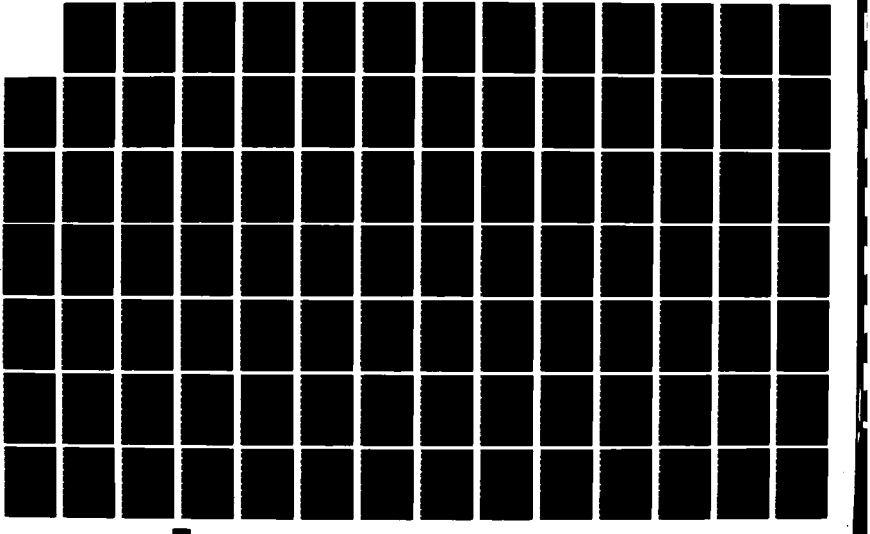
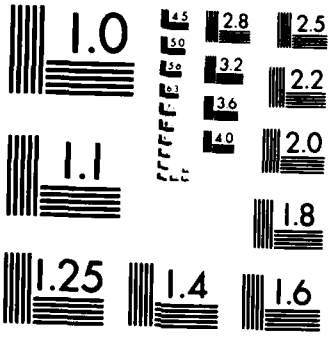
AD-A172 932 ANALYSIS OF TRANSMITTER PRODUCED INTERMODULATION INTERFERENCE IN COLOCATE (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI T J ZUZACK

1/4

UNCLASSIFIED JUN 86 AFIT/GE/ENG/86J-3 F/G 20/14 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ANALYSIS OF TRANSMITTER PRODUCED INTERMODULATION INTERFERENCE IN COLOCATED VHF SITES USING A MICROCOMPUTER

THESIS

Thomas J. Zuzack
Captain, USAF

AFIT/GE/ENG/86J-3

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

ANALYSIS OF TRANSMITTER PRODUCED INTERMODULATION INTERFERENCE IN COLOCATED VHF SITES USING A MICROCOMPUTER

THESIS

Thomas J. Zuzack
Captain, USAF

AFIT/GE/ENG/86J-3

A

Approved for public release; distribution unlimited

AFIT/GE/ENG/86J-3

ANALYSIS OF TRANSMITTER PRODUCED INTERMODULATION INTERFERENCE IN COLOCATED VHF SITES USING A MICROCOMPUTER

THESIS

Presented to the Faculty of the School of Engineering

Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Thomas J. Zuzack, Captain, USAF

B.S.E.E.

June 1986

PREFACE

As the proliferation of radio frequency communications equipment increases, so does the problem of electromagnetic compatibility (EMC). A particularly acute EMC problem is created when VHF equipment is colocated. The EMC engineer analyzing such a situation is faced with a complex and almost overwhelming task. The purpose of this thesis then is to assist the EMC engineer in performing one part of a colocated site analysis, the analysis of transmitter produced intermodulation interference. To do so, I have implemented an algorithm for performing an analysis of transmitter produced intermodulation interference as a Turbo Pascal program that will run on the Zenith Z-100 microcomputer under the MS-DOS operating system.

I would like to take this opportunity to thank Major Glenn E. Prescott and Captain David A. King for their constructive criticism and hours of proof reading my thesis. I would also like to thank Captain Dennis K. Greer for his expert advice on Turbo Pascal. He saved me countless hours while I was learning Turbo. But most of all I would like to thank my wife, Mia, for her loving support, gentle encouragement, and quiet suffering. I could not have finished this thesis without her.

Thomas J. Zuzack

TABLE OF CONTENTS

	Page
Preface	ii
List Of Figures	v
Abstract	vi
I. Introduction	I-1
Background	I-1
The Problem	I-2
The Solution	I-3
Overview Of The Remaining Chapters	I-3
II. The Problem	II-1
The Cause	II-1
The Mechanism	II-2
What Can Be Done	II-4
The Problem As Attacked By This Thesis	II-5
III. Results Of The Literature Search	III-1
The Intermodulation Products Of Interest	III-1
Factors Affecting Intermodulation Product Strength	III-5
Other Factors Requiring Consideration	III-13
IV. The Analysis Algorithm	IV-1
Introduction	IV-1
Collect Data	IV-3
Antenna Data	IV-4
Frequency Data	IV-7
Organize Data	IV-10
Antenna Data Organization	IV-12
Frequency Data Organization	IV-13
Select A Victim Frequency For Analysis	IV-13
Calculate The Intermodulation Product Frequencies	IV-14
The A+B Intermodulation Products	IV-15
The A-B Intermodulation Products	IV-17
The 2A-B Intermodulation Products	IV-19
The A+B-C Intermodulation Products	IV-23
The Case With The Victim Less Than All Culprits	IV-25

Page

The Case With The Victim Greater Than One Culprit IV-26
The Case With The Victim Greater Than Two Culprits IV-27
The Case With The Victim Greater Than All Culprits IV-29
The 3A-2B Intermodulation Products . . IV-31
Test The Intermodulation Product IV-32
Victim Receiver Thermal Noise IV-33
Antenna Separation Distances IV-34
Free Space Loss IV-35
Culprit Power IV-36
Selectivity IV-37
Transmit Intermodulation Power IV-38
Received Intermodulation Product Power IV-40
The Intermodulation Product Testing Algorithm IV-41
Review The Results And Update The Data . . IV-44
Repeat The Analysis Procedure IV-46

V. Comments And Recommendations V-1

Appendix A: User's Manual A-1

Appendix B: Source Code Listing B-1

Appendix C: Bibliography C-1

Appendix D: Vita D-1

Appendix E: 5.25" Floppy Disk Containing Source Code And Executable Files E-1

Report Documentation Page (DD Form 1473)

LIST OF FIGURES

Figure		Page
III-1.	A Power Amplifier And Its Nonlinear Transfer Function	III-1
III-2.	An Example Colocated Site	III-5
III-3.	Conservative Estimates Of Transmitter Output Selectivity	III-10
IV-1.	An Antenna Layout Prepared For Program Input	IV-5
IV-2.	An Antenna Table Example	IV-13
IV-3.	A Frequency Record Example	IV-14
IV-4.	Pseudo Code For The Calculation Of A+B Intermodulation Products	IV-17
IV-5.	Pseudo Code For The Calculation Of A-B Intermodulation Products	IV-19
IV-6.	Pseudo Code For The Calculation Of 2A-B Intermodulation Products	IV-24
IV-7.	Pseudo Code For The Calculation Of A+B-C Intermodulation Products	IV-30
IV-8.	Pseudo Code For The Calculation Of 3A-2B Intermodulation Products	IV-31
IV-9.	Pseudo Code For Testing A Two Signal Intermodulation Product	IV-42
IV-10.	The Format Of An Analysis Output	IV-45
A-1.	TIMAP's MAIN MENU	A-16
A-2.	TIMAP's ADD MENU	A-17
A-3.	An Antenna Layout Prepared For Program Input	A-20
A-4.	TIMAP's DISPLAY MENU	A-27
A-5.	TIMAP'S EDIT MENU	A-29
A-6.	TIMAP'S OPTIONS MENU	A-36

ABSTRACT

This thesis presents the theory behind the generation of intermodulation products in the final output power amplifier of a VHF transmitter and the calculations necessary to determine the power in a transmitter produced intermodulation product at the front end of a receiver. The results are then applied to the analysis of transmitter produced intermodulation interference in colocated VHF sites. An algorithm is developed for accomplishing this analysis and is implemented in the form of a Turbo Pascal program named TIMAP (Transmitter Intermodulation Analysis Program) which runs on the Zenith Z-100 microcomputer under the MS-DOS operating system. TIMAP provides a systematic and automated analysis tool for electromagnetic compatibility engineers in the field.

I. INTRODUCTION

Electromagnetic compatibility (EMC) is an issue of vital importance to military communicators. EMC is the ability of equipment or systems to function within their intended operational electromagnetic environment without adversely affecting the operation of, or being adversely affected by the operation of any other equipment or system (12:Chap1,1). As radio equipment becomes more sophisticated and powerful, EMC problems become more complex. The research reported by this thesis addresses the problem of predicting significant intermodulation interference generated by colocated transmission systems.

Background

It has long been recognized that the frequency spectrum is a limited resource requiring cooperation among all users. Indiscriminate or uncoordinated use of the frequency spectrum can result in undesired affects often referred to as electromagnetic interference (EMI). One of the problems caused by EMI is degradation of electronic communications quality, i.e., disrupted or noisy communications. In the early days of electronic communications, with fewer users competing for frequency allocations, the possibility of EMI was usually not considered until a system actually experienced performance degradation. Then, an attempt was

undertaken to eliminate or suppress the EMI by filtering, reallocation of frequencies, or relocation of equipment, thus providing EMC among the systems. As use of the frequency spectrum increased, EMI increased and improved techniques to assure EMC were needed. Engineers began to investigate specific causes of EMI, developing techniques to avoid EMI where possible, and to combat EMI when it occurred.

The Problem

A significant area in EMI research is the electromagnetic compatibility of colocated radio communications equipment. Specifically, in the area of colocated VHF (30 to 300 MHz) communications equipment, a significant cause of EMI was found to be intermodulation products created by nonlinear effects within the power amplifier stage of the radio transmitter. Present techniques for the prediction of these intermodulation products rely almost entirely on computer programs based on frequency calculations alone. When the number of input frequencies involved in the analysis is large, the output from these programs becomes overwhelming, and useful information is difficult to extract. Even when the number of input frequencies is relatively small, much work still remains before the EMC engineer can determine if the output product describes an actual threat. More sophisticated

programs, which take into consideration additional variables such as transmitter power and path propagation loss, are available but do not lend themselves to use by the engineer in the field because they execute on large mainframe computers with limited access (11).

The Solution

This thesis provides the theoretical background for, and development of, an interactive algorithm for computing intermodulation products based not only on frequency calculations, but also on frequency separation, antenna separation, cable loss, and transmit power. The algorithm is implemented in the form of a computer program developed to run on the Zenith Z-100 microcomputer under the MS-DOS operating system to enhance its availability to Air Force engineers in the field. The program is written in Turbo Pascal to keep the source code structured and readable and thus improve its maintainability and likelihood of upgrade.

Overview Of The Remaining Chapters

Chapter II provides a detailed description of the problem. The underlying cause and mechanism creating the problem are discussed followed by an explanation of what can be done and how this thesis attacks the problem. Finally, four questions are posed which act as an outline of Chapters III and IV of this thesis and of the steps taken to solve

the problem.

Chapter III presents results of the literature search and a discussion of factors affecting transmitter intermodulation interference and the corresponding theory.

Chapter IV describes an algorithm for performing an intermodulation interference analysis of transmitter produced intermodulation products in colocated VHF sites, and its implementation in a Turbo Pascal program that runs on the Zenith Z-100 microcomputer.

Chapter VI provides some final comments and recommendations.

Appendix A is a user's manual for the program. It explains how to use the program, what the inputs and outputs are, and what assumptions, criteria, definitions, and formulas the output is based on.

Appendix B is a source code listing of the program.

II. THE PROBLEM

The Cause

When transmitting and receiving antennas are located in close proximity, the equipment is said to be colocated. Colocation may result in relatively large undesired signals (e.g. greater than -40 dBm) being applied to receiver inputs and transmitter outputs and can significantly increase the probability of interference due to receiver adjacent channel and cochannel effects, receiver spurious responses and intermodulation, case penetration, "rusty bolt" intermodulation, local oscillator radiation, transmitter broadband noise and spurious emissions, and transmitter intermodulation (8:Chap1,5). For this reason colocation should be avoided. However, colocation may be necessary, as is the case in on-board ship or aircraft communications, or desirable for logistics, aesthetic, or economic reasons. When equipment is colocated, each of the possible interference mechanisms must be analyzed and dealt with to provide electromagnetic compatibility. This thesis presents a method for dealing with the problem of interference among colocated VHF equipment due to transmitter intermodulation products.

The Mechanism

An intermodulation product is a signal produced by the mixing of two or more signals across a non-linear device. When colocated transmitters are operated simultaneously, their carrier signals may mix across any non-linear device, such as a receiver's front end, transmitter's final output power amplifier, or even a corroding metal-to-metal contact such as on a rusting wire fence. Intermodulation products that are created in a transmitter's final output power amplifier are referred to as transmitter intermodulation products. Because the mixing is non-linear, the frequency of the intermodulation product may be any linear combination of the frequencies of the original signals. In other words, the frequency of the intermodulation product will be the result of sums and differences of integer multiples of the frequencies of the original signals (7:268). If a transmitter intermodulation product falls on or near an allotted receive frequency and is of sufficient strength, it will interfere with the reception of the desired signal.

Although the frequency of an intermodulation product may be any linear combination of the frequencies of the original signals, in practice, only a limited number of transmitter intermodulation product types are of sufficient strength to be of interest - two signal second order, two and three signal third order, and two signal fifth order

intermodulation products (5:Chap2,23). Therefore, the frequency of a two or three signal intermodulation product may be written as

$$F_{IM} = k_1A + k_2B + k_3C$$

where

F_{IM} is the frequency of the intermodulation product,

k_1,k_2, and k_3 are integer coefficients, and

A, B, and C are the carrier frequencies of the signals mixing to form the intermodulation product.

The order of the intermodulation product is defined as the sum of the absolute values of the integer coefficient terms or

$$ORDER_{IM} = |k_1| + |k_2| + |k_3|$$

The strength of an intermodulation product depends on many factors including the strength of the original signals and where the mixing occurs. Some of the strongest intermodulation products are produced in the final power amplifiers of transmitters. This is due to the intentional operation of many power output devices in a non-linear manner (1:Chap2,24). As mentioned earlier, colocation results in relatively large undesired signals being applied to the transmitter outputs. This occurs because the antenna

of one transmitter may act as a receive antenna to signals transmitted by one or more other transmitters and apply the unwanted signals to the output of the associated transmitter. If an intermodulation product of sufficient strength is created in a transmitter that falls on a nearby receive frequency, there is nothing the receiver can do to guard against it.

What Can Be Done

Most interference due to transmitter intermodulation may be alleviated by transmitter output filtering, antenna placement, judicious frequency assignments, or operational practices (3:1). Transmitter output filtering should be used when possible but is relatively expensive, causes loss in output power, and drastically reduces the operational flexibility of the transmitter. Antenna placement is often fixed or only slightly modifiable. Judicious frequency assignments and operational practices provide the most acceptable means of controlling transmitter intermodulation interference.

To make judicious frequency assignments or develop effective operational practices, much must be known about the transmitter intermodulation mechanism. In the past, these decisions were based almost entirely on frequency considerations. Computer programs, such as the

Intermodulation And Harmonic Analysis Program from the Electromagnetic Compatibility Analysis Center, were developed to calculate a set of intermodulation interference free frequencies (2:Chap2,51). To be intermodulation interference free, the set of frequencies had to produce no intermodulation product that fell on, or within a predetermined guard band of, any assigned receive frequency. Using this technique, as the number of frequencies is increased, a point is reached where virtually no new frequencies may be added even though actual interference due to transmitter intermodulation products seldom occurs. The significance from a military point of view is that commanders are being denied communications capabilities based on the weak possibility that the additional signals will disrupt existing capabilities. Intermodulation interference analysis based on signal strength as well as frequency could alleviate this problem.

The Problem As Attacked By This Thesis

This thesis establishes criteria based on intermodulation product strength and frequency for use in determining if a set of VHF frequencies poses an actual threat of interference due to transmitter intermodulation products and incorporates the criteria in a computer program so that colocated VHF communications sites may be analyzed methodically and with minimum likelihood of error. The

criteria establishes a set of worst case conditions such that, unless these conditions are exceeded, interference due to transmitter intermodulation products is considered highly unlikely. This permits the use of many frequencies previously denied under the old analysis technique. To accomplish this the following questions must be answered:

(1) What transmitter intermodulation product types are significant?

(2) What factors affect the strength of a transmitter intermodulation product?

(3) What other factors determine if a transmitter intermodulation product will cause interference?

(4) Can these factors be generalized and quantified for incorporation into a computer program to predict non-interference?

III. RESULTS OF THE LITERATURE SEARCH

The Intermodulation Products Of Interest

Assume, as is often the case, that the final output power amplifier of a transmitter is operated in a nonlinear manner resulting in an output function that may be represented by a MacLaurin series. Figure III-1 is a model for such a power amplifier and Equation 3.1 represents its output function.

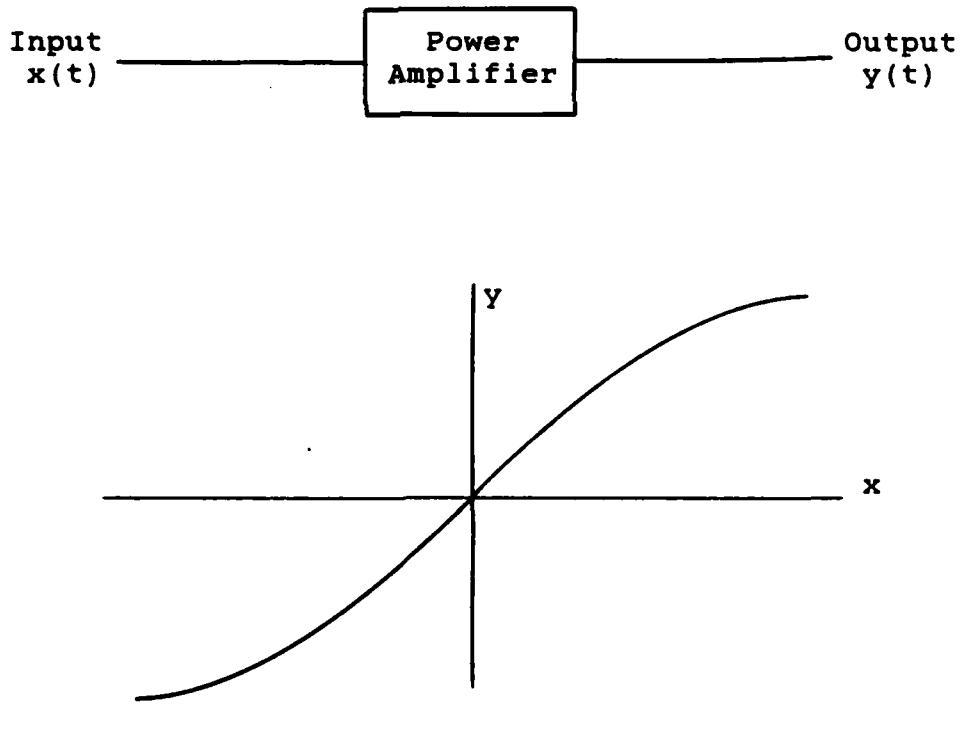


Figure III-1. A Power Amplifier And Its Nonlinear Transfer Function

$$y(x) = a_1x + a_2x^2 + a_3x^3 + a_4x^4 + \ldots \qquad (3.1)$$

Now suppose that the input signal, x(t), consists of the sum of two sinusoidal signals with radian frequencies of A and B such that

$$x(t) = b_1\cos(At) + b_2\cos(Bt) \qquad (3.2)$$

One of the sinusoidal signals may be the desired transmit signal and the other may be an undesired signal that has been applied to the amplifier's output from the transmitter's antenna and is fed into the amplifier's input via feedback circuitry within the amplifier itself. Then, by substituting Equation 3.2 into 3.1, the output becomes

$$\begin{aligned} y(t) = {} & a_1[b_1\cos(At) + b_2\cos(Bt)] \\ & + a_2[b_1\cos(At) + b_2\cos(Bt)]^2 \\ & + a_3[b_1\cos(At) + b_2\cos(Bt)]^3 \\ & + \ldots \end{aligned} \qquad (3.3)$$

Expanding the terms in brackets gives

$$\begin{aligned} y(t) = {} & a_1[b_1\cos(At) + b_2\cos(Bt)] \\ & + a_2[b_1^2\cos^2(At) + 2b_1b_2\cos(At)\cos(Bt) + b_2^2\cos^2(Bt)] \\ & + a_3[b_1^3\cos^3(At) + 3b_1^2b_2\cos^2(At)\cos(Bt) \\ & + 3b_1b_2^2\cos(At)\cos^2(Bt) + b_2^3\cos^3(Bt)] + \ldots \end{aligned} \qquad (3.4)$$

Finally, applying the trigonometric identities of Equation 3.5 and 3.6 to Equation 3.4, yields Equation 3.7 as the output of the amplifier.

$$\cos(X)\cos(Y) = (1/2)[\cos(X+Y) + \cos(X-Y)] \qquad (3.5)$$

$$\cos^2(X) = (1/2)[1 + \cos(2X)] \qquad (3.6)$$

$$\begin{aligned} y(t) = {} & a_1[b_1\cos(A\,t) + b_2\cos(B\,t)] \\ & + a_2\{(b_1^2/2)[1 + \cos(2At)] + b_1b_2[\cos([A+B]t) \\ & + \cos([A-B]t)] + (b_2^2/2)[1 + \cos(2B\,t)]\} \\ & + a_3\{(3b_1^3/4)\cos(At) + (b_1^3/4)\cos(3At) \\ & + (3b_1^2b_2/2)\cos(Bt) + (3b_1^2b_2/4)[\cos([2A-B]t) \\ & + \cos([2A+B]t)] + (3b_1b_2^2/2)\cos(At) \\ & + (3b_1b_2^2/4)[\cos([2B-A]t) + \cos([2B+A]t)] \\ & + (3b_2^3/4)\cos(Bt) + (b_2^3/4)\cos(3Bt)\} + \ldots \end{aligned} \qquad (3.7)$$

Examining Equation 3.7 shows that the output of the amplifier has component terms at frequencies other than the two original frequencies. The terms at integer multiples of the original frequencies are called harmonics. The terms made up of linear combinations of the original frequencies, i.e. sums and differences of integer multiples of the original frequencies, are called intermodulation products and are the terms of concern in this thesis. If the intermodulation products are created in the final output

power amplifier of a transmitter, they are called transmitter intermodulation products.

Although transmitter intermodulation products will be generated with frequencies at linear combinations of the frequencies that are mixed, not all intermodulation products will have sufficient strength to cause interference. To be significant, transmitter intermodulation products must fall within an adjacent channel of the transmitter in which the mixing occurs (5:Chap2,24). The term adjacent channel refers to the bandwidth of the transmitter centered around the tuned frequency or any of its harmonics. But generally, intermodulation products falling within an harmonic adjacent channel are of insufficient level to be of concern (6:Chap2,12). This greatly reduces the number of significant intermodulation product types. There are two possible second order intermodulation products known as the A + B and the A - B intermodulation products. Both are of interest when analyzing transmitter intermodulation interference. There are five types of third order intermodulation products but only the two signal 2A - B, and three signal A + B - C intermodulation products are of interest. The final intermodulation product of interest is the two signal, fifth order product 3A - 2B. All other higher order and sum combinations result in intermodulation products sufficiently far from the transmitter's fundamental

frequencies that they are usually not detected (5:Chap2,24).

Factors Affecting Intermodulation Product Strength

Figure III-2 represents a colocated radio communications site and will be used to describe how interference be caused by transmitter intermodulation products.

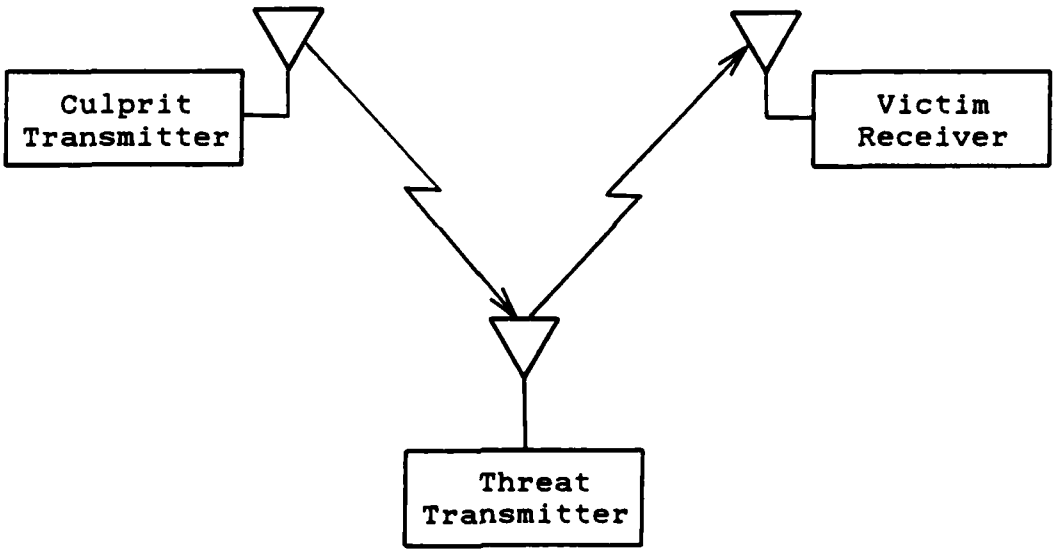


Figure III-2. An Example Colocated Site.

A "culprit frequency" is any frequency that may be involved in the creation of intermodulation interference. A

"culprit transmitter" is a transmitter producing a culprit frequency and a "culprit antenna" is an antenna connected to a culprit transmitter. The culprit transmitter in Figure III-2 is communicating with a receiver somewhere outside of the colocated site; however, the signal it is transmitting will also propagate across the colocated site and be picked up by the threat transmitter's antenna. The "threat transmitter" is the culprit transmitter in which the intermodulation product is being created and the threat transmitter's antenna is the "threat antenna". The carrier frequency to which the threat transmitter is tuned is referred to as the "threat frequency". When the culprit signal is picked up by the threat antenna, it is feed down the cable and applied to the threat transmitter's final output power amplifier, as described previously. Here the culprit signal mixes with the threat signal to create transmitter intermodulation products. The intermodulation products will then be transmitted along with the desired signal and be picked up by the victim receiver's antenna. The "victim receiver" is the receiver being interfered with and the victim receiver's antenna is the "victim antenna". The victim receiver's tuned frequency is referred to as the victim frequency. If one of the intermodulation products has a frequency within the passband of the victim receiver and is of sufficient strength, it will interfere with the reception of the desired signal.

The question then is how strong will an intermodulation product be by the time it reaches the front end of the victim receiver? Maiuzzo derived the following equation for estimating the power level of intermodulation products produced by the mixing of two signals in the output stage of a transmitter and thus provided a start at answering this question (9:136). An examination of the individual terms in the equation will reveal the important factors affecting the power level of an intermodulation product at the antenna port of the threat antenna.

$$P_{IM} = n(P_c - S_t(F_c)) - S_t(F_{IM}) - (n - 1)b - K_1 \qquad (3.8)$$

where

P_{IM} is the power, in dBm, of the intermodulation product at the antenna port of the threat antenna.

n is the harmonic number of the culprit frequency. The power of the intermodulation product is inversely proportional to the harmonic number of the culprit frequency and independent of the harmonic number of the threat frequency (9:133). This implies that the strength of the intermodulation product is inversely proportional to the order of the intermodulation product. It also implies a significant dependence on which transmitter is considered the threat transmitter. For example, for a two signal third order intermodulation product of the form 2A-B, the

intermodulation product will be stronger if A is considered the threat frequency than if B is considered the threat frequency if all other terms in Equation 3.8 are the same in both cases.

P_c is the power level, in dBm, of the culprit signal at the threat transmitter's antenna port. The power level of the intermodulation product is directly proportional to the power level of the culprit signal. This power level may be determined from (4:96)

$$P_c = P_{co} - L_{cc} + G_c - L_p + G_t - L_{tc} \qquad (3.9)$$

where

P_{co} is the output power of the culprit transmitter in dbm,

L_{cc} is the coupling loss, in dB, between the culprit transmitter and the culprit antenna,

G_c is the gain, in dB, of the culprit antenna in the direction of the threat antenna,

L_p is the free space propagation loss, in dB, between the culprit antenna and the threat antenna. Free space propagation loss may be determined from (4:96)

$$L_p = -37.85 + 20\text{Log}(DF) \qquad (3.10)$$

where

D is the distance, in feet, between the two antennas concerned, in this case the culprit and threat antennas,

F is the frequency, in MHz, of concern, in this case, the culprit frequency, and

Log() is the logarithm, base 10, of the expression in the parenthesis.

G_t in (3.9) is the gain, in dB, of the threat antenna in the direction of the threat antenna, and

L_{tc} is the coupling loss, in dB, between the threat antenna and the threat transmitter.

$S_t(f)$ in (3.8) is the selectivity in dB of the threat transmitter's power amplifier output stage to frequency f. Selectivity, which is the amount of attenuation an off tune frequency would experience when passing through the output stage of a transmitter as compared to the on tune frequency, varies with each transmitter but conservative estimates may be obtained from Figure III-3. The power level of the intermodulation product varies inversely with the frequency separation between the culprit and threat frequencies.

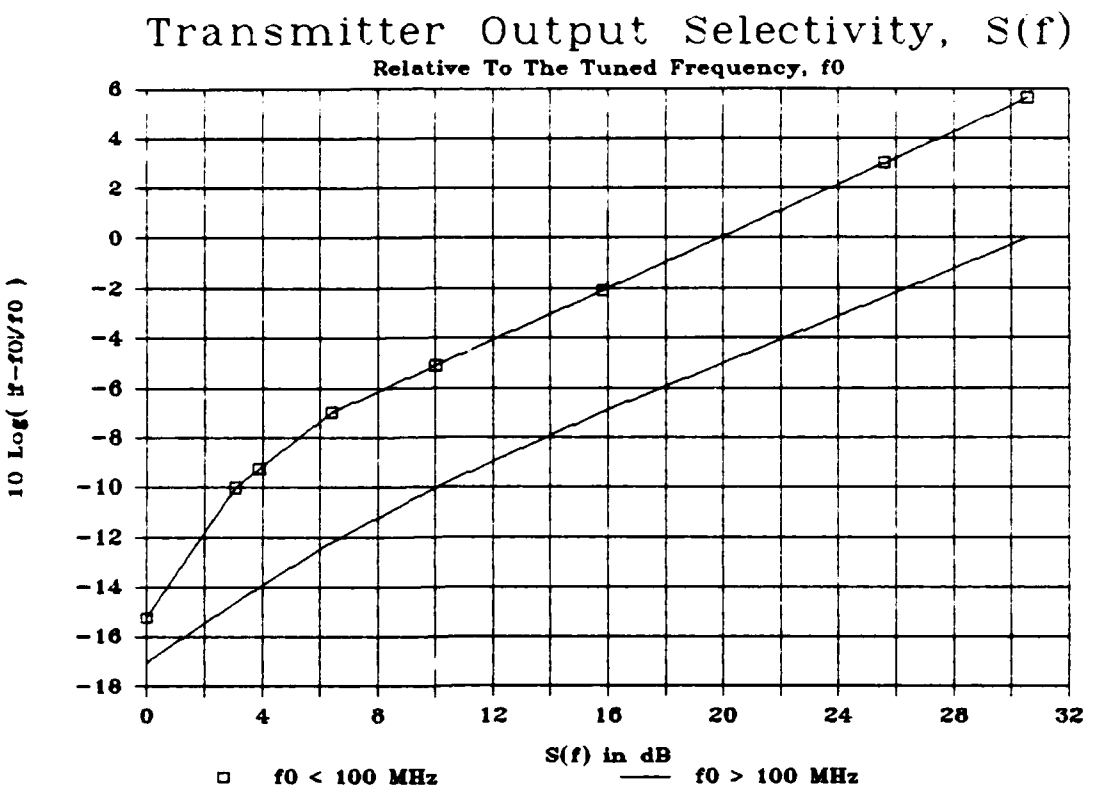


Figure III-3. Conservative Estimates Of Transmitter Output Selectivity (8:Chap3,5).

F_t is the frequency of the threat transmitter's carrier in Mhz. The power in the intermodulation product is not dependent on the threat frequency.

F_c is the frequency of the culprit transmitter's carrier in MHz. The strength of the intermodulation product is not dependent on the culprit frequency itself but on the frequency separation between the threat and culprit frequencies as reflected in the threat transmitter's

selectivity term. The greater the separation, the lower the power level of the intermodulation product.

F_{IM} is the frequency, in MHz, of the transmitter produced intermodulation product being examined. The strength of the intermodulation product is not dependent on the intermodulation product frequency itself but on the frequency separation between the threat frequency and the intermodulation product frequency, again as reflected in the threat transmitter's selectivity term. The strength of the intermodulation product is inversely proportional to the separation between the intermodulation product frequency and the threat frequency.

b is 30 dB if the threat transmitter's final power amplifier is a solid state type and 40 db if the threat transmitter's final power amplifier is a tube type. Maiuzzo determined these values empirically from an examination of the lower five transmitter intermodulation coefficients for 52 transmitters with tube power amplifiers and 3 transmitters with solid state power amplifiers (9:134).

K_1 is the threat transmitter's intermodulation coefficient, in dB, which varies with each transmitter. The values of K_1 for approximately 50 transmitters are being catalogued at the Electromagnetic Compatibility Analysis

Center but for a worst case analysis a K_1 value of 0 dB may be used (9:134,136).

Although Maiuzzo's equation was developed for the case of a two signal transmitter intermodulation product it is easily extended to handle the three signal case as well (11:Chap2,36). In this case

$$P_{IM} = n(P_{c1} - S_t(F_{c1})) + m(P_{c2} - S_t(F_{c2})) - (n + m - 1)b - K_1 - S_t(F_{IM}) \quad (3.11)$$

where

the subscripts c1 and c2 are used to relate the associated terms with culprit signal number one and culprit signal number two respectively, and

n and m are the harmonic numbers of culprit signal number one and culprit signal number two respectively.

All other terms maintain the definitions given above.

In summary, Maiuzzo's equation reveals the strength of a transmitter intermodulation product to be:

* inversely proportional to the order of the intermodulation product,
* dependent on which transmitter is the threat

transmitter,

* directly proportional to the culprit transmitter's output power,
* inversely proportional to the coupling loss between the transmitters and their associated antennas,
* directly proportional the the gains of the antennas,
* inversely proportional to the separation between the threat antenna and the culprit antennas,
* inversely proportional to the frequency separation between the threat frequency and the culprit frequencies, and
* dependent on the type of transmitters involved.

Other Factors Requiring Consideration

For a transmitter intermodulation product to cause interference it must have a frequency at or near the operating frequency of a receiver (i.e. within the narrowest predetector passband of the receiver) and a power level high enough to be detected at the front end of the receiver (13:Chap4,8). Equations for calculating the frequency and power level of an intermodulation product at the output of the threat transmitter have already been presented. It is now necessary to examine the requirements imposed by the receiver.

A receiver's output signal to noise ratio or error rate depends on the modulation type of the desired signal, the type of interfering signal, receiver characteristics, and the input signal to interference ratio (6:Chap1,8). The variability of these factors presents a significant computational problem but the utility of such computations is also questionable since determining what constitutes an acceptable amount of interference varies with the application. For this reason, the thermal noise level at the front end of the receiver is often used as the minimum interference signal level in performing a transmitter intermodulation interference analysis (13:Chap4,9). Thermal noise may be calculated from

$$N_t = 30 + 10 \text{ Log}(KTB) \qquad (3.12)$$

where

N_t is the thermal noise in dBm,

K is Boltzman's constant = 1.38×10^{-23} Watts/Degree-Kelvin-Hertz,

T is the operating temperature in degrees Kelvin, and

B is the receiver's bandwidth in Hertz.

Assuming operation at 290 degrees Kelvin and that the bandwidths of interest are more conveniently expressed in kHz this equation may be rewritten as

$$N_t = 10 \text{ Log}(B) - 144 \qquad (3.13)$$

where

B is now the receiver's bandwidth in kHz, and

N_t is still the thermal noise expressed in dBm.

Any undesired signal exceeding this level may cause interference. Therefore, the signal level of the intermodulation product at the front end of the receiver must be calculated. This may be accomplished using

$$P_{RIM} = P_{IM} - L_{tc} + G_t - L_p + G_v - L_{vc} \qquad (3.14)$$

where

P_{RIM} is the power, in dBm, of the intermodulation product at the front end of the victim receiver,

P_{IM} is the power level, in dBm, of the intermodulation product at the output of the threat transmitter,

L_{tc} is the coupling loss, in dB, between the threat transmitter and its associated antenna,

G_t is the gain, in dB, of the threat antenna in the direction of the victim receiver's antenna,

L_p is the propagation loss, in dB, between the threat and victim antennas,

G_v is the gain, in dB, of the victim receiver's antenna in the direction of the threat antenna, and

L_{vc} is the coupling loss, in dB, between the victim receiver and its associated antenna.

In summary, whether an intermodulation product at the front end of a receiver will cause interference or not depends on

* the power level of the intermodulation product at the threat transmitter (the stronger the intermodulation product, the more likely it is to create interference),
* the coupling loss between the threat antenna and the victim receiver (the greater the coupling loss, the less likely the intermodulation product is to create interference),
* the gain of the threat antenna in the direction of the victim receiver (the greater the gain, the more likely the intermodulation product is to create interference),
* the separation between the threat antenna and the victim receiver (the greater the separation, the less likely the intermodulation product is to create interference),
* the gain of the victim antenna in the direction of the threat antenna (the greater the gain, the more likely the intermodulation product is to create interference),
* the coupling loss between the victim antenna and the victim receiver (the greater the loss, the less likely the intermodulation product is to create interference),
* and the bandwidth of the victim receiver (the greater the bandwidth, the less likely the intermodulation product is to create interference).

IV. THE ANALYSIS ALGORITHM

Introduction

Chapter III presented the criteria and associated equations necessary to determine if a transmitter-produced intermodulation product should be considered a threat to communications or not. Although the application of this criteria in the analysis of a colocated VHF site may at first appear straightforward, an efficient and well organized method is required for their use because of the large number of calculations involved. For example, suppose the site being analyzed used only four frequencies: f1, f2, f3, and f4. To find the second order A+B intermodulation products that may cause interference, one might first add f1 to f2 and compare the result to f3 and f4. The next step would be to add f1 to f3 and compare the result to f2 and f4. This procedure would be continued, adding f1 to f4 and comparing to f2 and f3, adding f2 to f1 and comparing to f3 and f4, etc., until all possible sums were computed and comparisons made, resulting in a total of 12 summations and 24 comparisons. Generalizing this result for n frequencies, a total of n*(n-1) summations and n*(n-1)*(n-2) comparisons are needed. Of course, many more calculations would be needed to determine if the intermodulation products are within the bandwidths of the victim frequencies, what the strengths of the intermodulation products are, what the

noise levels at the front ends of the victim receivers are, and to calculate similar information for the other second, third, and fifth order intermodulation products. Making these calculations by hand is extremely time consuming and prone to error, but simply reproducing this procedure in a computer program will not solve the problem either. First of all, the procedure addresses only a small portion of the overall analysis problem, calculation of the intermodulation product frequencies and comparison with the victim frequencies. Secondly, the procedure is inefficient in terms of the number of calculations necessary. For example, the sum of frequencies f1 and f2 is the same as the sum of frequencies f2 and f1 and therefore one of the sums could be eliminated. And finally, the procedure does not produce results in an organized manner. Interference to frequency f3 may be found, then interference to frequency f1, then interference to f4, and then f3 again, etc. To place these results in order so that all possible combinations resulting in interference to f1 will be presented first and then all combinations resulting in interference to f2, etc., would require large amounts of computer memory not readily available in microcomputers.

This chapter presents an algorithm for accomplishing a systematic analysis of VHF sites with colocated equipment and its implementation in a Turbo Pascal program that runs

on the Zenith Z-100 microcomputer under the MS-DOS operating system. The overall algorithm is as follows:

(1) Collect data on the electromagnetic environment.

(2) Organize the data.

(3) Select a victim frequency for analysis

(4) Calculate an intermodulation product frequency

(5) If the intermodulation product will interfere with the victim frequency then print this combination.

(6) Repeat steps 4 and 5 until all possible intermodulation products have been calculated.

(7) Repeat steps 3, 4, 5, and 6 until all victim frequencies have been tested for interference.

(8) Review the results of the intermodulation analysis and update the data as appropriate.

(9) Repeat steps 3 through 8 until no more transmitter produced intermodulation interference is indicated.

Each step in the above algorithm will now be discussed in detail. During an analysis, each step and sub-step, as detailed below, should be executed in the order given.

Collect Data

This step may well be the most difficult and time consuming phase of the entire analysis. The computer

program indirectly provides tremendous assistance in this phase of the analysis by precisely defining the types of information needed as input. As with any computer program, the accuracy of the input data will determine the validity of the output results. The information required is divided into two categories, antenna data and frequency data.

Antenna Data

Antenna data is gathered separately from the frequency data to ease input into the program. Several frequencies may be used on a single antenna, so instead of entering all necessary antenna information with each frequency, only an associated antenna number will need to be entered with each frequency. This reduces the amount of information that must be entered into the program and also reduces the amount of memory necessary to store the information.

The strength of an intermodulation product is inversely proportional to the square of the distances between the antennas. To calculate these distances the program will need the antenna locations and heights expressed in feet. Antenna locations may be specified by obtaining or making a scale map of the site with all antenna positions plotted. Since the separation between antennas is all that's important, a grid may be overlaid on the map with an arbitrary origin and orientation. Each antenna should be

numbered sequentially beginning with one. The location of each antenna may then be specified by a pair of x y coordinates. The program expects the x,y coordinates to be in feet so the grid should be designed appropriately. Figure IV-1 is an example of an aircraft control tower antenna layout prepared appropriately for use as input to the program. Each antenna has been numbered and a suitable grid has been overlaid.

