TERRAIN ELEVATION DETERMINATION
USING A MICROPROCESSOR CONTROLLED VECTOR MAP

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Richard F. Goosen
March, 1985

OHIO UNIVERSITY
LIBRARY
ACKNOWLEDGEMENTS

I wish to sincerely thank Dr. Harold Klock for his assistance throughout the development of this Thesis. A special thanks is also given to my wife Rebecca and my family (Greg, Stacey, Molly and Rufus) for their support during my graduate study.
LIST OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background and Purpose</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Overview of Solution Development</td>
<td>4</td>
</tr>
<tr>
<td>2. RELEVANT OPERATIONAL FACTORS</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Description of Terrain Source Data</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Description of Pre-Flight Resources</td>
<td>14</td>
</tr>
<tr>
<td>2.3 Description of In-Flight Resources</td>
<td>17</td>
</tr>
<tr>
<td>2.4 Performance Requirements</td>
<td>20</td>
</tr>
<tr>
<td>3. DESCRIPTION OF TEST SYSTEM DEVELOPMENT</td>
<td>24</td>
</tr>
<tr>
<td>3.1 Overview of General Solution</td>
<td>24</td>
</tr>
<tr>
<td>3.2 Test System Development</td>
<td>29</td>
</tr>
<tr>
<td>4. TERRAIN DATA TO VECTOR MAP CONVERSION</td>
<td>34</td>
</tr>
<tr>
<td>4.1 Description of Pre-Flight Processing</td>
<td>34</td>
</tr>
<tr>
<td>4.2 Bordering Algorithm Description</td>
<td>36</td>
</tr>
<tr>
<td>4.3 Vector Encoding Algorithm Description</td>
<td>57</td>
</tr>
<tr>
<td>4.4 Formatting Algorithm Description</td>
<td>70</td>
</tr>
<tr>
<td>5. MAP PROCESSOR SYSTEM DESCRIPTION</td>
<td>84</td>
</tr>
<tr>
<td>5.1 Overview of In-Flight Operation</td>
<td>84</td>
</tr>
<tr>
<td>5.2 System Hardware Definition</td>
<td>86</td>
</tr>
<tr>
<td>5.3 System Software Definition</td>
<td>92</td>
</tr>
<tr>
<td>5.4 System Interface Definition</td>
<td>113</td>
</tr>
</tbody>
</table>
6. SYSTEM TESTING ........................................ 120
  6.1 Description of System Testing ............... 120
  6.2 Analysis of System Performance .............. 129
  6.3 Comparison of Vector Mapping and Direct
      Storage .......................................... 133

7. SUMMARY AND CONCLUSION ......................... 136
   7.1 Review of System Development ............... 136
   7.2 Suggestions for Further Study .............. 139

LIST OF REFERENCES ................................. 141

APPENDICES
   A  Map Processor Program Listing .............. 142
   B  Simulated Aircraft Computer I/O Listing .. 150
   C  An Alternate Vector Map Example ............ 152
LIST OF FIGURES

2.1 Terrain Sketch ................................. 10
2.2 Contour Plot of Terrain with Digitizing Grid. 12
2.3 Digitized Elevation Values ...................... 13
2.4 Pre-Flight Resources ............................ 16
2.5 In-Flight Resources ............................. 19
3.1 Generalized Vector Map System .................. 25
3.2 Microprocessor Controlled Vector Map Test System .................................. 31
4.1 Pre-Flight Processing ............................ 35
4.2 Border Detection Algorithm Output(300') ....... 39
4.3 Border Detection Algorithm Output(400') ...... 40
4.4 Border Detection Algorithm Output(500') ...... 41
4.5 Bordering Algorithm Program Listing .......... 44
4.6 Bordering Algorithm Functional Flowchart .... 46
4.7 Border Detection Algorithm Tape Save Format   48
4.8 Bordering Process .............................. 51
4.9 Border Point Detector (Vertical Section)..... 53
4.10 Vertical Border Point Detector Output ...... 55
4.11 Horizontal Border Point Detector Output ..... 56
4.12 Vector Encoded Border Point Set .............. 58
4.13 Vector Encoding Algorithm Operation ........ 60
4.14 Vector Encoding Algorithm Mainline Flowchart 62
4.15 "SCANNER" Flowchart .......................... 65
4.16 "NEIGHBOR" Subroutine Flowchart .......... 66
| 4.17 | 300' Border Elevation Processing | 71 |
| 4.18 | 400' Border Elevation Processing | 72 |
| 4.19 | 500' Border Elevation Processing | 73 |
| 4.20 | Vector Encoding Algorithm Output Data Format | 74 |
| 4.21 | Vector Encoding Algorithm Program Listing | 75 |
| 4.22 | Formatting Algorithm Program Listing | 79 |
| 4.23 | Vector Encoding Algorithm Output Data | 81 |
| 4.24 | EPROM Map Processor Data | 83 |
| 5.1  | Vector Map Processor Test System | 85 |
| 5.2  | Map Processor Functional Schematic | 88 |
| 5.3  | Map Processor System Memory Map | 90 |
| 5.4  | Map Processor Program Memory | 93 |
| 5.5  | Map Processor State Graph | 94 |
| 5.6  | Look Up Algorithm General Operation | 98 |
| 5.7  | Elevation Determination Example | 102 |
| 5.8  | Look Up Algorithm Program Flowchart | 108 |
| 5.9  | Test System Interface | 115 |
| 5.10 | Initialization Timing | 116 |
| 5.11 | Data Transfer Timing | 118 |
| 6.1  | Operational Test Functional Diagram | 121 |
| 6.2  | Control/Display Program(Manual Test Version) | 123 |
| 6.3  | Control/Display Program(Printout Version) | 125 |
| 6.4  | Map Processor Test System Elevation Values | 126 |
| 6.5  | Digitized Terrain Elevation Source Data | 127 |
CHAPTER 1
INTRODUCTION

1.1 Background and Purpose

The operational effectiveness and safety of aircraft is necessarily based upon the correct determination of various measurements made during aircraft operation. While the relative importance of these operational parameters is often determined by the specific type and purpose of the aircraft involved, one parameter which is always of critical importance is the value of the aircraft altitude to ground elevation differential. During the tremendous improvement in on board aircraft sensors and computational resources during recent years, a steady improvement in the determination of this critical parameter has occurred. In conventional applications at present the vertical differential is computed by the aircraft flight computer using a highly accurate pneumatic based measurement of aircraft altitude and stored elevation values of the terrain along the flight path. In addition the ranging radars or LASER sensors present on most larger aircraft often provide a highly accurate direct measurement of aircraft to ground separation.

In spite of such improvements, however, there remains a significant number of aircraft whose configuration or primary purpose restricts them from using such advanced
technology. For aircraft which do not fly fixed routes and/or have limited computational resources, the elevation of the earth which is used for the differential calculation is often simply a single average elevation value entered into the flight computer before each flight. This elevation value is then used by the flight computer for the calculation of aircraft to ground separation over the entire area of operation. This area can often be as large as 200 x 200 miles. Obviously such an approximation and especially one which is typically not analytically determined, usually results in a differential value which is seldom correct and is often so erroneous as to be unusable.

The need for the correct determination of the aircraft to ground vertical differential has, over the years, led to a number of theoretical solutions to the ground elevation problem. As aircraft computational resources have become more sophisticated, one recurrent approach to determine ground elevation has been to encode the spot elevations of a particular terrain section on a point by point basis. These elevations would then be entered into an aircraft computer during pre-flight operations. Once in the air, a direct access of this data base for a point of interest would be performed and an earth elevation value would be obtained. This technique, referred to as direct storage, is quite limited as to the area of coverage possible with a
usable resolution and has never been placed in general use.

A more involved approach to the central problem of storing a terrain map for the in-flight access of the aircraft computer is referred to as Vector Mapping. The Vector Mapping technique essentially segments a given terrain section into various elevation regions and then stores the borders of these regions as a set of line segment vectors. This compacted representation of the operational terrain can then be entered into the aircraft computer data base for in-flight access. The Vector Mapping technique therefore sacrifices resolution and simplicity of access in order to achieve a significant reduction in data storage requirements.

In spite of the promise of Vector Mapping, the technique has not, as of yet, been physically realized in an operating flight system. The objective of this Thesis is to explore the application of Vector Mapping as applied to terrain elevation storage and to realize a test system capable of demonstrating the feasibility of the technique. This realization utilizes typical aircraft operational constraints as limits during its development. The test system developed in this Thesis, while not precisely suited to any aircraft, will demonstrate solutions to the critical problems involved in the specific application of the technique necessary for an in-aircraft utilization. In
addition, the performance of the test system will be described and evaluated relative to alternate elevation determination means. During the definition and solution of the avionics specific problem, this Thesis will also attempt to describe the general nature of the Vector Mapping technique and its potential for alternate application.

1.2 Overview of Solution Development

The relevant operational factors which define the required characteristics and limiting factors for the Vector Mapping technique as applied in an aircraft environment are described in Chapter 2. The chapter begins with a description of the terrain elevation source data, describing its general format, availability and accuracy. Available pre-flight and in-flight resources are then examined in order to define an assumptive base for the design and development of a test system. Finally Chapter 2 describes the performance requirements which the typical aircraft application would demand. Chapter 3 utilizes the operational requirements and problem definition provided by Chapter 2, in order to realize a general solution to the aircraft elevation map problem using the Vector Map technique. This general solution is then realized by the test system developed as a part of this Thesis and
described in Chapter 3.

Chapter 4 describes the pre-processing steps necessary to create the Vector Map representation of a terrain section. Included in this chapter is a complete operational description of the necessary algorithms required to effect this conversion. In order to further clarify the operation and the critical aspects of these algorithms the creation of a Vector Map representation for a hypothetical terrain section is presented as an example. The Map Processor test system which performs the simulated in-flight elevation data retrieval function is described subsequent to the pre-flight activities. As described in Chapter 5, the relevant hardware and software aspects of the microprocessor based test system are presented in detail. Included in the hardware description is the functional layout of the Map Processor and its operational memory allocation. The software description of the Map Processor presents the 6502 microprocessor version of the critical Look Up Algorithm which is designed to process the Vector Map data and retrieve a selected coordinate elevation. Chapter 5 concludes with a detailed description of the parallel interface which links the Map Processor and the simulated flight computer.

The testing which was performed on the test system is described in Chapter 6. Included in this discussion is the methodology of testing and the resultant test data. An
analysis of the test data is performed and then used as a basis of comparison between the Vector Map and direct storage technique as mentioned in the background discussion of this chapter. Both techniques are evaluated in regard to time of access, accuracy, complexity of access and storage requirements. Chapter 7 summarizes the development of the Vector Map concept and the test system as presented in this Thesis. The chapter concludes with recommendations for further research including the use of the concept for the storage of digital picture data.
2.1 Description of Terrain Source Data

A basic operational factor which controlled the overall development of the Vector Map microprocessor system was the source data designed to model a specific terrain section of interest. This data was carefully selected to insure at least a functional similarity to the actual source data available for an in-aircraft operational system. The source data base of terrain elevation information has been historically subject to inaccuracy, incompleteness, and non-uniformity. This has been due to an overall lack of technology as well as a lack of a centralized mapping agency. Initially all elevation data for the earths' surface was obtained by ground survey methods and was often disjointed and inaccurate. With the development of the airplane during the 20th century, the acquisition of elevation data was revolutionized in terms of dramatic increases in speed of acquisition, area of coverage and precision. The improvements were still quite limited, however, due to the inviolability of political borders which served to prohibit the development of a global elevation data base of the earths surface. With the development of various satellite mapping systems during the
Last 20 years this problem has also been essentially eliminated. (1) At present, a satisfactory global elevation data base is available from various United States Agencies, most notably the Defense Mapping Agency (DMA) located in Fort Belvoir, Virginia.

The primary means of acquiring earth elevation data at present is through the use of satellite photogrammetric techniques. In photogrammetry a stereographic image of the earth's surface is obtained by a high altitude satellite such as LANDSAT and this image is transmitted to a ground receiver. The image is then processed on large main frame computers to extract the elevation values of a terrain section. The resultant data is then archived at the host agency for subsequent distribution. The actual form of the elevation data is a signed, 16 bit, integer array where each array position represents a specific geographic location on the earth's surface. This array which is conventionally referred to as the Digital Terrain Model (DTM) has a LSB resolution of one meter and provides approximately one data point for each 300 square meters of earth's surface. (2) The DTMs in this standardized format (See 3) cover essentially all regions of the earth (including the sea floor) and are maintained by the Defense Mapping Agency.

In order to insure that the microprocessor controlled Vector Map system developed in this Thesis, would
accurately typify an in-aircraft system, the specific terrain used for development and testing was designed to functionally model the existing elevation data base. Because the DTM format requires over 3000 spot elevation data points for each square kilometer, however, the source elevation data used for this Thesis is less dense. This allows meaningful terrain features (mountains, etc) to be manipulated using minimal memory requirements. The demonstration terrain shown in Figure 2.1 was used in the development of the microprocessor controlled vetor map system. This terrain section was selected to typify the gently rolling terrain frequently found in most areas of the world. A depression was included in the terrain map to exercise portions of the system software which deal with this frequently troublesome terrain structure. The absolute dimensions of this terrain section are not fundamental to the system development but in the interest of completeness are defined as 1 kilometer x 2 kilometer.

The process of creating a simulated DTM from a given terrain section is relatively straight forward. In this case a digitizing grid was applied to the terrain section and a spot elevation consisting of a signed 16 Bit integer was taken at each grid intersection. The specific size of the terrain section in this case equates to a spot elevation every 10,000 square meters which is approximately 30 times less dense than the grid spacing of an actual DTM.
The application of the digitizing grid is shown in Figure 2.2. In this figure, the terrain section is represented by its true contour elevation values. In this context a true contour elevation line is defined as locus of isoelevation points at a specific elevation level. In the example terrain displayed in Figure 2, contour lines are drawn for the 300, 400 and 500 foot levels. For purposes of this Thesis the elevation values are all defined in feet as apposed to the metric values of an actual DTM. This representation was chosen to provide consistancy with current aircraft elevation and altitude formats. It is also appropriate to note that the contour lines shown in Figure 2 represent "actual" contour lines. Due to the inherent quantization of the digitizing process, these lines are not directly reflected in the resultant DTM.

The simulated DTM representation is shown as the integer data array of Figure 2.3. This data array, which functionally models an actual DTM as previously described, will serve as the input terrain data array for the subsequent development of the microprocessor based Vector Map. As can be seen from the data array the minimum and maximum elevations present in this map are 200 and 550 feet respectively. All elevations are MSL (Mean Sea Level) based values and are typical of a coastal terrain area. The small size of this data array will facilitate processing on a
Figure 2.2

Contour Plot of Terrain with Digitizing Grid
Figure 2.3

Digitized Elevation Values
small computer system without departing from the relevant aspects of actual DTM data. In a similar context the small size of the terrain section used will provide an adequate demonstration of the microprocessor based Vector Map concepts due to the relatively complex terrain structures it contains. It should also be noted that the fact that the simulated DTM has elevation values in feet instead of meters and a LSB resolution of one foot instead of one meter is of no significance to the operational concepts of the system.

2.2 Description of Pre-Flight Resources

After defining the data source for the terrain elevation determination, the second critical area controlling the Vector Map Processor system design is the available resources for the solution of the problem. The first major source area of available resources can be grouped into the general category of pre-flight resources. The central resource available in this category is the airfield computer. This computer has the primary function of managing a large data base of pre-flight information. This data base is generally created by a servicing agency and is flexible in order to add or delete site specific data as requested by the user. This airfield computer is used by the aircraft flight crew to reference and/or
define necessary flight data for a specific flight. The data is then stored in a non-volatile memory module for transfer to the aircraft where it forms the flight data base for the aircraft computer. A typical use of such a system would be for the pre-flight definition of a sequence of destination locations stored as geographic coordinates. These locations would then be accessed by the aircraft computer in order to form a flight path for a specific flight. Figure 2.4 shows a functional diagram of the pre-flight resources.

The aircraft computer is the most valuable pre-flight resource in regard to its potential contribution to the Vector Mapping terrain determination system. This computer is difficult to describe in detail, however, due to the lack of uniformity in the systems currently in use at different airfields. In order to provide a representative example of such computer resources, a typical minimum system was chosen as a design parameter. The HP 1000, E series, computer is the computer system currently in use at most U.S. Air Force airfields for pre-mission planning. This computer system is a 16 bit minicomputer based on microprogrammed MSI (TTL) hardware. It typically operates in a system with a 10 MB Hard Disk containing a large data base of pre-flight information. This computer can be programmed in FORTRAN or other high level languages. Since the system operates at various remote military locations,
Flight Data Base

Airfield Computer

Non-Volatile Memory Module

To Aircraft Via Flight Crew

Manual Data Entry

Figure 2.4
Pre-Flight Resources
it can be regarded as a minimum stand alone system for most aircraft applications including commercial aviation. It should be noted that using a military computer system as a pre-flight resource example is particularly appropriate since the typical military aircraft mission is directly compatible with the Vector Mapping target application discussed in Chapter 1.

