# DEVELOPMENT OF A DIGITAL SIGNAL ANALYSIS SYSTEM FOR MINICOMPUTERS

By

JOHN EDWARD PERRAULT, JR.

Bachelor of Science in Mechanical Engineering

University of Tulsa

Tulsa, Oklahoma

1975

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the degree of
MASTER OF SCIENCE
May, 1977

Thesis
1977
P454d
cop.2

# DEVELOPMENT OF A DIGITAL SIGNAL ANALYSIS SYSTEM FOR MINICOMPUTERS

Thesis Approved:

Thesis Adviser

Dean of the Graduate College

977118

## ACKNOWLEDGEMENTS

During my graduate studies I received many ideas and much encouragement from my friends, my colleagues, and the entire faculty and staff of the School of Mechanical and Aerospace Engineering. I am grateful to them all.

I thank Dr. R. L. Lowery, my thesis adviser, and the other members of my committee, Dr. L. D. Zirkle, Dr. J. A. Wiebelt, and Dr. R. P. Rhoten for their advice and criticisms. I also thank Dr. L. R. Ebbesen for all his help and suggestions during the course of my work with the minicomputer and other studies.

My parents deserve a special thanks for the advice and encouragement they gave throughout my academic career. I thank Mrs. Mary Jane Hoag and Mrs. Janna Hemphill for typing the various drafts of this thesis.

And to Debbie, a very special thanks for her patience, understanding, and love.

TABLE OF CONTENTS


| Chapter | | Page |
|---|---|---|
| I. | INTRODUCTION | 1 |
| II. | OVERVIEW OF SIGNAL ANALYSIS | 4 |
| | The Fast Fourier Transform | 5 |
| | Power Spectral Density Via the FFT | 7 |
| | Auto-Correlation, Cross-Correlation and Convolution | 11 |
| | Miscellaneous Analysis | 17 |
| III. | CAPABILITIES AND LIMITATIONS OF MINI-COMPUTERS | 19 |
| IV. | CONCEPTUAL DESIGN OF THE SYSTEM | 25 |
| | Overlay Library | 27 |
| | Interactive Input Handler | 30 |
| | System Execution List | 33 |
| | Signal Data Manipulation | 36 |
| | Summary of the System | 40 |
| V. | SAMPLE SYSTEM | 43 |
| | The System Supervisor | 44 |
| | Interactive Input | 49 |
| | Overlay Linkage | 50 |
| | Input Data Files | 53 |
| | Temporary Data Storage | 54 |
| | Demonstration | 55 |
| VI. | CONCLUSIONS | 62 |
| BIBLIOGRAPHY | | 63 |
| APPENDIX A | | 65 |
| APPENDIX B | | 87 |

# LIST OF TABLES


Table Page

I. Minicomputer Peripherals. . . . . . . . . . . . . . . . . 23

II. Routines Which Might be Included in a Signal Analysis Overlay Library . . . . . . . . . . . . . . 31

III. Main Parts of the System Supervisor . . . . . . . . . . 45

IV. Command Summary for Demonstration of the Signal Analysis System . . . . . . . . . . . . . . . 57

V. Organization of Header Record for Input Files . . . . 70

VI. Interactive Command Summary. . . . . . . . . . . . . . 71

VII. Input Error Messages. . . . . . . . . . . . . . . . . . . 79

## LIST OF FIGURES


| Figure | | Page |
|---|---|---|
| 1. | The Minicomputer in an Interactive, Man-Machine, Problem Solving System . . . . . . . . . . . . . . | 2 |
| 2(a). | PSD of Sine Wave. . . . . . . . . . . . . . . . . . . . . | 9 |
| 2(b). | PSD of Wide-Band Noise . . . . . . . . . . . . . . . . . | 9 |
| 3(a). | Circular Correlation Functions . . . . . . . . . . . . . . | 14 |
| 3(b). | Separation of Circular Correlation Functions . . . . . | 14 |
| 4(a). | Auto-correlation of a Sine Function. . . . . . . . . . . | 15 |
| 4(b). | Auto-correlation of High Frequency Data. . . . . . . . | 15 |
| 4(c). | Auto-correlation of Low Frequency Data. . . . . . . . | 15 |
| 5. | Overlay Method of Memory Management . . . . . . . . | 28 |
| 6. | Diagram of Input Handler's Syntax Analyzer . . . . . | 32 |
| 7. | Circular List . . . . . . . . . . . . . . . . . . . . . . | 35 |
| 8. | Table Required to Maintain the Circular List . . . . . | 35 |
| 9. | Virtual Storage System . . . . . . . . . . . . . . . . | 38 |
| 10. | Conceptual Digital Signal Analysis System . . . . . . | 42 |
| 11. | Controller Interaction . . . . . . . . . . . . . . . . . . | 46 |
| 12. | Temporary Storage System . . . . . . . . . . . . . . . | 55 |
| 13. | Display of Header Information from Input File . . . . | 59 |

Figure | Page

14. Sample Plot of Input Data Sequence . . . . . . . . . . . . 60

15. Sample Plot of the PSD Estimate. . . . . . . . . . . . . . 61

16. Sample Load Map for the DSA. . . . . . . . . . . . . . . 77

# CHAPTER I

# INTRODUCTION

While the minicomputer is somewhat limited in terms of word and memory size compared to the larger, more powerful computers, it is finding usefulness in many smaller applications. Because of its size and cost, the mini can be put to use in situations which require a dedicated computer. In contrast to the user who utilizes the common batch processing methods of the larger computers, a researcher is able to obtain a more intimate interaction between himself and the system he is studying by using the smaller, dedicated machines.

Presently, there exists a definite trend toward the implementation of minicomputers as elements within a large system. The actual use of a minicomputer requires extensive knowledge of its machine level operation to be efficiently programmed. However, as part of a system, it can serve a large number of people who have very little familiarity with computers at all. This study is concerned with the use of the mini in a system such as that represented in Figure 1. More specifically, it concerns a system which is primarily designed for the analysis of digital signal data.

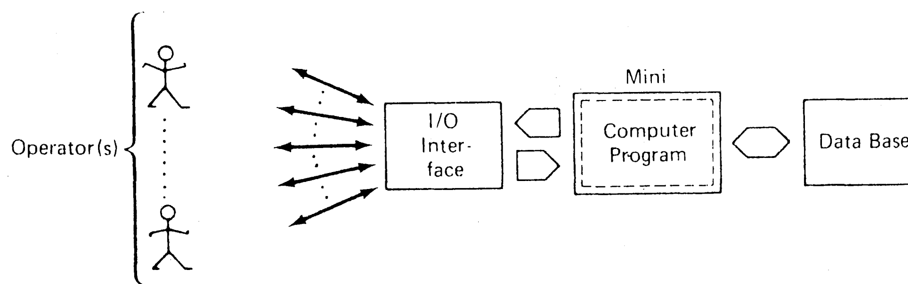


Figure 1. The Minicomputers in an Interactive, Man-Machine, Problem-Solving System [1]

Early methods of signal processing and analysis were mostly analog in nature, and special purpose analog equipment has been and is still being designed to carry out these methods. However, the advent of computers revitalized the digital signal analysis techniques. While analog methods are somewhat inflexible and expensive, the digital methods, implemented on general purpose computers, can be tailored to satisfy a multitude of analysis methods.

There exists many complex and sophisticated digital signal processing algorithms as well as special digital hardware. Several systems which incorporate these algorithms and hardware have been designed, built, and marketed. Such systems include special computers which incorporate specific hardware devices to perform signal processing, and "compilers" which translate processing input statements into sequences of machine code for execution by computers. There is, however, a lack of information and ideas which deal with the flexible implementation of signal analysis algorithms on general purpose minicomputers.

The objective of this study is to design and test the concepts of a digital signal analysis system for general purpose minicomputers. The concepts developed are general enough to be applied to most minicomputers on the market. The problems associated with small memories, slow speed, and input/output of data are considered. The system makes use of existing signal processing algorithms as well as software packages and operating systems supplied with minis. The justification for this study is twofold. First, there is a definite lack of software systems of this type available for minicomputers. Secondly, the ability to arrange and rearrange signal processing and analysis sequences without constant reprogramming of source algorithms gives a researcher more time to actually study the signal.

The study consists of five main parts. The first part, Chapter II, is an overview of digital signal analysis. The basic methods and computational steps required to compute a few of the main functions in signal analysis are outlined. The intent is to show some of the requirements necessary of this system. A brief discussion of the capabilities and limitations of minicomputers is included in Chapter III. Chapter IV details the concepts of the system in a general manner, while Chapter V applies the concepts to the Interdata Model 7/16 minicomputer. The last part, Chapter VI, presents the conclusions of the study and recommendations for further study. Two appendices are included which contain a users' manual for the OSU-MAE Digital Signal Analysis System and a listing of the main routines.

# CHAPTER II

## OVERVIEW OF SIGNAL ANALYSIS

Digital signal processing has for a long time been an effective tool in engineering and scientific studies. Its fundamentals are based on classical numerical analysis techniques developed in the 1600's. Important refinements to the techniques which provide the foundation for digital signal processing were evident in the development of sampled-data control systems in the 1940's and 1950's. The advent of high-speed electronic computers in the 1960's brought about even more refinements and applications making it a dynamic and rapidly growing field. Its effectiveness is now touching such diverse fields as biomedical engineering, acoustics, sonar, radar, seismology, speech communication, data communication, nuclear science, and many others [2].

The representation of signals by a sequence of numbers or symbols and the processing of these sequences is called digital signal processing. This processing may be designed to estimate certain parameters of a signal or modify a signal such that it is in some way more useful. For purposes of this study the phrase "digital signal analysis" is used to describe the methods employed for the

extraction of characteristic information from a signal. The phrase "digital signal processing" is used as it has been previously defined. This distinction is made only because most of the work done in this study involves signal analysis.

The fundamentals of digital signal analysis methods are well formulated and presented in many texts [2, 3, 4, 5]. Many complex and sophisticated algorithms based on these fundamentals have been developed. The age of computers has brought about flurries of literature on both analysis and processing algorithms [6]. The best known of these algorithms is the fast Fourier transform or FFT. Its development has led to the use of algorithms once considered impractical [6]. In fact, many new techniques utilizing integrated electronics are direct results of the fast Fourier transform.

Digital signal analysis is a broad area and certainly the amount of discussion which can be presented in this study cannot reveal all its many aspects. The remainder of this chapter summarizes the computational steps involved in calculating the major functions of signal analysis. The intention is to provide an insight into the requirements of the analysis system under study.

## The Fast Fourier Transform

The Fourier representation of finite-duration sequences is termed the discrete Fourier transform or DFT. Consider a sequence $x(n)$ of N equally spaced data values representing one cycle of a

periodic sequence. This sequence has finite-duration. The DFT is then represented by the following transform pair [2]:

$$X(k) = \begin{cases} \sum_{n=0}^{N-1} x(n)\ W_N^{kn} & , 0 \le k \le N-1 \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

$$x(n) = \begin{cases} \frac{1}{N}\sum_{k=0}^{N-1} X(k)\ W_N^{-kn} & , 0 \le n \le N-1 \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

$$W_N = e^{-j(2\pi/N)}$$

$X(k)$ is the Fourier transform coefficient for the $k^{th}$ harmonic. These coefficients are also periodic with period N.

The direct calculation of these two relations require computation times proportional to $N^2$. Most approaches to improving the efficiency of the computation of the DFT exploit one or both of the following special properties of $W_N^{kn}$:

1. $W_N^{k(N-n)} = (W_N^{kn})^*$

2. $W_N^{kn} = W_N^{k(n+N)} = W_N^{(k+N)n}$

The "*" denotes complex conjugation.

These two properties demonstrate the symmetry and periodicity of $W_N^{kn}$, and proper use of these properties results in computational schemes which greatly reduce the number of multiplications and additions. In 1965, J. W. Cooley and J. W. Tukey [7] published an

algorithm for the computation of the discrete Fourier transform that is applicable when N is composite number; i.e., N is the product of two or more integers. This and similar algorithms effectively reduced the computation time to an amount proportional to N log N. Collectively, the entire set of these algorithms are often loosely referred to as "the FFT" [ 8].

The FFT today is an important tool used in many digital signal analysis and processing techniques. Along with algorithms designed for general purpose computers, special hardware processors have been developed which compute transforms with such speed that real-time signal processors are state-of-the-art for many applications.

There are two excellent texts which provide a detailed development of the FFT [2,3]. Other articles can be found which describe refinements to the basic algorithm allowing transforms on large amounts of data using auxiliary memory [9]. The design of a digital signal analysis system should incorporate an efficient FFT algorithm and its capabilities.

## Power Spectral Density Via the FFT

One of the most important signal analysis techniques is that of estimating the mean square spectral density or, as it is commonly called, the power spectral density of a signal. The power spectral density, or PSD, is used primarily to establish the frequency composition of signal data. This in turn reflects some

basic characteristics of the system which generated the data. As an example, consider the analysis of vibration data from a rotating machine. By applying suitable PSD analysis techniques to this data, potential system problems might be detected. Information revealing things such as uneven bearing wear, or unbalanced components, might show up as peaks in the PSD at frequencies which are multiples of the rotation speed.

Many equivalent definitions of power spectral density can be given, but the most practical one is the following. It is a real function of frequency such that the total area under the PSD function from 0 to $\infty$ is the total mean square value of the signal. The partial area under the PSD function from $f_1$ to $f_2$ represents the mean square value in the signal between frequencies $f_1$ and $f_2$ [10].

Given a sequence of N data values, equally spaced $\Delta T$ in time, the spectral density at frequency $f_k$ is given by [11]:

$$G_k(f_k) = \frac{2\Delta T}{N} \left| X(k) \right|^2 \tag{2.3}$$

where X(k) is the DFT coefficient at the $k^{th}$ harmonic. Figure 2 shows the PSD vs. frequency for a sine wave and for wide-band random noise. As seen in Figure 2(a) the PSD of a sine wave has a single infinite component at its own frequency, whereas, for the wide-band noise shown in Figure 2(b), the spectrum is relatively smooth. The PSD exhibits peaks at the periodic components of a signal.

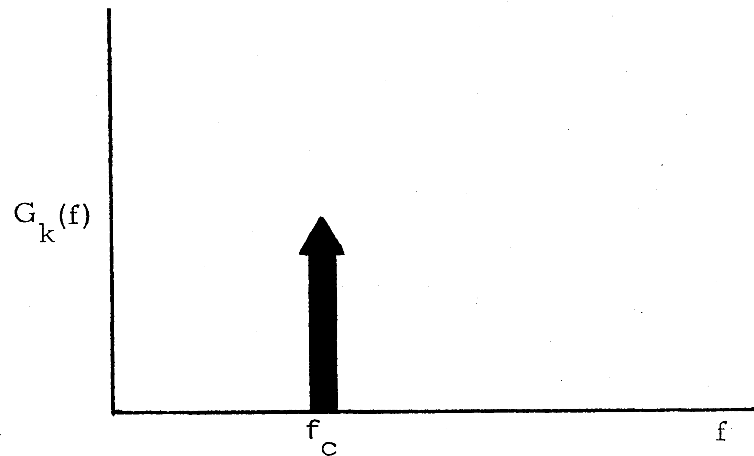


Fig. 2(a). PSD of Sine Wave

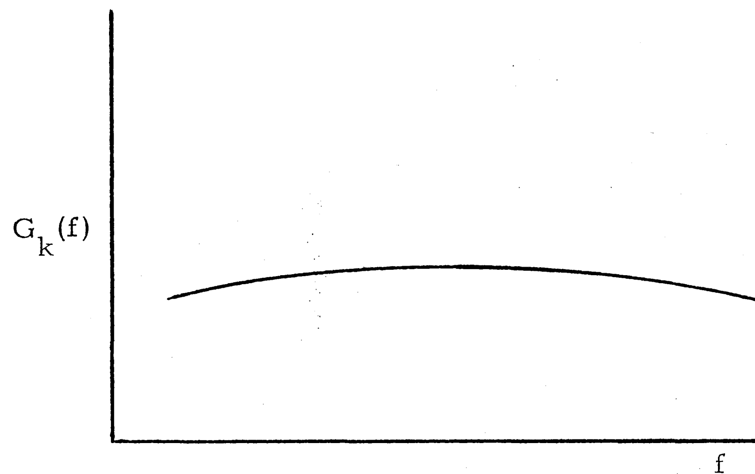


Fig. 2(b). PSD of Wide-Band Noise

The PSD can be calculated using the FFT, but there are two important problems to be considered. The first of these problems arises from the aperiodicity of the signal. Normally the section of signal being processed is regarded as a truncated version of the original signal. However, the DFT treats the section as one period of an infinitely long periodic signal. This effective signal has discontinuities at the ends which introduces considerable distortion into

the frequency domain representation. This phenomenon, sometimes called leakage, can be minimized by the application of different types of data windows to the signal. These windows are discussed in detail in references [12, 13].

The second problem is smoothing. Several papers have been written which present various spectrum smoothing techniques, but only a few are commonly employed. One of these methods is called "frequency averaging." The smoothed spectral estimate can be obtained by averaging L neighboring frequency components of the raw spectral estimate; that is, a smooth $G_k$ is given by:

$$G_k = \frac{1}{L}\left[G_k + G_{k+1} + \ldots + G_{k+L-1}\right] \tag{2.4}$$

Another method is time averaging [14]. This method is implemented in the following manner. Consider a stationary stochastic sequence divided into q separate sections, possibly overlapping. The raw spectral estimates are obtained for each section by equation 2.3. If $G_{k,q}$ represents the raw estimate at frequency $f_k$ of $q^{th}$ time section, then the final smooth spectral estimate is given by:

$$G_k = \frac{1}{q}\left[G_{k,1} + G_{k,2} + \ldots + G_{k,q}\right] \tag{2.5}$$

With the preceding information, it is now possible to summarize the computational steps involved in computing the PSD function of a signal [11].

1. Truncate the data sequence such that the FFT may be computed efficiently.
2. Taper the resulting sequence using a cosine taper, data window, or some other appropriate tapering.
3. Compute the FFT.
4. Compute the raw spectral estimate $G_k$.
5. Adjust these estimates with correction factors that arise due to tapering.
6. Average these corrected estimates with any desired averaging method.

These are general computational steps and there are several variations. However, this procedure alone should demonstrate the necessity of a computational system which makes PSD analysis convenient.

## Auto-Correlation, Cross-Correlation and Convolution

Another useful signal analysis function is auto-correlation. The auto-correlation measurement provides a tool for detecting periodic components which might exist in random data. It also provides information about the frequency range of data, i.e., is it composed of high or low frequencies. This function is obtained by delaying a signal relative to itself by some fixed time delay (called the lag), multiplying the original signal with the delayed signal, and averaging the resulting product over some desired portion of the signal length.

For a continuous signal, the auto-correlation function is mathematically defined as:

$$R_x(\tau) = \lim_{T \to \infty} \frac{1}{T} \int_0^T x(t)x(t + \tau)\, d\tau \qquad (2.6)$$

where $\tau$ is the time lag. If instead of delaying a signal relative to itself, it is delayed relative to a second signal such as y(t), the cross-correlation function results. The cross-correlation is used to establish the dependence between two different random signals and for the continuous signal is defined as:

$$R_{xy}(\tau) = \lim_{T \to \infty} \frac{1}{T} \int_0^T x(t)y(t + \tau) d\tau. \qquad (2.7)$$

The auto-correlation function of a random signal can be obtained by applying the Wiener-Khinchine Relation [2]. This relation states that the inverse Fourier transform of the PSD of a random signal is the auto-correlation function. Since the PSD can be computed with the FFT, the FFT can be applied to compute the auto-correlation. Thus the basic computational steps might be to compute the FFT of the signal, compute the raw spectrum, then compute the inverse FFT to obtain the auto-correlation. This approach may seem like a roundabout method for obtaining the correlation functions, but its computation is considerably faster than the direct calculation of the convolution integrals given in equations 2.6 and 2.7. There are, however, certain modifications to this approach which are necessary.

The above method does not yield the auto-correlation function, but a circular-correlation. The two parts of circular correlation are illustrated in Figure 3(a). This circular correlation may be avoided by adding zeros to the data before transformation with the FFT. The effect is to spread the two parts as shown in Figure 3(b). In particular, if N zeros are added, the result would be a complete separation of the two parts. In practice, the number of zeros added to the data need only be at least the number of time lags desired.

Figure 4 shows the auto-correlation functions for a sine-wave, high frequency random data, and data containing all low frequency components. The auto-correlation is periodic for the sine-wave. High frequency data has an auto-correlation which damps to zero rapidly, while the auto-correlation for low frequency data remains more flat.

In summary, the following steps are recommended to compute the auto-correlation function [11].