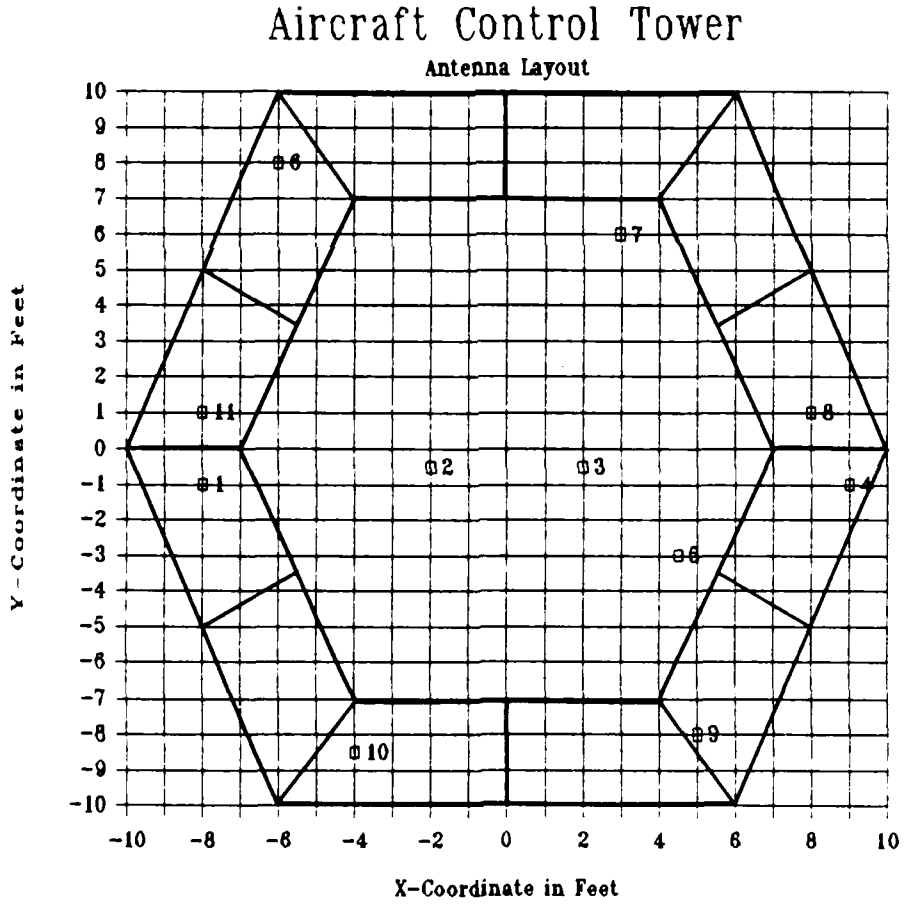


Figure IV-1. An Antenna Layout Prepared For Program Input. (10:Appendix 1)

Once the antenna numbers and locations have been assigned, the antenna heights must be determined. Since the relative heights are all that is needed to calculate antenna separations, any reference which eliminates terrain variations could be used. Height above sea level is assumed by the program but any elevation could be used.

Antenna gains are the final data needed. This presents a problem in that antenna gain varies with frequency and the look angle, or direction from the point of interest to the antenna relative to the antennas main beam axis. Maximum interference will result when the antenna gains involved are maximum. Maximum antenna gains could occur if the culprit, threat, and victim antennas were all pointing at each other and the gains of the antennas were the same for all frequencies involved. This represents the worst case situation and is the situation assumed by the program. Therefore, the gains needed are the gains of the antennas main lobe in dBi, or decibels above isotropic, at the specified operating frequency.

The information needed on each antenna then is:

* an antenna number assigned sequentially,
* the antenna location specified as x,y coordinates in feet,
* the antenna height in feet above sea level, and

* the antenna gain in dBi.

Frequency Data

All frequencies being used in and around the site of concern must be included in the analysis. The frequencies should be expressed in MHz. If the same frequency is used on more than one antenna, the frequency should be listed for each antenna used.

The antenna number of the antenna being used by each frequency should be listed along with each frequency. Since all other information about the antenna should have already been collected, the antenna number serves as a link between the antenna data and the frequency data.

To determine if the frequency may be considered a culprit and/or victim frequency, a "usage" code should be associated with each frequency. A frequency may be used as receive only, transmit only, or both transmit and receive. Consequently, a logical code - and the one used by the program - is R for receive only, T for transmit only, and B for both transmit and receive. The program uses this information to prevent a transmit-only frequency from being considered as a victim frequency or a receive-only frequency from being considered as a culprit frequency. It also uses this information during entry of the data to save the user

from having to enter data associated only with transmit frequencies.

Each signal's bandwidth should be determined and expressed in kHz. The bandwidth is used to determine the thermal noise level at the front end of the victim receiver.

Losses, in dB, between the radios and their antennas should be determined next. These losses will be referred to as cable losses but actually include insertion and return losses due to cables, connectors, multicouplers, etc. Cable losses are frequency dependent but for a worst case analysis may be assumed to be frequency independent and equal to the losses at the operating frequency of the radio. Therefore the cable losses need only be determined at each operating frequency.

Some frequencies are designated for use only at night, only during the day, or only during a contingency (war, disaster, etc.). This information may be used to reduce the number of frequencies that may combine to produce intermodulation products under certain conditions but is not implemented in the present program. This information should still be gathered, however, for use in interpreting the results of the program or for future use. For this purpose a "time used" code should be assigned to each frequency.

The codes used by the program are D for a frequency used only during the day, N for a frequency used only during the night, B for a frequency used both day and night, and C for a frequency used only during a contingency.

For transmit frequencies, the radio's output power is needed. This should be the output power, expressed in Watts, that is applied to the antenna port of the radio.

A transmitter with a tube type of final output power amplifier has different intermodulation performance characteristics than a transmitter with a solid state final output power amplifier. For this reason a "transmitter type" code should be assigned to each transmit frequency. The program uses T for a transmitter with a tube type of final output amplifier and S for a transmitter with a solid state type of output amplifier.

The final frequency data needed are the intermodulation coefficients, in dB, of each transmitter. The intermodulation coefficients vary with each transmitter model and are being catalogued by the Electromagnetic Compatibility Analysis Center (ECAC), Annapolis, Maryland 21402. For a worst case analysis these coefficients may be assumed to be 0 dB.

The data needed on each frequency is:

* The frequency, in MHz, of the carrier,
* The antenna number of the antenna used by the frequency,
* A usage code of T, R, or B denoting whether the frequency is used for transmit only, receive only, or both transmit and receive, respectively,
* The bandwidth, in kHz, occupied by the signal,
* The cable losses, in dB, between the radio and its antenna,
* A time used code of D, N, B, or C denoting whether the frequency is assigned for use during the day only, during the night only, during both day and night, or during a contingency only, respectively,
* The transmitter's output power, in Watts, for each transmit frequency,
* A transmitter type code of T or S denoting whether the transmitter's final output power amplifier is a tube type or a solid state type, respectively, for each transmit frequency, and
* A transmitter intermodulation coefficient, in dB, as assigned by ECAC for each transmit frequency.

Organize The Data

This step is necessary for two reasons. The first is that organizing the data enhances its usefulness to the

engineer. Errors or omissions will be easier to detect and reports and updates will be easier to prepare. The second reason is that by having the data organized, the analysis algorithm can take advantage of positional relationships of the among the data to reduce the number of computations necessary to complete the analysis. For example, suppose the frequencies are placed in increasing order so that frequency f2 is higher than frequency f1, frequency f3 is higher than frequency f2, f4 is higher than f3, etc. Then in calculating second order intermodulation products of the form A+B that may interfere with frequency f20, no frequencies above frequency f19 need be considered as either culprit A or B since they would already be higher than the victim frequency. Regardless of the actual frequencies involved, no frequency when added to frequency f21 or above could result in an intermodulation product falling on frequency f20.

The program provides several features to assist with this step of the algorithm. Entry and manipulation of the data is guided through the use of menus, prompts, and on screen instructions. Error checking is performed during entry to assure reasonable data input where possible. The data is sorted automatically and may be displayed on screen or written to disk in a format suitable for printing or manipulation with a text editor. Changes may be made to the

data using a built in, screen oriented editor. And finally, the data may be saved on a disk so it may be loaded back in later for updating, review, or analysis. Appendix A contains detailed instructions on the operation of the program. A description of the data organization is given below.

Antenna Data Organization

Antenna data is arranged in the form of a table with columns for the antenna number, x-coordinate, y-coordinate, height, and gain. The antennas are placed in ascending order by antenna number with no numbers skipped. If an antenna is deleted, all antenna numbers above the antenna number being deleted will be decreased by one so that the continuity of numbering will be maintained. The frequency data will be updated automatically with the new antenna numbers when an antenna is deleted and any frequencies using the deleted antenna will be deleted. Since antenna numbering is changed by the deletion of an antenna, the site map will have to be updated any time a deletion is made. Figure IV-2 is an example of the antenna table resulting from the antenna layout of Figure IV-1.

ANTENNA NUMBER	X COORDINATE (FEET)	Y COORDINATE (FEET)	HEIGHT (FEET)	GAIN (dBi)
1	-8.00	-1.00	1324.00	5.50
2	-2.00	-0.50	1327.00	6.00
3	2.00	-0.50	1327.00	6.00
4	9.00	-1.00	1324.00	5.50
5	4.50	-3.00	1327.00	6.00
6	-6.00	8.00	1324.00	5.50
7	4.00	6.00	1327.00	6.00
8	8.00	1.00	1324.00	5.50
9	5.00	-8.00	1324.00	5.50
10	-4.00	-8.50	1324.00	5.50
11	-8.00	1.00	1324.00	5.50

Figure IV-2. An Antenna Table Example.

Frequency Data Organization

The frequency data is organized into records containing all information pertaining to the frequency. The antenna data associated with a specific frequency is obtained by referencing the antenna table via the antenna number. The frequency records are arranged in ascending order by frequency. Figure IV-3 is an example of a frequency record.

Select A Victim Frequency For Analysis

The data should be analyzed for possible interference to all frequencies except those assigned as transmit only frequencies. To keep the results organized and therefore minimize the effort necessary to prepare the data for

```
Frequency :  323.00 MHz          Antenna Number :   7
Antenna X Coordinate : 4.00      Antenna Y Coordinate : 6.00
Antenna Height :  1327.00        Antenna Gain :    6.00
Bandwidth :   16.00 kHz          Cable Loss :    2.00 dB
Assigned as a receive only frequency.
For day and night time use.
Transmitter Output Power : N/A
Final Output Amplifier Type : N/A
Transmitter Intermodulation Coefficient : N/A
```

Figure IV-3. An Frequency Record Example.

inclusion in a report, victim frequencies should be selected in ascending order by frequency and all intermodulation products resulting in interference to one victim frequency should be indicated before selecting the next victim frequency for analysis. The program accomplishes all of the above automatically.

Calculate The Intermodulation Product Frequencies

Because a large number of calculations are inherent in an intermodulation interference analysis, efficient algorithms are needed to reduce the number of calculations whenever possible. This in turn reduces the time an engineer must wait for the results. With this in mind, a different algorithm has been developed for each of the intermodulation product types. The algorithms take advantage of the frequencies being arranged in ascending

order and capitalize on the unique mathematical characteristics of each intermodulation product type. The notation f1 will be used to represent the first frequency in the frequency list, f2 to represent the second frequency, f3 the third, and so on throughout the discussion of the intermodulation product calculating algorithms. The algorithms will also assume that all frequencies in the list to be analyzed are unique. This may not be the case in an actual analysis as identical frequencies may be assigned to more than one antenna with one assignment acting as the primary antenna and the remaining ones acting as backup. The program tests for this situation and analyzes each frequency/antenna pair separately assuming that the frequency will only be used on one antenna at a time.

The A+B Intermodulation Products

The victim frequency of an A+B intermodulation product must be higher than both culprit frequencies A and B. The program is able to take advantage of this fact and the fact that the frequencies are arranged in increasing order to prevent doing useless calculations on frequencies above the victim frequency. The algorithm is as follows:

(1) Select a frequency for A beginning with f1. Each time a new A is needed, select the next higher frequency until the frequency two frequencies below the victim

frequency is reached. For example, if the victim frequency were f5, then the first A frequency would be f1 and the last A frequency would be f3.

(2) Select a frequency for B beginning with the frequency just above the A frequency. Each time a new B is needed, select the next higher frequency until the frequency just below the victim frequency is reached. Continuing the example started in step (1) above with f5 selected as the victim frequency, when f1 is selected as A then the first B would be f2 and the last B would be f4. When f2 is selected as A then the first culprit B would be f3 and the last B would be f4.

(3) Calculate the intermodulation product. If A plus B is within the bandpass of the victim frequency then the possibility for interference exists and further processing is necessary. Otherwise continue with the next step.

(4) Repeat steps 2 and 3 until all eligible frequencies have been selected for B.

(5) Repeat steps 1, 2, and 3 until all eligible frequencies have been selected for A. Once this has been accomplished, all possible interference to the victim frequency from A+B intermodulation products will have been found. Pseudo code representing this algorithm is given in Figure IV-4.

```
For A = f1 to two_freqs_below_the_victim
  For B = the_freq_just_above_A to one_freq_below_the_victim
    The_im_product = A+B
    If the_im_product_is_within_the_bandwidth_of_the_victim
      then see_if_it's_strong_enough_to_cause_interference
    If the_im_product_is_strong_enough_to_cause_interference
      then print_the_pertinent_intermodulation_information
  Get_the_next_B
Get_the_next_A
```

Figure IV-4. Pseudo Code For The Calculation Of A+B Intermodulation Products

The A-B Intermodulation Products

The A frequency of an A-B intermodulation product must be higher than the victim frequency. The B frequency may be any frequency below the A frequency except the victim frequency. The algorithm exploiting these facts is given below. It should be noted that the selection of the B frequency is divided into two sequences or loops to eliminate the need for a test within the loop to assure that the victim frequency is not used as the B frequency. This complicates the algorithm slightly but results in increased execution speed.

(1) Select a frequency for A beginning with the frequency just above the victim frequency. Each time a new A is needed, select the next higher frequency until the highest frequency is reached.

(2) Select a frequency for B beginning with f1. Each time a new B is needed, select the next higher frequency until the frequency just below the victim frequency is reached.

(3) Calculate the intermodulation product. If A minus B is within the bandpass of the victim frequency then the possibility for interference exists and further processing is necessary. Otherwise continue with the next step.

(4) Repeat steps 2 and 3 until all eligible frequencies have been selected for B.

(5) Select a frequency for B beginning with the frequency just above the victim frequency. Each time a new B is needed select the next higher frequency until the frequency just below the A frequency is reached.

(6) Calculate the intermodulation product. If A minus B is within the bandpass of the victim frequency then the possibility for interference exists and further processing is necessary. Otherwise continue with the next step.

(7) Repeat steps 5 and 6 until all eligible frequencies have been selected for B.

(8) Repeat steps 1 through 7 until all eligible frequencies have been selected for A. Once this has been accomplished, all possible interference to the victim frequency from A-B intermodulation products will have

been found. Pseudo code representing this algorithm is given in Figure IV-5.

```
For A = one_freq_above_the_victim to the_highest_freq
  For B = f1 to one_freq_below_the_victim
    The_im_product = A-B
    If the_im_product_is_within_the_bandwidth_of_the_victim
      then see_if_it's_strong_enough_to_cause_interference
    If the_im_product_is_strong_enough_to_cause_interference
      then print_the_pertinent_intermodulation_information
  Get_the_next_B
  For B = one_freq_above_the_victim to one_freq_below_A
    The_im_product = A-B
    If the_im_product_is_within_the_bandwidth_of_the_victim
      then see_if_it's_strong_enough_to_cause_interference
    If the_im_product_is_strong_enough_to_cause_interference
      then print_the_pertinent_intermodulation_information
  Get_the_next_B
Get_the_next_A
```

Figure IV-5. Pseudo Code For The Calculation Of A-B Intermodulation Products

The 2A-B Intermodulation Products

The A frequency of a 2A-B intermodulation product may be any frequency except the victim frequency. By dividing the selection of the A frequency into two loops, a test within the loop to assure that the victim frequency is not used as the A frequency is eliminated. Again, this complicates the code but increases the execution speed. The B frequency may be any frequency except the victim or A

frequencies. Dividing the selection of the B frequency into three loops eliminates the need for two tests, one to assure that the victim frequency is not used for B and the other to assure that the A frequency is not used for B. The result is increased execution speed. The algorithm is given below.

(1) Select a frequency for A beginning with f1. Each time a new A is needed, select the next higher frequency until the frequency just below the victim frequency is reached.

(2) Select a frequency for B beginning with f1. Each time a new B is needed, select the next higher frequency until the frequency just below A is reached. Note that A will always be below the victim frequency for this part of the algorithm.

(3) Calculate the intermodulation product. If two times A minus B is within the bandpass of the victim frequency then the possibility for interference exists and further processing is necessary. Otherwise continue with the next step.

(4) Repeat steps 2 and 3 until all eligible frequencies have been selected for B.

(5) Select a frequency for B beginning with the frequency just above A. Each time a new B is needed select the next higher frequency until the frequency just below the victim. Again, A is always lower than the victim

frequency in this part of the algorithm.

(6) Calculate the intermodulation product. If two times A minus B is within the bandpass of the victim frequency then the possibility for interference exists and further processing is necessary. Otherwise continue with the next step.

(7) Repeat steps 5 and 6 until all eligible frequencies have been selected for B.

(8) Select a frequency for B beginning with the frequency just above the victim frequency. Each time a new B is needed select the next higher frequency until the highest frequency is reached.

(9) Calculate the intermodulation product. If two times A minus B is within the bandpass of the victim frequency then the possibility for interference exists and further processing is necessary. Otherwise continue with the next step.

(10) Repeat steps 8 and 9 until all eligible frequencies have been selected for B.

(11) Select a frequency for A beginning with the frequency just above the victim frequency. Each time a new A is needed, select the next higher frequency until the highest frequency is reached.

(12) Select a frequency for B beginning with f1. Each time a new B is needed, select the next higher frequency until the frequency just below the victim frequency is

reached. Note that the victim frequency will always be below A for this part of the algorithm.

(13) Calculate the intermodulation product. If two times A minus B is within the bandpass of the victim frequency then the possibility for interference exists and further processing is necessary. Otherwise continue with the next step.

(14) Repeat steps 12 and 13 until all eligible frequencies have been selected for B.

(15) Select a frequency for B beginning with the frequency just above the victim frequency. Each time a new B is needed select the next higher frequency until the frequency just below A. Again, the victim frequency is always lower than A in this part of the algorithm.

(16) Calculate the intermodulation product. If two times A minus B is within the bandpass of the victim frequency then the possibility for interference exists and further processing is necessary. Otherwise continue with the next step.

(17) Repeat steps 15 and 16 until all eligible frequencies have been selected for B.

(18) Select a frequency for B beginning with the frequency just above A. Each time a new B is needed select the next higher frequency until the highest frequency is reached.

(19) Calculate the intermodulation product. If two times A

minus B is within the bandpass of the victim frequency then the possibility for interference exists and further processing is necessary. Otherwise continue with the next step.

(20) Repeat steps 18 and 19 until all eligible frequencies have been selected for B.

(21) Repeat steps 1 through 20 until all eligible frequencies have been selected for A. Once this has been accomplished, all possible interference to the victim frequency from 2A-B intermodulation products will have been found. Pseudo code representing this algorithm is given in Figure IV-6.

The A+B-C Intermodulation Products

The need for increased execution speed has resulted in a complex algorithm for calculating A+B-C intermodulation products. In exchange for the complexity, execution speeds better than 15 times faster than a simple looping algorithm were achieved during development using as few as 30 frequencies. Since the validity of this particular algorithm is not intuitively obvious, a detailed discussion of the reasoning behind the algorithm will be given.

The calculation of A+B-C intermodulation products may be divided into four cases or loops based on the relative position of the victim frequency with respect to the culprit

```
For A = f1 to one_freq_below_the_victim
  For B = f1 to one_freq_below_A
    The_im_product = 2A-B
    If the_im_product_is_within_the_bandwidth_of_the_victim
      then see_if_it's_strong_enough_to_cause_interference
    If the_im_product_is_strong_enough_to_cause_interference
      then print_the_pertinent_intermodulation_information
  Get_the_next_B
  For B = one_freq_above_A to one_freq_below_the_victim
    The_im_product = 2A-B
    If the_im_product_is_within_the_bandwidth_of_the_victim
      then see_if_it's_strong_enough_to_cause_interference
    If the_im_product_is_strong_enough_to_cause_interference
      then print_the_pertinent_intermodulation_information
  Get_the_next_B
  For B = one_freq_above_the_victim to the_highest_freq
    The_im_product = 2A-B
    If the_im_product_is_within_the_bandwidth_of_the_victim
      then see_if_it's_strong_enough_to_cause_interference
    If the_im_product_is_strong_enough_to_cause_interference
      then print_the_pertinent_intermodulation_information
  Get_the_next_B
For A = one_freq_above_the_victim to the_highest_freq
  For B = f1 to one_freq_below_the_victim
    The_im_product = 2A-B
    If the_im_product_is_within_the_bandwidth_of_the_victim
      then see_if_it's_strong_enough_to_cause_interference
    If the_im_product_is_strong_enough_to_cause_interference
      then print_the_pertinent_intermodulation_information
  Get_the_next_B
  For B = one_freq_above_the_victim to one_freq_below_A
    The_im_product = 2A-B
    If the_im_product_is_within_the_bandwidth_of_the_victim
      then see_if_it's_strong_enough_to_cause_interference
    If the_im_product_is_strong_enough_to_cause_interference
      then print_the_pertinent_intermodulation_information
  Get_the_next_B
  For B = one_freq_above_A to the_highest_freq
    The_im_product = 2A-B
    If the_im_product_is_within_the_bandwidth_of_the_victim
      then see_if_it's_strong_enough_to_cause_interference
    If the_im_product_is_strong_enough_to_cause_interference
      then print_the_pertinent_intermodulation_information
  Get_the_next_B
Get_the_next_A
```

Figure IV-6. Pseudo Code For The Calculation Of 2A-B Intermodulation Products

frequencies. The first case is when the victim frequency, V, is lower than all three culprit frequencies, A, B, and C. In this case it will be shown that, for V=A+B-C to be possible, the C frequency must be the highest of the three culprit frequencies. But first, it should be noted that in all cases, the relative position of frequency A to frequency B is arbitrary. That is, assuming that the A frequency is less than the B frequency (A<B) or that B<A is arbitrary since the sum in either case is the same and greater than either A or B. However, since the C frequency is being subtracted, its position relative to the other culprit frequencies will be dictated by the relative position of V to the culprit frequencies and the constraint that V=A+B-C.

The Case With The Victim Less Than All Culprits

Returning now to the case where the victim frequency, V, is lower than all three of the culprit frequencies, A, B, and C. Assume that V<C<A<B. Again, the relative positions between A and B are arbitrary so V<C<B<A would be an equivalent assumption. Form the difference A-C. A-C will always be less than A but may be either greater than C or less than C. But A-C>C is impossible since adding any B to A-C would result in V=A+B-C>C which violates the initial assumption that V<C<A<B. Similar contradictions will be found for A-C<C. There are two cases that must be considered if A-C<C, either A-C is greater than V or A-C is

less than V. If A-C is greater than V (or even if A-C=V), adding any B to A-C will result in A+B-C also greater than V so that A+B-C=V is not realizable. If A-C is less than V, then adding any B, with B being greater than C by the initial assumption, results in V=A+B-C>A which violates the initial assumption. Therefore, under no circumstances may the conditions V=A+B-C and V<C<A<B be satisfied simultaneously.

Now assume that V<A<C<B and form the difference B-C. Since B is greater than C, B-C will be some positive value and adding any A will result in V=A+B-C being greater than A. This violates the initial assumption and therefore V=A+B-C and V<A<C<B can not be satisfied simultaneously. This leaves the case V<A<B<C. Assume that V<A<B<C and form the difference A-C. Since C is greater than A, A-C is some negative value. Adding any B, with B being less than C by the initial assumption, will result in A-C<V=A+B-C<A. Therefore, with the victim frequency being less than all three culprit frequencies, the only valid relationship among the frequencies is V<A<B<C.

<u>The Case With The Victim Greater Than One Culprit</u>

The second case is when the victim frequency, V, is greater than one of the culprit frequencies but less than the other two. In this case, there are three possible

relationships among the frequencies, A<V<B<C, C<V<A<B, and A<V<C<B.

Assume that A<V<B<C and form the difference B-C. Since C is greater than B, B-C is some negative value. Adding any A to B-C will result in V=A+B-C<A which violates the initial assumption. Therefore, A<V<B<C is an invalid assumption.

Assume that C<V<A<B and form the difference A-C. Since A is greater than C, A-C is some positive value which may be greater than, equal to, or less than V. If A-C is greater than or equal to V, then adding any B to A-C will result in a value greater than V. Therefore, V=A+B-C can not be realized. If A-C is less than V, adding any B will result in V=A+B-C being greater than B which violates the initial assumption.

Assume that A<V<C<B and form the difference B-C. Since B is greater than C, B-C will be some positive value. Adding any A would result in V=A+B-C>A which is the desired result. Therefore, the only valid relationship among the frequencies with the victim being greater than one of the culprits and less than the other two is A<V<C<B.

The Case With The Victim Greater Than Two Culprits

The third case is when the victim frequency, V, is

greater than two of the culprits but less than the remaining one. In this case, there are three possible relationships among the frequencies, A<B<V<C, C<A<V<B, and A<C<V<B.

Assume that A<B<V<C and form the difference A-C. Since C is greater than A, A-C is some negative number. Adding any B to A-C results in V=A+B-C<B which violates the initial assumption.

Assume that C<A<V<B and form the difference B-C. Since B is greater than C, B-C will be some positive value which may be greater than, equal to, or less than V. If B-C is greater than or equal to V, then adding any A to B-C will result in a value greater than V so that V=A+B-C can not be realized. If B-C is less than V, then adding any A to B-C, with A being greater than C by the initial assumption, will result in V=A+B-C>B. This violates the initial assumption and so C<A<V<B is an invalid assumption.

Assume that A<C<V<B and form the difference A-C. Since C is greater than A, A-C will be some negative value. Adding any B to A-C will result in A-C<V=A+B-C<B, which is the desired result. Therefore, for the case with the victim frequency greater than two of the culprit frequencies but less than the one remaining culprit frequency, the only valid relationship among the frequencies is A<C<V<B.

The Case With The Victim Greater Than All Culprits

The fourth and final case is when the victim frequency, V, is greater than all three culprit frequencies, A, B, and C. In this case, there are three possible relationships among the frequencies, A<B<C<V, B<C<A<V, and C<B<A<V.

Assume that A<B<C<V and form the difference B-C. Since C is greater than B, B-C is some negative value. If any A is added to B-C the result will be V=A+B-C<A which violates the initial assumption.

Assume that B<C<A<V and form the difference B-C. Since C is greater than B, B-C is some negative value. If any A is added to B-C the result will be V=A+B-C<A which violates the initial assumption.

Assume that C<B<A<V and form the difference B-C. Since B is greater than C, B-C will be some positive value. Adding any A to B-C will result in V=A+B-C>A, which is the desired result. Therefore, for the case with the victim frequency greater than all three of the culprit frequencies, the only valid relationship among the frequencies is C<B<A<V.

Pseudo code for the algorithm taking advantage of this reasoning is given in Figure IV-7.

```
CASE_1:
  For A = 1_freq_above_V to 2_freqs_below_the_highest_freq
    For B = 1_freq_above_A to 1_freq_below_the_highest_freq
      For C = 1_freq_above_B to the_highest_freq
        Calculate_the_im_product_and_check_for_interference
      Get_the_next_C
    Get_the_next_B
  Get_the_next_A

CASE_2:
  For A = f1 to 1_freq_below_V
    For C = 1_freq_above_V to 1_freq_below_the_highest_freq
      For B = 1_freq_above_C to the_highest_freq
        Calculate_the_im_product_and_check_for_interference
      Get_the_next_B
    Get_the_next_C
  Get_the_next_A

CASE_3:
  For A = f1 to 2_freqs_below_V
    For C = 1_freq_above_A to 1_freq_below_V
      For B = 1_freq_above_V to the_highest_freq
        Calculate_the_im_product_and_check_for_interference
      Get_the_next_B
    Get_the_next_C
  Get_the_next_A

CASE_4:
  For C = f1 to 3_freqs_below_V
    For B = 1_freq_above_C to 2_freqs_below_V
      For A = 1_freq_above_B to 1_freq_below_V
        Calculate_the_im_product_and_check_for_interference
      Get_the_next_A
    Get_the_next_B
  Get_the_next_C
```

Figure IV-7. Pseudo Code For The Calculation Of A+B-C Intermodulation Products

The 3A-2B Intermodulation Products

There are no positional relationships among the frequencies involved in the calculation of 3A-2B intermodulation products. The only requirement is, as was the case in all the algorithms, that all the frequencies be unique. Therefore, the algorithm for this calculation is a simple set of nested loops with the appropriate tests for uniqueness within the loops. The algorithm is as follows and a pseudo code implementation is given in Figure IV-8.

```
For A = f1 to the_highest_frequency
  If A is_not_equal_to the_victim then
    For B = f1 to the_highest_frequency
      If B is_not_equal_to the_victim or A then
        the_im_product = 3A-2B
        Is_the_im_product_within_the_passband_of_V
        If yes then
          Is_it_strong_enough_to_interfere
          If yes then
            print_the_intermodulation_information
    Get_the_next_B
Get_the_next_A
```

Figure IV-8. Pseudo Code For The Calculation Of 3A-2B Intermodulation Products

(1) Select a frequency for A beginning with f1. Each time a new A is needed select the next higher frequency until the highest frequency is reached but do not use

the victim frequency.

(2) Select a frequency for B beginning with f1. Each time a new B is needed select the next higher frequency until the highest frequency is reached but do not use the victim frequency or A.

(3) Calculate the intermodulation product. If three times A minus two times B is within the bandpass of the victim frequency then the possibility for interference exists and further processing is necessary. Otherwise continue with the next step.

(4) Repeat steps 2 and 3 until all eligible frequencies have been selected for B.

(5) Repeat steps 1 through 4 until all eligible frequencies have been selected for A. Once this has been accomplished, all possible interference to the victim frequency from 3A-2B intermodulation products will have been found.

Test The Intermodulation Product

Besides the calculations necessary to find intermodulation products that fall within the passband of the victim receiver, additional calculations must be made to determine if the strength of the intermodulation product is sufficient to cause interference to the victim receiver. Before discussing the algorithm which uses these calculations, a description of the calculations themselves

is given. Each of the calculations is implemented in the form of a Pascal function which is then available for use by any of the intermodulation product testing routines in the program. The implementation details are not discussed. These are readily available in the program listing contained in Appendix B.

Victim Receiver Thermal Noise

To cause interference, the power in the intermodulation product must exceed the thermal noise level at the front end of the victim receiver. Therefore, the thermal noise level at the front end of the victim receiver acts as the minimum level that the intermodulation product must attain to cause interference. This is not to say that interference will occur at this level but that below this level interference will not occur. The formula for calculating the thermal noise in any bandwidth was given in Chapter III and is repeated here.

$$N_t = 10 \text{ Log}(B) - 144 \qquad (4.1)$$

where

N_t is the thermal noise, in dBm, at the front end of victim receiver,

B is the victim frequency's bandwidth in kHz, and

Log() is the logarithm, base 10, of the expression inside the parentheses.

The factor 144 takes into consideration Boltzman's constant, the assumption of operation at 290 degrees Kelvin, and conversion factors needed to express the thermal noise in dBm. Since the victim receiver's thermal noise is not dependent on the intermodulation product being investigated, it need be calculated only once for each victim frequency and then used in testing for all intermodulation products. The program obtains the necessary bandwidth from the frequency data.

Antenna Separation Distances

The strength of the intermodulation product at the front end of the victim receiver is inversely proportional to the square of the distances between the culprit and threat antennas and the distance between the threat and victim antennas. All the information needed to calculate these distances is contained in the antenna data which is readily available to the program. The program references the antenna data to obtain the needed information and then calculates the distance between a given antenna pair using

$$D = \mathrm{Sqr}[\ (x1-x2)^2 + (y1-y2)^2 + (h1-h2)^2\ ] \qquad (4.2)$$

where

D is the distance, in feet, between the antennas,

x1 is the x-coordinate, in feet, of the first antenna,

x2 is the x-coordinate, in feet, of the second antenna,

y1 is the y-coordinate, in feet, of the first antenna,

y2 is the y-coordinate, in feet, of the second antenna,

h1 is the height, in feet above sea level, of the first antenna,

h2 is the height, in feet above sea level, of the second antenna, and

Sqr[] represents the square root of the expression between the brackets.

Free Space Propagation Loss

The reason that the strength of the intermodulation product at the front end of the victim receiver is inversely proportional to the square of the distances between the culprit and threat antennas and the distance between the threat and victim antennas is that the signals propagating these distances experience free space propagation loss. Free space propagation loss is directly proportional to the squares of the distance and the frequency involved. The program calculates the free space propagation loss for a given frequency by first calculating the appropriate distance as discussed above and then applying the formula

$$L_p = -37.85 + 20\text{Log}(DF) \qquad (4.3)$$

where

L_p is the free space propagation loss in dB,

D is the distance, in feet, between the two antennas concerned,

F is the frequency, in MHz, of concern, and

Log() is the logarithm, base 10, of the expression in the parenthesis.

Culprit Power

The power level of the intermodulation product at the front end of the victim receiver is directly proportional to the power level of the culprit signal at the antenna port of the threat transmitter. The formula for calculating the culprit power was given in Chapter III and is repeated here.

$$P_c = P_{co} - L_{cc} + G_c - L_p + G_t - L_{tc} \qquad (4.4)$$

where

P_c is the power level, in dBm, of the culprit signal at the threat transmitter's antenna port,

P_{co} is the output power, in dBm, of the culprit transmitter,

L_{cc} is the coupling loss, in dB, between the culprit transmitter and the culprit antenna,

G_c is the gain, in dB, of the culprit antenna in the direction of the threat antenna,

L_p is the free space propagation loss, in dB, between the culprit antenna and the threat antenna,

G_t is the gain, in dB, of the threat antenna in the

direction of the threat antenna, and

L_{tc} is the coupling loss, in dB, between the threat antenna and the threat transmitter.

The program obtains the output power of the culprit transmitter, coupling losses, and antenna gains from the antenna and frequency data and calculates the free space loss as described above. It then applies equation 4.4 to calculate the culprit power at the antenna port of the threat transmitter.

Selectivity

The level of the intermodulation product at the front end of the victim receiver is inversely proportional to the selectivity of the threat transmitter to the intermodulation product and culprit frequencies. In Chapter III, Figure III-3 provided conservative estimates of transmitter output selectivity in the form of a graph. The graph is a plot of 10*Log(|f-f0|/f0) vs. selectivity where f0 is the tuned frequency of the transmitter and f is the off tune frequency of interest. To provide the same information for use by the program, piecewise linear approximations of the curves on the graph were developed. If we let delta be 10*Log(|f-f0|/f0), then for the case of a transmitter tuned to a frequency of less than 100 MHz the piecewise linear approximation for the selectivity, S(f) in dB, is

$$S(f) = \begin{cases} 0 & ; \quad \text{delta} \le -16.9897 \\ 1.2922*\text{delta}+21.9456 & ; \quad -16.9897 < \text{delta} \le -13.9794 \\ 1.4197*\text{delta}+23.7368 & ; \quad -13.9794 < \text{delta} \le -12.2185 \\ 1.6272*\text{delta}+26.2733 & ; \quad -12.2185 < \text{delta} \le -10.0000 \\ 1.9367*\text{delta}+29.3668 & ; \quad -10.0000 < \text{delta} \le -6.9897 \\ 2.1074*\text{delta}+30.5600 & ; \quad -6.9897 < \text{delta} \end{cases}$$

(4.5)

For a transmitter tuned to a frequency of 100 MHz or higher the piecewise linear approximation for the selectivity, S(f) in dB, is

$$S(f) = \begin{cases} 0 & ; \quad \text{delta} \le -15.2288 \\ 0.5852*\text{delta}+8.9122 & ; \quad -15.2288 < \text{delta} \le -10.0000 \\ 1.1062*\text{delta}+14.1220 & ; \quad -10.0000 < \text{delta} \le -6.9897 \\ 1.9210*\text{delta}+19.8172 & ; \quad -6.9897 < \text{delta} \end{cases}$$

(4.6)

The program calculates delta first and then applies either equation 4.5 or 4.6 as appropriate.

Transmit Intermodulation Power

The power in the intermodulation product at the front end of the victim receiver is directly proportional to the power in the intermodulation product at the antenna port of the threat transmitter. The formula for calculating the

power of the intermodulation product at the antenna port of the threat transmitter was given in Chapter III and is repeated here.

$$P_{IM} = n(P_{c1} - S_t(F_{c1})) + m(P_{c2} - S_t(F_{c2})) - (n + m - 1)b - K_1 - S_t(F_{IM}) \quad (4.7)$$

where

P_{IM} is the power level, in dBm, of the intermodulation product,

n and m are the harmonic numbers of culprit signal number one and culprit signal number two respectively,

the subscripts c1 and c2 are used to relate the associated terms with culprit signal number one and culprit signal number two respectively,

P_c is the power level, in dBm, of the culprit signal at the threat transmitter's antenna port,

S_t (f) is the selectivity, in dB, of the threat transmitter's output stage to f,

b is 30 dB if the threat transmitter's final power amplifier is a solid state type and 40 db if the threat transmitter's final power amplifier is a tube type, and

K_1 is the threat transmitter's intermodulation coefficient, in dB.

The program obtains the b and K_1 terms from the frequency data but must calculate all the remaining terms, as described above, before it can apply equation 4.7.

Received Intermodulation Product Power

If the intermodulation product power exceeds the thermal noise power at the front end of the victim receiver then the intermodulation product is considered a threat to communications and the frequencies, antennas, and intermodulation product type will be printed. The formula for calculating the power level of the intermodulation product at the front end of the victim receiver was given in Chapter III and is repeated below.

$$P_{RIM} = P_{IM} - L_{tc} + G_t - L_p + G_v - L_{vc} \qquad (4.8)$$

where

P_{RIM} is the power, in dBm, of the intermodulation product at the front end of the victim receiver,

P_{IM} is the power level, in dBm, of the intermodulation product at the output of the threat transmitter,

L_{tc} is the coupling loss, in dB, between the threat transmitter and its associated antenna,

G_t is the gain, in dB, of the threat antenna in the direction of the victim receiver's antenna,

L_p is the propagation loss, in dB, between the threat and victim antennas,

G_v is the gain, in dB, of the victim receiver's antenna in the direction of the threat antenna, and

L_{vc} is the coupling loss, in dB, between the victim receiver and its associated antenna.

The program obtains the coupling losses and antenna gains from the antenna and frequency data but must calculate the propagation loss and power of the intermodulation product at the antenna port of the threat antenna, as discussed above, before it can apply equation 4.8.

The Intermodulation Product Testing Algorithm

Now that the necessary equations have been discussed, the algorithm using these equations to determine if an intermodulation product should be considered a threat to communications or not is trivial. The algorithm assumes that an intermodulation product has been found that falls within the passband of the victim receiver and that the noise level at the front end of the victim receiver has already been calculated. It then selects one of the culprits to be analyzed as the threat and calculates the power in the intermodulation product at the front end of the victim receiver. If the intermodulation product level exceeds the thermal noise level then information identifying the particular frequencies, antennas, and intermodulation type are printed. The procedure is then repeated selecting

each of the culprits to act as the threat. Of course, only two or three repetitions are needed because only two and three signal intermodulation products are being calculated. Pseudo code for this algorithm is given in Figure IV-9 for the two signal case. The changes needed to extend the code for the three signal case should be readily apparent.

```
The_threat = A
The_culprit = B
If The_intermodulation_product_power > the_thermal_noise
  then print_identifying_information
The_threat = B
The_culprit = A
If The_intermodulation_product_power > the_thermal_noise
  then print_identifying_information
```

Figure IV-9. Pseudo Code For Testing A Two Signal Intermodulation Product

Of more interest is the sequence of calculations necessary to find the power at the front end of the victim receiver. The equations for doing so have already been discussed and the sequence for the three signal case follows. For the two signal case, calculations are needed for only one culprit.

(1) Calculate the distance between the first culprit and the threat antennas.

(2) Calculate the free space propagation loss of the first culprit frequency between its antenna and the threat's antenna.

(3) Calculate the power in the first culprit frequency at the antenna port of the threat transmitter.

(4) Calculate the distance between the second culprit and the threat antennas.

(5) Calculate the free space propagation loss of the second culprit frequency between its antenna and the threat antenna.

(6) Calculate the power in the second culprit frequency at the antenna port of the threat transmitter.

(7) Calculate the selectivity of the threat transmitter to the first culprit frequency.

(8) Calculate the selectivity of the threat transmitter to the second culprit frequency.

(9) Calculate the selectivity of the threat transmitter to the intermodulation product.

(10) Calculate the power in the intermodulation product at the antenna port of the threat transmitter.

(11) Calculate the distance between the threat and victim antennas.

(12) Calculate the propagation loss of the intermodulation product between the threat and victim antennas.

(13) Calculate the power in the intermodulation product at the front end of the victim receiver.

Once the strength of the intermodulation product has been checked, a new intermodulation product may be calculated. This procedure is continued until all possible intermodulation products falling on a particular victim frequency have been tested. Then a new victim frequency is selected and the entire process repeated until all victim frequencies have been analyzed for interference.

Review The Results And Update The Data

If the analysis indicates possible interference, corrective measures may be taken immediately or onsite testing may be accomplished to determine if actual interference will occur. The program provides assistance in both cases. The results of the analysis are presented in a form similar to Figure IV-10. The frequencies involved are noted along with the antennas they appear on. Which frequency is considered the victim is also indicated along with the amount, in dB, that the intermodulation product level exceeded the thermal noise level at the victim receiver's front end.

