A COMPUTER-AIDED METHOD FOR SLEEP SIGNAL CLASSIFICATION

by

James A. Reimer

A thesis
presented to the University of Manitoba
in partial fulfillment of the
requirements for the degree of
Master of Science
in
Electrical Engineering

Winnipeg, Manitoba, 1982

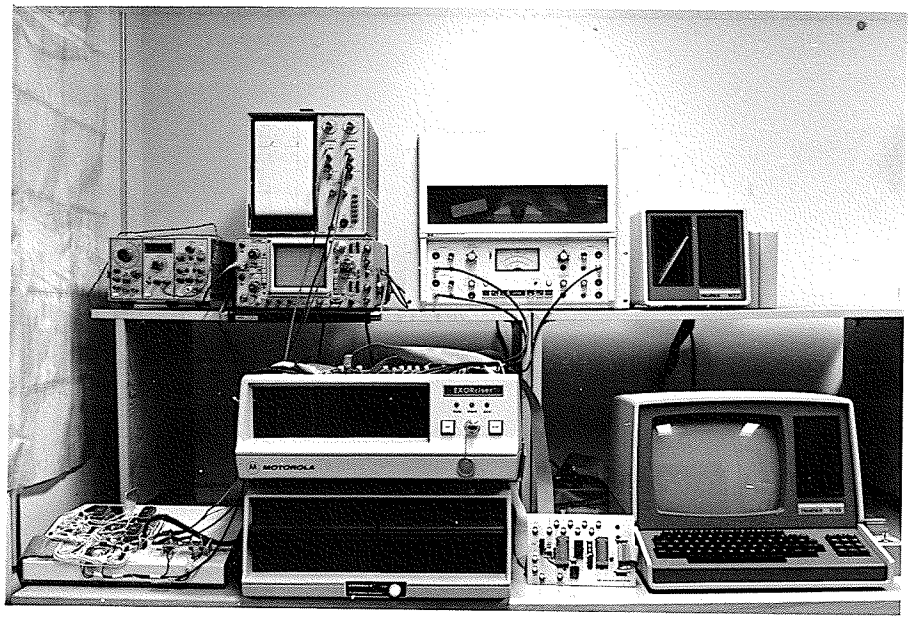
A COMPUTER-AIDED METHOD FOR SLEEP SIGNAL CLASSIFICATION

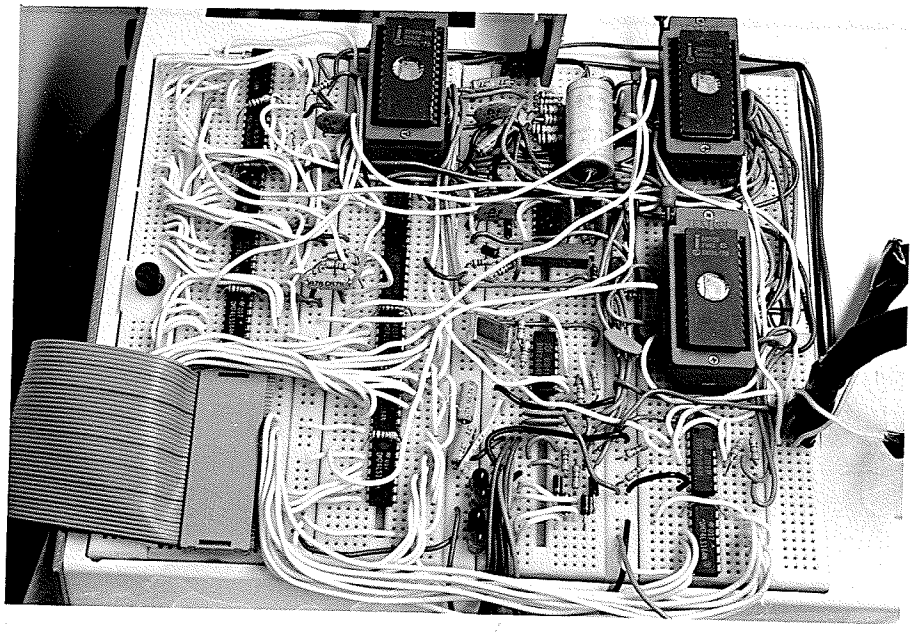
BY

JAMES A. REIMER

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE


EXORciser
MOTOROLA



I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

James A. Reimer

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

James A. Reimer

The University of Manitoba requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

ABSTRACT

This study considered the development of an aid to sleep classification system, and discussed a method which used the clinician as feedback to the classification process. This scheme allowed the the clinician to effect the heuristic process of sleep classification, while allowing the computer to reduce the effort expended in the classification of like segments.

