundary flag slices denoted by L (left), T (top), B (bottom), and R (right).

If this is the last iteration in the data block the array vector link table (VLNKTAB) is updated from the boundary tables. The T/F flag bits denoted by bit positions 0 and 16 determine how the VLNKTAB is updated. Figure 3-44 shows an example of the four possible combinations of T/F values. The arrows on the array vectors A and B denote the direction in which they were vectorized.

Figure 3-45 shows how the VLNKTAB is updated for the four different cases. Prior to storing the array vectors into VLNKTAB the T/F flag bits are 'switched off' by utilizing the mask generated at the beginning of the routine. The steps are shown below:

1. Store array boundary table word in X response store.
2. Generate the mask in Y response store.
3. 'Logically AND' X with Y.
4. Output X bits 0 through 31.

Figure 3-46 shows this process. When all words within the array boundary tables have been used to update the VLNKTAB the routine is completed.

3.17.3.4 Master Vector Table Routine - This routine is a Level 0 routine called MVECT which generates the master vector list from the Array Vector Link Table. Figure 3-47 shows the top-level flow diagram of the Master Vector Table Routine. Following is a detailed description of the new routine. Figure 3-48 shows the associated Flow Diagram.

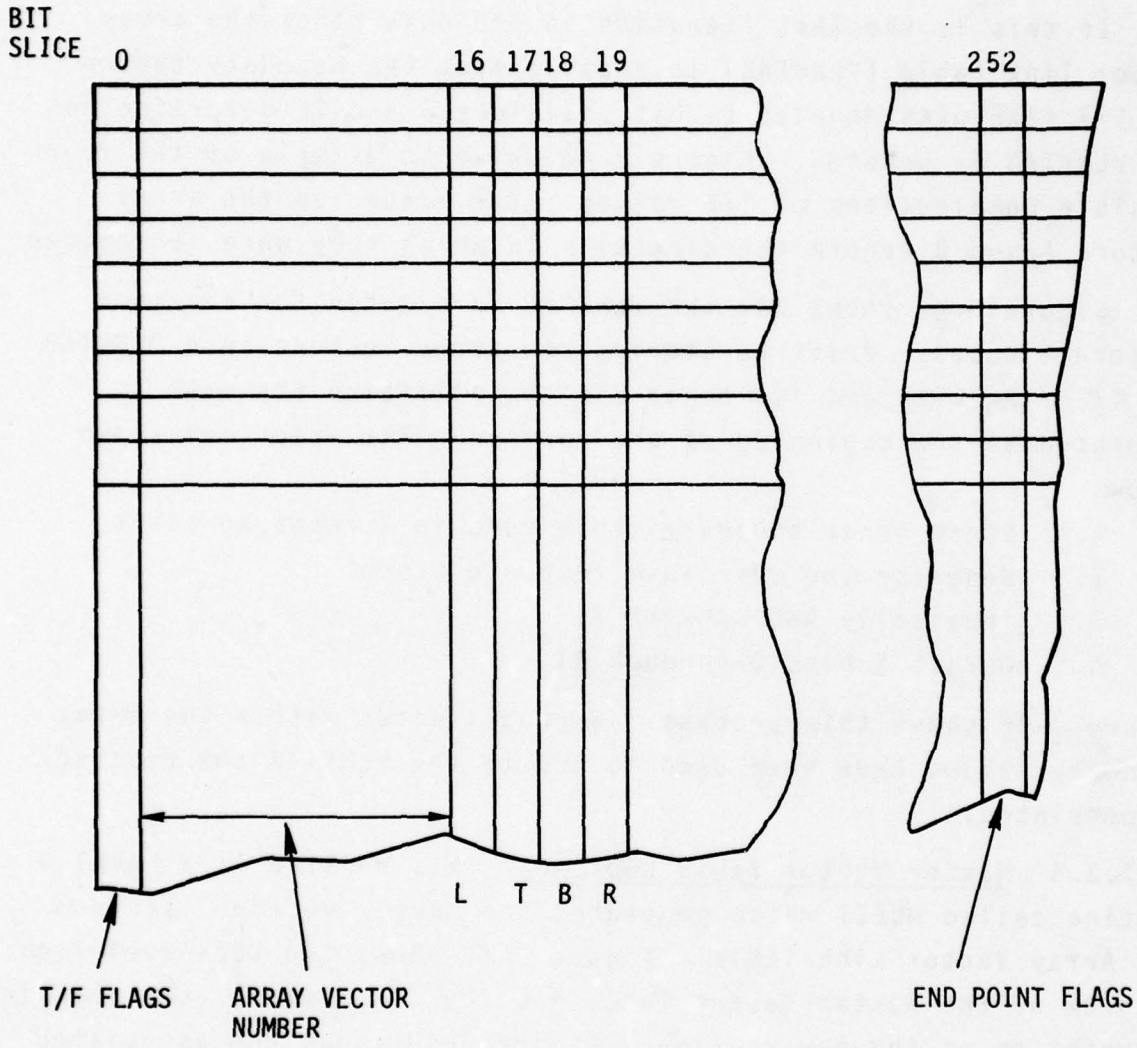


Figure 3-43. Array Layout prior to Updating the Array Vector Link Table

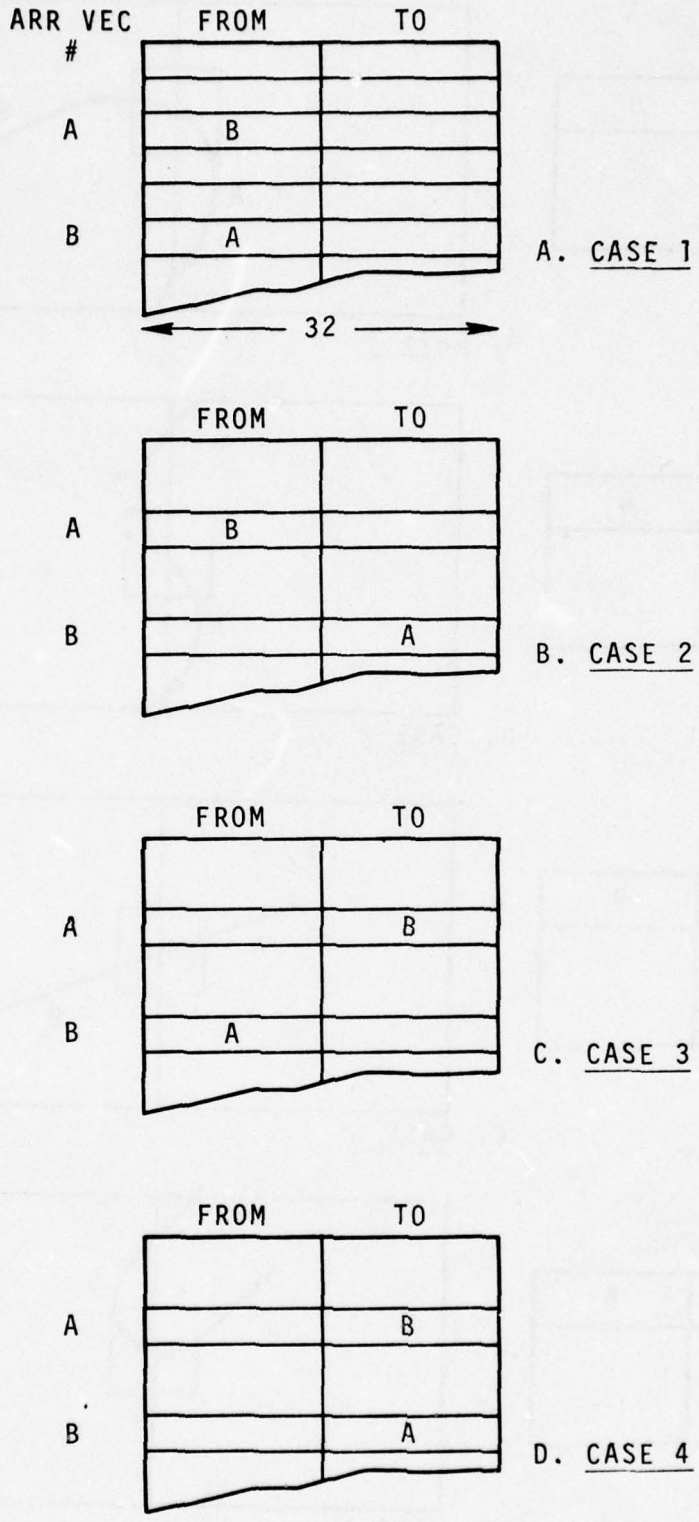


Figure 3-44. T/F Combinations

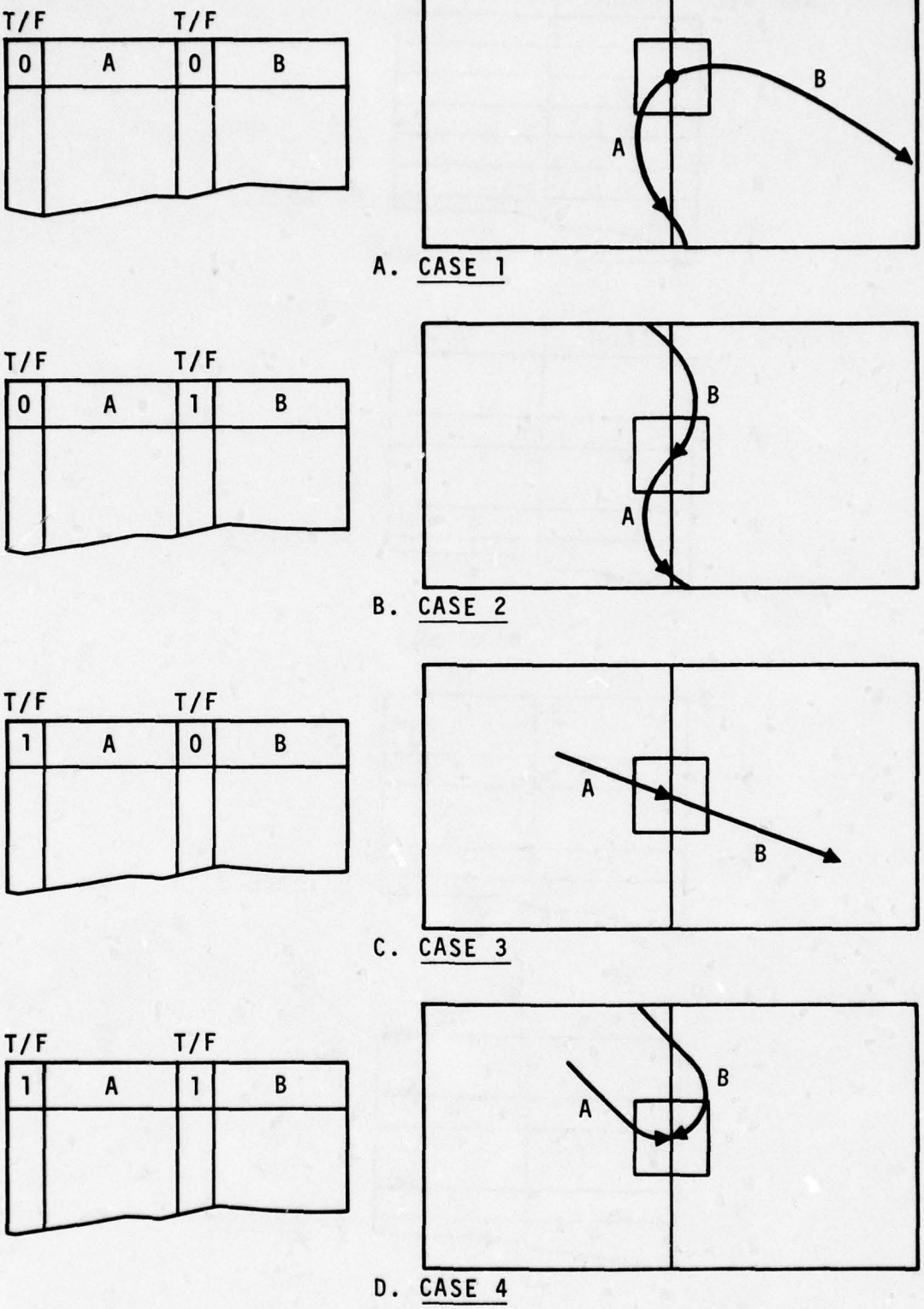
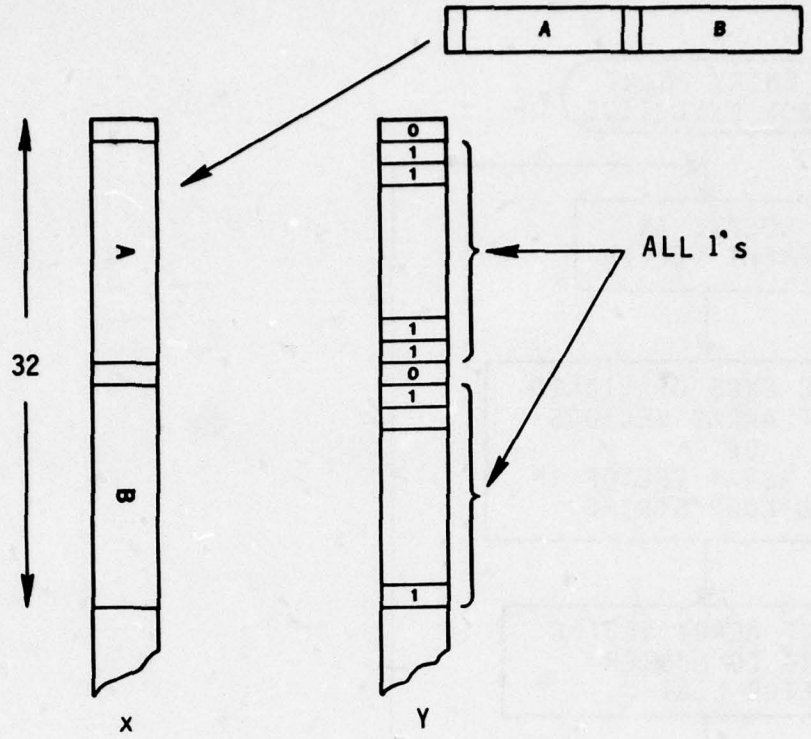
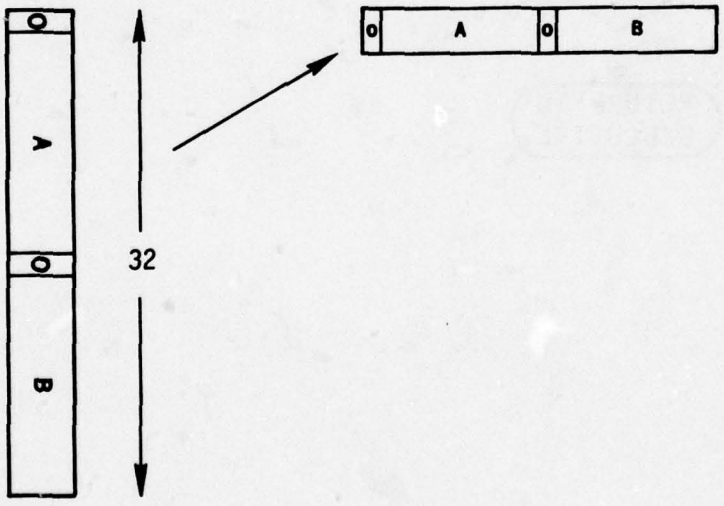