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Testing for interference to  40.000 MHz on antenna #4
19 May 1986                                              04:52

      100.000 MHz on antenna # 10 the threat
    -  60.000 MHz on antenna # 6
-------------
       40.000 MHz on antenna # 4
Noise level exceeded by   7.53 dB
```

Figure IV-10. The Format Of An Analysis Output.

Two of the potentially least costly corrective measures are to assign new frequencies or to assign the present frequencies to different antennas. If the assignment of present frequencies to new antennas is chosen, the separation distance needed between the threat and culprit antennas or the threat and victim antennas is indicated by the amount that the intermodulation product level exceeded the thermal noise level. Any changes to the antenna data or frequency data are easily accomplished with the help of the program's screen oriented editing capability. If a frequency is changed it will be placed in its proper position among the other frequencies automatically. To reassign a frequency to a new antenna, all that need be done is change the frequency's associated antenna number. However, the program does not presently support the capability of identifying frequency assignments which would

not create interference if added to the present list.

If onsite testing is chosen, only the frequency/antenna combinations indicated in the analysis need be tested. Since worst case conditions are assumed in the analysis, it can be expected that some of the possible interference indicated in the analysis will not result in actual interference. The amount that the thermal noise level was exceeded by may be used as an indication of whether actual interference will occur or not. The greater this value, the more likely actual interference will occur.

Repeat The Analysis Procedure

Any changes to the antenna or frequency data require a reanalysis since such changes represent a change to the electromagnetic environment which can not be accomplished in isolation. That is, a change which corrects a problem in one area may create a problem in another area. Although time consuming, a reanalysis is also effortless since the analysis portion of the program runs without operator attention once started.

V. COMMENTS AND RECOMMENDATIONS

This thesis deals strictly with intermodulation products created in the output stage of a transmitter. A closely related problem is intermodulation products created in the front end of a receiver. Most of the work accomplished in this thesis may be applied directly to the analysis of receiver intermodulation products and therefore, including the capability to perform such an analysis would be a logical extension to this thesis. Although there is little information published on transmitter produced intermodulation products, the literature, including most of the references used in this thesis, deals extensively with receiver intermodulation products. The effort then would be concentrated on condensing the available information into a set of generalized criteria which could be put into a form suitable for implementation in the analysis program.

Significant improvement in the validity of the analysis would be made if antenna directionality were taken into account. This would involve, as a minimum, including the capability of entering azimuth and elevation information with each antenna to indicate the axis of its main beam. An antenna pattern could then be calculated using the antenna's gain, azimuth, and elevation for use in determining the gain of the antenna in any given direction. This would tend to

decrease the number of intermodulation products flagged as possible causes of interference and make the analysis a closer model of the actual electromagnetic environment. Further improvement could be realized by taking antenna polarization into account.

The program could be extended to identify frequency assignment conflicts, include an harmonic interference analysis, and to suggest corrective actions for problems found. This would slow down the operation of the program but increase it's utility.

It may be possible to increase the speed of the intermodulation product calculating algorithms. For example, in the algorithm for calculating the A+B intermodulation products, there is no need to select the next higher B frequency for testing once A+B is greater than the victim frequency. The loop in which B is selected could be exited as soon as A+B exceeded the victim frequency instead of waiting until the frequency just below the victim frequency is selected for B. Any benefit in execution time gained by exiting the loop early would have to be compared to the execution time needed to test for the condition to determine if any overall speed increase is realized.

The program's requirement of assigning sequential

antenna numbers should be replaced with user supplied antenna identifiers. This would require the user to enter an identifier for each antenna but would eliminate the problem of renumbering the site map when an antenna is deleted.

A possible follow on to this thesis would be experiments to establish a correlation between actual intermodulation product levels and those predicted by this program. One area of particular concern would be the relationship between the threat transmitter's output power and the strength of the intermodulation product. The equation developed by Maiuzzo for the intermodulation product power at the threat transmitter's antenna port is independent of the threat transmitter's output power. A second area of concern is Maiuzzo's b factor (see equation 3.8) which was obtained from experimental data on equipment operating in the HF (3-30 MHz) band. Experiments are needed to develop a b factor for equipment operating in other bands, and the VHF (30-300 MHz) band in particular. And finally, the program was developed for use in the VHF band but should provide valid results over a much broader range. Experiments or further research could be used to determine frequency limits for the program.

APPENDIX A

A USER'S MANUAL

FOR

TIMAP VERSION 1.00

THE

TRANSMITTER INTERMODULATION

ANALYSIS PROGRAM

BY

THOMAS J. ZUZACK

TABLE OF CONTENTS

	Page
List Of Figures	iv
Introduction	A-1
Requirements	A-2
General Description	A-3
Some Background And Terminology	A-4
What TIMAP Does	A-6
Before Getting Started	A-14
Getting Started	A-14
How To Make TIMAP Work	A-15
Main Menu - Item 1 - Add data	A-16
Add Menu - Item 1 - Add antenna data	A-17
Add Menu - Item 2 - Add frequency data	A-21
Add Menu - Item 3 - Clear current data	A-25
Add Menu - Item 4 - Return to MAIN MENU	A-26
Main Menu - Item 2 - Print current data	A-26
Main Menu - Item 3 - Display the current data	A-27
Display Menu - Item 1 - Display antenna data	A-27
Display Menu - Item 2 - Display frequency data	A-28
Display Menu - Item 3 - Return to MAIN MNU	A-28
Main Menu - Item 4 - Change or delete data	A-28
Edit Menu - Item 1 - Edit antenna information	A-29
Edit Menu - Item 2 - Edit frequency information	A-31
Edit Menu - Item 3 - Return to MAIN MENU	A-33

Page

Main Menu - Item 5 - Save the data to disk A-33

Main Menu - Item 6 - Load data from disk A-34

Main Menu - Item 7 - Perform an analysis A-34

Main Menu - Item 8 - Set Options A-35

Options Menu - Item 1 - Specify amplifier type . A-36

Options Menu - Item 2 - Specify IM coefficient . A-37

Options Menu - Item 3 - Set path name A-38

Options Menu - Item 4 - Return to MAIN MENU . . A-38

Main Menu - Item 9 - Exit to the system A-39

Comments And Warnings A-40

LIST OF FIGURES

Figure		Page
A-1.	TIMAP's MAIN MENU	A-16
A-2.	TIMAP's ADD MENU	A-17
A-3.	An Antenna Layout Prepared For Program Input .	A-20
A-4.	TIMAP's DISPLAY MENU	A-27
A-5.	TIMAP'S EDIT MENU	A-29
A-6.	TIMAP'S OPTIONS MENU	A-36

I. Introduction

The colocation of VHF communications equipment gives rise to the possibility of disrupted or degraded service due to many types of interference. Among these are receiver adjacent channel interference, receiver spurious responses, and receiver intermodulation, all of which occur within the receiver itself. Transmitters may also create interference due to transmitter broad band noise, spurious emissions, and transmitter intermodulation. Still another type of interference, often referred to as "rusty bolt intermodulation", may occur in neither the transmitter nor receiver. The analysis of a colocated site for electromagnetic compatibility (EMC) must take into consideration all of these interference types and is therefore a complex and time consuming process. To assist the EMC engineer, this program was developed to perform an automated interference analysis of transmitter produced intermodulation products in colocated VHF sites using the Zenith Z-100 computer. For want of a better name, I call the program TIMAP, which stands for Transmitter InterModulation Analysis Program.

Since TIMAP was developed for use by EMC engineers, an indepth discussion of the mechanisms and theory behind transmitter produced intermodulation interference will not

be given in this user's manual. I will assume you have a basic understanding of the interference mechanisms involved and of the Z-100 computer and its operation. However, I will discuss some important terminology and the assumptions and equations used in TIMAP so that you may gain an understanding of the calculations performed and have confidence in the program's results. But first I'd better cover what you'll need to make TIMAP run.

Requirements

I developed TIMAP on a Heathkit H-120 All-In-One computer running Version 2.22 of the MS-DOS operating system. My developmental computer has 768k bytes of user memory, the full 192k bytes of video memory, two 5.25" 360k byte floppy disk drives, and an 8 MHz system clock. To run the program you'll need:

* A Heathkit H-100 series or Zenith Z-100 series computer with a minimum of 128k bytes of user memory, 32k bytes of video memory, and one 5.25" 360k byte floppy disk drive. (The program has not been tested on a computer with this minimum configuration.)
* The MS-DOS operating system, Version 2.xx.
* A 5.25" floppy disk containing the files TIMAP.COM and

To save time and paper and permit reformatting of the information to suit the user's needs, all of TIMAP's outputs are directed to a disk file in a text format. This speeds up operation of the program since the calculations will not be delayed while the data is being printed and also allows the user to edit the information with the text processor of his choice. Therefore, a text processor and printer would also enhance the utility of TIMAP.

General Description

TIMAP was written in Turbo Pascal, Version 3.01A. It is entirely menu and prompt driven, which means to operate TIMAP you select options from a list of options (a menu) or answer questions (prompts) posed on the screen by typing on the keyboard. TIMAP allows you to enter antenna and frequency data describing a colocated VHF site and then analyzes that data to determine if any transmitter intermodulation products are strong enough to represent a threat to communications. The results of a TIMAP analysis are based on worst case conditions. Because of this, the results are more valid in determining that interference will not occur than in determining if interference will occur.

Some Background And Terminology

Transmitter intermodulation products are created when the signals from one or more "culprit" transmitters are coupled into the final output stage of another "threat" transmitter via the threat transmitter's antenna. Because of the nonlinear operation of the threat transmitter's final output stage, the threat and culprit signals mix to create new signals at frequencies equal to the sums and differences of integer multiples of the original signal frequencies. These new signals are called transmitter intermodulation products. If a transmitter intermodulation product falls within the bandwidth of an allotted receive frequency and is of sufficient strength, it will interfere with the reception of the desired signal. Some terminology associated with transmitter intermodulation products will now be defined.

Let A, B, and C represent the fundamental (culprit) frequencies from three culprit transmitters. Any one of the culprit transmitters may act as the threat transmitter if that is where the mixing takes place. In other words, the term culprit is used in association with any of the signals that mix to create a transmitter intermodulation product. Culprit transmitters and antennas are those used with a culprit frequency. The term threat is used in association with the transmitter in which the mixing takes place. The

threat transmitter's desired or tuned frequency is then the threat frequency and the threat transmitter's antenna is the treat antenna.

When an intermodulation product threatens the reception of an allotted receive frequency, the threatened frequency is called the victim frequency. Of course, the term victim will then be used in association with the victim frequency so that the receiver tuned to receive the victim frequency is called the victim receiver and the victim receiver's antenna is called the victim antenna.

Another term associated with intermodulation products is their order. The order of an intermodulation product is defined as the sum of the harmonic numbers of the culprit and threat frequencies. For example, the second harmonic of A may be represented by 2A and the fundamental or first harmonic of B is B. If an intermodulation product is created with a frequency equal to 2A-B, then that intermodulation product would be called a third order intermodulation product.

What TIMAP Does

TIMAP considers only two or three signal intermodulation products. This means that the products considered by TIMAP are those from the mix of either two or three signals only.

TIMAP calculates second order intermodulation products of the type A+B and A-B, third order intermodulation products of the type 2A-B and A+B-C, and fifth order intermodulation products of the type 3A-2B. Any transmitter intermodulation products that fall within the passband of an allotted receive frequency are then evaluated to see if they have sufficient strength to cause interference. What constitutes "sufficient strength" would, in general, vary with each piece of equipment and its application but, to keep things manageable, TIMAP considers any intermodulation product that exceeds the thermal noise present at the front end of the victim receiver to be of sufficient strength. To calculate the thermal noise, TIMAP uses the equation

$$N_t = 10 \text{ Log}(B) - 144 \qquad (A.1)$$

where

N_t is the thermal noise, in dBm, at the front end of victim receiver,

B is the victim frequency's bandwidth in kHz, and

Log() is the logarithm, base 10, of the expression inside the parentheses.

The factor 144 takes into consideration Boltzman's constant, a simplifying assumption of operation at 290 degrees Kelvin, and conversion factors needed to express the thermal noise in dBm.

Once TIMAP knows what level the intermodulation product must exceed, it then calculates the power in the intermodulation product at the front end of the victim receiver. Since this power will vary depending on which transmitter is considered the threat, TIMAP considers each culprit transmitter in turn as the threat. In each case the same type of calculations are necessary and are accomplished in a sequence as follows.

(1) TIMAP calculates the distances between the culprit and threat antennas. For a two signal intermodulation product there will only be one culprit and one threat so only one distance will be calculated. But for a three signal intermodulation product there will be two culprits and one threat so two distances will be calculated, one between the first culprit and the threat antennas and the other between the second culprit and threat antennas. The equation used to calculate these distances is

$$D = \mathrm{Sqr}[\ (x1-x2)^2 + (y1-y2)^2 + (h1-h2)^2\ ] \qquad (A.2)$$

where

D is the distance, in feet, between the antennas,

x1 is the x-coordinate, in feet, of the first antenna,

x2 is the x-coordinate, in feet, of the second antenna,

y1 is the y-coordinate, in feet, of the first antenna,

y2 is the y-coordinate, in feet, of the second antenna,

h1 is the height, in feet above sea level, of the first antenna,

h2 is the height, in feet above sea level, of the second antenna, and

Sqr[] represents the square root of the expression between the brackets.

(2) TIMAP calculates the free space loss experienced by the culprit frequencies between their antennas and the threat antenna. Again, for the two signal case there will only be one free space loss calculation but for the three signal case there will be two free space loss calculations. The equation used by TIMAP is

$$L_p = -37.85 + 20\mathrm{Log}(DF) \qquad (A.3)$$

where

L_p is the free space propagation loss in dB,

D is the distance, in feet, between the two antennas concerned,

F is the frequency, in MHz, of concern, and

Log() is the logarithm, base 10, of the expression in the parenthesis.

(3) TIMAP then calculates the power in the culprit frequencies at the antenna port of the threat transmitter. One calculation is needed for each culprit frequency. To make these calculations, TIMAP makes a not very valid but simplifying assumption that the gains of the antennas involved are valid in all directions and for all frequencies. The equation used is

$$P_c = P_{co} - L_{cc} + G_c - L_p + G_t - L_{tc} \qquad (A.4)$$

where

P_c is the power level, in dBm, of the culprit signal at the threat transmitter's antenna port,

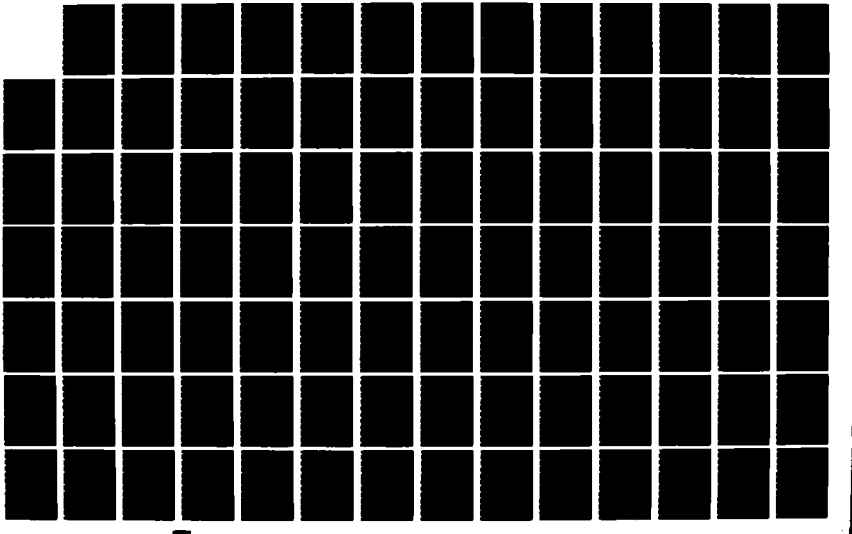
P_{co} is the output power, in dBm, of the culprit transmitter,

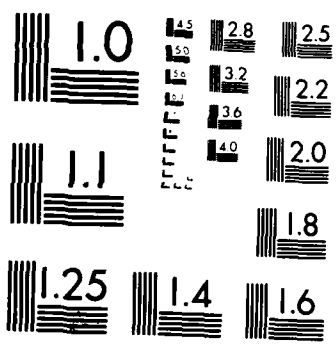
L_{cc} is the coupling loss, in dB, between the culprit transmitter and the culprit antenna,

G_c is the gain, in dB, of the culprit antenna in the direction of the threat antenna,

L_p is the free space propagation loss, in dB, between the culprit antenna and the threat antenna,

G_t is the gain, in dB, of the threat antenna in the direction of the threat antenna, and





L_{tc} is the coupling loss, in dB, between the threat antenna and the threat transmitter.

(4) Next, TIMAP calculates the selectivity of the threat transmitter to each of the culprit frequencies and to the intermodulation product. Selectivity is the amount of attenuation an "off tune" signal would experience when passing through the circuitry of a transmitter as compared to the "on tune" signal. Selectivity varies with each transmitter but conservative estimates are made by applying the following two equations. Let f be the off tune signal of interest and f0 be the threat transmitter's operating frequency. If we let delta be 10*Log(|f-f0|/f0), then for the case of a transmitter tuned to a frequency of less than 100 MHz, the selectivity, S(f) in dB, may be approximated by

$$S(f) = \begin{cases} 0 & ; \quad \text{delta} \le -16.9897 \\ 1.2922*\text{delta}+21.9456 & ; \quad -16.9897 < \text{delta} \le -13.9794 \\ 1.4197*\text{delta}+23.7368 & ; \quad -13.9794 < \text{delta} \le -12.2185 \\ 1.6272*\text{delta}+26.2733 & ; \quad -12.2185 < \text{delta} \le -10.0000 \\ 1.9367*\text{delta}+29.3668 & ; \quad -10.0000 < \text{delta} \le -6.9897 \\ 2.1074*\text{delta}+30.5600 & ; \quad -6.9897 < \text{delta} \end{cases}$$

(A.5)

For a transmitter tuned to a frequency of 100 MHz or higher, the selectivity, S(f) in dB, may be approximated by

$$S(f) = \begin{cases} 0 & ; \quad \text{delta} \le -15.2288 \\ 0.5852*\text{delta}+8.9122 & ; \quad -15.2288 < \text{delta} \le -10.0000 \\ 1.1062*\text{delta}+14.1220 & ; \quad -10.0000 < \text{delta} \le -6.9897 \\ 1.9210*\text{delta}+19.8172 & ; \quad -6.9897 < \text{delta} \end{cases}$$

(A.6)

(5) Now TIMAP can calculate the power in the intermodulation product at the antenna port of the threat transmitter. The equation for calculating this power is an extension of the equation developed by Maiuzzo in a paper titled "Transmitter Intermodulation Product Amplitudes", which may be found in the 1981 IEEE International Symposium On Electromagnetic Compatibility, pages 133 through 138. The equation used by TIMAP is

$$P_{IM} = n(P_{c1} - S_t(F_{c1})) + m(P_{c2} - S_t(F_{c2})) - (n + m - 1)b - K_1 - S_t(F_{IM}) \quad \text{(A.7)}$$

where

P_{IM} is the power level, in dBm, of the intermodulation product,

n and m are the harmonic numbers of culprit signal number one and culprit signal number two respectively (if there is only one culprit then m would be zero),

the subscripts c1 and c2 are used to relate the associated terms with culprit signal number one

and culprit signal number two respectively,

P_c is the power level, in dBm, of the culprit signal at the threat transmitter's antenna port,

S_t (f) is the selectivity, in dB, of the threat transmitter's output stage to f,

b is 30 dB if the threat transmitter's final power amplifier is a solid state type and 40 db if the threat transmitter's final power amplifier is a tube type, and

K_1 is the threat transmitter's intermodulation coefficient, in dB, which may be assumed to be zero for a worst case analysis or may be obtained for each transmitter model from the Electromagnetic Compatibility Analysis Center, Annapolis, Maryland 21402.

(6) Finally TIMAP is ready to calculate the power in the transmitter intermodulation product that appears at the front end of the victim receiver. Again for this calculation, TIMAP assumes that the antenna gains are valid for all frequencies and in all directions. The equation used is

$$P_{RIM} = P_{IM} - L_{tc} + G_t - L_p + G_v - L_{vc} \qquad (A.8)$$

where

P_{RIM} is the power, in dBm, of the intermodulation

product at the front end of the victim receiver,

P_{IM} is the power level, in dBm, of the intermodulation product at the output of the threat transmitter,

L_{tc} is the coupling loss, in dB, between the threat transmitter and its associated antenna,

G_t is the gain, in dB, of the threat antenna in the direction of the victim receiver's antenna,

L_p is the propagation loss, in dB, between the threat and victim antennas,

G_v is the gain, in dB, of the victim receiver's antenna in the direction of the threat antenna, and

L_{vc} is the coupling loss, in dB, between the victim receiver and its associated antenna.

As you may have noticed, *TIMAP calculates many of the* values needed to determine the level of the transmitter intermodulation product at the front end of the victim receiver but that other values were assumed to be known. For example, to calculate the separation distances between antennas, the antenna locations and heights were needed. This is the information you must enter before running an analysis, which bring us to the point of starting the program, but . . .

Before Getting Started

As with any software you own, you should make a working copy of TIMAP and store the original away in a safe place. Any software not backed up will eventually be lost. To make a working copy

(1) Boot your system following the instructions given in your operating system manual.

(2) Format a blank disk. If you want to make the working disk bootable, format with the FORMAT/S command. I invariably format using a 9 sector format with the FORMAT/9 command because it results in more storage space on the disk than the standard FORMAT command. Again, the instructions for formatting a disk are in your operating system manual. Label this disk TIMAP (Working Copy).

(3) Copy the files TIMAP.COM and TIMAP.000 from your original TIMAP disk to your working copy disk.

(4) Put your original disk away in a safe place.

Getting Started

To use TIMAP you must first boot your system. TIMAP works with MS-DOS Version 2.xx for the H/Z-100 computers and

the instructions for booting are contained in the manual that came with your operating system. Place your working copy of TIMAP in the default drive and type TIMAP <RETURN>. **Note: <RETURN> means press the key labeled RETURN. You may also use the <ENTER> key for a return.** The disk drive will run for awhile and then TIMAP will display its MAIN MENU. At this point you're up and ready to go.

How To Make TIMAP Work

This section of the manual explains how to use TIMAP. As mentioned earlier, TIMAP is menu driven, which means that to make it do something you select functions or items from a menu. You may select an item in one of two ways.

The first way: you may press the number corresponding to the item in the menu. For this you may use the number keys along the top of your keyboard or in the numeric keypad on your right. You will not need to press return if you select an item in this way.

The second way: you may use the cursor keys (arrow keys) in the numeric keypad to highlight the item number in the menu and then press <RETURN>.

Either method will get you the same result. The rest of this section is arranged by menu items beginning with the

MAIN MENU (See Figure A-1).

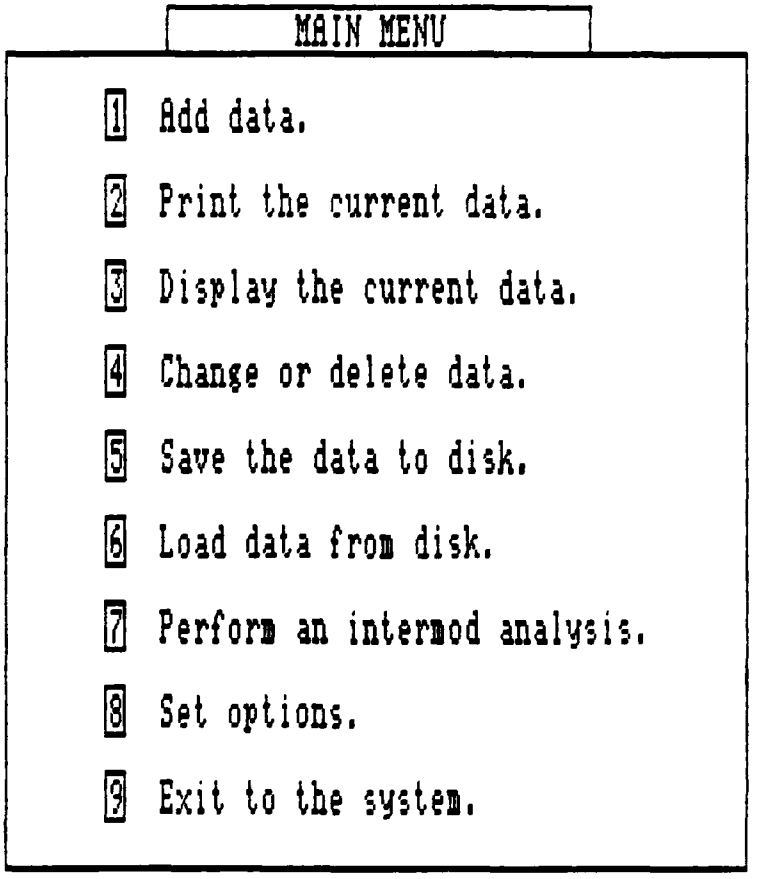


Figure A-1. TIMAP's MAIN MENU.

Main Menu - Item 1 - Add data.

Select this item to enter the data to be analyzed. You may either add new data to an existing data file or start a new data file. When you select this item, the ADD MENU will

be displayed. The specific data needed to run an analysis is described in conjunction with the ADD MENU (See Figure A-2), which is discussed next.

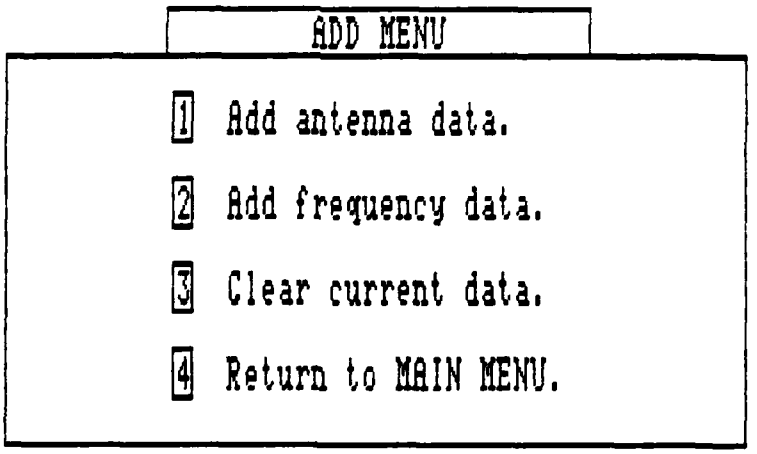


Figure A-2. TIMAP's ADD MENU.

Add Menu - Item 1 - Add antenna data.

Select this item to enter data describing the antennas in the colocated site. Antenna data is gathered separately from the frequency data to ease input into the program. Several frequencies may be used on a single antenna so instead of entering all necessary antenna information with each frequency, only an associated antenna number will need to be entered with each frequency. This reduces the amount of information that must be entered into the program and

also reduces the amount of memory necessary to store the information.

When you select this item the screen will clear and you will be prompted for the antenna information beginning with the x-coordinate of the antenna. If you entered this function by mistake, don't worry. Just press <RETURN> without entering an x-coordinate and you will be returned to the ADD MENU. Otherwise, enter the data as asked for. You may backspace over typing errors upon entry as long as you have not pressed the <RETURN> key. If you do make entry errors and don't notice them until after you've pressed <RETURN>, don't panic. You'll be able to to correct them using item 4 on the MAIN MENU, Change or delete data. When you've finished entering your antenna data, you may return to the ADD MENU by pressing <RETURN> when prompted for the next x-coordinate. The information needed on each antenna is:

* an antenna number assigned sequentially,
* the antenna location given as x,y coordinates in feet,
* the antenna height in feet above sea level, and
* the antenna gain in dBi.

Details on the antenna information are given below.

The strength of an intermodulation product is inversely proportional to the distances between the antennas. To calculate these distances the program will need the antenna locations and heights expressed in feet. Antenna locations may be specified by obtaining or making a scale map of the site with all antenna positions plotted. Since the separation between antennas is all that's important, a grid may be overlaid on the map with an arbitrary origin and orientation. Each antenna should be numbered sequentially beginning with one. The location of each antenna may then be specified by a pair of x,y coordinates. The program expects the x,y coordinates to be in feet so the grid should be designed appropriately. Figure A-3 is an example of an aircraft control tower antenna layout prepared appropriately for use as input to the program. Each antenna has been numbered and a suitable grid has been overlaid. TIMAP organizes the antenna information by the antenna number and assigns antenna numbers automatically for internal use. During data entry you will always be prompted for information on the next available antenna number.

Once the antenna numbers and locations have been assigned, the antenna heights must be determined. Since the relative heights are all that's needed to calculate antenna separations, any reference which eliminates terrain variations could be used. Height above sea level is assumed

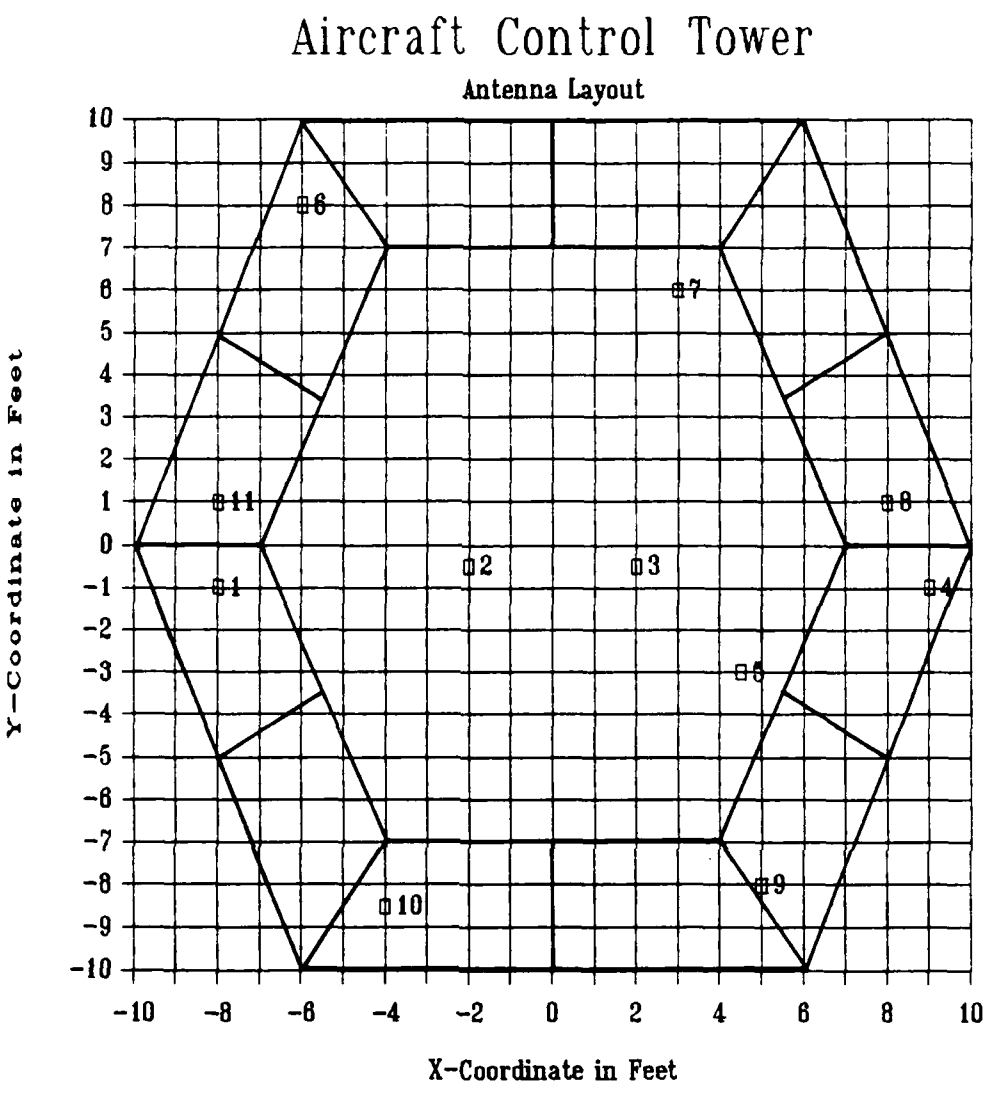


Figure A-3. An Antenna Layout Prepared For Program Input.

by the program but any elevation could be used.

Antenna gains are the final data needed. The gains needed are the gains of the antennas main lobe in dBi, or

decibels above isotropic, at the specified operating frequency.

Add Menu - Item 2 - Add frequency data.

Select this item to enter data describing the frequencies used in the colocated site. When you select this item the screen will clear and you will be prompted for frequency related information beginning with the frequency itself. If you've entered this function by mistake, don't worry. Just press <RETURN> without entering a frequency to return to the ADD MENU. Otherwise, enter the information as asked for. You may backspace over typing errors upon entry as long as you have not pressed the <RETURN> key. If you do make entry errors and don't notice them until after you've pressed <RETURN>, don't panic. You'll be able to to correct them using item 4 on the MAIN MENU, Change or delete data. When you have finished entering your frequency data, you may return to the ADD MENU by pressing <RETURN> when prompted for the next frequency. The frequency information will be sorted by frequency automatically before returning you to the ADD MENU. The information needed on each frequency is:

* The frequency, in MHz, of the carrier,
* The antenna number of the antenna used by the frequency,
* A usage code of T, R, or B denoting whether the

frequency is used for transmit only, receive only, or both transmit and receive, respectively,

* The bandwidth, in kHz, occupied by the signal,
* The coupling losses, in dB, between the radio and its antenna,
* A time used code of D, N, B, or C denoting whether the frequency is assigned for use during the day only, during the night only, during both day and night, or during a contingency only, respectively,
* The transmitter's output power, in Watts, for each transmit frequency,
* A transmitter type code of T or S denoting whether the transmitter's final output power amplifier is a tube type or a solid state type, respectively, for each transmit frequency, and
* A transmitter intermodulation coefficient, in dB, as assigned by ECAC for each transmit frequency.

Details on the frequency information are given below.

All frequencies being used in and around the site of concern must be included in the analysis. The frequencies should be expressed in MHz. If the same frequency is used on more than one antenna, the frequency should be listed for each antenna used.

The antenna number of the antenna being used by each frequency should be listed along with each frequency. Since all other information about the antenna is contained in the antenna data, no other antenna data must be entered with the frequency data. TIMAP will use the antenna number as a link between the antenna data and the frequency data.

A frequency may be used as receive only, transmit only, or both transmit and receive. Therefore, a "usage" code of R for receive only, T for transmit only, and B for both transmit and receive should be associated with each frequency. TIMAP uses this information to prevent a transmit only frequency from being considered as a victim frequency or a receive only frequency from being considered as a culprit frequency. It also uses this information during entry of the data to save the user from having to enter data associated only with transmit frequencies.

Each signal's bandwidth should be determined and expressed in kHz. The bandwidth is used to determine the thermal noise level at the front end of the victim receiver.

Losses, in dB, between the radios and their antennas should be determined next. These losses are referred to as coupling losses and include insertion and return losses due to cables, connectors, multicouplers, etc. Coupling losses

are frequency dependent but for a worst case analysis may be assumed to be frequency independent and equal to the losses at the operating frequency of the radio. Therefore the coupling losses need only be determined at each operating frequency.

Some frequencies are designated for use only at night, only during the day, or only during a contingency (war, disaster, etc.). This information may be used to reduce the number of frequencies that may combine to produce intermodulation products under certain conditions but is not implemented in the present program. This information should still be gathered, however, for use in interpreting the results of the program or for future use. For this purpose a "time used" code of D for a frequency used only during the day, N for a frequency used only during the night, B for a frequency used both day and night, and C for a frequency used only during a contingency should be assigned to each frequency.

For transmit frequencies, the radio's output power is needed. This should be the output power, expressed in Watts, that is applied to the antenna port of the radio.

A transmitter with a tube type of final output power amplifier has different intermodulation performance

characteristics than a transmitter with a solid state final output power amplifier. For this reason a "transmitter type" code of T for a transmitter with a tube type of final output amplifier and S for a transmitter with a solid state type of output amplifier must be assigned to each transmit frequency.

The final frequency data needed are the intermodulation coefficients, in dB, of each transmitter. The intermodulation coefficients vary with each transmitter model and are being catalogued by the Electromagnetic Compatibility Analysis Center (ECAC), Annapolis, Maryland 21402. For a worst case analysis these coefficients may be assumed to be 0 dB.

Add Menu - Item 3 - Clear current data.

Select this item if you want to discard any data already in the program so that you may enter new data. When you select this item all data presently in the computer's memory will be discarded and you will be returned to the ADD MENU. **IMPORTANT**: Data entered from the ADD MENU is **NOT** automatically saved to disk. You must use item 5 of the MAIN MENU, Save the data to disk, to save the data. If you select item 3 from the ADD MENU, Clear current data, without first saving your data, it will be discarded. However, this function will have no affect on any data saved on a disk.

Add Menu - Item 4 - Return to MAIN MENU.

Select this item if you have entered the ADD MENU by mistake or if you have finished adding data. When you select this function you will be returned to the MAIN MENU.

Main Menu - Item 2 - Print the current data.

Select this item if you would like a printout of the current antenna and frequency information. This function does not actually send the information to the printer. It sends the information to a disk file in a text format suitable for printing or manipulation with a text editor. When you select this item the screen will clear and you will be asked for a filename for the print files. All print files on the presently logged directory will be displayed to help you choose a name. Your filename may contain up to 8 characters and should not include an extension. A file extension of .APR will be assigned to the print file of your antenna data automatically and a file extension of .FPR will be assigned to the print file of your frequency data. If you specify a filename that already exists, that file will be overwritten. If you entered this function by mistake, just press <RETURN> without entering a filename to return to the MAIN MENU. Otherwise, enter your filename and press <RETURN>. TIMAP will print the information to the disk and then return you to the MAIN MENU.

Main Menu - Item 3 - Display the current data.

Select this item if you would like to view the data in the computer's memory on the screen. When you select this item you will enter the DISPLAY MENU (See Figure A-4).

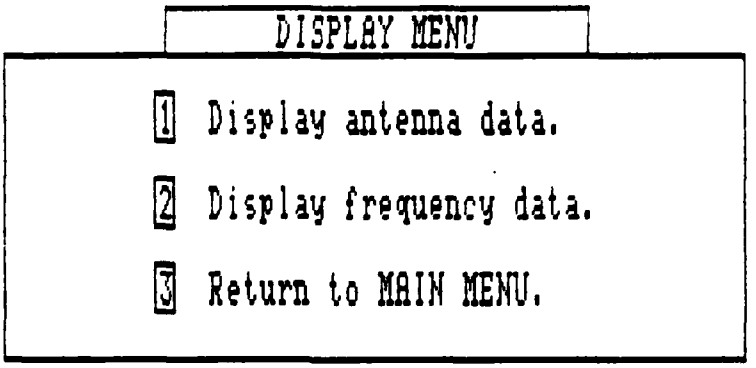


Figure A-4. TIMAP's DISPLAY MENU.

Display Menu - Item 1 - Display antenna data.

Select this item if you would like to view the current antenna data on the screen. When you select this item the screen will clear and one screen full of antenna data, i.e. antenna number, x-coordinate, y-coordinate, height, and gain, will be displayed in antenna number order beginning with antenna number one. If you would like to view the next screen full of antenna data press the space bar. If you would like to return to the DISPLAY MENU press <RETURN>.

When the last antenna has been displayed a message will appear on the screen. If you press the space bar to view more antennas when the last antenna has already been displayed, you will be returned to the DISPLAY MENU.

Display Menu - Item 2 - Display frequency data.

Select this item if you would like to view the current frequency data on the screen. When you select this item the screen will clear and all the information pertaining to the first frequency in the computer's memory will be displayed. If you press a space bar the next frequency will be displayed. If you press <RETURN> you will be returned to the DISPLAY MENU. When the last frequency has been displayed a message will be printed on the screen. If you press the space bar to view more frequencies after the last frequency has already been displayed, you will be returned the DISPLAY MENU.

Display Menu - Item 3 - Return to MAIN MENU.

Select this item if you have entered the DISPLAY MENU by mistake or you are finished viewing the data. When you select this item you will be returned to the MAIN MENU.

Main Menu - Item 4 - Change or delete data.

Select this item if you want to change or delete any of the antenna or frequency data. When you select this item

you will enter the EDIT MENU (See Figure A-5).

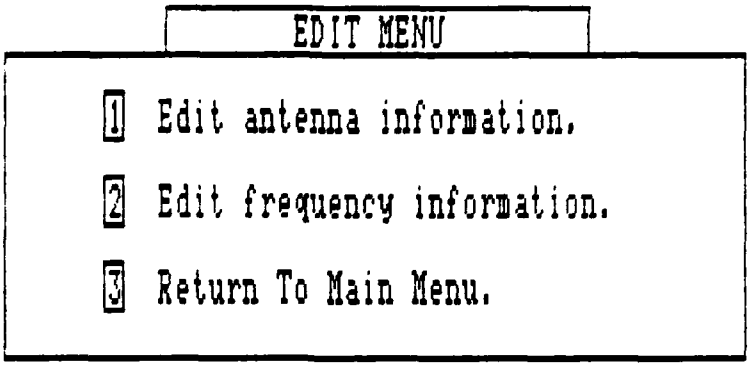


Figure A-5. TIMAP's EDIT MENU.

Edit Menu - Item 1 - Edit antenna information.

Select this item if you want to change the x-coordinate, y-coordinate, height, or gain of any antenna or to delete an antenna. When you select this item the screen will clear and the information for up to 12 antennas will be displayed on the screen beginning with antenna number 1. The cursor will be positioned beside the x-coordinate of antenna number 1.

If the antenna you wish to edit or delete is displayed, you may move the cursor to the data field of interest using the cursor (arrow) keys of the numeric keypad on the right

side of the keyboard and edit the data as described below. If the antenna you wish to edit is not displayed, use the <U> key, for Up, to display the next 12 antennas. TIMAP will always display 12 antennas at a time if there are at least 12 antennas in the current antenna list. To display the 12 antennas below the present display, use the <D>, for Down, key.

To edit data, move the cursor to the field containing the data to be changed. For example, if you had just entered the edit antenna mode and wanted to edit the height of antenna number three, you could move the cursor to the corresponding data field on the screen by pressing the down cursor key twice and then the right cursor key twice. Once you're in the data field you want to change, just press the number keys at the top of the keyboard or in the numeric keypad on the right of the keyboard to enter the new data. Press <RETURN> to make the change final. If you start to make a change and then decide you'd like to leave the entry as it was, just backspace until all of your entry is erased and then press <RETURN>. The original entry will be restored.

To delete an antenna just position the cursor in any field of data to the right of the antenna number you want to delete and press the <DELETE> key. Deleting an antenna has

two consequences. First, all the antenna numbers above the antenna you delete will be changed. This is done to maintain an unbroken sequence of antenna numbers. This means that you will have to update your map of the site appropriately. TIMAP will update the frequency data automatically. The second consequence is that any frequencies assigned to the deleted antenna will also be deleted. This could be especially hazardous if done unintentionally since there is no way to insert an antenna. For this reason, before deleting the antenna, TIMAP will ask you if you're sure. If you are, then just press <Y>, for Yes. Pressing any other key will abort the deletion and return you to the edit mode.

If you entered this function by mistake or are finished editing the data just press <HOME> to return to the EDIT MENU.

Edit Menu - Item 2 - Edit frequency information.

Select this item if you want to change any of the information associated with a frequency or to delete a frequency. When you select this item the screen will clear and you will be asked which frequency you want to edit. You must enter a frequency in the current frequency list. If you entered this function by mistake, just press <RETURN> without entering a frequency to return to the EDIT MENU.

Once TIMAP finds the frequency you entered, it will display all the pertinent data and the cursor will be sitting on the frequency. At this point you may step through the frequency list, edit the data, or delete a frequency.

To step through the frequency list use the <+>, plus, or <->, minus, keys. Pressing <+> will display the next higher frequency until the highest frequency in the list is reached. Pressing <+> after the highest frequency is already displayed just redisplays the highest frequency. Pressing <-> will display the next lower frequency until the lowest frequency in the frequency list is reached. Pressing <-> once the lowest frequency is displayed just redisplays the lowest frequency.

To edit the data, use the cursor keys to position the cursor on the data you wish to change. Instructions as to the type of data expected will be given slightly below mid screen. Enter the appropriate data and press <RETURN>. In any data field requiring more than a single key press to enter the data, i.e. the frequency, bandwidth, cable loss, transmitter output power, transmitter intermodulation coefficient, and antenna number, you may abort a change by backspacing until the entire entry is erased and then press <RETURN>. The original entry will be restored. Changes to

the frequency will cause the frequency list to be resorted.

To return to the EDIT MENU press <HOME>.

Edit Menu - Item 3 - Return to MAIN MENU.

Select this item if you have entered the EDIT MENU by mistake or you are finished making changes to the data. When you select this item you will be returned to the MAIN MENU.

Main Menu - Item 5 - Save the data to disk.

Select this item if you want to save the current data for later review, update, or reanalysis. When you select this item the screen will clear and you will be asked for a filename for your data file. To help you pick a name, you will be provided a list of data files already on the logged directory. You may use up to 8 characters in your filename and should not include an extension. For your antenna data a .ATD extension will be appended to your filename automatically. For your frequency data a .FQD extension will be appended to your filename automatically. If you entered this function by mistake, just press <RETURN> without entering a filename and you will be returned to the MAIN MENU. Otherwise, enter the filename for your data file and press <RETURN>. Your antenna and frequency data will be written to the disk in the presently logged directory and

you will be returned to the MAIN MENU. **NOTE: Use of this function is the only time TIMAP makes changes to data files on the disk.**

Main Menu - Item 6 - Load data from disk.

Select this item if you have previously saved some data files to disk and you now want to load that data into the program. When you select this item the screen will clear and you will be provided a list of all data files in the currently logged directory. You will be asked to give the filename of the data file you want to load. You may use up to 8 characters in your filename and should not include the extension. If you entered this function by mistake, just press <RETURN> without entering a filename and you will be returned to the MAIN MENU. Otherwise, enter your filename and press <RETURN>. The data will be loaded into TIMAP and you will be returned to the MAIN MENU.

Main Menu - Item 7 - Perform an analysis.

Select this item if you have data loaded into TIMAP and are ready to have it analyzed. When you select this item the screen will clear and you will asked for a filename for the results of the analysis. A list of all analysis output files in the currently logged directory will also be given to help you select a name. You may use up to 8 characters in your filename and should not include an extension. Your

filename will be given a .OUT extension automatically for storing the results of the analysis. If you entered this function by mistake, just press <RETURN> without entering a filename and you will be returned to the MAIN MENU. Otherwise, enter your filename and press <RETURN>. The screen will clear and a message announcing that the first receive frequency is being analyzed will appear at the top of the screen. The length of time it takes to complete the analysis is dependent on the number of frequencies involved. As analysis begins on each new receive frequency the message at the top of the screen will be updated showing which frequency is being analyzed. The results are continuously written to the disk so **DO NOT REMOVE THE DISK WHILE THE ANALYSIS IS BEING PERFORMED.** It is also important to make sure there is enough room on the disk to save all the results. It is good practice to set aside a blank (but formatted) disk for use during the analysis. You will then have to log in that disk before beginning the analysis as described in Main Menu - Item 8 - Set options. The results are written to the disk in a text format suitable for printing or editing with a text processor.

Main Menu - Item 8 - Set options.

Select this item to change any of the three options provided by TIMAP. The purpose of each option is described below. When you select this item you will enter the OPTIONS

MENU (See Figure A-6).

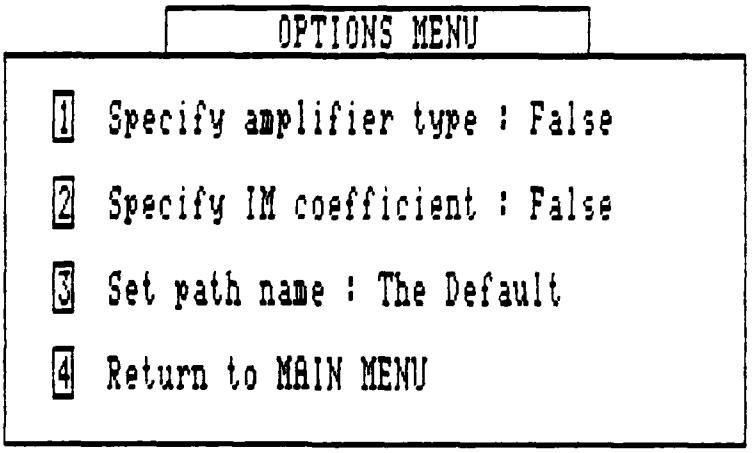


Figure A-6. TIMAP's OPTIONS MENU.

Options Menu - Item 1 - Specify amplifier type.

Select this item to modify the way data is requested when in TIMAP's "Add frequency data" mode. When you select this item you will be provided a description of the purpose for the option and asked how you want it set, either true or false. With the option set to true, you will be asked what type of amplifier, either tube or solid state, is used in the final output stage of transmitters during TIMAP's "Add frequency data" mode. Tube and solid state amplifiers have different intermodulation performance characteristics which TIMAP will take into consideration. With this option set to false, TIMAP will assume all amplifiers are solid state and

not ask for the information while in the "Add frequency data" mode. The present setting is always displayed in the OPTIONS MENU to the right of the option. You should set this option to false if you do not know what the amplifier types are or if you want to do a worst case analysis. The default setting is false.

Options Menu - Item 2 - Specify IM coefficient.

Select this item to modify the way data is requested when in TIMAP's "Add frequency data" mode. When you select this item you will be provided a description of the purpose for the option and asked how you want it set, either true or false. With the option set to true, you will be asked what the intermodulation coefficient, in dB, is for each transmitter during TIMAP's "Add frequency data" mode. Intermodulation coefficients vary with each transmitter model and are being catalogued by the Electromagnetic Compatibility Analysis Center, Annapolis, Maryland, 21402. With this option set to false, TIMAP will assume all transmitters have an intermodulation coefficient of 0 dB and not ask for the information while in the "Add frequency data" mode. The present setting is always displayed in the OPTIONS MENU to the right of the option. You should set this option to false if you do not know what the intermodulation coefficients are or if you want to do a worst case analysis. The default setting for this option is

false.

Options Menu - Item 3 - Set path name.

Select this option if you want to log in a new disk drive or directory for use in reading or writing data, print, and output files. When you select this item you will be given instruction about the use of this option, the current path will be displayed, and you will be asked for a new path. A description of paths is given in your MS-DOS manual. If you press <RETURN> the default path and drive, i.e. the path (or directory) and drive you used when you first invoked TIMAP, will be assumed. Otherwise, enter your new path. This is one area where I did no checking for valid input. I couldn't justify the time and effort it would have taken to determine exactly what all valid paths would look like. Changing the path only affects where data files are read from or written to, where analysis output files are written, and where print files are written. The TIMAP files, TIMAP.COM and TIMAP.000, must remain in the default directory during operation. The default setting for this option is "The Default".

Options Menu - Item 4 - Return to MAIN MENU.

Select this item if you have entered the OPTIONS MENU by mistake or you have finished changing any options. When you select this item you will be returned to the MAIN MENU.

Main Menu - Item 9 - Exit to the system.

Select this item when you are through using TIMAP and wish to return to MS-DOS. Two things may happen when you select this item.

(1) If you have made any changes to the data and not saved the data to disk you will be given a warning message and asked if you really want to exit to the system. If you do, type <Y> for Yes. You will be returned to MS-DOS. Otherwise, press any other key and you will be returned to TIMAP's MAIN MENU. You may then save the data using item 5 before selecting this item again to exit to MS-DOS.

(2) If no changes to the data have been made since the last save, you will be returned to MS-DOS.

I highly recommended that you use this option to exit TIMAP since it warns of changed data, clears the 25th line on your screen, and assures you will have a cursor after exiting TIMAP. If you exit by using a control C, this can not be guaranteed.

Comments And Warnings

TIMAP will handle up to 200 antennas and 200 frequencies; however, I sincerely hope you never have to analyze a site that large. The processing time would be horrendous.

I have included instructions on screen during many of the interactions with TIMAP that were not described in the manual. I hope they make learning and using TIMAP as painless as possible and thereby promote its use. I also check for reasonable data during input when practical but the program is not idiot proof.

No computer program is ever "finished" and so it goes with TIMAP. One feature sorely lacking in the present version of TIMAP is the ability to view an analysis output file from within TIMAP. You presently must exit TIMAP and enter a text processor to look at the output file. Maintaining consecutive antenna numbers is also an inconvenience to the user that should be eliminated. And finally, I would like to incorporate some enhancements, such as the inclusion of receiver intermodulation analysis and the automatic suggestion of corrective measures for problems found.

If you have more than 128k bytes of user memory, I recommend you use some of that memory for a memory disk (ram disk) and copy TIMAP to the memory disk for execution. This is because TIMAP uses overlays. That means that every time you select a new item from the main menu, TIMAP must access the disk to load the desired functions into memory. Executing TIMAP from a memory disk speeds up operation and saves wear and tear on your disk drives. MS-DOS comes with a file called MDISK.DVD that allows you to create 64k byte memory disks. The Programmer's Utility Pack for MS-DOS comes with the assembly source code for a version of MDISK that allows you to specify the size of the memory disk. And there is other software available to do the same thing. Since I have 768k bytes of user memory, I always set up a 450k byte memory disk using the Programmer's Utility Pack version of MDISK. The instructions are in the manual that comes with the Programmer's Utility Pack.

I believe TIMAP performs its primary purpose well and is very usable in its present form but there are many things that time constraints prohibited me from cleaning up. If you run across any bugs or have any comments, please send them to me. My permanent address is:

Thomas J. Zuzack
1214 Waverly Drive
Latrobe, PA. 15650

APPENDIX B

SOURCE CODE LISTING

FOR

TIMAP VERSION 1.00

BY

THOMAS J. ZUZACK

TABLE OF CONTENTS

	Page
TIMAP.PAS	B-1
ADD_DATA.PAS	B-3
Sort_Frequencies	B-3
Swap	B-3
Add_Data	B-4
Add_Antenna_Data	B-4
Get_X_Coord	B-5
Get_Y_Coord	B-6
Get_Z_Coord	B-7
Get_Gain	B-8
Add_Frequency_Data	B-9
Get_Frequency	B-10
Get_Antenna_No	B-11
Get_Usage_Code	B-12
Get_Bandwidth	B-13
Get_Time_Used	B-14
Get_Cable_Loss	B-15
Get_Output_Power	B-16
Get_IM_Coefficient	B-17
Get_Amplifier_Type	B-18
Clear_Data	B-20
Return_To_Main_Menu	B-20
ANALYZE.PAS	B-21
Do_Analysis	B-21
Open_Output_File	B-21
Display_Instructions	B-22
Get_Filename	B-23
Announce_New_Victim_Frequency	B-24
Log	B-24
Thermal_Noise	B-25
Prop_Loss	B-25
Distance	B-26
Culprit_Power	B-27
Selectivity	B-28
TX_IM_Power	B-29
RX_IM_Power	B-30
Print_The_Intermodulation_Information	B-30
Write_A_Plus_B_Info	B-31
Write_A_Minus_B_Info	B-31
Write_2A_Minus_B_Info	B-32
Write_A_Plus_B_Minus_C_Info	B-32
Write_3A_Minus_2B_Info	B-33

Page

Is_The_IM_Frequency_Within_The_Victim_Bandwidth . B-34
Test_For_A_Plus_B_IM_Interference B-34
Are_Victim_And_Both_Culprit_Frequencies_Unique B-35
Select_Culprit_1_As_The_Threat B-35
Select_Culprit_2_As_The_Threat B-35
Does_The_IM_Power_At_The_Victim_Exceed_The_Noise B-36
Check_For_Interference B-36
Test_For_A_Minus_B_IM_Interference B-37
Are_Victim_And_Both_Culprit_Frequencies_Unique B-38
Select_Culprit_1_As_The_Threat B-38
Select_Culprit_2_As_The_Threat B-38
Does_The_IM_Power_At_The_Victim_Exceed_The_Noise B-39
Check_For_Interference B-40
Test_For_2A_Minus_B_IM_Interference B-41
Are_Victim_And_Both_Culprit_Frequencies_Unique B-41
Select_Culprit_1_As_The_Threat B-42
Select_Culprit_2_As_The_Threat B-42
Does_The_IM_Power_At_The_Victim_Exceed_The_Noise B-43
Check_For_Interference B-43
Test_For_A_Plus_B_Minus_C_IM_Interference B-45
Are_Victim_And_All_Culprit_Frequencies_Unique . B-45
Select_Culprit_1_As_The_Threat B-46
Select_Culprit_2_As_The_Threat B-46
Select_Culprit_3_As_The_Threat B-46
Does_The_IM_Power_At_The_Victim_Exceed_The_Noise B-47
Check_For_Interference B-48
Loop_1 . . . . . B-49
Loop_2 . . . . . B-49
Loop_3 . . . . . B-50
Loop_4 . . . . . B-50
Test_For_3A_Minus_2B_IM_Interference B-51
Are_Victim_And_Both_Culprit_Frequencies_Unique B-52
Select_Culprit_1_As_The_Threat B-52
Select_Culprit_2_As_The_Threat B-52
Does_The_IM_Power_At_The_Victim_Exceed_The_Noise B-53
Check_For_Interference B-53

DECLARE.PAS B-55

EDITDATA.PAS B-56

Edit_Data B-56
Bad_Data B-56
Display_Frequency_Instructions B-56
Get_Frequency_To_Edit B-57

Page

Delete_Frequency B-58
Display_Usage B-58
Display_Time_used B-59
Display_Amplifier_Type B-59
Display_Output_Power B-60
Display_IM_Coef B-60
Display_Info_For_Frequency B-61
Delete_This_Frequency B-62
GotoFreqField B-63
Display_Edit_Instructions B-64
Display_Edit_Error B-65
Backspace B-65
Get_Frequency B-66
Get_Bandwidth B-67
Get_TX_RX_Or_Both B-68
Get_Day_Night_Or_Both B-69
Get_Cable_Loss B-70
Get_Output_Power B-71
Get_Amplifier_Type B-72
Get_TX_IM_Coef B-73
Get_Antenna_Number B-74
Process_Input B-75
Edit_Next_Frequency B-75
Edit_Previous_Frequency B-76
Edit_Frequency B-76
Process_Esc B-77
Change_Frequency_Data B-78
Update_Antenna_Numbers_In_Freq_List B-79
Delete_Antenna_Number B-79
Display_Antenna_Headings B-80
Display_Antenna_Instructions B-80
Display_Antennas_Starting_At_Number B-81
GotoAntField B-82
Scroll_Screen_Up B-82
Scroll_Screen_Down B-83
Get_Xcoord B-84
Get_Ycoord B-85
Get_Zcoord B-86
Get_Gain B-87
Process_Ant_Input B-88
Delete_This_Antenna B-89
Change_Antenna_Data B-90
Process_Esc B-90
Return_To_Main_Menu B-91

EXIT2SYS.PAS B-92

Exit_To_System B-92

Page

GRAPHICS.PAS B-93

Line_25 . . . . . B-94
Clrline B-94
Graphics_On B-95
Display_Driver B-95
Initialize_Display_Driver B-96
Graphics_Off B-97
Reverse_Video B-97
Normal_Video B-98
Cursor B-99
Clrs2end B-100
Clrs2beg B-100
Clrl2beg B-101
Byteaddr B-101
Pset B-102
Preset B-103
Drawline B-103
Case1 . . . . . B-104
Case2 . . . . . B-104
Case3 . . . . . B-104
Case4 . . . . . B-105
Case5 . . . . . B-105
Case6 . . . . . B-105
Case7 . . . . . B-106
Case8 . . . . . B-106
Drawbox B-108
Drawcircle B-109
Reflect B-109

LOADDATA.PAS B-111

Load_data B-111
Display_Instructions B-111
Alert_User B-112
Find_File B-113
Get_Data B-114

MENUS.PAS B-115

Menu_Item_Box B-115
Display_Main_Menu B-116
Display_Add_Menu B-117
Display_Display_Menu B-118
Display_Print_Menu B-119
Display_Options_Menu B-120
Display_Edit_Menu B-121
Get_Menu_Selection B-121
Process_Cr B-122
Process_Esc B-122

Page

Process_Up_Arrow B-122
Process_Down_Arrow B-123
Process_Number_Key B-124

OPTIONS.PAS B-125

Initialize_Defaults B-125
Set_Options B-125
Set_Amplifier_Option B-126
Set_IM_Coefficient_Option B-127
Set_Path_Name B-128
Return_To_Main_Menu B-129

PRNTDATA.PAS B-130

Print_Data B-130
Display_Instructions B-130
Get_Filename B-131
Write_Antenna_Data_To_File B-132
Write_Antenna_Headings B-132
Write_Frequency_Data_To_File B-133
Write_Frequency_Headings B-133

SAVEDATA.PAS B-135

Save_Data B-135
Display_Instructions B-135
Get_Filename B-136
Write_Data_To_File B-137

SHOWDATA.PAS B-138

Display_Data B-138
No_More B-138
Display_Antenna_Data B-139
Display_Antenna_Headings B-139
Display_One_Screen_Of_Antennas B-140
Display_Frequency_Data B-141
Display_One_Screen_Of_Frequencies B-141
Return_To_Main_Menu B-142

UTILITY.PAS B-144

Date B-144
Time B-145
Cwrite B-146
Tab B-146
Spc B-147
Display_Directory B-148
Directory_Full B-150