The use of inexpensive microprocessing hardware for sleep signal spectral estimation was considered, and a scheme using digital filters implemented with Intel 2920s was developed. The estimation subsystem was interfaced with a Motorola EXORcisor microcomputer for real-time data collection, and the collected estimates transferred to an Amdahl mainframe computer for post-analysis.

Preliminary studies showed a gross correlation between the extracted spectral estimates and the manually classified sleep record. The spectral estimates, as sampled once a second, were found to be non-stationary within manually classified sleep stage segments. The use of spectral averaging was considered, and found to simplify the correlation between the spectral estimates and the manual classification for normal sleep. For abnormal sleep, however, it was found that such averaging could blur the correlation.

ACKNOWLEDGEMENTS

I would like to thank Dr. Witold Kinsner for providing guidance and encouragement, and for acting as the head of my thesis advisory committee. Dr. Kinsner's dedication, while heading hectic and demanding activities in establishing the Industrial Applications of Microelectronics Centre, Inc. (IAMC), is sincerly appreciated.

I would also like to thank the other members of the advisory committee: Dr. Meir Kryger, Mr. Brian Trenholm, and Mr. Peter West from St. Boniface General Hospital; and Mr. Stan Hodge from Manitoba Hydro. The committee provided an invaluable resource of experience and expertise. I am indebted to many colleagues for their help, particularly that of Messrs. Bill Joll, Victor Shkawrytko, Greg Smith, and Andy Weirich. Thanks also to Mr. John Warga and Manitoba Data Services for assistance in printing the program listings.

The financial support of this research provided by the IAMC, Manitoba Hydro, St. Boniface General Hospital, and the Department of Electrical Engineering is also gratefully acknowledged.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter I

INTRODUCTION

1.1 BACKGROUND

The analysis and evaluation of human sleep has become a valuable aid in the diagnosis and treatment of sleep related disorders, and is widely used today. The United States Food and Drug Administration, for example, requires that sleep studies be used in the evaluation of all new hypnotics and sedatives [1]. The evaluation of sleep is also very useful in the investigation and assessment of sleep related disorders. Insomnia, narcolepsy, chronic hypersomnia, obstructive sleep apnea, sudden infant death syndrome, and many other disorders have been studied with sleep evaluation [1,2,3]. It has been speculated that with increasing understanding and related technical advances, the evaluation of sleep can become a standard clinical procedure [1].

Although sleep has been investigated scientifically for more than 100 years, most activity has occured in the past 25 years. The evaluation of sleep state is based primarily on the electroencephalogram (EEG), the electrooculogram (EOG), and the electromyogram (EMG). There have been many studies which have examined these signals as they occur during sleep, and these studies have led to a set of commonly accepted standards and rules for classifying sleep [1,4]. These rules define the sleep stages numbered 1 through 4, as well as the rapid eye movement (REM) and awake sleep stages. Traditionally, the sleep record is divided

into fixed (time) length segments or epochs, and the record is staged or scored on an epoch-by-epoch basis. This scoring is made primarily on the basis of the spectral distribution of the waveforms within each epoch.

In one of the largest sleep studies conducted, the rules for scoring sleep were exercised through the classification of over 6000 recordings of sleep periods for several hundreds of patients [1]. These and other studies have revealed that the signals observed during sleep, while characteristic to the rules of scoring, are quite variable with each patient [5]. Even the signals observed within one sleep session can vary significantly. The variability of these signals can arise from a variety of physiological differences as a function of patient age and state, as well as monitoring problems in electrode contact and noise.