1. Augment the data sequence by adding N zeros to the end of it to obtain a new sequence of length 2N.
2. Compute the FFT of the 2N-point data sequence.
3. Compute the raw spectrum using equation 2.3.
4. Compute the inverse FFT and multiply by a scale factor of N/(N-r) ot obtain $R_r$ for r = 0, 1, ..., 2N-1.
5. Discard the last half of $R_r$ to obtain the results.

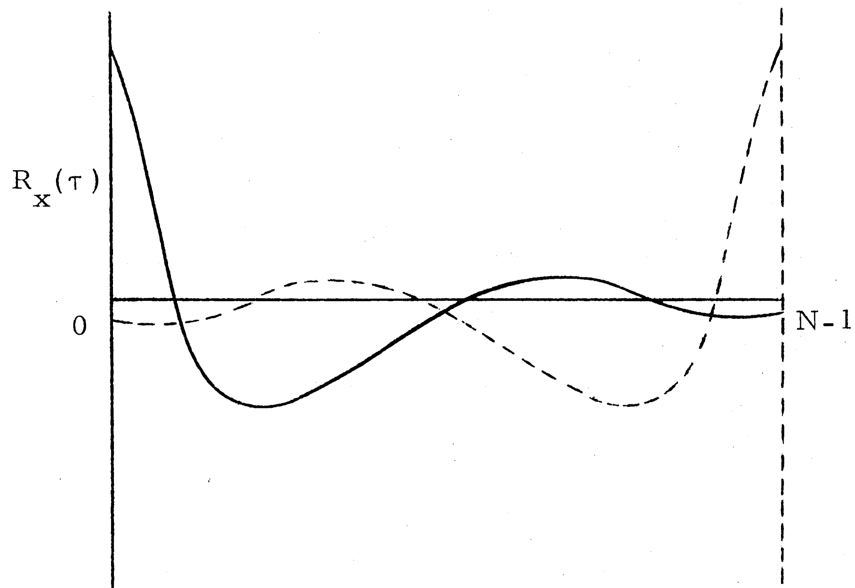


Figure 3(a). Circular Correlation Functions

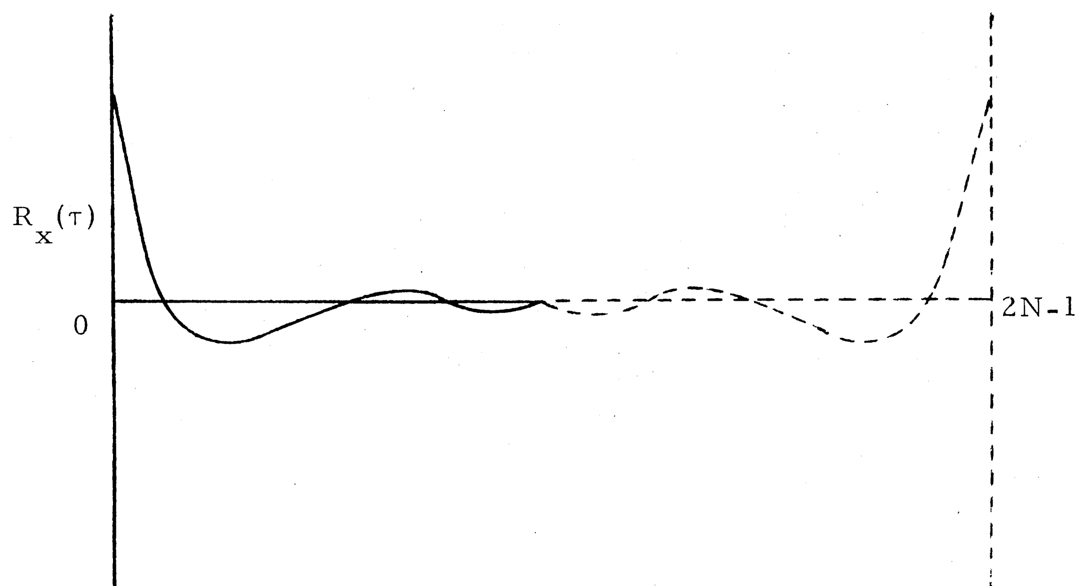


Figure 3(b). Separation of Circular Correlation Functions

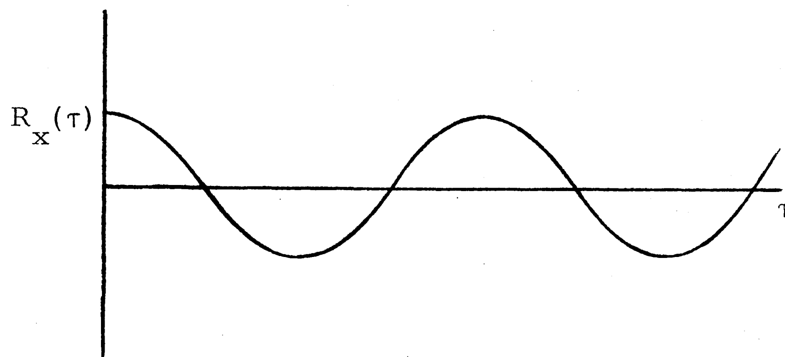


Figure 4(a). Auto-correlation of a Sine Function

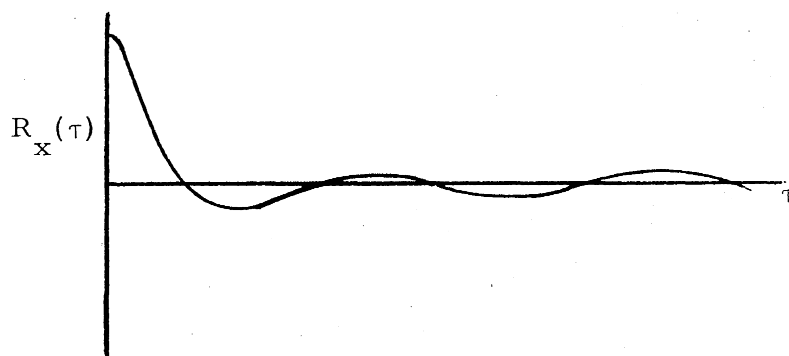


Figure 4(b). Auto-correlation of High Frequency Data

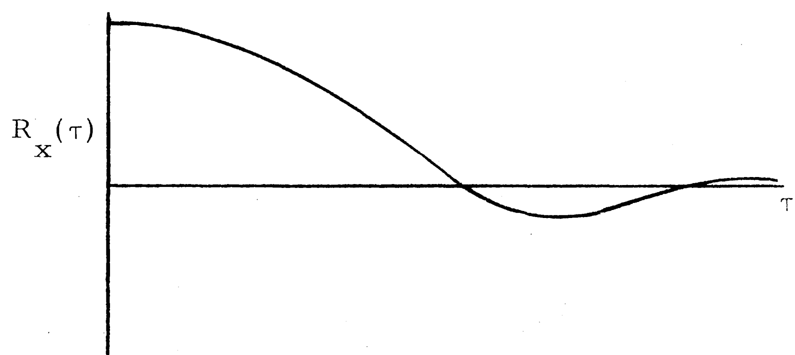


Figure 4(c). Auto-correlation of Low Frequency Data

A method similar to that outlined for the auto-correlation function can be used to calculate the cross-correlation function. Before stating the steps required for this method, a useful relation of the FFT needs to be shown. This relation is used to pair two real sequences for simultaneous calculation of the FFT.

For two real sequences x(t) and y(t) a third sequence is obtained by

$$z(t) = x(t) + jy(t) \tag{2.8}$$

The FFT is calculated and the coefficients Z(k) are obtained. X(k) and Y(k) are now given by the relations:

$$X(k) = \frac{Z(k) + Z^*(N-k)}{2}$$

$$Y(k) = \frac{Z(k) - Z^*(N-k)}{2j} \qquad k = 0, 1, 2, \ldots, N-1 \tag{2.9}$$

The "*" denotes complex conjugation.

The computation steps required for the cross-correlation function are:

1. Obtain the sequence z(t) by using the two sequences for which cross-correlation is desired.
2. Augment this new sequence with N complex zeros to obtain a sequence of length 2N.
3. Compute the 2N-point FFT to obtain Z(k).
4. Use equation 2.9 to determine X(k) and Y(k).

5. Compute the raw cross-spectral density estimate $G_{xy}(f)$ using

$$G_{xy}(f) = \frac{2\Delta t}{N} X(k)Y(k)$$

6. Compute the inverse transforms, multiply the results by the correction factor N/(N−r) to obtain $R_{xy}(\tau)$.
7. Discard the last half of the sequence as before.

Miscellaneous Analysis

There exists several other analysis functions which provide useful information about signals. These functions will not be dealt with in any detail in this section. Bendat [11] provides an excellent, detailed summary of these additional functions. A brief summary of some of these functions follows.

1. Statistics.

   Probability density functions.
   Coherence functions.
   Ensemble analysis.

2. Filtering functions.

   Recursive and non-recursive.
   Frequency sampling.
   Low pass, high pass, and band pass.

3. Data tapering functions.

4. Trend removal.

   Average slope method.
   Least Squares methods.

5. Functions for analysis of non-stationary and transient data.

6. Transfer functions and frequency response.

# CHAPTER III

## CAPABILITIES AND LIMITATIONS OF MINICOMPUTERS

A minicomputer can be described in terms of how it differs from larger, non-mini systems, such as limited physical size, 8- to 18-bit word size, limited memory size, limited processing capability, low cost, limited built-in diagnostic and error-checking features, and limited software support [1]. There are exceptions to this description since some systems which are classified as minis have word sizes of 32-bits and memory sizes approaching one million words. Systems like these are usually more powerful in all aspects, and might really be considered as midis or small computers [1].

Despite its limitations, the mini has the same basic elements found in its larger counterpart. For some basic processes, such as input/output and communication, the capabilities of the mini can easily be matched with the capabilities found on large mainframes, though on a smaller scale. Hence, minicomputer system components generally fall into these categories:

1. Processor
2. Memory

3. Input/Output

4. Software

5. Peripherals

The central processor usually consists of all the hardware controllers for addressing, arithmetic, and instruction fetching. There may be up to sixteen general-purpose hardware registers, and provisions for floating-point registers which may occupy some reserved space in memory. Fast hardware multiply and divide is usually available as an option, along with hardware floating-point arithmetic, memory protection, and privileged instruction protection. Because of the lack of hardware arithmetic functions, use of minis for large amounts of numerical calculations does not seem very attractive. Manufacturers do supply software that will simulate most of the nonexistent operations, but this capability results in a considerable slowdown in calculation speeds.

The majority of minis have small memory sizes, usually between 6K and 32K words (1K = 1024). This limitation usually arises from the range of address values that the 8- to 16-bit processors can represent. For a 16-bit processor, the maximum number of locations which can be addressed directly are $2^{16}$ or 64K. A further limitation in useful memory size stems from the fact that a certain amount of software is sometimes present in the memory to control the routine operations of the machine, i.e., input/output, arithmetic simulation, and trap and interrupt handling. This

software is known as the operating system.

For some applications stand-alone programs which run without an operating system and control all their own machine functions exist. The code for these types of programs becomes fairly complex and usually requires machine language or assembler type coding for certain parts. These programs are tailored toward one specific machine and one specific job. Operating systems relieve some of these restrictions, allow higher level languages such as Fortran to be used, and operate with a wide variety of programs.

Because of the small memory size, it is sometimes difficult to use large programs, or programs which manipulate large amounts of data in a mini. A signal analysis system is just such a program and its routines require large amounts of memory to store instructions and large arrays to hold data. It is therefore necessary to efficiently manage the memory. One of the larger machines, the IBM 370, uses a "virtual storage" technique to help get the most use of its real memory. This technique requires special hardware, known as Dynamic Address Translation (DAT) hardware, as well as special routines and tables within the operating system [15]. Virtual storage relieves the user of problems associated with memory management. Minicomputers do not usually have this type of hardware or software, so other ways of memory management must be used.

Input/output is an integral part of most minicomputers. While the larger machine has many I/O schemes, the mini is usually

limited to two or three. One method uses the central processor and a program to control the I/O. Special machine commands which use one or more registers within the processor are issued by the program to actually perform the data transfers. This method usually ties up the entire processor and the executing task must wait for completion of the I/O. Another method, called direct memory access (DMA) operates on a memory cycle-stealing basis with the processor. This method transfers data directly to and from memory, is the fastest type of I/O, and is usually used for block transfer to and from disk or other external high-speed devices.

Minicomputer software is very limited, mainly because of development costs. Manufacturers generally supply several basic software packages for their machines. These may be operating systems, assemblers, high-level compilers such as Fortran, debugging aids, and utility routines for file management and text editing. Software is the main concern of this study, and will be discussed further in later chapters.

Generally, large machine peripherals do not interface directly with minicomputers. A few exceptions do exist but for the most part, minis have peripherals designed especially for them. Table I lists a few of the more common devices generally used with minis. Peripheral equipment is the determining cost of most mini systems, and some equipment is more expensive than the processor itself. There is a great deal of latitude in interfacing minicomputers to external

TABLE I

MINICOMPUTER PERIPHERALS

| PERIPHERAL EQUIPMENT | USAGE |
|---|---|
| 1. Magnetic Storage Systems<br>A. Fixed and movable head disk drives.<br>B. Drums<br>C. Nine track tape drives.<br>D. Cassette tape drives. | Auxiliary memory and storage.<br>Program storage.<br>Data base storage. |
| 2. Paper Tape Punches and Readers<br>3. Card Readers<br>4. Line Printers | Bulk program and data input/output. |
| 5. CRT Displays<br>6. Typewriter Consoles | Interactive communication.<br>Operation consoles. |
| 7. Graphic Display Terminals<br>8. Plotting Systems | Graphic displays of data such as bar charts.<br>Hard copy plotting and drawing. |
| 9. Analog conversion equipment<br>10. Digital conversion equipment<br>11. Special I/O interface | Provides link between the mini and external systems.<br>Data acquisition systems.<br>Process control.<br>Instrumentation. |

systems. Minicomputer architecture is designed to facilitate a wide variety of special user built interface circuits for application in data acquisition, process control, instrumentation, and analysis systems.

## CHAPTER IV

## CONCEPTUAL DESIGN OF THE SYSTEM

The analysis of digital signal data with general purpose computers often requires a series of specific computational steps. As shown in Chapter II, the PSD function requires computational steps that taper the ends of the data sequence with a data window, calculate the FFT, and finally calculate and smooth the PSD estimate. It may be desired to obtain several separate PSD results each of which is smoothed by a different method or has had its original data sequence tapered by different data windows. This chapter details the main components of digital signal analysis system which offers users an efficient and flexible method of performing the computational steps described above.

A common approach to programming an analysis is to develop a program with sections of code or subroutines which each perform a certain step in the calculation. The researcher will then submit the program for execution in a batch processing stream of a large computer, or enter it through a time sharing terminal. Depending on the outcome, he may either reprogram parts of the code or change the order of sections in the code and resubmit the job. This method

has three major drawbacks. First, it is inflexible in that the program is usually designed for one type of analysis and one type of signal. The second drawback is that modification of the code is required in order to see the effects of changes in tapering, smoothing or filtering schemes. Finally the time required for the whole process, often causes the researcher to lose touch with the analysis, and possibly accept erroneous results.

A few software systems have been designed to help reduce these problems. One such system developed by Harrison [16], utilizes an alphanumeric-graphic display terminal on line to a general purpose computer. While originally designed for a special filtering problem, the systems' capabilities have been increased to include transfer function analysis, correlation, signal modification, and power spectral density estimation. Users of the system perform analysis by entering interactive commands and then see their results plotted on the screen seconds later.

A second system designed by Tenorio [17] includes several analysis and statistic functions built into a complete program package. It does not run interactively, but is submitted as a batch job to a large computer (Control Data 6600 or 7600). Users write input data which defines the type and order of analysis to be performed. The system also includes utility routines for plotting, listing, and modification of the signal data.

Both of these systems derive their usefulness from the abilities

afforded to them by the large machine and its extensive supporting software. Implementation of such systems on minicomputers has several problems. Methods designed to overcome these problems and hopefully make signal analysis more convenient for minicomputers are detailed in the remaining sections.

## Overlay Library

The small memory size of a minicomputer creates one of the biggest problems in designing a digital signal analysis system. Primarily, routines which offer more efficient computation algorithms and decrease the execution time do so at the expense of memory. This trade off can be considered desirable if the machine is not equipped with high-speed arithmetic hardware, if there is an ample supply of memory and auxiliary storage such as disk, and if the user desires rapid processing. However, even the most compact code of a signal analysis system which includes FFT, PSD, correlation, filter, plotting, and interactive command routines would not fit into the memory of a mini and operate efficiently.

There are, fortunately, techniques available to aid in the implementation of large software systems. The technique utilized in this study makes use of a very important feature of the loader programs of most minis. This feature is known as overlaying. Overlaying allows the user to break his program into smaller subroutines, then load each subroutine separately into a designated region of

memory as it is needed. Each new subroutine loaded is overlaid in memory over the previous subroutine. This means only one overlay may occupy a region of memory at one time. An overlay system is illustrated in Figure 5. Note that a small section of code remains in memory at all times to supervise the overlaying. This section of code is commonly called the root segment.

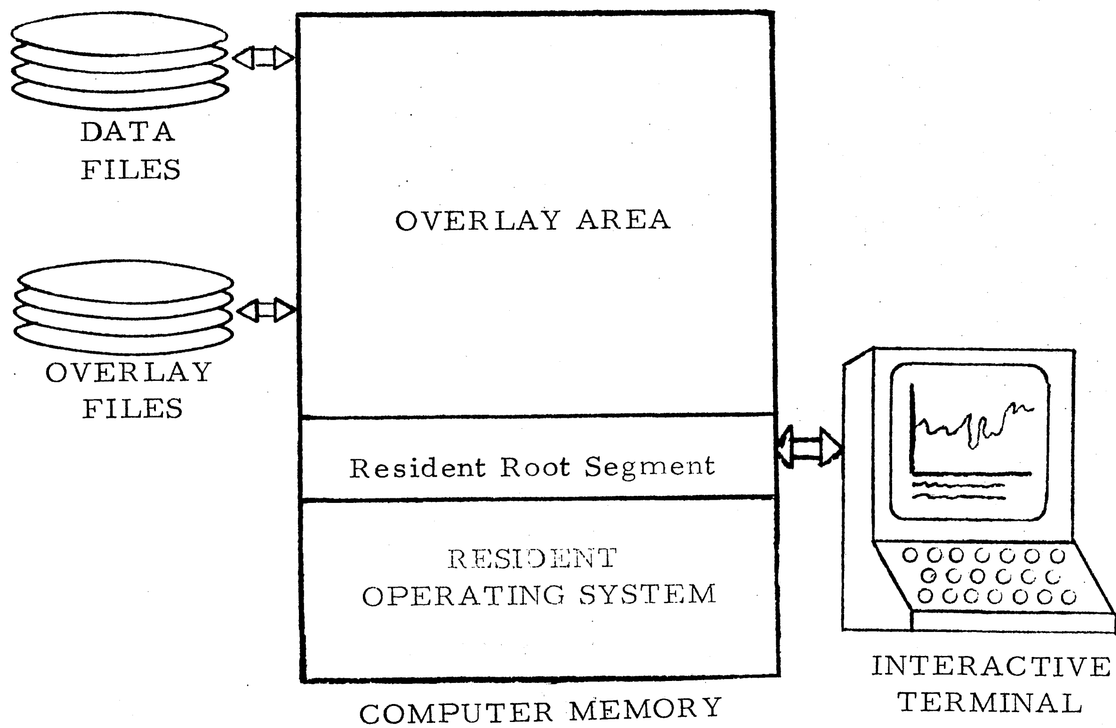


Figure 5. Overlay Method of Memory Management

Another method that might be used is to write several complete programs. Each program would then be loaded and executed as it is needed to perform a series of calculations. Each program could

read data from a common data file then list or plot its results. This seems like an easy solution, but it has several drawbacks. First, the user would need a more extensive knowledge of the computer's operation to load and execute these programs. Secondly, more external storage would be needed to store the programs since each one would need to have stored with it all the supporting routines which plot, list or handle interactive input.

Overlaying offers some advantages over the method discussed above. The loading and execution of routines is controlled by the system and except for loading time, its operation is invisible to the user. An overlay library also requires less external storage, all that needs to be stored is the routine itself. Any supporting utility routines would be part of the root segment, or overlays themselves.

Digital signal analysis is usually a step-by-step computation procedure. By properly fragmenting the system, a library of routines, each performing a specific operation on the data, can be built. These routines can be overlayed and executed in a sequence which corresponds to the conventional step-by-step methods. For example, consider the calculation of PSD function. One routine in the library tapers and truncates the data sequence. Another performs the FFT calculation and generates a file of real and imaginary sequences. The third routine calculates and smoothes the PSD estimate. The last routine might plot the results.