2.3 Description of In-Flight Resources

The in-flight resources available for the actual operation of the Vector Mapping system constitute a critical design guideline. By adhering to this guideline as the basis for the demonstration system, a practical solution insuring a valid feasibility determination is assured. The central aspects of the in-flight resources are the characteristics of a typical aircraft computer and the non-volatile memory module mentioned in the previous section of this chapter. Together these two aspects define the operational memory size of the map processor and the gross size limitations of the Vector Map terrain representation.

The non-volatile memory module, often referred to as a Data Transfer Module (DTM), is an electrically eraseable; electrically programmable memory module in current aircraft usage. This pocket size device is available in 8 K x 8, 16 K x 8, and -32 K x 8 sizes. In addition the
module may be configured at manufacture to accommodate various word lengths other than the standard 8 Bit data word. The most common configuration of the module is the 8 K x 8 model. In normal usage the module is hand carried by the aircrew to the aircraft and loaded into the aircraft computer prior to flight. The module conventionally contains only navigation data for the flight and seldom modifies the operational program of the aircraft computer.

The aircraft flight computer is difficult to describe due to the fact that each computer is often developed for a specific type of aircraft. This lack of standardization occurs since the aircraft computer, primarily part of a navigation system, is generally developed to provide specific features of operation without regard to internal architecture. In general, however, it is valid to characterize the typical small aircraft computer as a small (~ 64K core memory) special purpose computer which operates in a defined operational cycle (~ 50-100 msec). Primarily stressing fixed point arithmetic calculations, an important secondary aspect is its extensive I/O capabilities. These I/O capabilities consist of several serial, parallel and discrete channels. These ports are primarily used to control external sensors and to maintain necessary instrument displays. This typical aircraft computer is shown in the operational block diagram of in-flight resources (Figure 2.5).
Figure 2.5

In-Flight Resources
2.4 Performance Requirements

The final category of the relevant operational factors serving as system development guidelines can be grouped together as performance requirements. Under this broad category are grouped all performance aspects of the operational aircraft environment which contribute to the definition of the hardware and software requirements of the microprocessor controlled Vector Mapping system. These requirements, rather than absolute design criteria, represent a reasonable assumptive base to the needs of an actual in-aircraft system. The requirements may be further separated into the specific areas of system configuration, use definition and map definition. Each area contributes on an equal basis to the overall demonstration system development presented in Chapter 3.

The performance requirements comprising the area of system configuration are an assortment of human factors and existing aircraft computer requirements. The central human factors guideline can be simply stated as the requirement that any Vector Mapping system should not increase the in-flight workload of aircraft crew members. In a general version of this requirement, the system should be operationally transparent to the aircrew. This guideline is based on the fact that any increase in the typically heavy aircrew workload would invariably cause a decrease in some existing area of aircraft performance and is hence
 unacceptable. Within the more quantitative areas of system configuration, two possible implementations of the Vector Mapping system must be considered. If the Vector Map processor system is to be imbedded into the existing aircraft computer software, no more than 2 K Bytes of program space (approximately 3% of a typical 64 K computer) may be used. In addition, this software must not alter the basic operation of the aircraft computer. This requirement indirectly forces the Vector Mapping system to be implemented as a free time computational task most probably operating in lieu of the conventional free time task of Built In Test (BIT). Alternatively, if the Vector Map processing system is to be a separate peripheral to the aircraft processor, its configuration should be limited to a physical size less than .5 ft$^3$. In this configuration, the Vector Mapping system is also limited to a single communication channel with the aircraft computer. In either the imbedded or the peripheral configuration, the addition of minimal support software functions on the part of the aircraft computer, (such as geographic to Vector Map coordinate transformation) is permissable.

Use definition describes the expected operational role of the Vector Mapping system. The most critical aspect in this area is that the Vector Mapping system as it is discussed in this Thesis is not intended to be used for terrain avoidance. By restricting the system to the
advisory status of other aircraft sensors, no autopilot control or safety functions need be evaluated. The anticipated operational use of the system can be defined as the determination of the elevation value of a specific vector map location as defined by the aircraft processor (or its flight management software for the imbedded configuration) on a non-real time basis. The number of unknown elevation points requiring resolution during a typical aircraft flight is expected to be small enough to preclude any significant processing time restrictions on the Vector Map processor software.

The final area of performance requirements is the area of map definition. The significance of this area is due to the concept that the potential contribution of the Vector Mapping system to existing aircraft operation can be interpreted as roughly proportional to the terrain area that the system can cover. In this regard, therefore, the non-volatile memory module which must serve as the transfer medium for the Vector Map data between the ground and aircraft computer resources, restricts the map data to a maximum 32 K x 8 data block. In order to insure the area of coverage is of a useable size a 25 x 25 kilometer (15 x 15 mile) terrain section must be capable of being stored in this memory size with useable resolution. The resolution requirement is in turn defined as 100 feet (i.e. the terrain point elevation value must be reliable to 100
feet). Ideally the Vector Map solution should introduce no other elevation error into the system beyond the inherent quantization error (See 2.1) and this resolution ambiguity.
3.1 Overview of the General Solution

The general solution to the elevation determination problem which is the central motivation of this Thesis was developed using the restrictions and guidelines described in Chapter 2. This solution is shown in Figure 3.1 which displays the generalized flow of an elevation value from a terrain feature through the Vector Map system to the eventual aircraft display of the value. The general solution is designed to utilize the existing operational elements currently available to most United States aircraft. In order to systematically describe the operation of the solution, it may be broken down into the functional areas of operations performed by centralized facilities, operations performed by on-site airfield facilities and in-aircraft operations. A description of these functional areas will provide an overview of a possible operational system and will provide the necessary base for understanding the background of the demonstration system development.

The operations performed by the centralized facilities are responsible for obtaining global terrain elevation data and distributing regional terrain elevation data to the using airfield facility in a format appropriate for Vector Map creation. As discussed in Chapter 2, a global or near
Regional, Modified DTM Data

Airfield Computer Facility → Vector Map of Flight Region → N.V. Memory Module

Aircraft Computer

1. Vector Map Coordinate
2. Coordinate Elevation

Aircrew Control/Aircraft Sensors

Figure 3.1
Generalized Vector Map System
The global data base of terrain elevation data is currently available at the Defense Mapping Agency (DMA). This data, which has been obtained principally by satellite acquisition is not, however, in the optimum format for Vector Map development. For this reason, a centralized distribution facility is required to reformat the Digital Terrain Model for the various user airfields. The need for region specific data is obvious since a global data base at each airfield is not feasible or necessary. The reformatting task is somewhat less obvious but can be summarized by the tasks of elevation conversion and data reduction. Elevation conversion involves no more than the conversion of the DTM standard metric elevation format to the conventional aircraft representation (feet). The need for the data reduction task is based on the excessive sampling density present in the standard DTM data. As was described in Chapter 2, the DTM data provides an elevation value for each 300 m². It has been shown, however, (See 4) that the grid size (sampling interval) necessary to maintain a ±5 meter accuracy for a 10 meter resolution in map elevation data is 50 meters (2500 m²). For this reason a 8 to 1 reduction in DTM data should be possible without compromising the 100 foot resolution required for aircraft usage. Without further investigation, however, this implies at least a significant decrease in DTM data density and a corresponding increase in area of coverage per regional
data base can be gained by implementing a data reduction function at the central distribution facility.

Once the regional reformated DTM data has been transferred from the central distribution facility to the on-site airfield facility, the data must be further processed by existing computer resources into the flight specific Vector Map data. This task is accomplished by various high level language programs run on the airfield computer resources described in Chapter 2. These programs begin with the modified DTM data and sequentially process the data for a specific (15 Km x 15 Km) aircrew defined region into its Vector Map representation. Specifically the modified DTM data is segmented into elevation regions in a process directly analogous to the thresholding process used in image processing (See 5). The number of elevation regions are indirectly determined by the resolution requirements and area of coverage necessary for a specific flight. Once the modified DTM data field is segmented, additional processing reduces the borders of these segments or elevation regions into a series of straight line segments. The final processing step then encodes these straight line segments into their vector representation. When all segments for a given terrain section have been encoded, the airfield computer then transfers the vector data collectively referred to as the Vector Map into the non-volatile memory module described in Chapter 2. The
non-volatile memory module is physically transferred by the aircrew prior to the flight into the aircraft where it becomes part of the Vector Map processor memory.

The configuration of the in-aircraft portion of the Vector Map system shown in Figure 3.1 is an implementation of one of the two configuration options discussed in Chapter 2. In this case it was decided to make the Vector Map processor a peripheral to the aircraft computer instead of imbedding the Vector Map processing software into the existing flight management software. This configuration allows more development freedom on part of the map processing software since only minimal integration with the aircraft computer is necessary. This implementation creates a more general solution (aircraft independent) and minimizes the possibility of disrupting aircraft computer operation.

The aircraft processor shown in Figure 3.1 initiates all map processor operation. In normal operation the aircrew or an aircraft sensor will initiate operation by submitting a geographic coordinate with unknown elevation. The aircraft computer then converts the coordinate into the Vector Map cartesian coordinate and transmits it to the map processor via a communication channel. When the unknown elevation coordinate is received by the map processor the elevation value is then determined by a special algorithm that determines the highest elevation region which contains
the coordinate. Once this is determined, the elevation value of the region is transmitted to the aircraft computer as the elevation value of the submitted coordinate. The aircraft processor services the map processor communication channel as part of its operational cycle. When an elevation determination has been completed by the map processor, the aircraft computer receives the elevation value and displays or otherwise uses the information.

3.2 Test System Development

It is quite apparent that a comprehensive development of a test system which would effectively model all the functional areas present in the generalized Vector Map system is beyond the necessary scope of this Thesis. For this reason a test system designed to functionally demonstrate the most critical areas of the general system was developed. This test or demonstration system presents at least a rudimentary form of the critical algorithms which create the Vector Map representation of a digitized terrain section and then use the Vector Map representation to extract the elevation of any coordinate within the section. The operation of the test system will demonstrate the feasibility of the Vector Map concept without the unnecessary involvement of the areas of satellite acquisition, centralized facility operations and the
aircraft flight management software. A functional diagram of the test system is shown in Figure 3.2.

A description of the test system begins with a manually digitized terrain section. For demonstration purposes, the representative terrain section shown in Figure 2.1 was used. The terrain section was arbitrarily defined to be 1 Km x 2 Km and was digitized using a 100 x 100 meter grid size as shown in Figure 2.2. Although the actual area of coverage is not critical to the operation of the Vector Mapping algorithms, the terrain section represents a reasonable section of the earth's surface and contains the necessary terrain features required for feasibility evaluation. The 100 meter grid size, while not duplicating actual DTM data, produces a data density and form which could be obtained by direct conversion from DTM data by the centralized distribution facility shown in Figure 3.1. The integer data field produced by the manual digitization process is shown in Figure 2.3. This data provides the terrain representation used for the development of the Vector Mapping test system and will also provide the basis of evaluation for the systems performance.

In the Vector Mapping test system, the integer data field representing the terrain section is entered into a small computer system used to model the airfield computer facility of the general solution. The computer system used
Figure 3.2

Microprocessor Controlled Vector Map Test System
in the test system is a small personal computer manufactured by Commodore Business Machines, Inc. This computer is a 6502 based machine with 64 K Bytes of physical memory. The test system utilizes three algorithms implemented in BASIC to provide the conversion of the integer data field (representing the regional, modified DTM data of the general system) into the Vector Map representation of the terrain section. These three algorithms were implemented in three separate programs due to the limited program memory available for array manipulation. In an actual operational system the significantly more powerful HP 1000, E series, computer defined in Chapter 2 as an assumptive resource, would allow all three algorithms to be combined into a single more efficient FORTRAN program. The Vector Map produced by the test system is transferred into a single 2716, 2 K x 8, EPROM designed to model the non-volatile memory module of the general solution.

The focus of the Vector Map test system is the map processor. In the demonstration system, a small 6502 microprocessor based system performs the map processor function of extracting the elevation of a selected coordinate from the Vector Map stored in the map EPROM. This is a stand alone processor which functions as a peripheral to the simulated aircraft computer.

The communication channel which serves to transfer the
Vector Map coordinate and return the determined elevation value is a bidirectional parallel bus. The map processor is completely dedicated to the task of receiving a coordinate via this bus and returning an elevation value. To perform this function, the map processor system includes a 2 K x 8 EPROM program memory and 2 K x 8 Bytes of RAM used as scratch pad memory. The communication interface is accomplished using a 6522 VIA (Versatile Interface Adapter).

The aircraft computer which utilizes the map processor is simulated using the same computer previously used to create the Vector Map. In this configuration it is required to input a Vector Map coordinate from an operator and transfer this coordinate to the map processor. The simulated aircraft computer then waits until the map processor has completed the elevation determination process whereupon it then receives the elevation value from the map processor, converts it to feet and displays the value to the operator. No geographic to Vector Map coordinate transformation or other flight management functions are performed by the simulated aircraft computer.
CHAPTER 4

TERRAIN DATA TO VECTOR MAP CONVERSION

4.1 Description of Pre-Flight Processing

The pre-flight processing discussed in this chapter consists of three algorithms which sequentially operate on the terrain elevation data supplied by the hand digitization process. The input data, designed to simulate modified DTM data as discussed in Chapter 3, is shown in Figure 2.3. After the algorithms have processed this data, the resultant is a Vector Map representation of the defined terrain section. The three algorithms, referred to as the Bordering Algorithm, the Vector Encoding Algorithm and the Formatting Algorithm, operate in a serial fashion as is shown in Figure 4.1. All the algorithms were implemented in BASIC for application on the small computer system which functions as the simulated airfield computer described in the general solution discussion of Chapter 3. It should be noted, however, that for actual use, these algorithms would most probably be combined into a single, more efficient, FORTRAN program. The three were developed separately for the purposes of this Thesis in order to maximize the amount of memory available for array manipulation and to separate the distinct function of each algorithm.

The pre-flight processing creates the Vector Map representation by segmenting the input data area into different elevation regions. This is performed by the Bordering Algorithm which produces output data arrays of
Terrain Section

Elevation Data

(See Figure 2.3)

BORDERING ALGORITHM

Border Point Data Arrays

VECTOR ENCODING ALGORITHM

Vector Map

FORMATTING ALGORITHM

Functional Vector Map

2 K X 8 EPROM

Figure 4.1

Pre-Flight Processing
the same size as the input array. The output arrays of the Bordering Algorithm, however, reflect only the edges of each elevation region. The Vector Encoding Algorithm processes these output data arrays from the Bordering Algorithm converting each segment into its vector representation \((\Delta X, \Delta Y)\) and combining the segments of all elevation regions in order to produce a string of vector values forming a Vector Map. The final step of pre-flight processing is accomplished by the Formatting Algorithm which converts the Vector Map to a form compatible with the in-flight processing software. The Formatting Algorithm stores its Vector Map output in the 2 K x 8 EPROM which serves as the non-volatile transfer medium previously discussed.

4.2 Bordering Algorithm Description

The first task of the Bordering Algorithm is to input an integer data array whose values and position reflect a digitized model of a terrain section. The algorithm is then required to create a representation of this input data field by forming a set of closed isoelevation paths otherwise known as contour lines which are defined by a set of border points. The elevation level of each set of border points is determined by an operator selected contour interval which in turn effectively defines the resolution
of the resultant border representation.

The bordering process can be envisioned by imagining a terrain field sliced horizontally at fixed intervals. The resulting sections effectively create two types of data points. The set of points with elevation \( \geq X \) can be referred to as set \( S \). The remaining regions of points with elevation \(< X \) are therefore represented by the set \( S' \). In the digitized data field, the border criterion is concerned with the four adjacencies in proximity to a selected point. If these points are defined as;

\[
P_n, P_w, P_e, P_s
\]

and if each point is assigned a value \( =1 \) if \( P \in S \) and a value \( =0 \) if \( P \in S' \) then the border criterion can be expressed by the Boolean expression;

\[
P \land (P_n \lor P_s \lor P_e \lor P_w)
\]

(where if the expression \( =1 \), \( P \) is defined as a border point of the set \( S \))

To illustrate this expression consider the following neighborhoods of points;

<table>
<thead>
<tr>
<th>400</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>530</td>
<td>500</td>
</tr>
<tr>
<td>530</td>
<td>500</td>
</tr>
<tr>
<td>510</td>
<td>600</td>
</tr>
<tr>
<td>a.</td>
<td>b.</td>
</tr>
</tbody>
</table>

If the elevation level of interest is 500', the Boolean
expressions become:

\[ 1 \land (1 \lor 0 \lor 1 \lor 0) \quad \land (0 \lor 0 \lor 0 \lor 0) \]

Therefore the center point in a. is a border point by definition whereas the center point in b. is not. It should be noted that the expression evaluates only a single location at a time and thus allows isolated points to be determined as border point locations.