Figure 3-45. VLNKTAB updated for T/F combinations



A. STEPS 1 & 2



B. STEPS 3 & 4

Figure 3-46. Resetting Leftmost T/F Flag Array Vectors

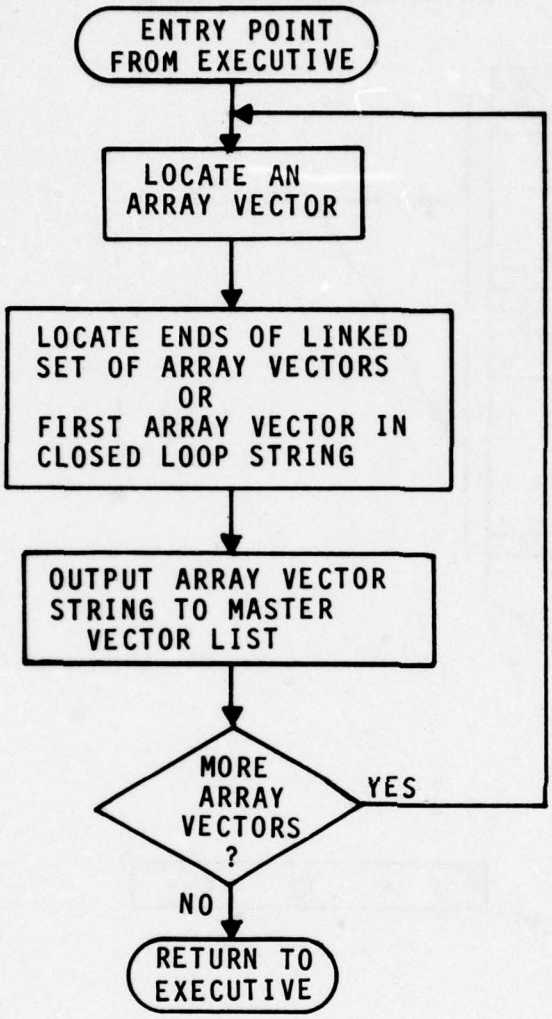


Figure 3-47. Top Level Diagram of Master Vector List Routine

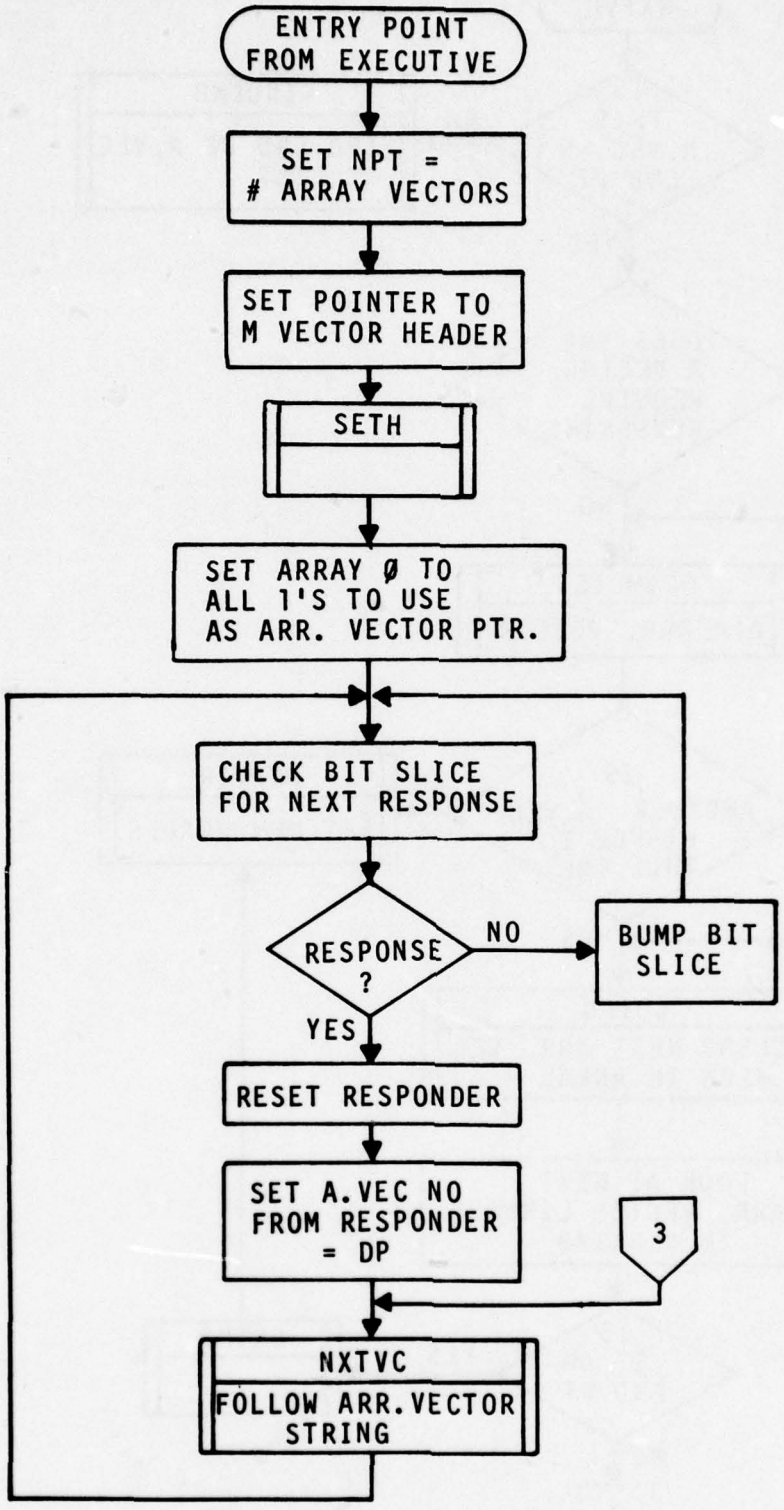


Figure 3-48. Flow Diagram for Master Vector Routine (sheet 1 of 8)

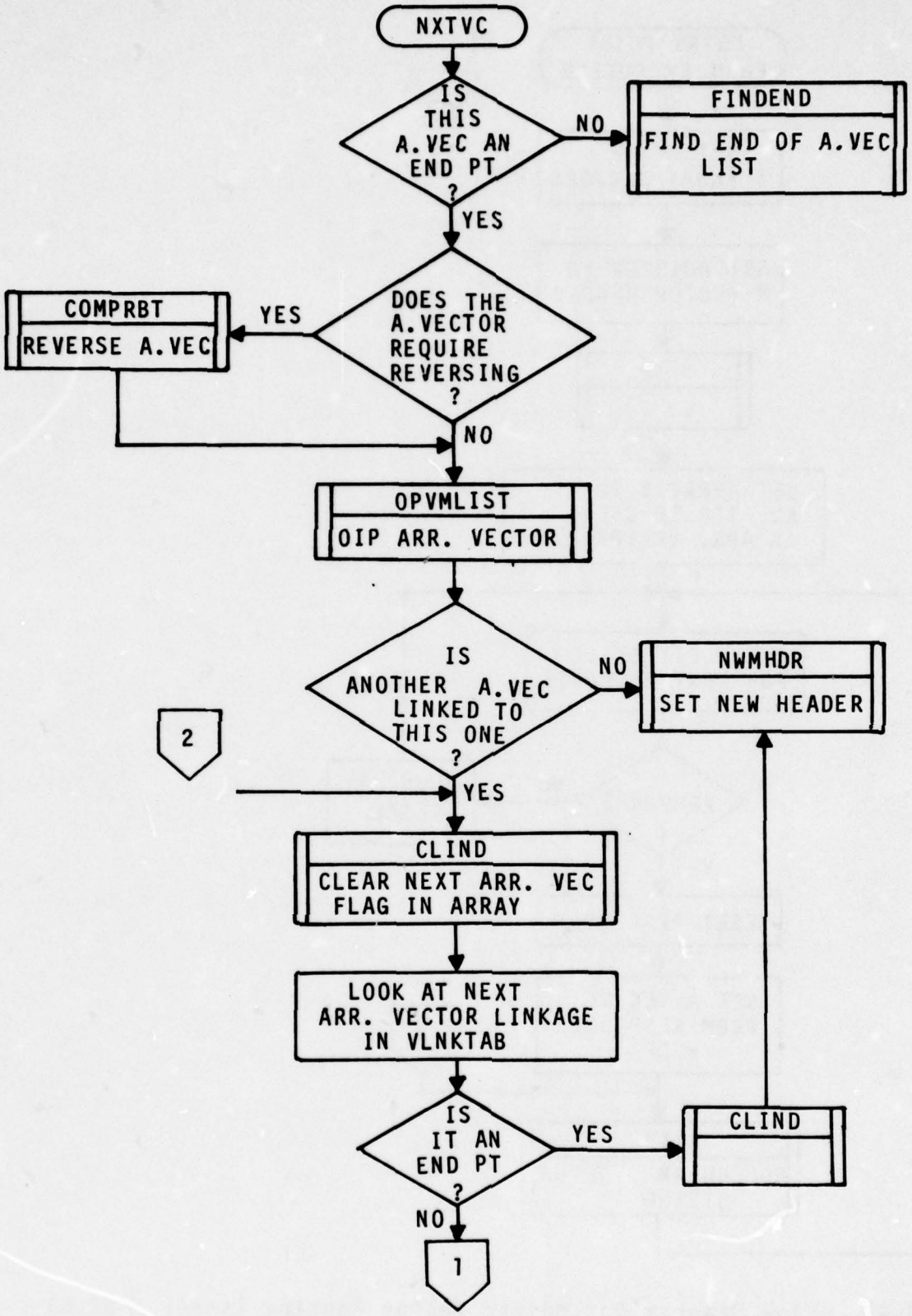


Figure 3-48. Flow Diagram for Master Vector Routine (sheet 2 of 8)

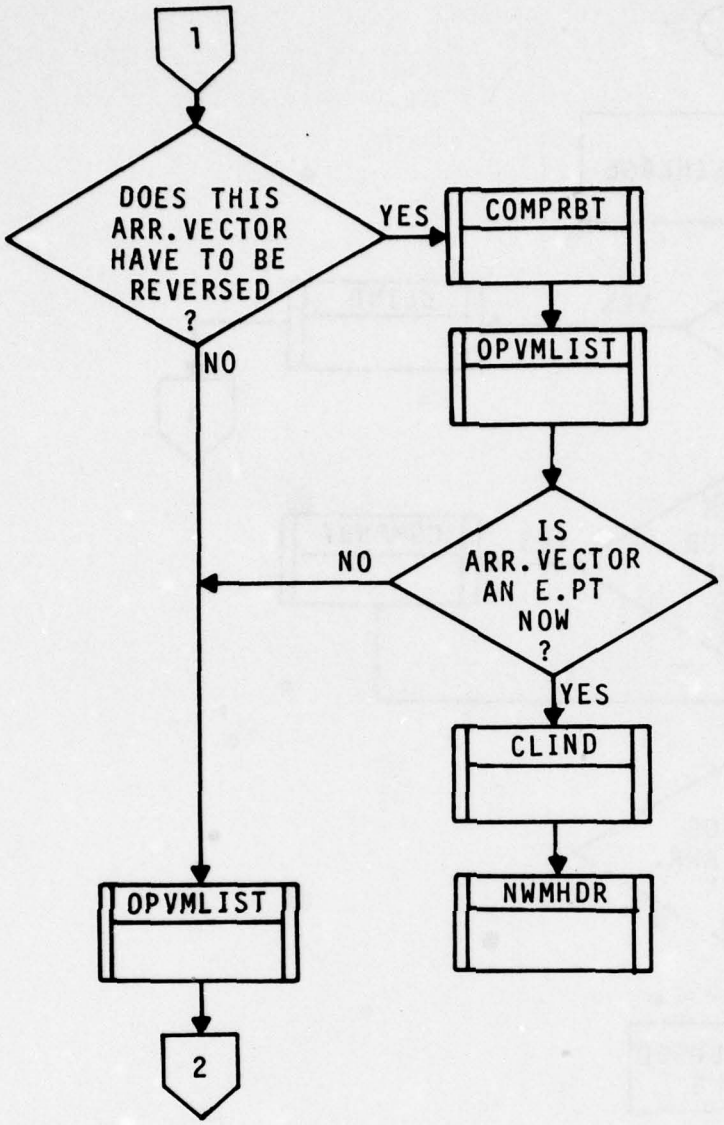


Figure 3-48. Flow Diagram for Master Vector Routine (sheet 3 of 8)