```
{*************************************************************
*************************************************************
***********       TRANSMITTER INTERMODULATION       ***********
***********            ANALYSIS PROGRAM             ***********
***********          By Thomas J. Zuzack            ***********
***********           5783 Gross Drive              ***********
***********           Dayton, OH 45431              ***********
***********            (513) 258-8713               ***********
***********                                         ***********
***********   Version 1.00          5 JUNE 1986     ***********
*************************************************************
*************************************************************
*********** Source File       : TIMAP.PAS           ***********
*********** Executable Files  : TIMAP.COM           ***********
***********                     TIMAP.000           ***********
*********** Included Files    : ADD_DATA.PAS        ***********
***********                     ANALYZE .PAS        ***********
***********                     DECLARE .PAS        ***********
***********                     EDITDATA.PAS        ***********
***********                     EXIT2SYS.PAS        ***********
***********                     GRAPHICS.PAS        ***********
***********                     LOADDATA.PAS        ***********
***********                     MENUS   .PAS        ***********
***********                     OPTIONS .PAS        ***********
***********                     PRNTDATA.PAS        ***********
***********                     SAVEDATA.PAS        ***********
***********                     SHOWDATA.PAS        ***********
***********                     UTILITY .PAS        ***********
*********** Operating System  : MS-DOS 2.22         ***********
*********** Language          : TurboPascal 3.01A   ***********
*********** Computer          : H/Z-100             ***********
***********                     768k User Memory    ***********
***********                     192k Video Memory   ***********
***********                     8 MHz Clock         ***********
***********                     2-5.25" Floppies    ***********
*************************************************************
*************************************************************}

{$I DECLARE .PAS }       {NOTE: The ordering of these       }
{$I GRAPHICS.PAS }       {      include files is important. }
{$I UTILITY .PAS }
{$I MENUS   .PAS }
{$I ADD_DATA.PAS }
{$I PRNTDATA.PAS }
{$I SHOWDATA.PAS }
{$I EDITDATA.PAS }
{$I SAVEDATA.PAS }
{$I LOADDATA.PAS }
{$I ANALYZE .PAS }
{$I OPTIONS .PAS }
{$I EXIT2SYS.PAS }
```

```
BEGIN { Main Program }

  Graphics_On ;
  no_o_antennas := 0 ;
  no_o_frequencies := 0 ;
  data_changed := FALSE ;
  Initialize_Defaults ;
  REPEAT ;
    Display_Main_Menu ;
    finished := FALSE ;
    CASE ( Get_Menu_Selection( 4, 9, 25 ) ) OF

      1 : Add_Data ;
      2 : Print_Data ;
      3 : Display_Data ;
      4 : Edit_Data ;
      5 : Save_Data ;
      6 : Load_Data ;
      7 : Do_Analysis ;
      8 : Set_Options ;
      9 : Exit_To_System ;

    END ; { CASE STATEMENT }

  UNTIL finished ;

END . { Main Program }

{*****************************************************************}
```

```
{************************************************************
* This procedure sorts the frequency list by frequency in
* increasing order.  It is not placed in an overlay so that
* it will be readily available to both the "add data" and
* "edit data" routines. (Version 1.00   5 June 1986)
************************************************************}

PROCEDURE Sort_Frequencies ;

  VAR
    i, j, k : INTEGER ;

{////////////////////////////////////////////////////////////
/ This procedure swaps two entries in the frequency list.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE SWAP ;

  VAR
    temp : frequency_info ;

BEGIN { swap }
  temp := freq_list[j] ;
  freq_list[j] := freq_list[j+k] ;
  freq_list[j+k] := temp ;
END ; { swap }

{////////////////////////////////////////////////////////////}

BEGIN { Sort_Frequencies }
  k := no_o_frequencies DIV 2 ;
  WHILE k > 0 DO BEGIN
    FOR i := k + 1 TO no_o_frequencies DO BEGIN
      j := i - k ;
      WHILE j > 0 DO BEGIN
        IF freq_list[j].frequency > freq_list[j+k].frequency
          THEN BEGIN
            SWAP ;
            j := j - k ;
        END { IF freq(j) > freq(j+k) }
        ELSE j := 0 ;
      END ; { WHILE j > 0 }
    END ; { FOR i }
    k := k DIV 2 ;
  END ; { WHILE k }
END ; { Sort_Frequencies }
```

```
{************************************************************
* This procedure allows adding the data required for an
* intermodulation interference analysis from the keyboard.
* (Version 1.00  5 June 1986)
************************************************************}

OVERLAY PROCEDURE Add_Data ;

  VAR
    done : BOOLEAN ;

{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the antenna's
/ x-coordinate, y-coordinate, height, and gain and stores
/ the entries in the antenna list.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Add_Antenna_Data ;

  VAR
    antenna_no    : INTEGER ;
    quit          : BOOLEAN ;
```

```
{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the antenna's
/ x-coordinate, checks for valid input, and stores the
/ entry in the antenna list.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_X_Coord ;

  VAR
    temp         : REAL ;
    result       : INTEGER ;
    input_string : STRING[12];
    good_input   : BOOLEAN ;

BEGIN { Get_X_Coord }
  quit := FALSE ;
  Reverse_Video ;
  Cwrite('Press <RETURN> without entering an X coordinate', 22) ;
  Cwrite('to return to the ADD MENU.', 23);
  Normal_Video ;
  GOTOXY( 1,1 );
  REPEAT
    WRITE('What is the X coordinate (in feet) of antenna # ',
          antenna_no,' ?  ' ) ;
    Cursor( block, noblink, noclick, on );
    READLN(input_string) ;
    IF LENGTH(input_string) = 0
      THEN BEGIN
        quit := TRUE ;
        EXIT ;
    END ; { IF LENGTH }
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) ;
    IF NOT good_input
      THEN BEGIN
        Reverse_Video ;
        WRITELN(input_string,' is not a valid X coordinate.');
        WRITELN('Please try again.');
        Normal_Video ;
        GOTOXY( 1,1 ) ; clrline ;
    END ; { IF result }
  UNTIL good_input ;
  data_changed := TRUE ;
  ant_list[antenna_no].xcoord := temp ;
END ; { Get_X_Coord }
```

```
{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the antenna's
/ y-coordinate, checks for valid input, and stores the
/ entry in the antenna list.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_Y_Coord ;

  VAR
    temp         : REAL ;
    result       : INTEGER ;
    input_string : STRING[12];
    good_input   : BOOLEAN ;

BEGIN { Get_Y_Coord }
  Clrs2end ;
  REPEAT
    WRITE('What is the Y coordinate (in feet) of antenna # ',
          antenna_no,' ?  ' ) ;
    Cursor( block, noblink, noclick, on );
    READLN(input_string) ;
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND
                  (LENGTH( input_string ) <> 0) ;
    IF NOT good_input
      THEN BEGIN
        Reverse_Video ;
        WRITELN(input_string,' is not a valid Y coordinate.');
        WRITELN('Please try again.');
        Normal_Video ;
        GOTOXY( 1,2 ) ; clrline ;
    END ; { IF result }
  UNTIL good_input ;
  ant_list[antenna_no].ycoord := temp ;
END ; { Get_Y_Coord }
```

```
{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the antenna's
/ height, checks for valid input, and stores the entry
/ in the antenna list.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_Z_Coord ;

  VAR
    temp         : REAL ;
    result       : INTEGER ;
    input_string : STRING[12];
    good_input   : BOOLEAN ;

BEGIN { Get_Z_Coord }
  Clrs2end ;
  REPEAT
    WRITE('What is the height (in feet above sea level ) ',
          'of antenna # ',antenna_no,' ?  ' ) ;
    Cursor( block, noblink, noclick, on );
    READLN(input_string) ;
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND
                  (LENGTH( input_string ) <> 0) ;
    IF NOT good_input
      THEN BEGIN
        Reverse_Video ;
        WRITELN(input_string,' is not a valid Z coordinate.');
        WRITELN('Please try again.');
        Normal_Video ;
        GOTOXY( 1,3) ; clrline ;
    END ; { IF result }
  UNTIL good_input ;
  ant_list[antenna_no].zcoord := temp ;
END ; { Get_Z_Coord }
```

```
{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the antenna's gain,
/ checks for valid input, and stores the entry in the
/ antenna list.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_Gain ;

  VAR
    temp         : REAL ;
    result       : INTEGER ;
    input_string : STRING[12];
    good_input   : BOOLEAN ;

BEGIN { Get_Gain }
  Clrs2end ;
  REPEAT
    WRITE('What is the gain (in dBi) of antenna # ',
          antenna_no,' ?  ' ) ;
    Cursor( block, noblink, noclick, on );
    READLN(input_string) ;
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND
                  (LENGTH( input_string ) <> 0) ;
    IF NOT good_input
      THEN BEGIN
        Reverse_Video ;
        WRITELN(input_string,' is not a valid gain.');
        WRITELN('Please try again.');
        Normal_Video ;
        GOTOXY( 1,4) ; clrline ;
    END ; { IF result }
  UNTIL good_input ;
  ant_list[antenna_no].gain := temp ;
END ; { Get_Gain }
```

```
{////////////////////////////////////////////////////////////}

BEGIN { Add_Antenna_Data }
  antenna_no := no_o_antennas ;
  REPEAT
    CLRSCR ;
    antenna_no := antenna_no + 1 ;
    Get_X_Coord ;
    IF NOT quit
      THEN BEGIN
        Get_Y_Coord ;
        Get_Z_Coord ;
        Get_Gain ;
    END ; { IF NOT finished }
  UNTIL quit ;
  no_o_antennas := antenna_no - 1 ;
END ; { Add_Antenna_Data }




{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the frequency,
/ antenna number, usage code, bandwidth, time used code,
/ cable loss, transmitter output power, transmitter IM
/ coefficient, and transmitter final output amplifier type
/ and stores the entries in the frequency list.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Add_Frequency_Data ;

  VAR
    freq_no : INTEGER ;
    quit    : BOOLEAN ;
```

```
{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the frequency, checks
/ for valid input, and stores the entry in the frequency
/ list.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_Frequency ;

  VAR
    temp         : REAL ;
    result       : INTEGER ;
    input_string : STRING[7];
    good_input   : BOOLEAN ;

BEGIN { Get_Frequency }
  quit := FALSE ;
  Reverse_Video ;
  Cwrite('Press <RETURN> without entering a frequency', 22) ;
  Cwrite('to return to the ADD MENU.', 23);
  Normal_Video ;
  GOTOXY( 1,1 );
  REPEAT
    WRITE('What is the frequency (in MHz) ?  ') ;
    Cursor( block, noblink, noclick, on );
    READLN(input_string) ;
    IF LENGTH(input_string) = 0
      THEN BEGIN
        quit := TRUE ;
        EXIT ;
    END ; { IF LENGTH }
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) ;
    IF NOT good_input
      THEN BEGIN
        Reverse_Video ;
        WRITELN(input_string,' is not a valid X coordinate.');
        WRITELN('Please try again.');
        Normal_Video ;
        GOTOXY( 1,1 ) ; clrline ;
    END ; { IF result }
  UNTIL good_input ;
  data_changed := TRUE ;
  freq_list[freq_no].frequency := temp ;
END ; { Get_Frequency }
```

```
{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the number of the
/ antenna being used by previously entered frequency, checks
/ for valid input, and stores the entry in the frequency
/ list.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_Antenna_No ;

  VAR
    temp         : INTEGER ;
    result       : INTEGER ;
    input_string : STRING[7];
    good_input   : BOOLEAN ;

BEGIN { Get_Antenna_No }
  Clrs2end ;
  REPEAT
    WRITE('Which antenna uses ',
          freq_list[freq_no].frequency:7:3,' MHz ?  ') ;
    Cursor( block, noblink, noclick, on );
    READLN(input_string) ;
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND
                  (LENGTH( input_string ) <> 0) ;
    IF NOT good_input
      THEN BEGIN
        Reverse_Video ;
        WRITELN(input_string,
          ' is not a valid antenna number.') ;
        WRITELN('Please try again.');
        Normal_Video ;
        GOTOXY( 1,2 ) ; clrline ;
    END ; { IF result }
  UNTIL good_input ;
  freq_list[freq_no].antenna_no := temp ;
END ; { Get_Antenna_No }
```

```
{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the usage code,
/ checks for valid input, and stores the entry in the
/ frequency list.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_Usage_Code ;

  VAR
    input_char    : CHAR ;
    good_input    : BOOLEAN ;

BEGIN { Get_Usage_Code }
  Clrs2end ;
  REPEAT
    WRITE('Is ',freq_list[freq_no].frequency:7:3,
          ' MHz used for (T)ransmit, (R)eceive, or (B)oth ?  ') ;
    Cursor( block, noblink, noclick, on );
    READ(KBD,input_char) ;
    WRITELN( UPCASE( input_char ) ) ;
    CASE UPCASE( input_char ) OF

      'T','R','B' : good_input := TRUE ;
      ELSE BEGIN
             good_input := FALSE ;
             Reverse_video ;
             WRITELN('Your input must be a T, R, or B.') ;
             WRITELN('Please try again.') ;
             Normal_Video ;
             GOTOXY( 1,3 ) ;
             Clrline ;
           END ; { ELSE }

    END ; { CASE UPCASE( input_char ) }
  UNTIL good_input ;
  freq_list[freq_no].usage := UPCASE( input_char ) ;
END ; { Get_Usage_Code }
```

```
{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the bandwidth of the
/ frequency previously entered, checks for valid input, and
/ stores the entry in the frequency list.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_Bandwidth ;

  VAR
    temp         : REAL ;
    result       : INTEGER ;
    input_string : STRING[7];
    good_input   : BOOLEAN ;

BEGIN { Get_Bandwidth }
  Clrs2end ;
  REPEAT
    WRITE('What bandwidth (in kHz) does ',
          freq_list[freq_no].frequency:7:3,' MHz ',
          'occupy ?  ') ;
    Cursor( block, noblink, noclick, on );
    READLN(input_string) ;
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND
                  (LENGTH( input_string ) <> 0) ;
    IF NOT good_input
      THEN BEGIN
        Reverse_Video ;
        WRITELN(input_string,
          ' is not a valid bandwidth.') ;
        WRITELN('Please try again.');
        Normal_Video ;
        GOTOXY( 1,4 ) ; clrline ;
    END ; { IF result }
  UNTIL good_input ;
  freq_list[freq_no].bandwidth := temp ;
END ; { Get_Bandwidth }
```

```
{////////////////////////////////////////////////////////////
/ This procedure prompts the user for a time used code,
/ checks for valid input, and stores the entry in the
/ frequency list.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_Time_Used ;

  VAR
    input_char    : CHAR ;
    good_input    : BOOLEAN ;

BEGIN { Get_Time_Used }
  Clrs2end ;
  REPEAT
    WRITE('When is ',freq_list[freq_no].frequency:7:3,
          ' MHz used (D)ay, (N)ight, (B)oth, or ',
          '(C)ontingency ?  ') ;
    Cursor( block, noblink, noclick, on );
    READ(KBD,input_char) ;
    WRITELN( UPCASE( input_char ) ) ;
    CASE UPCASE( input_char ) OF

      'D','N','B','C' : good_input := TRUE ;
      ELSE BEGIN
        good_input := FALSE ;
        Reverse_video ;
        WRITELN('Your input must be a D, N, B, or C.') ;
        WRITELN('Please try again.') ;
        Normal_Video ;
        GOTOXY( 1,5 ) ;
        Clrline ;
      END ; { ELSE }

    END ; { CASE UPCASE( input_char ) }
  UNTIL good_input ;
  freq_list[freq_no].time := UPCASE( input_char ) ;
END ; { Get_Time_Used }
```

```
{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the cable loss,
/ checks for valid input, and stores the entry in the
/ frequency list.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_Cable_Loss ;

  VAR
    temp         : REAL ;
    result       : INTEGER ;
    input_string : STRING[7];
    good_input   : BOOLEAN ;

BEGIN { Get_Cable_Loss }
  Clrs2end ;
  REPEAT
    WRITE('How much loss (in dB) is there between the ',
          'antenna and the radio ?  ') ;
    Cursor( block, noblink, noclick, on );
    READLN(input_string) ;
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND
                  (LENGTH( input_string ) <> 0) ;
    IF NOT good_input
      THEN BEGIN
        Reverse_Video ;
        WRITELN(input_string,
          ' is not a valid loss.') ;
        WRITELN('Please try again.');
        Normal_Video ;
        GOTOXY( 1,6 ) ; clrline ;
    END ; { IF result }
  UNTIL good_input ;
  freq_list[freq_no].cable_loss := temp ;
END ; { Get_Cable_Loss }
```

```
{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the transmitter
/ output power, checks for valid input, and stores the
/ entry in the frequency list.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_Output_Power ;

  VAR
    temp         : REAL ;
    result       : INTEGER ;
    input_string : STRING[7];
    good_input   : BOOLEAN ;

BEGIN { Get_Output_Power }
  Clrs2end ;
  REPEAT
    WRITE('What is the transmitter output power (in Watts) ',
          'for ',freq_list[freq_no].frequency:7:3,' MHz ',
           '?  ') ;
    Cursor( block, noblink, noclick, on );
    READLN(input_string) ;
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND
                  (LENGTH( input_string ) <> 0) ;
    IF NOT good_input
      THEN BEGIN
        Reverse_Video ;
        WRITELN(input_string,
          ' is not a valid output power.') ;
        WRITELN('Please try again.');
        Normal_Video ;
        GOTOXY( 1,7 ) ; clrline ;
    END ; { IF result }
  UNTIL good_input ;
  freq_list[freq_no].output_power := temp ;
END ; { Get_Output_Power }
```

```
{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the transmitter IM
/ coefficient, checks for valid input, and stores the
/ entry in the frequency list.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_IM_Coefficient ;

  VAR
    temp         : REAL ;
    result       : INTEGER ;
    input_string : STRING[7];
    good_input   : BOOLEAN ;

BEGIN { Get_IM_Coefficient }
  IF NOT specify_im_coef
    THEN BEGIN
      freq_list[freq_no].im_coef_K1 := 0 ;
      EXIT ;
    END ; { IF NOT specify_im_coef }
  Clrs2end ;
  REPEAT
    WRITE('What is the transmitter''s intermodulation ',
          'coefficient (in dB) ?  ') ;
    Cursor( block, noblink, noclick, on );
    READLN(input_string) ;
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND
                  (LENGTH( input_string ) <> 0) ;
    IF NOT good_input
      THEN BEGIN
        Reverse_Video ;
        WRITELN(input_string,
          ' is not a valid intermodulation coefficient.') ;
        WRITELN('Please try again.');
        Normal_Video ;
        GOTOXY( 1,8 ) ; clrline ;
    END ; { IF result }
  UNTIL good_input ;
  freq_list[freq_no].im_coef_K1 := temp ;
END ; { Get_IM_Coefficient }
```

```
{////////////////////////////////////////////////////////////////
/ This procedure prompts the user for the tranmitter's
/ final output power amplifier type, checks for valid
/ input, and stores the entry in the frequency list.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////////}

PROCEDURE Get_Amplifier_Type ;

  VAR
    input_char : CHAR ;
    good_input : BOOLEAN ;

BEGIN { Get_Amplifier_Type }
  IF NOT specify_amp_type
    THEN BEGIN
      freq_list[freq_no].im_coef_b := 0 ;
      EXIT ;
    END ; { IF NOT specify_amp_type
  REPEAT
    GOTOXY(1,9) ; Clrs2end ;
    WRITELN('What type of amplifier (Solid state or Tube) ',
      'is used in the final output stage ');
    WRITE('of the transmitter (S or T) ? ');
    Cursor( block, noblink, noclick, on );
    READ(KBD,input_char) ;
    good_input := TRUE ;
    CASE UPCASE(input_char) OF
      'S' : freq_list[freq_no].im_coef_b := 30 ;
      'T' : freq_list[freq_no].im_coef_b := 40 ;
      ELSE good_input := FALSE ;
    END ; { CASE UPCASE(input_char) }
  UNTIL good_input ;
END ; { Get_Amplifier_Type }
```

```
{////////////////////////////////////////////////////////////}

BEGIN { Add_Frequency_Data }
  freq_no := no_o_frequencies ;
  REPEAT
    CLRSCR ;
    freq_no := freq_no + 1 ;
    Get_Frequency ;
    IF NOT quit
      THEN BEGIN
        Get_Antenna_No ;
        Get_Usage_Code ;
        Get_Bandwidth ;
        Get_Time_Used ;
        Get_Cable_Loss ;
        IF (freq_list[freq_no].usage = 'T') OR
           (freq_list[freq_no].usage = 'B')
          THEN BEGIN
            Get_Output_Power ;
            Get_IM_Coefficient ;
            Get_Amplifier_Type ;
          END
          ELSE BEGIN
            freq_list[freq_no].output_power := -1 ;
            freq_list[freq_no].im_coef_k1 := -1 ;
            freq_list[freq_no].im_coef_b := -1 ;
        END ; { IF usage = T or B }
    END ; { IF NOT quit }
  UNTIL quit ;
  no_o_frequencies := freq_no - 1 ;
  sort_frequencies ;
END ; { Add_Frequency_Data }
```

```
{////////////////////////////////////////////////////////////
/ This procedure clear the antenna and frequency data
/ from memory.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Clear_Data ;

BEGIN { Clear_Data }
  no_o_frequencies := 0 ;
  no_o_antennas    := 0 ;
END ; { Clear_Data }




{////////////////////////////////////////////////////////////
/ This procedure will return turn off the cursor and return
/ the user to the MAIN MENU.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Return_To_Main_Menu ;

BEGIN { Return_To_Main_Menu }
  Cursor( block, noblink, noclick, off ) ;
  done := TRUE ;
END ; { Return_To_Main_Menu }




{////////////////////////////////////////////////////////////}

BEGIN { Add_Data }
  REPEAT ;
    Display_Add_Menu ;
    done := FALSE ;
    CASE ( Get_Menu_Selection( 4, 4, 29 ) ) OF

      1 : Add_Antenna_Data ;
      2 : Add_Frequency_Data ;
      3 : Clear_Data ;
      4 : Return_To_Main_Menu ;

    END ; { CASE STATEMENT }

  UNTIL done ;

END ; { Add_Data }
```

```
{*************************************************************
* This procedure performs the transmitter intermodulation
* interference analysis.  It first gets a filename from the
* user to store the results of the analysis in and then
* begins the analysis.  Each receive frequency is tested
* for interference in order from the lowest frequency to
* the highest frequency.  A+B, A-B, 2A-B, A+B-C, and
* 3A-2B intermodulation products are calculated.  The
* strengths of intermodulation products falling within the
* bandpass of the receive frequency under test is compared
* to the thermal noise at the front end of the victim
* receiver.  If the strength of the intermodulation product
* exceeds the thermal noise, information identifying the
* frequencies, antennas, and intermodulation product type
* is written to the output file previously specified by the
* user.  (Version 1.00  5 June 1986)
*************************************************************}

OVERLAY PROCEDURE Do_Analysis ;

  VAR
    victim_freq_no      : INTEGER ;
    victim_freq         : REAL ;
    victim_bandwidth    : REAL ;
    victim_noise_level  : REAL ;
    im_freq             : REAL ;
    rcv_im_power        : REAL ;
    amount_exceeded     : REAL ;
    filename            : STRING[30] ;
    output_file         : TEXT ;
    file_not_ready      : BOOLEAN ;
    yes                 : BOOLEAN ;


{*************************************************************
* This procedure displays the output files in the current
* directory, prompts the user for an output filename, and
* prepares the file to accept the analysis output.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Open_Output_File ;
```

```
{////////////////////////////////////////////////////////////
/ This procedure provides instructions to the user as to
/ how to specify an output file for the analysis results.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Display_Instructions;

  VAR
    index : INTEGER ;

BEGIN { Display_Instructions }
  Reverse_Video ;
  Cwrite
    ('YOUR FILENAME MAY CONTAIN UP TO 8 CHARACTERS.', 18 );
  Cwrite('DO NOT INCLUDE A FILE EXTENSION.', 19 ) ;
  Cwrite
    ('Press <RETURN> without entering a filename', 21);
  Cwrite(CONCAT
    ('to return to the main menu without saving the ',
     'data.'), 22);
  Normal_Video ;
  GOTOXY(1,5) ;
  WRITE('These are the output files already on ') ;
  IF path = ''
    THEN WRITE('the default drive')
    ELSE WRITE(path) ;
END ; { Display_Instructions }
```

```
{////////////////////////////////////////////////////////////
/ This procedure prompts the user for the output filename
/ to be used for the analysis results and prepares the file
/ to accept the information.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_Filename ;

  VAR
    result     : BYTE ;
    file_ready : BOOLEAN ;
    size       : INTEGER ;

BEGIN { Get_Filename }
  REPEAT
    GOTOXY( 1, 2 ) ; Clrline ;
    WRITE('What do you want to call your output file ?  ') ;
    Cursor( block, noblink, noclick, on ) ;
    READLN( filename ) ;
    Cursor( block, noblink, noclick, off ) ;
    IF ( LENGTH( filename ) = 0 )
      THEN BEGIN
        file_not_ready := TRUE ;
        EXIT;
    END ; { IF }
    file_not_ready := FALSE ;
    size := pos('.',filename) ;
    IF size <> 0
      THEN filename := COPY(filename,1,size-1) ;
    ASSIGN( output_file,path+filename+'.OUT');
    {$I-} REWRITE( output_file ) {$I+} ;
    result := IOresult ;
    file_ready := ( result = 0 ) ;
    IF (result = $F1) THEN Directory_Full ;
  UNTIL file_ready ;
END ; { Get_Filename }

{////////////////////////////////////////////////////////////}

BEGIN { Open_Output_File }
  CLRSCR;
  Display_Instructions ;
  Display_Directory(7,'????????.OUT') ;
  Get_Filename ;
END ; { Open_Output_File }
```

```
{*************************************************************
* This procedure displays a message on the screen indicating
* which receive frequency is under test for interference and
* also prints a similar message to the output file
* indicating the receive frequency under test and the time
* and date that the analysis of that frequency began.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Announce_New_Victim_Frequency( Antenna_No :
  INTEGER) ;

BEGIN
  GOTOXY( 1,1 ) ; Clrline ;
  WRITELN('Testing for interference to ',victim_freq:7:3,
          ' MHz on antenna #',antenna_no) ;
  WRITELN(output_file,
'* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *') ;
  WRITELN(output_file,'Testing for interference to ',
          victim_freq:7:3,' MHz on antenna #',antenna_no) ;
  WRITELN(output_file,Date,
  '                                          ',Time) ;
  WRITELN(output_file) ;
END ;




{*************************************************************
* This function calculates the base 10 logarithm of         *
* "number".  (Version 1.00  5 June 1986)                    *
*************************************************************}

FUNCTION Log(number : REAL )
  { RETURN } : REAL ;

BEGIN { Log }
     Log := ln(number)/ln(10) ;
END ; { Log }
```

```
{*************************************************************
* This function calculates the thermal noise level, in      *
* dBm, that would be present in a bandwidth of "Bandwidth"  *
* kilohertz if the temperature is assumed to be 290         *
* degrees Kelvin.  (Version 1.00  5 June 1986)              *
*************************************************************}

FUNCTION Thermal_Noise( Bandwidth : REAL )
  { RETURN } : REAL ;

BEGIN { Thermal_Noise }
  Thermal_Noise := 10 * Log( Bandwidth ) - 144 ;
END ; { Thermal_Noise }
```

```
{*************************************************************
* This function calculates the free space propagation loss,*
* in dB, experienced by a signal at "Freq" Mhz over a       *
* distance of "dist" feet.  (Version 1.00  5 June 1986)     *
*************************************************************}

FUNCTION Prop_Loss( dist, Freq : REAL )
  { RETURN } : REAL ;

BEGIN { Prop_Loss }
     Prop_Loss := -37.85 + 20 * log( dist*Freq ) ;
END ; { Prop_Loss }
```

```
{*************************************************************
* This function calculates the distance between two         *
* antennas given their antenna numbers.  It uses the        *
* antenna numbers to get the x,y,z coordinates, in feet,     *
* of the antennas and returns the distance in feet.         *
* (Version 1.00  5 June 1986)                               *
*************************************************************}