The results of the bordering process for the representative data field used in this Thesis are shown in Figures 4.2, 4.3, and 4.4. These figures represent the edges of the 300 foot cross section, 400 foot cross section and the 500 foot cross section respectively. The plots reflect an operator selected contour level of 100 feet. It should be noted that in the development of these borders, all regions beyond the edges of the map are considered to be equal to the minimum map elevation value, (200 ft.). Since no elevation information is assumed for locations beyond the edges of the mapped area, this default value is permissible. The bordering process uses this information when evaluating points at the edges of the map. As can be seen in Figures 4.2, 4.3, and 4.4 the effect of this 200' datum assumption is that all edge points with elevations greater than or equal to the elevation level of interest are determined to be border points. This effect can be analytically verified by the Boolean expression presented previously.
Figure 4.2

Border Detection Algorithm Output (300')
Figure 4.3

Border Detection Algorithm Output (400')

(* - denotes Border Point)
Figure 4.4

Border Detection Algorithm Output (500')

(* - denotes Border Point)
The central function of the bordering process is to identify the edges of the map regions with elevations greater than or equal to the elevation threshold being evaluated. In order to define these regions, all border points must be defined as members of one or more border point sets. Each border point set defines the edge of a specific map region. The fact that the processing assumes all elevations beyond the edge of the map have an elevation equal to the minimum map elevation (referred to as the map datum) forces most terrain structures to appear as mountains during the thresholding process. The implication of this aspect is that most border point sets will form the external edges of regions with elevations greater than or equal to the elevation level being evaluated. In some cases, however, a border point set can also define the internal edge of an elevated region which completely surrounds a region with elevations below the threshold level. If all border point sets are regarded as completely surrounding or enclosing a map region, this latter case can be considered a border point set enclosing a depression. A key aspect in the encoding of individual border points into border point sets (presented later in this Chapter) will be to create sets which form a closed external edge of a map region without addressing the elevation characteristics of the region. For this reason all border point sets may be regarded as simply enclosing a map
The operation of the basic program containing the Bordering Algorithm and shown in the commented listing of Figure 4.5 is represented by the functional flowchart shown in Figure 4.6. The format of the input data is as referenced in the flowchart. The data storage to tape which provides the input data for the subsequent vector encoding algorithm is defined by the format of Figure 4.7.
0 REM:  *******************************************************
1 REM:  "BORDERING ALGORITHM"
2 REM:  THIS PROGRAM ACCEPTS A SIMULATED DTM DATA ARRAY 21 X 11. IT
3 REM:  ACCEPTS AN OPERATOR INPUT BORDER ELEVATION INTERVAL AND
4 REM:  PRODUCES A I/O OUTPUT ARRAY FOR EACH ELEVATION LEVEL. IN
5 REM:  THE OUTPUT ARRAY A "1" INDICATES A BORDER POINT AT THAT
6 REM:  LOCATION. AN OUTPUT TO SCREEN & RS 232C PRINTER IS PROVIDED
7 REM:  WITH A "*" INDICATING A BORDER POINT.
8 REM:  *******************************************************
12 OPEN 126,2,3,CHR$(131)+CHR$(32) :REM: RS 232C PRINTER ********
15 OPEN 1,3 :REM: CRT ********
16 OPEN 2,1,0 :REM: TAPE INPUT ********
18 DIM Ti$(21,11) :REM: INPUT INTEGER ARRAY ********
19 DIM Ix$(21,11) :REM: OUTPUT ARRAY ********
20 S#=CHR$(32)
21 R#=CHR$(13)
22 FOR K=1TO21 :REM: BEGIN READ INPUT
30 FOR L=1TO11
40 INPUT I%,Ix$(K,L)
60 PRINT H,Ix$(K,L)
37 HEW X67 PRINT H,RI
61 NEXT K :REM: END READ INPUT
70 CLOSE 2
71 SML%=Ix$(1,1) :REM: BEGIN DETECTION OF MAX & MIN VALUES
76 LAR%=Ix$(1,1)
77 FOR K=1TO21
78 FOR L=1TO11
79 IF Ix$(K,L)>SML% THEN GOTO81
80 SML%=Ix$(K,L)
81 IF Ix$(K,L)<LAR% THEN GOTO83
82 LAR%=Ix$(K,L)
83 NEXT L
84 NEXT K
85 PRINT H,M MINIMUM VALUE = ";SML% :REM: DISPLAY MIN & MAX
86 PRINT H,M MAXIMUM VALUE = ";LAR% :REM: END DETECTION
87 INPUT "SELECT CONTOUR INTERVAL";INC% :REM: GET BORDER ELV. INTERVAL
88 CHK%=SML%+INC% :REM: OUTPUT TAPE HEADER
89 F=0:PRINT H,CHK%,R#
90 FOR K=1TO21 :REM: BEGIN VERTICAL BORDER PT. DETECTION
91 FOR L=1TO11
92 IF F=1 THEN 95
93 IF Ix$(K,L)<=CHK% THEN 97
94 TS(K,L)=X$:F=1:GOTO98
95 IF Ix$(K,L)>CHK% THEN 97
96 L=L-1:TS(K,L)=X$:L=L+1:F=0
97 TS(K,L)=Y$
98 NEXT L
99 L=L-1
100 IF F=0 THEN 105

Figure 4.5
Bordering Algorithm Program Listing
\begin{verbatim}
101 T$(K,L)=X$:L=L+1;F=0:00T0105
105 NEXT K  :REM: END VERTICAL BORDER PT. DETECTION *****
110 FOR L=1T011   :REM: BEGIN HORIZONTAL BORDER POINT DETECTION
111 FOR K=1T021
112 IF F=1 THEN 115
113 IF T$(K,L)<CHK% THEN 117
114 T$(K,L)=X$:F=1:00T0117
115 IF T$(K,L)>CHK% THEN 117
116 K=K-1:T$(K,L)=X$:K=K+1:F=0
117 NEXT K
118 IF F=0 THEN 120
119 K=K-1:T$(K,L)=X$:K=K+1:F=0
120 NEXT L :REM: END HORIZONTAL BORDER POINT DETECTION*****
125 FOR K=1T021:PRINT1,SS;:PRINT128,SS,SS,SS; :REM: BEGIN PRINT *****
129 FOR L=1T011
130 PRINT1,T$(K,L);SS,SS,SS,SS; :PRINT128,T$(K,L);SS,SS,SS;
131 NEXT L
132 PRINT1,R$:PRINT128,R$:FOR J=1T0308:NEXT J
133 NEXT K :REM: END PRINT *****
134 FOR J=1T060:NEXT J
135 FOR K=1T021 :REM: BEGIN CONVERT TO BINARY **************
136 FOR L=1T011
137 Z%=ASC(T$(K,L))
138 IF Z%=42 THEN 140
139 Z%=0:00T0141
140 Z%=1
141 PRINT3,Z%,R$: :REM: SAVE BINARY VALUE TO TAPE ***************
142 NEXT L
143 NEXT K :REM: END CONVERT TO BINARY **************
144 PRINT "PRESS C FOR NEXT CONTOUR LEVEL" :REM: GET PROMPT NEXT LEVEL
145 GET AS:IF AS=<"THEN 145
146 CHK%=CHK%+INC%;IF CHK%>LAR% THEN PRINT3,LAR%,R$:GOTO 150
147 GO TO 89 :REM: END PROGRAM IF NEXT LEVEL > MAX ELV. ********
150 CLOSE 128
151 CLOSE 1
152 PRINT3:CLOSE 3
153 END
\end{verbatim}

Figure 4.5 (Continued)

Bordering Algorithm Program Listing
START

Input Integer Data Array

Determine Maximum & Minimum Map Elev. Values

Display Max. & Min. Values; Get Elev. Interval

Save Max. & Min. Values


Save Border Elevation Level

Border Pt. Detector

(See Figure 2.3)

Elevation Interval

(See Figure 4.7)

(See Figure 4.7)

Figure 4.6

Bordering Algorithm Functional Flowchart
Figure 4.6 (Continued)

Bordering Algorithm Functional Flowchart
Figure 4.7

Border Detection Algorithm Tape Save Format
The operation of the border detection function is the central part of the Bordering Algorithm. This algorithm has been designed to detect border points at a specific border elevation by testing horizontal and vertical adjacencies in two subsequent passes over the input data array. The algorithm passes over the input data array from top to bottom by rows in order to detect vertical border points. If a point is a border point, it is written into a resultant array as the ASCII representation of "*" otherwise an ASCII "SPACE" is written. Once the entire input data array has been processed for vertical border elements the next portion of the Bordering Algorithm passes over the input data array by columns from left to right. When a border point is detected an ASCII "*" is written into the resultant array. After the latter portion of the Bordering Algorithm has passed over the input data array, the resultant array represents the logical OR of all vertical and horizontal border points. The fact that the Bordering Algorithm produces a character resultant array is only due to the desire to provide a distinct and simple print out. As Figure 4.7 shows the resultant array written to tape has been converted by the program to a more efficient integer resultant array where a "1" indicates a Border point location and a "0" indicates a non-border point location.

The exact methodology of the Bordering Algorithm is
clarified by the sketch presented in Figure 4.8. This sketch shows the vertical border point detector outputs for a typical input data array row.
Figure 4.8

Bordering Process
Points (0) and (12) represent the minimum elevation datum assigned to all non-array positions. The flow chart shown in Figure 4.9 defines the exact operation of the vertical and horizontal border point detector. In general, however, the border detection mechanism initializes itself to the minimum elevation present in the input data array corresponding to the map datum. As the border detection mechanism steps point by point across (or down) the data array it compares each point encountered to the border elevation. It labels the first point encountered that is ≥ the border elevation as a border point (a step up condition). It then continues to step across (or down) the array searching for the first point below the border elevation value. When such a point is encountered (a step down condition) it labels the point encountered immediately before the present point as a border point. Selecting the datum elevation as the minimum elevation value serves to ensure that any step up border point has a corresponding step down border point in the same row or column. It should be noted, however, that some points may represent more than a single step (See column 7, Figure 4.8). This has the important result of forcing all isoelevation border points to form a closed contour line. For an example of the outputs from the vertical and horizontal border detection sections see Figures 4.10 and 4.11.
Border Point Detector (Vertical Section)
Figure 4.9 (Continued)

Border Point Detector (Horizontal Section)
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>4</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>5</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>6</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>12</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>13</td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>14</td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>15</td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>16</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>17</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>18</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>19</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>20</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

(* - denotes Border Point)

Figure 4.10

Vertical Border Point Detector Output
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

(* - denotes Border Point)

**Figure 4.11**

Horizontal Border Point Detector Output
4.3 Vector Encoding Algorithm Description

The task of the Vector Encoding Algorithm is to reduce the border points of a given border elevation into a set of vectors. Each set of border points is reduced by this algorithm to a string of resultant vectors of the form shown in Figure 4.12. Due to the fact that each set of border points completely encloses a region, the following relationship is produced.

\[ O = \sum_{n=1}^{N} \Delta x_n \]
\[ O = \sum_{n=1}^{N} \Delta y_n \]

In these expressions \( \Delta x \) and \( \Delta y \) are the cartesian components of the \( n \)th resultant vector for a border point set having \( N \) resultant vectors. It should be noted that in the development of the Vector Encoding Algorithm all coordinates and offsets are specified in integer representation using the unit distances specified by the row and column positions in the input data array.

The Vector Encoding Algorithm operates by scanning for a starting point on the edge of the region of points described by a set of border points. This starting point is predictably determined by the scanning method utilized. Once the starting point on a set of border points is found, the encoding algorithm proceeds to step point by point around the border of a region of points enclosed by a given set of border points. This process is necessarily complicated in any complex border point array due to the
Figure 4.12
Vector Encoded Border Point Set
fact that at a single border elevation level, several sets of border points each enclosing different regions may be present. For example in the sketch shown in Figure 4.12 two regions A and B exist at the same elevation level. In the more complex instances, a set of border points may enclose regions which share border points. A region defined by a set of border points may also completely contain a region defined by another set of points. Referencing Figure 4.12, both of these cases are present since region A includes region B (a depression) and the border point sets are adjacent at the shared border point z.

One critical aspect of the stepping process, therefore, is to provide the capability of stepping to border points of only the same region. The stepping process is terminated when the Vector Encoding Algorithm returns to the starting point of that specific set of border points. This event is guaranteed by the closed property of each set of border points. When the starting point is reached, the algorithm has completely enclosed a specific region of points as is shown in Figure 4.13.

The second aspect of the Vector Encoding Algorithm is the reduction of the border point data array into a vector representation. To accomplish this the algorithm continually tracks its direction of travel as it steps through the border data array. When a step is taken which causes a change of direction the algorithm calculates the
Figure 4.13

Vector Encoding Algorithm Operation
\( \Delta X \) and \( \Delta Y \) offsets in terms of unit differences from the last point which caused a change of direction. These offsets represent a vector which exactly reflects the magnitude and direction of a particular straight line segment of a given set of border points contained in the border data array. It is these vectors, saved as a string of integers, which constitute the input to the Formatting Algorithm. The functional flowchart of Figure 4.14 depicts the general operation of the Vector Encoding Algorithm.

The initial task of the Vector Encoding Algorithm is to determine a starting location on a set of border points contained in the border point data array of a specific elevation level. This is accomplished by a section of the Vector Encoding Algorithm referred to as "SCANNER". This section scans a particular border point data array by row top to bottom and will detect the first border point location which is a "1". An exit to the next program section occurs when the first "1" is found, implying the start of a border point set enclosing a region. An alternate exit occurs when no "1"s are found and this condition implies all border points of a given border elevation level have been processed.

Once a starting point is found the next operation is to read the border point data array values for all eight neighboring points of a given border point. The positional labelling used for this operation is defined as;
START


Read Border Point Data Array

Output Border Point Data Array to Screen

"SCANNER" (Finds Border Pt. Set Start Pt.)

"NEIGHBOR" (Finds Contents of 8 Neighbor Points)

"DECIDER" Determines Next Step Point

Next Step = Change of Direction ?

Move to Next Point

N

N

Current Position = Start Point ?

Calculate & Save ΔX, ΔY

Calculate & Save Final ΔX, ΔY

Print Processed Border Point Data Array

Read Next Border Elev.

Border Elev ≥ Largest Elev. ?

Save All Vectors

END

(No unused Border Pts.)

Y

A

Figure 4.14

Vector Encoding Algorithm Mainline Flowchart
This function is performed by a program subroutine called "NEIGHBOR" which returns the values of the neighboring points in an eight element linear array.

When the neighboring points of a given border point are obtained, the algorithm then utilizes a subroutine called "DECIDER" to determine which adjacent point the algorithm should step to. With the information returned by "DECIDER" the algorithm first determines if the border point was an isolated point. If the point was isolated, a return to "SCANNER" occurs. For a non-isolated point, the algorithm steps to the next point and adjusts the direction of travel if necessary. The prior point is overwritten in the border point data array with a sequence number which starts at value =2 and increments at each step. If a change of direction has occurred, the previous point is stored as the last change of direction point and vector offsets are calculated from the prior last change of direction point. If the new point matches the starting point, the set of border points enclosing a region has been completely stepped and the final vector offsets are calculated from the last change of direction point. Vector offsets when calculated are stored in a resultant data array. If the start point has not been reached, control reverts to
"NEIGHBOR". If the start point is reached control reverts to "SCANNER" to locate the start point of the next set of border points.

The detailed operation of "SCANNER" is straightforward and is effectively described by the flowchart of Figure 4.15. The operation of "NEIGHBOR" is somewhat more complicated by having to deal with all possible edge directions of the border point data array. "NEIGHBOR" efficiently accommodates this problem by accessing a series of eight subroutines each designed to load the value of a single neighbor point position into its relative position in an eight element linear array. By insuring that the linear array is cleared initially and by use of the present position of the algorithm, "NEIGHBOR" inhibits any attempt to evaluate points outside the border point data array. For detailed flowchart of "NEIGHBOR" see Figure 4.16.