Table II shows what might be included in a typical digital signal

TABLE II

ROUTINES WHICH MIGHT BE INCLUDED IN A SIGNAL ANALYSIS OVERLAY LIBRARY

| Routine | Function |
|---|---|
| FFT | Routine for calculating the fast Fourier transform of a data sequence with data held in memory. |
| FFTEXT | Routine for calculating the fast Fourier transform of a large number of data points using auxiliary storage. |
| TAPER 1<br>TAPER 2<br>TAPER 3 | Data Tapering routines based on various windows. |
| RAWPSD | Routine for calculating the raw PSD function. |
| SMPSD | Routine for estimating the smoothed PSD function. |
| AUTOCR | Routine for calculating auto-correlation function. |
| CROSS | Routine for calculating the cross-correlation function. |
| PLOT | Plots a data sequence. |
| STATIS | Calculates various statistics for a data sequence. |
| LIST | Lists a selected data sequence. |
| FILTER | Aids in the design of digital filters. |

analysis overlay library. There are several functions which taper data, a smoothing algorithm, correlation algorithms, filtering routines, statistic routines, and utility routines to generate plots and listings. This offers a great deal of flexibility to the user, allowing him to experiment with various routines and sequences and see the effects without concern for actual programming.

## Interactive Input Handler

An interactive input handler is needed to supply the interface between the user and the mini. Its main function is to prompt the user for input, accept the input, interpret it, then coordinate some action based on the input. The input handler allows the use of an input language which is not as restricted as normal input to programs and supplies error messages for erroneous input immediately.

The input handler is in a sense a syntax analyzer. When prompted, the user inputs a command. The handler then searches a table containing a list of key items for commands. After a match is found for the command, it is directed to a specific section of code which decodes the statement further and checks for errors. If no errors are found, the action designated by the command is executed. The diagram in Figure 6 helps to demonstrate the flow of this process.

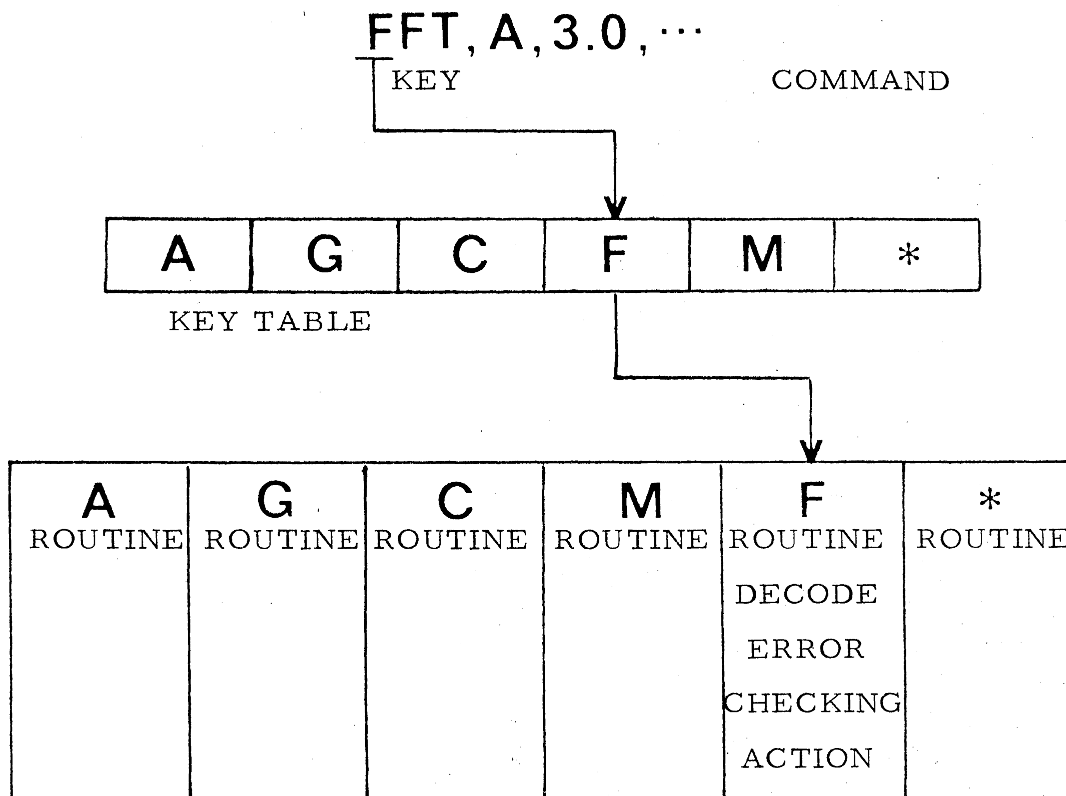


Figure 6. Diagram of Input Handler's Syntax Analyzer

Depending on the number of commands it is capable of analyzing, the size of the input handler can become fairly large. It may therefore become necessary to overlay the input handler instead of including it in the system's root segment. When overlaying the input handler, the not-so-obvious problem of reentrancy must be dealt with. A reentrant routine is one that does not store temporary results within its own string of code. This allows the routine to be entered at any time from any routine.

The input handler does not need to be made fully reentrant, but provisions for storing intermediate flags and pointers outside the

routine's bounds must be considered. This is necessary because the input handler can be overlayed at any time by another routine from the library. When the input handler is reloaded back into memory, it will need the temporary pointers to be able to determine the present status of the system.

A method generally used in Fortran programming to achieve partial reentrancy involves the use of common blocks. Common blocks are generally set up at a single place in memory either within the root segment or the overlay itself when a program is initially loaded. Blocks in the root segment remain unaltered by any overlay loading operations and can only be modified by routines which make specific requests to the common block. The common block also provides a convenient way for data to be passed from the input handler to the newly overlayed routine.

## System Execution List

Overlaying routines require time to search the library for a routine and time to actually load the routine. If a routine was loaded from the library and executed, then the input handler was again overlaid immediately afterwards, a large amount of time would be wasted in moving the input handler into memory. A simple and effective way to help reduce this time would be to have the input handler stack the routines to be executed in an execution list. This way several routines can be executed before a return to the input handler is necessary.

Such a list is illustrated in Figure 7. This is a circular list which allows information to be added to the top or bottom. Information may also be removed from either end. A small table of pointers is usually required to maintain such a list. An example of such a table is shown in Figure 8. The particular table shown in the figure is for a byte oriented minicomputer. Each pointer is contained in one byte of memory. Some machines have special instructions which allow automatic manipulation of the list. Execution of one of these machine instructions enters or removes data from the bottom or the top of the list and automatically updates the pointer table.

By utilizing such lists as those in the analysis system, the user can essentially build an interactive program. Each routine name which is input to the system is placed in the list along with arguments to be passed to it. A special command to the input handler would then cause a branch to the root segment of the system. The root segment would then fetch and execute each routine sequentially from the list. Once the list is emptied, the root segment would then reload the input handler.

Since the list is made part of the root segment, another advantage is gained. Routines loaded from the library can themselves add routines to the list for execution. Thus, a whole procedure can be initiated with a single command.

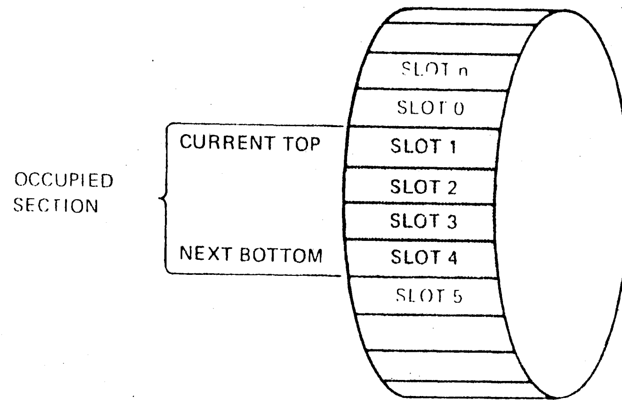


Figure 7. Circular List [18]

| 0 — 7 | 8 — 15 |
| --- | --- |
| NUMBER OF SLOTS | NUMBER USED |
| CURRENT TOP | NEXT BOTTOM |
| SLOT 0 | |
| SLOT 1 | |
| ≈ | ≈ |
| SLOT N | |

Figure 8. Table Required to Maintain
the Circular List [18]

## Signal Data Manipulation

Digitization of signals often results in large amounts of digital numbers. The number of data points resulting from digitization is dependent on the highest frequency of the signal and its duration. The sampling theorem states that the sampling rate of an analog signal must be at least twice the highest frequency contained in the signal to prevent aliasing effects [11]. Consider a signal with high frequency components in the range of 10,000 Hz. Sampling at twice this rate for one second would result in 20,000 data values. If the high frequency components are of primary interest, then the sampling rate would have to be increased still further to improve the resolution of the analysis. More information on sampling can be found in texts cited in Chapter II.

Besides the input data sequences, intermediate sequences also become a source for large amounts of data. The FFT can either replace the input sequence with the transformed data or generate a separate real-imaginary sequence. Replacing of the input sequence is sometimes undesirable since it may be required later by some other analysis.

It is quite difficult to use a minicomputer to handle and analyze extremely large amounts of data. But moderate amounts of data can be manipulated quite easily with the aid of auxiliary storage. Methods which utilize auxiliary storage are fairly common and are used on

larger systems as well as minis.

A method first considered was to simulate a virtual storage system, utilizing a disk for memory page storage. A specific section of memory is allocated to the virtual storage executive software. This includes space for memory pages and space for pointer tables. A virtual system is depicted in Figure 9. Data is input into the virtual memory by calls to a special routine and retrieved by calling another routine. This is a word-by-word exchange requiring a routine call to fetch or store each single word.

Analysis routines used with this system would require extensive modification. Every statement that used a specific data point from memory would require a call to the virtual executive routine. For instance, the Fortran assign statement

```
DATA(I) = A*B+2.0
```

would be changed to

```
CALL  STOR (DATA,I,A*B+2.0) ,
```

and

```
A = DATA(I)
```

would possibly become

```
A = FETCH (DATA,I) .
```

The storage executive uses the variable DATA to indicate a specific array, and the integer variable I to determine which word of the array is to be used. The executive then searches its page tables to determine if the data point is in core. If it is not, a page in the paging area

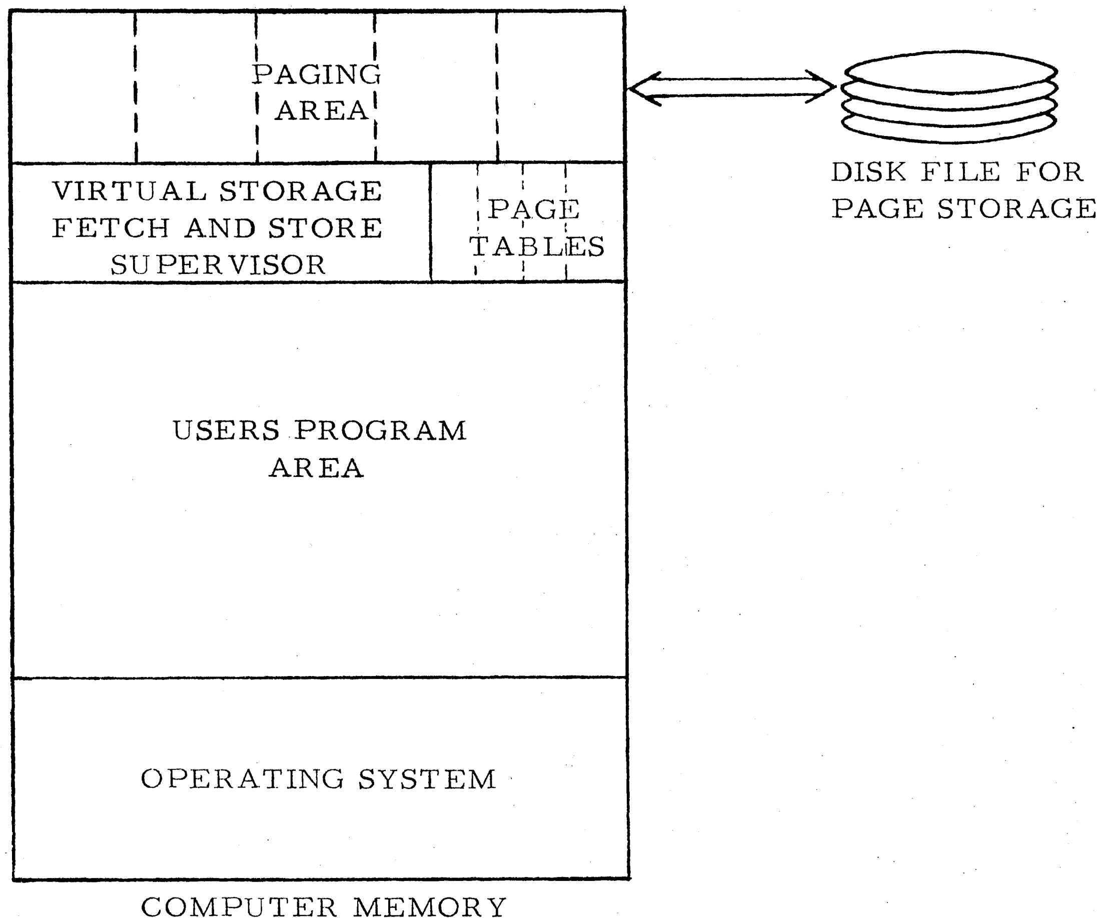


Figure 9. Virtual Storage System

is replaced with the page from the disk containing the data point.

I/O time required for paging becomes excessive, especially when existing FFT algorithms are executing. The binary bit reversal used in the more efficient FFT algorithms [3] requires data in a non-sequential order. Depending on the page size, each access to the memory could require a paging operation, resulting in greatly increased calculation times. Sequential data accesses are less time consuming but the need for source code modification still makes this virtual storage method less attractive.

A preferred method, because of the nature of digitized signal data, is to move data in blocks between auxiliary storage and user defined buffers. In this way any size block of data can be moved by the executing routine. As an example, consider an FFT routine loaded into memory with enough room remaining to hold 4,000 data points. Before FFT calculations begin, the routine calls a utility routine in the root segment which moves 4,000 data points from auxiliary storage into the buffer. The FFT executes and the transformed values are moved back out to auxiliary storage.

Temporary storage of data sequences is accomplished using one large disk file. A small system of pointers is maintained to indicate where certain sequences begin and end in the file. All accesses to temporary data is made through the utility routines. Additional information about the sequence is held in a header record at the beginning or end of each sequence. The header contains information indicating the type of data, i.e., real, complex, or integer, the title of the data, the digitization interval used in sampling, the total number of data values, and various flags.

Header records are common ways of identifying information contained in a file. By making the headers conform to certain preset standards defined by a particular system, data from a wide range of applications can be analyzed. Headers also make identification more positive. They contain all the information needed to perform the analysis efficiently.

## Summary of the System

The important concepts of the digital signal analysis system can be summarized as follows.

1. The system utilizes an overlay library containing named signal analysis routines.
2. Interactive communication between the user and machine is achieved by the use of interactive terminals and an interactive input handler routine.
3. The system contains a root segment of code which remains resident in memory. The root segment contains the system controller, the execution list, and utility routines commonly used by all routines.
4. The system uses a circular execution list, maintained by the system controller, which allows routines to be stacked for sequential loading and execution. Routine names can be added to the list by routines other than the input handler allowing a routine from the library to automatically call another routine.
5. The system manipulates large data sequences using auxiliary disk storage. Headers are placed at the beginning of data files for identifying the information.
6. The system requires minimal alteration of existing signal analysis algorithms and uses existing minicomputer software.

A diagramatic representation of the entire conceptual system is shown in Figure 10. The common storage block is shown at the top of memory for clarity only and on some minicomputers it may be actually located in the root segment or within the overlay area. The buffer area for data transfers is shown with a movable partition since each overlay defines its own buffer sizes.

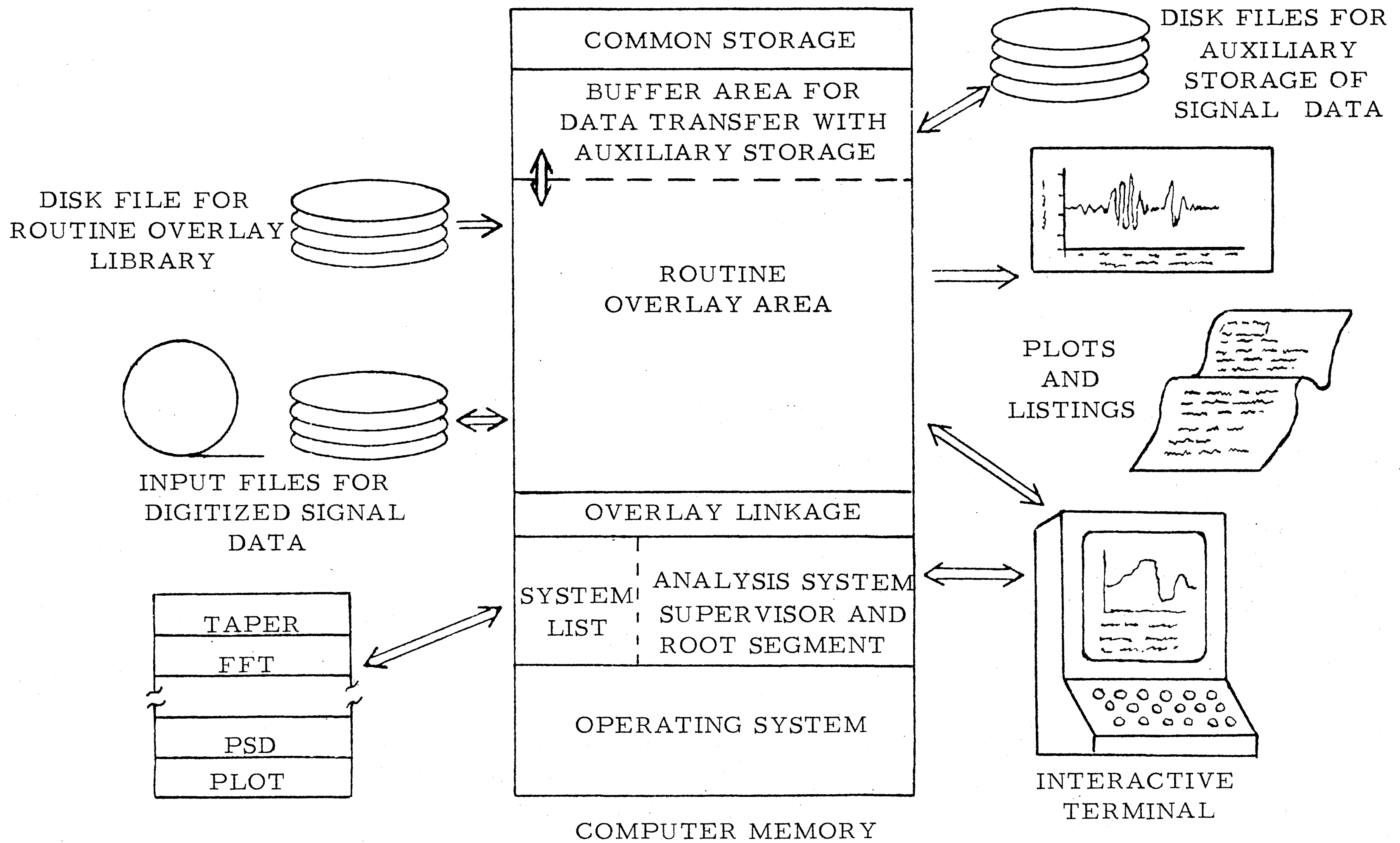


Figure 10. Conceptual Digital Signal Analysis System

# CHAPTER V

## SAMPLE SYSTEM

A system based on the concepts presented in Chapter IV has been developed as part of this study. It was developed on an Interdata Model 7/16 Basic minicomputer with 64k bytes of memory. The disk system was comprised of a 10 megabyte Control Data Model 9427 Hawk disk drive and a Zebec Model XDF-50 disk controller. Results were plotted on a Calcomp Model 565 drum plotter and listings were printed on a Centronics 165 character per second dot matrix line printer. A Teletype typewriter terminal was used to supply the interactive commands.

The analysis system was tested with the Interdata DOS operating system. The system should run under other operating systems such as the OS-16/MT2 multi-tasking system. Unavailability of other operating systems prevented further testing. It is felt that a few minor changes will be necessary to make the system execute properly with other operating systems.

The majority of the routines in the system are written Fortran. A few machine dependent routines are written in assembler and Fortran V (a special language allowing assembler and Fortran code

to be intermixed). All routines that perform signal analysis are written in Fortran and are generally existing subroutines.