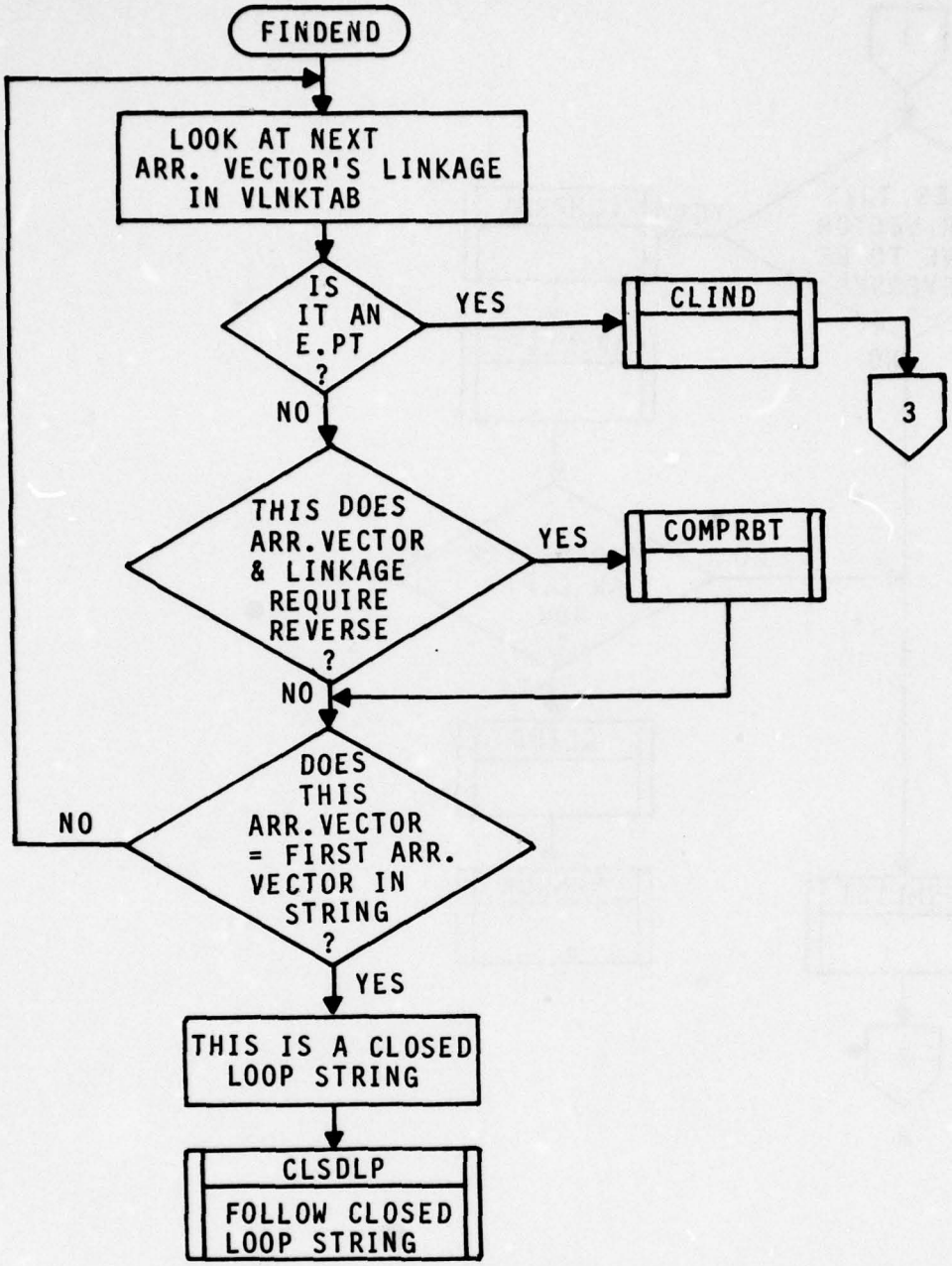


Figure 3-48. Flow Diagram for Master Vector Routine (sheet 4 of 8)

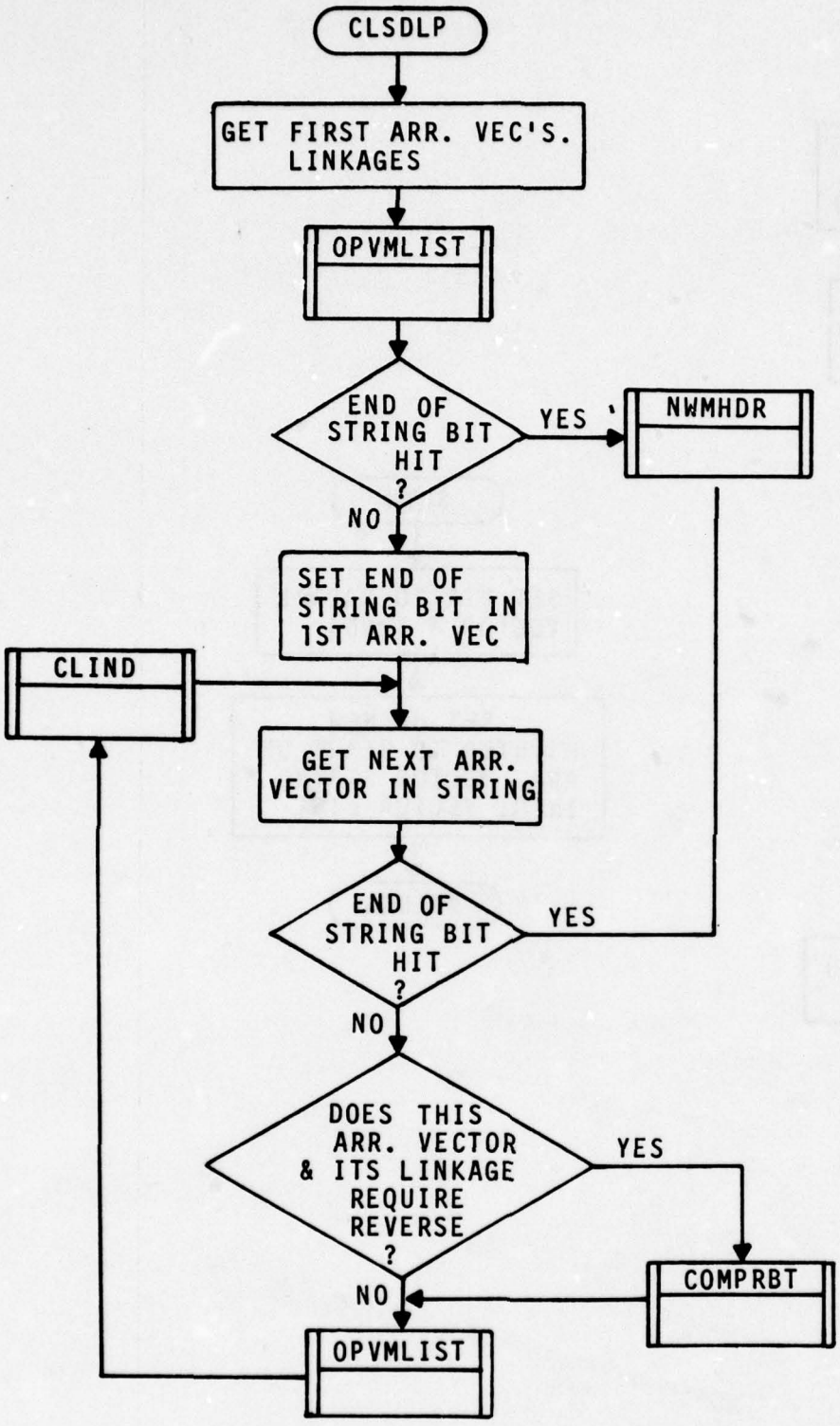


Figure 3-48. Flow Diagram for Master Vector Routine (sheet 5 of 8)

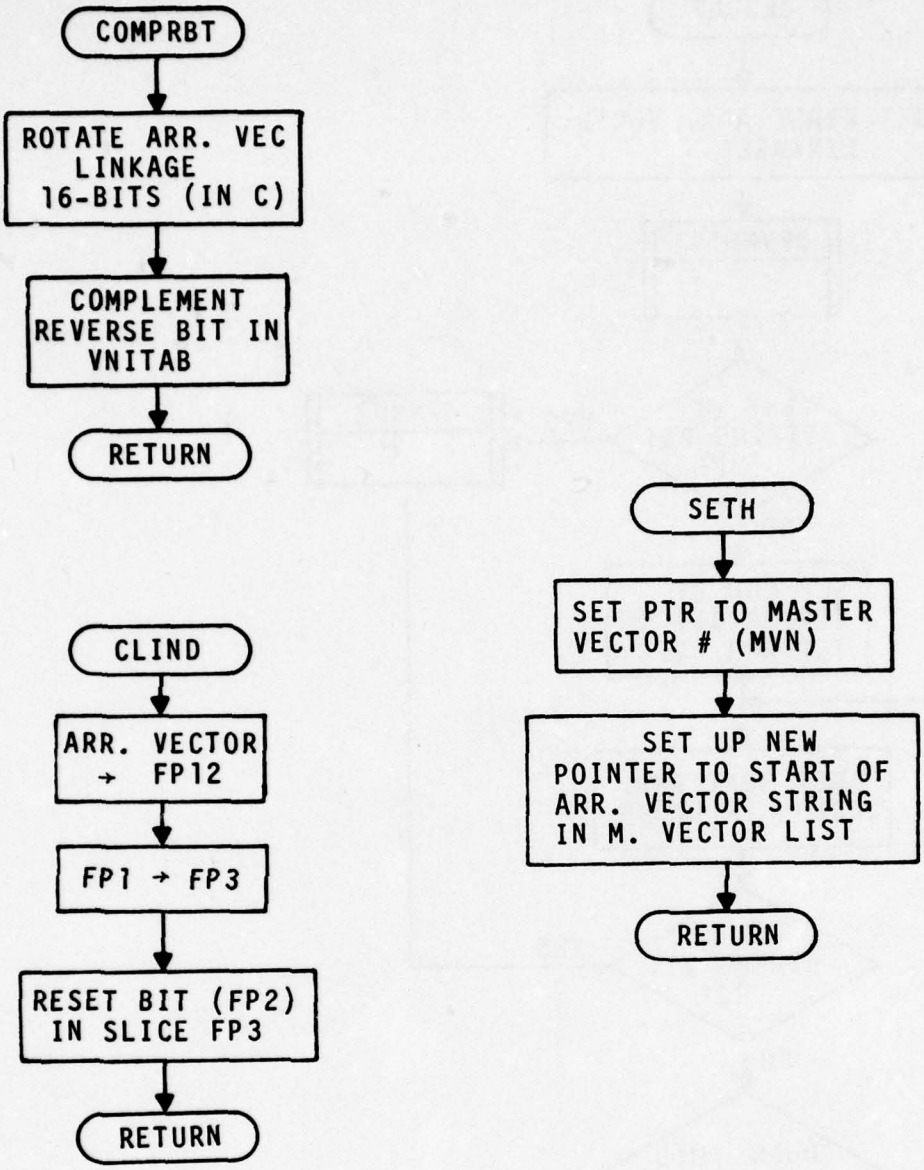


Figure 3-48. Flow Diagram for Master Vector Routine (sheet 6 of 8)