FUNCTION Distance( From_Ant_No, To_Ant_No : INTEGER )
  { RETURN } : REAL ;

  VAR
    x1, y1, z1 : REAL ;
    x2, y2, z2 : REAL ;

BEGIN { Distance }
  WITH Ant_List[ Freq_List[ From_Ant_No ].Antenna_No ] DO
  BEGIN { WITH }
    x1 := xcoord ;
    y1 := ycoord ;
    z1 := zcoord ;
  END ; { WITH }

  WITH Ant_List[ Freq_List[To_Ant_No].Antenna_No ] DO
  BEGIN { WITH }
    x2 := xcoord ;
    y2 := ycoord ;
    z2 := zcoord ;
  END ; { WITH }

  distance := SQRT(SQR(x2-x1) + SQR(y2-y1) + SQR(z2-z1)) ;

END ; { Distance }
```

```
{*************************************************************
* This function calculates the power, in dBm, of the        *
* culprit signal appearing at the antenna port of the       *
* threat transmitter given the culprit frequency number     *
* and the threat frequency number.  It uses the frequency   *
* numbers to get the information it needs from the          *
* frequency and antenna lists.  (Version 1.00  5 June 1986)*
*************************************************************}

FUNCTION Culprit_Power( culp_no, thrt_no : INTEGER )
  { RETURN } : REAL ;

  VAR
    Pco : REAL ;
    Lcc : REAL ;
    Gc  : REAL ;
    Lp  : REAL ;
    Gt  : REAL ;
    Ltc : REAL ;

BEGIN { Culprit_Power }

  WITH Freq_List[ culp_no ], Ant_List[ Antenna_No ] DO
  BEGIN { WITH }
    Pco := 10*Log(output_power*1000) ; {Convert Watts to dBm}
    Lcc := cable_loss ;
    Gc  := gain ;
    Lp  := Prop_Loss(distance(thrt_no,culp_no),Frequency) ;
  END ; { WITH }

  WITH Freq_List[ thrt_no ], Ant_List[ Antenna_No ] DO
  BEGIN { WITH }
    Gt  := gain ;
    Ltc := cable_loss ;
  END ; { WITH }

  Culprit_Power := Pco - Lcc + Gc - Lp + Gt - Ltc ;

END ; { Culprit_Power }
```

```
{*************************************************************
* This function calculates the amount of attenuation, in    *
* db, a "Test_Freq"uency, in MHz, would experience when     *
* passing through the output stage of a transmitter tuned   *
* to "Tuned_Freq"uency MHz.                                 *
*************************************************************}

FUNCTION Selectivity( Test_Freq, Tuned_Freq : REAL )
  { RETURN } : REAL ;

  VAR
    Delta      : REAL ;

BEGIN { Selectivity }
  Delta := 10*log(ABS(Test_Freq - Tuned_Freq)/Tuned_Freq) ;
  CASE (Tuned_Freq < 100.0) OF
    TRUE  : BEGIN { CASE TRUE }
              IF Delta <= -16.9897
                THEN Selectivity := 0 ;
              IF (-16.9897 < Delta) AND (Delta <= -13.9794)
                THEN Selectivity := 1.2922*Delta + 21.9456 ;
              IF (-13.9794 < Delta) AND (Delta <= -12.2185)
                THEN Selectivity := 1.4197*Delta + 23.7368 ;
              IF (-12.2185 < Delta) AND (Delta <= -10.0000)
                THEN Selectivity := 1.6272*Delta + 26.2723 ;
              IF (-10.0000 < Delta) AND (Delta <= -6.9897)
                THEN Selectivity := 1.9367*Delta + 29.3668 ;
              IF Delta > -6.9897
                THEN Selectivity := 2.1074*Delta + 30.5600 ;
            END ; { CASE TRUE }

    FALSE : BEGIN { CASE FALSE }
              IF Delta <= -15.2288
                THEN Selectivity := 0 ;
              IF (-15.2288 < Delta) AND (Delta <= -10.0)
                THEN Selectivity := 0.5852*Delta + 8.9122 ;
              IF (-10.0 < Delta) AND (Delta <= -6.9897)
                THEN Selectivity := 1.1062*Delta + 14.1220 ;
              IF Delta > -6.9897
                THEN Selectivity := 1.9210*Delta + 19.8172 ;
            END ; { CASE FALSE }
  END ; { CASE STATEMENT }
END ; { Selectivity }
```

```
{*************************************************************
* This function calculates the power, in dBm, of the        *
* intermodulation product at the antenna port of the         *
* threat transmitter given the culprit frequency numbers,    *
* the threat frequency number, the harmonic numbers          *
* associated with the culprit frequencies, and the           *
* intermodulation product frequency in MHz  It uses the      *
* frequency numbers to get any additional information        *
* needed from the frequency list.                            *
*************************************************************}

FUNCTION TX_IM_Power( c1, c2, t, n, m : INTEGER )
  { RETURN } : REAL ;

  VAR
    Pc1          : REAL ;
    StFc1        : REAL ;
    Pc2          : REAL ;
    StFc2        : REAL ;
    b            : REAL ;
    K1           : REAL ;
    StFim        : REAL ;
    Culp_Freq_1  : REAL ;
    Culp_Freq_2  : REAL ;
    Thrt_Freq    : REAL ;

BEGIN { TX_IM_Power }
  Culp_Freq_1 := Freq_List[c1].Frequency ;
  Culp_Freq_2 := Freq_List[c2].Frequency ;
  Thrt_Freq := Freq_List[t].Frequency ;
  Pc1 := Culprit_Power( c1, t ) ;
  StFc1 := Selectivity( Culp_Freq_1, Thrt_Freq ) ;
  Pc2 := Culprit_Power( c2, t ) ;
  StFc2 := Selectivity( Culp_Freq_2, Thrt_Freq ) ;
  b := 30 ;                                  {Freq_List[t].IM_coef_b ;}
  K1 := 0 ;                                  {Freq_List[t].IM_coef_K1 ;}
  StFim := Selectivity( im_freq, Thrt_Freq ) ;
  TX_IM_Power := n * ( Pc1 - StFc1 ) + m * ( Pc2 - StFc2 ) -
                 ( n + m - 1 ) * b - K1 - StFim ;
END ; { TX_IM_Power }
```

```
{*************************************************************
* This function calculates the power, in dBm, of the        *
* intermodulation product appearing at the antenna port of  *
* the victim receiver given the power, in dBm, of the       *
* intermodulation product at the antenna port of the        *
* threat transmitter, the frequency, in MHz, of the         *
* intermodulation product, the threat frequency number,     *
* and the victim frequency number.  It uses the threat and  *
* victim frequency numbers to get other information needed.*
*************************************************************}

FUNCTION RX_IM_Power(Pim : REAL; thrt_no, vict_no : INTEGER)
  { RETURN } : REAL ;

  VAR
    Ltc : REAL ;
    Gt  : REAL ;
    Lp  : REAL ;
    Gv  : REAL ;
    Lvc : REAL ;

BEGIN { RX_IM_Power }
  Ltc := Freq_List[thrt_no].cable_loss ;
  Gt  := Ant_List[Freq_List[thrt_no].antenna_no].gain ;
  Lp  := Prop_Loss( Distance( thrt_no, vict_no ), im_freq ) ;
  Gv  := Ant_List[Freq_List[vict_no].antenna_no].gain ;
  Lvc := Freq_List[vict_no].cable_loss ;
  RX_IM_Power := Pim - Ltc + Gt - Lp + Gv - Lvc ;
END ; { RX_IM_Power }



{*************************************************************
* This procedure prints the intermodulation interference
* information to the output file.  The frequencies involved,
* their associated antennas, which frequency is considered
* the threat, how the frequencies were combined, and how
* much the strength of the intermodulation product exceeded
* the thermal noise at the front end of the victim receiver
* are all written to the file.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Print_The_Intermodulation_Information( a, b, c
  : REAL; a_No, b_No, c_No, Threat, IM_Type : INTEGER ) ;
```

```
{////////////////////////////////////////////////////////////
/ This procedure writes the information concerning an A+B
/ intermodulation product to the output file.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

  PROCEDURE Write_A_PLUS_B_INFO ;

  BEGIN { Write_A_PLUS_B_INFO }
    WRITE(output_file,'      ',a:7:3,' MHz on antenna # ') ;
    WRITE(output_file,Freq_List[a_No].Antenna_No) ;
    IF (Threat = a_No)
      THEN WRITELN(output_file,' the threat')
      ELSE WRITELN(output_file) ;
    WRITE(output_file,'    + ',b:7:3,' MHz on antenna # ') ;
    WRITE(output_file,Freq_List[b_No].Antenna_No) ;
    IF (Threat = b_No)
      THEN WRITELN(output_file,' the threat')
      ELSE WRITELN(output_file) ;
    WRITELN(output_file,'-------------') ;
  END ; { Write_A_PLUS_B_INFO }




{////////////////////////////////////////////////////////////
/ This procedure writes the information concerning an A-B
/ intermodulation product to the output file.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

  PROCEDURE Write_A_MINUS_B_INFO ;

  BEGIN { Write_A_MINUS_B_INFO }
    WRITE(output_file,'      ',a:7:3,' MHz on antenna # ') ;
    WRITE(output_file,Freq_List[a_No].Antenna_No) ;
    IF (Threat = a_No)
      THEN WRITELN(output_file,' the threat')
      ELSE WRITELN(output_file) ;
    WRITE(output_file,'    - ',b:7:3,' MHz on antenna # ') ;
    WRITE(output_file,Freq_List[b_No].Antenna_No) ;
    IF (Threat = b_No)
      THEN WRITELN(output_file,' the threat')
      ELSE WRITELN(output_file) ;
    WRITELN(output_file,'-------------') ;
  END ; { Write_A_MINUS_B_INFO }
```

```
{////////////////////////////////////////////////////////////
/ This procedure writes the information concerning an 2A-B
/ intermodulation product to the output file.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

  PROCEDURE Write_2A_MINUS_B_INFO ;

  BEGIN { Write_2A_MINUS_B_INFO }
    WRITE(output_file,'  2 * ',a:7:3,' MHz on antenna # ') ;
    WRITE(output_file,Freq_List[a_No].Antenna_No) ;
    IF (Threat = a_No)
      THEN WRITELN(output_file,' the threat')
      ELSE WRITELN(output_file) ;
    WRITE(output_file,'    - ',b:7:3,' MHz on antenna # ') ;
    WRITE(output_file,Freq_List[b_No].Antenna_No) ;
    IF (Threat = b_No)
      THEN WRITELN(output_file,' the threat')
      ELSE WRITELN(output_file) ;
    WRITELN(output_file,'-------------') ;
  END ; { Write_2A_MINUS_B_INFO }




{////////////////////////////////////////////////////////////
/ This procedure writes the information concerning an A+B-C
/ intermodulation product to the output file.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

  PROCEDURE Write_A_PLUS_B_MINUS_C_INFO ;

  BEGIN { Write_A_PLUS_B_MINUS_C_INFO }
    WRITE(output_file,'      ',a:7:3,' MHz on antenna # ') ;
    WRITE(output_file,Freq_List[a_No].Antenna_No) ;
    IF (Threat = a_No)
      THEN WRITELN(output_file,' the threat')
      ELSE WRITELN(output_file) ;
    WRITE(output_file,'    + ',b:7:3,' MHz on antenna # ') ;
    WRITE(output_file,Freq_List[b_No].Antenna_No) ;
    IF (Threat = b_No)
      THEN WRITELN(output_file,' the threat')
      ELSE WRITELN(output_file) ;
    WRITE(output_file,'    - ',c:7:3,' MHz on antenna # ') ;
    WRITE(output_file,Freq_List[c_No].Antenna_No) ;
    IF (Threat = c_No)
      THEN WRITELN(output_file,' the threat')
      ELSE WRITELN(output_file) ;
    WRITELN(output_file,'-------------') ;
  END ; { Write_A_PLUS_B_MINUS_C_INFO }
```

```
{////////////////////////////////////////////////////////////
/ This procedure writes the information concerning a 3A-2B
/ intermodulation product to the output file.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

  PROCEDURE Write_3A_MINUS_2B_INFO ;

  BEGIN { Write_3A_MINUS_2B_INFO }
    WRITE(output_file,'  3 * ',a:7:3,' MHz on antenna # ') ;
    WRITE(output_file,Freq_List[a_No].Antenna_No) ;
    IF (Threat = a_No)
      THEN WRITELN(output_file,' the threat')
      ELSE WRITELN(output_file) ;
    WRITE(output_file,'- 2 * ',b:7:3,' MHz on antenna # ') ;
    WRITE(output_file,Freq_List[b_No].Antenna_No) ;
    IF (Threat = b_No)
      THEN WRITELN(output_file,' the threat')
      ELSE WRITELN(output_file) ;
    WRITELN(output_file,'-------------') ;
  END ; { Write_3A_MINUS_2B_INFO }




{////////////////////////////////////////////////////////////}

BEGIN { Print_The_Intermodulation_Information }
  CASE IM_Type OF
    1 : WRITE_A_PLUS_B_INFO ;
    2 : WRITE_A_MINUS_B_INFO ;
    3 : WRITE_2A_MINUS_B_INFO ;
    4 : WRITE_A_PLUS_B_MINUS_C_INFO ;
    5 : WRITE_3A_MINUS_2B_INFO ;
  END ; { CASE IM_Type }
  WRITE(output_file,'      ',victim_freq:7:3,' MHz on ') ;
  WRITE(output_file,'antenna # ') ;
  WRITELN(output_file,Freq_List[victim_freq_no].Antenna_No);
  IF amount_exceeded > 0.009
    THEN WRITELN(output_file,'Noise level exceeded by ',
                 amount_exceeded:6:2,' dB.') ;
  WRITELN(output_file) ;

END ; { Print_The_Intermodulation_Information }
```

```
{*************************************************************
* This procedure checks to see if the intermodulation
* product falls within the bandpass of the victim frequency,
* i.e. if the intermodulation product is within the victim
* frequency plus or minus half the victim frequency's
* bandwidth.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Is_The_IM_Frequency_Within_The_Victim_Bandwidth ;

BEGIN { Is_The_IM_Frequency_Within_The_Victim_Bandwidth }
  yes :=
    ABS( victim_freq - im_freq ) <= victim_bandwidth/2000 ;
END ; { Is_The_IM_Frequency_Within_The_Victim_Bandwidth }




{*************************************************************
* This procedure selects transmit frequencies for A and B,
* insures that the culprit and victim frequencies are
* unique, calculates the A+B intermodulation product, checks
* to see if the intermodulation product falls within the
* passband of the victim frequency, tests for sufficient
* strength, and prints the intermodulation interference
* information via calls to other procedures.  Each culprit
* frequency is selected in turn to act as the threat.  In
* other words, this procedure analyzes the frequency
* information for interference to the victim frequency due
* to A+B intermodulation products.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Test_For_A_Plus_B_IM_Interference ;

  VAR
    a, b                : REAL ;
    Culprit             : INTEGER ;
    Culprit_1           : INTEGER ;
    Culprit_2           : INTEGER ;
    Threat              : INTEGER ;
```

```
{////////////////////////////////////////////////////////////
/ This procedure checks to see that the victim and both
/ culprit frequencies are unique.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Are_Victim_And_Both_Culprit_Frequencies_Unique ;

BEGIN { Are_Victim_And_Both_Culprit_Frequencies_Unique }
  a := Freq_List[Culprit_1].Frequency ;
  b := Freq_List[Culprit_2].Frequency ;
  yes :=
    (victim_freq <> a) AND (victim_freq <> b) AND (a <> b) ;
END ; { Are_Victim_And_Both_Culprit_Frequencies_Unique }



{////////////////////////////////////////////////////////////
/ This procedure assigns the first culprit frequency to act
/ as the threat.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Select_Culprit_1_As_The_Threat ;

BEGIN { Select_Culprit_1_As_The_Threat }
  Threat := Culprit_1 ;
  Culprit := Culprit_2 ;
END ; { Select_Culprit_1_As_The_Threat }



{////////////////////////////////////////////////////////////
/ This procedure assigns the second culprit frequency to act
/ as the threat.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Select_Culprit_2_As_The_Threat ;

BEGIN { Select_Culprit_2_As_The_Threat }
  Threat := Culprit_2 ;
  Culprit := Culprit_1 ;
END ; { Select_Culprit_2_As_The_Threat }
```

```
{////////////////////////////////////////////////////////////
/ This procedure checks to see if the strength of the
/ intermodultion product exceeds the thermal noise level
/ at the front end of the victim receiver.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;

  VAR
    Trans_IM_Power    : REAL ;

BEGIN { Does_The_IM_Power_At_The_Victim_Exceed_The_Noise }
  Trans_IM_Power :=
    TX_IM_Power( Culprit, Culprit, Threat, 1, 0 ) ;
  rcv_im_power :=
    RX_IM_POWER( Trans_IM_Power, Threat, victim_freq_no ) ;
  amount_exceeded := rcv_im_power - victim_noise_level ;
  yes := amount_exceeded  > 0.009 ;
END ; { Does_The_IM_Power_At_The_Victim_Exceed_The_Noise }




{////////////////////////////////////////////////////////////
/ This procedure calls the preceeding procedures to test
/ for A+B interference.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Check_For_Interference ;

BEGIN { Check_For_Interference }
  Are_Victim_And_Both_Culprit_Frequencies_Unique ;
  IF yes THEN BEGIN
    im_freq := a + b ;
    Is_The_IM_Frequency_Within_The_Victim_Bandwidth ;
  END ; { IF yes }
  IF yes THEN BEGIN
    Select_Culprit_1_As_The_Threat ;
    Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;
    IF yes
      THEN Print_The_Intermodulation_Information(a, b, 0,
        Culprit_1, Culprit_2, 0, Threat, 1) ;

    Select_Culprit_2_As_The_Threat ;
    Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;
    IF yes
      THEN Print_The_Intermodulation_Information(a, b, 0,
        Culprit_1, Culprit_2, 0, Threat, 1) ;
  END ; { IF }
END ; { Check_For_Interference }
```

```
{////////////////////////////////////////////////////////////}

BEGIN { Test_For_A_Plus_B_IM_Interference }

FOR Culprit_1 := 1 TO (victim_freq_no - 2)
DO IF freq_list[culprit_1].usage <> 'R' THEN
  FOR Culprit_2 := (Culprit_1 + 1) TO (victim_freq_no - 1)
  DO IF freq_list[culprit_2].usage <> 'R' THEN
    Check_For_Interference ;

END ; { Test_For_A_Plus_B_IM_Interference }




{************************************************************
* This procedure selects transmit frequencies for A and B,
* insures that the culprit and victim frequencies are
* unique, calculates the A-B intermodulation product, checks
* to see if the intermodulation product falls within the
* passband of the victim frequency, tests for sufficient
* strength, and prints the intermodulation interference
* information via calls to other procedures.  Each culprit
* frequency is selected in turn to act as the threat.  In
* other words, this procedure analyzes the frequency
* information for interference to the victim frequency due
* to A-B intermodulation products.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Test_For_A_Minus_B_IM_Interference ;

  VAR
    Threat            : INTEGER ;
    Culprit           : INTEGER ;
    Culprit_1         : INTEGER ;
    Culprit_2         : INTEGER ;
    a, b              : REAL ;
```

```
{////////////////////////////////////////////////////////////
/ This procedure checks to see that the victim and both
/ culprit frequencies are unique.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Are_Victim_And_Both_Culprit_Frequencies_Unique ;

BEGIN { Are_Victim_And_Both_Culprit_Frequencies_Unique }
  a := Freq_List[Culprit_1].Frequency ;
  b := Freq_List[Culprit_2].Frequency ;
  yes :=
    (victim_freq <> a) AND (victim_freq <> b) AND (a <> b) ;
END ; { Are_Victim_And_Both_Culprit_Frequencies_Unique }



{////////////////////////////////////////////////////////////
/ This procedure assigns the first culprit frequency to act
/ as the threat.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Select_Culprit_1_As_The_Threat ;

BEGIN { Select_Culprit_1_As_The_Threat }
  Threat := Culprit_1 ;
  Culprit := Culprit_2 ;
END ; { Select_Culprit_1_As_The_Threat }



{////////////////////////////////////////////////////////////
/ This procedure assigns the second culprit frequency to act
/ as the threat.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Select_Culprit_2_As_The_Threat ;

BEGIN { Select_Culprit_2_As_The_Threat }
  Threat := Culprit_2 ;
  Culprit := Culprit_1 ;
END ; { Select_Culprit_2_As_The_Threat }
```

```
{////////////////////////////////////////////////////////////
/ This procedure checks to see if the strength of the
/ intermodultion product exceeds the thermal noise level
/ at the front end of the victim receiver.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;

  VAR
    Trans_IM_Power    : REAL ;

BEGIN { Does_The_IM_Power_At_The_Victim_Exceed_The_Noise }
  Trans_IM_Power :=
    TX_IM_Power( Culprit, Culprit, Threat, 1, 0 ) ;
  rcv_im_power :=
    RX_IM_POWER( Trans_IM_Power, Threat, victim_freq_no ) ;
  amount_exceeded := rcv_im_power - victim_noise_level ;
  yes := amount_exceeded  > 0.009 ;
END ; { Does_The_IM_Power_At_The_Victim_Exceed_The_Noise }
```

```
{////////////////////////////////////////////////////////////
/ This procedure calls the preceeding procedures to test
/ for A-B interference.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Check_For_Interference ;

BEGIN { Check_For_Interference }
  Are_Victim_And_Both_Culprit_Frequencies_Unique ;
  IF yes THEN BEGIN
    im_freq := a - b ;
    Is_The_IM_Frequency_Within_The_Victim_Bandwidth ;
  END ; { IF yes }
  IF yes THEN BEGIN
    Select_Culprit_1_As_The_Threat ;
    Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;
    IF yes
      THEN Print_The_Intermodulation_Information(a, b, 0,
        Culprit_1, Culprit_2, 0, Threat, 2) ;

    Select_Culprit_2_As_The_Threat ;
    Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;
    IF yes
      THEN Print_The_Intermodulation_Information(a, b, 0,
        Culprit_1, Culprit_2, 0, Threat, 2) ;
  END ; { IF }
END ; { Check_For_Interference }




{////////////////////////////////////////////////////////////}

BEGIN { Test_For_A_Minus_B_IM_Interference }

  FOR Culprit_1 := (victim_freq_no + 1) TO No_O_Frequencies
  DO IF freq_list[culprit_1].usage <> 'R' THEN BEGIN
    FOR Culprit_2 := 1 TO (victim_freq_no - 1)
    DO IF freq_list[culprit_2].usage <> 'R' THEN
      Check_For_Interference ;
    FOR Culprit_2 := (victim_freq_no + 1) TO (Culprit_1 - 1)
    DO IF freq_list[culprit_2].usage <> 'R' THEN
      Check_For_Interference ;
  END ; { FOR Culprit_1 IF }

END ; { Test_For_A_Minus_B_IM_Interference }
```

```
{*************************************************************
* This procedure selects transmit frequencies for A and B,
* insures that the culprit and victim frequencies are
* unique, calculates the 2A-B intermodulation product,
* checks to see if the intermodulation product falls within
* the passband of the victim frequency, tests for sufficient
* strength, and prints the intermodulation interference
* information via calls to other procedures.  Each culprit
* frequency is selected in turn to act as the threat.  In
* other words, this procedure analyzes the frequency
* information for interference to the victim frequency due
* to 2A-B intermodulation products.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Test_For_2A_Minus_B_IM_Interference ;

  VAR
    a, b       : REAL ;
    n          : INTEGER ;
    Threat     : INTEGER ;
    Culprit    : INTEGER ;
    Culprit_1  : INTEGER ;
    Culprit_2  : INTEGER ;

{////////////////////////////////////////////////////////////
/ This procedure checks to see that the victim and both
/ culprit frequencies are unique.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Are_Victim_And_Both_Culprit_Frequencies_Unique ;

BEGIN { Are_Victim_And_Both_Culprit_Frequencies_Unique }
  a := Freq_List[Culprit_1].Frequency ;
  b := Freq_List[Culprit_2].Frequency ;
  yes :=
    (victim_freq <> a) AND (victim_freq <> b) AND (a <> b) ;
END ; { Are_Victim_And_Both_Culprit_Frequencies_Unique }
```

```
{////////////////////////////////////////////////////////////
/ This procedure assigns the first culprit frequency to act
/ as the threat.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Select_Culprit_1_As_The_Threat ;

BEGIN { Select_Culprit_1_As_The_Threat }
  Threat  := Culprit_1 ;
  Culprit := Culprit_2 ;
  n := 1 ;
END ; { Select_Culprit_1_As_The_Threat }



{////////////////////////////////////////////////////////////
/ This procedure assigns the second culprit frequency to act
/ as the threat.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Select_Culprit_2_As_The_Threat ;

BEGIN { Select_Culprit_2_As_The_Threat }
  Threat  := Culprit_2 ;
  Culprit := Culprit_1 ;
  n := 2 ;
END ; { Select_Culprit_2_As_The_Threat }
```

```
{////////////////////////////////////////////////////////////
/ This procedure checks to see if the strength of the
/ intermodultion product exceeds the thermal noise level
/ at the front end of the victim receiver.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;

  VAR
    Trans_IM_Power    : REAL ;

BEGIN { Does_The_IM_Power_At_The_Victim_Exceed_The_Noise }
  Trans_IM_Power :=
    TX_IM_Power( Culprit, Culprit, Threat, n, 0 ) ;
  rcv_im_power :=
    RX_IM_POWER( Trans_IM_Power, Threat, victim_freq_no ) ;
  amount_exceeded := rcv_im_power - victim_noise_level ;
  yes := amount_exceeded  > 0.009 ;
END ; { Does_The_IM_Power_At_The_Victim_Exceed_The_Noise }




{////////////////////////////////////////////////////////////
/ This procedure calls the preceeding procedures to test
/ for 2A-B interference.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Check_For_Interference ;

BEGIN { Check_For_Interference }
  Are_Victim_And_Both_Culprit_Frequencies_Unique ;
  IF yes THEN BEGIN
    im_freq := 2*a - b ;
    Is_The_IM_Frequency_Within_The_Victim_Bandwidth ;
  END ; { IF yes }
  IF yes THEN BEGIN
    Select_Culprit_1_As_The_Threat ;
    Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;
    IF yes
      THEN Print_The_Intermodulation_Information(a, b, 0,
        Culprit_1, Culprit_2, 0, Threat, 3) ;

    Select_Culprit_2_As_The_Threat ;
    Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;
    IF yes
      THEN Print_The_Intermodulation_Information(a, b, 0,
        Culprit_1, Culprit_2, 0, Threat, 3) ;
  END ; { IF yes }
END ; { Check_For_Interference }
```

```
{////////////////////////////////////////////////////////////}

BEGIN { Test_For_2A_Minus_B_IM_Interference }

  FOR Culprit_1 := 1 TO victim_freq_no - 1
  DO IF freq_list[culprit_1].usage <> 'R' THEN BEGIN
    FOR Culprit_2 := 1 TO Culprit_1 - 1
    DO IF freq_list[culprit_2].usage <> 'R' THEN
      Check_For_Interference ;
    FOR Culprit_2 := Culprit_1 + 1 TO victim_freq_no - 1
    DO IF freq_list[culprit_2].usage <> 'R' THEN
      Check_For_Interference ;
    FOR Culprit_2 := victim_freq_no + 1 TO No_O_Frequencies
    DO IF freq_list[culprit_2].usage <> 'R' THEN
      Check_For_Interference ;
  END ; { FOR Culprit_1 := 1 IF }
  FOR Culprit_1 := victim_freq_no + 1 TO No_O_Frequencies
  DO IF freq_list[culprit_1].usage <> 'R' THEN BEGIN
    FOR Culprit_2 := 1 TO victim_freq_no - 1
    DO IF freq_list[culprit_2].usage <> 'R' THEN
      Check_For_Interference ;
    FOR Culprit_2 := victim_freq_no + 1 TO Culprit_1 - 1
    DO IF freq_list[culprit_2].usage <> 'R' THEN
      Check_For_Interference ;
    FOR Culprit_2 := Culprit_1 + 1 TO No_O_Frequencies
    DO IF freq_list[culprit_2].usage <> 'R' THEN
      Check_For_Interference ;
  END ; { FOR Culprit_1 := victim_freq_no + 1 }

END ; { Test_For_2A_Minus_B_IM_Interference }
```

```
{************************************************************
* This procedure selects transmit frequencies for A, B, and
* C, insures that the culprit and victim frequencies are
* unique, calculates the A+B-C intermodulation product,
* checks to see if the intermodulation product falls within
* the passband of the victim frequency, tests for sufficient
* strength, and prints the intermodulation interference
* information via calls to other procedures.  Each culprit
* frequency is selected in turn to act as the threat.  In
* other words, this procedure analyzes the frequency
* information for interference to the victim frequency due
* to A+B-C intermodulation products.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Test_For_A_Plus_B_Minus_C_IM_Interference ;

  VAR
    a, b, c      : REAL ;
    Threat       : INTEGER ;
    First_Culp   : INTEGER ;
    Second_Culp  : INTEGER ;
    Culprit_1    : INTEGER ;
    Culprit_2    : INTEGER ;
    Culprit_3    : INTEGER ;

{////////////////////////////////////////////////////////////
/ This procedure checks to see that the victim and all
/ three culprit frequencies are unique.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Are_Victim_And_All_Culprit_Frequencies_Unique ;

BEGIN { Are_Victim_And_All_Culprit_Frequencies_Unique }
  a := Freq_List[Culprit_1].Frequency ;
  b := Freq_List[Culprit_2].Frequency ;
  c := Freq_List[Culprit_3].Frequency ;
  yes :=
    (victim_freq <> a) AND (victim_freq <> b) AND
    (victim_freq <> c) AND (a <> b) AND (a <> c) AND
    (b <> c) ;
END ; { Are_Victim_And_All_Culprit_Frequencies_Unique }
```

```
{////////////////////////////////////////////////////////////
/ This procedure assigns the first culprit frequency to act
/ as the threat.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Select_Culprit_1_As_The_Threat ;

BEGIN { Select_Culprit_1_As_The_Threat }
  Threat  := Culprit_1 ;
  First_Culp := Culprit_2 ;
  Second_Culp := Culprit_3 ;
END ; { Select_Culprit_1_As_The_Threat }



{////////////////////////////////////////////////////////////
/ This procedure assigns the second culprit frequency to act
/ as the threat.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Select_Culprit_2_As_The_Threat ;

BEGIN { Select_Culprit_2_As_The_Threat }
  Threat  := Culprit_2 ;
  First_Culp := Culprit_1 ;
  Second_Culp := Culprit_3 ;
END ; { Select_Culprit_2_As_The_Threat }



{////////////////////////////////////////////////////////////
/ This procedure assigns the third culprit frequency to act
/ as the threat.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Select_Culprit_3_As_The_Threat ;

BEGIN { Select_Culprit_3_As_The_Threat }
  Threat  := Culprit_3 ;
  First_Culp := Culprit_1 ;
  Second_Culp := Culprit_2 ;
END ; { Select_Culprit_3_As_The_Threat }
```

```
{////////////////////////////////////////////////////////////
/ This procedure checks to see if the strength of the
/ intermodultion product exceeds the thermal noise level
/ at the front end of the victim receiver.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;

  VAR
    Trans_IM_Power    : REAL ;

BEGIN { Does_The_IM_Power_At_The_Victim_Exceed_The_Noise }
  Trans_IM_Power := TX_IM_Power( First_Culp, Second_Culp,
                                 Threat, 1, 1 ) ;
  rcv_im_power := RX_IM_POWER( Trans_IM_Power, Threat,
                               victim_freq_no ) ;
  amount_exceeded := rcv_im_power - victim_noise_level ;
  yes := amount_exceeded  > 0.009 ;
END ; { Does_The_IM_Power_At_The_Victim_Exceed_The_Noise }
```

```
{////////////////////////////////////////////////////////////
/ This procedure calls the preceeding procedures to test
/ for A+B-C interference.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Check_For_Interference ;

BEGIN { Check_For_Interference }
  Are_Victim_And_All_Culprit_Frequencies_Unique ;
  IF yes THEN BEGIN
    im_freq := a + b - c ;
    Is_The_IM_Frequency_Within_The_Victim_Bandwidth ;
  END ; { IF yes }
  IF yes THEN BEGIN
    Select_Culprit_1_As_The_Threat ;
    Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;
    IF yes
      THEN Print_The_Intermodulation_Information(a, b, c,
        Culprit_1, Culprit_2, Culprit_3, Threat, 4) ;

    Select_Culprit_2_As_The_Threat ;
    Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;
    IF yes
      THEN Print_The_Intermodulation_Information(a, b, c,
        Culprit_1, Culprit_2, Culprit_3, Threat, 4) ;

    Select_Culprit_3_As_The_Threat ;
    Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;
    IF yes
      THEN Print_The_Intermodulation_Information(a, b, c,
        Culprit_1, Culprit_2, Culprit_3, Threat, 4) ;
  END ; { IF }
END ; { Check_For_Interference }
```

```
{////////////////////////////////////////////////////////////
/ This procedure checks for A+B-C interference with the
/ conditions that V<A<B<C.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE  Loop_1 ;

BEGIN { Loop_1 }

  FOR Culprit_1 := victim_freq_no + 1 TO No_O_Frequencies - 2
  DO IF freq_list[culprit_1].usage <> 'R' THEN
    FOR Culprit_2 := Culprit_1 + 1 TO No_O_Frequencies - 1
    DO IF freq_list[culprit_2].usage <> 'R' THEN
      FOR Culprit_3 := Culprit_2 + 1 TO No_O_Frequencies
      DO IF freq_list[culprit_3].usage <> 'R' THEN
        Check_For_Interference ;

END ; { Loop_1 }
```

```
{////////////////////////////////////////////////////////////
/ This procedure checks for A+B-C interference with the
/ conditions that A<V<C<B.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE  Loop_2 ;

BEGIN { Loop_2 }

  FOR Culprit_1 := 1 TO victim_freq_no - 1
  DO IF freq_list[culprit_1].usage <> 'R' THEN
    FOR Culprit_3 := victim_freq_no + 1 TO No_O_Frequencies - 1
    DO IF freq_list[culprit_3].usage <> 'R' THEN
      FOR Culprit_2 := Culprit_3 + 1 TO No_O_Frequencies
      DO IF freq_list[culprit_2].usage <> 'R' THEN
        Check_For_Interference ;

END ; { Loop_2 }
```

```
{////////////////////////////////////////////////////////////
/ This procedure checks for A+B-C interference with the
/ conditions that A<C<V<B.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE  Loop_3 ;  { Search with A<C<V<B }

BEGIN { Loop_3 }

  FOR Culprit_1 := 1 TO victim_freq_no - 2
  DO IF freq_list[culprit_1].usage <> 'R' THEN
    FOR Culprit_3 := Culprit_1 + 1 TO victim_freq_no - 1
    DO IF freq_list[culprit_3].usage <> 'R' THEN
      FOR Culprit_2 := victim_freq_no + 1 TO No_O_Frequencies
      DO IF freq_list[culprit_2].usage <> 'R' THEN
        Check_For_Interference ;

END ; { Loop_3 }



{////////////////////////////////////////////////////////////
/ This procedure checks for A+B-C interference with the
/ conditions that C<B<A<V.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE  Loop_4 ;  { Search with C<B<A<V }

BEGIN { Loop_4 }

  FOR Culprit_3 := 1 TO victim_freq_no - 3
  DO IF freq_list[culprit_3].usage <> 'R' THEN
    FOR Culprit_2 := Culprit_3 + 1 TO victim_freq_no - 2
    DO IF freq_list[culprit_2].usage <> 'R' THEN
      FOR Culprit_1 := Culprit_2 + 1 TO victim_freq_no - 1
      DO IF freq_list[culprit_1].usage <> 'R' THEN
        Check_For_Interference ;

END ; { Loop_4 }
```

```
{////////////////////////////////////////////////////////////}

BEGIN { Test_For_A_Plus_B_Minus_C_IM_Interference }
  Loop_1 ;
  Loop_2 ;
  Loop_3 ;
  Loop_4 ;
END ; { Test_For_A_Plus_B_Minus_C_IM_Interference }



{************************************************************
* This procedure selects transmit frequencies for A and B,
* insures that the culprit and victim frequencies are
* unique, calculates the 3A-2B intermodulation product,
* checks to see if the intermodulation product falls within
* the passband of the victim frequency, tests for sufficient
* strength, and prints the intermodulation interference
* information via calls to other procedures.  Each culprit
* frequency is selected in turn to act as the threat.  In
* other words, this procedure analyzes the frequency
* information for interference to the victim frequency due
* to 3A-2B intermodulation products.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Test_For_3A_Minus_2B_IM_Interference ;

  VAR
    a, b       : REAL ;
    n          : INTEGER ;
    Threat     : INTEGER ;
    Culprit    : INTEGER ;
    Culprit_1  : INTEGER ;
    Culprit_2  : INTEGER ;
```

```
{////////////////////////////////////////////////////////////
/ This procedure checks to see that the victim and both
/ culprit frequencies are unique.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Are_Victim_And_Both_Culprit_Frequencies_Unique ;

BEGIN { Are_Victim_And_Both_Culprit_Frequencies_Unique }
  a := Freq_List[Culprit_1].Frequency ;
  b := Freq_List[Culprit_2].Frequency ;
  yes :=
    (victim_freq <> a) AND (victim_freq <> b) AND (a <> b) ;
END ; { Are_Victim_And_Both_Culprit_Frequencies_Unique }



{////////////////////////////////////////////////////////////
/ This procedure assigns the first culprit frequency to act
/ as the threat.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Select_Culprit_1_As_The_Threat ;

BEGIN { Select_Culprit_1_As_The_Threat }
  Threat  := Culprit_1 ;
  Culprit := Culprit_2 ;
  n := 2 ;
END ; { Select_Culprit_1_As_The_Threat }



{////////////////////////////////////////////////////////////
/ This procedure assigns the second culprit frequency to act
/ as the threat.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Select_Culprit_2_As_The_Threat ;

BEGIN { Select_Culprit_2_As_The_Threat }
  Threat  := Culprit_2 ;
  Culprit := Culprit_1 ;
  n := 3 ;
END ; { Select_Culprit_2_As_The_Threat }
```

```
{////////////////////////////////////////////////////////////
/ This procedure checks to see if the strength of the
/ intermodultion product exceeds the thermal noise level
/ at the front end of the victim receiver.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;

  VAR
    Trans_IM_Power    : REAL ;

BEGIN { Does_The_IM_Power_At_The_Victim_Exceed_The_Noise }
  Trans_IM_Power :=
    TX_IM_Power( Culprit, Culprit, Threat, n, 0 ) ;
  rcv_im_power :=
    RX_IM_POWER( Trans_IM_Power, Threat, victim_freq_no ) ;
  amount_exceeded := rcv_im_power - victim_noise_level ;
  yes := amount_exceeded  > 0.009 ;
END ; { Does_The_IM_Power_At_The_Victim_Exceed_The_Noise }




{////////////////////////////////////////////////////////////
/ This procedure calls the preceeding procedures to test
/ for 3A-2B interference.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Check_For_Interference ;

BEGIN { Check_For_Interference }
  Are_Victim_And_Both_Culprit_Frequencies_Unique ;
  IF yes THEN BEGIN
    im_freq := 3*a - 2*b ;
    Is_The_IM_Frequency_Within_The_Victim_Bandwidth ;
  END ; { IF yes }
  IF yes THEN BEGIN
    Select_Culprit_1_As_The_Threat ;
    Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;
    IF yes
      THEN Print_The_Intermodulation_Information(a, b, 0,
        Culprit_1, Culprit_2, 0, Threat, 5) ;

    Select_Culprit_2_As_The_Threat ;
    Does_The_IM_Power_At_The_Victim_Exceed_The_Noise ;
    IF yes
      THEN Print_The_Intermodulation_Information(a, b, 0,
        Culprit_1, Culprit_2, 0, Threat, 5) ;
  END ; { IF }
END ; { Check_For_Interference }
```

```
{////////////////////////////////////////////////////////////}

BEGIN { Test_For_3A_Minus_2B_IM_Interference }

  FOR culprit_1 := 1 TO no_o_frequencies
    DO IF freq_list[culprit_1].usage <> 'R' THEN
    FOR culprit_2 := 1 TO no_o_frequencies
      DO IF freq_list[culprit_2].usage <> 'R' THEN
        Check_For_Interference ;

END ; { Test_For_3A_Minus_2B_IM_Interference }



{////////////////////////////////////////////////////////////}

BEGIN { Do_Analysis }
  Open_Output_File ;
  IF file_not_ready THEN EXIT ;
  CLRSCR ;
  FOR victim_freq_no := 1 TO No_O_Frequencies DO BEGIN
    WITH Freq_List[victim_freq_no],Ant_List[antenna_no] DO
    IF usage <> 'T'
      THEN BEGIN
        victim_freq := frequency ; { In MHz }
        victim_bandwidth := bandwidth ; { In kHz }
        victim_noise_level :=
          Thermal_Noise(victim_bandwidth); { In dBm }
        Announce_New_Victim_Frequency( Antenna_No ) ;
        Test_For_A_Plus_B_IM_Interference ;
        Test_For_A_Minus_B_IM_Interference ;
        Test_For_2A_Minus_B_IM_Interference ;
        Test_For_A_Plus_B_Minus_C_IM_Interference ;
        Test_For_3A_Minus_2B_IM_Interference ;
      END ; { IF usage }
  END ; { FOR victim_freq_no }
  CLOSE( output_file ) ;
END ; { Do_Analysis }
```

```
{*************************************************************
* The following variable types, constants, and variables are
* global to the entire program.  (Version 1.00  5 June 1986)
*************************************************************}

TYPE
  Antenna_info = RECORD
                    xcoord : REAL ;
                    ycoord : REAL ;
                    zcoord : REAL ;
                    gain   : REAL ;
                 END ;

  Frequency_info = RECORD
                     frequency      : REAL ;
                     antenna_no     : INTEGER ;
                     usage          : CHAR ;    { R, T, B }
                     bandwidth      : REAL ;
                     output_power   : REAL ;
                     time           : CHAR ;    { D, N, B, C }
                     cable_loss     : REAL ;
                     im_coef_K1     : REAL ;
                     im_coef_b      : REAL ;
                   END;

  iostring = STRING[255] ;
  string60 = STRING[60] ;
  string7  = STRING[7] ;

  Char12arr = ARRAY[1..12] OF CHAR ;
  Char60arr = ARRAY[1..60] OF CHAR ;


CONST
  max_no = 200  ;
  esc    = #27  ;
  cr     = #13  ;
  bs     = #8   ;
  DELETE = #127 ;


VAR
  freq_list         : ARRAY[1..max_no] OF Frequency_info ;
  ant_list          : ARRAY[1..max_no] OF Antenna_info ;
  no_o_antennas     : INTEGER ;
  no_o_frequencies  : INTEGER ;
  finished          : BOOLEAN ;
  data_changed      : BOOLEAN ;
  specify_im_coef   : BOOLEAN ;
  specify_amp_type  : BOOLEAN ;
  path_name         : Char60arr ;
  path              : String60 ;
```

```
{************************************************************
* This procedure allows editing the present antenna and
* frequency information.  (Version 1.00  5 June 1986)
************************************************************}

OVERLAY PROCEDURE Edit_Data ;

  VAR
    done          : BOOLEAN ;
    last_ant_row : INTEGER ;

{************************************************************
* This procedure displays an error message on the screen
* indicating to the user that he has made an improper
* input.  (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Bad_Input( bad_data, kind : iostring ) ;