The most critical section of the Vector Encoding Algorithm is located in the subroutine "DECIDER". This solution is responsible for the determination of isolated points and for the decision making capability for identifying the next point and next direction of travel for the Encoding Algorithm. "DECIDER" begins operation with the values of all eight neighboring points stored in a linear array by "NEIGHBOR". "DECIDER" examines these neighboring points first for a 1 (new border point) or 2 (starting border point). If no such points are found, "DECIDER"
Figure 4.15
"SCANNER" Flowchart
Figure 4.16

"NEIGHBOR" Subroutine Flowchart
Figure 4.16 (Continued)

"NEIGHBOR" Subroutine Flowchart
"NEIGHBOR" Subroutines (Typical):

SUBROUTINE # 1

1

\[ P_{LL,LK-1} > 0 \]?

N

\[ N \% (1) = P_{LL,LK-1} \]

RETURN

Y

Similar Subroutines for the other six adjacent locations are not shown.

8

1 2

7 \[ P_{LL,LK} \]

3

6 5 4

SUBROUTINE # 2

2

\[ P_{LL+1,LK-1} > 0 \]?

N

Y

\[ N \% (2) = P_{LL+1,LK-1} \]

RETURN

Figure 4.16 (Continued)

"NEIGHBOR" Subroutine Flowchart
checks for any previously overwritten (used) point in the border point data array. It is this sequence of detection that forces the stepping process to primarily step to unused border points until the start point is detected.

Once a specific type of point is detected, "DECIDER" then proceeds to examine all adjacent points in a specific sequence. It is this sequence of examination that keeps the algorithm moving around the outside of a given region even in the presence of various complex border (terrain) configurations. The exact method of operation can be seen for a given point BP as defined in the following sketch.

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>BP</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>LP</td>
</tr>
</tbody>
</table>
```

In this example a step from LP to BP has occurred with the direction of travel (direction=8) specified by the arrow. "DECIDER" first determines a search start point (position 5) which is 225° clockwise from the direction of travel. This biases the search to a position immediately outside the enclosed elevation region and keeps the algorithm from "chasing its tail". From the start position neighboring points are searched clockwise for the first appropriate point (usually an unused border point). In this
example, "DECIDER" will discover the BORDER point in position 7. Once an appropriate point is found "DECIDER" will use this information to establish the next step point in this example as point $X=BP_{x-1}$, $Y=BP_y$ and next direction =7. It is this information that the central part of the algorithm will use to make its next step.

The Vector Encoding Algorithm with its attendant subroutines has been shown to operate successfully on relatively complex terrain configurations such as that previously presented in this Thesis. Figures 4.17, 4.18 and 4.19 show the stepping mechanism in actual operation around the terrain structures of the 300', 400' and 500' border elevation levels. The sequence of stepping can be determined by the sequence numbers which have been overwritten into the original data array. The vector data created is partially noted on Figure 4.17 and the exact format of the data produced is shown in Figure 4.20. A listing of the Vector Encoding Algorithm is shown as Figure 4.21.

4.4 Formatting Algorithm Description

After the border point arrays created by the Bordering Algorithm have been reduced to a vector representation by the Vector Encoding Algorithm, it is necessary to store this representation in a non-volatile manner. For purposes
Input Border Point Data Array

Vector # 1
\[ \Delta X = 4, \Delta Y = 0 \]

```
0 0 0 0 0 0 2 3 4 5 6
0 0 0 0 0 0 37 0 0 0 7
0 0 0 0 0 0 36 2 3 0 8
0 0 31 32 0 0 0 7
0 0 30 0 33 34 0 6 5 0 10
0 0 29 28 27 26 0 0 0 0 11
0 0 0 0 0 0 25 0 0 0 12
0 0 0 0 0 0 24 0 0 0 13
0 0 0 0 0 0 23 0 0 0 14
0 0 0 0 0 0 22 0 0 0 15
0 0 0 0 0 0 21 0 0 0 16
2 3 4 5 6 0 0 0 20 19 17
31 0 0 0 0 7 0 0 0 0 18
30 0 0 0 0 8 0 0 0 0 0
29 0 0 0 0 9 0 0 0 0 0
28 0 0 0 0 10 0 0 0 0 0
27 0 0 0 0 11 0 0 0 0 0
26 0 0 0 0 0 12 0 0 0 0
25 0 0 0 0 0 13 0 0 0 0
24 0 0 0 0 0 0 14 0 0 0
23 22 21 20 19 18 17 16 15 0 0
```

Vector # 2
\[ \Delta X = 0, \Delta Y = -12 \]