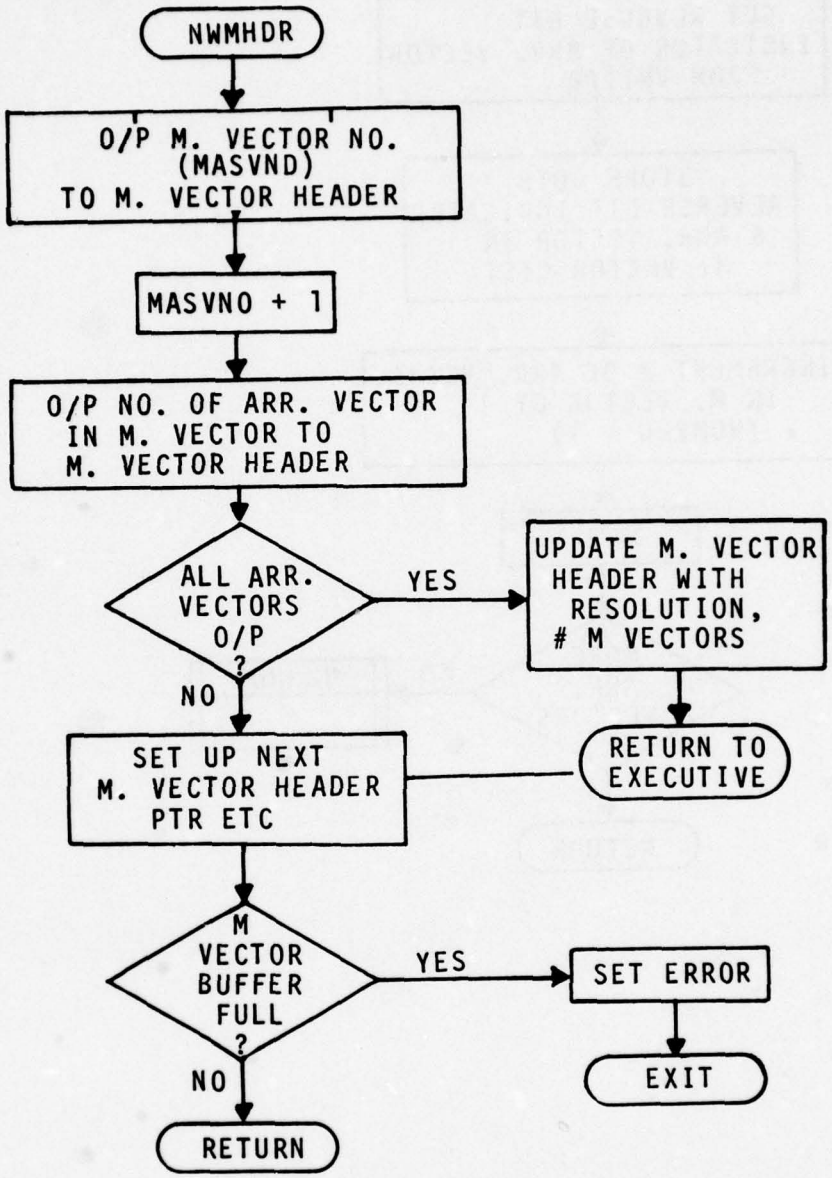


Figure 3-48. Flow Diagram for Master Vector Routine (sheet 7 of 8)

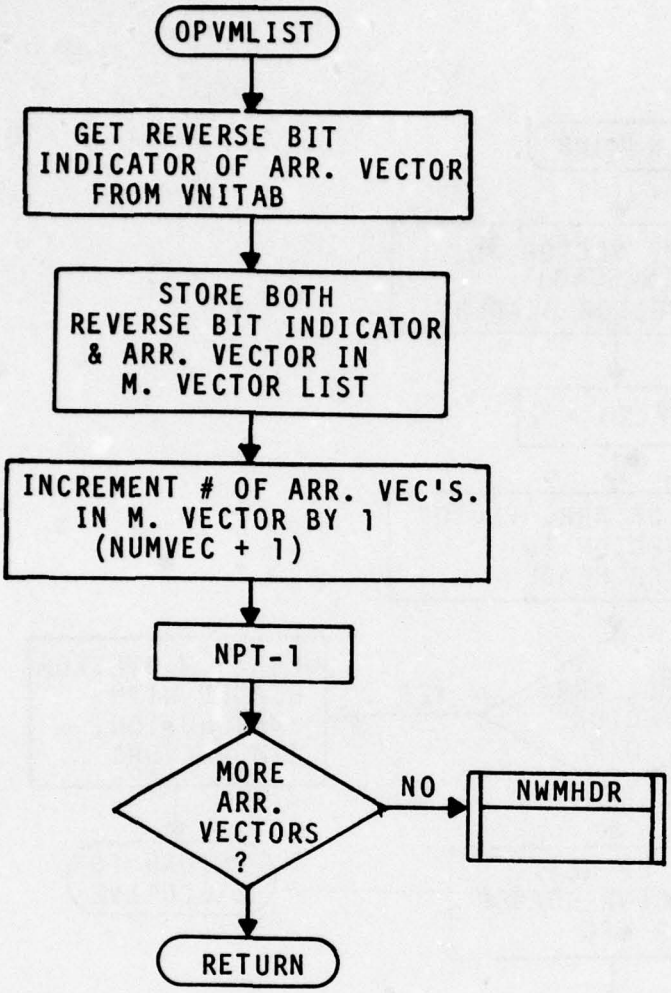


Figure 3-48. Flow Diagram of Master Vector Routine (sheet 8 of 8)

The operations involved in the generation of the Master Vector Table are as follows.

1. Initialize the number of array vectors generated (NPT). Then, set all bits in array 0 to one. These bits are used on a slice-by-slice basis to identify which array vectors have not been output as part of a master vector.
2. Locate the next array vector not output already to master vector list.
3. Follow the array vector linkage reversing array vectors where necessary (see Figure 3-49) until either:
   a) An end point is located
   b) The starting array vector in a closed loop string is hit for the second time (see Figure 3-50).
4. Output all the array vectors in this string to the master vector table, and identified as a unique master vector number.
5. Delete all the associated array vector flags from array 0.
6. Repeat steps 2 through 5 until all array vectors have been output (NPT = zero).
7. Update the header of the master vector table with the resolution, and the number of master vectors generated.

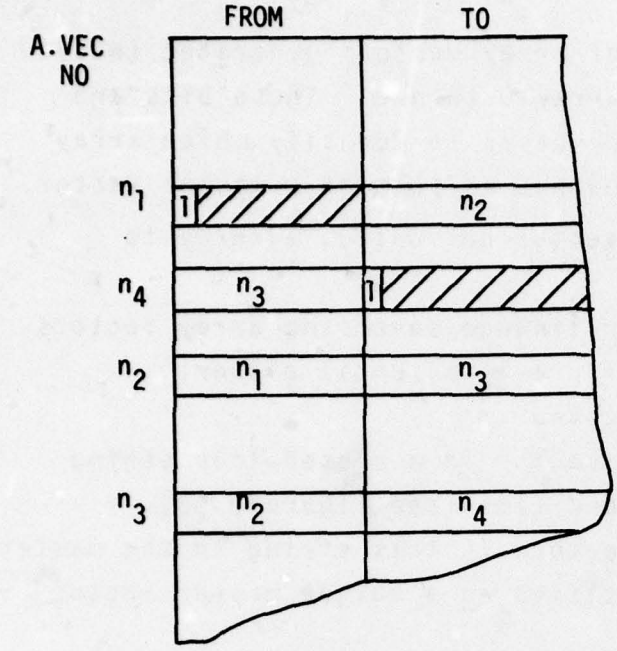
3.18 CONTOUR TAGGING INVESTIGATION

3.18.2 Function

The function of this investigation was to determine techniques for tagging vectorized contours generated from raster scanned contour maps.