Several fast Fourier transform routines based on algorithms from references [6, 19] were tested on the minicomputer. The lack of hardware multiply and divide functions resulted in slow execution of all the routines. Algorithms written by Norman Brenner [19] executed most efficiently in terms of speed and utility and were therefore selected for use in the analysis system.

The following sections describe the system and its implementation on the minicomputer. An application problem is included to illustrate its utility. Appendix A contains a brief users' manual for the analysis system and Appendix B contains the listing of the major routines required by the system.

## The System Controller

The main parts of the system controller are listed in Table III with their interaction illustrated in Figure 11. The sections listed in the table comprise the root segment of the entire program. The external data files shown in the figure comprise the system's data base.

The main program is the system coordinator. It controls the overlaying of all routines, passes control to the overlaid routines, and regains control when they finish execution. The main program also initializes the system at start-up and loads the interactive input handler when it is needed.

TABLE III

MAIN PARTS OF THE SYSTEM CONTROLLER

| ROUTINES AND COMMONS | FUNCTION |
|---|---|
| 1. MAIN PROGRAM | Initializes the system and controls the fetching of routines from the overlay library. |
| 2. EXECUTION LIST | Contains the names and arguments for routines to be loaded and executed. |
| 3. SYSTEM | Adds routines to the execution list and stores the arguments to be passed to the routine when it is loaded. |
| 4. IFETCH | Searches an overlay library for a named routine then loads the overlay into memory. |
| 5. PUT | Transfers a buffer of data to temporary storage. |
| 6. GET | Loads a defined buffer with a block of data from temporary storage. |
| 7. COMMON SIGNAL | System common block containing pointers, flags, work space, and the argument buffer. |

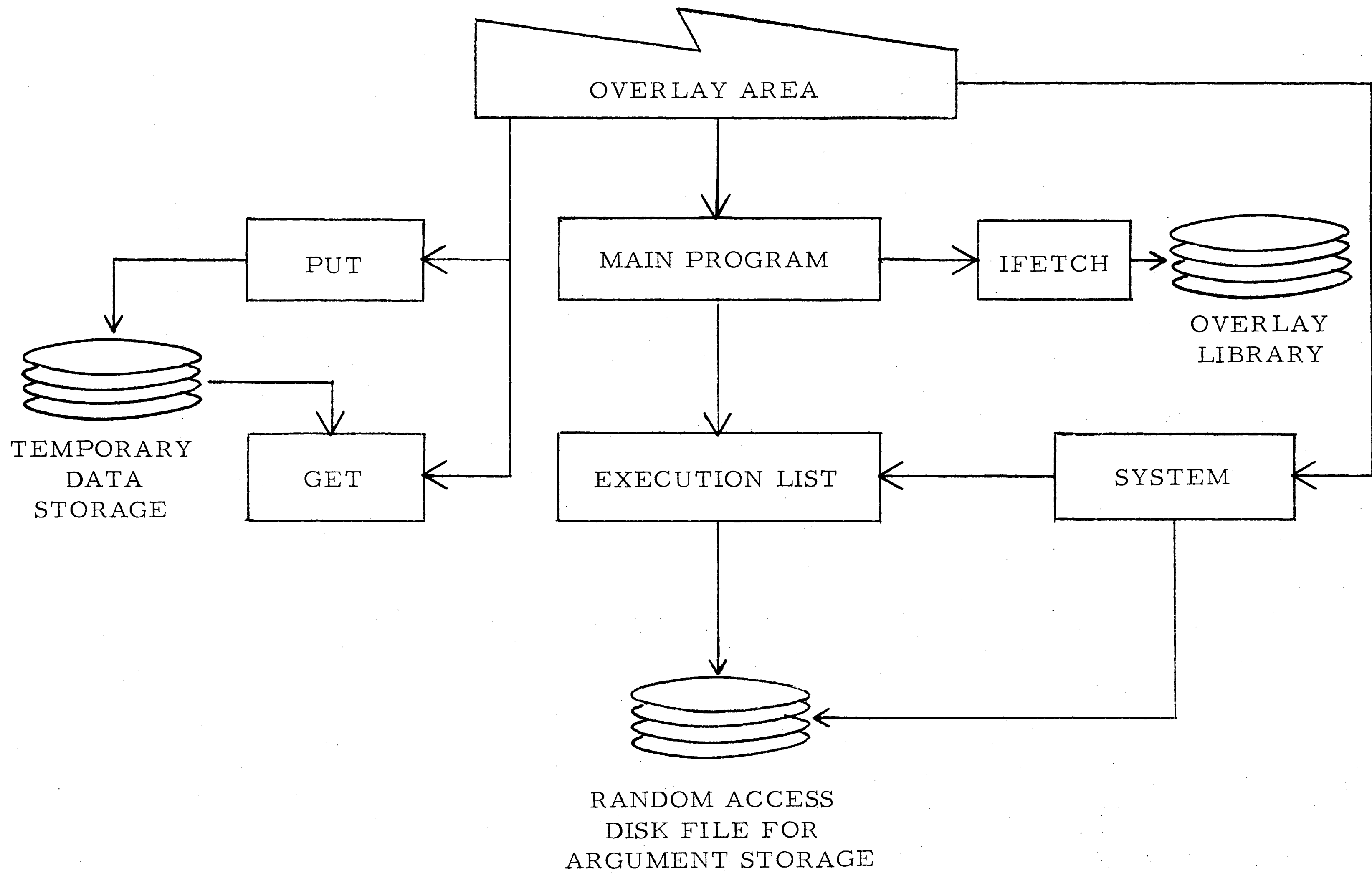


Figure 11. Controller Interaction

The execution list consists of three parts. The first part is the list of actual routine names. Six 8-bit bytes of storage are used to hold each six character routine name. The second part is the actual circular list as described in the Interdata 16-bit Reference Manual [17]. The number of arguments to be passed to the routine are stored in this list. The items of this list can only be two byte words, therefore it is not possible to store the names in the list. The third part is a disk file which contains the actual arguments to be passed to the routines. This disk file is a random access, direct physical file. Each record of this file is capable of holding 256 bytes of argument information. All three lists can be manipulated using the pointer table of the second list.

The system controller makes extensive use of a special disk access method available in the DOS operating system. This method is known as direct physical access. Disk files are divided into sectors, tracks and cylinders. There are twenty-four 256-byte sectors per track and two tracks per cylinder. Disk space is allocated in cylinders. Direct physical access permits transfers of data directly to or from a specified buffer and the disk. By specifying a random address, data can be transferred between memory and any sector on the disk file. This method of data transfer is the fastest available on the mini but its use is not a requirement.

The loading of overlays for a library required special consideration. First the software available with the Interdata mini and

supported by the DOS operating system is not capable of loading named overlays. Routines that are to be overlaid have to be stored in separate files or in one file, in the order they were going to be called. A special Fortran V routine, IFETCH, was developed which made the fetching of named overlays possible.

The main program is the only program that calls IFETCH. The form of the call is

CALL IFETCH (NAME, LU, ISTAT)

where: NAME is the routine name, padded right to six characters with blanks,

LU is the logical unit assigned to the overlay library file,

ISTAT is a status code returned by the subroutine.

0 = no error, 1 = error.

The main program fetches a name from the top of the execution list and then moves the corresponding arguments from the disk to the argument buffer in common SIGNAL. A call to IFETCH is made and the routine is found and loaded into the overlay area. The main program then executes a call to the overlaid routine.

Subroutine SYSTEM is an assembler routine which adds the names of overlays to the execution list. It also stores the arguments for the routine on the disk file. SYSTEM can add routines to either the bottom or the top of list. The form of a call to SYSTEM is:

CALL SYSTEM(NAME, ABUFF, ±NARG, IFLG)

where: NAME is the overlay name to be added to the list.

ABUFF is the address of an array containing the arguments.

NARG is the number of arguments in ABUFF.

If NARG $< 0$, the routine name is added to the bottom of the list.

If NARG $= 0$, no action is taken.

If NARG $> 0$, the routine name is added to the bottom of the list.

IFLG is a return error flag (see Appendix A).

SYSTEM uses the ATL (add to top of the list) and ABL (add to the bottom of the list) machine instructions of the Interdata to manipulate the circular list. Use of these instructions automatically updates the pointer table associated with the list. The main program always executes routines from the top of the list and uses the RTL (remove from the top of the list) machine instruction to remove routines from the list after they are loaded.

## Interactive Input

Interactive input is under direct control of the input handler, DSAIN. This routine exists as an overlay and is loaded automatically by the main program. It is written entirely in Fortran and uses common SIGNAL to achieve partial reentrancy.

DSAIN accepts two types of commands from the user. One type of command causes an immediate action in the system. The second type causes no action other than to place a routine name in the execution list. The immediate action commands perform the following tasks:

1. Defines signal data input files,
2. Allocates temporary disk storage for data sequences,
3. Moves data from input files to temporary storage,
4. Starts the execution of routines in the list.

Free format input consists of a command word beginning with a key letter and subsequent arguments separated by commas. The command is then decoded using the scheme shown in Figure 6 of Chapter IV. If the command does not contain a key letter as the first letter, then it is treated as a routine name and is placed in the execution list.

## Overlay Linkage

Creation of the overlay library was accomplished with the aid of the Interdata loader program. The loader has a built-in overlay function which allows overlays to be created on an external file. All external subroutine references are resolved at the time the overlay is created. The loader also has the facility to name the overlay, thus making the whole overlay library idea feasible.

Routines which are to be placed in the overlay library require a small section of Fortran code to set up the proper linkage between itself and the root segment. This code precedes all other code for the routine. Besides providing proper linkage, it checks the arguments passed for errors.

A closer examination of this code is necessary at this point. Suppose the following subroutine is to be added to the overlay library:

```
SUBROUTINE FFT (ID,NUM,SIGN,ARG)
```

where ID contains character data. SIGN and ARG are real arguments, and NUM is an integer argument.

It is desired that the following command to the input handler be used to activate this routine:

```
FFT,ID,NUM,SIGN,ARG
```

The Fortran entry code for this routine would be:

```
      SUBROUTINE DSAMOD
      COMMON/SIGNAL/...,...,...,ABUFF(64),NARG
          .
          .
          .
      EQUIVALENCE (ABUFF(1),ID),(ABUFF(2),NUM),
     1             (ABUFF(3),SIGN),(ABUFF(4),ARG)
          .
          .
          .
```

```
NUM=IFIX(ABUFF(2))
     .
     .
     .
Argument error checking
     .
     .
     .
Code for routine FFT or
CALL FFT (ID,NUM,SIGN,ARG)
RETURN
END
```

The subroutine name DSAMOD is used to aid in obtaining proper linkage when the overlay is created by the loader. The loader resolves external references by subroutine name. The main program of the system always executes a call to subroutine DSAMOD when it passes control to an overlay. The name FFT, however, would be used as the routine label when the overlay is created with the loader program.

Arguments are passed to the overlay via common SIGNAL, therefore it must be included in SUBROUTINE DSAMOD. The EQUIVALENCE statement aids in the separation of arguments. The input handler decodes all numeric arguments as real numbers and all character data remains as left justified characters. The statement

```
NUM=IFIX(ABUFF(2))
```

is used to convert the real argument in ABUFF(2) to an integer argument. By using these programming conventions any subroutine

can be added to the overlay library with its own argument definitions. No modification of the system controller is necessary.

## Input Data Files

Signal data sequences must be prestored in tape or disk files before they can be input to the signal analysis system. Since the system does not do real-time analysis, this restriction is necessary. The input files must also conform to a certain format. Disk files with a direct physical attribute are recommended since they can be read rapidly, but provisions have been made for non-disk files.

An input file must contain one 256 byte header record followed by as many 256 byte data records as desired. The file header contains the following information as detailed in Appendix A:

1. Discretization interval in milliseconds or Hertz.
2. Discretization indicator; 0 = time, 1 = frequency.
3. Starting time of data.
4. Number of records with the file.
5. Word type indicator;
   0 = REAL*4 (64 words per record)
   1 = COMPLEX (32 words per record)
   2 = INTEGER*2 (128 words per record)
6. Gage factor.
7. Title information.

A maximum of ten input files can be handled by the analysis system at one time. The capability to handle multiple input

sequence is desirable for statistical ensemble analysis and correlations.

## Temporary Data Storage

Since most signal analysis functions operate on one sequence and generate another, temporary storage is needed to hold the intermediate results. It may also be desirable to hold the results of one function so that it may be used repeatedly as input to other functions. The Fourier transform coefficients are an example of one sequence which might need to be held. This means that the FFT of an input sequence need only be computed once.

Temporary storage is maintained on a single disk file. Individual data sequences are stored in subfiles with a table of pointers marking their position. This arrangement is illustrated in Figure 12. A header record is also stored with each temporary file. The format of this header is quite arbitrary, but for the most part, it contains the same information as is included in the input headers described previously.

The utility subroutines PUT and GET are used to access data in this file. Subroutine PUT transfers data from a designated buffer to a designated subfile. Subroutine GET transfers in the opposite direction. The caller supplies the subfile ID, the relative starting record number, and the number of records to be transferred. The caller must also supply the start address of the buffer to or from which

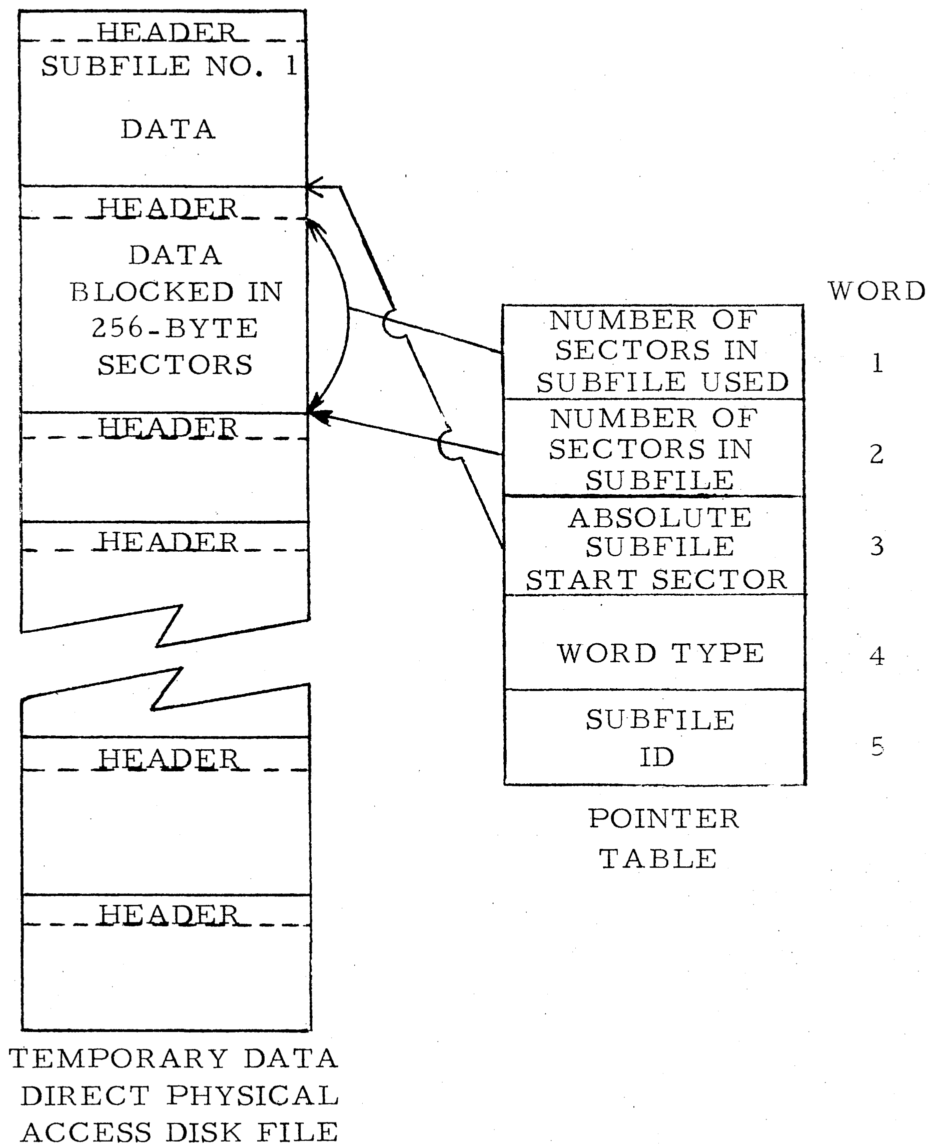


Figure 12. Temporary Storage System

data is to be transferred.

## Demonstration

The utility of the digital signal analysis system cannot be fully appreciated without a demonstration. Therefore, a simple analysis

is included here to help show how the system works. All the figures that follow are actual results from the system.

Canine heart sounds were recorded on an analog tape recorder. This signal was then digitized with a Biomation Waveform Recorder at a sampling rate of 0.2 milliseconds. One entire heartbeat sound was represented in 2048 data points. With the aid of a special program, the digitized signal was transferred from the waveform recorder to the minicomputer. The data was then stored in a disk file which conformed to the input data file specifications of the signal analysis system. This file was named HEART.

The signal analysis system was compiled and stored as a binary load module in a file named DSA. An overlay library was created in a file named DSALIB. This library contained the routines PLOT, TAPER, FFT, and SMPSD. Table IV lists the commands that were then input on the Teletype with their resulting action.

The question marks in Table IV are prompts from the system. The commands beginning with $$ direct the system to perform an immediate action such as defining the input file, requesting a temporary storage file and moving data from the input file to the temporary file. The commands that do not begin with a special character are routine names from the overlay library DSALIB. These names are placed in the execution list. The GO command starts the execution of the routines in the list, and END stops the DSA system.

TABLE IV

COMMAND SUMMARY FOR DEMONSTRATION OF THE SIGNAL ANALYSIS SYSTEM

| Interactive Commands and Prompts | Resulting Action and Descriptions |
|---|---|
| AC HEART,1 | File HEART becomes logical unit 1. |
| RU DSA | The Digital Signal Analysis system executes. |
| OSU-MAE DIGITAL ANALYSIS SYSTEM | Introductory message from the analysis system. |
| ENTER LIBRARY NAME | Request for file name which contains the overlay library. |
| DSALIB | Overlay library file name. |
| $$INPUT,1<br>? | Informs the analysis system that logical unit 1 can be used for input. |
| $$REQUEST,F1,33<br>? | Requests for a temporary storage file with ID = F1 and length = 33 records. |
| $$ALLOCATE<br>? | Allocates the disk space for temporary files. |
| $$MOVE,1,F1<br>? | Copies the date from the input file on logical unit 1 to the temporary storage file F1. |

TABLE IV (Continued)

| Interactive Commands and Prompts | Resulting Action and Descriptions |
|---|---|
| $$DISPLAY<br>? | Lists header information from the input file (see Figure 13). |
| PLOT,F1<br>? | Routine name PLOT and argument F1 is placed in the execution list. The PLOT routine will plot any data sequence (see Figure 14 and Figure 15). |
| TAPER,F1,F1<br>? | Routine name TAPER and arguments F1 and F1 are placed in the execution list. TAPER will use a data window to tape the data sequence in F1 and then will place the results back in F1. |
| FFT,F1,F1<br>? | Routine name FFT and arguments F1 and F1 are placed in the execution list. FFT will transform the data in F1 then place the results back in F1. |
| SMPSD,F1,F1<br>? | Routine name SMPSD and arguments F1 and F1 are placed in the execution list. SMPSD will calculate the smooth power spectral density estimate of the transformed data in F1 then place the results back in F1. |
| PLOT,F1<br>? | Same action as the previous PLOT command. |
| $$GO<br>? | Instructs the analysis system to begin executing the routines in the execution list. |
| $$END | Stops the analysis system. |

The $$DISPLAY command causes the system to display information from the header of a file on the line printer. Figure 13 shows an example of this display. Figure 14 and Figure 15 are examples of the plots produced by the PLOT routine on the Calcomp plotter.

```
FILE TITLE: NORMAL CANINE HEART SOUNDS    BAND NO.  1
DISCRETIZATION INTERVAL:     0.200000 MSEC
STARTING AT   0.000000 SECONDS
REAL*4 FILE CONTAINING      32 SECTORS
@ 64 WORDS PER SECTOR
DEFINED SECTORS:       1 TO     32 FOR A TOTAL OF       32
```