  BEGIN { bad_input }
    Reverse_Video ;
    WRITELN(bad_data,' is not a valid ',kind,'.');
    WRITELN('Please try again.');
    Normal_Video ;
  END ; { bad_input }



{************************************************************
* This procedure displays some instructions on the screen to
* help the user when entering frequency data.  The message
* reminds the user how to get back to the EDIT MENU.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Display_Frequency_Instructions ;

BEGIN { Display_Frequency_Instructions }
  CLRSCR ;
  Reverse_Video ;
  Cwrite('Press <RETURN> without entering a frequency', 22);
  Cwrite('to return to the EDIT MENU.', 23);
  Normal_Video ;
END ; { Display_Frequency_Instructions }
```

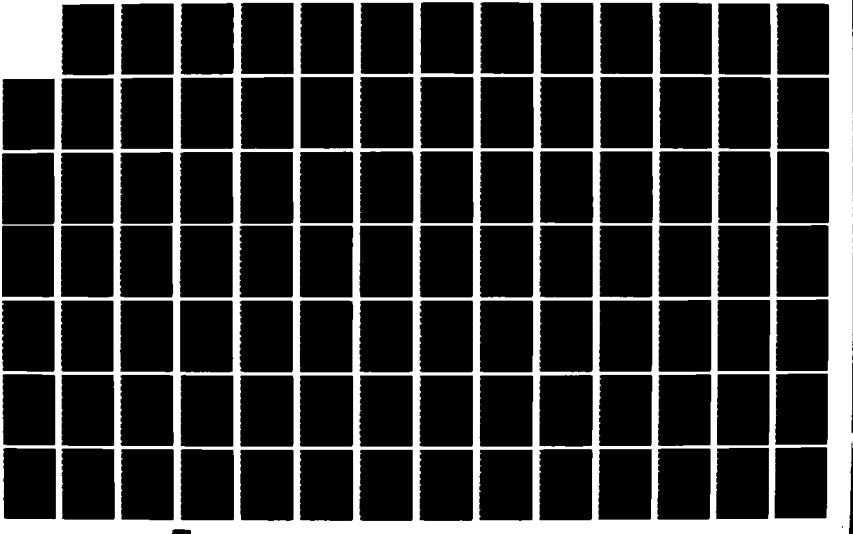
```
{************************************************************
* This function prompts the user for a frequency to edit,
* checks for a valid input, and returns the entry to the
* calling routine.  (Version 1.00  5 June 1986)
************************************************************}

FUNCTION Get_Frequency_To_Edit : REAL ;

  VAR
    input_string : iostring ;
    frequency    : REAL ;
    result       : INTEGER ;
    good_input   : BOOLEAN ;

BEGIN { Get_Frequency_To_Edit }
  REPEAT
    GOTOXY( 1,1 ) ; CLREOL ;
    WRITE('What frequency do you want to edit ?  ') ;
    Cursor( block, noblink, noclick, on );
    READLN(input_string) ;
    IF LENGTH(input_string) = 0
      THEN BEGIN
        Get_Frequency_To_Edit := 0 ;
        EXIT ;
    END ; { IF LENGTH(input_string) }
    VAL( input_string, frequency, result ) ;
    good_input := ( result = 0 ) ;
    IF NOT good_input
      THEN Bad_Input(input_string,'frequency') ;
  UNTIL good_input ;
  Get_Frequency_To_Edit := frequency ;
END ; { Get_Frequency_To_Edit }
```

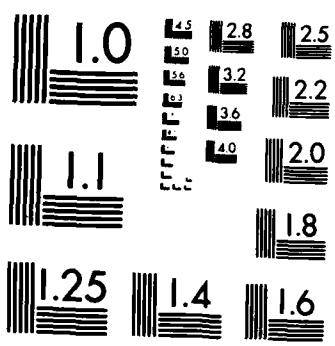
```
{*************************************************************
* This procedure removes a frequency entry from the
* frequency list.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Delete_Frequency( number : INTEGER ) ;

  VAR
    index  : INTEGER ;
    answer : CHAR ;

BEGIN { Delete_Frequency }
  FOR index := number+1 TO no_o_frequencies DO
    freq_list[index-1] := freq_list[index] ;
  no_o_frequencies := no_o_frequencies - 1 ;
  data_changed := TRUE ;
END ; { Delete_Frequency }




{*************************************************************
* This procedure interprets the usage code and displays an
* appropriate message correlating to the usage code on the
* screen.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Usage( usage : CHAR ) ;

BEGIN { Display_Usage }
  CASE usage OF
    'T' : WRITELN('Assigned as a transmit only frequency.');
    'R' : WRITELN('Assigned as a receive only frequency.') ;
    'B' : WRITELN('Assigned as both a transmit and ',
                  'receive frequency.') ;
  END ; { CASE usage }
END ; { Display_Usage }
```

AD-A172 932 ANALYSIS OF TRANSMITTER PRODUCED INTERMODULATION INTERFERENCE IN COLOCATE (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI T J ZUZACK

UNCLASSIFIED JUN 86 AFIT/GE/ENG/86J-3 F/G 20/14 NL

3/4





```
{*************************************************************
* This procedure interprets the time used code and displays
* a corresponding message on the screen.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Time_Used( inchar : CHAR ) ;

BEGIN { Display_Time_Used }
  CASE inchar OF
    'D' : WRITELN('For day time use only.') ;
    'N' : WRITELN('For night time use only.') ;
    'B' : WRITELN('For both day and night time use.') ;
    'C' : WRITELN('For use in a contingency only.') ;
  END ; { CASE inchar }
END ; { Display_Time_Used }



{*************************************************************
* This procedure interprets the b factor (amlifier type
* code) and displays a corresponding message on the screen.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Amplifier_Type( number : REAL ) ;

BEGIN { Display_Amplifier_Type }
  WRITE('Final Output Amplifier Type : ') ;
  IF number = -1 THEN WRITELN('N/A') ;
  IF number = 30 THEN WRITELN('Solid State') ;
  IF number = 40 THEN WRITELN('Tube') ;
END ; { Display_Amplifier_Type }
```

```
{*************************************************************
* This procedure reads the output power from the frequency
* list and print the output power or the message "N/A" if
* the frequency is assigned as receive only.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Output_Power( power : REAL ) ;

BEGIN { Display_Output_Power }
  WRITE('Transmitter Output Power : ');
  IF power = -1
    THEN WRITELN('N/A')
    ELSE WRITELN(power:7:2,' Watts') ;
END ; { Display_Output_Power }



{*************************************************************
* This procedure gets the IM coefficient from the frequency
* list and displays it on the screen or displays the
* message "N/A" if the frequency is assigned as receive
* only.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_IM_Coef( coefficient : REAL ) ;

BEGIN { Display_IM_Coef }
  WRITE('Transmitter Intermodulation Coefficient : ');
  IF coefficient = -1
    THEN WRITELN('N/A')
    ELSE WRITELN(coefficient:7:2,' dB') ;
END ; { Display_IM_Coef }
```

```
{*************************************************************
* This procedure displays all the information available on
* a particular frequency.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Info_For_Frequency( number : INTEGER ) ;

BEGIN { Display_Info_For_Frequency }
  CLRSCR ;
  WITH freq_list[number], ant_list[antenna_no] DO BEGIN
    GOTOXY( 1,2 ) ;
    WRITE('Frequency : ', frequency:7:3,' MHz') ;
    GOTOXY(40,2) ; WRITELN('Antenna Number : ', antenna_no:3) ;
    WRITE('Antenna X Coordinate : ') ;
    WRITE(xcoord:7:2) ;
    GOTOXY(40,3) ; WRITE('Antenna Y Coordinate : ') ;
    WRITELN(ycoord:7:2) ;
    WRITE('Antenna Height : ') ;
    WRITE(zcoord:7:2,' feet') ;
    GOTOXY(40,4) ; WRITE('Antenna Gain : ') ;
    WRITELN(gain:7:2,' dBi') ;
    WRITELN('Bandwidth : ', bandwidth:7:2,' kHz.') ;
    Display_Usage( usage ) ;
    Display_Time_Used( time ) ;
    WRITELN('Cable Loss : ', cable_loss:7:2,' dB') ;
    Display_Output_Power( output_power ) ;
    Display_Amplifier_Type( im_coef_b ) ;
    Display_IM_Coef( im_coef_k1 ) ;
  END ; { WITH freq_list, ant_list }
END ; { Display_Info_For_Frequency }
```

```
{*************************************************************
* This procedure prompts the user to be sure he wants to
* delete the frequency selected and if so it deletes the
* frequency.  Otherwise, the procedure returns to the
* calling routine without disturbing the frequency list.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Delete_This_Frequency( VAR number : INTEGER ) ;

  VAR
    answer : CHAR ;

BEGIN { Delete_This_Frequency } ;
  GOTOXY( 1,14 ) ; Clrs2end ;
  Cwrite('Are You Sure You Want To Delete This Frequency '+
         '(Y/N) ? ',14) ;
  READ(KBD,answer) ;
  IF UPCASE(answer) = 'Y' THEN Delete_Frequency( number ) ;
  IF number > no_o_frequencies
    THEN number := no_o_frequencies ;
  Display_Info_For_Frequency( number ) ;
END ; { Delete_This_Frequency }
```

```
{************************************************************
* This procedure converts the "field" to the x,y character
* coordinates of the frequency field data on the screen
* and moves the cursor to that position.  It also displays
* a reminder to the user about what information is expected
* to be in that field.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE GotoFreqField(field : INTEGER ) ;

BEGIN { GotoFreqField }
  GOTOXY(1,14) ; Clrs2end ;
  Reverse_Video ;
  Cwrite('Use the cursor keys to change fields,  ',19) ;
  Cwrite('    <+> to view the next frequency,    ',20) ;
  Cwrite('    <-> to view the previous frequency,',21) ;
  Cwrite('    <DELETE> to delete the frequency,  ',22) ;
  Cwrite(' or <HOME> to return to the EDIT MENU. ',23) ;
  GOTOXY(1,14) ;
  CASE field OF
    1 : BEGIN
          WRITE('Enter the new frequency in MHz.') ;
          GOTOXY(13,2) ;
        END ;
    2 : BEGIN
          WRITE('Enter the new bandwidth in kHz.') ;
          GOTOXY(13,5) ;
        END ;
    3 : BEGIN
          WRITELN('Enter T for a transmit only frequency') ;
          WRITELN('      R for a receive only frequency ') ;
          WRITE('   or B if both transmit and receive.') ;
          GOTOXY(13,6) ;
        END ;
    4 : BEGIN
          WRITELN('Enter D if used only during the day,',
                  Spc(7)) ;
          WRITELN('      N if used only during the night,',
                  Spc(5)) ;
          WRITELN('      B if used both day and night,',
                  Spc(8)) ;
          WRITELN('   or C if used only during a ',
                  'contingency.') ;
          GOTOXY(1,7) ;
        END ;
    5 : BEGIN
          WRITE('Enter the new cable loss in dB.') ;
          GOTOXY(14,8) ;
        END ;
```

```
      6 : BEGIN
            WRITE('Enter the new output power in Watts.') ;
            GOTOXY(28,9) ;
          END ;
      7 : BEGIN
            WRITELN('Enter S for a solid state output ',
                    'amplifier') ;
            WRITELN('   or T for a tube type.',Spc(19)) ;
            GOTOXY(31,10) ;
          END ;
      8 : BEGIN
            WRITE('Enter the new intermodulation coefficient',
                  ' in dB.') ;
            GOTOXY(43,11) ;
          END ;
      9 : BEGIN
            WRITE('Enter the new antenna number.') ;
            GOTOXY(57,2) ;
          END ;
    END ; { CASE }
    Normal_Video ;
  END ; { GotoFreqField }



  {*************************************************************
  * This procedure displays some instructions on the screen
  * to remind the user how he may abort his present editing
  * without corrupting the data that was there when he
  * started.  (Version 1.00  5 June 1986)
  *************************************************************}

  PROCEDURE Display_Edit_Instructions( instring : iostring ) ;

  BEGIN { Display_Edit_Instructions }
    Reverse_Video ;
    Cwrite('Press <RETURN> without entering a'+instring, 22) ;
    Cwrite('to restore the previous '+instring+'.', 23);
    Normal_Video ;
  END ; { Display_Edit_Instructions }
```

```
{*************************************************************
* This procedure displays an error message indicating that
* the user has made an improper input.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Edit_Error( VAR bad_data : string7 ;
                                  kind     : iostring ) ;

BEGIN { Display_Edit_Error }
  Reverse_Video ;
  Cwrite(bad_data+' is not a valid '+kind,17) ;
  Cwrite('Please try again.',18) ;
  Normal_Video ;
  bad_data := '' ;
END ; { Display_Edit_Error }
```

```
{*************************************************************
* This procedure processes the <BACKSPACE> key so that the
* previous entry is erased and the cursor is moved back one
* space.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Backspace( VAR instring : string7 ) ;

  VAR
    string_length : INTEGER ;

BEGIN { Backspace }
  string_length := LENGTH(instring)-1 ;
  IF string_length < 1
    THEN instring := ''
    ELSE instring := COPY(instring,1,string_length) ;
END ; { Backspace }
```

```
{************************************************************
* This procedure processes keystrokes when the cursor is in
* the frequency field.  (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Get_Frequency( first_character  : CHAR;
                         frequency_number : INTEGER ) ;

  VAR
    temp          : REAL ;
    result, num   : INTEGER ;
    input_string  : string7 ;
    input_done    : BOOLEAN ;
    next_char     : CHAR ;
    good_input    : BOOLEAN ;

BEGIN { Get_Frequency }
  IF (first_character < '0') OR (first_character > '9')
    THEN EXIT ;
  GOTOXY( 1,14 ) ; Clrs2end ;
  Display_Edit_Instructions(' frequency') ;
  input_string := first_character ;
  REPEAT
    REPEAT
      input_done := FALSE ;
      GOTOXY( 13,2 ) ; WRITE('       ') ;
      GOTOXY( 13,2 ) ; WRITE(input_string) ;
      READ(KBD,next_char) ;
      CASE next_char OF
        cr        : input_done := TRUE ;
        bs        : Backspace( input_string ) ;
        '0'..'9' : input_string := input_string+next_char ;
        '.'       : input_string := input_string+next_char ;
      END ; { CASE next_char }
    UNTIL input_done ;
    IF input_string = ''
      THEN BEGIN
        GOTOXY( 13,2 ) ;
        WRITE(freq_list[frequency_number].frequency:7:3) ;
        EXIT ;
      END ; { IF }
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND (temp <= 999.999)
                  AND (temp >= 0) ;
    IF NOT good_input
      THEN Display_Edit_Error(input_string,'frequency.') ;
  UNTIL good_input ;
  data_changed := TRUE ;
  GOTOXY( 13,2 ) ; WRITE(temp:7:3) ;
  freq_list[frequency_number].frequency := temp ;
  Sort_Frequencies ;
END ; { Get_Frequency }
```

```
{************************************************************
* This procedure processes keystrokes when the cursor is in
* the bandwidth field.  (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Get_Bandwidth( first_character  : CHAR;
                         frequency_number : INTEGER ) ;

  VAR
    temp         : REAL ;
    result       : INTEGER ;
    input_string : string7 ;
    input_done   : BOOLEAN ;
    next_char    : CHAR ;
    good_input   : BOOLEAN ;
    num          : INTEGER ;

BEGIN { Get_Bandwidth }
  IF (first_character < '0') OR (first_character > '9')
    THEN EXIT ;
  GOTOXY( 1,14 ) ; Clrs2end ;
  Display_Edit_Instructions(' bandwidth') ;
  input_string := first_character ;
  REPEAT
    REPEAT
      input_done := FALSE ;
      GOTOXY( 13,5 ) ; WRITE('       ') ;
      GOTOXY( 13,5 ) ; WRITE(input_string) ;
      READ(KBD,next_char) ;
      CASE next_char OF
        cr       : input_done := TRUE ;
        bs       : Backspace(input_string) ;
        '0'..'9' : input_string := input_string+next_char ;
        '.'      : input_string := input_string+next_char ;
      END ; { CASE next_char }
    UNTIL input_done ;
    IF input_string = ''
      THEN BEGIN
        GOTOXY( 13,5 ) ;
        WRITE(freq_list[frequency_number].bandwidth:7:2) ;
        EXIT ;
      END ; { IF }
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND (temp <= 999.999)
                  AND (temp >= 0) ;
    IF NOT good_input
      THEN Display_Edit_Error(input_string,'bandwidth.') ;
  UNTIL good_input ;
  data_changed := TRUE ;
  GOTOXY( 13,5 ) ; WRITE(temp:7:2) ;
  freq_list[frequency_number].bandwidth := temp ;
END ; { Get_Bandwidth }
```

```
{************************************************************
* This procedure processes keystrokes when the cursor is in
* the usage code field.  (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Get_Tx_Rx_Or_Both( inchar           : CHAR;
                             frequency_number : INTEGER ) ;

  VAR
    change : BOOLEAN ;

BEGIN { Get_Tx_Rx_Or_Both }
  WITH freq_list[frequency_number] DO BEGIN
    change := TRUE ;
    CASE UPCASE(inchar) OF
      'T' : BEGIN
              usage := 'T' ;
              IF im_coef_b = -1 THEN im_coef_b := 30 ;
              IF im_coef_k1 = -1 THEN im_coef_k1 := 0 ;
              IF output_power = -1
                THEN output_power := 0.0001 ;
            END ;
      'R' : BEGIN
              usage := 'R' ;
              im_coef_b := -1 ;
              im_coef_k1 := -1 ;
              output_power := -1 ;
            END ;
      'B' : BEGIN
              usage := 'B' ;
              IF im_coef_b = -1 THEN im_coef_b := 30 ;
              IF im_coef_k1 = -1 THEN im_coef_k1 := 0 ;
              IF output_power = -1
                THEN output_power := 0.0001 ;
            END ;
      ELSE change := FALSE ;
    END ; { CASE inchar }
    IF change
      THEN BEGIN
        data_changed := TRUE ;
        GOTOXY(1,6) ; CLREOL ; Display_Usage( usage ) ;
        GOTOXY( 1,9 ) ; CLREOL ;
        Display_Output_Power( output_power ) ;
        GOTOXY( 1,10 ) ; CLREOL ;
        Display_Amplifier_Type( im_coef_b ) ;
        GOTOXY( 1,11 ) ; CLREOL ;
        Display_IM_Coef( im_coef_k1 ) ;
    END ; { IF change }
  END ; { WITH freq_list }
END ; { Get_Tx_Rx_Or_Both }
```

```
{*************************************************************
* This procedure processes keystrokes when the cursor is in
* the time used field.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Get_Day_Night_Or_Both( inchar         : CHAR;
                                 frequency_number : INTEGER ) ;

  VAR
    change : BOOLEAN ;

BEGIN { Get_Day_Night_Or_Both }
  WITH freq_list[frequency_number] DO BEGIN
    change := TRUE ;
    CASE UPCASE(inchar) OF
      'D' : time := 'D' ;
      'N' : time := 'N' ;
      'B' : time := 'B' ;
      'C' : time := 'C' ;
      ELSE change := FALSE ;
    END ; { CASE inchar }
    GOTOXY(1,7) ; CLREOL ;
    Display_Time_Used( time ) ;
    IF change THEN data_changed := TRUE ;
  END ; { WITH freq_list }
END ; { Get_Day_Night_Or_Both }
```

```
{************************************************************
* This procedure processes keystrokes when the cursor is in
* the cable loss field.  (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Get_Cable_Loss( first_character  : CHAR;
                          frequency_number : INTEGER ) ;

  VAR
    temp         : REAL ;
    result       : INTEGER ;
    input_string : string7 ;
    input_done   : BOOLEAN ;
    next_char    : CHAR ;
    good_input   : BOOLEAN ;
    num          : INTEGER ;

BEGIN { Get_Cable_Loss }
  IF (first_character < '0') OR (first_character > '9')
    THEN EXIT ;
  GOTOXY( 1,14 ) ; Clrs2end ;
  Display_Edit_Instructions('cable loss') ;
  input_string := first_character ;
  REPEAT
    REPEAT
      input_done := FALSE ;
      GOTOXY( 14,8 ) ; WRITE('       ') ;
      GOTOXY( 14,8 ) ; WRITE(input_string) ;
      READ(KBD,next_char) ;
      CASE next_char OF
        cr       : input_done := TRUE ;
        bs       : Backspace(input_string) ;
        '0'..'9' : input_string := input_string+next_char ;
        '.'      : input_string := input_string+next_char ;
      END ; { CASE next_char }
    UNTIL input_done ;
    IF input_string = ''
      THEN BEGIN
        GOTOXY( 14,8 ) ;
        WRITE(freq_list[frequency_number].cable_loss:7:2) ;
        EXIT ;
      END ; { IF }
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND (temp <= 999.999)
                  AND (temp >= 0) ;
    IF NOT good_input
      THEN Display_Edit_Error(input_string,'cable loss.') ;
  UNTIL good_input ;
  data_changed := TRUE ;
  GOTOXY( 14,8 ) ; WRITE(temp:7:2) ;
  freq_list[frequency_number].cable_loss := temp ;
END ; { Get_Cable_Loss }
```

```
{*************************************************************
* This procedure processes keystrokes when the cursor is in
* the transmitter output power field.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Get_Output_Power( first_character  : CHAR;
                            frequency_number : INTEGER ) ;

  VAR
    temp          : REAL ;
    result, num   : INTEGER ;
    input_string  : string7 ;
    input_done    : BOOLEAN ;
    next_char     : CHAR ;
    good_input    : BOOLEAN ;

BEGIN { Get_Output_Power }
  IF ((first_character < '0') OR (first_character > '9')) AND
     (first_character <> '.') THEN EXIT ;
  GOTOXY( 1,14 ) ; Clrs2end ;
  Display_Edit_Instructions('n output power') ;
  input_string := first_character ;
  REPEAT
    REPEAT
      input_done := FALSE ;
      GOTOXY( 28,9 ) ; WRITE('       ') ;
      GOTOXY( 28,9 ) ; WRITE(input_string) ;
      READ(KBD,next_char) ;
      CASE next_char OF
        cr        : input_done := TRUE ;
        bs        : Backspace(input_string) ;
        '0'..'9' : input_string := input_string+next_char ;
        '.'       : input_string := input_string+next_char ;
      END ; { CASE next_char }
    UNTIL input_done ;
    IF input_string = ''
      THEN BEGIN
        GOTOXY( 28,9 ) ;
        WRITE(freq_list[frequency_number].output_power:7:2);
        EXIT ;
      END ; { IF }
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND (temp <= 999.999)
                  AND (temp >= 0) ;
    IF NOT good_input
      THEN Display_Edit_Error(input_string,'output power.');
  UNTIL good_input ;
  data_changed := TRUE ;
  GOTOXY( 28,9 ) ; WRITE(temp:7:2) ;
  freq_list[frequency_number].output_power := temp ;
END ; { Get_Output_Power }
```

```
{*************************************************************
* This procedure processes keystrokes when the cursor is in
* the transmitter final output power amplifier type field.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Get_Amplifier_Type( inchar           : CHAR;
                              frequency_number : INTEGER ) ;

  VAR
    change : BOOLEAN ;

BEGIN { Get_Amplifier_Type }
  change := TRUE ;
  WITH freq_list[frequency_number] DO BEGIN
    CASE UPCASE(inchar) OF
      'S' : im_coef_b := 30 ;
      'T' : im_coef_b := 40 ;
      ELSE change := FALSE ;
    END ; { CASE inchar }
    GOTOXY( 1,10 ) ; CLREOL ;
    Display_Amplifier_Type( im_coef_b ) ;
    IF change THEN data_changed := TRUE ;
  END ; { WITH freq_list }
END ; { Get_Amplifier_Type }
```

```
{************************************************************
* This procedure processes keystrokes when the cursor is in
* the transmitter IM coefficient field.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Get_Tx_IM_Coef( first_character  : CHAR;
                          frequency_number : INTEGER ) ;

  VAR
    temp          : REAL ;
    result, num   : INTEGER ;
    input_string  : string7 ;
    input_done    : BOOLEAN ;
    next_char     : CHAR ;
    good_input    : BOOLEAN ;

BEGIN { Get_Tx_IM_Coef }
  IF (first_character < '0') OR (first_character > '9')
    THEN EXIT ;
  GOTOXY( 1,14 ) ; Clrs2end ;
  input_done := FALSE ;
  Display_Edit_Instructions('n IM coefficient') ;
  input_string := first_character ;
  REPEAT
    REPEAT
      input_done := FALSE ;
      GOTOXY( 43,11 ) ; WRITE('       ') ;
      GOTOXY( 43,11 ) ; WRITE(input_string) ;
      READ(KBD,next_char) ;
      CASE next_char OF
        cr        : input_done := TRUE ;
        bs        : Backspace(input_string) ;
        '0'..'9' : input_string := input_string+next_char ;
        '.'       : input_string := input_string+next_char ;
      END ; { CASE next_char }
    UNTIL input_done ;
    IF input_string = '' THEN BEGIN
      GOTOXY( 43,11 ) ;
      WRITE(freq_list[frequency_number].im_coef_k1:7:2) ;
      EXIT ;
    END ; { IF }
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND (temp <= 999.999)
                  AND (temp >= 0) ;
    IF NOT good_input THEN
      Display_Edit_Error(input_string,'IM coefficient.') ;
  UNTIL good_input ;
  data_changed := TRUE ;
  GOTOXY( 43,11 ) ; WRITE(temp:7:2) ;
  freq_list[frequency_number].im_coef_k1 := temp ;
END ; { Get_Tx_IM_Coef }
```

```
{************************************************************
* This procedure processes keystrokes when the cursor is in
* the antenna number field.  (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Get_Antenna_Number( first_char       : CHAR;
                              frequency_number : INTEGER ) ;

  VAR
    temp, result, num       : INTEGER ;
    input_string            : string7 ;
    good_input, input_done  : BOOLEAN ;
    next_char               : CHAR ;

BEGIN { Get_Antenna_Number }
  IF (first_char < '0') OR (first_char > '9') THEN EXIT ;
  GOTOXY( 1,14 ) ; Clrs2end ;
  Display_Edit_Instructions(' antenna number') ;
  input_string := first_char ;
  REPEAT
    REPEAT
      input_done := FALSE ;
      GOTOXY( 57,2 ) ; WRITE('       ') ;
      GOTOXY( 57,2 ) ; WRITE(input_string) ;
      READ(KBD,next_char) ;
      CASE next_char OF
        cr        : input_done := TRUE ;
        bs        : Backspace(input_string) ;
        '0'..'9' : input_string := input_string+next_char ;
      END ; { CASE next_char }
    UNTIL input_done ;
    IF input_string = '' THEN BEGIN
      GOTOXY( 57,2 ) ;
      WRITE(freq_list[frequency_number].antenna_no:3) ;
      EXIT ;
    END ; { IF }
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND (temp >= 1) AND
                  (temp <= no_o_antennas) ;
    IF NOT good_input THEN
      Display_Edit_Error(input_string, 'antenna number.') ;
  UNTIL good_input ;
  data_changed := TRUE ;
  GOTOXY( 57,2 ) ; WRITE(temp:3) ;
  freq_list[frequency_number].antenna_no := temp ;
  WITH ant_list[temp] DO BEGIN
    GOTOXY( 24,3 ) ; WRITE(xcoord:7:2) ;
    GOTOXY( 63,3 ) ; WRITE(ycoord:7:2) ;
    GOTOXY( 18,4 ) ; WRITE(zcoord:7:2) ;
    GOTOXY( 55,4 ) ; WRITE(gain:7:2) ;
  END ; { WITH ant_list }
END ; { Get_Antenna_Number }
```

```
{*************************************************************
* This procedure sends the first editing key pressed to the
* appropriate servicing routine. (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Process_Input( key            : CHAR;
                         field, number : INTEGER ) ;

BEGIN { Process_Input }
  CASE field OF
    1 : Get_Frequency( key, number ) ;
    2 : Get_Bandwidth( key, number ) ;
    3 : Get_Tx_Rx_Or_Both( key, number ) ;
    4 : Get_Day_Night_Or_Both( key, number ) ;
    5 : Get_Cable_Loss( key, number ) ;
    6 : Get_Output_Power( key, number ) ;
    7 : Get_Amplifier_Type( key, number ) ;
    8 : Get_Tx_IM_Coef( key, number ) ;
    9 : Get_Antenna_Number( key, number ) ;
  END ; { CASE field }
END ; { Process_Input }



{*************************************************************
* This procedure gets the next frequency for editing and
* displays the associated information on the screen.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Edit_Next_Frequency( VAR number : INTEGER ) ;

BEGIN { Edit_Next_Frequency }
  number := number + 1 ;
  IF number > no_o_frequencies
    THEN number := no_o_frequencies;
  Display_Info_For_Frequency( number ) ;
END ; { Edit_Next_Frequency }
```

```
{*************************************************************
* This procedure gets the prevoius frequency for editing
* and displays the associated information on the screen.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Edit_Previous_Frequency( VAR number : INTEGER ) ;

BEGIN { Edit_Next_Frequency }
  number := number - 1 ;
  IF number < 1 THEN number := 1 ;
  Display_Info_For_Frequency( number ) ;
END ; { Edit_Next_Frequency }



{*************************************************************
* This procedure is the main driver for editing a frequency.
* It allows the user to move the cursor to the various
* fields of frequency data, edit the data, display a
* previous frequency or next frequency for editing, or
* return to the EDIT MENU.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Edit_Frequency( number : INTEGER ) ;

  VAR
    key : CHAR ;
    done_editing : BOOLEAN ;
    field : INTEGER ;
    toss  : CHAR ;
```

```
{////////////////////////////////////////////////////////////
/ This procedure determines if a cursor key or <HOME> has
/ been pressed and then performs the appropriate action.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Process_Esc ;

  VAR
    char2 : CHAR ;

BEGIN { Process_Esc }
  READ(KBD,char2) ;
  WHILE keypressed DO READ(KBD,toss) ;
  CASE char2 OF
    'A' : field := field - 1;     { Cursor Up Key }
    'B' : field := field + 1;     { Cursor Down Key }
    'C' : field := 9 ;            { Cursor Right Key }
    'D' : CASE field OF           { Cursor Left Key }
            1..8 : ;
            9 : field := 1 ;
          END ; { CASE field }
    'H' : done_editing := TRUE ; { Home Key }
  END ; { CASE char2 }
  IF field < 1 THEN field := 1;
  IF field > 9 THEN field := 9;
END ; { Process_Esc }

{////////////////////////////////////////////////////////////}

BEGIN { Edit_Frequency }
  Display_Info_For_Frequency( number ) ;
  field := 1 ;
  done_editing := FALSE ;
  REPEAT
    WHILE keypressed DO READ(KBD,toss) ;
    GotoFreqField(field) ;
    READ(KBD,key) ;
    CASE key OF
      esc    : Process_Esc ;
      '+'    : Edit_Next_Frequency( number ) ;
      '-'    : Edit_Previous_Frequency( number ) ;
      DELETE : Delete_This_Frequency( number ) ;
      ELSE     Process_Input(key, field, number ) ;
    END ; { CASE }
  UNTIL done_editing ;
END ; { Edit_Frequency }
```

```
{*************************************************************
* This procedure prompts the user for a frequency to edit,
* checks for valid input, finds the frequency to edit, and
* then enters TIMAP's edit frequency mode.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Change_Frequency_Data ;

  VAR
    index : INTEGER ;
    freq  : REAL ;
    frequency_found : BOOLEAN ;

BEGIN { Change_Frequency_Data } ;
  Display_Frequency_Instructions ;
  freq := Get_Frequency_To_Edit ;
  IF freq = 0 THEN EXIT ;
  index := 1 ;
  frequency_found := FALSE ;
  REPEAT
    IF (freq_list[index].frequency = freq) OR
       (index > no_o_frequencies)
      THEN frequency_found := TRUE
      ELSE index := index + 1 ;
  UNTIL frequency_found ;
  IF index > no_o_frequencies
    THEN EXIT
    ELSE Edit_Frequency( index ) ;
END ; { Change_Frequency_Data }
```

```
{*************************************************************
* When an antenna is deleted, the antenna numbers above the
* antenna number deleted are decremented by one.  The
* frequency data must reflect these new antenna assignments.
* This procedure updates the frequency list to reflect the
* new antenna assignments.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Update_Antenna_Numbers_In_Freq_List( number
  : INTEGER) ;

  VAR
    index : INTEGER ;

BEGIN { Update_Antenna_Numbers_In_Freq_List }
  FOR index := 1 TO no_o_frequencies DO
    WITH freq_list[index] DO
      IF antenna_no > number THEN
        antenna_no := antenna_no - 1 ;
END ; { Update_Antenna_Numbers_In_Freq_List }



{*************************************************************
* This procedure removes an antenna from the antenna list,
* updates the antenna numbers in the antenna list to remove
* the gap created by the deleted antenna, and updates the
* frequency data to reflect the new antenna assignments.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Delete_Antenna_Number( number : INTEGER ) ;

  VAR
    index : INTEGER ;

BEGIN { Delete_Antenna_Number }
  FOR index := number+1 TO no_o_antennas DO
    ant_list[index-1] := ant_list[index] ;
  no_o_antennas := no_o_antennas - 1 ;
  index := 1 ;
  WHILE index <= no_o_frequencies DO
    IF freq_list[index].antenna_no = number
      THEN Delete_Frequency( index )
      ELSE index := index + 1 ;
  Update_Antenna_Numbers_In_Freq_List( number ) ;
  data_changed := TRUE ;
END ; { Delete_Antenna_Number }
```

```
{*************************************************************
* The antenna data is displayed in columnar format for
* editing.  This procedure provides column headings to
* identify the information in each column.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Antenna_Headings ;

BEGIN { Display_Antenna_Headings }
  CLRSCR ;
  GOTOXY( 1,1 )  ; WRITE('ANTENNA') ;
  GOTOXY( 1,2 )  ; WRITE(' NUMBER') ;
  GOTOXY( 16,1 ) ; WRITE('    X') ;
  GOTOXY( 16,2 ) ; WRITE('COORDINATE') ;
  GOTOXY( 16,3 ) ; WRITE(' (FEET)') ;
  GOTOXY( 30,1 ) ; WRITE('    Y') ;
  GOTOXY( 30,2 ) ; WRITE('COORDINATE') ;
  GOTOXY( 30,3 ) ; WRITE(' (FEET)') ;
  GOTOXY( 45,1 ) ; WRITE('HEIGHT') ;
  GOTOXY( 45,3 ) ; WRITE('(FEET)') ;
  GOTOXY( 60,1 ) ; WRITE('GAIN') ;
  GOTOXY( 60,3 ) ; WRITELN('(dBi)') ;
END ; { Display_Antenna_Headings }



{*************************************************************
* This procedure provides some instructions to remind the
* user how to operate in the antenna editing mode.  The
* instructions are displayed continuously as long as the
* user is in the antenna editing mode.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Antenna_Instructions ;

BEGIN { Display_Antenna_Instructions }
  GOTOXY( 1,17 ) ; Clrs2end ;
  Reverse_Video ;
  Cwrite('Use U or D to scroll the antenna list,  ', 19) ;
  Cwrite('    cursor keys to change fields,      ', 20) ;
  Cwrite('    number keys to enter data,         ', 21) ;
  Cwrite('    <DELETE> to delete the antenna, and ', 22) ;
  Cwrite('    <HOME> to return to the EDIT MENU.  ', 23) ;
  Normal_Video ;
END ; { Display_Antenna_Instructions }
```

```
{*************************************************************
* The antenna data is displayed for 12 antennas at a time
* as long as 12 antennas are available.  This procedure
* displays up to 12 antennas and provides a message when
* the end of the antenna list has been reached.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Antennas_Starting_At_Number( VAR number
                                                : INTEGER ) ;

  VAR
    index       : INTEGER ;

BEGIN { Display_Antennas_Starting_At_Number }
  Cursor( block, noblink, noclick, off ) ;
  GOTOXY( 1,4 ) ; Clrs2end ;
  Display_Antenna_Instructions ;
  index := 1 ;
  WHILE (number <= no_o_antennas) AND
        (index <= 12 ) DO BEGIN
    WITH ant_list[number] DO BEGIN
      GOTOXY(1,index+4)   ; WRITE(number:5) ;
      GOTOXY(15,index+4)  ; WRITE(xcoord:8:2) ;
      GOTOXY(29,index+4)  ; WRITE(ycoord:8:2) ;
      GOTOXY(43,index+4)  ; WRITE(zcoord:8:2) ;
      GOTOXY(57,index+4)  ; WRITE(gain:8:2  ) ;
      number := number + 1 ;
      last_ant_row := index ;
      index := index + 1 ;
    END ; { WITH ant_list }
  END ; { WHILE antenna_no }
  IF number > no_o_antennas THEN BEGIN
    Reverse_Video ;
    Cwrite('This Is The End Of The Antenna '+
           'List.', index+4) ;
    Normal_Video ;
  END ; { IF number }
END ; { Display_Antennas_Starting_At_Number }
```

```
{*************************************************************
* This procedure moves the cursor to a new field within
* the displayed antenna data.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE GotoAntField( row, column : INTEGER ) ;

BEGIN { GotoAntField }
  Cursor( block, noblink, noclick, on ) ;
  GOTOXY((column-1)*14+15,row+4) ;
END ; { GotoAntField }



{*************************************************************
* This procedure will display the next 12 antennas from the
* antenna list.  If there are not 12 more antennas
* available, then the last 12 antennas will be displayed.
* If there are less than 12 antennas, the antennas will be
* displayed beginning with antenna number 1.
*************************************************************}

PROCEDURE Scroll_Screen_Up( VAR num : INTEGER;
                            row, column : INTEGER ) ;

BEGIN { Scroll_Screen_Up }
  IF num > no_o_antennas THEN EXIT ;
  IF num + 11 > no_o_antennas
    THEN BEGIN
      num := no_o_antennas - 11 ;
      IF num < 1 THEN num := 1 ;
  END ; { IF num + 11 }
  Display_Antennas_Starting_At_Number( num ) ;
  GotoAntField( row, column ) ;
END ; { Scroll_Screen_Up }
```

```
{*************************************************************
* This procedure will display the previous  12 antennas
* from the antenna list.  If there are fewer than 12
* previous antennas, the first 12 antennas will be
* displayed.  If there are fewer than 12 antennas, the
* antennas will be displayed beginning with antenna number
* 1.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Scroll_Screen_Down( VAR num : INTEGER;
                              row, column : INTEGER ) ;

BEGIN { Scroll_Screen_Down }
  IF num < 14 THEN EXIT ;
  num := num - 24 ;
  IF num < 1 THEN num := 1 ;
  Display_Antennas_Starting_At_Number( num ) ;
  GotoAntField( row,column ) ;
END ; { Scroll_Screen_Down }
```

```
{************************************************************
* This procedure processes keystrokes when in the
* x-coordionate field of the antenna data.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Get_Xcoord( key                    : CHAR;
                      ant_no, row, column : INTEGER ) ;

  VAR
    temp                     : REAL ;
    result, num              : INTEGER ;
    input_string             : string7 ;
    input_done, good_input : BOOLEAN ;
    next_char                : CHAR ;
    field                    : STRING[14] ;

BEGIN { Get_Xcoord }
  GOTOXY( 1,19 ) ; Clrs2end ;
  Display_Edit_Instructions('n X coordinate') ;
  input_string := key ;
  REPEAT
    REPEAT
      input_done := FALSE ;
      GotoAntField( row,column ) ; WRITE('         ') ;
      GotoAntField( row,column ) ; WRITE(input_string) ;
      READ(KBD,next_char) ;
      CASE next_char OF
        cr        : input_done := TRUE ;
        bs        : Backspace(input_string) ;
        '0'..'9' : input_string := input_string+next_char ;
        '.'       : input_string := input_string+next_char ;
        '-'       : input_string := input_string+next_char ;
      END ; { CASE next_char }
    UNTIL input_done ;
    IF input_string = '' THEN BEGIN
      GotoAntField( row,column ) ;
      WRITE(ant_list[ant_no].xcoord:8:2) ;
      EXIT ;
    END ; { IF }
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND (temp <= 99999.99)
                  AND (temp >= -9999.99) ;
    IF NOT good_input THEN
      Display_Edit_Error(input_string,'X coordinate.');
  UNTIL good_input ;
  data_changed := TRUE ;
  GotoAntField( row,column ) ; WRITE(temp:8:2) ;
  ant_list[ant_no].xcoord := temp ;
END ; { Get_Xcoord }
```

```
{*************************************************************
* This procedure processes keystrokes when in the
* y-coordionate field of the antenna data.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Get_Ycoord( key                    : CHAR;
                      ant_no, row, column : INTEGER ) ;

  VAR
    temp                    : REAL ;
    result, num             : INTEGER ;
    input_string            : string7 ;
    input_done, good_input  : BOOLEAN ;
    next_char               : CHAR ;
    field                   : STRING[14] ;

BEGIN { Get_Ycoord }
  GOTOXY( 1,19 ) ; Clrs2end ;
  Display_Edit_Instructions(' Y coordinate') ;
  input_string := key ;
  REPEAT
    REPEAT
      input_done := FALSE ;
      GotoAntField( row,column ) ; WRITE('         ') ;
      GotoAntField( row,column ) ; WRITE(input_string) ;
      READ(KBD,next_char) ;
      CASE next_char OF
        cr        : input_done := TRUE ;
        bs        : Backspace(input_string) ;
        '0'..'9' : input_string := input_string+next_char ;
        '.'       : input_stri   := input_string+next_char ;
        '-'       : input_string := input_string+next_char ;
      END ; { CASE next_char }
    UNTIL input_done ;
    IF input_string = '' THEN BEGIN
      GotoAntField( row,column ) ;
      WRITE(ant_list[ant_no].ycoord:8:2) ;
      EXIT ;
    END ; { IF }
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND (temp <= 99999.99)
                  AND (temp >= -9999.99) ;
    IF NOT good_input THEN
      Display_Edit_Error(input_string,'Y coordinate.');
  UNTIL good_input ;
  data_changed := TRUE ;
  GotoAntField( row,column ) ; WRITE(temp:8:2) ;
  ant_list[ant_no].ycoord := temp ;
END ; { Get_Ycoord }
```

```
{*************************************************************
* This procedure processes keystrokes when in the height
* field of the antenna data.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Get_Zcoord( key                    : CHAR;
                      ant_no, row, column : INTEGER ) ;

  VAR
    temp                    : REAL ;
    result, num             : INTEGER ;
    input_string            : string7 ;
    input_done, good_input  : BOOLEAN ;
    next_char               : CHAR ;
    field                   : STRING[14] ;

BEGIN { Get_Zcoord }
  GOTOXY( 1,19 ) ; Clrs2end ;
  Display_Edit_Instructions(' height') ;
  input_string := key ;
  REPEAT
    REPEAT
      input_done := FALSE ;
      GotoAntField( row,column ) ; WRITE('         ') ;
      GotoAntField( row,column ) ; WRITE(input_string) ;
      READ(KBD,next_char) ;
      CASE next_char OF
        cr        : input_done := TRUE ;
        bs        : Backspace(input_string) ;
        '0'..'9' : input_string := input_string+next_char ;
        '.'       : input_string := input_string+next_char ;
      END ; { CASE next_char }
    UNTIL input_done ;
    IF input_string = '' THEN BEGIN
      GotoAntField( row,column ) ;
      WRITE(ant_list[ant_no].zcoord:8:2) ;
      EXIT ;
    END ; { IF }
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND (temp <= 99999.99)
                  AND (temp >= 0) ;
    IF NOT good_input
      THEN Display_Edit_Error(input_string,'height.') ;
  UNTIL good_input ;
  data_changed := TRUE ;
  GotoAntField( row,column ) ; WRITE(temp:8:2) ;
  ant_list[ant_no].zcoord := temp ;
END ; { Get_Zcoord }
```