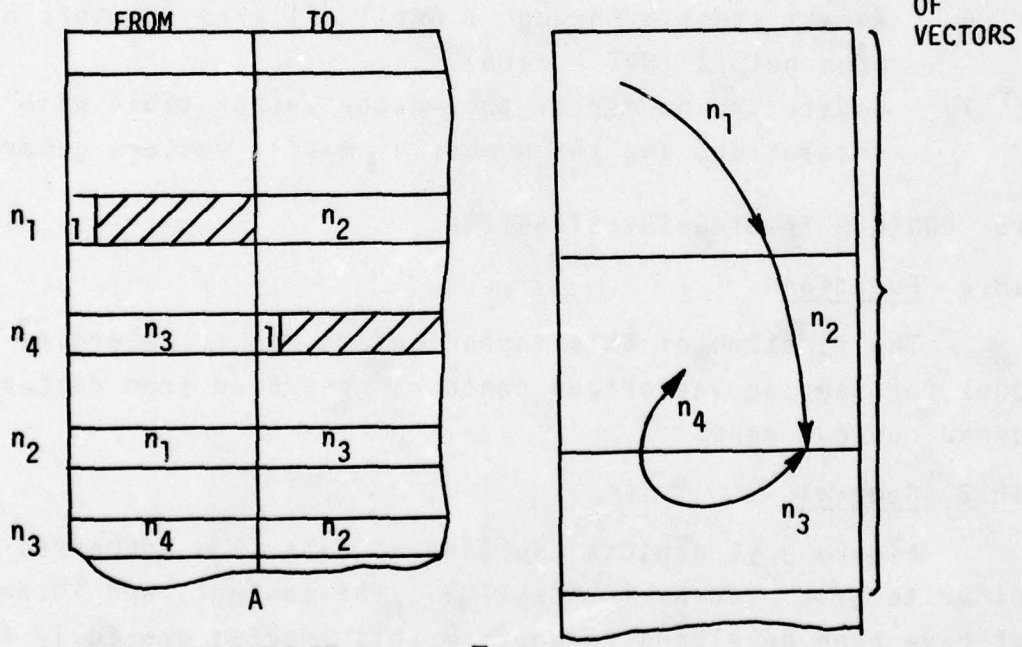
3.18.2 General

Figure 3-51 depicts the general data flow conceived for contour to grid process (CONTAGRID). The concepts and techniques that have been developed to perform this program are fully documented in the PROPOSAL FOR CONTOUR TAGGING AND GRIDDING USING STARAN/CDC SOFTWARE. Many of the conclusions reached are a direct result of this contour tagging investigation. Following is a brief synopsis of the conclusions.



MASTER VECTOR CONSISTS OF n_1, n_2, n_3, n_4

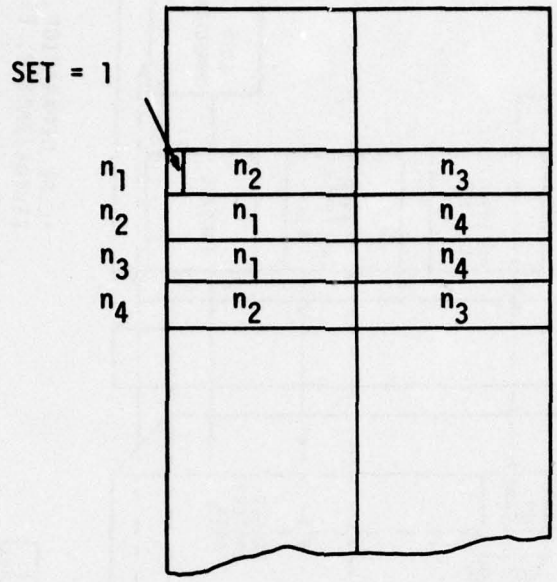
A. CASE 1



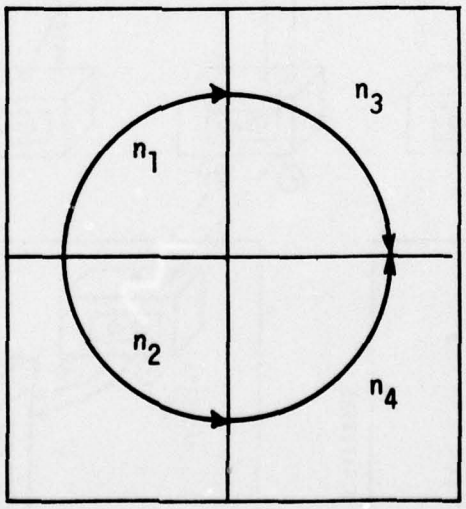
MASTER VECTOR CONSISTS OF n_1, n_2, $\bar{n}_3$, n_4
WHERE $\bar{n}_3$ DENOTES A REVERSED A.VECTOR

B. CASE 2

Figure 3-49. Example of Linked Array Vectors



A. MASTER VECTORS CONSISTS OF n_1, n_3, $\bar{n}_4$, $\bar{n}_2$



B. FOUR ADJACENT ARRAYS OF VECTORS

Figure 3-50. Example of Closed Loop of Array Vectors

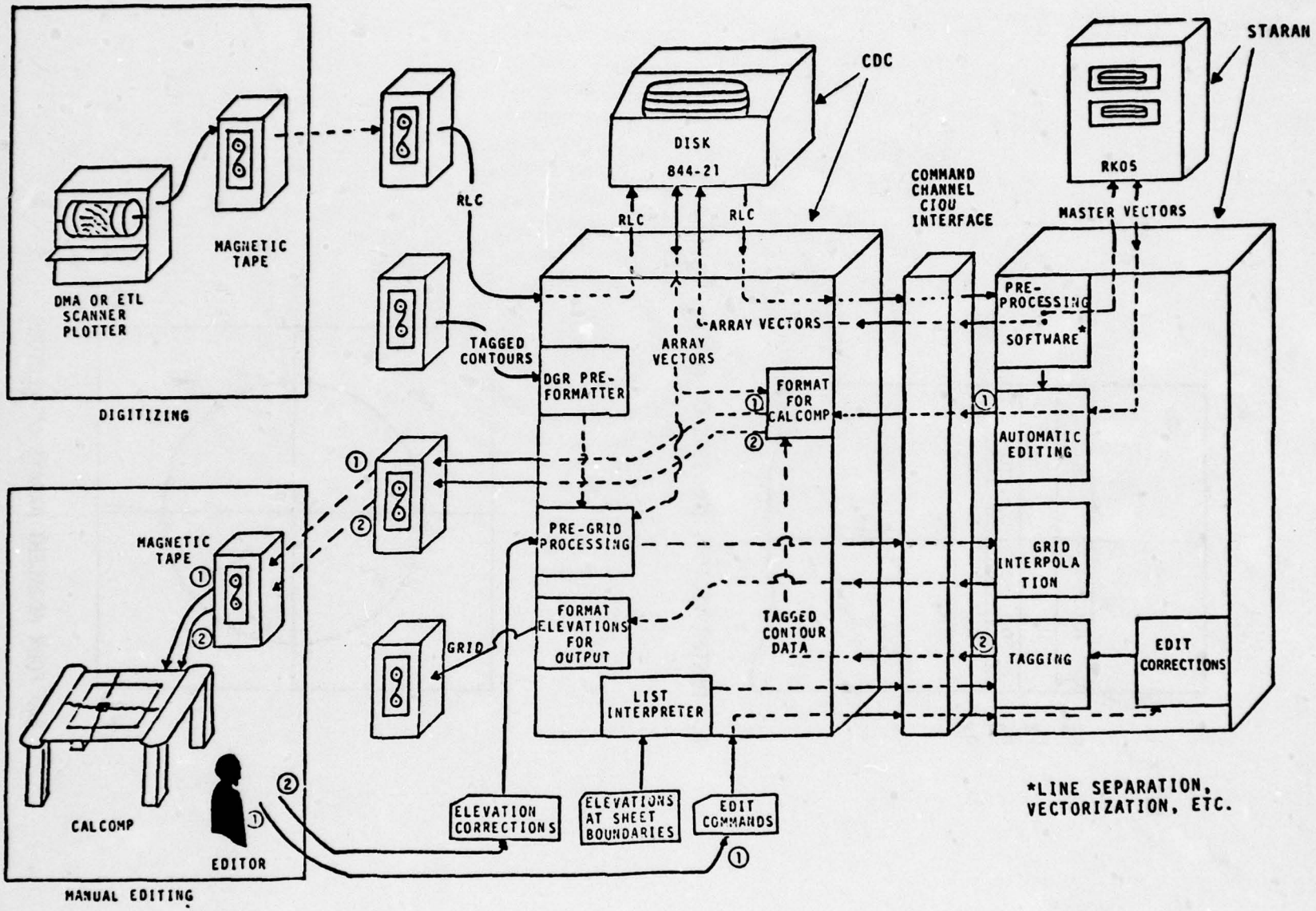


Figure 3-51. Generalized Data Flow in CONTAGRID

Contour overlays will be scanned on either the DMA or ETL scanner plotters at 2-mil resolution. The resulting tapes will be read in on a block basis by the CDC and broken up into four array-load groups. These data will then be stored on the 844 disk.

As needed, the array load groups will be read from the disk and transferred via the command channel interface to the STARAN. Here, the data will be loaded into the arrays (converted from run-length to binary) and the contour tagging process will begin.

The first part of the Contagrid process requires identification of index contours as opposed to non-index contours. This will be done by separation based on line weight.

After separation the data will be vectorized. Array vector tables and address tables will be shipped to CDC core and stored on the 844 disk as buffer space becomes filled. Vector headers will reflect the index or non-index status of the line. Data required to form master vectors and the master vectors themselves will be stored on the STARAN RK05 disk.

Once the master vectors have been formed, the automatic editing processes will begin. Numerics will be identified and flagged for deletion. Depression will be recognized, ticks flagged for deletion and intermediate vectors joined to form continuous contours. Deep cuts will be recognized and joined appropriately to corresponding master vectors. Breaks caused by the placement of numerics on the contours will be located. Appropriate endpoints will be paired and joined in a straight line manner. The joining array vectors will be shipped to the CDC for disk storage and new master vectors will be created which reflect this linkage. Should registration marks along the sheet boundaries be required by the gridding process, they will be identified and flagged at this time.