```
Completed Border Point Data Array
```

* Note; Point \( \boxed{7} \) is shared by two Border Point sets.

Figure 4.17

300' Border Elevation Processing
Figure 4.18

400' Border Elevation Processing
Figure 4.19

500' Border Elevation Processing
<table>
<thead>
<tr>
<th>Length of Data Array</th>
<th>Base Elev. (Smallest)</th>
<th>Border Pt. Set #1 Elev.</th>
<th>Set #1 Start X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set #1 Start Y</th>
<th>Set #1 $\Delta X_1$</th>
<th>Set #1 $\Delta Y_1$</th>
<th>Set #1 $\Delta X_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta X_n$</td>
<td>$\Delta Y_n$</td>
<td>Border Pt. Set #2 Elev.</td>
<td></td>
</tr>
<tr>
<td>Set #2 Start X</td>
<td>Set #2 Start Y</td>
<td>Set #2 $\Delta X_1$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set #N $\Delta X_n$</th>
<th>Set #N $\Delta Y_n$</th>
<th>Termination Elev. (Largest)</th>
</tr>
</thead>
</table>

Figure 4.20
Vector Encoding Algorithm Output Data Format
Figure 4.21
Vector Encoding Algorithm Program Listing
Figure 4.21 (Continued)
Vector Encoding Algorithm Program Listing
870 \texttt{U=\text{LK}:V=\text{LL}+1} \tag{REM: START "NEIGHBOR" SUBR.}
871 \text{IF } \text{IX}(U,V)>0 \text{ THEN } \text{NX}(3)=\text{IX}(U,V)
872 \text{RETURN}
873 \text{U=\text{LK}+1:V=\text{LL}+1}
874 \text{IF } \text{IX}(U,V)>0 \text{ THEN } \text{NX}(4)=\text{IX}(U,V)
875 \text{RETURN}
876 \text{U=\text{LK}+1:V=\text{LL}}
877 \text{IF } \text{IX}(U,V)>0 \text{ THEN } \text{NX}(5)=\text{IX}(U,V)
878 \text{RETURN}
879 \text{U=\text{LK}+1:V=\text{LL}-1}
880 \text{IF } \text{IX}(U,V)>0 \text{ THEN } \text{NX}(6)=\text{IX}(U,V)
881 \text{RETURN}
882 \text{U=\text{LK}:V=\text{LL}-1}
883 \text{IF } \text{IX}(U,V)>0 \text{ THEN } \text{NX}(7)=\text{IX}(U,V)
884 \text{RETURN}
885 \text{U=\text{LK}-1:V=\text{LL}-1}
886 \text{IF } \text{IX}(U,V)>0 \text{ THEN } \text{NX}(8)=\text{IX}(U,V)
887 \text{RETURN}
888 \text{U=\text{LK}-1:V=\text{LL}}
889 \text{IF } \text{IX}(U,V)>0 \text{ THEN } \text{NX}(1)=\text{IX}(U,V)
890 \text{RETURN}
891 \text{U=\text{LK}-1:V=\text{LL}+1}
892 \text{IF } \text{IX}(U,V)>0 \text{ THEN } \text{NX}(2)=\text{IX}(U,V)
893 \text{RETURN} \tag{REM: END "NEIGHBOR" SUBR.}
894 \text{FOR } I=1 \text{ TO } X \tag{REM: START CRT DISPLAY}
895 \text{FOR } J=1 \text{ TO } 11
896 \text{PRINT} I, \text{IX}(I,J); \tag{REM: END CRT DISPLAY}
897 \text{NEXT } J
898 \text{NEXT } I
899 \text{RETURN} \tag{REM: END CRT DISPLAY}
of this Thesis, EPROM storage was utilized. Secondary to the storage medium, however, is the importance of modifying the Vector Encoding Algorithm output data to a format compatible for utilization by the map processor Look Up Algorithm. This algorithm is resident in the map processor system as described in the next chapter. A short BASIC language program was written for this purpose and is shown in the listing of Figure 4.22.

The formatting program completely reads the output data of the Vector Encoding Algorithm and performs two tasks. The first task is to convert the border elevation values to tens of feet in place of the original units of feet representation. In addition the border elevation levels sacrifice the Least Significant Bit (LSB) of each byte for use by the map processor Look Up Algorithm as an exact flag bit. Although this program does not reflect this latter aspect (since the example border elevations are in increments of 100 feet) the new effect of both restrictions are to limit the Border elevation to a range of values from 0-2540 feet in a minimum, 20 foot increment size. The second task of the formatting program is to convert the vector offsets to a sign-magnitude representation. This is required solely to facilitate easy manipulation by the map processor Look Up Algorithm. The net effect of this aspect is that the range of values acceptable for vector offsets is ± 127 units.
REM: "FORMATTING PROGRAM"
REM: PROM. CHANGES VECTOR ENCODING ALGORITHM OUTPUT DATA TO SIGN
REM: MAGNITUDE FORMAT. BORDER ELEVATION VALUES ARE IN 10'S OF FEET.
REM: INSERTS AUTO-STARTING HEADER. PROVIDES PRINT OUT.

OPEN 4,4
DIM R%(700)
R$=CHR$(13)
INPUT#2,0
FOR C=0TO799
INPUT#2,R%(C)
NEXT C
Z=0
FOR C=0TO799
Z=Z+1
IF Z>9 THEN Z=0:PRINT#4,R$:
PRINT#4,R%(C):
NEXT C
PRINT#4:CLOSE4
POKE 26624,0
POKE 26625,72
POKE 26626,0
POKE 26627,0
POKE 26628,65
POKE 26629,48
POKE 26630,195
POKE 26631,194
POKE 26632,285
D=26633
FOR C=0TO799
IF R%(C)>100 THEN R%(C)=R%(C)/10:REM: CONV. TO 10'S OF FEET
IF R%(C)<0 THEN R%(C)=-R%(C)+128:REM: CONV. TO SGN-MAG.
POKE D,R%(C)
D=D+1
NEXT C
CLOSE2
END

Figure 4.22
Formatting Algorithm Program Listing
The Vector Encoding Algorithm output data is shown in Figure 4.23 with the output of the formatting program. Figure 4.24 shows the hexadecimal values of the formatting program output as they actually were stored on the demonstration EPROM used in this Thesis. It is this 107 byte data block representation of the original 11 x 21, 231 integer input data array which contains all the terrain information utilized by the map processor Look Up Algorithm.

The presentation in this chapter has covered the essential concepts necessary to describe the terrain data to Vector Map conversion process. The specific example utilized in this development, however, should not be regarded as comprehensive. Appendix C has been provided to show an alternate example of Vector Map development which involves several aspects beyond the scope of this chapter.
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 200| 300| 1 | 7 | 4 | 0 | 0 | 12 | -1 | -1 | -1 | 0 | -2 | -2 | 0 | -3 |
| -1 | -1 | -3 | 0 | 0 | -2 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | -1 | 0 | -3 | 300 |
| 3  | 8  | 1 | 0 | 1 | 1 | -1 | 1 | -1 | 0 | -1 | -1 | 1 | -1 | 300 | 12 | 1 |
| 4  | 0  | 1 | 1 | 0 | 3 | 2 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | -8 | 0 | 0 | -9 |
| 400| 5  | 4 | 1 | 0 | 0 | 1 | -1 | 0 | 0 | -1 | 400 | 15 | 1 | 3 | 0 | 0 |
| 3  | 2  | 2 | 0 | 1 | -5 | 0 | 0 | -6 | 500 | 18 | 1 | 2 | 0 | 0 | 2 | 1 | 1 |
| -3 | 0  | 0 | -3 | 550 |

*Note; Values are Integers ( 2 Bytes / value )

Figure 4.23
Vector Encoding Algorithm Output Data
<table>
<thead>
<tr>
<th>DATUM ELEV.</th>
<th>BORDER PT. ELEV.</th>
<th>START X</th>
<th>START Y</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>30</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>129</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>129</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>130</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>129</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>131</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>129</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>129</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>129</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>129</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td>129</td>
</tr>
<tr>
<td>30</td>
<td>12</td>
<td>1</td>
<td></td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>136</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>137</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>5</td>
<td>4</td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>129</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>129</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>15</td>
<td>1</td>
<td></td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>133</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>134</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>18</td>
<td>1</td>
<td></td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>131</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>131</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Maximum Elevation 55

Figure 4.23 (Continued)

Vector Encoding Algorithm Output Data
<table>
<thead>
<tr>
<th>Address</th>
<th>Data Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>A008 XX</td>
<td>14 1E 01 07 04 00 00</td>
</tr>
<tr>
<td>A010 0C</td>
<td>81 81 81 00 82 82 00</td>
</tr>
<tr>
<td>A018 83</td>
<td>81 83 00 00 82 01</td>
</tr>
<tr>
<td>A020 00</td>
<td>01 01 00 01 81 00</td>
</tr>
<tr>
<td>A028 83</td>
<td>1E 03 08 01 00 01 01</td>
</tr>
<tr>
<td>A030 81</td>
<td>01 81 11 81 81 01 81</td>
</tr>
<tr>
<td>A038 1E</td>
<td>0C 01 04 00 01 01 00</td>
</tr>
<tr>
<td>A040 03</td>
<td>02 02 00 01 01 01 00</td>
</tr>
<tr>
<td>A048 01</td>
<td>88 00 00 89 28 05 04</td>
</tr>
<tr>
<td>A050 01</td>
<td>00 00 01 81 00 00 81</td>
</tr>
<tr>
<td>A058 28</td>
<td>0F 01 03 00 00 03 02</td>
</tr>
<tr>
<td>A060 02</td>
<td>00 01 85 00 00 86 32</td>
</tr>
<tr>
<td>A068 12</td>
<td>01 02 00 00 02 01 01</td>
</tr>
<tr>
<td>A070 83</td>
<td>00 00 83 37 XX XX XX</td>
</tr>
</tbody>
</table>

Figure 4.24

EPROM Map Processor Data
CHAPTER 5
MAP PROCESSOR SYSTEM DESCRIPTION

5.1 Overview of In-Flight Operation

The in-flight operation of the map processor system is presented in its general form by the system block diagram shown in Figure 5.1. As this figure shows, the map processor system is a 6502 microprocessor based system which operates as a stand alone peripheral to the simulated aircraft computer. The simulated aircraft computer is the small personal computer described in section 3.2. During operation, the aircraft computer periodically provides to the map processor system a coordinate of a Vector Map location with unknown elevation. In the case of the test system this coordinate is directly entered into the simulated aircraft computer by the operator. This coordinate, is entered in its Vector Map cartesian format (X,Y).

The reader will recall that in an actual aircraft system the Vector Map coordinate is determined from a positions' geographic coordinate as an imbedded function of the aircraft computer (See Figure 3.1). Once transferred to the map processor the unknown elevation is determined by a special purpose Look Up Algorithm resident in the map processor. The elevation value is then returned to the simulated aircraft computer via the same bidirectional parallel communication bus that handled the coordinate transfer. In steady state operation, the map processor
Figure 5.1
Vector Map Processor Test System
is either processing an elevation or waiting for the next coordinate transfer. All bidirectional transfers are initiated and controlled by the simulated aircraft computer.

A description of the map processor system begins with a presentation of the system hardware. In this section, all construction details of the test system hardware are described. The operation of the test system is then described by its resident software and more specifically by the operation of the Look Up Algorithm. This algorithm is separately presented from the I/O operational software of the map processor system. The final relevant area of the microprocessor based test system is contained in the interface description which details the operation of the bidirectional parallel bus.

5.2 System Hardware Definition

The map processor system was developed from readily available, commercially manufactured circuitry. Specifically a 6502 based single board computer was modified for the task of implementing the Vector Map Look Up Algorithm and the necessary I/O functions. The required circuitry utilized by the map processor system includes only a small portion of the available resources of the single board computer. A schematic of this computer is shown in Reference 6. Actually designed as a small personal
computer oriented to BASIC language programming, the entire operating system of the single board computer was bypassed in order to reconfigure the system into a small, dedicated map processing peripheral. In this mode all the original I/O functions including magnetic tape and CRT interface were eliminated in order to utilize only a single bidirectional communication bus. The adaptation of such a system for the implementation of the map processor serves to emphasize the versatility of the Vector Map concept while providing a minimum cost demonstration system.

The map processing test system only utilizes a portion of the available resources of the single board computer. As is shown in Figure 5.2 the operational resources of the test system are directed to supporting the 6502A microprocessor which is the system Central Processing Unit (CPU). In the test system the CPU operates at a 1.1 MHz clock rate provided by a crystal controlled oscillator section. System power is provided at +5 VDC from a rectified 115VAC, 60 cycle power source. The operational program of the CPU, once initialized, is furnished by a 2716, 2 K x 8, EPROM containing the machine language implementation of the Vector Map Look Up Algorithm. No portion of the 8 K x 8 Resident Operating System ROM (including subroutines) is utilized by the test system when in steady state operation. The Vector Map containing the terrain representation as previously developed is provided
SYSTEM DATA BUS

Clock Gen. 1.1 MHz

6502A CPU

O/S ROM 8Kx8, (E000-FFFF)

RAM 5Kx8 (0000-1FFF)

SYSTEM ADDRESS BUS

2 - 58725 (2Kx8)
2 - 2114 (1Kx4)

2 - 58725 (2Kx8)
2 - 2114 (1Kx4)

24 Pin Connector

PB0-PB7 CB1 CB2 PA2

52 Pin Connector

MAP PROCESSOR

6522 VIA (9110-911F)

6522 VIA

EPROM EPROM 2716, 2Kx8 (4800-48FF)

VECTOR MAP EPROM 2716, 2Kx8 (A000-A7FF)

SIMULATED AIRCRAFT COMPUTER

Figure 5.2
Map Processor Functional Schematic
by an additional 2716, 2 K x 8 EPROM. The map processor test system has a total of 5 K x 8 static RAM available for operational use. Of this memory only the small amount (less than 100 bytes) necessary for "scratch pad" and system Stack requirements are actually used.

All interfacing to the map processor test system is performed by a 6522 Versatile Interface Adapter (VIA). The VIA provides the CPU direct memory mapped control of the 11 line, bidirectional communication bus used by the test system. In the demonstration system, the map processor VIA is directly interfaced by ribbon connector to the VIA resident in the simulated aircraft computer.

The memory map of the Map Processor test system is shown in Figure 5.3. The lowest 1K Bytes of memory are reserved for system stack usage during subroutine operations. The remaining 4K Bytes of RAM ($0FFF-$1FFF) are available as a computational "scratch pad". The central operating program of the system is 2K Bytes of EPROM memory at addresses $4800 to $4FFF. The 6522 VIA operational registers occupy a 16 byte block of memory space from addresses $9100 to $911F. The Vector Map terrain representation is stored in a second 2K Byte EPROM section ($A000-$A7FF). The map processor memory map is completed by the 8K byte section of ROM ($E000-$FFFF) resident at the top of the 6502 memory space.

The final aspect of the map processor test system
Figure 5.3
Map Processor System Memory Map

* Note; All Address Values are Hexadecimal
hardware is the reconfiguration of the system at the time power is initially applied. This reconfiguration serves to change the single board computer system into its map processor form by effectively bypassing the normal operating system of the computer. When power is applied and the system is released from its RESET condition, the 6502 CPU is vectored to address $FFFC. From this address, the CPU is sent to an initialization routine resident in the Operating System ROM at address $FD22. This routine, shown in its assembly language version on the first page of Appendix A, calls a subroutine at address $FD3F which checks the first 5 bytes of the Vector Map EPROM beginning at $A004. If these 5 bytes match a specific 5 byte sequence in the Operating System ROM beginning at $FD4D, the subroutine will return to the calling routine with the zero bit of the 6502 status register set. The initialization routine tests this bit and if it is found set the CPU will be vectored to the address indicated at Vector Map location $A000. Assuming a valid Vector Map was present, this action sends the CPU to the start of the map processor program at address $4B00 and no further operating system software is utilized.
5.3 System Software Definition

The steady state operation of the map processor test system is defined by its operational software stored in the 2 K x 8 map processor EPROM shown in Figure 5.2. This 6502 machine code memory section is directly responsible for the communication with the simulated aircraft computer and for the central task of determining the elevation of a specific Vector Map coordinate. Written in 6502 assembly language in order to maximize the efficiency of the map processor system, the source code of this memory section can be subdivided into distinct functional areas or modules. Figure 5.4 indicates the functional areas of the map processor program in regard to their specific locations in memory. In the test system approximately 1 K bytes of the available EPROM memory is used for the map processor program.

The operation of the map processor and the inter-relationship of its functional modules can be clearly examined by characterizing the map processor as a Finite State Machine. In this characterization, the operation of the map processor may be segmented into five distinct states of operation as is shown in Figure 5.5. Of the five states, states I, III and V are interface related. The software modules responsible for system operation in these states are referred to as "INITIALIZATION", "HANDSHAKE IN" and "HANDSHAKE OUT" respectively. These modules are
Figure 5.4
Map Processor Program Memory
POWER ON
RESET

I

II

III

IV

V

a

b
c
d
e

<table>
<thead>
<tr>
<th>STATE/EXIT CONDITION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Initialization State</td>
</tr>
<tr>
<td>II</td>
<td>Map Data Set Up State</td>
</tr>
<tr>
<td>III</td>
<td>X,Y Coordinate Input State</td>
</tr>
<tr>
<td>IV</td>
<td>Map Processing State</td>
</tr>
<tr>
<td>V</td>
<td>Elevation Value Output State</td>
</tr>
<tr>
<td>a</td>
<td>Initialization Complete</td>
</tr>
<tr>
<td>b</td>
<td>Map Data Initialized</td>
</tr>
<tr>
<td>c</td>
<td>X,Y Coordinate Received</td>
</tr>
<tr>
<td>d</td>
<td>Elevation Determined</td>
</tr>
<tr>
<td>e</td>
<td>Elevation Transfer Complete</td>
</tr>
</tbody>
</table>

Figure 5.5
Map Processor State Graph
primarily concerned with the proper initialization and exchange of data across the bidirectional parallel bus. Their operation is described in Section 5.4. The remaining states II and IV are responsible for the actual terrain elevation determination using the Vector Map data. These states are controlled by the operational implementation of the Look Up Algorithm which represents the most critical aspect of map processor operation.

In operation, the map processor enters State I immediately after the power on initialization described in the previous section. Once the software module "INITIALIZATION" has determined the simulated aircraft processor is operational, the map processor enters State II where the Look Up Algorithm sets up various program parameters necessary to begin elevation processing. Control then transfers to State III where the map processor waits for the submission of an unknown coordinate via "HANDSHAKE IN". When a coordinate is received from the simulated aircraft computer State IV is entered and the elevation determination is made by the Look Up Algorithm. Once the elevation determination is complete the system enters State V where the "HANDSHAKE OUT" module transfers the elevation value to the simulated aircraft computer. State V completes the operational cycle of the map processor and the system then returns to State II for the preparations necessary to process the next submitted coordinate.