Figure 13. Display of Header Information from Input File

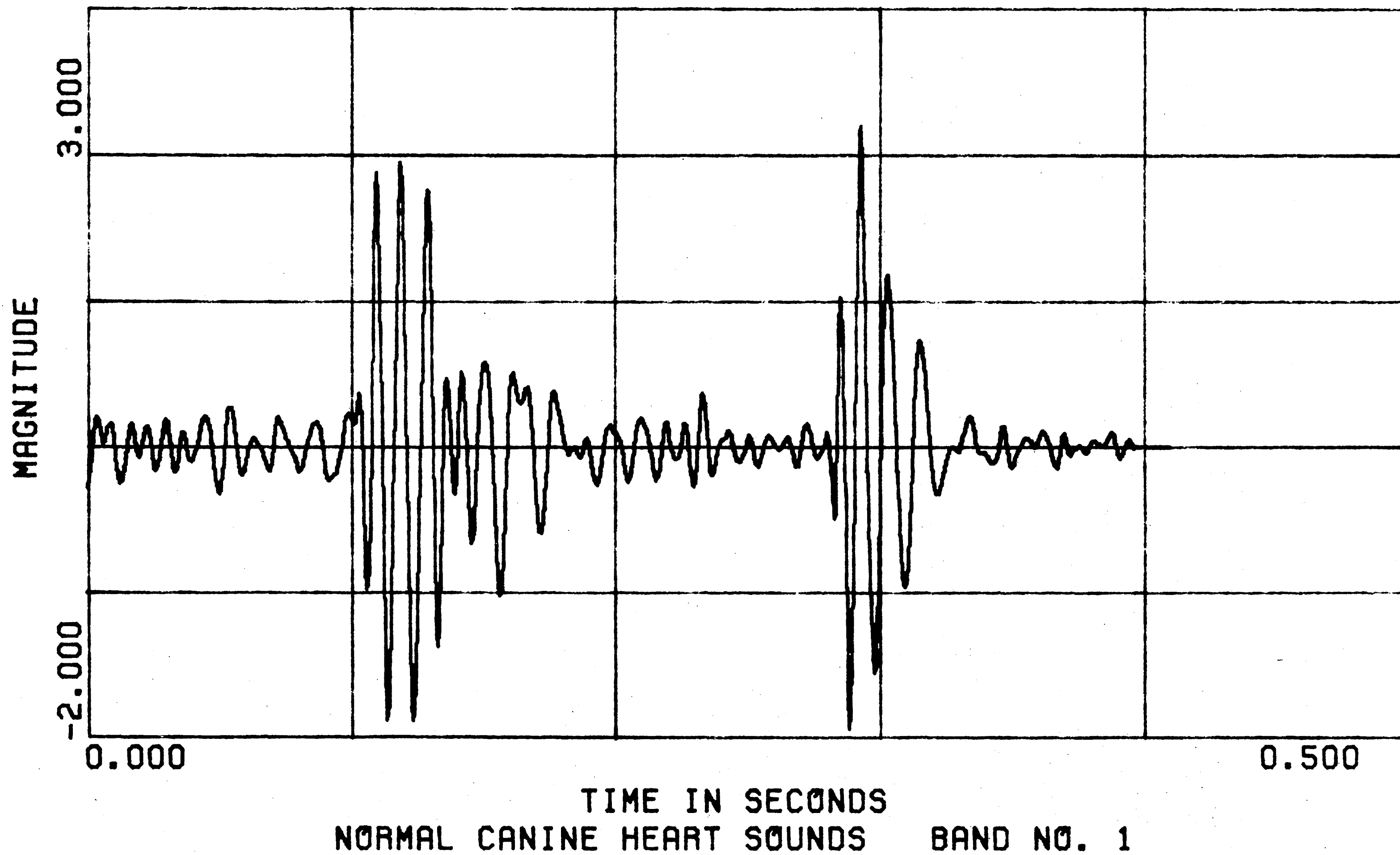


Figure 14. Sample Plot of Input Data Sequence

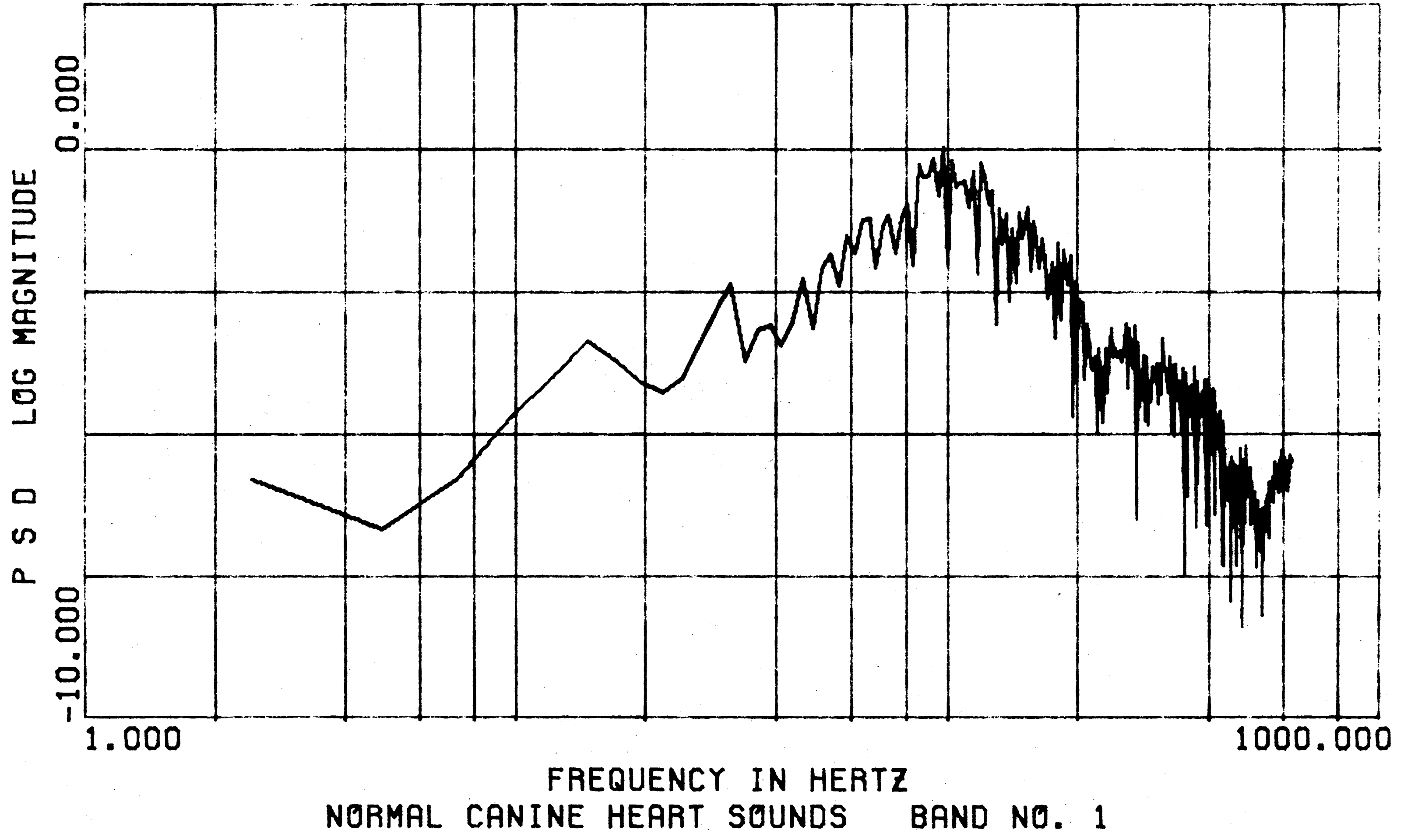


Figure 15. Sample Plot of the PSD Estimate

# CHAPTER VI

# CONCLUSIONS

An efficient system for the analysis of signal data via minicomputers has been designed. Techniques for overcoming some of the major problems associated with programming large systems on minicomputers have also been developed. Finally, a sample system based on these techniques was implemented on an Interdata 7/16 Basic minicomputer.

The major conclusion is that moderately large systems can effectively be implemented on minicomputers and that large data sequences can be analyzed easily. Of secondary importance is the generality of the concepts. The concepts are not restricted entirely to signal analysis, but can be applied to a wide variety of computer systems.

## BIBLIOGRAPHY


1. Weitzman, Cay. Minicomputer Systems, Structure, Implementation, and Application, New Jersey, Prentice-Hall, 1974.

2. Oppenheim, Alan V., and Ronald W. Schafer. Digital Signal Processing, New Jersey, Prentice-Hall, 1975.

3. Chirlian, Paul M. Signals, Systems and the Computer, New York, Intext Press Inc., 1973.

4. Gold, Bernard, and Charles M. Rader. Digital Processing of Signals, New York, McGraw Hill, 1969.

5. Doebelin, Ernest O. Measurement Systems, Application and Design, Revised edition, New York, McGraw Hill, 1975.

6. IEEE Transactions on Audio and Electroacoustics, (Special issue on the fast Fourier transform), Vol. AU-15, No. 2, June, 1967.

7. Cooley, James W., and John W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, Vol. 19, No. 90, April, 1967, pp. 297-301.

8. Cockran, William T. et al. "What is the Fast Fourier Transform," IEEE Transactions on Audio and Electroacoustics, Vol. AU-15, No. 2, June, 1967, pp. 45-55.

9. Singleton, R.C. "A Method for Computing the FFT with Auxiliary Memory and Limited High-Speed Storage," IEEE Transactions on Audio and Electroacoustics, Vol. AU-15, No. 2, June, 1967, pp. 91-98.

10. Kale, T.S., and S.K.R. Iyengar. "Digital Spectral Analysis - The Use of the Fast Fourier Transform," Fluid Power Research Conference, Oklahoma State University, Paper No. P75-33, October 7, 1975.

11. Bendat, Julius S., and Alan G. Piersol, Random Data: Analysis and Measurement Procedures, Wiley-Interscience, 1971.

12. Otnes, R.K., and L. Enochson. Digital Time Series Analysis, Wiley, 1972.

13. Blackman, R.B., and John W. Tukey. The Measurement of Power Spectra, New York, Dover Publications Inc., 1958.

14. Welch, Peter D. "The Use of Fast Fourier Transforms for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short Modified Periodograms," IEEE Transactions on Audio and Electroacoustics, Vol. AU-15, No. 2, June, 1967, pp. 70-73.

15. Introduction to Virtual Storage in System 370, Student Text, New York, International Business Machines Corp., 1972.

16. Harrison, Jonathon R. "Interactive Signal Processing System Using an Alphanumeric Graphic Computer Terminal, Based on the Fast Fourier Transform," IEEE Convention of Electrical and Electronics Engineers in Israel, 8th Proceedings, Tel Aviv, Vol. 8, April 30 - May 3, 1973, pp. 1-11.

17. Tenorio, Ramon A. Permanent Files and Control Cards for SIGANAL and SIGANAL2 at the Air Force Weapons Laboratory, Air Force Weapons Laboratory, Albuquerque, New Mexico.

18. 16-Bit Series Reference Manual, Publication Number 29-398R01, Oceanport, New Jersey, Interdata Inc., 1974.

19. Prescott, J., and R.L. Jenkins. "An Improved Fast Fourier Transform," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-22, No. 3, June, 1974, pp. 226-227.

20. Brenner, Norman. "FOURT-Cooley-Tukey Fast Fourier Transform," from the IBM Contributed Program Library, No. 360D-13.4.001 (1969 revision), New York, International Business Machines Corporation.

21. Disk Operating System (DOS) Reference Manual, Publication Number 29-293RO5, Oceanport, New Jersey, Interdata Inc., 1972.

# APPENDIX A

# USERS' GUIDE FOR THE OSU-MAE DIGITAL SIGNAL ANALYSIS SYSTEM

## APPENDIX A

## USERS' GUIDE FOR THE OSU-MAE DIGITAL SIGNAL ANALYSIS SYSTEM

### Introduction

This appendix presents a guide to the use of the OSU-MAE Digital Signal Analysis System, hereafter called the DSA. The guide is divided into six sections. The first section describes the capabilities and features found in the DSA. The second section describes the preparation of the files which will contain the digitized signal data. The third and fourth sections outline the commands used by the interactive input handler and describe the operation of the DSA with the DOS operating system. The fifth section lists the error messages and their meanings. The last section describes the procedures for adding routines to the overlay library.

At present, the DSA is limited to running with the DOS operating system on the Interdata Model 7/16 minicomputer. Should it be desired to change any of the main routines within the DSA, the user should carefully examine the listings of the source programs. These listings are included in Appendix B. Since the overlay

libraries are not yet complete, descriptions of signal analysis routines within the libraries are not included in this guide. It is suggested that an information sheet for each overlay library be maintained as routines are added.

## Capabilities and Features of the DSA

The DSA is an interactive minicomputer software system which is specifically designed to aid in the analysis of data sequences. The system requires the data sequences to be prestored in external files. Users enter interactive commands which manipulate the data files, direct the analysis which is to be performed, and control the output of results. The signal analysis routines are stored in a library as labelled overlays. This library is easily expandable by the user.

The remaining major capabilities and features of the DSA are summarized as follows:

1. The input handler of the DSA accepts free format input commands.
2. The DSA allows up to ten input files to be used at any one time.
3. The user may define up to ten temporary storage files to store intermediate results.
4. The DSA uses signal analysis routines which are stored in overlay libraries. These libraries are easily expanded by

the user. No changes to the main programs of the DSA are necessary when the libraries are expanded.

5. Users can define their own arguments for the commands which are used to execute routines from the overlay libraries.
6. Two user-oriented utility routines are available for transferring data to and from the temporary storage files.
7. The user can include routines in an overlay library which automatically call other routines from the same library.

Once a suitable library of overlays has been built, the analysis of signals becomes a simple matter of entering commands on the input console. Thus, subsequent users need not have any computer programming background to operate the DSA.

## Preparation of Input Data Files

The data which is to be analyzed by the DSA must be prestored in external files. The files should be either tape or disk files. If disk files are to be used, the file should be given an attribute of "direct physical" with the DOS attribute command. All files must conform to the following specifications.

1. All files should have a fixed record length of 256 bytes. Each record will therefore accommodate 64 real numbers, 32 complex numbers, or 128 integer numbers.

2. The first record of each file must be a header record.

3. The maximum length of disk and tape files is limited to 32,767 records. The maximum length is otherwise limited by the amount of physical disk or tape storage actually available.

The headers of the data files must be arranged as shown in Table V. The DSA does not require all 256 bytes of the header record and the remaining bytes may be defined in any manner the user desires.

## Interactive Command Summary

Interactive commands are read and handled by the DSA's input handler DSAIN. The DSAIN routine is an overlay which is loaded into the overlay area of memory automatically by the DSA. The DSAIN routine is loaded at system initialization and whenever the execution list is exhausted.

The DSAIN routine accepts free-format commands. Each command consists of an operation code followed by arguments separated by commas. Table VI is a summary of the commands and their action. When commands are entered to the system, the operation code must be preceded by the characters $$. Only the first two letters of the operation code need be entered, however, as many characters and blanks as desired can be input before the first comma. As an example, consider the command

TABLE V

ORGANIZATION OF HEADER RECORD FOR INPUT FILES

| Item Number | Number of Bytes | Word Type | Description |
|---|---|---|---|
| 1 | 4 | Real | Digitization interval in milliseconds or Hertz. |
| 2 | 2 | Integer | Digitization indicator<br>0 = Time (msec)<br>1 = Frequency (Hertz) |
| 3 | 4 | Real | Data starting value (based on indicator above) |
| 4 | 2 | Integer | Total number of records in file |
| 5 | 2 | Integer | Data word type:<br>0 = Real*4 (64 words per second)<br>1 = Complex (32 words per second)<br>2 = Integer*2 (128 words per second) |
| 6 | 2 | Integer | Gage factor (not used at present) |
| 7 | 50 | N/A | 50 character file title including trailing blanks |
| 8 | 50 | N/A | 50 character label for Y-axis of plot |
| 9 | 50 | N/A | 50 character label for X-axis of plot |
| 10 | 90 | N/A | Unused by DSA at present |

TABLE VI

INTERACTIVE COMMAND SUMMARY

| Command | Action Taken |
| --- | --- |
| $$INPUT, LU, STRREC, ENDREC | Defines an input file by logical unit.<br><br>LU - the logical unit to which the input file has been assigned.<br><br>STRREC - The starting record number of the input file from which data is to be taken<br><br>ENDREC - The last record of input from which data is to be taken<br><br>STRREC AND ENDREC are optional. If omitted STRREC defaults to 1 and ENDREC defaults to the number of records as given in the file header |
| $$REQUEST, ID, ±NUMREC | Request a temporary storage file with the name ID<br><br>ID - A two character file identifier, the first character of which must be an A-Z |

TABLE VI (Continued)

| Command | Action Taken |
|---|---|
| | NUMREC - The number of records to be reserved for the file.<br><br>If NUMREC < 0 then the temporary file is marked as a complex file.<br><br>If NUMREC > 0 the file is marked as a real file.<br><br>If NUMREC is omitted a total of 48 records will be reserved for the file.<br><br>A total of ten temporary files may be requested. |
| $$ALLOCATE | This command allocates the disk space required for the temporary files. The command is entered one time after all temporary files have been requested by the $$REQUEST command. |
| $$KILL | Deallocates the disk space that was allocated by the $$ALLOCATE command. All requested temporary files are destroyed and the data that was in them is lost. |
| $$MOVE,LU,ID | Moves the data from the input file LU to the temporary file ID. If the temporary file is complex and the input file is real, the data is moved to the real part of the temporary file. The imaginary part is set to zero. |

TABLE VI (Continued)

| Command | Action Taken |
|---|---|
| | If the temporary file is complex, the input file is real, and LU < 0, the data is moved to the imaginary part of the temporary files. The real part of the temporary file remains unaltered. In this manner, two input files may be paired for simultaneous FFT operations. |
| $$OUTPUT, ID, LU | Moves data from temporary file ID to file LU. File LU must be previously allocated by DOS and assigned to logical unit LU. This is a straight copy operation and the output file will have the same characteristics as the temporary file. A standard header is also written to the output file, therefore, the output file can later be used as an input file. |
| $$PAUSE | Causes the DSA to pause execution and return control to DOS. |
| $$DISPLAY, LU or ID | Information from the header of the input or temporary file (LU or ID) is displayed on the line printer. |
| $$GO | The DSA begins execution of the routines in the execution list. |
| $$END | The DSA ends execution and stops. Control is returned to DOS. |

$$REQUEST,F1,32

This command can also be input as

$$RE,F1,32

or

$$REQUEST TEMPORARY FILE,F1,32

In this manner the commands may be briefly documented as they are input.

Commands which are not preceded by the characters $$ are treated as overlay library routine names. These names, along with the arguments, are placed in the DSA's execution list. The execution list is capable of holding up to 48 routine names. Examples of these commands are

PLOT,F1

FFT,F1,F2,1.0

The arguments for these commands are defined by the overlay library routine which they name. Further information about these commands and their arguments can be found in the section of this appendix outlining the procedure for adding routines to the library.

There are two types of files that the DSA recognizes--temporary files and input files. Temporary files are identified by a two-character ID and input files are identified by LU number (1-10). The ID's for temporary files are assigned when the file is requested by the $$REQUEST command. The LU's are assigned to the input files by DOS with the ACTIVATE command.

## Operation of the DSA with DOS

The DSA runs under the Interdata DOS operating system. It is suggested that users have some knowledge of the DOS commands which activate files, assign logical units to physical units, allocate logical units to physical units, allocate disk space, assign attributes to files, and load and run programs. A complete description of the DOS commands can be found in the "Disk Operating System (DOS) Reference Manual," [21].

The DSA normally resides as an absolute load module in a disk file named DSA. Should it become necessary to recreate the object module, the following procedure is recommended.

1. Compile the following Fortran IV programs.
   - A. GET
   - B. PUT
   - C. DSAIN
2. Compile the following Fortran V programs.
   - A. DSA (main program)
   - B. SYSTEM (the execution list is contained in this routine)
   - C. IFETCH
   - D. FINISH
   - E. DECODE
   - F. CHECK
   - G. PACKN
   - H. ALLOCT

3. Allocate a binary disk file named DSA three cylinders in length.

4. Start the Interdata loader program and instruct it to create a load module on the file prepared in step 3 using OUT.

5. Request space for labeled common 300 hexadecimal bytes long with the LC command. Bias the load to a convenient starting address above the operating system using the loader BIAS command.

6. Load the DSA object program and link with the subroutines SYSTEM, IFETCH, GET, PUT, and FINISH.

7. Edit the Fortran run-time library to resolve all Fortran references.

8. The root segment of the DSA is now complete. Use the loader XOUT command to finish the load.

9. Instruct the loader to create an overlay with the OV command. This overlay is the input handler, DSAIN. Use the loader OUT command with label DSAIN such that the DSAIN overlay will reside on the file DSA immediately following the root segment previously loaded.