The contour data will now be ready for manual editing. Data will be formatted for output to plotting devices. CalComp formatting will be performed by the CDC. (Any raster plotting required will be formatted by the STARAN.) Numerics, flagging areas of change (adds or deletes) and problem areas (open endpoints), will be used to communicate corrections back into the system.

Next, the elevation data will be read in and associated with the open index contours. As part of the manual input, a list will be made (counter clockwise from the upper left-hand corner of the sheet) of the elevations of all index contours which cross the sheet boundaries. The open contours will be ordered in a counter-clockwise manner. By matching the input list (card format) to the master vectors flagged as indices in the ordered list in the STARAN, the association of elevation to linear list can be easily made.

When the open contours (opens) have been appropriately tagged, the closed contours will be tagged by finding the order of embedded contours, and their proximity to known opens. This process will involve rasterization of contour data. Prior to rasterization, the master vector numbers must be associated with the array vectors which comprise the master vectors. Because the array vectors are resident on the CDC disk, the association will be made in the CDC. The tagging of closed contours will then proceed.

If, after the contours have been automatically tagged, there remain contours with unknown elevations, these will be formatted for output to the CalComp where the editor will assign appropriate elevations. The data will then be prepared for the gridding process. The tagged contour vector strings will be reduced to the resolution of the grid output (10-mils). Saddle and ridge (S/R) lines and spot elevations will be input via CDC tape and converted from DGR format to a format compatible with the contour data. Intersections of S/R lines with contour lines will be found and a grid of known elevation points will be created. (It should be noted that the modularity of the Contagrid software will allow for direct input of DGR tagged contour data into the gridding portion of the system, bypassing the auto-tagging software entirely.) This grid of points will be partitioned into array load groups and shipped to the STARAN where planar interpolation will be used to find the missing elevation points.

It is highly unlikely that the input sheet will be so poorly registered on the scanner as to require alignment correction.

However, since skew could be a problem area a possible solution is addressed at the end of the software subsystems section. If such corrections are necessary, they will be performed prior to the tagging.

The final grid will be sent back to the CDC and formatted into DGR format. The final output will be written onto magnetic tape.

3.18.3 Line Separation of index contours - It is intended to use the Line Separation routines discussed in subsection 3.16 to perform the separation of the index and non-index contours.

In order to determine the reaction of these routines to 2-mil contour data the following has been performed.

Goodyear Aerospace has separated a 2-mil data tape of contours generated on the DMA scanner/plotter. The results of this process are being forwarded to ETL for plotting on the scanner/plotter. Conclusions will be drawn from the resulting plots.

APPENDIX A - MAGNETIC TAPE FORMATS

1. GENERAL

This appendix describes the tape record formats used to provide the input/output data to the STARAN AAP-raster processing.

There are four Input/Output forms that can be accommodated by the current software; these are:

1) ETL Plot Format Input data - ETL Plot Format - Output data,
2) ETL Run-Length Coded (RLC) Input data - ETL Plot format Output data,
3) DMA Run-Length Coded Input data - DMA Run-Length Coded Output data, and
4) DMA Run-Length Coded Input data - ETL Plot Output data.

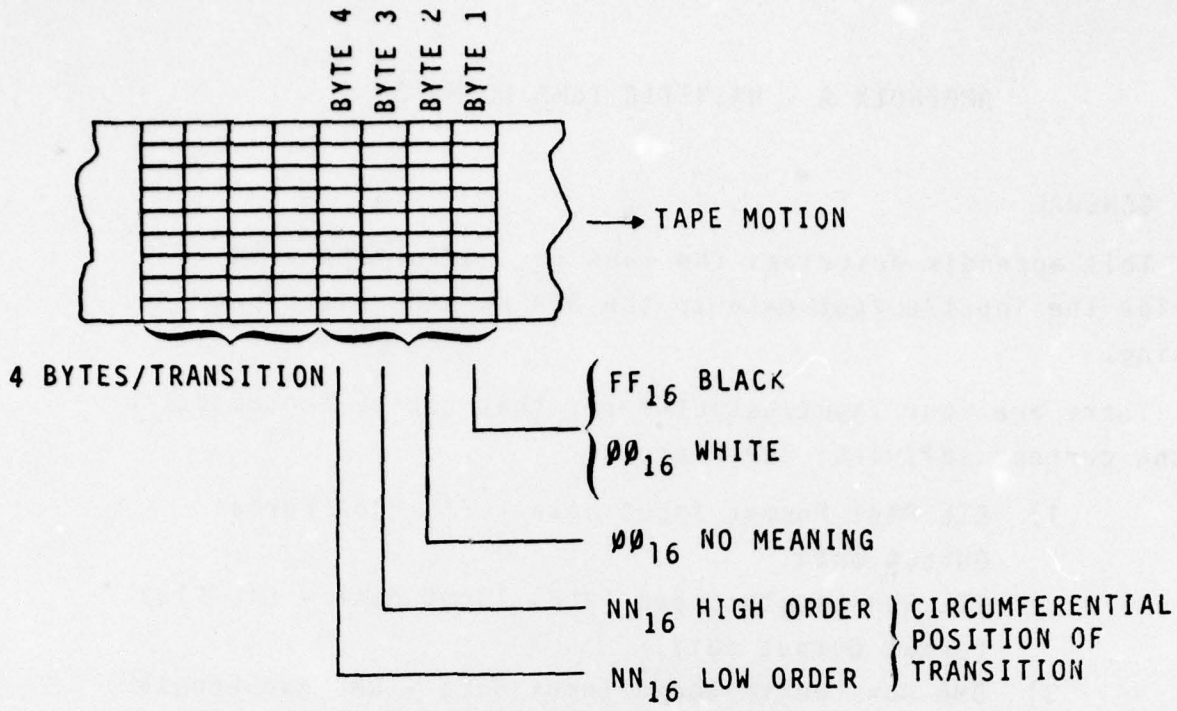
Each of the three formats - 1) ETL Run-Length Coded Format, 2) ETL Plotter Format, and 3) DMA Run-Length Coded Format - are described below.

2. ETL SCANNER RLC FORMAT

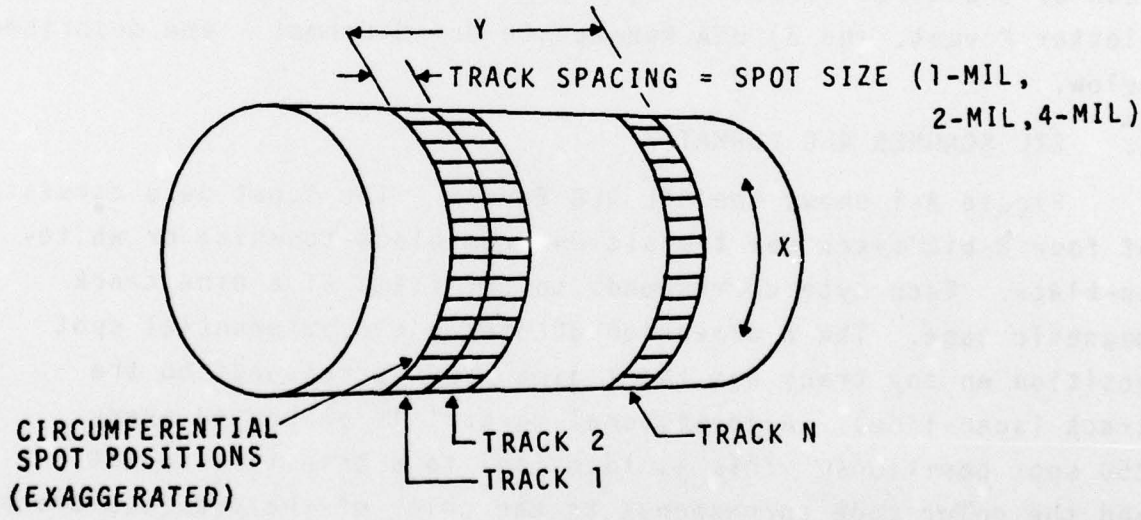
Figure A-1 shows the ETL RLC format. The input data consists of four 8-bit bytes per transition from black-to-white or white-to-black. Each byte corresponds to one frame of a nine track magnetic tape. The X dimension denotes a circumferential spot position on any track and the Y dimension corresponds to the track (scan-line). A 'positional marker' is generated every 250 spot positions. This is identical to a transition point and the color code corresponds to the color of the previous transition.

3. ETL PLOTTER FORMAT

Plot tape format uses 1 byte (8-bits + parity) of data to



(A) TAPE FORMAT



(B) SCANNING CONFIGURATION

Figure A-1. ETL Run-Length-Coded (RLC) Format

control 8 light emitting diodes for simultaneous exposure of 8 tracks on the drum. Thus, one tape record contains the data for all circumferential positions of 8 successive tracks. As with binary scanning, the plot records control each circumferential spot. The bits controlling the light emitting diodes are either set (1) or not (0), indicating exposure or not.

Records lengths are computed as with binary scanning, by the number of spot positions around the drum (for one track). The number of records is equal to 1/8 the number of scan records, since 8 tracks are exposed by each record.