The Look Up Algorithm which controls the operation of the map processor test system in States II and IV represents the most significant aspect of the map processor program. As is shown in Figure 5.5, the portion of the Look Up Algorithm operating when the map processor is in State II is responsible for setting up initial map parameters. The operation of the map data initialization section ("NEXT XY") of the Look Up Algorithm simply fills a portion of RAM memory with the Vector Map data necessary to begin processing a submitted coordinate. The specific values loaded are of two types. The first type of data that is initialized is the general map data parameters consisting of the map datum elevation and the border elevation interval. The second type of data that is initialized is the additional data necessary to begin processing the initial border point set of the lowest elevation level. These parameters are the elevation level, starting X coordinate and the starting Y coordinate. This section also clears all algorithm flags and sets the elevation value of any unknown point equal to the map datum elevation. As a final task "NEXT XY" stores the 8 bit index to subsequent map data prior to moving the map processor to State III. The initialization module "NEXT XY" is utilized after initial power on and after any requested elevation has been output. When the module is completed the algorithm is ready to begin the elevation determination process as
soon as specific X, Y coordinates are received.

The major portion of Look Up Algorithm operation occurs during the map processor State IV. This state is entered when the map processor receives the coordinates of a point with unknown elevation. It is during this state that the actual elevation determination is made for the submitted unknown point. The Look Up Algorithm makes this determination by examining the map regions defined by the vectors of each border point set. The algorithm seeks to determine if the known point lies within the map region with terrain elevation greater than or equal to the elevation level in process. The fundamental task of the algorithm is to decide whether an unknown point is enclosed by a given border point set and to adjust its elevation accordingly. A flowchart showing the general operation of the Look Up Algorithm when performing this function is presented as Figure 5.6.

The method utilized by the algorithm to determine the relative location between border point set and submitted point is based on the point by point evaluation of each border point represented by the vectors of a set. The algorithm sequentially steps a unit distance from border point to border point around a specific map region in a clockwise direction. After each step the algorithm examines for a match of X coordinates between its current location and the unknown point. When a match is found the Y
Figure 5.6

Look Up Algorithm General Operation
coordinate is evaluated. If both X and Y coordinates match, the unknown point is a border point and its elevation is adjusted to the value of the elevation level being processed. If the Y coordinate does not match that of the submitted point, the algorithm then seeks to determine if the stepping process has passed above and below the point during consecutive X coordinate matches. This condition, which may be described as bracketing the submitted location, forces an update of its elevation. Should the elevation of the submitted point be below the elevation level in process, its value is updated to that of the elevation level. If the submitted point elevation is already at the elevation level in process, then it is lowered by one elevation interval. It should be noted that more than one bracket condition is possible for a single border point set modelling complex terrain. The elevation of a submitted point may "toggle" between values several times during the processing of a single border point set.
The elevation status of a submitted point cannot be determined until all border point sets of a given elevation level are processed. The Look Up Algorithm is designed to adjust the unknown elevation value for submitted points which lie within the regions defined by one or more border point sets of the same elevation level. This is necessary in order to correctly define those regions whose edges are identified by more than one border point set. The algorithm accomplishes this by extending the bracketing and elevation adjustment process through all border point sets of a given elevation level. When the elevation level is moved upward by one elevation interval after all border point sets of the previous level have been evaluated, a minimum elevation value for the submitted point is effectively established. Subsequent processing at higher elevation levels cannot lower this minimum elevation value.

The final elevation value for a submitted point is determined when all border point sets in the Vector Map have been evaluated. A single exception is the case where an entire level has been processed without a net increase in the elevation of the submitted point. For such instances no subsequent changes in elevation are possible for higher elevation levels and map processing is determined complete. This early determination feature results in a significant reduction in average processing time for the algorithm since the elevation of most points is determined before the
higher elevation levels are processed.

The operation of the Look Up Algorithm is functionally described by the alternate example shown in Figure 5.7. In this figure the elevation of an input coordinate is determined using the sequential stepping and decision making logic present in the Look Up Algorithm. The elevation determination process begins with the initial conditions shown in Figure 5.7 a. The map processor initiates operation on the submitted coordinate from the X, Y Coordinate Input State (State III, Figure 5.5). In this instance the point with unknown elevation lies in a depression having an elevation between 100 and 200 feet. This depression is surrounded by terrain having an elevation between 200 and 300 feet. The Vector Map of this region is formed by the border point sets enclosing surface #1 and the subset surface #2. As operation begins the Look Up Algorithm is initialized at the beginning of the Vector Map designated starting point of surface #1.

As the elevation determination process begins, the Look Up Algorithm begins stepping clockwise along the initial vector segment of surface #1. As successive comparisons between the algorithm present position X coordinate and the input point X coordinate are made a
Starting Point

Input Coord.

Border Surface #1

Border Surface #2 (Depression)

*Note; Datum Elevation = 100'

Initial Condition (UNKELV=100')

Crossover Condition (UNKELV=100')

1st Enclosure (UNKELV=200')

Surface #1 Complete (UNKELV=200')

Crossover Condition (UNKELV=200')

2nd Enclosure (UNKELV=100')

Figure 5.7

Elevation Determination Example
Figure 5.7 (Continued)

Elevation Determination Example
match is found at the location shown in Figure 5.7 b. At this point the algorithm sets its internal flags to record the crossover condition. Since this point constitutes the initial crossover for the processing of surface #1 no change in the elevation of the unknown location is made. It should be noted that the algorithm initialized this unknown elevation to the 100' map datum elevation as part of its operation in the Map Data Set Up State (State II, Figure 5.5).

The Look Algorithm continues the stepping process until the second crossover location is detected on the third vector segment of surface #1. This condition is shown in Figure 5.7 c. The algorithm at this point checks its internal flags and determines that since two successive crossovers of opposite sense (one above and one below) have occurred, the unknown elevation point has been bracketed. This condition causes the algorithm to update the unknown elevation value to 200' before continuing the stepping process. No further crossovers occur as the algorithm returns to the starting point as shown in Figure 5.7 d. The situation shown in the figure reflects the completed processing of surface #1. The algorithm at this point examines the next Vector Map value for either the end of the Vector Map or the beginning of a new border point set. In the example case, the starting point for surface #2 at the 200' elevation level is found.
The algorithm proceeds along the initial vector segment of surface #2 until a crossover point is reached as shown in Figure 5.7 e. At this location the algorithm sets its internal flags and continues to step along the Vector Map representation of surface #2. The next crossover occurs at the location shown in Figure 5.7 f. When the second crossover condition is reached the algorithm logic once again determines a bracket condition by examining its internal flags. For this condition, however, the algorithm detects that a second border point set at the 200' level has bracketed the input coordinate. This decision is accomplished by the comparison of the current elevation level and the elevation of the submitted point. Since a second bracket at the same elevation level has occurred, the algorithm adjusts the unknown elevation value one elevation interval (100') downward.

The Look Up Algorithm continues the stepping procedure on the vector segments of surface #2 until a return to the starting point occurs as shown in Figure 5.7 g. At this location the algorithm determines the end of the Vector Map has been reached (all surfaces have been stepped) and the final elevation of the input coordinate is 100'. The map processor then returns the unknown elevation as part of the HANDSHAKE OUT function of the Elevation Value Output State (State V, Figure 5.5).

A rather subtle aspect of the Look Up Algorithm should
be noted at this point. A central concept involved in the look up process is that all border point sets (represented in vector format) enclose a specific map region. As discussed in Chapter 4 no specific elevation information can be assumed about the enclosed region. The critical concept is that at any elevation level the regions with elevations greater than or equal to the border elevation can be determined by identifying a region and subtracting out subset regions as depressions. In a physical sense this means as the map is traversed an exterior edge is always encountered first, followed by an interior edge (if present). Using this concept the map processor has only to determine whether an unknown point lies within the region defined by the vector representation of a border point set. Once all border point sets at an elevation level have been processed a point which lies within the region(s) bounded by an even number of border point sets will invariably be located outside the map region with elevation greater than or equal to the border elevation. Conversely a point which is enclosed by an odd number of border point sets will have an elevation greater than or equal to the border elevation level. The difficulty in creating a practical version of the Look Up Algorithm is centered on applying this method to the myriad of border point configurations present in actual terrain. Appendix C presents the development of an alternate example which is
designed to demonstrate the operation of the Vector Map concept on a more complex map representation.

The exact operation of the Look Up Algorithm is defined in more complete detail in the flowcharts of Figure 5.8 and the commented 6502 Assembly language listing of Appendix A.
## FLOWCHART LABELS

<table>
<thead>
<tr>
<th>LABEL</th>
<th>DEFINITION/USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNKY</td>
<td>Y (Row) Coordinate of Look Up Point</td>
</tr>
<tr>
<td>UNKX</td>
<td>X (Column) Coordinate of Look Up Point</td>
</tr>
<tr>
<td>UNKELV</td>
<td>Elevation of Look Up Point</td>
</tr>
<tr>
<td>BELV</td>
<td>Border Elevation of current border point set</td>
</tr>
<tr>
<td>E</td>
<td>EXACT Flag (1 if PPX=UNKX &amp; PPY=UNKY, otherwise 0)</td>
</tr>
<tr>
<td>C</td>
<td>CROSSING Flag (1 if PPX=UNKX, otherwise 0)</td>
</tr>
<tr>
<td>U</td>
<td>UP Flag (1 if UNKY &gt; PPY, otherwise 0)</td>
</tr>
<tr>
<td>PU</td>
<td>PRIOR UP Flag (Same as U Flag for direction of last crossing during current border point set)</td>
</tr>
<tr>
<td>PC</td>
<td>PRIOR CROSSING Flag (Same as C Flag for prior crossing during current border point set)</td>
</tr>
<tr>
<td>UF</td>
<td>UPDATE Flag (1 if an update of UNKELV occurred during current border point set)</td>
</tr>
<tr>
<td>MDP</td>
<td>MAP DATA POINTER (Index for map data starting at $A000)</td>
</tr>
<tr>
<td>PPX</td>
<td>PRESENT POSITION X (X (Column) coordinate of point being evaluated)</td>
</tr>
<tr>
<td>PPY</td>
<td>PRESENT POSITION Y (Y (Row) coordinate of point being evaluated)</td>
</tr>
<tr>
<td>CDX</td>
<td>CURRENT DELTA X (ΔX value of current vector)</td>
</tr>
<tr>
<td>CDY</td>
<td>CURRENT DELTA Y (ΔY value of current vector)</td>
</tr>
<tr>
<td>CDEL</td>
<td>CURRENT DELTA ELEVATION (Border point elevation interval)</td>
</tr>
</tbody>
</table>

Figure 5.8

Look Up Algorithm Program Flowchart
Power On Reset → "AUTO START"

A/S Successful ?

Y → To Operating System

N → "INITIALIZATION"

Initialization Successful ?

Y → G

Set Up Initial Map Data

Set Up Initial Border Set Data

"HANDSHAKE IN" (UNKX, UNKY)

Compare UNKX - PPX, UNKY - PPY

Set Flags

*Note; Not Shown, Similar to +ΔX, +ΔY

Compare Current UNKX - PPX, UNKY - PPY

Set Flags

Test Signs of ΔX, ΔY

* -ΔX, -ΔY

* +ΔX, -ΔY

* -ΔX, +ΔY

* +ΔX, +ΔY

Figure 5.8 (Continued)

Look Up Algorithm Program Flowchart
Figure 5.8 (Continued)

Look Up Algorithm Program Flowchart
Look Up Algorithm Program Flowchart

Figure 5.8 (Continued)
Figure 5.8 (Continued)

Look Up Algorithm Program Flowchart
5.4 System Interface Definition

The control of the bidirectional parallel bus of the test system can be described as a distinct task separate from the Vector Map processing by the Look Up Algorithm. While the interface related software modules are not particularly unique and are not critical to the development of the Vector Map concept, they do perform a critical function. The use of "off the shelf" hardware for the map processor system, while facilitating a minimum cost realization, serves to accentuate the complexity of the interface task. In this case, with only a portion of a single 6522 VIA available as an I/O port a unique system of interface protocol was developed. Figure 5.2 shows the 11 line bidirectional bus used in the test system. The control of this bus is a joint function of the simulated aircraft computer and the map processor comprising the test system. The necessary map processor software responsible for bus management is shown in the listing of Appendix A as the INITIALIZATION, HANDSHAKE IN and HANDSHAKE OUT modules. The assembly language listing of the corresponding software modules resident in the simulated aircraft computer are shown in Appendix B.

In developing the general concept of operation for the parallel bus, several guidelines were used. Specifically these guidelines were that the test system operate independently of the order of power application and that in
steady state operation, no bus contention is permissible. Using these guidelines the simulated aircraft computer was designated the bus controller while the map processor system was assigned a peripheral status consistent with the system design concept.

All transfers via the bus are initiated by the simulated aircraft computer. A single, active high, Bus Grant (BG) line indicates that the bus controller has placed the bus in an input mode. The peripheral is prohibited from enabling its output drivers until this line is asserted. In addition to BG (shown in Figure 5.9), two other handshake lines are used for bus control in the test system. These two lines, HS1 and HS2, are edge read signals used to handshake the data transfer across the bus. The source of HS2 is the simulated aircraft computer and the source of HS1 is the map processor. Data traverses the bus in parallel via eight bidirectional data lines labelled PBO-PB7. Bus format assigns the least significant bit of the transferred data word to PB0.

The initial task of the interface related software is to provide a reliable transition into steady state operation from a power on condition. The modules responsible for this transition are labelled INITIALIZATION and are shown in Appendix A and Appendix B. A timing diagram showing a typical initialization sequence is shown in Figure 5.10. As is shown in this figure, the critical
Figure 5.9
Test System Interface
Figure 5.10

Initialization Timing
aspect of the initialization sequence is the transfer of a specific data word ($2A) in a defined manner. For the map processor to complete initialization it is required to read $2A on the data bus for 48 consecutive reads. This equates to about .6 msec. The map processor is forced to continue reading the bus in 12.7 usec cycles until this is accomplished. The simulated aircraft computer takes control of the bus during its initialization. As bus controller it then places the data word $2A on the data bus for approximately 3 msec. When the map processor has read the necessary .6 msec of data =$2A, it signals via HS1 that it has completed initialization. Should no return signal be detected within 3 msec, the simulated aircraft computer declares an initialization error and returns to program control.

In steady state operation the task of the bus control software is to transfer the X and Y Vector Map coordinate into the map processor and to then return the elevation value associated with that coordinate to the simulated aircraft computer. Consistent with the finite state model of the map processor presented earlier, the transfer sequence X coordinate, Y coordinate and elevation value are always fixed. The transfer is initiated by the simulated aircraft computer when it transfers the X coordinate value. Using the straight forward edge controlled handshaking shown in Figure 5.11 the Y coordinate transfer begins
a. Coordinate Transfer (HANDSHAKE IN)

b. Elevation Transfer (HANDSHAKE OUT)

Figure 5.11

Data Transfer Timing
immediately upon completing the X coordinate transfer. The operation of the coordinate transfer is controlled by the software module HANDSHAKE IN (See Appendix A) and DATA TRANSFER (See Appendix B) in the map processor and simulated aircraft computer respectively. Upon complete coordinate transfer BG is asserted and bus control passes to the map processor.

The other half of the data transfer cycle is the transfer of the elevation value from the map processor. Also shown in Figure 5.11, this transfer is initiated by the map processor when it has completed the elevation determination. The same edge controlled handshaking is used for this transfer and bus control reverts to the simulated aircraft computer when the transfer is completed. The transfer of the elevation value completes the data transfer cycle and the next bus related operation is the transfer of a new unknown elevation coordinate. The software modules for the elevation transfer are labelled HANDSHAKE OUT (See Appendix A) and DATA TRANSFER (See Appendix B).
CHAPTER 6
SYSTEM TESTING

6.1 Description of System Testing

The feasibility of the Vector Map concept was established by an operational test of the map processor test system. In preparation for this testing a representative terrain section 1 Km by 2 Km was selected and digitized using a 100 meter digitizing grid. This process, which is described in Chapter 2, resulted in an integer array of size 11 x 21. This array was then processed on a small computer system simulating the airfield computer described in Chapter 3. The simulated ground station processing utilized the BASIC language implementation of the Bordering, Vector Encoding and Formatting Algorithms and produced a 107 byte Vector Map. As outlined in Chapter 4 this Vector Map terrain representation was then transferred to EPROM storage in order to simulate a non-volatile memory module.

The operational test of the map processor system described in Chapter 5 was conducted using the EPROM containing the Vector Map terrain representation. The test set up, shown in Figure 6.1, utilized the map processor test system configured as a peripheral to a simulated aircraft computer. Testing was initiated by the introduction of the EPROM Vector Map into the map processor test system. Power was then applied to both systems and the
Figure 6.1
Operational Test Functional Diagram