10. Link the subroutines DSAIN, DECODE, CHECK, PACKN, and ALLOCT.

11. Edit the Fortran run-time library and complete the load with the XOUT command.

This completes the creation of the DSA load module. A sample load map is shown in Figure 16. The DOS RUN command may now be used to execute the DSA.

```
RFL PROGS:
  3600 DSAMN     3B6E SYSTEM    3DDC IFETCH    3EEA PUT
  40C8 GET       42CC FINISH    434C SVC4      435E IOERR
  43D2 SYSIO     4484 .S        4498 .P        453C .Q
  45E4 .O        462C .MES      46A8 .U        46D8 .V
  46E6 @R        470A @Z        472E @H        4774 @H5
  77C8 DECODE    7B26 CHECK     7BB0 ALLOCT    7CFA PACKN
  7DDC POSITN    7E3E MIN0      7E50 .1        7E60 $1
  7E72 $2        7EAF IABS      7EE8 @G        8002 .H
  8048

ABS PROGS:
NONE

ENTRY-POINTS:
  3B6E SYSTEM    3C46 NAME      3D66 LIST      3E00 IFETCH
  3F0E PUT       40EC GET       42F0 FINISH    434C SVC4
  435E IOERR     43D2 SYSIO     4484 .S        44B8 .P
  453C .Q        45E4 .O        462C .MES      46B4 .U
  46DC .V        46E6 @R        470A @Z        472E @H
  47BE @H5       5B76 DSAMOD    77EC DECODE    7B4A CHECK
  7BD4 ALLOCT    7D1E PACKN     7DDC POSITN    7E3E MIN0
  7E50 .1        7E60 $1        7E72 $2        7EAE IABS
  7EE8 @G        8002 .H

COMMON-BLOCKS:
  FCFE SIGNAL

UNDEFINED:
 NONE
```

Figure 16. Sample Load Map for the DSA

The DSA automatically makes the following logical unit assignments.

1. Logical unit 0 (zero) is assigned to the plotter interface (physical unit 31).
2. Logical unit B is assigned to the Teletype.
3. Logical unit C is assigned to file containing the overlay library. When the system is started, users will be prompted for the overlay library file name.
4. Logical unit D is assigned to line printer (physical unit 62).
5. Logical unit E is assigned to file VSTOR which contains the temporary data storage for the DSA. This file is automatically allocated and deleted by the DSA.
6. Logical unit F is assigned to the file ARG. This file is used to store the arguments which will be passed to routines from the overlay library. This file is also allocated and deleted automatically by the DSA.

The logical units 1 through A are for input data files (see Table VI, $$INPUT command). The analysis system requires a file named INT2 (one cylinder, record length at least 12 bytes) to exist.

## Error Messages

There are three sources for error messages within the DSA. The first is the interactive input handler DSAIN. Table VII summarizes these messages and their meanings. The second source is from the DSA's main programs. Should the DSA not be able to locate an overlay name on the overlay library it prints the following

TABLE VII

INPUT ERROR MESSAGES

| Error Message | Reason |
|---|---|
| DUPLICATE TEMP, FILE NAME - REQUEST DENIED | A request for a temporary file with an ID that is already in use was made. |
| EOF ENCOUNTERED ON MOVE - REDEFINE INPUT | An error was detected during a move operation. Probable cause is an invalid input file. |
| ERR. ARG. LENGTH | One or more arguments in the command is too long. Maximum length for character arguments is two characters and for numeric arguments, ten characters. |
| ERR. MAX. TEMP. FILE | The last $$REQUEST exceeded the maximum number of temporary files allowed. |
| INPUT ERR. | The command is not recognizable. |
| INPUT FILE UNDEFINED | An operation was attempted on an input file not yet defined by the $$INPUT command. |
| MOVE ILL. BEFORE ALLOCATE | A move was attempted before any temporary storage space was allocated. |
| MOVE TO REAL FILE ILL. | An attempt to move a complex input file to a real temporary file was attempted. |

TABLE VII (Continued)

| Error Message | Reason |
| --- | --- |
| NOTHING TO ALLOCATE | A $$ALLOCATE was attempted before any temporary files were requested. |
| REQUEST ILL. AFTER ALLOCATE | An attempt to request another temporary file after space had already been allocated was made. |
| TOO MUCH SPACE REQUESTED - ALLOCATE FAILED | The number of disk cylinders required for the temporary files exceeds 300. |
| UNREQUESTED ID = XX | The ID in the command has not been associated with any temporary file. |

message

ROUTINE=XXXXXX DOES NOT RESIDE ON LIBRARY=ZZZZZZ.

where XXXXXX is the routine name requested and ZZZZZZ is the current overlay library name. Immediately after printing this message, the DSA reloads the input handler and the ? prompt is printed. The user then has two options available. He may reenter the overlay routine command and the new name will be placed at the top of the execution list. Or, he may just enter a blank line (typing a carriage return only) and any subsequent overlay routine names entered will be placed at the bottom of the execution list. In either case, the $$GO command is required to start the DSA executing routines from the execution list again.

The third source of error messages is from the individual overlay routines themselves. These messages are defined by the individual routines and their meanings should be included with the routine descriptions on the overlay library information sheet.

## Adding Routines to the Overlay Libraries

The DSA allows easy addition of routines to overlay libraries. No modification of the main programs of the DSA is necessary and only slight modification of existing signal analysis programs is required. These modifications involve mostly input/output of data. Routines that are to be added to the libraries specify their own commands and argument lists as well as error messages. A

facility is included in the DSA which allows one routine in a library to automatically call any other routine from the same library.

Each routine that is to be added to the overlay libraries may have a special linkage subroutine which decodes the arguments being passed, checks the arguments for errors, and if necessary, reads the header information from the file that is to be processed. This subroutine is always named DSAMOD. A more detailed description of DSAMOD can be found in the section entitled "Overlay Linkage" of Chapter V. A listing of a sample DSAMOD is included in Appendix B to serve as a guide for coding this subroutine.

Each routine to be added to the libraries must handle its own input and output of data. If the data to be processed resides on an input file, the routine should use unformatted read statements to fetch the data. For data which resides on temporary files, two utility subroutines, which are part of the DSA's root segment, must be used for data transfers. These subroutines are called GET and PUT and can be used in the following manner.

To fetch data from a temporary file use subroutine GET as follows

CALL GET(ID, BUFF, STRREC, NUMREC, IFLG)

where: ID – the two character file identifier from which data is to be transferred.

BUFF – the start address of the buffer to which the data is to be transferred.

STRREC – the starting record number in the temporary file where data transfer is to begin. Record number zero always contains the file header.

NUMREC – the number of records which are to be transferred.

IFLG – error flag returned by GET

0 = no error

–1 = undefined ID

1 = I/O error or record number out of range

To write data to a temporary file, use subroutine PUT as follows:

CALL PUT(ID,BUFF,STRREC,NUMREC,IFLG)

The arguments are defined the same as those for GET.

If it is desired to have the routine automatically call other overlays within the same library, subroutine SYSTEM is used to add these routines to the execution list. The usage of subroutine SYSTEM is

CALL SYSTEM(NAME, ABUFF,NARG,IFLG)

where: NAME – Six character name of routine to be added to list. The name must be a full six characters, left justified in the array, and padded right with blanks if necessary.

ABUFF – Start address of the argument buffer. To help standardize arguments, it is recommended that all numeric arguments be passed as real variables and

character arguments be left justified in a real variable.

NARG - Number of real arguments in ABUFF to be passed to the called routine.

If NARG < 0 the routine is added to the bottom of the execution list.

If NARG > 0 the routine is added to the top of the execution list.

If NARG = 0 no action is taken.

IFLG - error flag returned by SYSTEM

0 = no error, 1 = list overflow.

Care should be taken when using SYSTEM to insure that the called routines will be executed in the proper sequence. A simple rule to follow is that the first routine added to the top of list will be the last to be executed. It should also be noted that the calling routine will be overlayed by the called routine. If a return to the calling routine is desired after the called routines have executed, the calling routine should add itself to the top of list first. The DSA always executes routines from the top of the execution list. After the desired routines have been added to the list, the calling routine simply branches back to root segment of the DSA and the routines will be executed.

If for some reason the linkage subroutine DSAMOD detects an error, the input handler can be requested by setting the variable IDECF of common SIGNAL to 1 and executing a return. The user

is then allowed the two options described in the error message section of this appendix.

Once a routine, which is to be added to the library has been written and compiled it can be placed in the library as an overlay in the following manner.

1. The root segment must be loaded first to a dummy load module file as described in the procedure of the section entitled "Operation of the DSA with DOS." Only the first eight steps of this procedure should be performed. Use a null file for this step and not the file named DSA. The bias of this load must be the same as that which was used when the DSA file was created.
2. Position the overlay library file after the last routine on the file. This step is necessary only if the loader used does not position the file automatically.
3. Use the loader OV command to inform it an overlay is about to be linked.
4. Use the loader OUT command to direct the overlay to the library file. The label field of the OUT command must be included. This label will be the command word which is entered to the DSA when it is desired to execute the new routine.
5. Link the DSAMOD subroutine first followed by the routine and all additional routines that are required.

6. Edit the Fortran run-time library if necessary.

7. Complete the load with the loader XOUT command.

The new routine has now been added to the overlay library and is ready for use. The load map of this load operation should be compared with the load map obtained when the DSA file was created. The entry point address of DSAMOD must be the same on both maps.