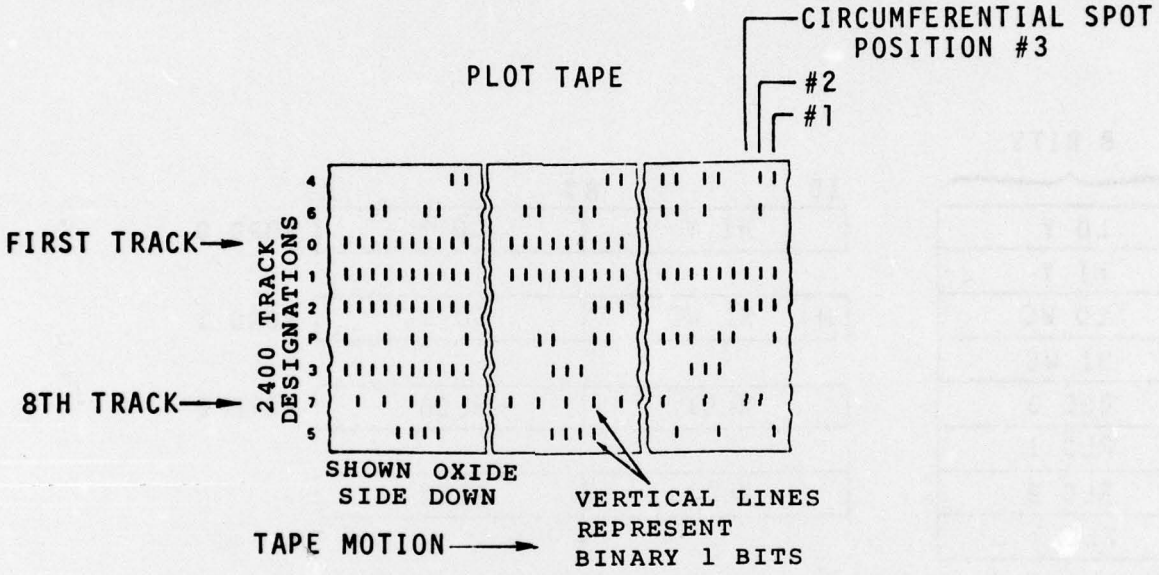
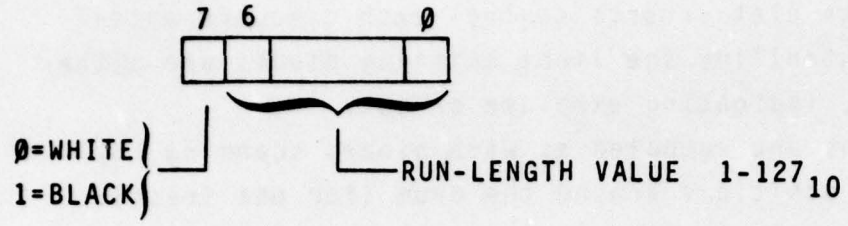


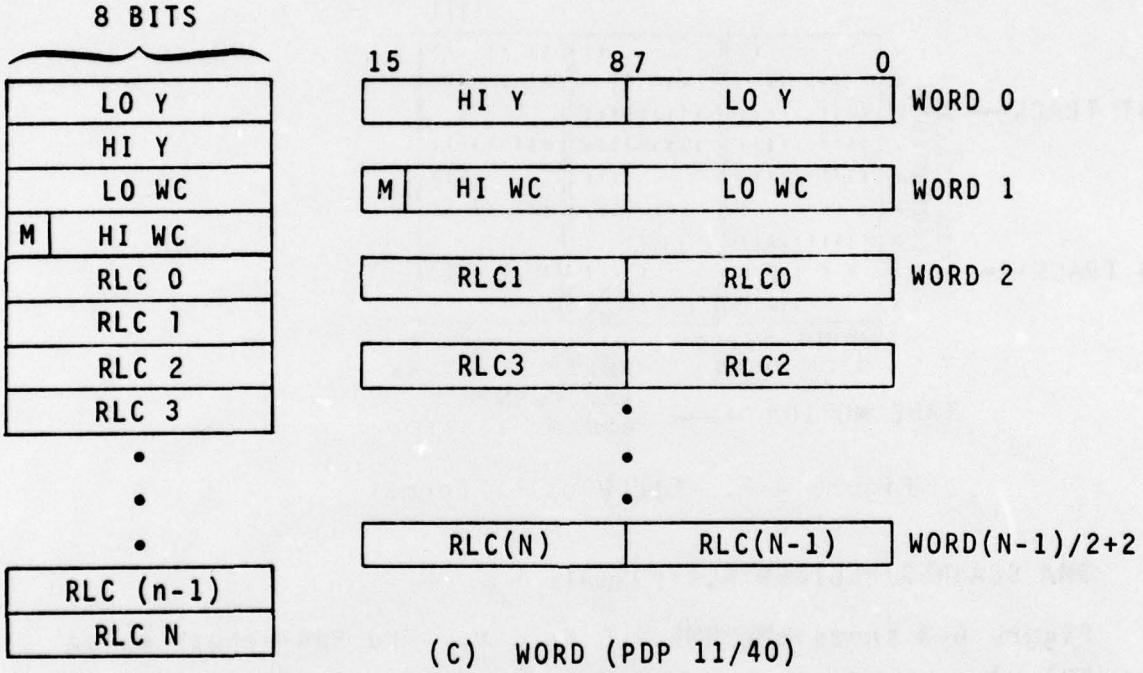
Figure A-2. ETL Plotter Format

4. DMA SCANNER/PLOTTER RLC FORMAT

Figure A-3 shows the DMA RLC format. The Run-Length value ($1 - 127_{10}$) represents the number of spot increments to power-on or power-off the laser beam along the X-coordinate (scan) line. Each Magnetic Tape (MT) RLC record contains a 16-bit Y-address, a mode bit, the word count, and the RLC data. The Y-address represents a scan line. The mode is 0 for RLC data. The word count defines the actual number of 16-bit data words which follow the word count.



(A) RUN-LENGTH VALUE



(B) MT FRAMES

(C) WORD (PDP 11/40)

Figure A-3. DMA Run-Length-Coded (RLC) Format

Two magnetic tapes are generated by the DMA scanner/plotter. One tape contains data with even Y-addresses, and one contains data with odd Y-addresses.

APPENDIX B - I/O MERGING/SEPARATING ROUTINES

The STARAN S-500 system supports only two tape drives; therefore, two off-line utility routines were written to handle DMA run-length coded data tapes. The functions of these two routines were:

1) to merge two DMA run-length-coded input tapes into one input tape containing alternate records from each tape, and
2) to separate a STARAN processed DMA run-length-coded output tape into two tapes, one containing records (0,2,4,...) etc., and the other containing records (1,3,5,....) etc.

These utility routines are written in FORTRAN and are executed on the XEROX SIGMA 9 which currently supports three tape drives.

APPENDIX C - COMTAL DEBUG AID

The Comtal Debug Aid routine (COMTAL) is an APPLE program executed from Bulk Core Memory. This routine is called by the Executive when requested. It transfers a pair of arrays of data to the left-half of the Comtal screen, and subsequently an adjacent pair of arrays of data to the right-half of the screen. This data is in raster-binary format and is generated as a graphic overlay (black (off), color (on)) on the screen.

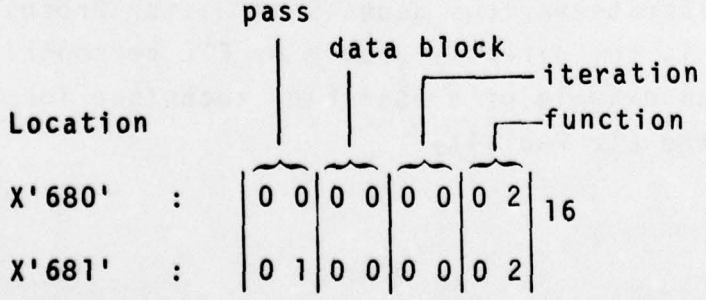
The user generates requests for viewable data via a list of four parameters. These parameters identify:

1) Pass Number (1,2,) etc.,
2) Data Block Number (0,1,2,...) etc.,
3) Iteration Number within data block (0,1,2,...) and
4) Function within iteration (1,2,3...) etc., to be displayed.

The parameters are input via the keyboard (SDM) as 32-bit values to the HSDB, where each 8-bits of each word identifies one of the four parameters. The first HSDB location is identified by the label RCHK and is at location $X'680'_{16}$.

Example

Display the results after line thinning, and after line symbol generation during the processing of track data. Display these results for the first and second iterations of the first data block. The resulting parameters generated in successive words of HSDB would be:



The current parameters pass (p), data block (d), iteration (i), and function(f) are constantly monitored by the executive. When the above parameters match those currently being checked in the HSDB list, then the data in the arrays is output to the Comtal screen (left-half). Later, during processing, the adjacent two arrays of data (p, d+1, i,f) are output to the screen (right-half), to generate a 2 x 2 set of arrays on the screen.

The Comtal contains three graphic displays, green, blue, and red. The viewable data sets (four arrays) are output to the graphics alternating from green to blue, and then to red. As each colored graphic is displayed, the preceding graphic is saved and may be retrieved later by the user. However, once all three graphics have been used, the next request for graphic data will overwrite the first of the three graphics saved, etc. Thus, only three graphic data sets can be saved for viewing at any one time. The parameter list must also be generated in ascending value with respect to the numbers associated with p, d, i, and f.

This routine allows the user to:

1) view the results of some Raster Processing function, Thinning, Symbol, Generation, etc., on-line without processing the whole overlay sheet and obtaining some hard copy output,
2) terminate a processing procedure if a problem is encountered due to:
   a) bug in a program,
   b) strange data form encountered, and
3) demonstrate various aspects of Raster Processing.

While this routine is not directly usable by ETL personnel, it has been saved as an example of a possible technique for future implementation in the ETL Facility.