6502 machine language I/O program shown in Appendix B was loaded into the simulated aircraft computer. This was followed by the loading of a BASIC language control/display program into the simulated aircraft computer. This program whose listing is shown in Figure 6.2 was designed to simulate the control and display function of an actual aircraft computer. In operation it provided for the correct initialization of the bidirectional bus and the operator interface necessary to enter a map coordinate and display the resultant elevation value.

Once the simulated aircraft computer had been programmed, testing was initiated by running the control/display program. Proper bidirectional bus initialization was verified by the operator at this point as a prerequisite to further testing. The remainder of the operational test consisted of the operator insertion of a Vector Map coordinate using the simulated aircraft computer keyboard. This coordinate was then transferred to the map processor look up program using the bidirectional bus. The map processor then obtained the unknown elevation value and returned it to the simulated aircraft computer. The control/display program displayed the elevation value to the operator who then verified the value using the original elevation data array.

The operational testing of the map processor system established that for each location in the original data
0 REM:  "CONTROL/DISPLAY PROGRAM"
1 REM:  THIS PROGRAM PROVIDES AN OPERATOR INTERFACE TO THE SIMULATED
2 REM:  AIRCRAFT COMPUTER DURING MAP PROCESSOR TESTING. INITIALIZATION OF
3 REM:  THE BI-DIRECTIONAL BUS IS PERFORMED AND AN ERROR CONDITION IS
4 REM:  INDICATED FOR INCORRECT INITIALIZATION. THE OPERATOR IS PROMPTED
5 REM:  TO ENTER A VECTOR MAP (X,Y) COORDINATE AND IMPROPER ENTRIES ARE
6 REM:  FLAGGED. THE MAP COORDINATE IS PASSED TO THE MAP PROCESSOR BY AN
7 REM:  I/O SUBROUTINE. THE RESULTANT ELEVATION VALUE IS RECEIVED FROM THE
8 REM:  MAP PROCESSOR BY I/O SUBROUTINE. THE ELEVATION IS CONVERTED TO FEET
9 REM:  AND DISPLAYED.
10 REM:  
11 REM:  
12 POKE$:0 :REM: SAVE SPACE AT TOP OF MEMORY*********
13 POKE$2,64
14 POKE$5,0
15 POKE$6,64
16 SYS$(18432) :REM: CALL INITIALIZATION SUBROUTINE****
17 IF PEEK$(18503)$=255 THEN PRINT"INIT. ERROR":GOTO200
18 INPUT"ENTER HORIZ. COORD.";X :REM: PROMPT COORDINATE ENTRY*********
19 IF X>1270RX<0 THENPRINT"INPUT ERROR":GOTO120
20 INPUT"ENTER VERT. COORD.";Y
21 POKE$18454,X :REM: LOAD I/O COORDINATE REGISTER********
22 POKE$18585,Y
23 SYS$(18507) :REM: CALL I/O SUBROUTINE***************
24 Z=PEEK$(18506) :REM: GET ELV. FROM I/O REGISTER**********
25 IF Z=1 THEN Z=Z-1:A$=CHR$(32) :REM: CHECK FOR EXACT ELEVATION FLAG****
26 Z=Z*18 :REM: CONVERT TO FEET***************
27 PRINT X;Y;Z;A$ :REM: DISPLAY TO OPERATOR*****************
28 PRINT"PRESS 0 TO END, ANY OTHER KEY TO CONTINUE"
29 GET$;IF$=""THENGOTO187 :REM: CHECK FOR TEST TERMINATION********* 
30 IF B$=CHR$(64)THENGOTO200
31 GOTO 100
32 END

Figure 6.2

Control/Display Program (Manual Test Version)
array the map processor determined elevation matched the digitized source elevation data within the 100' resolution requirement. In order to provide a complete record of this performance a modified version of the BASIC control/display program was written to provide a printed output of the elevation data. In this program, whose listing appears in Figure 6.3, all points in the terrain elevation data array are sequentially submitted to the map processor test system for elevation determination. These results are then printed in the same 11 x 21 array format as the terrain elevation source data. Figure 6.4 shows the output of this program for the operational test of the map processor system. Figure 6.4 may be compared to the original elevation data array shown in Figure 6.5 on a point by point basis. This comparison indicates that for all locations the Vector Map derived elevation value is within 100' of the original terrain elevation source data.

As an additional aspect of system testing the look up time for the map processor was measured using the system clock present in the simulated aircraft computer operating system. This clock which was read by a subroutine call from a modified control display test program provided time measurements with an accuracy of 1/60 of a second. Using this method the worst case look up time for a Vector Map elevation was approximately 180 msec. The worst case time measurement was found to occur for locations with
Control/Display Program (Printout Version)

Figure 6.3
<table>
<thead>
<tr>
<th>Figure 6.4</th>
</tr>
</thead>
</table>

Map Processor Test System Elevation Values
<table>
<thead>
<tr>
<th>200</th>
<th>200</th>
<th>200</th>
<th>200</th>
<th>200</th>
<th>300</th>
<th>300</th>
<th>300</th>
<th>300</th>
<th>300</th>
<th>300</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>200</td>
<td>225</td>
<td>250</td>
<td>250</td>
<td>225</td>
<td>250</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>200</td>
<td>225</td>
<td>300</td>
<td>300</td>
<td>275</td>
<td>275</td>
<td>300</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>300</td>
</tr>
<tr>
<td>200</td>
<td>250</td>
<td>350</td>
<td>450</td>
<td>400</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>325</td>
<td>300</td>
</tr>
<tr>
<td>200</td>
<td>250</td>
<td>350</td>
<td>400</td>
<td>400</td>
<td>300</td>
<td>300</td>
<td>325</td>
<td>325</td>
<td>325</td>
<td>300</td>
</tr>
<tr>
<td>200</td>
<td>225</td>
<td>250</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>300</td>
<td>310</td>
<td>350</td>
<td>325</td>
<td>300</td>
</tr>
<tr>
<td>200</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>300</td>
</tr>
<tr>
<td>200</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>300</td>
<td>375</td>
<td>350</td>
<td>325</td>
<td>300</td>
</tr>
<tr>
<td>200</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>225</td>
<td>300</td>
</tr>
<tr>
<td>200</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>225</td>
<td>250</td>
<td>250</td>
<td>275</td>
<td>300</td>
<td>325</td>
<td>325</td>
</tr>
<tr>
<td>300</td>
<td>300</td>
<td>300</td>
<td>350</td>
<td>325</td>
<td>275</td>
<td>250</td>
<td>275</td>
<td>300</td>
<td>310</td>
<td>300</td>
</tr>
<tr>
<td>300</td>
<td>350</td>
<td>325</td>
<td>350</td>
<td>325</td>
<td>275</td>
<td>250</td>
<td>280</td>
<td>275</td>
<td>280</td>
<td>300</td>
</tr>
<tr>
<td>300</td>
<td>350</td>
<td>375</td>
<td>350</td>
<td>350</td>
<td>325</td>
<td>275</td>
<td>250</td>
<td>275</td>
<td>275</td>
<td>275</td>
</tr>
<tr>
<td>400</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>375</td>
<td>325</td>
<td>275</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>400</td>
<td>450</td>
<td>450</td>
<td>450</td>
<td>375</td>
<td>325</td>
<td>275</td>
<td>275</td>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>400</td>
<td>475</td>
<td>475</td>
<td>400</td>
<td>375</td>
<td>350</td>
<td>300</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>225</td>
</tr>
<tr>
<td>500</td>
<td>525</td>
<td>500</td>
<td>450</td>
<td>375</td>
<td>350</td>
<td>325</td>
<td>300</td>
<td>275</td>
<td>250</td>
<td>225</td>
</tr>
<tr>
<td>500</td>
<td>550</td>
<td>500</td>
<td>450</td>
<td>400</td>
<td>375</td>
<td>350</td>
<td>320</td>
<td>290</td>
<td>250</td>
<td>225</td>
</tr>
<tr>
<td>500</td>
<td>550</td>
<td>525</td>
<td>475</td>
<td>425</td>
<td>400</td>
<td>375</td>
<td>325</td>
<td>300</td>
<td>275</td>
<td>225</td>
</tr>
<tr>
<td>500</td>
<td>550</td>
<td>550</td>
<td>500</td>
<td>450</td>
<td>410</td>
<td>380</td>
<td>350</td>
<td>325</td>
<td>275</td>
<td>250</td>
</tr>
</tbody>
</table>

**Figure 6.5**

Digitized Terrain Elevation Source Data
elevations exceeding 500 feet. This result is consistent with system operation since, for the test region, the look up routine cannot conclusively determine the elevation of such points until the entire Vector Map representation has been processed.
6.2 Analysis of System Performance

The comparison of the original terrain data array (Figure 6.5) and the map processor elevation output (Figure 6.4) suggested in the previous section shows that for all locations the Vector Map derived value is within 100' of the source data. In order to interpret the significance of this measure, however, it must be remembered that the resolution of the Vector Map of a terrain section is determined by the operator selected border elevation interval at the time the map is created. In the case of the test data a 100' resolution was used in order to maintain consistency with the aircraft operational requirement discussed in Chapter 2. In the general case, however, the Vector Mapping concept as presented in this Thesis will produce elevation determinations with no more error than any selected border elevation interval. In addition to this concept it should be noted that the proper interpretation for a Vector Map derived elevation is that the actual elevation at that point is between the elevation value and the elevation value plus one border elevation interval. As an example a 200' Vector Map derived elevation indicates an actual elevation at that location greater than or equal to 200' but less than 300' (assuming a 100' border elevation interval was used for map synthesis). Since the Vector Map concept uses reduced resolution to minimize data storage requirements, the operator is free to trade off resolution
for area of coverage.

Having established that the only error inherent in the Vector Map process is bounded by the operator controlled resolution of the map, there remains only one additional error source in the determination of an actual terrain elevation. This error source is the quantization error produced by the digitizing process. Directly analogous to the frequency of sampling in analog to digital signal conversion this error is a direct function of the digitizing grid size and the ruggedness of the terrain in the mapped area. In as much as the density of spot elevations is controlled by current satellite photogrammetric limitations, the Vector Map concept cannot improve upon the accuracy of its source data.

An important aspect of the storage requirements for the test system is the impact of the selected elevation interval. In general it can be stated that the Vector Map concept trades storage requirements for resolution. For example in the case of the test terrain section a reduction in resolution requirements to 200' would have produced a Vector Map with a datum of 200' and a single elevation level at 400'. This would have required the encoding of only the border point sets shown in Figure 4.3 and would have resulted in a Vector Map requiring only 28 bytes of storage. This amount of storage equates to better than a 16:1 data compaction ratio gained by a 100' reduction in
resolution. It must be cautioned however that an exact estimate of Vector Map storage requirements is dependent upon the physical nature of the terrain as well as resolution requirements. The Vector Map process achieves maximum efficiency when the terrain model is composed of straight border point line segments which can be encoded with a minimum number of long vectors. Curvilinear segments therefore are inherently inefficient in Vector Mapping since they require a large number of short vector segments for modelling. For this reason the optimization of Vector Map storage requirements is trade off between resolution and the characteristics of the covered terrain.

The performance of the map processor test system can be quantitatively analyzed beyond a discussion of its error. In absolute terms the demonstration system started with a 1 Km x 2 Km terrain section having 231 integer (2 byte) values in its digitized array form. The test system used a total of 107 bytes for its Vector Map representation of this terrain section with the resolution of the model set at 100'. In absolute terms this represents better than a 4:1 reduction in storage requirements. In order to achieve this compaction, however, the map processing system utilized approximately 1000 bytes of program memory for the look up function.

The map processor test system may also be evaluated in terms of its conformance with the original design.
guidelines presented in Chapter 2. Such an evaluation is particularly appropriate since adherence to these guidelines insures a feasible system for actual in-aircraft use. In this case the map processor configured as a peripheral sensor would require no operator control during actual operation. This meets the human engineering requirement of transparent operation. The .5 ft$^3$ maximum physical size requirement is well beyond the size of the necessary hardware required by the test system. The support programming modification necessary for any aircraft computer system to use the map processor is also minimal. Specifically the Vector Map to geographic coordinate transformation and the necessary interface support software is adequately small to facilitate installation in a typical aircraft computer. The small program memory requirements of the map processor system is also well within the 2 K byte limitation for an implementation of the map processor as a software module to an existing aircraft computer flight program. The final design guideline requiring a 25 Km x 25 Km area of terrain coverage is also met by the system. In this case having shown a representative terrain section 1 Km x 2 Km can be stored to the necessary 100' resolution in only 107 bytes indicates that a 25 Km x 25 Km terrain section would require 33,000 bytes for similar storage. Since this is approximately equal to the largest non-volatile memory module size (32 K x 8) the guideline
is essentially satisfied. (It should be re-emphasized, however, that any evaluation of memory storage requirements is very dependent upon the characteristics of the covered terrain. This evaluation is based upon the assumption that the terrain in the hypothetical region is at least moderately complex in comparison to most of the earth's surface.)

6.3 Comparison of Vector Mapping and Direct Storage

A reasonable expectation of the Vector Map concept is that it provide a significant improvement over the direct storage of terrain elevation values. The direct storage concept as discussed early in this Thesis represents nothing more than the storage of the digitized elevation values on a point by point basis. This produces an elevation data array similar to the digitized values shown in Figure 2.3. Once entered into an aircraft computer these values can be accessed directly to obtain the elevation of any mapped location. This technique which requires a fixed number of points per unit area has not seen widespread practical implementation due to its extreme memory storage requirements.

The most significant advantage of direct storage with respect to Vector Mapping is in the area of simplicity. Conceptually straightforward, direct storage requires no preprocessing of data and an elevation value may be obtained by direct positional accessing of the data array.
Vector Mapping, however, requires significant preprocessing in order to create the terrain map representation. During use the Vector Mapping method requires additional special purpose software to extract the elevation value from the Vector Map. In regards to elevation data availability this means the direct storage technique requires only the time necessary to access the data array while Vector Mapping requires significant time (especially for large maps) to determine the elevation value.

The accuracy comparison of the two techniques is somewhat more complex. Direct storage can provide original data source accuracy while Vector Mapping is restricted to the operator defined resolution limitation. In as much as both techniques can trade resolution for area of coverage a direct comparison is not possible. Since direct storage is somewhat limited to byte increments in resolution due to conventional memory structure, however, it would appear that Vector Mapping is more flexible. This is due to the fact that Vector Mapping is done at discrete elevation levels with the resolution of each level set by a defined border elevation interval. This implies that the Vector Map elevations may have any defined interval or resolution.

The central advantage of the Vector Mapping technique over direct data storage is in the area of coverage versus necessary storage memory. For a practical example consider the 1 Km x 2 km area utilized in this Thesis. To store the
source data array for this area direct storage would require 462 bytes of memory. Even with reduced resolution the storage requirement for this technique would be 231 bytes (single byte precision). For this example the Vector Map representation is 107 bytes. When it is remembered that the terrain section is quite rugged for this example and that terrain irregularity severely limits Vector Mapping efficiency, this 2:1 storage advantage would seem quite conservative.

In summary it can be seen that the direct storage technique while offering extreme simplicity of implementation, provides no efficiency of storage. Vector Mapping which requires a more complex implementation is significantly more flexible and efficient in storage memory utilization. Since the Vector Map concept is terrain flexible, large areas of regular terrain may be stored in a minimal amount of memory. It is the flexibility and efficiency of the Vector Map concept that makes it much more feasible for the small flight computers present in the majority of current aircraft.
CHAPTER 7
SUMMARY AND CONCLUSION

7.1 Review of System Development

The development of the Vector Map approach to terrain elevation determination was based upon the need for a more efficient method of elevation information storage in current aircraft. Although most aircraft do in fact have the technological resources to obtain some form of terrain elevation information, there is currently no self contained, passive means of elevation determination available to aircraft flight management systems. Previous efforts to answer this need have attempted to use existing aircraft computer resources to store terrain elevation information on a point by point basis in order to provide in-flight direct access of elevation data. While such methods provide ready access of elevation information their inefficient use of available storage memory has restricted their use to impractically small areas of coverage.

The Vector Map concept is designed to be a practical means of providing elevation information to current aircraft flight management computer systems. As presented in this Thesis, the technique uniquely utilizes existing terrain elevation data bases and existing aircraft resources to produce an efficient terrain representation compatible with the small amount of memory in typical aircraft computer systems. Configurable as an imbedded
software module or a small peripheral system, the in-flight processing requirements of the Vector Map concept are also well within practical limits. In operation the Vector Map system offers a reasonable approach to terrain elevation determination.

In order to establish the validity of the Vector Map concept and to provide a coherent development for a test/demonstration system, this Thesis initially reviewed available terrain elevation source data and the operational requirements of a feasible system. By the careful definition of the typical characteristics of various in-aircraft and ground based resources, the practical boundaries for system development were established. A general solution within the development guidelines was then presented. The presentation of the general solution explained the various necessary elements of a practical Vector Map implementation.

The central aspect of this Thesis is the realization of a microprocessor controlled Vector Map system. This system was developed for the purpose of demonstrating the feasibility of the Vector Map concept as described in the general solution. Designed to demonstrate all aspects of a working system, the test system development was divided into the separate areas of ground based operations and in-flight operations. The ground based operations are those steps necessary to convert elevation data base information