# APPENDIX B

# ROUTINE LISTINGS

```
$ASSM
DSAMN PROG  MAINLINE ROUTINE FOR DSA SYSTEM (ROOT SEGMENT)
      SCRAT
$FORT
C
C   THIS IS THE MAIN ROUTINE FOR THE DIGITAL SIGNAL
C  ANALYSIS SYSTEM (DSA).   IT HANDLES THE OVERLAY LOADING
C  AND TRANSFERS.   THE ROUTINE IS ONLY PART OF THE SYSTEM
C  ROOT SEGMENT.   THE OTHER ROUTINES INCLUDED IN THE ROOT
C  ARE:
C        SYSTEM
C        PUT
C        GET
C        IFETCH
C        FINISH
C
      IMPLICIT INTEGER*2 (I-N)
C
      COMMON/SIGNAL/IDENT(5,20),IAL,IPOINT,ILU,IDECF,HEAD(128),
     1  ABUF(64),NARG,IPRLU,IPLTLU
C
      INTEGER*2 IDENT,HEAD,LNAME(6),LACTIV(7),MES1(6),MES2(8),
     1  MES3(7),MES4(8),MES5(3),MES6(3),MES7(3),MES8(5),MES9(4),
     2  MES10(5)
C
      INTEGER*2 HEADLN(16),CURNAM(3)
C
      DATA ISLU/12/
      DATA HEADLN/'OSU-MAE DIGITAL SIGNAL ANALYZER '/
      DATA IBLNK/'  '/
      DATA LACTIV/'AC',' D','SA','LI','B,','C ',X'0D20'/
      DATA MES1/'AC INT2,C ',X'200D'/
      DATA MES2/'AL ARG,E,1,256',X'200D'/
      DATA MES3/'AT ARG,0020 ',X'0D20'/
      DATA MES4/'AS 802,031,DC2',X'200D'/
      DATA MES5/'LI C',X'200D'/
      DATA MES6/'VSTOR '/
      DATA MES7/'ARG   '/
      DATA IDID/0/
      DATA MES8/'DE VSTOR',X'200D'/
      DATA MES9/'DE ARG',X'200D'/
      DATA MES10/'AC DSA,C',X'200D'/
      DATA IHAVE/0/
C
C   SET THE LUS
C
      IPRLU=13
      IPLTLU=0
      ILU=11
C
      IF(IDID.NE.0) GO TO 5
C
C   PRELIMENARY FILE CHECK
C
      CALL SVC4(MES4)
      CALL SYSIO(32,ILU,ISTDN,ISTDEV,HEADLN,HEADLN(16),2,0,0)
      CALL SVC4(MES1)
      CALL SVC4(MES5)
      ENDFILE 12
      REWIND 12
 3    READ(12,1110,END=4) LNAME
 1110 FORMAT(6A2)
      IF(LNAME(1).NE.MES6(1).AND.LNAME(1).NE.MES7(1)) GO TO 3
      IF(LNAME(2).NE.MES6(2).AND.LNAME(2).NE.MES7(2)) GO TO 3
      IF(LNAME(3).NE.MES6(3).AND.LNAME(3).NE.MES7(3)) GO TO 3
      IF(LNAME(1).EQ.MES6(1)) CALL SVC4(MES8)
      IF(LNAME(1).EQ.MES7(1)) CALL SVC4(MES9)
      CONTINUE
      GO TO 3
      GO TO 3
 4    CALL SVC4(MES2)
      CALL SVC4(MES3)
 5    CALL SVC4(MES10)
      IDID=1
C
C   FETCH THE INPUT HANDLER OVERLAY
C
      CALL IFETCH('DSAIN ',12,ISTAT)
      IHAVE=1
      WRITE(ILU,1000)
      READ(ILU,1100) LNAME
      IS=7
 13   IS=IS-1
      IF(IS.EQ.0) GO TO 14
      IF(LNAME(IS).EQ.IBLNK) GO TO 13
      CONTINUE
$ASSM
*
*   FIX LACTIV ARRAY FOR THE CORRECT LIBRARY
*
      LIS     3,1
      LIS     1,0
GOAG  LB      2,LNAME(1)      GET FIRST CHAR OF NEW LIB
      STB     2,LACTIV+2(3)   AND STORE IT IN LACTIV
      CLH     3,IS            FINISHED YET?
      BNL     REST
      AIS     3,1             INCREMENT THE POINTERS
      AIS     1,2
      B       GOAG            BACK FOR MAORE CHARS
REST  LHI     2,C',C'
      STB     2,LACTIV+3(3)   STORE THE COMMA AND C
      EXBR    2,2
      STB     2,LACTIV+4(3)
```

```
      LHI      2,X'0D20'
      STB      2,LACTIV+5(3)   STORE THE LAST PART
      EXBR     2,2
      STB      2,LACTIV+6(3)
$FORT
 14   CALL SVC4(LACTIV)
      DO 10 I=1,20
      DO 10 J=1,5
 10   IDENT(J,I)=-1
      IAL=0
      IPOINT=9
      IDECF=0
      NARG=0
 12   IF(IHAVE NE 0) GO TO 15
      CALL SVC4(MES10)
      CALL IFETCH('DSAIN ',12,ISTAT)
      CALL SVC4(LACTIV)
      IHAVE=1
C
C   CALL THE OVERLAY
C
 15   IF(IHAVE.EQ.0) WRITE(ILU,1110) CURNAM
      CALL DSAMOD
      IF(IDECF.NE.0) GO TO 12
      CONTINUE
C
C   THIS IS THE RETURN POINT  SO CHECK THE LIST
C   FOR POSSIBLE ROUTINES IN THE EXECUTION LIST
C
$ASSM
      EXTRN  NAME,LIST,IFETCH,IOERR
*
*   GET THE LIST'S CURRENT TOP
*
      LIS     1,2
      LIS     2,0
      LB      3,LIST(1)         GET THE CURRENT TOP.
      RTL     1,LIST            GET THE BYTE COUNT FROM THE LIST
      BC      $P12              GO GET INPUT IF LIST IS EMPTY
      SRLS    1,2               DIVID THE BYTE COUNT BY FOUR
      SHI     1,1
      STH     3,RAND            STORE THE POINTER FOR ARGUMENT FETCH
      STH     1,NARG            STORE THE NUMBER OF ARGUMENTS
      SVC     1,PARBLK          GET THE ARGUMENTS
      LH      4,IST             GET THE STATUS
      BZ      GOGO              GOOD STATUS
      BAL     15,IOERR
      DC      X'0004'
      DC      IST
      B       $P12
GOGO  MH      2,SIX             MULTIPLY POINTER BY SIX
      LHI     2,NAME(3)         GET THE NAME ADDRESS.
      STM     13,TSAV
      LIS     1,0
      CLH     1,IHAVE
      BNE     STO
      LM      13,CURNAM         GET THE CURRENT NAME
      CLH     13,0(2)           AND CHECK TO
      BNE     STO
      CLH     14,2(2)           SEE IF IT IS THE
      BNE     STO
      CLH     15,4(2)           SAME AS WHAT IS WANTED NOW.
      BNE     STO
      LM      13,TSAV
      B       $P15              EQUAL SO CALL OVERLAY
STO   LM      13,0(2)           GET THE NEW NAME
      STM     13,CURNAM         AND MAKE IT THE CURNAM
      STH     2,NADD            STORE THE NAME ADDRESS
      BAL     15,IFETCH         AND FETCH THE OVERLAY TO CORE.
      DC      X'0008'
NADD  DC      0
      DC      ISLU
      DC      IST
      LM      13,TSAV
      LH      1,IST             GET THE FETCH STATUS
      BNZ     $P30              GO TO WRITE ERROR
      LIS     1,0
      STH     1,IHAVE
      B       $P15              NOW CALL OVERLAY.
*
*
PARBLK DB     92,14
IST   DC      0
      DC      ABUF
      DC      ABUF+255
RAND  DC      0
SIX   DC      6
TSAV  DS      6
*
*
$FORT
 30   CONTINUE
      WRITE(ILU,1010) CURNAM,LNAME
      GO TO 12
C
C   FORMAT STATEMENTS
C
 1000 FORMAT('ENTER LIBRARY NAME')
 1100 FORMAT(6A1)
 1010 FORMAT('ROUTINE',1X,3A2,' NOT FOUND ON LIB = ',6A1)
C
      STOP
      END
```

```
$ASSM
SYSTEM PROG FORTRAN CALLABLE ROUTINE FOR DSA SYSTEM
       SCRAT
       SQUEZ
       CROSS
$FORT
C
C      SUBROUTINE SYSTEM (NAME,BUFFER,IPUT,IFLG)
C
C     THE SYSTEM SUBROUTINE IS CALLED TO ADD A ROUTINE TO THE EXECUTION
C   STREAM OF THE DSA  SYSTEM MAINTAINS A CIRCULAR LIST
C   AS DESCRIBED IN THE 16-BIT PROCESSOR MANUAL (INTERDATA).
C   ROUTINES ARE ALWAYS EXECUTED FROM THE TOP OF THE LIST.   USERS
C   MAY INCLUDED A ROUTINE TO EITHER THE TOP OR BOTTOM OF THE LIST.
C   A MAXIMUM OF 256 BYTES ARE PREMITTED FOR ARGUMENTS.
C
C   THE ARGUMENTS ARE DEFINED AS :
C      NAME = FULL SIX CHARACTER ROUTINE NAME (LEFT JUSTF.  PADDED
C             WITH BLANKS).
C      BUFFER = THE ARGUMENT BUFFER TO BE PASSED TO THE CALLED
C               ROUTINE.
C      IPUT = THE NUMBER OF BYTES IN THE ARGUMENT BUFFER.
C             IF IPUT<0 THE NAMED ROUTINE IS ADDED TO THE BOTTOM
C                        OF THE LIST.
C             IF IPUT = 0  NO ACTION IS TAKEN
C             IF IPUT>0 THE ROUTINE IS ADDED TO THE TOP OF THE LIST.
C      IFLG = ERROR FLAG RETURN TO THE CALLING ROUTINE.
C             IFLG=0  NO ERRORS
C             IFLG = 1  LIST OVERFLOW (NO MORE ROOM)
C             IFLG = 2 INCOMPLETE TRANSACTION
C
$ASSM
       ENTRY  SYSTEM,LIST,NAME          ROUTINE ENTRY POINT MARKER
       EXTRN  .O,IOERR
SYSTEM STM    7,REGSAV        SAVE THE CALLERS REGS.
       LH     14,0(15)        GET THE ARGUMENT COUNT
       SIS    14,10           CHECK THE ARGUMENT COUNT.
       BZS    ARGOK           THEY ARE OK PROCEED.
       LHI    11,C'33'        SET RTL ERROR MSG NUMBER
       BAL    15,.O           GO PRINT ERROR MSG
       B      SET2            SET ERROR FLAG AND RETURN
*
ARGOK  XHR    8,8             ZERO R8.
       LHR    12,15           MAKE R12 THE LINK POINTER
       LH     10,2(12)
       LM     13,0(10)        GET THE NAME
       LH     10,6(12)        GET THE BYTE COUNT
       LH     7,0(10)
       BZ     GORET           GO GORET FROM LIST IF ZERO
       CHI    7,0             IS THE COUNT POSITIVE OR NEGITIVE ?
       BM     NEGCNT          BRANCH IF IT IS NEG.
       LB     9,POINT         GET THE TOP OF LIST POINTER
       ATL    7,LIST          ADD THE BYTE COUNT TO THE TOP OF LIST.
       BO     SET1            IF OVERFLOW GO SET FLAG.
       SIS    7,1             DECREMENT THE COUNT BY ONE.
       MH     8,SIX           MULTIPLY BY SIX.
       STM    13,NAME(9)      AND STORE THE NAME
       DH     8,SIX           RESTORE THE POINTER
       B      GOGO            BRANCH TO WRITE ARGUMENTS.
NEGCNT XHI    7,X'FFFF'       MAKE THE COUNT POSITIVE
       AIS    7,1
       LB     9,POINT+1       GET TO CURRENT BOTTOM POINTER
       ABL    7,LIST          AND ADD THE BYTE COUNT TO THE BOTTOM.
       BO     SET1            IF OVERFLOW GO SET FLAG
       SIS    7,1             DECREMENT THE BYTE COUNT.
       MH     8,SIX           MULTIPLY BY SIX.
       STM    13,NAME(9)      AND STORE THE NAME.
       DH     8,SIX           RESTORE THE POINTER.
*
GOGO   LH     13,4(12)        GET THE BUFFER ADDRESS.
       STH    13,STADD        AND STORE IN START ADDRESS OF PARBLK.
       AHR    13,7            ADD THE BYTE COUNT TO GET FINAL ADD.
       STH    13,FIADD        AND STORE IT TOOOO!
       STH    9,RAND          STORE THE POINTER IN THE ADDRESS BLOCK.
       SVC    1,PARBLK        WRITE THE AGRUMENTS
       LB     13,STAT         GET THE STATUS OF OPERATION
       CHI    13,0            IS IT ZERO ?
       BE     GORET           YES !  GOOD RETURN
       BAL    15,IOERR        GO WRITE ERROR MSG.
       DC     X'0004'
       DC     STAT
SET2   LH     13,8(12)        GET THE FLG ADDRESS.
       LIS    14,2            AND SET IT
       STH    14,0(13)        AND STORE IT
       B      RET             AND RETURN
SET1   LH     13,8(12)        GET THE FLG ADD.
       LIS    14,1            AND SET THE
       STH    14,0(13)        FLAG VALUE FOR OVERFLOW
       B      RET             THEN RETURN
GORET  LH     13,8(12)        GET THE FLAG ADD.
       LIS    14,0            AND SET FOR NO ERR
       STH    14,0(13)
RET    LM     7,REGSAV        RESTORE CALLERS REGS.
       AH     15,0(15)
       BR     15              AND RETURN
*
PARBLK DC     X'380E'
STAT   DC     0
STADD  DC     0
FIADD  DC     0
RAND   DC     0
SIX    DC     6
*
*  SYSTEM EXECUTION LIST BEGINS HERE
```

```
********************************************************
NAME    DS      288
LIST    DB      48,0
POINT   DC      0
        DS      96              96 BYTES FOR LIST
********************************************************
REGSAV DS       18              SAVE SPACE FOR CALLER REGS.
        END
$FORT
      STOP
      END
```

```
$ASSM
IFETCH   PROG FORTRAN OVERLAY FETCHING ROUTINE
         SCRAT
         CROSS
         SQUEZ
$FORT
      SUBROUTINE IFETCH(NAME,LU,ISTAT)
         IMPLICIT INTEGER*2(A-Z)
C*******************************************************************
C   THIS ROUTINE LOADS A NAMED OVERLAY FROM A LIBRARY
C OF OVERLAYS THE AGRUMENTS ARE:
C
C   NAME - SIX CHARACTER BINARY NAME OF OVERLAY TO BE
C          LOADED.  LEFT JUSTIFIED, PADDED TO SIX
C          SIX CHARACTERS WITH BLANKS.
C
C   LU -   LOGICAL UNIT FROM WHICH OVERLAY IS TO BE LOADED.
C
C   ISTAT - RETURN CODE  0=ALL OK,  1=END OF FILE ERROR.
C
C*******************************************************************
         INTEGER*2 BUFER(6),NAME(1)
         DATA FFFF,0F0F/X'FFFF',X'0F0F'/
$ASSM
*  PUT THE NAME AND LU INTO FETCH PARBLK
*
         LH   1,NAME               GET THE ADDRESS OF NAME.
         STM  13,BUFER             STORE THE FORTRAN REGS.
         LM   13,0(1)              GET THE NAME AND STORE
         STM  13,FETCH             IT IN THE PARBLK.
         LM   13,BUFER             RESTORE THE FORTRAN REGS.
         LH   1,LU                 GET THE LU ADDRESS.
         LH   2,0(1)               GET THE LU NUMBER.
         STH  2,FETLU+2            AND STORE IT.
         STB  2,PARBLK+1           AND ALSO IN THE I/O PARBLK.
$FORT
         REWIND LU
 23      CONTINUE
$ASSM
         SVC  1,PARBLK             GET A RECORD.
         LB   2,PARBLK+2           GET THE STATUS.
         CLHI 2,0                  AND CHECK IF ZERO
         BNE  $P102                SET FLAG IF NOT.
$FORT
         IF(BUFER(1).NE.FFFF) GO TO 23
         CONTINUE
$ASSM
         LB    1,BUFER+4           GET THE
         SRLS  1,4                 CONTROL ITEM.
         CLB   1,0F0F              IS IT F?
         BNE   $P23                NO, UNLABELED PROGRAM.
         SHR   1,1                 ZERO REG 1
```

```
LOOP    LB      4,FETCH(1)         GET THE NEXT CHAR.
        LB      2,BUFER+4(1)       GET PART OF CHAR.
        LB      3,BUFER+5(1)       GET SECOND PART.
        EXBR    3,3                PUT SECOND PART IN HIGH END.
        RRL     2,4                GET CHAR IN HIGH PART OF REG 3.
        SRLS    3,8                PUT IT IN LOW PART OF 3.
        CLHR    3,4                ARE THE CHARS EQUAL?
        BNE     $P20               NO, GO READ AGAIN.
        AIS     1,1                INCREMENT REG 1.
        CHI     1,6                SIX CHARACTERS ?
        BL      LOOP
BKS     SVC     1,BACK
        SVC     5,FETCH            FETCH THE OVERLAY
        B       $P30               DONE, BACK TO FORTRAN.
BACK    DC      X'A001'
        DC      0
PARBLK  DC      X'5800'
        DC      X'0000'
        DC      BUFER
        DC      BUFER+11
        DC      0
FETCH   DC      0,0,0
FETLU   DC      0,0
$FORT
 30     ISTAT=0
        RETURN
 100    ISTAT=1
        RETURN
        END
```

```
$ASSM
FINISH PROG EOJ ROUTINE FOR DSA SYSTEM
 SCRAT
$FORT
      SUBROUTINE FINISH
      IMPLICIT INTEGER*2 (I-N)
C
C     SUBROUTINE EOJ TERMINATES THE DSA SYSTEM OPERATION.
C
C   WHEN A CALL TO FINISH IS MADE, ALL AYATEM FILES ARE DELETED
C   AND AN SVC3 (EOJ) CALL IS MADE.
C
      COMMON/SIGNAL/IDENT(5,20),IAL,IPOINT,ILU,IDECF,HEAD(128),
     1  ABUF(64),NARG,IPRLU,IPLTLU
C
      CALL SVC4('DE ARG  ')
      IF(IAL.EQ.0) GO TO 10
      CALL SVC4('DE VSTOR  ')
 10   CONTINUE
      IPOINT=9
$ASSM
      SVC  3,0          ISSUE END OF JOB
$FORT
      STOP
      END
```

```
      SUBROUTINE PUT(ID,STADD,STSEC,NUMSEC,IFLG)
C
C     SUBROUTINE PUT STORES SIGNAL DATA IN THE TEMP FILE
C     ID   IT STARTS STORING AT SECTOR STSEC AND STORES NUMSEC
C     SECTORS.   IF THE FILE WILL NOT HOLD ALL DATA THAT IS WRITTEN
C     TO IT THE FLAG IS SET AND A RETURN IS MADE
C     STADD IS THE STARTING ADDRESS OF THE BUFFER FROM WHICH THE
C     DATA IS TO BE WRITTEN.
C
      IMPLICIT INTEGER*2 (I-N)
      INTEGER*2 STADD(1),STSEC
C
      INTEGER*2 IDENT,HEAD
C
      COMMON/SIGNAL/ IDENT(5,20),IAL,IPOINT,ILU,IDECF,HEAD(128),
     1  ABUF(64),NARG,IPRLU,IPLTLU
C
      DATA ITLU/15/
C
C   SEARCH IDENT FOR ID
C
      IF(IPOINT.LE.9) GO TO 21
      DO 20 I=10,IPOINT
      IF(IDENT(5,I).EQ.ID) GO TO 22
 20   CONTINUE
 21   IFLG=-1
      RETURN
 22   ILD=I
      ISTR=IDENT(3,ILD)
      ISTR=ISTR+STSEC
      IF(NUMSEC+STSEC-1.GT.IDENT(2,ILD)) GO TO 90
      NUM=NUMSEC*128
      CALL SYSIO(60,ITLU,ISTDN,ISTDEV,STADD,STADD(NUM),2,ISTR,0)
      IF(ISTDN.EQ.0) GO TO 100
      CALL IOERR(ISTDEV)
 90   IFLG=1
      RETURN
 100  IFLG=0
      RETURN
      END
```

```
      SUBROUTINE GET(ID,STADD,STSEC,NUMSEC,IFLG)
C
C     SUBROUTINE GET READS DATA FROM A TEMP STORAGE FILE
C   AND STORES IT IN THE BUFFER STARTING AT STADD.   THE
C   ROUTINE STARTS FETCHING AT SECTOR STSEC AND RETRIVES
C   NUMSEC SECTORS OF DATA.   IFLG IS THE ERROR FLAG.
C
      IMPLICIT INTEGER*2 (I-N)
      INTEGER*2 IDENT,HEAD,STSEC,STADD(1)
C
      COMMON/SIGNAL/IDENT(5,20),IAL,IPOINT,ILU,IDECF,HEAD(128),
     1  ABUF(64),NARG,IPRLU,IPLTLU
C
      DATA ITLU/15/
C
      IF(IPOINT.LE.9) GO TO 21
C
C   SEARCH IDENT FOR ID
C
      DO 20 I=10,IPOINT
      IF(IDENT(5,I).EQ.ID) GO TO 22
 20   CONTINUE
 21   IFLG=-1
      RETURN
 22   ILD=I
      IF(IDENT(1,ILD).LT.0) GO TO 21
      ISTR=IDENT(3,ILD)
      ISTR=ISTR+STSEC
      IF(NUMSEC+STSEC-1.GT.IDENT(2,ILD)) GO TO 90
      NUM=NUMSEC*128
      CALL SYSIO(92,ITLU,ISTDN,ISTDEV,STADD,STADD(NUM),2,ISTR,0)
      IF(ISTDN.EQ.0) GO TO 100
      CALL IOERR(ISTDEV)
 90   IFLG=1
      RETURN
 100  IFLG=0
      RETURN
      END
```

```
      SUBROUTINE DSAMOD
C
C       INPUT IS THE MAIN ROUTINE FOR HANDILING SIGNAL PROCESSOR INPUT
C     IT IS IN THE FORM OF AN OVER LAY AND IS CALLED WHENEVER
C     THE SYSTEM REQUIRES INPUT
C
C       THE FOLLOWING DESCRIBES THE BASIC COMMANDS THAT THE
C    PROCESSOR ACTS ON.
C
C     COMMAND                   ACTION
C------------------------------------------------------------------------------
C     INPUT,LU,ISTR,IEND        DEFINES AN INPUT LU WHICH STARTS
C                               WITH THE SECTOR ISTR AND ENDS AT THE
C                               SECTOR IEND FROM THE INPUT FILE.
C     REQUEST,ID,+-NUM,...      REQUEST IS ENTERED TO ASK THE SYSTEM
C                               TO ALLOCATED TEMPORARY STORAGE FOR
C                               INTERMEDIATE WORK SPACE.  UP TO 10
C                               TEMP. FILES MAY BE REQUESTED.  ID IS
C                               A ONE OR 2 CHARACTER FILE IDENTIFIER
C                               NUM IS THE AMOUNT OF STORAGE IN SECTORS
C                               NEEDED IN THAT FILE.  IF NUM < 0
C                               THE ROUTINE ALLOCATES A COMPLEX STOR-
C                               AGE FILE OF LENGTH NUM
C     ALLOCATE                  THE ROUTINE ALLOCATES DISK SPACE FOR
C                               ALL NAMED TEMPORARY FILE IDS.  ONCE
C                               ALLOCATE IS MADE THE ROUTINE WILL
C                               NOT EXCEPT ANY MORE REQUEST FOR TEMP.
C                               STORAGE.  THE KILL COMMAND WILL CHANGE
C                               THIS SITUATION
C     KILL                      KILLS REQUESTED TEMPORARY STORAGE.
C                               WHEN KILL IS ENTERED ALL DISK SPACE
C                               FOR TEMP FILES IS DEALLOCATED AND
C                               ALL REQUESTED TEMP STORAGE FILES ARE
C                               DELEATED.  ANY DATA IN THE TEMP. FILES
C                               IS LOST.
C     MOVE,+-LU,ID,...          THIS COMMAND CAUSES THE ROUTINE TO MOVE
C                               DATA INTO A TEMP STORAGE FILE .
C                               THIS IS NECESSARY IF THE ROUTINE THAT
C                               OPERATES ON THE DATA REWRITES THE DATA IN