```
{*************************************************************
* This procedure processes keystrokes when in the gain
* field of the antenna data.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Get_Gain( key                    : CHAR;
                    ant_no, row, column : INTEGER ) ;

  VAR
    temp                     : REAL ;
    result, num              : INTEGER ;
    input_string             : string7 ;
    input_done, good_input   : BOOLEAN ;
    next_char                : CHAR ;
    field                    : STRING[14] ;

BEGIN { Get_Gain }
  GOTOXY( 1,19 ) ; Clrs2end ;
  Display_Edit_Instructions(' gain') ;
  input_string := key ;
  REPEAT
    REPEAT
      input_done := FALSE ;
      GotoAntField( row,column ) ; WRITE('         ') ;
      GotoAntField( row,column ) ; WRITE(input_string) ;
      READ(KBD,next_char) ;
      CASE next_char OF
        cr        : input_done := TRUE ;
        bs        : Backspace(input_string) ;
        '0'..'9' : input_string := input_string+next_char ;
        '.'       : input_string := input_string+next_char ;
      END ; { CASE next_char }
    UNTIL input_done ;
    IF input_string = '' THEN BEGIN
      GotoAntField( row,column ) ;
      WRITE(ant_list[ant_no].gain:8:2) ;
      EXIT ;
    END ; { IF }
    VAL( input_string, temp, result ) ;
    good_input := ( result = 0 ) AND (temp <= 999.99)
                  AND (temp >= 0) ;
    IF NOT good_input THEN
      Display_Edit_Error(input_string,'gain.') ;
  UNTIL good_input ;
  data_changed := TRUE ;
  GotoAntField( row,column ) ; WRITE(temp:8:2) ;
  ant_list[ant_no].gain := temp ;
END ; { Get_Gain }
```

```
{*************************************************************
* This procedure accepts the first keypressed that may be
* used for editing and sends it to the appropriate
* servicing routine based on which data field the cursor
* was in at the time.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Process_Ant_Input( key : CHAR;
             row, column, num : INTEGER ) ;

  VAR
    ant_num : INTEGER ;

BEGIN { Process_Ant_Input }
  ant_num := num - last_ant_row - 1 + row ;
  CASE column OF
    1 : Get_Xcoord( key, ant_num, row, column ) ;
    2 : Get_Ycoord( key, ant_num, row, column ) ;
    3 : Get_Zcoord( key, ant_num, row, column ) ;
    4 : Get_Gain( key, ant_num, row, column ) ;
  END ; { CASE column }
  Display_Antenna_Instructions ;
  GotoAntField( row,column ) ;
END ; { Process_Ant_Input }
```

```
{*************************************************************
* This procedure displays a warning message to the user,
* reminding him of the consequences of deleting an antenna,
* and then asks if he still wants to delete the antenna.
* If the user presses <Y> the antenna will be deleted.
* Pressing any other key will return the user to the edit
* mode without deleting the antenna.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Delete_This_Antenna( VAR num, row : INTEGER;
                                   column       : INTEGER ) ;

  VAR
    ant_no : INTEGER ;
    a_num  : STRING[3] ;
    toss   : CHAR ;
    answer : CHAR ;


BEGIN { Delete_This_Antenna }
  num := num - last_ant_row ;
  ant_no := num - 1 + row ;
  STR(ant_no,a_num) ;
  GOTOXY( 1,19 ) ; Clrs2end ;
  Reverse_Video ;
  Cwrite('Deleting antenna '+a_num+' will also delete',19) ;
  Cwrite('any frequencies using antenna '+a_num,20) ;
  Normal_Video ;
  Cwrite('Is That What You Want To Do (Y/N) ? ',22) ;
  WHILE keypressed DO READ(KBD,toss) ;
  READ(KBD,answer) ;
  IF UPCASE(answer) = 'Y'
    THEN BEGIN
      Delete_Antenna_Number( ant_no ) ;
      IF num + 12 > no_o_antennas THEN num := num - 1 ;
      IF num < 1 THEN num := 1 ;
  END ; { IF UPCASE(answer) }
  Display_Antennas_Starting_At_Number(num) ;
  IF row > last_ant_row THEN row := last_ant_row ;
  GotoAntField( row, column ) ;
END ; { Delete_This_Antenna }
```

```
{************************************************************
* This procedure processes keystrokes to allow the user to
* change fields within the displayed antenna data, edit
* antenna data, display more antenna information for
* editing, or return to the EDIT MENU.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Change_Antenna_Data ;

  VAR
    done_editing      : BOOLEAN ;
    num, row, column  : INTEGER ;
    key               : CHAR ;

{////////////////////////////////////////////////////////////
/ This procedure determines if a cursor key or the home key
/ has been pressed and takes the appropriate action.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Process_Esc ;

  VAR
    char2 : CHAR ;
    toss  : CHAR ;

BEGIN { Process_Esc }
  READ(KBD,char2) ;
  CASE char2 OF
    'A' : BEGIN
            row := row - 1 ;
            IF row < 1 THEN row := 1 ;
          END ;
    'B' : BEGIN
            row := row + 1 ;
            IF row > 12 THEN row := 12 ;
            IF row > last_ant_row THEN row := last_ant_row ;
          END ;
    'C' : BEGIN
            column := column + 1 ;
            IF column > 4 THEN column := 4 ;
          END ;
    'D' : BEGIN
            column := column - 1 ;
            IF column < 1 THEN column := 1 ;
          END ;
    'H' : done_editing := TRUE ;
  END ; { CASE char2 }
  GotoAntField( row, column ) ;
END ; { Process_Esc }
```

```
{////////////////////////////////////////////////////////////}

BEGIN { Change_Antenna_Data }
  Display_Antenna_Headings ;
  num := 1 ;
  Display_Antennas_Starting_At_Number(num) ;
  row := 1 ; column := 1 ;
  GotoAntField(row, column) ;
  done_editing := FALSE ;
  REPEAT
    READ(KBD,key) ;
    CASE UPCASE(key) OF
      esc       : Process_Esc ;
      'U'       : Scroll_Screen_Up( num, row, column ) ;
      'D'       : Scroll_Screen_Down( num, row, column ) ;
      '0'..'9'  : Process_Ant_Input( key, row, column, num ) ;
      '-'       : Process_Ant_Input( key, row, column, num ) ;
      DELETE    : Delete_This_Antenna( num, row, column ) ;
    END ; { CASE key }
  UNTIL done_editing ;
  Cursor( block, noblink, noclick, off ) ;
END ; { Change_Antenna_Data }



{************************************************************
* This procedure returns the user to TIMAP's MAIN MENU.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Return_To_Main_Menu ;

BEGIN { Return_To_Main_Menu }
  Cursor( block, noblink, noclick, off ) ;
  done := TRUE ;
END ; { Return_To_Main_Menu }

{////////////////////////////////////////////////////////////}

BEGIN { Edit_Data }
  REPEAT ;
    Display_Edit_Menu ;
    done := FALSE ;
    CASE ( Get_Menu_Selection( 4, 3, 25 ) ) OF
      1 : Change_Antenna_Data ;
      2 : Change_Frequency_Data ;
      3 : Return_To_Main_Menu ;
    END ; { CASE STATEMENT }
  UNTIL done ;
END ; { Edit_Data }
```

```
{************************************************************
* This procedure permits a graceful return to MS-DOS from
* TIMAP.  If changes to the data have been made and the data
* has not been saved then this procedure warns the user and
* permits him to return to the MAIN MENU where he may save
* the data.  Otherwise it clears the screen, including the
* 25th line TIMAP banner, turns on the cursor, and returns
* the user to the MS-DOS operating system prompt.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Exit_To_System ;

  VAR
    input_char : CHAR ;

BEGIN { Exit_To_System }
  CLRSCR ;
  Cursor( Block, Noblink, Noclick, On ) ;
  IF data_changed
    THEN BEGIN
      Reverse_Video ;
      Cwrite('Press <Y> to exit to the system or '+
             'any other key', 22);
      Cwrite('to return to the MAIN MENU.', 23) ;
      Cwrite('You have made changes and not saved the '+
             'data.', 10) ;
      Normal_Video ;
      Cwrite('Are You Sure You Want To Quit ?  ', 12) ;
      READ(KBD, input_char) ;
      IF UPCASE( input_char ) <> 'Y' THEN EXIT ;
  END ; { IF data_changed }
  Graphics_Off ;
  finished := TRUE ;
END ; { Exit_To_System }
```

```
{*************************************************************
* This file is a collection of graphics related routines
* and constants.  (Version 1.00  5 June 1986)
*************************************************************}

CONST
  Block       = TRUE;      {                                  }
  Blink       = TRUE;      {                                  }
  Click       = TRUE;      {                                  }
  On          = TRUE;      { These constants are useful       }
  Enable      = TRUE;      { for use with the cursor          }
  Underscore  = FALSE;     { procedure below.                 }
  Noblink     = FALSE;     {                                  }
  Noclick     = FALSE;     {                                  }
  Off         = FALSE;     {                                  }
  Disable     = FALSE;     {                                  }
  Black       = 0;       {                                    }
  Blue        = 1;       {                                    }
  Red         = 2;       { These constants are useful for     }
  Magenta     = 3;       { use with routines, such as the     }
  Green       = 4;       { line drawing routine, where a      }
  Cyan        = 5;       { color must be specified.           }
  Yellow      = 6;       {                                    }
  White       = 7;       {                                    }

VAR
  old_vcr : INTEGER;
  aspect : ARRAY[0..225] OF INTEGER;
  current_ratio : REAL;
```

```
{*************************************************************
*
* PROCEDURE NAME   :  Enable the 25th line.
*
* SYNOPSIS         :  Line_25(enable) ;
*
* DESCRIPTION      :  This procedure permits writing to the
*                     screen's 25th line when "enable" is
*                     true and prohibits writing to the
*                     25th line when "enable" is false.
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE Line_25( enable : BOOLEAN );

  CONST
    Escape = #27 ;

BEGIN { Line_25 }

  CASE enable OF
    TRUE  : WRITE(Escape,'x1'); { Enable 25th Line  }
    FALSE : WRITE(Escape,'y1'); { Disable 25th Line }
  END; { CASE enabled }

END ; { Line_25 }



{*************************************************************
*
* PROCEDURE NAME   :  Clear The Line
*
* SYNOPSIS         :  Clrline;
*
* DESCRIPTION      :  This procedure clears the entire line
*                     containing the cursor.
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE Clrline;

  CONST
    Escape = #27 ;

BEGIN { Clrline }
  WRITE( Escape,'l' ) ;
END ; { Clrline }
```

```
{*************************************************************
*
* PROCEDURE NAME   :  Turn on the graphics capabilities.
*
* SYNOPSIS         :  Graphics_on ;
*
* DESCRIPTION      :  This procedure prepares the system for
*                     use of the other graphics procedures
*                     in this file.  It saves the status
*                     of the video control register so that
*                     the status may be returned later,
*                     enables CPU access to video memory,
*                     initializes an aspect ratio array for
*                     use with the circle routines, and
*                     installs a fast screen driver routine.
*                     Details involving video control
*                     register programming and related topics
*                     may be found in the Z-100 Technical
*                     Manual, Hardware, beginning on page
*                     4.30.
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE Graphics_on ;

  CONST
    video_control_register = $D8 ;
VAR
  i : INTEGER ;


{////////////////////////////////////////////////////////////
/
/ PROCEDURE NAME   :  Fast Screen Driver
/
/ SYNOPSIS         :  Display_Driver( character );
/
/ DESCRIPTION      :  This procedure replaces the MS-DOS
/                     screen driver to permit faster
/                     updating of information on the screen.
/
/ VERSION & DATE   :  1.00 - 5 June 1986
/
////////////////////////////////////////////////////////////}

PROCEDURE Display_Driver( Output_Char : CHAR ) ;

BEGIN { Display_Driver }
  INLINE( $8A/$46/$04/$FC/$9A/$19/$00/$01/$FE ) ;
END ; { Display_Driver }
```

```
{//////////////////////////////////////////////////////////////
/
/ PROCEDURE NAME   :  Initialize the screen driver.
/
/ SYNOPSIS         :  Initialize_Display_Driver ;
/
/ DESCRIPTION      :  This procedure installs the replaces
/                     fast screen driver described above.
/
/ VERSION & DATE   :  1.00 - 5 June 1986
/
//////////////////////////////////////////////////////////////}

PROCEDURE Initialize_Display_Driver;

BEGIN { Initialize_Display_Driver }
  MEMW[SEG(ConOutPtr):OFS(ConOutPtr)]:=OFS(Display_Driver);
END ; { Initialize_Display_Driver }



{//////////////////////////////////////////////////////////////}

BEGIN { Graphics_On }
   Initialize_Display_Driver ;
   Line_25( Enable ) ;
   GOTOXY( 1,25 ) ; Clrline ;
   GOTOXY( 1, 1 ) ; CLRSCR ;
   old_vcr := PORT[video_control_register] ;
   PORT[video_control_register] := old_vcr AND $7F ;
   FOR i := 0 TO 225 DO
      aspect[i] := ROUND(i * 0.4843) ;
   current_ratio := 1.0 ;
END ; { Graphics_On }
```

```
{*************************************************************
*
* PROCEDURE NAME   :  Turn the graphics capabilities off
*
* SYNOPSIS         :  Graphics_off ;
*
* DESCRIPTION      :  This procedure restores the video
*                     control register to the status
*                     it was in before the Graphics-on
*                     procedure was executed and also
*                     assures that access to the 25th line
*                     is disabled.
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE Graphics_Off ;

  CONST
    video_control_register = $D8 ;

BEGIN { Graphics_Off }
   PORT[video_control_register] := old_vcr;
   GOTOXY( 1, 25 ) ; Clrline ;
   GOTOXY( 1, 1 )  ; CLRSCR ;
   Line_25( Disable ) ;
END ; { Graphics_Off }



{*************************************************************
*
* PROCEDURE NAME   :  Enter Reverse Video Mode
*
* SYNOPSIS         :  Reverse_Video ;
*
* DESCRIPTION      :  Sets the display to the reverse
*                     ( Black on white ) video mode.
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE Reverse_Video ;

  CONST
    Escape = #27 ;

BEGIN { Reverse_Video }
  WRITE( Escape,'p' ) ;
END ; { Reverse_Video }
```

```
{*************************************************************
*
* PROCEDURE NAME   :  Enter Normal Video Mode
*
* SYNOPSIS         :  Normal_Video ;
*
* DESCRIPTION      :  Sets the display to the normal
*                     ( White on black ) video mode.
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE Normal_Video ;

  CONST
    Escape = #27 ;

BEGIN { Normal_Video }
  WRITE( Escape,'q' ) ;
END ; { Normal_Video }
```

```
{*************************************************************
*
* PROCEDURE NAME   :  Cursor
*
* SYNOPSIS         :  Cursor( block, blink, click, on);
*
* DESCRIPTION      :  This procedure allows you to modify the
*                     cursor as follows:
*
*                     block = true  - block cursor
*                     block = false - underscore cursor
*
*                     blink = true  - blinking cursor
*                     blink = false - nonblinking cursor
*
*                     click = true  - clicking cursor
*                     click = false - nonclicking cursor
*
*                     on    = true  - cursor displayed
*                     on    = false - no cursor displayed
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE Cursor( block, blink, click, on : BOOLEAN );

  CONST
    Escape = #27 ;

BEGIN { *** Cursor *** }
  CASE block OF
    TRUE  : WRITE(Escape,'x4'); { Block Cursor       }
    FALSE : WRITE(Escape,'y4'); { Underscore Cursor }
  END ; { CASE block }

  CASE blink OF
    TRUE  : WRITE(Escape,'y;'); { Blinking Cursor    }
    FALSE : WRITE(Escape,'x;'); { Nonblinking Cursor }
  END ; { CASE blink }

  CASE click OF
    TRUE  : WRITE(Escape,'y2'); { Key Click On  }
    FALSE : WRITE(Escape,'x2'); { Key Click Off }
  END ; { CASE click }

  CASE on OF
    TRUE  : WRITE(Escape,'y5'); { Cursor On  }
    FALSE : WRITE(Escape,'x5'); { Cursor Off }
  END ; { CASE on }
END ; { *** Cursor *** }
```

```
{*************************************************************
*
* PROCEDURE NAME   :  Clear Screen To End
*
* SYNOPSIS         :  Clrs2end;
*
* DESCRIPTION      :  This procedure clears the screen from
*                     the line containing the cursor to the
*                     end of the screen but does not clear
*                     the 25th line.
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE Clrs2end;

  CONST
    Escape = #27 ;

BEGIN { *** Clrs2end *** }
  WRITE(Escape,'J');
END;  { *** Clrs2end *** }



{*************************************************************
*
* PROCEDURE NAME   :  Clear Screen To Beginning
*
* SYNOPSIS         :  Clrs2beg;
*
* DESCRIPTION      :  This procedure clears the screen from
*                     the line containing the cursor to the
*                     beginning of the screen.
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE Clrs2beg;

  CONST
    Escape = #27 ;

BEGIN { *** Clrs2beg *** }
  WRITE(Escape,'b');
END;  { *** Clrs2beg *** }
```

```
{*************************************************************
*
* PROCEDURE NAME   :  Clear To The Beginning Of The Line
*
* SYNOPSIS         :  Clrl2beg ;
*
* DESCRIPTION      :  This procedure clears the line
*                     containing the cursor from the cursor
*                     position to the beginning of the line.
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE Clrl2beg ;

  CONST
    Escape = #27 ;

BEGIN { Clrl2beg }
  WRITE( Escape,'o' ) ;
END ; { Clrl2beg }



{*************************************************************
*
* FUNCTION NAME    :  Convert pixel address to byte address
*
* SYNOPSIS         :  Byteaddr(x,y);
*
* DESCRIPTION      :  This function converts a pixel address
*                     given as "x,y" integers into the
*                     corresponding memory location in the
*                     H/Z-100's video memory.
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

FUNCTION Byteaddr (x,y:INTEGER) : INTEGER;
  VAR
    xbyte,ychar,yscan : INTEGER;
BEGIN { Byteaddr }
  xbyte := x DIV 8;
  ychar := y DIV 9;
  yscan := y MOD 9;
  byteaddr := ychar*2048 + yscan*128 + xbyte;
END ; { Byteaddr }
```

```
{*************************************************************
*
* PROCEDURE NAME   :  Turn on a pixel
*
* SYNOPSIS         :  Pset(x,y,color);
*
* DESCRIPTION      :  This procedure turns on a "color"
*                     pixel at pixel location "x,y".
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE pset (x,y,color : INTEGER) ;

  CONST
    sgment : ARRAY [0..2] OF INTEGER = ( $C000,$D000,$E000 ) ;
    { These are the segment addresses for the blue, red,
      and green video RAM respectively }
  VAR
    bit_off,location,xbit,plane : INTEGER ;
    vidchr : ^byte;

BEGIN
  location := byteaddr (x,y);
  xbit := $80 SHR (x MOD 8);
  bit_off:=NOT xbit;
  FOR plane := 0 TO 2 DO BEGIN
     vidchr := ptr(sgment[plane],location);
     IF (color AND (1 SHL plane) > 0) THEN
        vidchr^ := vidchr^ OR xbit
     ELSE
        vidchr^ := vidchr^ AND bit_off;
  END;
END;
```

```
{*************************************************************
*
* PROCEDURE NAME   :  Turn off a pixel
*
* SYNOPSIS         :  Preset(x,y);
*
* DESCRIPTION      :  This procedure turns off a pixel at
*                     pixel location "x,y".
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE Preset (x,y : INTEGER);

  CONST
    sgment : ARRAY [0..2] OF INTEGER = ( $C000,$D000,$E000 ) ;
    { These are the segment addresses for the blue, red,
      and green video RAM respectively }

  VAR
    bit_off,location,xbit,plane : INTEGER;

BEGIN { Preset }
  location := byteaddr (x,y);
  xbit := $80 SHR (x MOD 8);
  bit_off:=NOT xbit;
  FOR plane := 0 TO 2 DO
     mem[sgment[plane]:location] :=
     mem[sgment[plane]:location] AND bit_off;
END ; { Preset }

{*************************************************************
*
* PROCEDURE NAME   :  Draw a line
*
* SYNOPSIS         :  Drawline(x1,y1, x2,y2 color) ;
*
* DESCRIPTION      :  This procedure draws a "color" line
*                     from pixel location "x1,y1" to pixel
*                     location "x2,y2".
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE Drawline( x1,y1, x2,y2, color : INTEGER ) ;

VAR
   dev, dx, dy, x, y : INTEGER;
```

```
{////////////////Sub-Procedure Of Drawline/////////////////}

PROCEDURE case1;

BEGIN { case1 }
   FOR x := (x1 + 1) TO x2 DO BEGIN
      dev := dev + dy + dy;
      IF dev > dx THEN BEGIN
         y := y + 1;
         dev := dev - dx - dx
      END ; { IF dev }
      pset(x,y,color);
   END ; { FOR x }
END ; { case1 }

{////////////////Sub-Procedure Of Drawline/////////////////}

PROCEDURE case2;

BEGIN { case2 }
   FOR y := (y1 + 1) TO y2 DO BEGIN
      dev := dev + dx + dx;
      IF dev > dy THEN BEGIN
         x := x + 1;
         dev := dev - dy - dy;
      END ; { IF dev }
      pset(x,y,color);
   END ; { FOR y }
END ; { case2 }

{////////////////Sub-Procedure Of Drawline/////////////////}

PROCEDURE case3;

BEGIN { case3 }
   FOR x := (x1 + 1) TO x2 DO BEGIN
      dev := dev + dy + dy;
      IF dev > dx THEN BEGIN
         y := y - 1;
         dev := dev - dx - dx;
      END ; { IF dev }
      pset(x,y,color);
   END ; { FOR x }
END ; { case3 }
```

```
{////////////////Sub-Procedure Of Drawline/////////////////}

PROCEDURE case4;

BEGIN { case4 }
   FOR y := (y1 - 1) DOWNTO y2 DO BEGIN
      dev := dev + dx + dx;
      IF dev > dy THEN BEGIN
         x := x + 1;
         dev := dev - dy - dy;
      END ; { IF dev }
      pset(x,y,color);
   END ; { FOR y }
END ; { case4 }

{////////////////Sub-Procedure Of Drawline/////////////////}

PROCEDURE case5;

BEGIN { case5 }
   FOR x := (x1 - 1) DOWNTO x2 DO BEGIN
      dev := dev + dy + dy;
      IF dev > dx THEN BEGIN
         y := y + 1;
         dev := dev - dx - dx;
      END ; { IF dev }
      pset(x,y,color);
   END ; { FOR x }
END ; { case5 }

{////////////////Sub-Procedure Of Drawline/////////////////}

PROCEDURE case6;

BEGIN { case6 }
   FOR y := (y1 + 1) TO y2 DO BEGIN
      dev := dev + dx + dx;
      IF dev > dy THEN BEGIN
         x := x - 1;
         dev := dev - dy - dy;
      END ; { IF dev }
      pset(x,y,color);
   END ; { FOR y }
END ; { case6 }
```

```
{/////////////////Sub-Procedure Of Drawline/////////////////}

PROCEDURE case7;

BEGIN { case7 }
   FOR x := (x1 - 1) DOWNTO x2 DO BEGIN
      dev := dev + dy + dy;
      IF dev > dx THEN BEGIN
         y := y - 1;
         dev := dev - dx - dx;
      END ; { IF dev }
      pset(x,y,color);
   END ; { FOR x }
END ; { case7 }

{/////////////////Sub-Procedure Of Drawline/////////////////}

PROCEDURE case8;

BEGIN { case8 }
   FOR y := (y1 - 1) DOWNTO y2 DO BEGIN
      dev := dev + dx + dx;
      IF dev > dy THEN BEGIN
         x := x - 1;
         dev := dev - dy - dy;
      END ; { IF dev }
      pset(x,y,color);
   END ; { FOR y }
END ; { case8 }
```

```
{////////////////////////////////////////////////////////////////////}

BEGIN {drawline}
   IF x1 = x2 THEN
      IF y1 < y2 THEN
         FOR y := y1 TO y2 DO
            pset(x1,y,color)
      ELSE
         FOR y := y1 DOWNTO y2 DO
            pset(x1,y,color)
   ELSE IF y1 = y2 THEN
      IF x1 < x2 THEN
         FOR x := x1 TO x2 DO
            pset(x,y1,color)
      ELSE
         FOR x := x1 DOWNTO x2 DO
            pset(x,y1,color)
   ELSE BEGIN
      pset(x1,y1,color);
      dev := 0;
      x := x1; y := y1;
      dx := ABS(x2 - x1);
      dy := ABS(y2 - y1);
      IF x2 >= x1 THEN
         IF y2 >= y1 THEN
            IF dx >= dy THEN case1 ELSE case2
         ELSE
            IF dx >= dy THEN case3 ELSE case4
      ELSE
         IF y2 >= y1 THEN
            IF dx >= dy THEN case5 ELSE case6
         ELSE
            IF dx >= dy THEN case7 ELSE case8;
   END;
END ; { Drawline }
```

```
{*************************************************************
*
* PROCEDURE NAME   :  Draw a box
*
* SYNOPSIS         :  Drawbox(x1,y1, x2,y2 color) ;
*
* DESCRIPTION      :  This procedure draws a "color"
*                     rectangle whose upper left hand coner
*                     is at pixel location "x1,y1" and whose
*                     lower right hand corner is at pixel
*                     location "x2,y2".
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
*************************************************************}

PROCEDURE Drawbox( x1,y1, x2,y2, color : INTEGER );

BEGIN { Drawbox }
   drawline(x1,y1,x2,y1,color);
   drawline(x2,y1,x2,y2,color);
   drawline(x2,y2,x1,y2,color);
   drawline(x1,y2,x1,y1,color);
END ; { Drawbox }
```

```
{************************************************************
*
* PROCEDURE NAME   :  Draw an ellipse
*
* SYNOPSIS         :  Drawcircle(x,y,radius,color) ;
*
* DESCRIPTION      :  This procedure draws a "color" ellipse
*                     centered on pixel location "x,y" with a
*                     radius of "radius" in x pixels and an
*                     aspect ratio x to y of "aspect_ratio".
*                     If "aspect_ratio" is 1.0 the ellipse
*                     will be a circle; if less than 1.0 the
*                     ellipse will be longer along the
*                     horizontal axis; if greater than 1.0
*                     the ellipse will be longer along the
*                     vertical axis.
*
* VERSION & DATE   :  1.00 - 5 June 1986
*
************************************************************}

PROCEDURE Drawcircle( ix,iy, radius, color : INTEGER;
                      aspect_ratio : REAL);

  VAR
   x,y,dev : INTEGER;
   ta : ARRAY[0..225] OF INTEGER;
   i : INTEGER;

{/////////////////Sub-Procedure Of Drawcircle/////////////////}

PROCEDURE reflect;

BEGIN { reflect }
   pset(ix+x,iy+aspect[y],color);
   pset(ix-x,iy+aspect[y],color);
   pset(ix+x,iy-aspect[y],color);
   pset(ix-x,iy-aspect[y],color);
   IF x <> y THEN BEGIN
      pset(ix+y,iy+aspect[x],color);
      pset(ix-y,iy+aspect[x],color);
      pset(ix+y,iy-aspect[x],color);
      pset(ix-y,iy-aspect[x],color);
   END ; { IF x <> Y }
END ; { reflect }
```

```
{////////////////////////////////////////////////////////////}

BEGIN {drawcircle}
   IF aspect_ratio <> current_ratio THEN BEGIN
      FOR i := 0 TO 225 DO
         aspect[i] := ROUND(i * (0.4843 * aspect_ratio));
      current_ratio := aspect_ratio;
   END ; { IF aspect_ratio }
   x := radius;
   y := 0;
   dev := 0;
   pset(ix+radius,iy,color);
   pset(ix,iy+aspect[radius],color);
   pset(ix-radius,iy,color);
   pset(ix,iy-aspect[radius],color);
   WHILE y < x DO BEGIN
      dev := dev + y + y + 1;
      y := y + 1;
      IF dev > x THEN BEGIN
         dev := dev - x - x + 1;
         x := x - 1;
      END ; { IF dev }
      reflect;
   END ; { WHILE y }
END ; { Drawcircle }
```

```
{************************************************************
* This procedure displays a list of all data files in the
* current directory, prompts the user for the name of the
* file he wants to retrieve and retrieves the antenna and
* frequency data from the disk..
* (Version 1.00  5 June 1986)
************************************************************}

OVERLAY PROCEDURE Load_Data ;

  VAR
    freq_file   : FILE OF frequency_info ;
    ant_file    : FILE OF antenna_info ;
    source_file : STRING[8] ;
    file_found  : BOOLEAN ;
    size        : INTEGER ;

{************************************************************
* This procedure displays a reminder to the user about
* entering a filename for the data file to be retrieved or
* how to exit this function without retrieving a data file.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Display_Instructions;

  VAR
    index : INTEGER ;

BEGIN { Display_Instructions }

Reverse_Video ;
  Cwrite
    ('YOUR FILENAME MAY CONTAIN UP TO 8 CHARACTERS.', 19) ;
  Cwrite('DO NOT INCLUDE A FILE EXTENSION.', 20) ;
  Cwrite('Press <RETURN> without entering a filename', 22) ;
  Cwrite
    ('to return to the MAIN MENU without loading data.', 23 ) ;
  Normal_Video ;
  GOTOXY(1,5) ;
  WRITE('These are the data files on ') ;
  IF path = ''
    THEN WRITE('the default drive')
    ELSE WRITE(path) ;
END ; { Display_Instructions }
```

```
{************************************************************
* This procedure displays a message telling the user that
* the file he wanted to retrieve is not in the default
* directory.  (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Alert_User;

  CONST
    bell = #7 ;

BEGIN { Alert_User }
  WRITELN( bell ) ;
  GOTOXY( 1 , 16 ) ; Clrline ;
  Cwrite
    (CONCAT('Sorry, I couldn''t find a file named "',
     path+source_file,'".'), 16) ;
  Cwrite('Please try again.', 17) ;
END ; { Alert_User }
```

```
{************************************************************
* This procedure prompts the user for the name of the file
* containing the data he wants to retrieve, checks for valid
* input, and prepares the file for reading the data.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Find_File ;

BEGIN { Find_File }
  REPEAT
    GOTOXY( 1, 2 ) ; Clrline ;
    WRITE('What is the name of the data file you want ') ;
    WRITE('to load ?  ') ;
    Cursor( block, noblink, noclick, on ) ;
    READLN( source_file ) ;
    IF LENGTH( source_file ) = 0
      THEN BEGIN
        file_found := FALSE ;
        EXIT;
    END ; { IF }
    Cursor( block, noblink, noclick, off ) ;
    size := pos('.',source_file) ;
    IF size <> 0
      THEN source_file := COPY(source_file,1,size-1) ;
    ASSIGN( freq_file,path+source_file+'.FQD' ) ;
    {$I-}  RESET(freq_file) {$I+} ;
    file_found := (IOresult = 0) ;
    IF file_found
      THEN BEGIN
        ASSIGN( ant_file,path+source_file+'.ATD' ) ;
        {$I-}  RESET(ant_file) {$I+} ;
        file_found := (IOresult = 0) ;
    END ; { IF file_found }
    IF NOT file_found THEN Alert_User ;
  UNTIL file_found ;
END ; { Find_File }
```

```
{*************************************************************
* This procedure reads the frequency and antenna data from
* two separate disk files.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Get_Data ;

  VAR
    count : INTEGER;

BEGIN { Get_Data }
  count := 1 ;
  WHILE NOT (EOF(freq_file)) DO BEGIN
    READ(freq_file,freq_list[count]);
    count := count + 1 ;
  END ; { WHILE NOT EOF }
  CLOSE(freq_file);
  no_o_frequencies := count - 1 ;
  count := 1 ;
  WHILE NOT (EOF(ant_file)) DO BEGIN
    READ(ant_file,ant_list[count]);
    count := count + 1 ;
  END ; { WHILE NOT EOF }
  CLOSE(ant_file);
  Data_Changed := FALSE ;
  no_o_antennas := count - 1 ;
END ; { Get_Data }



{//////////////////////MAIN-PROCEDURE////////////////////////}

BEGIN { Load_Data }
  CLRSCR;
  Display_Instructions ;
  Display_Directory(7,'????????.??D') ;
  Find_File ;
  IF file_found THEN Get_Data ;
END ; { Load_Data }
```

```
{*****************************************************************
* This file contains a collection of procedures for
* dislpaying the TIMAP menus and for processing keystrokes
* while in a TIMAP menu to allow menu item (option)
* selection.
*****************************************************************



*****************************************************************
* This procedure draws a small box to provide a border
* around menu item numbers.  (Version 1.00  5 June 1986)
*****************************************************************}

PROCEDURE Menu_Item_Box( x1, y1, color : INTEGER ) ;
  BEGIN { Menu_Item_Box }
    Drawbox( x1, y1, x1+9, y1+10, color ) ;
  END ; { Menu_Item_Box }
```

```
{*************************************************************
* This procedure displays TIMAP's MAIN MENU on the screen.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Main_Menu ;

  VAR
    x1, y1, index : INTEGER ;

BEGIN { Display_Main_Menu }

  Cursor( Block, Noblink, Noclick, Off ) ;
  CLRSCR ;

  GOTOXY( 1,25 )  ; Clrline ;
  Cwrite(CONCAT('TRANSMITTER INTERMODULATION ANALYSIS',
    ' PROGRAM - by ZUZACK - Version 1.00'), 25 ) ;

  Cwrite('MAIN MENU', 2 ) ;
  Cwrite('1  Add data.                        ', 4 ) ;
  Cwrite('2  Print the current data.          ', 6 ) ;
  Cwrite('3  Display the current data.        ', 8 ) ;
  Cwrite('4  Change or delete data.           ', 10 ) ;
  Cwrite('5  Save the data to disk.           ', 12 ) ;
  Cwrite('6  Load data from disk.             ', 14 ) ;
  Cwrite('7  Perform an intermod analysis.    ', 16 ) ;
  Cwrite('8  Set options.                     ', 18 ) ;
  Cwrite('9  Exit to the system.              ', 20 ) ;

  Drawbox( 144, 18, 494, 190, white ) ;
  Drawbox( 220, 8, 420, 18, white ) ;

  x1 := 191 ;
  FOR index := 1 TO 9 DO
  BEGIN { FOR index }
      y1 := (index+1)*18-10 ;
      Menu_Item_Box( x1, y1, white ) ;
  END ; { FOR index }

END ; { Display_Main_Menu }
```

```
{*************************************************************
* This procedure displays TIMAP's ADD MENU on the screen.
* (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Add_Menu ;

  VAR
    x1, y1, index : INTEGER ;

BEGIN { Display_Add_Menu }

  Cursor( Block, Noblink, Noclick, Off ) ;
  CLRSCR ;
  Cwrite('ADD MENU', 2 ) ;
  Cwrite('1  Add antenna data.    ', 4 ) ;
  Cwrite('2  Add frequency data. ', 6 ) ;
  Cwrite('3  Clear current data. ', 8 ) ;
  Cwrite('4  Return to MAIN MENU.', 10 ) ;

  Drawbox( 144, 18, 494, 100, white ) ;
  Drawbox( 220, 8, 420, 18, white ) ;

  x1 := 223 ;
  FOR index := 1 TO 4 DO
  BEGIN { FOR index }
      y1 := (index+1)*18-10 ;
      Menu_Item_Box( x1, y1, white ) ;
  END ; { FOR index }

END ; { Display_Add_Menu }
```

```
{*************************************************************
* This procedure displays TIMAP's DISPLAY MENU on the
* screen.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Display_Menu ;

  VAR
    x1, y1, index : INTEGER ;

BEGIN { Display_Display_Menu }

  Cursor( Block, Noblink, Noclick, Off ) ;
  CLRSCR ;
  Cwrite('DISPLAY MENU', 2 ) ;
  Cwrite('1  Display antenna data.  ', 4 ) ;
  Cwrite('2  Display frequency data.', 6 ) ;
  Cwrite('3  Return to MAIN MENU.   ', 8 ) ;

  Drawbox( 144, 18, 494, 82, white ) ;
  Drawbox( 220, 8, 420, 18, white ) ;

  x1 := 215 ;
  FOR index := 1 TO 3 DO
  BEGIN { FOR index }
      y1 := (index+1)*18-10 ;
      Menu_Item_Box( x1, y1, white ) ;
  END ; { FOR index }

END ; { Display_Display_Menu }
```

```
{************************************************************
* This procedure displays TIMAP's PRINT MENU on the
* screen.  (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Display_Print_Menu ;

  VAR
    x1, y1, index : INTEGER ;

BEGIN { Display_Print_Menu }

  Cursor( Block, Noblink, Noclick, Off ) ;
  CLRSCR ;
  Cwrite('PRINT MENU', 2 ) ;
  Cwrite('1  Print antenna data.    ', 4 ) ;
  Cwrite('2  Print frequency data.  ', 6 ) ;
  Cwrite('3  Return to MAIN MENU.   ', 8 ) ;

  Drawbox( 144, 18, 494, 82, white ) ;
  Drawbox( 220, 8, 420, 18, white ) ;

  x1 := 215 ;
  FOR index := 1 TO 3 DO
  BEGIN { FOR index }
      y1 := (index+1)*18-10 ;
      Menu_Item_Box( x1, y1, white ) ;
  END ; { FOR index }

END ; { Display_Print_Menu }
```

```
{*************************************************************
* This procedure displays TIMAP's OPTIONS MENU on the
* screen.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Options_Menu ;

  CONST
    Column = 22 ;

  VAR
    x1, y1, index : INTEGER ;

BEGIN { Display_Options_Menu }

  Cursor( Block, Noblink, Noclick, Off ) ;
  CLRSCR ;
  Cwrite('OPTIONS MENU', 2 ) ;
  GOTOXY( column,4 ) ;
  WRITE('1  Specify amplifier type : ') ;
  IF specify_amp_type
    THEN WRITE('True')
    ELSE WRITE('False') ;
  GOTOXY( column,6 ) ;
  WRITE('2  Specify IM coefficient : ') ;
  IF specify_im_coef
    THEN WRITE('True')
    ELSE WRITE('False') ;
  GOTOXY( column,8 ) ;
  WRITE('3  Set path name : ') ;
  index := 1 ;
  WHILE (path_name[index] <> ' ') AND (index <= 20) DO BEGIN
    WRITE(path_name[index]) ;
    index := index + 1 ;
  END ; { WHILE }
  IF index = 1 THEN WRITE('The Default') ;
  GOTOXY( column,10 ) ;
  WRITE('4  Return to MAIN MENU') ;

  Drawbox( 144, 18, 494, 100, white ) ;
  Drawbox( 220, 8, 420, 18, white ) ;

  x1 := 167 ;
  FOR index := 1 TO 4 DO
  BEGIN { FOR index }
      y1 := (index+1)*18-10 ;
      Menu_Item_Box( x1, y1, white ) ;
  END ; { FOR index }

END ; { Display_Options_Menu }
```

```
{************************************************************
* This procedure displays TIMAP's EDIT MENU on the
* screen.  (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Display_Edit_Menu ;

  VAR
    x1, y1, index : INTEGER ;

BEGIN { Display_Edit_Menu }

  Cursor( Block, Noblink, Noclick, Off ) ;
  CLRSCR ;
  Cwrite('EDIT MENU', 2 ) ;
  Cwrite('1  Edit antenna information.    ', 4 ) ;
  Cwrite('2  Edit frequency information.  ', 6 ) ;
  Cwrite('3  Return To Main Menu.         ', 8 ) ;

  Drawbox( 144, 18, 494, 82, white ) ;
  Drawbox( 220, 8, 420, 18, white ) ;

  x1 := 191 ;
  FOR index := 1 TO 3 DO
  BEGIN { FOR index }
      y1 := (index+1)*18-10 ;
      Menu_Item_Box( x1, y1, white ) ;
  END ; { FOR index }

END ; { Display_Edit_Menu }
```

```
{************************************************************
* This function processes keystrokes to allow the user to
* make selections from a menu.  Selections may be made by
* pressing the menu item number or by moving the cursor to
* the menu item number and then pressing <RETURN>.  Any
* other keystrokes are ignored. (Version 1.00  5 June 1986)
************************************************************}

FUNCTION Get_Menu_Selection( min, max, column : INTEGER )
         {RETURN} : INTEGER;

  VAR
    y, y1 : INTEGER ;
    row, last : INTEGER ;
    chr1, chr2 : CHAR ;
    finished : BOOLEAN ;
```

```
{////////////////////SUB-PROCEDURE////////////////////////
/ If the <RETURN> or <ENTER> key is pressed then the user
/ wants to select the menu item highlighted by the cursor.
/ This procedure prepares the main function,
/ "Get_Menu_Selection", to return a value equal to the
/ menu item number presently highlighted by the cursor.
/ (Version 1.00  5 June 1986)
//////////////////////////////////////////////////////////}

PROCEDURE Process_Cr ;

BEGIN { Process_Cr }
  Cursor( Block, Noblink, Noclick, Off ) ;
  Get_Menu_Selection := y ;
  finished := TRUE ;
END ; { Process_Cr }



{////////////////////SUB-PROCEDURE////////////////////////
/ If a cursor key, the <ESC> key, or other special function
/ key is pressed an escape sequence is issued to the
/ computer.  If an up or down cursor key is pressed we want
/ to move the cursor appropriately.  This procedure
/ positions the cursor and prepares the main function,
/ "Get_Menu_Selection", to return a value equal to the
/ menu item number presently highlighted by the cursor in
/ case the next key pressed is the <RETURN> or <ENTER>.
/ (Version 1.00  5 June 1986)
//////////////////////////////////////////////////////////}

PROCEDURE Process_Esc;



{//////////////////////////////////////////////////////////
/ This procedure processes the up cursor ( up arrow ) key.
/ (Version 1.00  5 June 1986)
//////////////////////////////////////////////////////////}

  PROCEDURE Process_Up_Arrow;  { ESC A = UP ARROW }

    BEGIN { Process_Up_Arrow }
      row := row - 2;
      y := y - 1;
      IF (y < 1)
        THEN BEGIN { IF y < 1 }
          row := min;
          y := 1;
          Get_Menu_Selection := y;
      END ; { IF y < 1 }
    END ; { Process_Up_Arrow }
```

```
{////////////////////////////////////////////////////////////
/ This procedure processes the down cursor ( down arrow )
/ key.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

  PROCEDURE Process_Down_Arrow;  { ESC B = UP ARROW }

    BEGIN { Process_Down_Arrow }
      row := row + 2;
      y := y + 1;
      Get_Menu_Selection := y;
      IF ( y > max )
      THEN BEGIN { IF y > max }
        row := 2*max+min-2;
        y := max;
        Get_Menu_Selection := y;
      END; { IF y > max }
    END ; { Process_Down_Arrow }



{////////////////////////////////////////////////////////////}

BEGIN { Process_Esc }
  READ(KBD,chr2);
  CASE chr2 OF

    'A' : Process_Up_Arrow;     { ESC A = Up Arrow }
    'B' : Process_Down_Arrow;   { ESC B = Down Arrow }