into its Vector Map configuration. In this Thesis the preprocessing of data was described in detail by the presentation of various pre-flight algorithms used to develop the Vector Map concept. These algorithms were presented as a continuation to the preprocessing requirements. Centrally concerned with the extraction of an elevation value from the Vector Map terrain representation, the in-flight aspects were presented by the realization of the microprocessor based map processor system.

The performance of the map processor test system was demonstrated by operational testing. Using the Vector Map representation of the hypothetical terrain section, proper operation of the test system was verified for all mapped locations. In addition, operational testing using a more complex terrain representation was performed as described in Appendix C. Using the results of this testing, the overall feasibility of the Vector Map concept was then established by re-examining and comparing the original development guidelines with the relevant aspects of the test system. As a final analytical element, the Vector Map concept was critically evaluated in regards to its potential improvement over the direct storage method of elevation determination. The results of the test system performance analysis indicated the Vector Map concept to be a feasible and practical solution to the problem of ground elevation determination in contemporary aircraft.
7.2 Suggestions For Further Study

The development of the Vector Map concept and the demonstration system as discussed in this Thesis can be regarded as a near minimal realization. While such a small scale development does facilitate a clear presentation of the critical aspects of the concept, it does however leave significant room for future development. One significant area open, for further development is the implementation of the concept on a much larger scale. In such development, the researcher would use actual map agency data bases and test the concept using improved versions of the necessary algorithms run on a large computer system. This would conclusively establish system feasibility by demonstrating correct operation over large, actual terrain sections. The use of large terrain sections is necessary to adequately exercise the critical algorithms over the infinite variety of actual terrain structures. Such testing would also be necessarily mandatory prior to any actual in-aircraft use of the concept.

Perhaps the most significant opportunity for the future development of the Vector Map concept is in areas other than avionic applications. Of the potential alternate applications one of the most appropriate would be in the area of image processing. In image processing there is a great need for improved efficiency in the storage of digital picture data. In this case the volume of
information acquired from typical image sensors is extremely large. For non-real time processing needs and archival storage, the volume of image data can become overwhelming. The application of the Vector Map concept to this problem can be described by a typical example. Consider a conventional 512 x 512 pixel representation of a black square on a white background. In conventional format this image would require over 250,000 bytes of memory for archival storage. The direct application of the Vector Map concept to such an image would produce a Vector Map of about 10 bytes. The Vector Map could be stored and later accessed by the Look Up Algorithm presented in this Thesis for a complete reconstruction of the original picture. While this example can be regarded as a special case, the central point is that since image data is essentially an integer array and most images have a significant amount of gray level redundancy, the Vector Map concept should provide a significant improvement in digital picture storage efficiency.
LIST OF REFERENCES


APPENDIX A

******* AUTO-START PORTION OF O/S *******
FFFC 22 FD ;6502 RESET VECTOR

FD22 LDX#$FF
FD24 SEI
FD25 TXS
FD26 CLD
FD27 JSR$FD3F ;TO COMPARISON SUBROUTINE

FD2A BNE$FD2F ;BRANCH TO O/S IF AUTO-START BAD
FD2C JMP($A000) ;JMP TO MAP PROCESSOR ($4B00)

FD3F LDX#$05 ;COMPARISON SUBROUTINE
FD41 LDA$FD4C,X
FD44 CMP$A003,X
FD47 BNE$FD4C
FD49 DEX
FD4A BNE$FD41
FD4C RTS

FD4D 41 30 C3 C2 CD ;COMPARISON TABLE

**** ***** MAP PROCESSOR LOOK UP PROGRAM ***** ****

;***MAP SET UP***
4800 LDA$A009
4803 STA$1202 ;SET UNKELV=MAP DATUM
4806 LDA$A00A
4809 STA$1203 ;SET CURRENT BORDER ELEV.(BELV)
480C SEC
480D SBC$1202 ;SET BORDER ELEV.INTERVAL(CDEL)
4810 STA$1211
4813 LDA#$00
4815 LDX#$09
4817 DEX
4818 STA$1204,X ;INITIALIZE PROGRAM FLAGS
481B BNE$4817
481D LDX#$0B
481F LDA$A000,X
4822 STA$120D ;SET START Y COORD (PPY)
4825 INX
4826 LDA$A000,X
4829 STA$120E ;SET START X COORD (PPX)
482C INX
482D STX$120C ;SAVE MAP DATA POINTER (MDP)
4830 JMP$4834 ;***MAP SET UP COMPLETE***
4833 NOP

142
4834  JMP$4B30  ;*** GET UNKX,UNKY ***
4848  LDX$120E  ;*** BORDER DIRECTOR***
484B  LDA$120D
484E  JSR$4F5D  ;CHECK FOR STARTING PT.MATCH
4851  LDA$1205  ;READ CROSSING FLAG(C)
4854  BEQ$4860  ;BRANCH FOR NO CROSSING
4856  STA$1208  ;RESET PRIOR CROSSING FLAG(PC)
4859  LDA$1206
485C  STA$1207  ;PRIOR UP(PU) FLAG=UP FLAG(U)
485F  NOP
4860  LDX$120C  ;*** VECTOR DIRECTOR ***
4863  LDA$120D  ;CHECK FOR STARTING PT. MATCH
4866  CMP$1200
4869  BNE$4878
486B  LDA$120E
486E  CMP$1201
4871  BNE$4878
4873  LDY#$80  ;PPX=UNKX,PPY=UNKY,SET EXACT FLAG(E)
4875  STY$1204
4878  LDA$A000,X  ;GET \( \Delta X \) (CDX)
487B  BMI$488D  ;TO \( - \Delta X \)
487D  STA$120F  ;SET CURRENT DELTA X(CDX)
4880  INX
4881  LDA$A000,X  ;GET \( \Delta Y \)(CDY)
4884  BMI$489F  ;CONVERT \( - \Delta Y \)
4886  STA$1210  ;SET CURRENT DELTA Y(CDY)
4889  INX
488A  JMP$48C0  ;TO \( + \Delta X, + \Delta Y \)
488D  AND#$7F  ;\(- \Delta X TO + \Delta X\)
488F  STA$120F  ;SET CURRENT DELTA X(CDX)
4892  INX
4893  LDA$A000,X  ;GET \( \Delta Y \)
4896  BMI$48A8  ;CONVERT \(- \Delta Y \)
4898  STA$1210  ;SET CURRENT DELTA Y(CDY)
489B  INX
489C  JMP$4952  ;TO \(- \Delta X, + \Delta Y \)
489F  AND#$7F  ;\(- \Delta Y = + \Delta Y \)
48A1  STA$1210  ;SET CURRENT DELTA Y(CDY)
48A4  INX
48A5  JMP$490E  ;TO \(+ \Delta X, - \Delta Y \)
48A8  AND#$7F  ;\(- \Delta Y = + \Delta Y \)
48AA  STA$1210  ;SET CURRENT DELTA Y(CDY)
48AD  INX
48AE  JMP$4980  ;TO \(+ \Delta X, - \Delta Y \)
48C0  STX$120C  ;*** END DIRECTOR ***
48C3  LDA#$00
48C5  STA$1DF1  ;CLEAR OLD DIVISION RESULT
48C8  LDA$120F  ;GET CDX
48CB  BEQ$48FA  ;CDX=0?
48CD  STA$1DF0  ;SET DIVISOR = CDX
48D0  LDA$1210
***SPECIAL CASE COMPLETE TO EV.***

***+~X,-6Y***

***VECTOR COMPLETE TO EVALUATOR***

***+~X,-6Y***

***SPECIAL CASE +X=0***

***VECTOR COMPLETE TO EVALUATOR***

***+~X,+~Y***

STEP INCREMENT Y = PPY + QUOTIENT

DIVISION RESULT

STEP PPX +1

SET NEW PPX

SET NEW PPY

TO COMPARISON SUBROUTINE

STEP UNTIL CDX=0

***VECTOR COMPLETE TO EVALUATOR***

***+~X,+~Y***

SPECIAL CASE ~X=0***

GET PPY

GET PPY

TO COMPARISON SUBROUTINE

STEP UNTIL CDY=0

***SPECIAL CASE COMPLETE TO EV.***

***+~X,-~Y***

***VECTOR COMPLETE TO EVALUATOR***

***+~X,-~Y***

SPECIAL CASE ~X=0***

***SPECIAL CASE COMPLETE TO EV.***

***-~X, +~Y***
LDA$1210
STA$1DF1
JSR$4F37
LDX$120E
LDA$120D
DEX
CLC
ADC$1DF1
STA$120D
JSR$4F5D
DEC$120F
BNE$4F64
LDA$120C
LDA$120F
STA$1DF0
LDA$1210
STA$1DF1
JSR$4F37
LDX$120E
LDA$120D
DEX
SEC
SBC$1DF1
STA$120D
JSR$4F5D
DEC$120F
BNE$4F64
LDA$120C
LDA$120F
STA$1DF0
BNE$4F64
JMP$4A01
***VECTOR COMPLETE; TO EVALUATOR***
*** - △ X, - △ Y***

LDA$1210
STA$1DF1
JSR$4F37
LDX$120E
LDA$120D
DEX
CLC
ADD$1DF1
STA$120D
JSR$4F5D
DEC$120F
BNE$4F64
LDA$120C
LDA$120F
STA$1DF0
LDA$1210
STA$1DF1
JSR$4F37
LDX$120E
LDA$120D
DEX
SEC
SBC$1DF1
STA$120D
JSR$4F5D
DEC$120F
BNE$4F64
LDA$120C
LDA$120F
STA$1DF0
BNE$4F64
JMP$4A01
***VECTOR COMPLETE; TO EVALUATOR***

LDA#$FF

***EVALUATOR****
BIT$1204
BEQ$49BE
; CHECK EXACT FLAG (E) = 1
JSR$4F7A
; UPDATE UNKELV AS EXACT SOL.

BNE$4A01
JMP$4A01
; TO "END"

BNE$4A01
JMP$4A01
; TO "END", NO ENCLOSURE

BNE$4A01
JMP$4A01
; TO "END", NO ENCLOSURE
49DD  LDA$1203 ;***ENCLOSURE***
49EB  CMP$1202 ;CHECK FOR LAST ENCL. AT SAME ELV.
49E3  BEQ$49EB
49E5  JSR$4FA2 ;ENCLOSURE ELEVATION ADJUST
49E8  JMP$4A01 ;TO "END"
49EB  JMP$4A50 ;DEPRESSION ENCL., TO "SUBTRACT"

49F8  LDA$1206 ;PRIOR UP FLAG (PU) = UP FLAG (U)
49FB  STA$1207 ;SET PRIOR CROSSOVER FLAG (PC)
4A01  LDA#$00 ;**END**
4A03  STA$1204 ;CLEAR EXACT Flag (E)
4A06  STA$1205 ;CLEAR CROSS Flag (C)
4A09  STA$1206 ;CLEAR UP Flag (U)
4A0C  LDX$120C ;GET MAP DATA POINTER (MDP)
4A0F  LDA$A000,X ;GET NEXT MAP VALUE
4A12  BPL$4A1A ;CONVERT NEXT MAP VALUE TO +
4A14  TAY
4A15  LDA#$7F
4A17  AND$A000,X
4A1A  SEC
4A1B  SBC$1203 ;SUBTRACT CURRENT BORDER ELEV (BELV)
4A1E  BMI$4A2A ;IF NEXT VALUE > CURRENT BORDER ELEV
4A20  BEQ$4A2D ;IF NEXT VALUE = CURRENT BORDER ELEV
4A22  CMP$1211
4A25  BEQ$4A2D ;IF NEW BORDER ELEV = BORDER ELEV
4A27  JMP$4B70 ;"HAND SHAKE OUT" MAP COMPLETE
4A2A  JMP$4860 ;NEW CDX GO TO START OF "VECTOR DIR"
4A2D  LDA#$00 ;CLEAR PRIOR CROSSING FLAG (PC)
4A2F  STA$1208
4A32  LDA$A000,X ;SET UP FOR NEW BORDER PT ELEV
4A35  STA$1203 ;SET NEW BORDER PT ELEV (BELV)
4A38  INX
4A39  LDA$A000,X ;SET START Y COORD (PPY)
4A3C  STA$120D
4A3F  INX
4A40  LDA$A000,X
4A43  STA$120E ;SET START X COORD (PPX)
4A46  INX
4A47  STX$120C ;SAVE MAP DATA POINTER (MDP)
4A4A  JMP$4A6B
4A4D  NOP
4A4E  NOP
4A4F  NOP
4A50  LDA#$01 ;**SUBTRACT**
4A52  BIT$1202 ;TEST FOR EXACT BIT SET
4A55  BNE$4A66 ;BYPASS IF EXACT
4A57  BIT$1209 ;TEST (UF) FOR UPDATE AT THIS BORDER PT SET
4A5A  BNE$4A66
4A5C  SEC
4A5D  LDA$1202
4A60  SBC$1211
4A63  STA$1202
4A66  JMP$4A01 ;***EXACT SOL. NO SUBTRACT TO "END"
4A69  NOP  ;***ADDITION TO "END"***
4A6A  NOP  ;CLEAR SAME BORDER FLAG
4A6B  LDA#$00
4A6D  STA$1209
4A70  LDA$1202
4A73  CLC
4A74  ADC$1211  ;BELV + CDEL
4A77  AND#$FE  ;CLEAR EXACT BIT
4A79  CMP$1203  ;COMPARE TO UNK EL
4A7C  BMI$4A81  ;IF < ,MAP COMPLETE
4A7E  JMP$4848  ;TO START OF "BORDER DIRECTOR"
4A81  JMP$4B70  ;TO "HANDSHAKE OUT"
4B00  LDA#$00  ;***START MAP PROCESSOR PROGRAM***
4B02  STA$9112  ;***START UP HANDSHAKE MODULE***
4B05  STA$9113
4B08  LDY#$00
4B0A  LDA#$00
4B0C  STA$9110
4B0F  LDA$9110
4B12  CMP#$2A
4B14  BEQ$4B18
4B16  LDY#$00
4B18  INY
4B19  CPY#$14
4B1B  BNE$4B1F
4B1D  LDA#$E0
4B1F  STA$911C
4B22  LDA$9110
4B25  JMP$4800  ;***TO "MAP SET UP"***
4B30  LDA$9110  ;***HANDSHAKE IN, GET UNKX, UNKY***
4B33  LDA#$10
4B35  BIT$911D
4B38  BEQ$4B35
4B3A  LDA$9110
4B3D  STA$1201
4B40  LDA#$C0
4B42  STA$911C
4B45  LDA#$E0
4B47  STA$911C
4B4A  LDA#$10
4B4C  BIT$911D
4B4F  BEQ$4B4C
4B51  LDA$9110
4B54  STA$1200
4B57  LDA#$C0
4B59  STA$911C
4B5C  LDA#$E0
4B5E  STA$911C
4B61  JMP$4848  ;***HANDSHAKE IN COMPLETE***
4B70  LDA$9115  ;***HANDSHAKE OUT, SEND UNKELV***
AND#$04
BEQ$4B70
LDA#$FF
STA$9112
LDA$1202
STA$9110
LDA#$SC0
STA$911C
LDA#$E0
STA$9112
LDA$911F
AND#$04
STA$911C
LDA#$10
BIT$911D
BEQ$4B8E
LDA#$OO
STA$9112
LDA$911F
AND#$04
BNE$4B98
JMP$4800
STA$lDF3
STX$lDF4
LDA#$OO
LDX#$08
ASL$lDF1
ROL
CMP$lDF0
BCC$4F50
SBC$lDF0
INC$lDF1
DEX
BNE$4F41
STA$lDF2
LDY#$01
STY$1204
STA$lDF4
LDA$1203
JMP$4800
***COMPLETE HANDSHAKE OUT***
***TO MAP SET UP***
***BEGIN DIVISION SUBROUTINE***
STA$1DF3
STX$1DF4
LDA#$S00
LDX#$S08
ASL$lDF1
ROL
CMP$lDF0
BCC$4F50
SBC$lDF0
INC$lDF1
DEX
BNE$4F41
STA$lDF2
LDY#$1201
STY$1204
STA$lDF4
LDA$1203
RTS
***END DIVISION SUBROUTINE***
TAY
***COMPARISON SUBROUTINE***
TXA
***RETURN***
CMP$1201
;CHECK PPX-UNKX
BNE$4F79
JSR$4F8A
;CROSS OVER OR MATCH TO "X COORD MATCH"
TAX
TYA
PPX=UNKX, PPY<UNKY, SET UP FLAG
;PPX=UNKX, PPY<UNKY, SET UP FLAG
RTS
;****RETURN***
JSR$4F96
;EXACT MATCH, SET EXACT FLAG
LDA$1204
RTS
;****RETURN****
STA$1DF4
;****EXACT ELEV UPDATE****
LDA$1203
LDY#$01
STY$1204
STA$1DF4
LDA$1203
JSR$4F79
;****RETURN****
CLC
STA$1DF4
CLR
LDA$1203
;****EXACT ELEV UPDATE****
LDA$1203
4F80  CLC
4F81  ADC#$01
4F83  STA$1202
4F86  LDA$1DF4
4F89  RTS
4F8A  STA$1DF4
4F8D  LDA#$01
4F8F  STA$1205
4F92  LDA$1DF4
4F95  RTS
4F96  STA$1DF4
4F99  LDA#$01
4F9B  STA$1206
4F9E  LDA$1DF4
4FA1  RTS
4FA2  LDY$1202
4FA5  STA$1202
4FA8  TYA
4FA9  AND#$01
4FAB  CLC
4FAC  ADC$1202
4FAF  STA$1202
4FB2  LDR#$01
4FB4  STA$1209
4FB7  LDA$1202
4FBA  RTS

; ***EXACT UPDATE RETURN***
; *** X COORD MATCH***
; SET CROSSING FLAG
; ***X COORD MATCH RETURN***
; ***SET UP FLAG***
; ***UP FLAG SET RETURN***
; ***ENCLOSURE ELEV. ADJUST***
; UNKEL=BORDER ELEV
; RETAIN EXACT BIT STATUS IN UNKEL
;SET CURRENT BORDER CHANGE FLAG

**** ELEV. ADJUST RETURN ****
APPENDIX B

SIMULATED AIRCRAFT COMPUTER I/O LISTING

***INITIALIZATION***

4800 LDA$9113
4803 ORA#$04 ;SET PA2 (BG) TO OUTPUT
4805 STA$9113
4808 LDA$911F
480B AND#$FB
480D STA$911F ;SET BG LOW
4810 LDA#$FF
4812 STA$9112 ;SET DATA BUS TO OUTPUT
4815 LDA$911C
4818 ORA#$FO
481A STA$911C
481D LDA$9110
4820 LDY#$00 ;INITIALIZE TIME COUNTER
4822 LDA#$2A
4824 STA$9110 ;SET DATA BUS = #$2A
4827 INY
4828 LDA#$10 ;TEST FOR HS1 EDGE
482A BIT$911D
482D BNE$4836
482F CPY#$EF
4831 BNE$4827 ;NO EDGE, CHECK 6MSEC TIMER
4833 JMP$483C
4836 LDX#$00
4838 STX$4847
483B RTS ;SUCCESSFUL INITIALIZATION RETURN
483C LDX#$00
483E STX$9110
4841 LDX#$FF
4843 STX$4847
4846 RTS ;INITIALIZATION ERROR RETURN
4848 ;X COORDINATE
4849 ;Y COORDINATE
484A ;ELEVATION VALUE

***DATA TRANSFER (I/O SUBROUTINES)***

484B LDA$9110
484E LDA$4848 ;SEND X COORDINATE
4851 STA$9110
4854 LDA$911C
4857 AND#$DF
4859 STA$911C
485C LDA$911C
485F ORA#$FO
4861 STA$911C
4864 LDA#$10
4866 BIT$911D ;WAIT FOR HS1 EDGE
4869 BEQ$4866

150
LDA$4849 ;SEND Y COORDINATE
STA$9110
LDA$911C
AND#$DF
STA$911C
LDA$911C
ORA#$FO
STA$911C
LDA#$10
BIT$911D ;WAIT FOR HS1 EDGE
BEQ$4883
LDA#$00
STA$9112 ;SET DATA BUS TO INPUT
LDA$9110
LDA$911F
ORA#$04
STA$911F ;SET BG HIGH
LDA$911D
AND#$10
BNE$48A2 ;WAIT FOR HS1 EDGE, ELV. READY
JMP$4898
LDA$9110 ;GET ELEVATION VALUE
STA$484A
LDA$911C
AND#$DF
STA$911C
LDA$911C
ORA#$FO
LDA#$FF
STA$9112 ;SET DATA BUS TO OUTPUT
STA$911F
AND#$FB
STA$911F
RTS
APPENDIX C

AN ALTERNATE VECTOR MAP EXAMPLE

The determination of terrain elevation using the Vector Map concept involves the precise interaction of the Bordering, Vector Encoding and Look Up Algorithms. During the presentation shown in Chapters 4 and 5 a typical example was used to demonstrate the basic operation of these algorithms. Although all critical concepts were covered during this presentation, several aspects of the Vector Map concept can be more clearly demonstrated using an alternate example. In the example developed in this Appendix a significantly different and more complex exercise of the map concept is created by the application of a different border elevation interval to the digitized terrain model presented in Figure 2.3.

The output data arrays of the Bordering Algorithm can be changed by the selection of a border elevation interval of 120'. When this interval is applied to the original DTM data array, border point data arrays are produced at 320' and 440'elevation levels. The output of the Bordering Algorithm at these elevation levels is shown in the upper half of Figures C1 and C2. If the 320' border point array is examined, an unusual line of border points is apparent in the upper right hand section of the array. Another unique aspect appearing at the 440' level is the presence of an isolated point. Upon initial examination it would
Figure C1

320' Border Elevation Processing
Figure C2

440' Border Elevation Processing
appear that the border points in each case do not represent an enclosed region and would therefore cause significant processing problems for the Look Up Algorithm.

If both the border arrays are submitted to the Vector Encoding Algorithm the output shown in the lower half of Figures C1 and C2 is produced. The movement of the Vector Encoding Algorithm has been defined by the encoding sequence numbers shown for each border point set. Processing starts and ends on the border point labelled 2. What this discloses is that the border point segments shown in the upper right hand section of Figure C1 are in fact part of a closed surface. Such segments may be identified in terms of their physical significance as the Vector Map representation of a narrow ridgeline. The reason why such ridgeline points do not appear to be part of a closed surface is that they contain two pieces of information. In this case both edges of the ridgeline are encoded into single border points.

In general terms it can be shown that a border point can contain as much as four pieces of information as is indicated in the four variable OR section of the border point Boolean definition given on page 37. The most common border point is one which forms the edge of a large region and contains only a single piece of information. Ridgeline points contain two pieces of information and are less
common. A point at the end of a ridgeline contains three pieces of information. Such a point is shown in the upper upper right hand corner of Figure C1. The most complex point is the isolated point shown in the central region of of Figure C2. This single point contains four pieces of information. The Vector Encoding Algorithm has been designed to process all four types of border points into the vectors defining a completely enclosed map region. This can be verified by examining the stepping sequence shown in in Figures C1 and C2. Notice that in Figure C2 the isolated point has been correctly encoded as an entire enclosed map region consisting of a single point.

The resultant Vector Map representation of the border point data arrays is shown in Figure C3. In this example the reduction of the resolution of the Vector Map (from 100' to 120') has produced a representation of the terrain which requires 97 bytes of memory as opposed to the 107 bytes required by the original example. This shows how reduced resolution may be used to minimize storage requirements. It should be recalled that this flexibility forms a critical advantage of the Vector Map concept over alternate methods of terrain information storage.

The operation of the map processor Look Up Algorithm on the 120' Vector Map representation is consistent with the description presented for the text example using a 100' border elevation interval Vector Map. When tested in the
Figure C3
Vector Map For 120' Elevation Interval
map processor demonstration system the resultant output for each map location was verified correct. The resultant data is shown in the original array format in Figure C4.

The operation of the Look Up Algorithm in the more complex 120' elevation interval Vector Map can be clarified by a typical example. If a point with X coordinate = 9 and Y coordinate = 10 is submitted to the map processor, a total of six crossovers occur during the processing of the 320' border point set containing the point. The point with unknown elevation and the six crossovers are shown in Figure C5. The map processor will begin the elevation look up process in the upper left hand corner. As the border point set is stepped in a clockwise manner the first X coordinate match (crossover) occurs at point a. Since this is the first crossover no bracket is possible. The program flags note a crossover has occurred and that at the time of crossover the point in question was below the Y coordinate being processed. As the program steps to the second crossover (point b) the algorithm determines the point has been bracketed by two successive crossovers; one above and one below the point in question. At this time the elevation of the submitted point is updated to 320'. As the stepping process continues, subsequent crossover points c, d, e and f do not provide a change in elevation since in all cases the point in question lies below the position of the algorithm. Without successive crossovers above and
Figure C4
Map Processor Elevation Output
Figure C5

Look Up Algorithm Example
below the point being evaluated no further elevation changes occur and the final elevation of the point remains at 320'.

The performance of the map processor test system was verified using the Vector Map representation shown in Figure C3. As indicated in the output data array shown in Figure C4, the elevation of each map location was correctly determined. This testing adequately demonstrates the versatility of the Vector Map concept on significantly complex terrain representations. It may also be verified from the example presented in this Appendix that the resolution of a map representation is inversely related to its storage requirements.