C                               PLACE  SUCH AS THE FFT ROUTINE
C                               IF LU <0 THE ROUTINE MOVES THE DATA
C                               FROM ABS(LU) INTO THE COMPLEX PART OF TEMP ID
C     ROUTINE,ARG1,ARG2,...     ROUTINE SPECIFIES AN OVERLAY LIB ROUTINE
C                               NAME TO BE ADDED TO THE EXECUTION LIST
C                               THE ARGUMENT LIST IS TO BE PASSED TO THE
C                               ROUTINE   THIS COMMAND CAUSES NO ACTION
C                               OTHER THAN ADDING THE ROUTINE TO THE LIST.
C     GO                        THE INPUT ROUTINE FINISHES ITS CHORES
C                               AND EXITS TO THE SYSTEM.  THE SYSTEM THEN
C                               STARTS EXECUTION OF THE ROUTINES
C                               NAMED IN THE EXECUTION LIST.  WHEN THE
C                               LIST IS EMPTIED  THE INPUT ROUTINE IS
C                               AGAIN OVERLAYED INTO THE SYSTEM
C                               FOR MORE INPUT.
C     DISPLAY,ID OR LU          DISPLAYS THE HEADER INFO
C                               FROM THE FILE ASSOCIATED WITH
C                               THE ID OR LU.  ALL PRINTING GOES TO
C                               THE LINE PRINTER FOR FASTER RESULTS.
C
      IMPLICIT INTEGER*2 (I-N)
      INTEGER*2 IDENT,HEAD
      INTEGER*2 OPCODE(10,2),IMAGE(80),COMMA,RPREN,LPREN,MYCOMA(10),
     1  NAME(3),TITLE(25)
C
      COMMON/SIGNAL/IDENT(5,20),IAL,IPOINT,ILU,IDECF,HEAD(128),
     1  ABUF(64),NARG,IPRLU,IPLTLU
      DIMENSION RBUF(128),RBUF2(64)
      EQUIVALENCE (HEAD(1),DINV),(HEAD(3),IDIN),(HEAD(4),STIME),
     1            (HEAD(6),NUMSEC),(HEAD(7),ITYPE),(HEAD(8),IGF),
     2            (HEAD(9),TITLE(1))
      DATA OPCODE/'I ','M ','R ','K ','G ','O ','A ','P ','D ','E ',
     1            'N ','O ','E ','I ','O ','U ','L ','A ','I ','N '/
      DATA NOP/10/
      DATA IBLNK/'  '/
      DATA COMMA/',,'/
      DATA IDOLAR/'$ '/
C
C
C     ZERO MYCOMA ARRAY
C
 2    DO 10 I=1,10
 10   MYCOMA(I)=0
C
 5    WRITE(ILU,2000)
      READ(ILU,1000) IMAGE
C
C     CHECK IF IMMEDIATE CODE
C
      IADD=0
      IS=0
 12   IS=IS+1
      IF(IS.LT.81) GO TO 13
      IDECF=0
      GO TO 5
 13   IF(IMAGE(IS).EQ.IBLNK) GO TO 12
      IF(IMAGE(IS).EQ.IDOLAR) GO TO 11
      IADD=1
      GO TO 16
 11   DO 15 J=1,NOP
      IF(IMAGE(IS+2).EQ.OPCODE(J,1).AND.IMAGE(IS+3).EQ.OPCODE(J,2))
     1  GO TO 16
 15   CONTINUE
      WRITE(ILU,1110)
      GO TO 5
```

```
C
C    SET MYCOMA FOR DECODE
C
 16    IPUT=0
       DO 20 I=1,80
       IF(IMAGE(I).NE.COMMA) GO TO 20
       IPUT=IPUT+1
       IF(IPUT.GT.10) GO TO 900
       MYCOMA(IPUT)=I
 20    CONTINUE
       IS=81
 22    IS=IS-1
       IF(IMAGE(IS).EQ.IBLNK) GO TO 22
       IPUT=IPUT+1
       MYCOMA(IPUT)=IS+1
       IF(IADD.NE.0) GO TO 700
       GO TO (100,150,200,250,300,350,400,450,800,850),J
C
C    INPUT STATEMENT PROCESSOR
C
 100   CALL DECODE(MYCOMA,IMAGE,1,-1,R,LU,IFLG)
       IF(IFLG.NE.0) GO TO 910
       CALL DECODE(MYCOMA,IMAGE,2,-1,R,ISTR,IFLG)
       IF(IFLG.LT.0) ISTR=1
       IF(IFLG.GT.1) GO TO 910
       IF(IFLG.GT.0) GO TO 900
       IF(ISTR.LT.1) ISTR=1
       CALL POSITN(LU,0)
       READ(LU) HEAD
       CALL DECODE(MYCOMA,IMAGE,3,-1,R,IEND,IFLG)
       IF(IFLG.GT.1) GO TO 910
       IF(IFLG)111,112,900
 111   IEND=NUMSEC
       GO TO 117
 112   IF(IEND.GT.NUMSEC) IEND=NUMSEC
       IF(IEND.GT.32000) IEND=32000
       IF(IEND.LT.ISTR) GO TO 910
 117   IDENT(1,LU)=ISTR
       IDENT(2,LU)=IEND
       IDENT(4,LU)=ITYPE
       CALL DECODE(MYCOMA,IMAGE,4,1,R,IPD,IFLG)
       IF(IFLG.GT.1) GO TO 910
       IF(IFLG)113,114,900
 113   IPD=0
       GO TO 115
 114   IPD=1
 115   IDENT(3,LU)=IPD
       GO TO 2
C
C    MOVE PROCESSOR
C
 150   IF(IRL.EQ.0) GO TO 930
       CALL DECODE(MYCOMA,IMAGE,1,-1,R,LU,IFLG)
       IF(IFLG.NE.0) GO TO 910
       IL=IABS(LU)
       IF(IDENT(1,IL).EQ.-1) GO TO 960
       CALL DECODE(MYCOMA,IMAGE,2,1,R,ID,IFLG)
       IF(IFLG.EQ.1) GO TO 900
       IF(IFLG.NE.0) GO TO 910
C
C    SEARCH IDENT FOR ID
C
       DO 152 I=1,IPOINT
       IF(IDENT(5,I).EQ.ID) GO TO 153
 152   CONTINUE
       GO TO 920
 153   ILD=I
C
C    MOVE THE HEADER AND UPDATE IT
C
       CALL POSITN(IL,0)
       READ(IL,END=158) HEAD
       HEAD(6)=MIN0(IDENT(2,ILD)-1,IDENT(2,IL))
       HEAD(7)=IDENT(4,ILD)
       CALL PUT(ID,HEAD,0,1,IFLG)
       IF(IFLG.NE.0) GO TO 158
       IF(IDENT(4,ILD)-1) 154,170,154
C
C    TEMP FILE IS REAL
C
 154   ISTR=IDENT(1,IL)
       IEND=MIN0(IDENT(2,IL),IDENT(2,ILD))
       IF(ISRT.EQ.-1) GO TO 960
       ITP=IDENT(4,IL)
       IF(ITP.EQ.1) GO TO 940
C
C    REAL TO REAL
C
C    INTEGER TO REAL
C
       IDENT(4,ILD)=0
       IF(ITP.EQ.2) IDENT(4,ILD)=2
       N=0
 160   DO 156 ISEC=ISTR,IEND
       CALL POSITN(IL,ISEC)
       READ(IL,END=158) HEAD
       N=N+1
       CALL PUT(ID,HEAD,N,1,IFLG)
       IF(IFLG.NE.0) GO TO 158
 156   CONTINUE
       IDENT(1,ILD)=1
       GO TO 2
 158   IDENT(1,ILD)=-1
       GO TO 950
```

```
C
C   REAL TO COMPLEX MOVE OR COMPLEX TO COMPLEX)
C
 170  ISTR=IDENT(1, IL)
      IEND=IDENT(2, IL)
      ITP=IDENT(4, IL)
      IF(ITP. EQ. 1) GO TO 160
C
C   REAL TO COMPLEX
C
      DO 172 I=1, 128
 172  RBUF(I)=0. 0
      N=-1
      DO 175 ISEC=ISTR, IEND
      CALL POSITN(IL, ISEC)
      READ(IL, END=176) RBUF2
      DO 174 I=1, 64
      J=I*2-1
 174  RBUF(J)=RBUF2(I)
      N=N+2
      CALL PUT(ID, RBUF, N, 2, IFLG)
      IF(IFLG. NE. 0) GO TO 176
 175  CONTINUE
      IDENT(1, ILD)=1
      GO TO 2
 176  IDENT(1, ILD)=-1
      GO TO 950
C
C   REQUEST PROCESSOR
C
 200  IF(IAL. EQ. 1) GO TO 991
      IF(IPOINT. EQ. 20) GO TO 970
      CALL DECODE(MYCOMA, IMAGE, 1, 1, R, ID, IFLG)
      IF(IFLG. EQ. 1) GO TO 900
      IF(IFLG. NE. 0) GO TO 910
      CALL DECODE(MYCOMA, IMAGE, 2, -1, R, NUM, IFLG)
      IF(IFLG. EQ. -1) NUM=47
      IF(IFLG. GT. 1) GO TO 910
      IF(IFLG. EQ. 1) GO TO 902
      DO 205 I=1, IPOINT
      IF(IDENT(5, I). EQ. ID) GO TO 992
 205  CONTINUE
      IPOINT=IPOINT+1
      IDENT(5, IPOINT)=ID
      IDENT(1, IPOINT)=-1
      IF(IABS(NUM). GT. 32000) INUM=32000
      INUM=IABS(NUM)
      IDENT(2, IPOINT)=INUM+1
      IDENT(4, IPOINT)=0
      IF(NUM. LT. 0) IDENT(4, IPOINT)=1
C
C   SET START CYLINDER
```

```
C
      N=0
      DO 210 I=10, IPOINT
      IF(I. EQ. IPOINT) IDENT(3, IPOINT)=N
 210  N=N+IDENT(2, IPOINT)+2
      GO TO 2
C
C
C   KILL PROCESSOR
C
 250  IPOINT=9
      IF(IAL. EQ. 0) GO TO 2
      CALL SVC4('DE VSTOR ')
      IAL=0
      GO TO 2
C
C   GO STATEMENT
C
 300  RETURN
C
C   OUTPUT STATEMENT
C
 350  WRITE(ILU, 3111)
 3111 FORMAT('OUTPUT NOT FUNCTIONAL')
      GO TO 2
C
C   ALLOCATE
C
 400  IF(IAL. NE. 0) GO TO 2
      IF(IPOINT. EQ. 9) GO TO 980
C
C   GET THE NUMBER OF CYLINDERS TO ALLOCATE AND DECODE
C
      NUMS=IDENT(3, IPOINT)+IDENT(2, IPOINT)+2
      NUMC=NUMS/48+1
      CALL ALLOCT(NUMC, IFLG)
      IF(IFLG. NE. 0) GO TO 990
      IAL=1
      GO TO 2
C
C   PAUSE PROCESSOR
C
 450  PAUSE
      CONTINUE
      GO TO 2
      GO TO 2
C
C   REQUEST ROUTINE PROCESSOR
C
 700  DO 710 I=1, 64
 710  RBUF(I)=0. 0
      N=0
```

```
      DO 750 I=1,10
      IF(MYCOMA(I).EQ.0) GO TO 750
      CALL DECODE(MYCOMA,IMAGE,I,1,R,ABUF(I),IFLG)
      IF(IFLG.EQ.2) CALL DECODE(MYCOMA,IMAGE,I,0,ABUF(I),IV,IFLG)
      IF(IFLG.EQ.-1) ABUF(I)=0.0
      IF(IFLG.EQ.1) GO TO 900
      N=N+1
 750  CONTINUE
      N=N+4
      IF(N.EQ.0) N=1
      N=-N
      IF(IDECF.NE.0) N=-N
C
C   FIND THE LENGTH OF THE NAME PACK IT
C
      IS=0
 760  IS=IS+1
      IF(IMAGE(IS).EQ.IBLNK) GO TO 760
      ILEN=MYCOMA(1)-IS
      IF(ILEN.GT.6) GO TO 900
      CALL PACKN(IMAGE(IS),ILEN,NAME)
      CALL SYSTEM(NAME,ABUF,N,IFLG)
      IDECF=0
      IF(IFLG.EQ.1) WRITE(ILU,1220)
      IF(IFLG.EQ.2) WRITE(ILU,1230)
      IF(IFLG.EQ.0) WRITE(ILU,1313) NAME
      GO TO 2
C
C   DISPLAY COMMAND - DISPLAYS INFO FROM FILE HEADERS
C
 800  IR=0
      CALL DECODE(MYCOMA,IMAGE,1,-1,R,ID,IFLG)
      IF(IFLG.EQ.2)CALL DECODE(MYCOMA,IMAGE,1,1,R,ID,IFLG)
      IF(IFLG.NE.0) GO TO 910
      IF(ID.LE.10.AND.ID.GE.1) GO TO 810
      CALL GET(ID,HEAD,0,1,IFLG)
      IF(IFLG.EQ.-1) GO TO 960
      IF(IFLG.NE.0) GO TO 910
      IR=1
      GO TO 820
 810  IF(IDENT(1,ID).EQ.-1) GO TO 960
      CALL POSITN(ID,0)
      READ(ID) HEAD
C
C   WRITE THE INFO
C
 820  WRITE(IPRLU,5000) TITLE
      IF(IDIN.EQ.0) WRITE(IPRLU,5100) DINV,STIME
      IF(IDIN.EQ.1)WRITE(IPRLU,5110) DINV,STIME
      IF(ITYPE.EQ.0) WRITE(IPRLU,5120) NUMSEC
      IF(ITYPE.EQ.1) WRITE(IPRLU,5130) NUMSEC
      IF(ITYPE.EQ.2) WRITE(IPRLU,5140) NUMSEC
```

```
      IF(IR.NE.0) GO TO 2
      INS=IDENT(2,ID)-IDENT(1,ID)+1
      WRITE(IPRLU,5150) IDENT(1,ID),IDENT(2,ID),INS
      GO TO 2
C
C   END PROCESSOR
C
 850  CALL FINISH
C
 900  WRITE(ILU,1100)
      GO TO 2
 910  WRITE(ILU,1110)
      GO TO 2
 920  WRITE(ILU,1120) ID
      GO TO 2
 930  WRITE(ILU,1130)
      GO TO 2
 940  WRITE(ILU,1140)
      GO TO 2
 950  WRITE(ILU,1150)
      GO TO 2
 960  WRITE(ILU,1160)
      GO TO 2
 970  WRITE(ILU,1170)
      GO TO 2
 980  WRITE(ILU,1180)
      GO TO 2
 990  WRITE(ILU,1190)
      GO TO 2
 991  WRITE(ILU,1191)
      GO TO 2
 992  WRITE(ILU,1192)
      GO TO 2
 999  STOP
C
C   FORMAT STATEMENTS
C
 1000 FORMAT(80A1)
 1100 FORMAT('ERR  ARG  LENGTH')
 1110 FORMAT('INPUT ERR ')
 1120 FORMAT('UNREQUESTED ID = ',A2)
 1130 FORMAT('MOVE ILL. BEFORE ALLOCATE')
 1140 FORMAT('MOVE TO REAL FILE ILL.')
 1150 FORMAT('EOF ENCOUNTERED ON MOVE - REDEFINE INPUT')
 1160 FORMAT('INPUT FILE UNDEFINED')
 1170 FORMAT('ERR: MAX. TEMP. FILE')
 1180 FORMAT('NOTHING TO ALLOCATE')
 1190 FORMAT('TO MUCH SPACE REQUESTED - ALLOCATE FAILED')
 1191 FORMAT('REQUEST ILL. AFTER ALLOCATE')
 1192 FORMAT('DUPLICATE TEMP. FILE NAME - REQUEST DENIED')
 1313 FORMAT('?')
 1220 FORMAT('LIST OVERFLOW')
```

```
1230 FORMAT('SYSTEM ERROR')
1313 FORMAT('ROUTINE = ',3A2,' HAS BEEN PLACED IN THE LIST')
5000 FORMAT(1H0/1H0,'FILE TITLE: ',25A2)
5100 FORMAT(1X,'DISCRETIZATION INTERVAL: ',F10.6,
    1  ' MSEC',/1X,'STARTING AT',F10.6,' SECONDS')
5110 FORMAT(1X,'DISCRETIZATION INTERVAL: ',F10.6,
    1  ' HERTZ',/1X,'STARTING AT',F10.6,' HERTZ')
5120 FORMAT(1X,'REAL*4 FILE CONTAINING',I6,' SECTORS'/
    1  ' @ 64 WORDS PER SECTOR')
5130 FORMAT(1X,'COMPLEX FILE CONTAINING',I6,' SECTORS'/
    2  ' @ 32 WORDS PER SECTOR')
5140 FORMAT(1X,'INTEGER*2 FILE CONTAINING',I6,' SECTORS'/
    3  ' @ 128 WORDS PER SECTOR')
5150 FORMAT(1X,'DEFINED SECTORS: ',I5,' TO ',I5,' FOR A TOTAL',
    1  ' OF ',I6,' SECTORS')
     END
```

```
$ASSM
DECODE PROG DECODE ROUTINE FOR DSAIN
 SCRAT
$FORT
      SUBROUTINE DECODE(MYCOMA,IMAGE,ARGNO,TYPE,RVALUE,IVALUE,IFLG)
      IMPLICIT INTEGER*2 (I-N)
C
C     SUBROUTINE DECODE IS AN INPUT HANDLER ROUTINE WHICH
C   DECODES THE ARGUMENTS OF A COMMAND.   THE ROUTINE WILL
C   DECODE A REAL, INTEGER, OR CHARACTER DATA VALUE.
C   THE ROUTINE USES SUBROUTINE CHECK TO DETERMINE CHARACTER
C   DATA.
C
C
C     MYCOMA = ARRAY CONTAINING THE COMMA LOCATIONS IN THE INPUT
C              STRING
C     IMAGE  = ARRAY CONTAING THE ARGUMENTS AS CHARACTER DATA
C     ARGNO  = THE ARGUMENT NUMBER TO BE RETURNED FROM THE IMAGE
C     TYPE   = TYPE OF ARGUMENT EXPECTED
C              -1 = INTEGER*2 VALUE
C               0 = REAL*4 VALUE
C              +1 = CAHARACTER VALUE (2 CHAR. MAX LEN)
C     RVALUE = THE REAL VARIABLE RETURN LOCATION
C     IVALUE = THE INTEGER VALUE RETURN LOCATION
C              ALSO USED TO RETURN CHARACTERS
C     IFLG   = ERROR FLAG
C              -1 = NO ARGUMENT FOUND
C               0 = ALL OK.
C              +1 = ARGUMENT LENGTH GREATER THEN TEN CHARACTERS
C               2 = NUMBER REQUESTED AND CHAR FOUND (A-Z)
C                   OR CHAR REQUESTED AND NUMBER FOUND
C
      INTEGER*2 TYPE,ARGNO,MYCOMA(1),IMAGE(1),IFORM(4),IAFORM(4),
     1 IDIGIT(10),MES(66)
      DATA IFORM/'(G',' ','.0',') '/
      DATA IAFORM/'( ',' ','A1',') '/
      DATA IDIGIT/' 1',' 2',' 3',' 4',' 5',' 6',' 7',' 8',' 9','10'/
C
      ILOC=MYCOMA(ARGNO)
      IF(ILOC.EQ.0) GO TO 100
      ILOC=ILOC+1
      ILEN=MYCOMA(ARGNO+1)
      IF(ILEN.EQ.0) GO TO 100
      ILEN=ILEN-ILOC
      IF(ILEN.GT.10) GO TO 200
      IF(ILEN.EQ.0) GO TO 100
      CALL CHECK(IMAGE(ILOC),ISET)
      IF(TYPE.LE.0.AND.ISET.EQ.0) GO TO 400
      IF(TYPE.GT.0.AND.ISET.EQ.1) GO TO 400
      IEND=ILOC+ILEN-1
      IAFORM(2)=IDIGIT(ILEN)
      ENCODE(MES,IAFORM)(IMAGE(L),L=ILOC,IEND)
```

```
      IF(TYPE) 10,50,80
C
C   INTEGER VALUE
C
 10   IFORM(2)=IDIGIT(ILEN)
      DECODE(MES,IFORM) IVALUE
      GO TO 300
C
C   REAL VALUE
C
 50   IFORM(2)=IDIGIT(ILEN)
      DECODE(MES,IFORM) RVALUE
      GO TO 300
C
C   CHARACTER DATA
C
 80   IF(ILEN.GT.2) GO TO 200
      IVALUE=MES(1)
 300  IFLG=0
      RETURN
 100  IFLG=-1
      RETURN
 200  IFLG=1
      RETURN
 400  IFLG=2
      RETURN
      END
```

```
      SUBROUTINE CHECK(ICHAR,IFLG)
C
C     THIS SUBROUTINE CHECKS THF CHARACTER IN ICHAR
C   TO DETERMINE IF IT IS A ALPHA CAHRACTER BETWEEN
C   'A' AND 'Z'.    IF IT IS A CHARACTER IN THAT RANGE
C   THE FLAG IFLG IS SET TO ZERO.  OTHERWISE IFLG IS RETURNED
C   AS ONE.
C
      IMPLICIT INTEGER*2 (I-N)
      INTEGER*2 A,Z,ITEST
      DATA ITEST/0/
      DATA A,Z/X'0041',X'005A'/
$ASSM
*
*   STRIP THE PARITY AND MAKE LOWER CASE UPPER CASE
*
      LH 1,ICHAR
      LH 2,0(1)
      EXBR 2,2
      NHI 2,X'005F'
      STH 2,ITEST
$FORT
      IFLG=0
      IF(ITEST.LT.A.OR.ITEST.GT.Z) IFLG=1
      RETURN
      END
```

```
$ASSM
PACKN PROG ROUTINE TO PACK OVERLAY NAMES
 SCRAT
$FORT
      SUBROUTINE PACKN(IMAGE,ILEN,NAME)
      IMPLICIT INTEGER*2 (I-N)
C
C   SUBROUTINE PACKN IS USED TO PACK A SIX CHARACTER NAME
C   STORED IN AN ARRAY IN 'A1' FORMAT.  THE PACKED NAME
C   GOES INTO AN ARRAY OF LENGTH 3 HALFWORDS.  THE NAME
C   WILL BE A FULL SIX CHARACTERS PADDED RIGHT WITH BLANKS.
C
C     IMAGE - CONTAINS THE NAME TO BE PACKED, ONE CHARACTER
C             PER HALFWORD.  (INTEGER ARRAY).
C     ILEN - THE NUMBER OF CHARACTERS TO BE PACKED
C     NAME - INTEGER ARRAY OF LENGTH 3 WHERE THE PACKED NAME
C            IS TO BE PLACED.
C
      INTEGER*2 IDENT,HEAD
C
      COMMON/SIGNAL/IDENT(5,20),IAL,IPOINT,ILU,IDECF,HEAD(128),
     1  ADUF(64),NARG,IPRLU,IPLTLU
C
      INTEGER*2 IMAGE(1),NAME(3)
C
      DATA IBLNK/'  '/
      IF(ILEN.GT.6.OR.ILEN.LE.0) GO TO 100
      IL2=ILEN+1
$ASSM
      LH     1,IMAGE             GET THE IMAGE ADDRESS
      LH     2,NAME              GET THE NAME ADDRESS.
$FORT
      DO 10 I=1,6
$ASSM
      LB     3,0(1)              GET THE CHARACTER.
      CLH    5,IL2               ALL CHARACTERS DONE ?
      BL     NEXT
      LHI    3,X'2020'           SET TO BLANK
NEXT  STB    3,0(2)              STORE THE CHARACTER
      AIS    1,2                 INCREMENT R1.
      AIS    2,1                 INCREMENT THE NAME POINTER
$FORT
 10   CONTINUE
      RETURN
 100  NAME(1)=IBLNK
      NAME(2)=IBLNK
      NAME(3)=IBLNK
      RETURN
      END
```

```
$ASSM
ALLOCT PROG ROUTINE FOR ALLOCATING TEMP STORAGE ON DISK
 SCRAT
 SQUEZ
$FORT
      SUBROUTINE ALLOCT(NUMC,IFLG)
      IMPLICIT INTEGER*2 (I-N)
C
C      SUBROUTINE ALLOCT ALLOCATES A DOS DISK FILE AND
C   ASSIGNS IT AN ATTRIBUTE (DIRECT PHYSICAL).
C
C       NUMC = THE NUMBER OF CYLINDERS TO ALLOCATED TO THE FILE.
C
C       IFLG = ERROR FLAG; (IFLG=0, NO ERROR) (IFLG=1, NO ALLOCATE).
C
      INTEGER*2 IALOC(10),D1,D2,D3
      DATA IALOC/'AL',' V','ST','OR',',F',',  ','  ',',2','56',X'200D'/
      D1=NUMC/100
      IF(D1.GT.3) GO TO 20
      D2 = NUMC/10-D1*10
      D3 = NUMC-D1*100-D2*10
$ASSM
      LH     1,D1            GET THE FIRST DIGIT
      OHI    1,X'30'         MAKE IT A CHARACTER.
      STB    1,IALOC+11      AND STORE IT
      LH     1,D2
      OHI    1,X'30'
      STB    1,IALOC+12
      LH     1,D3
      OHI    1,X'30'
      STB    1,IALOC+13
$FORT
      CALL SVC4(IALOC)
      CALL SVC4('AT VSTOR,0020 ')
      IFLG=0
      RETURN
 20   IFLG=1
      RETURN
      END
```

```
      SUBROUTINE DSAMOD
      IMPLICIT INTEGER*2 (I-N)
C
C     THIS IS A SAMPLE OF THE LINKAGE ROUTINE USED WHEN ADDING
C   ROUTINES TO THE OVERLAY LIBRARY.  ID IS A CHARACTER
C   ARGUMENT WHICH IS TO BE PASSED TO THIS ROUTINE.
C
      INTEGER*2 IDENT, IHEAD, HEAD(128)
      DIMENSION BUFS(200)
C
      COMMON/SIGNAL/IDENT(5,20), IAL, IPOINT, ILU, IDECF, IHEAD(128),
     1  ABUF(64), NARG, IPRLU, IPLTLU
C
      EQUIVALENCE (ABUF(1), ID)
      EQUIVALENCE (HEAD(1), DISC), (HEAD(3), ITYPE), (HEAD(4), STIME),
     1  (HEAD(6), NUMSEC), (HEAD(7), IWT), (HEAD(8), IGF),
     2  (HEAD(9), TITLE(1)), (HEAD(34), YLABEL(1)),
     3  (HEAD(59), XLABEL(1)), (HEAD(120), ITP)
C
C   GET THE HEADER
C
      CALL GET(ID, HEAD, 0, 1, IFLG)
      IF(IFLG)10, 20, 30
 10   WRITE(ILU, 300)
 300  FORMAT('TEMP FILE IS UNDEFINED OR EMPTY',
     1  ' - REENTER COMMAND AT PROMPT')
      IDECF=1
      RETURN
 30   WRITE(ILU, 100)
100   FORMAT('PLOT ROUTINE DEV. ERR.  - REENTER COMMAND AT PROMPT')
      IDECF=1
      RETURN
 20   CALL PLOTS(BUFS, 800, 0)
      IF(ITYPE.EQ.1.AND.IWT.EQ.0) ITP=1
      IF(ITYPE.EQ.0) DISC=DISC*.001
      CALL SIGPLT(ID, NUMSEC, ITYPE, STIME, DISC, TITLE, XLABEL, YLABEL)
      RETURN
      END
```

VITA

John Edward Perrault, Jr.

Candidate for the Degree of

Master of Science

Thesis: DEVELOPMENT OF A DIGITAL SIGNAL ANALYSIS SYSTEM FOR MINICOMPUTERS

Major Field: Mechanical Engineering

Biographical:

Personal Data: Born in Tulsa, Oklahoma, March 10, 1953, the son of Mr. and Mrs. John E. Perrault.

Education: Graduated from Bishop Kelley High School, Tulsa, Oklahoma, in May, 1971; received the Bachelor of Science in Mechanical Engineering degree, from the University of Tulsa, Tulsa, Oklahoma, in May, 1975; completed the requirements for the Degree of Master of Science in May, 1977.

Professional Experience: Engineer, Marvel Photo Company, Tulsa, Oklahoma, from 1970 to 1975; Research Assistant, Oklahoma State University, 1976 to present.

Professional Organizations: Member of American Society of Mechanical Engineers, Institute of Electrical and Electronics Engineers, Tau Beta Pi, and Eta Kappa Nu.