  END; {CASE STATEMENT }

END ; { Process_Esc }
```

```
{/////////////////////SUB-PROCEDURE/////////////////////////
/ If a number key is pressed the user wants to select that
/ menu item.  This procedure converts the number key pressed
/ into its equivalent integer value and prepares the main
/ function, "Get_Menu_Selection", to return that value.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Process_Number_Key;

  VAR
    item_number : INTEGER ;

BEGIN { Process_Number_Key }
  Cursor( Block, Noblink, Noclick, Off ) ;
  item_number := INTEGER(chr1)-48 ;
  IF item_number <= max
    THEN BEGIN
      Get_Menu_Selection := item_number ;
      finished := TRUE ;
  END ; { IF item_number }
END ; { Process_Number_Key }



{////////////////////////////////////////////////////////////}

BEGIN {Get_Menu_Selection}
  row := min ;
  last := 30 + max ;
  y := 1 ;
  finished := FALSE ;
  REPEAT
    GOTOXY( column, row ) ;
    Cursor( Block, Noblink, Noclick, On ) ;
    READ(KBD,chr1) ;
    CASE chr1 OF

      cr        : Process_Cr ;
      esc       : Process_Esc;
      '1'..'9'  : Process_Number_Key;

    END; { CASE chr1 OF }

  UNTIL finished;

END ; {Get_Menu_Selection}
```

```
{*************************************************************
* This procedure sets the TIMAP options to their default
* settings.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Initialize_Defaults ;

  VAR
    index : INTEGER ;

BEGIN { Initialize_Defaults }
  specify_amp_type := FALSE ;
  specify_im_coef := FALSE ;
  FILLCHAR(path_name,SIZEOF(path_name),' ') ;
  path := '' ;
END ; { Initialize_Defaults }



{*************************************************************
* This procedure permits the user to change the setting of
* TIMAP's options.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Set_Options ;

  VAR
    Done : BOOLEAN ;
```

```
{/////////////////////SUB-PROCEDURE/////////////////////////
/ This procedure displays the purpose behind the Specify
/ Amplifier Type option, prompts the user for how he wants
/ the option set, checks the user's response for valid
/ input, and sets the option appropriately.  The OPTIONS
/ MENU is updated to show the present setting.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Set_Amplifier_Option ;

  VAR
    input_char    : CHAR ;
    answer        : STRING[5] ;
    good_input    : BOOLEAN ;

BEGIN { Set_Amplifier_Option }
  REPEAT
    GOTOXY( 1,16 ) ; Clrs2end ;
    WRITELN('When this option is TRUE you must specify the',
      ' type of amplifier (solid state or') ;
    WRITELN('tube) used in the final output stage of the ',
      'transmitters during data input.') ;
    WRITELN('When this option is FALSE a solid state type ',
      'of amplifier is assumed.') ;
    WRITELN('Set this option to FALSE for a worst case ',
      'analysis or if the amplifier types') ;
    WRITELN('are not known.');
    WRITELN ;
    WRITE('What do you want this option to be ',
      '(T or F) ?  ') ;
    Cursor( block, noblink, noclick, on );
    READ(KBD,input_char) ;
    Cursor( block, noblink, noclick, off );
    CASE UPCASE(input_char) OF
      'T' : BEGIN
              specify_amp_type := TRUE ;
              answer := 'True ' ;
              good_input := TRUE ;
            END ;
      'F' : BEGIN
              specify_amp_type := FALSE ;
              answer := 'False' ;
              good_input := TRUE ;
            END ;
      ELSE good_input := FALSE ;
    END ; { CASE UPCASE(input_char) }
  UNTIL good_input ;
  GOTOXY( 1,16 ) ; Clrs2end ;
  GOTOXY( 50,4 ) ; WRITE(answer) ;
END ; { Set_Amplifier_Option }
```

```
{/////////////////////SUB-PROCEDURE/////////////////////////
/ This procedure displays the purpose behind the Specify IM
/ Coefficient option, prompts the user for how he wants the
/ option set, checks the user's response for valid input,
/ and sets the option appropriately.  The OPTIONS MENU is
/ updated to show the present setting.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Set_IM_Coefficient_Option ;

  VAR
    input_char    : CHAR ;
    answer        : STRING[5] ;
    good_input    : BOOLEAN ;

BEGIN { Set_IM_Coefficient_Option }
  REPEAT
    GOTOXY( 1,16 ) ; Clrs2end ;
    WRITELN('When this option is TRUE you must enter the ',
      'intermodulation coefficient') ;
    WRITELN('spe ification (in dB) for each transmitter ',
      'during data input.  Intermodulation') ;
    WRITELN('coefficients are being catalogued by ECAC.  ',
      'Set this option to FALSE for a') ;
    WRITELN('worst case analysis or if the IM coefficients',
      ' are not known.') ;
    WRITELN ;
    WRITE('What do you want this option to be (T or F) ',
      '?  ') ;
    Cursor( block, noblink, noclick, on );
    READ(KBD,input_char) ;
    Cursor( block, noblink, noclick, off );
    CASE UPCASE(input_char) OF
      'T' : BEGIN
              specify_im_coef := TRUE ;
              answer := 'True ' ;
              good_input := TRUE ;
            END ;
      'F' : BEGIN
              specify_im_coef := FALSE ;
              answer := 'False' ;
              good_input := TRUE ;
            END ;
      ELSE good_input := FALSE ;
    END ; { CASE UPCASE(input_char) }
  UNTIL good_input ;
  GOTOXY( 1,16 ) ; Clrs2end ;
  GOTOXY( 50,6 ) ; WRITE(answer) ;
END ; { Set_IM_Coefficient_Option }
```

```
{/////////////////////SUB-PROCEDURE/////////////////////
/ This procedure displays the purpose behind the Set
/ Pathname option, prompts the user for how he wants the
/ option set, and sets the option appropriately.  The
/ OPTIONS MENU is updated to show the present setting.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////}

PROCEDURE Set_Path_Name ;

  VAR
    input_string : STRING[60] ;
    good_input   : BOOLEAN ;
    index        : INTEGER ;

BEGIN { Set_Path_Name }
  REPEAT
    GOTOXY( 1,15 ); Clrs2end ;
    WRITELN('This option allows you to specify the path ',
      'to your data directory.  The path') ;
    WRITELN('will be used for reading and writing data ',
      'files and writing output files.') ;
    WRITELN('Pressing <RETURN> will cause the default ',
      'drive and directory to be used.') ;
    WRITELN('The current path is :',^M^J);
    IF path = ''
      THEN WRITELN('The Default',^M^J)
      ELSE WRITELN(path,^M^J) ;
    WRITE('What do you want for a new path name ?  ');
    Cursor( block, noblink, noclick, on );
    READLN(path_name) ;
    Cursor( block, noblink, noclick, off );
    good_input := TRUE ;
    path := '' ;
    index := 1 ;
    WHILE (path_name[index] <> ' ') AND (index <= 60) DO
      BEGIN
        path := path + path_name[index] ;
        index := index + 1 ;
      END ; { WHILE }
  UNTIL good_input ;
  GOTOXY( 1,15 ) ; Clrs2end ;
  GOTOXY( 41,8 ) ; WRITELN('                    ') ;
  GOTOXY( 41,8 ) ;
  IF path = ''
    THEN WRITE('The Default')
    ELSE WRITE(COPY(path,1,20)) ;
END ; { Set_Path_Name }
```

```
{//////////////////////SUB-PROCEDURE//////////////////////////
/ This procedure returns the user to TIMAP's MAIN MENU.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Return_To_Main_Menu ;

BEGIN { Return_To_Main_Menu }
  Cursor( block, noblink, noclick, off ) ;
  done := TRUE ;
END ; { Return_To_Main_Menu }




{////////////////////////////////////////////////////////////}

BEGIN { Set_Options }
  Display_Options_Menu ;
  Done := FALSE ;
  REPEAT ;
    CASE ( Get_Menu_Selection( 4, 4, 22 ) ) OF
      1 : Set_Amplifier_Option ;
      2 : Set_IM_Coefficient_Option ;
      3 : Set_Path_Name ;
      4 : Return_To_Main_Menu ;
    END ; { CASE STATEMENT }
  UNTIL Done ;
  cursor(Block, Noblink, Noclick, On ) ;
END ; { Set_Options }
```

```
{************************************************************
* This procedure outputs the currently loaded data to a text
* file in a format suitable for printing or editing with a
* word processor.  (Version 1.00  5 June 1986)
************************************************************}

OVERLAY PROCEDURE Print_Data ;

  VAR
    freq_file        : TEXT ;
    ant_file         : TEXT ;
    destination_file : STRING[8] ;
    file_ready       : BOOLEAN ;

{/////////////////////SUB-PROCEDURE/////////////////////////
/ This procedure displays reminders to the user about how
/ specify a filename or how to return to the MAIN MENU
/ without printing the data.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Display_Instructions;

  VAR
    index : INTEGER ;

BEGIN { Display_Instructions }
  Reverse_Video ;
  Cwrite
    ('YOUR FILENAME MAY CONTAIN UP TO 8 CHARACTERS.', 18 );
  Cwrite('DO NOT INCLUDE A FILE EXTENSION.', 19 ) ;
  Cwrite
    ('Press <RETURN> without entering a filename', 21);
  Cwrite(CONCAT
    ('to return to the MAIN MENU without saving the ',
     'data.'), 22);
  Normal_Video ;
  GOTOXY(1,5) ;
  WRITE('These are the print files already on ') ;
  IF path = ''
    THEN WRITE('the default drive')
    ELSE WRITE(path) ;
END ; { Display_Instructions }
```

```
{/////////////////////SUB-PROCEDURE/////////////////////////
/ This procedure prompts the user for a name for the file
/ contain the printable data, checks for valid input, and
/ prepares the file to accept the data.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Get_Filename ;

  VAR
    result : BYTE ;
    size   : INTEGER ;

BEGIN { Get_Filename }
  REPEAT
    GOTOXY( 1, 2 ) ; Clrline ;
    WRITE('What do you want to call your print files ?  ') ;
    Cursor( block, noblink, noclick, on ) ;
    READLN( destination_file ) ;
    Cursor( block, noblink, noclick, off ) ;
    IF ( LENGTH( destination_file ) = 0 )
      THEN BEGIN
        file_ready := FALSE ;
        EXIT;
    END ; { IF }
    size := pos('.',destination_file) ;
    IF size <> 0
      THEN destination_file := COPY(destination_file,1,size-1) ;
    ASSIGN(freq_file,path+destination_file+'.FPR');
    {$I-} REWRITE( freq_file ) {$I+} ;
    result := IOresult ;
    file_ready := ( result = 0 ) ;
    IF (result = $F1) THEN Directory_Full ;
    ASSIGN(ant_file,path+destination_file+'.APR');
    {$I-} REWRITE( ant_file ) {$I+} ;
    result := IOresult ;
    file_ready := ( result = 0 ) ;
    IF (result = $F1) THEN Directory_Full ;
  UNTIL file_ready ;
END ; { Get_Filename }
```

```
{//////////////////////SUB-PROCEDURE//////////////////////
/ This procedure writes the antenna data to a disk file.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////}

PROCEDURE Write_Antenna_Data_To_File(antenna_no : INTEGER) ;

  VAR
    index, index2 : INTEGER ;

{//////////////////////SUB-PROCEDURE//////////////////////
/ This procedure writes column headings for the antenna
/ information to the disk.  (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////}

PROCEDURE Write_Antenna_Headings ;

BEGIN { Write_Antenna_Headings }
  WRITELN(ant_file,'Antenna data printed at ',Time,' on ',
          Date,'.',^M^J) ;
  WRITELN(ant_file,'ANTENNA              X                Y',
          '            HEIGHT     GAIN') ;
  WRITELN(ant_file,' NUMBER          COORDINATE     ',
          'COORDINATE');
  WRITELN(ant_file,'                    (FEET)          ',
          '(FEET)        (FEET)     (dBi)',^M^J) ;
END ; { Write_Antenna_Headings }

{////////////////////////////////////////////////////////}

BEGIN { Write_Antenna_Data_To_File }
  IF antenna_no > no_o_antennas THEN BEGIN
      CLOSE(ant_file) ;
      EXIT ;
  END ; { IF }
  Write_Antenna_Headings ;
  index := 6 ;  { Accounts for the six lines of heading }
  WHILE (antenna_no <= no_o_antennas) AND
        (index <= 52 ) DO BEGIN
    WITH ant_list[antenna_no] DO BEGIN
      WRITE(ant_file,antenna_no:5) ;
      WRITE(ant_file,Spc(11),xcoord:7:2) ;
      WRITE(ant_file,Spc(8),ycoord:7:2) ;
      WRITE(ant_file,Spc(7),zcoord:7:2) ;
      WRITELN(ant_file,Spc(3),gain:7:2  ) ;
      antenna_no := antenna_no + 1 ;
      index := index + 1 ;
    END ; { WITH ant_list }
  END ; { WHILE antenna_no }
  FOR index2 := (index+1) TO 66 DO WRITELN(ant_file) ;
  Write_Antenna_Data_To_File( antenna_no ) ;
END ; { Write_Antenna_Data_To_File }
```

```
{//////////////////////SUB-PROCEDURE//////////////////////////
/ This procedure writes the frequency data to a disk file.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Write_Frequency_Data_To_File( frequency_no
                                        : INTEGER ) ;

  VAR
    index  : INTEGER ;
    index2 : INTEGER ;

{//////////////////////SUB-PROCEDURE//////////////////////////
/ This procedure writes the current time and date as a
/ heading to the frequency data file.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Write_Frequency_Headings ;

BEGIN { Write_Frequency_Headings }
  WRITELN(freq_file,'Frequency data printed at ',Time,
          ' on ',Date,'.',^M^J) ;
END ; { Write_Frequency_Headings }




{////////////////////////////////////////////////////////////}

BEGIN { Write_Frequency_Data_To_File }
  IF frequency_no > no_o_frequencies THEN BEGIN
      CLOSE(freq_file) ;
      EXIT ;
  END ; { IF }
  Write_Frequency_Headings ;
  index := 1 ;
  WHILE (frequency_no <= no_o_frequencies) AND
        (index <= 4 ) DO BEGIN
    WITH freq_list[frequency_no], ant_list[antenna_no]
      DO BEGIN
      WRITELN(freq_file,'Frequency : ',frequency:7:2,' MHz',
              Spc(13),'Antenna Number : ',antenna_no:3);
      WRITELN(freq_file,'Antenna X Coordinate : ',
              xcoord:7:2, Spc(6),'Antenna Y Coordinate : ',
              ycoord:7:2);
      WRITELN(freq_file,'Antenna Height : ',zcoord:7:2,
              ' feet',Spc(7),'Antenna Gain : ',gain:7:2,
              ' dBi') ;
      WRITELN(freq_file,'Bandwidth : ', bandwidth:7:2,
              ' kHz.') ;
```

```
        CASE usage OF
          'T' : WRITELN(freq_file,'Assigned as a transmit ',
                        'only frequency.');
          'R' : WRITELN(freq_file,'Assigned as a receive ',
                        'only frequency.') ;
          'B' : WRITELN(freq_file,'Assigned as both a ',
                        'transmit and receive frequency.') ;
        END ; { CASE usage }

        CASE time OF
          'D' : WRITELN(freq_file,'For day time use.') ;
          'N' : WRITELN(freq_file,'For night time use.') ;
          'B' : WRITELN(freq_file,'For day and night time ',
                        'use.') ;
          'C' : WRITELN(freq_file,'For use in a ',
                        'contingency.') ;
        END ; { CASE time }

        WRITELN(freq_file,'Cable Loss : ', cable_loss:7:2,
                ' dB') ;
        IF (usage = 'T') OR (usage = 'B') THEN BEGIN
          WRITE(freq_file,'Transmitter Output Power : ');
          WRITELN(freq_file,output_power:7:2,' Watts') ;
          WRITE(freq_file,'Transmitter''s Final Output ',
                'Amplifier ');
          IF im_coef_b = 30
            THEN WRITELN(freq_file,'Type : Solid State')
            ELSE WRITELN(freq_file,'Type : Tube') ;
          WRITELN(freq_file,'Transmitter Intermodulation ',
                  'Coefficient : ',im_coef_K1:7:2,' dB') ;
        END ; { IF usage = T or B }
        WRITELN(freq_file,^M^J) ;
        frequency_no := frequency_no + 1 ;
        index := index + 1 ;
      END ; { WITH freq_list }
    END ; { WHILE frequency_no }
    FOR index2 := 3+(index-1)*12 TO 66 DO WRITELN(freq_file) ;
    Write_Frequency_Data_To_File( frequency_no ) ;
  END ; { Write_Frequency_Data_To_File }

  {////////////////////////////////////////////////////////////}

  BEGIN { Print_Data }
    CLRSCR;
    Display_Instructions ;
    Display_Directory(7,'????????.?PR') ;
    Get_Filename ;
    IF file_ready THEN BEGIN
        Write_Antenna_Data_To_File( 1 ) ;
        Write_Frequency_Data_To_File( 1 ) ;
    END ; { IF file_ready }
  END ; { Print_Data }
```

```
{*************************************************************
* This procedure displays a list of all data files in the
* current directory, prompts the user for the name of the
* file to store the data in and saves the antenna and
* frequency data to the disk..
* (Version 1.00  5 June 1986)
*************************************************************}

OVERLAY PROCEDURE Save_Data ;

  VAR
    freq_file        : FILE OF frequency_info ;
    ant_file         : FILE OF antenna_info ;
    destination_file : STRING[8] ;
    file_ready       : BOOLEAN ;

{*************************************************************
* This procedure displays a reminder to the user about
* entering a filename for the  file to be used for saving
* the data and how to exit this function without saving the
* data.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Display_Instructions;

  VAR
    index : INTEGER ;

BEGIN { Display_Instructions }
  Reverse_Video ;
  Cwrite
    ('YOUR FILENAME MAY CONTAIN UP TO 8 CHARACTERS.', 18 );
  Cwrite('DO NOT INCLUDE A FILE EXTENSION.', 19 ) ;
  Cwrite
    ('Press <RETURN> without entering a filename', 21);
  Cwrite(CONCAT
    ('to return to the main menu without saving the ',
     'data.'), 22);
  Normal_Video ;
  GOTOXY(1,5) ;
  WRITE('These are the data files already on ') ;
  IF path = ''
    THEN WRITE('the default drive')
    ELSE WRITE(path) ;
END ; { Display_Instructions }
```

```
{************************************************************
* This procedure prompts the user for the name of a file
* to save the data in, checks for valid input, and prepares
* the file for faccepting the data.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Get_Filename ;

  VAR
    result : BYTE ;
    size   : INTEGER ;

BEGIN { Get_Filename }
  REPEAT
    GOTOXY( 1, 2 ) ; Clrline ;
    WRITE('What do you want to call your data file ?  ') ;
    Cursor( block, noblink, noclick, on ) ;
    READLN( destination_file ) ;
    Cursor( block, noblink, noclick, off ) ;
    IF ( LENGTH( destination_file ) = 0 )
      THEN BEGIN
        file_ready := FALSE ;
        EXIT;
    END ; { IF }
    size := pos('.',destination_file) ;
    IF size <> 0
      THEN destination_file := COPY(destination_file,
                                    1,size-1) ;
    ASSIGN(freq_file,path+destination_file+'.FQD');
    {$I-} REWRITE( freq_file ) {$I+} ;
    result := IOresult ;
    file_ready := ( result = 0 ) ;
    IF (result = $F1) THEN Directory_Full ;
    ASSIGN(ant_file,path+destination_file+'.ATD');
    {$I-} REWRITE( ant_file ) {$I+} ;
    result := IOresult ;
    file_ready := ( result = 0 ) ;
    IF (result = $F1) THEN Directory_Full ;
  UNTIL file_ready ;
END ; { Get_Filename }
```

```
{*************************************************************
* This procedure writes the antenna and frequency data to
* the disk file.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Write_Data_To_File ;

  VAR
    index : INTEGER;

BEGIN { Write_Data_To_Disk }
  FOR index := 1 TO no_o_frequencies DO
    WRITE(freq_file,freq_list[index]);
  CLOSE(freq_file);
  FOR index := 1 TO no_o_antennas DO
    WRITE(ant_file,ant_list[index]);
  CLOSE(ant_file);
  data_changed := FALSE ;
END ; { Write_Data_To_Disk }



{//////////////////////MAIN-PROCEDURE////////////////////////}

BEGIN { Save_Data }
  CLRSCR;
  Display_Instructions ;
  Display_Directory(7,'????????.??D') ;
  Get_Filename ;
  IF file_ready THEN Write_Data_To_File ;
END ; { Save_Data }
```

```
{************************************************************
* This procedure displays a menu allowing the user to
* display the currently loaded antenna or frequency data,
* or to return to TIMAP's MAIN MENU.  The antenna data
* is displayed in a columnar format with each row
* providing all the information on a particular antenna.
* The frequency information is displayed with all
* information on one frequency displayed on the screen at
* a time.  (Version 1.00  5 June 1986)
************************************************************}

OVERLAY PROCEDURE Display_Data ;

  VAR
    done        : BOOLEAN ;

{************************************************************
* This function is used to compare the present frequency
* or antenna number to the highest frequency or antenna
* number to determine if any more data is available.  It
* also prompts the user to press a space bar (for more
* data) or the <RETURN> to return to the DISPLAY MENU.
* (Version 1.00  5 June 1986)
************************************************************}

FUNCTION No_More( number1, number2 : INTEGER) :
          {RETURN} BOOLEAN ;

  CONST
    Space       = ' ' ;

  VAR
    input_char : CHAR ;
    good_input : BOOLEAN ;

BEGIN { No_More }
  no_more := FALSE ;
  Reverse_Video ;
  Cwrite('Press <SPACE BAR> for more or <RETURN>', 22) ;
  Cwrite('to return to the DISPLAY MENU.', 23) ;
  Normal_Video ;
  REPEAT
    good_input := TRUE ;
    READ(KBD,input_char) ;
    CASE input_char OF
      Cr    : no_more := TRUE ;
      Space : IF (number1 > number2)
                THEN no_more := TRUE ;
                ELSE good_input := FALSE ;
    END ; { CASE input_char }
  UNTIL good_input ;
END ; { No_More }
```

```
{************************************************************
* This procedure displays the antenna data one screen at a
* time beginning at antenna number 1.  The information for
* each antenna is in one row and several antennas' data
* are displayed at once in columnar form.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Display_Antenna_Data ;

  VAR
    antenna_no    : INTEGER ;

{//////////Sub-Procedure of Display_Antenna_Data//////////
/ This procedure displays a heading at the top of the
/ screen to identify the antenna data in the columns.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////}

PROCEDURE Display_Antenna_Headings ;

BEGIN { Display_Antenna_Headings }
  CLRSCR ;
  GOTOXY( 1,1 )   ; WRITE('ANTENNA') ;
  GOTOXY( 1,2 )   ; WRITE(' NUMBER') ;
  GOTOXY( 16,1 ) ; WRITE('    X') ;
  GOTOXY( 16,2 ) ; WRITE('COORDINATE') ;
  GOTOXY( 16,3 ) ; WRITE(' (FEET)') ;
  GOTOXY( 30,1 ) ; WRITE('    Y') ;
  GOTOXY( 30,2 ) ; WRITE('COORDINATE') ;
  GOTOXY( 30,3 ) ; WRITE(' (FEET)') ;
  GOTOXY( 45,1 ) ; WRITE('HEIGHT') ;
  GOTOXY( 45,3 ) ; WRITE('(FEET)') ;
  GOTOXY( 55,1 ) ; WRITE('GAIN') ;
  GOTOXY( 55,3 ) ; WRITELN('(dBi)') ;
END ; { Display_Antenna_Headings }
```

```
{//////////Sub-Procedure of Display_Antenna_Data//////////
/ This procedure will display a single screen full of
/ antenna data.  A message is displayed to the user when
/ the end of the antenna list is reached.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////}

PROCEDURE Display_One_Screen_Of_Antennas ;

  VAR
    index       : INTEGER ;

BEGIN { Display_One_Screen_Of_Antennas }
  Display_Antenna_Headings ;
  index := 1 ;
  WHILE (antenna_no <= no_o_antennas) AND
        (index < 16 ) DO BEGIN
    WITH ant_list[antenna_no] DO BEGIN
      GOTOXY(1,index+4)   ; WRITE(antenna_no:5) ;
      GOTOXY(15,index+4)  ; WRITE(xcoord:8:2) ;
      GOTOXY(29,index+4)  ; WRITE(ycoord:8:2) ;
      GOTOXY(43,index+4)  ; WRITE(zcoord:8:2) ;
      GOTOXY(53,index+4)  ; WRITE(gain:8:2  ) ;
      antenna_no := antenna_no + 1 ;
      index := index + 1 ;
    END ; { WITH ant_list }
  END ; { WHILE antenna_no }
  IF antenna_no > no_o_antennas
    THEN BEGIN
      Reverse_Video ;
      Cwrite('No More Antennas', index+4) ;
      Normal_Video ;
  END ; { IF antenna_no }
END ; { Display_One_Screen_Of_Antennas }

{////////////////////////////////////////////////////////}

BEGIN { Display_Antenna_Data }
  antenna_no := 1 ;
  REPEAT
    Display_One_Screen_Of_Antennas ;
  UNTIL No_More(antenna_no,no_o_antennas) ;
END ; { Display_Antenna_Data }
```

```
{**************************************************************
* This procedure displays the frequency data in increasing
* order by frequency one screen at a time.
* (Version 1.00  5 June 1986)
**************************************************************}

PROCEDURE Display_Frequency_Data ;

  VAR
    freq_no : INTEGER ;

{/////////Sub-Procedure of Display_Frequency_Data//////////
/ This procedure displays a single screen full of frequency
/ information.  All information available on a single
/ frequency is displayed.
/ (Version 1.00  5 June 1986)
////////////////////////////////////////////////////////////}

PROCEDURE Display_One_Screen_Of_Frequencies ;

BEGIN { Display_One_Screen_Of_Frequencies }
  CLRSCR ;
  IF freq_no > no_o_frequencies THEN BEGIN
    Reverse_Video ;
    GOTOXY( 31,12 ) ; WRITELN('No More Frequencies') ;
    Normal_Video ;
    EXIT ;
  END ; { IF freq_no }
  WITH freq_list[freq_no], ant_list[antenna_no]  DO BEGIN
    WRITELN;
    WRITE('Frequency : ', frequency:7:2,' MHz') ;
    GOTOXY(40,2) ;
    WRITELN('Antenna Number : ',antenna_no:3) ;
    WRITE('Antenna X Coordinate : ',xcoord:7:2) ;
    GOTOXY(40,3) ;
    WRITELN('Antenna Y Coordinate : ',ycoord:7:2);
    WRITE('Antenna Height : ',zcoord:7:2,' feet') ;
    GOTOXY(40,4) ;
    WRITELN('Antenna Gain : ',gain:7:2,' dBi') ;
    WRITE('Bandwidth : ', bandwidth:7:2,' kHz.') ;
    GOTOXY(40,5) ;
    WRITELN('Cable Loss : ',cable_loss:7:2,' dB') ;
    CASE usage OF
      'T' : WRITELN('Assigned as a transmit only ',
                    'frequency.');
      'R' : WRITELN('Assigned as a receive only ',
                    'frequency.') ;
      'B' : WRITELN('Assigned as both a transmit and ',
                    'receive frequency.') ;
    END ; { CASE usage }
```

```
      CASE time OF
        'D' : WRITELN('For day time use.') ;
        'N' : WRITELN('For night time use.') ;
        'B' : WRITELN('For day and night time use.') ;
        'C' : WRITELN('For use in a contingency.') ;
      END ; { CASE time }
      IF (usage = 'T') OR (usage = 'B') THEN BEGIN
        WRITE('Transmitter Output Power : ');
        WRITELN(output_power:7:2,' Watts') ;
        WRITE('Final Output Amplifier Type : ') ;
        IF im_coef_b = 30
          THEN WRITELN('Solid State')
          ELSE WRITELN('Tube') ;
        WRITE('Transmitter Intermodulation Coefficient : ');
        WRITELN(im_coef_K1:7:2,' dB') ;
      END ; { IF usage = T or B }
      freq_no := freq_no + 1 ;
    END ; { WITH freq_list }
  END ; { Display_One_Screen_Of_Frequencies }

{////////////////////////////////////////////////////////////}

BEGIN { Display_Frequency_Data }
  freq_no := 1 ;
  REPEAT
    Display_One_Screen_Of_Frequencies ;
  UNTIL No_More(freq_no,no_o_frequencies) ;
END ; { Display_Frequency_Data }



{************************************************************
* This procedure returns the user to TIMAP's MAIN MENU.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Return_To_Main_Menu ;

BEGIN { Return_To_Main_Menu }
  Cursor( block, noblink, noclick, off ) ;
  done := TRUE ;
END ; { Return_To_Main_Menu }
```

```
{/////////////////////////////////////////////////////////////////////////}

BEGIN { Display_Data }
  REPEAT ;
    Display_Display_Menu ;
    done := FALSE ;
    CASE ( Get_Menu_Selection( 4, 3, 28 ) ) OF
      1 : Display_Antenna_Data ;
      2 : Display_Frequency_Data ;
      3 : Return_To_Main_Menu ;
    END ; { CASE STATEMENT }
  UNTIL done ;
END ; { Display_Data }
```

```
{************************************************************
* This function gets the date from the operating system and
* converts it to a string.  (Version 1.00  5 June 1986)
************************************************************}

FUNCTION date
  { RETURN } : iostring ;


  VAR
    Day   : STRING[2] ;
    Month : STRING[9] ;
    Year  : STRING[4] ;
    Regs: RECORD CASE INTEGER OF
            1: (AX,BX,CX,DX,BP,DI,SI,DS,ES,Flags: INTEGER);
            2: (AL,AH,BL,BH,CL,CH,DL,DH: Byte);
          END;

BEGIN { Date }
  WITH Regs DO BEGIN
    AH:=$2A;
    Flags:=0;
    MsDos(Regs);
    STR(CX:4,Year);
    STR(DL:2,Day);
    CASE DH OF
       1 : Month := 'January' ;
       2 : Month := 'February' ;
       3 : Month := 'March' ;
       4 : Month := 'April' ;
       5 : Month := 'May' ;
       6 : Month := 'June' ;
       7 : Month := 'July' ;
       8 : Month := 'August' ;
       9 : Month := 'September' ;
      10 : Month := 'October' ;
      11 : Month := 'November' ;
      12 : Month := 'December' ;
    END ; { CASE DH }
  END; { With Regs }
  DATE := Day+' '+Month+' '+Year ;
END ; { Date }
```

```
{************************************************************
* This function gets the time from the operating system and
* returns it as a string (in 24 hour format).
* (Version 1.00  5 June 1986)
************************************************************}

FUNCTION Time : iostring ;


  PROCEDURE ZeroFill(VAR S: iostring);

    VAR
      I: INTEGER;

    BEGIN
      FOR I:=1 TO LENGTH(S) DO IF S[I]=' ' THEN S[I]:='0';
    END;


  VAR
    Hour, Min : STRING[2];
    Timestring : iostring ;
    Regs: RECORD CASE INTEGER OF
            1: (AX,BX,CX,DX,BP,DI,SI,DS,ES,Flags: INTEGER);
            2: (AL,AH,BL,BH,CL,CH,DL,DH: Byte);
          END;

BEGIN { Time }
  WITH Regs DO BEGIN
    AH:=$2C;
    Flags:=0;
    MsDos(Regs);
    STR(CH:2,Hour);
    STR(CL:2,Min);
    TimeString:=Hour+':'+Min ;
    ZeroFill(TimeString);
    Time := TimeString ;
  END ; { WITH Regs }
END ; { Time }
```

```
{************************************************************
* This procedure centers the "input_string" on the screen
* at line number "line".  (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Cwrite( input_string : iostring ;
                            line : INTEGER ) ;

BEGIN { Cwrite }
  GOTOXY(41-ROUND(LENGTH(input_string)/2), line ) ;
  WRITE( input_string ) ;
END ; { Cwrite }



{************************************************************
* This procedure positions the cursor at column number
* "column" on the current line of the screen.
* (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Tab(column : INTEGER);

  VAR
    cursor_position : STRING[3] ;
    line            : INTEGER ;

BEGIN { Tab }
  cursor_position := '   ' ;
  WRITE (#27,'n');
  READ(KBD,cursor_position) ;
  line := ORD(cursor_position[2])-31 ;
  GOTOXY( column, line ) ;
END ; { Tab }
```

```
{**************************************************************
* This function returns a string of "number" spaces.
* (Version 1.00  5 June 1986)
**************************************************************}

FUNCTION Spc( column : INTEGER ) : iostring ;

  VAR
    temp  : iostring ;
    index : INTEGER ;

BEGIN { Spc }
  temp := '' ;
  FOR index := 1 TO (column - 1) DO temp := temp + ' ' ;
  Spc := temp ;
END ; { Spc }
```

```
{************************************************************
* This procedure is an adaptation of a sample program given
* in Turbo Tutor.  It provides a directory listing of all
* applicable data files on the default (currently logged)
* drive.  (Version 1.00  5 June 1986)
************************************************************}

PROCEDURE Display_Directory( line : INTEGER;
                             filespec : Char12arr ) ;

TYPE
  String20              = STRING[ 20 ] ;
  RegRec =
    RECORD
      AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : INTEGER ;
    END ;

VAR
  Regs                  : RegRec ;
  DTA                   : ARRAY [ 1..43 ] OF BYTE ;
  Mask                  : Char60arr ;
  NamR                  : String20 ;
  Error, I              : INTEGER ;
  Column                : INTEGER ;
  Index                 : INTEGER ;
  array_length          : INTEGER ;

BEGIN { Display_Directory }
  FILLCHAR(DTA,SIZEOF(DTA),0);          { Initialize the DTA buffer }
  FILLCHAR(mask,SIZEOF(mask),' ');      { Initialize the mask }
  FILLCHAR(NamR,SIZEOF(NamR),0);        { Initialize the file name }
  Column := 1;                          { Begin the lising in the fi
  index := 1 ;
  array_length := 0 ;
  WHILE (path_name[index] <> ' ') AND (array_length <= 60) DO
    BEGIN
      array_length := array_length + 1 ;
      index := index + 1 ;
  END ;
```

```
    Regs.AX := $1A00;           { Function used to set the DTA }
    Regs.DS := SEG(DTA);        { store the parameter segment in DS }
    Regs.DX := OFS(DTA);        {   "     "       "     offset in DX }
    MSDOS(Regs);                { Set DTA location }
    Error := 0;
    FOR index := 1 TO array_length DO
      Mask[index] := path_name[index] ;
    FOR index := array_length+1 TO array_length+12 DO
      Mask[index] := filespec[index-array_length] ;
    Regs.AX := $4E00;           { Get first directory entry }
    Regs.DS := Seg(Mask);       { Point to the file Mask }
    Regs.DX := Ofs(Mask);
    Regs.CX := 22;              { Store the option }
    MSDOS(Regs);                { Execute MSDos call }
    Error := Regs.AX AND $FF;   { Get Error return }
    I := 1;                     { initialize 'I' to the first element
    IF (Error = 0) THEN
      REPEAT
        NamR[I] := CHR(MEM[SEG(DTA):OFS(DTA)+29+I]);
        I := I + 1;
      UNTIL NOT (NamR[I-1] IN [' '..'~']) OR (I>20);

    NamR[0] := CHR(I-1);          { set string length because assign
                                  { by element does not set length }
    GOTOXY( column, line ) ;
    WRITE( NamR ) ;
    column := column + 16 ;

    WHILE (Error = 0) DO BEGIN
      Error := 0;
      Regs.AX := $4F00;           { Function used to get the next }
                                  { directory entry }
      Regs.CX := 22;              { Set the file option }
      MSDOS( Regs );              { Call MSDos }
      Error := Regs.AX AND $FF;   { get the Error return }
      I := 1;
      REPEAT
        NamR[I] := CHR(MEM[SEG(DTA):OFS(DTA)+29+I]);
        I := I + 1;
      UNTIL NOT (NamR[I-1] IN [' '..'~'] ) OR (I > 20);
      NamR[0] := CHR(I-1);
      IF (Error = 0) THEN BEGIN
        GOTOXY( column, line ) ; WRITE(NamR) ;
        column := column + 16 ;
        IF column > 67 THEN BEGIN
          column := 1;
          line := line + 1;
        END ; { IF column }
      END ; { IF Error }
    END ; { WHILE Error }
  END ; { Display_Directory }
```

```
{*************************************************************
* This procedure prints an error message to the user when
* a write is attempted and there is no room left in the
* current directory.  (Version 1.00  5 June 1986)
*************************************************************}

PROCEDURE Directory_Full ;

  CONST
    bell = #7 ;

BEGIN { Directory_Full }
  WRITELN( bell ) ;
  GOTOXY( 1 , 16 ) ; Clrline ;
  Cwrite('Sorry, your directory is full.', 16) ;
  Cwrite('Insert a new disk and try again.', 18) ;
END ; { Directory_Full }
```

APPENDIX C

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Atlantic Research Corporation. Interference Notebook. Griffiss Air Force Base, New York: Rome Air Development Center, August 1969. (RADC-TR-66-1, Vol. 2)

2. Catalog Of Computer Program Abstracts - Volume I - Analysis Capabilities. Annapolis, Maryland: Electromagnetic Compatibility Analysis Center, March 1985. (ECAC-HDBK-85-047-1)

3. Denny, H.W., D.W. Acree, J.C. Mantovani. EMC Measurement Techniques For Aircraft. Griffiss Air Force Base, New York: Rome Air Development Center, July 1983. (RADC-TR-83-166)

4. Gagliardi, Robert M. Introduction To Communications Engineering. New York, New York: John Wiley & Sons, Inc., 1978.

5. Handbook On Radio Frequency Interference - Volume 1 - Fundamentals Of Electromagnetic Interference. Wheaton, Maryland: Frederick Research Corporation, 1962.

6. Handbook On Radio Frequency Interference - Volume 2 - Fundamentals Of Electromagnetic Interference Prediction And Measurement. Wheaton, Maryland: Frederick Research Corporation, 1962.

7. Keiser, Bernhard E. Principles Of Electromagnetic Compatibility. Dedham, Massachusetts: Artech House, Inc., 1979.

8. Lustgarten, M., A. Rosen, M. Maiuzzo. Cosite Analysis Handbook, Volume I: Cosite Analysis Procedures. Annapolis, Maryland: Electromagnetic Compatibility Analysis Center, March 1984. (ECAC-HDBK-82-057A)

9. Maiuzzo, Michael A., Edward Mackouse. "Transmitter Intermodulation Product Amplitudes," IEEE International Symposium On Electromagnetic Compatibility, 133-138 (1981)

10. Stone, Glenn A., Glen E. Bahr. EMC/EMI Study For Collocation Waiver Of VHF/UHF Radios At Malstrom AFB, MT. Report AFCC 485 EIG-SES-83-11. Griffiss AFB, New York: 485 EIG/EIEUS Special Engineering Section, May 1983.

11. Wheeler, D. Cosite Analysis User's Manual Volume I: COSAM II (DECAL/PECAL) User's Manual. Annapolis, Maryland: Electromagnetic Compatibility Analysis Center, December 1984. (ECAC-UM-84-009-1)

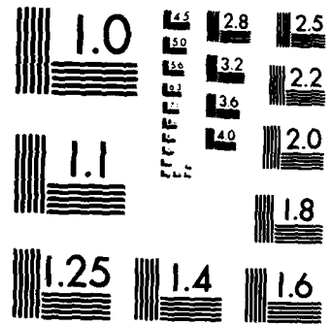
12. White, Donald R. J. Electromagnetic Interference And Compatibility - Volume 3 - EMI Control Methods And Techniques. Gainesville, Virginia: Don White Consultants, Inc., 1981.

13. White, Donald R. J. Electromagnetic Interference And Compatibility - Volume 5 - EMI Prediction And Analysis Techniques. Germantown, Maryland: Don White Consultants, Inc., 1972.

APPENDIX D

VITA

AD-A172 932 ANALYSIS OF TRANSMITTER PRODUCED INTERMODULATION 4/4
INTERFERENCE IN COLOCATE (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI T J ZUZACK
UNCLASSIFIED JUN 86 AFIT/GE/ENG/86J-3 F/G 20/14 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

VITA

Captain Thomas J. Zuzack was born on 12 February 1953 in Latrobe, Pennsylvania. He graduated from Greater Latrobe Senior High School in 1971 and that same year enlisted in the United States Air Force. After basic training, Captain Zuzack received ten months of electronics training and became a digital subscriber terminal and cryptographic equipment systems repairman. In 1974 Captain Zuzack married Miss Mia C. Cho and in 1975 he entered the Airman Scholarship and Commissioning Program. Captain Zuzack attended Oklahoma State University where he was elected to membership in the Phi Kappa Phi National Honor Society and, in 1977, earned the degree of Bachelor of Science in Engineering Technology. Upon graduation he was commissioned a Second Lieutenant and became a distinguished graduate of both the Communications-Electronics Officers School at Keesler AFB, Mississippi and the Wideband Systems Evaluation School at Scott AFB, Illinois. Captain Zuzack is listed in the Outstanding Young Men of America for 1985 and in that year was also elected to membership in the Tau Beta Pi National Engineering Honor Society and earned the degree of Bachelor of Science in Electrical Engineering from the Air Force Institute of Technology. He then continued on at the Air Force Institute of Technology pursuing a masters degree in Electrical Engineering.

Permanent Address : 1214 Waverly Drive
Latrobe, PA. 15650

APPENDIX E

5.25" FLOPPY DISK CONTAINING SOURCE CODE

AND EXECUTABLE FILES

FOR

TIMAP - VERSION 1.00

BY

THOMAS J. ZUZACK

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/86J-1	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 1843 Engineering Installation Group 8b. OFFICE SYMBOL (If applicable) 1843 EIG/EIEM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code) Hickam Air Force Base Hawaii 96853	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) See Box 19	
12. PERSONAL AUTHOR(S) Thomas J. Zuzack, Captain, USAF	
13a. TYPE OF REPORT MS Thesis 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1986 June 15. PAGE COUNT 291
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB. GR.
09	02,04	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Electromagnetic Compatibility, Intermodulation, Electromagnetic Interference, Interference

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

Title: ANALYSIS OF TRANSMITTER PRODUCED INTERMODULATION INTERFERENCE IN COLOCATED VHF SITES USING A MICROCOMPUTER.

Thesis Chairman: David A. King, Captain, USAF
Instructor of Electrical Engineering

Approved for public release: IAW AFR 190-1.
LYNN E. WOLAVER 3 Jul 86
Dean for Research and Professional Development
Air Force Institute of Technology (AFIT)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL David A. King, Captain, USAF	22b. TELEPHONE NUMBER (Include Area Code) (513) 255-3576 22c. OFFICE SYMBOL AFIT/ENG

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS

This thesis presents the theory behind the generation of intermodulation products in the final output power amplifier of a VHF transmitter and the calculations necessary to determine the power in a transmitter produced intermodulation product at the front end of a receiver. The results are then applied to the analysis of transmitter produced intermodulation interference in colocated VHF sites. An algorithm is developed for accomplishing this analysis and is implemented in the form of a Turbo Pascal program named TIMAP (Transmitter Intermodulation Analysis Program) which runs on the Zenith Z-100 microcomputer under the MS-DOS operating system. TIMAP provides a systematic and automated analysis tool for electromagnetic compatibility engineers in the field.

END

11-86

DTIC