The value of sleep evaluation as a clinical technique is offset by the labour-intensity and tediousness of the manual scoring of sleep signals recorded on paper. This labour-intensity represents a significant barrier in the investigation of sleep related disorders. The classification of a severely disturbed (8 hour) sleep record can require up to one man-week of effort. There have been many attempts to automate the scoring of sleep signal records [6,7,8,9,10,11,12,13,14]. Such automation is desirable to reduce the labour-intensity of the activity as well as to reduce the variability of interpretation from clinician to clinician [15]. The automation of the sleep signal classification process is difficult, however, and past attempts have achieved only limited success and marginal acceptance [15].

Most attempts at automated sleep signal classification have focused on the use of large main-frame computers and analog signal processing

equipment in the activity of classifying normal sleep. A number of factors have contributed to the limited success of these attempts. Classification of sleep is based primarily on the spectral distribution observed in the sleep signals [4]. This implies that the frequency spectrum of these signals must be estimated. In the past, such estimation was feasible only with delicate and expensive analog hardware or powerful main-frame computers. The large costs associated with such schemes have been a significant factor limiting their adoption. These costs are dropping, however, as recent developments in microelectronics have made inexpensive digital implementations of spectral estimation hardware possible.

A more serious problem has been that of developing a system which is capable of accommodating the high variability which is present in signals observed during sleep. This problem is similar to that of speech recognition. In both problems, it is difficult to construct a system which is able to classify observed signals at a given moment of time for a broad class of subjects. The clinician's evaluation of a sleep record is based on a wide range of experience and understanding of signal considered. Often the clinician must scan back and forth through a sleep record in order to locate representative segments of each sleep stage. These representative segments are used as reference points in the relational classification of the record. When a particular segment of a patient's record is classified, it is desired that 'similar' segments be classified in the same way throughout the record. The term 'similar' implies similarity in the most general sense; that is, a generalized metric considering the syntactic or context relations of the segment. It

has been observed that the clinician frequently deviates from the rigidly defined scoring rules for reasonable but poorly defined reasons [15]. The heuristic methods used by clinicians are difficult to describe. Most attempts at automatic sleep staging, however, have pursued a search for a simple, causal model for sleep classification. In other words, they have attempted to classify the sleep record with no perception of the record as a whole, or input from the clinician [7]. This implies that these studies have attempted to have the computer replicate the activity of the clinician with less information than the clinician requires. It can be argued that with the current state of technology and understanding of sleep, it is not possible to replicate the entire activity of the clinician [7].

Another significant limitation of these studies is that they have focused on the classification of normal sleep. Normal, healthy sleep has the characteristic that sleep stage transitions typically occur over extended time intervals. As a result, most studies have based their classification scheme on a fixed-length epoch rule. The entire night's record is divided into fixed length periods with only one decision or classification made over each interval. Epoch lengths of 30 seconds to 2 minutes have commonly been used. It has been found that patients with sleep disorders can demonstrate cyclic disruptions or periodicities in their sleep patterns. These changes can occur as frequently as every 15 seconds. For such patients, the application of a fixed-length epoch rule may prohibit the identification of brief sleep segments. An investigator may wish to relate sleep to some other physiological variable (respiration, for example). If the resolution of the sleep classification is

much less than the time constant associated with the other physiological variable, the correlation of the two functions may be lost. While past studies have achieved controlled successes on fixed healthy patient groups, the direct applicability of the work to a broad class of patients is questionable.

1.2 OBJECTIVES OF STUDY

The approach to sleep signal classification considered in this study is the use of the computer to assist the clinician in the process of classifying sleep. As in past studies, the computer will be used to extract features from the signals which are presently used to stage sleep. Then, however, the clinician will be used to identify examplary segments to the computer for like classification. In this way, the experience of the clinician is used to effect the heuristic process of sleep signal classification, and the strength of the computer is used to reduce the effort expended in the classification of repeated segments.

The method under study represents a significant departure from past attempts in the use of computers for sleep classification in that the experience of the clinician is used as feedback to the system. Since it is desired that like record segments be classified in a like manner, and since sleep patterns tend to repeat themselves in cyclic patterns, the use of such a computer-aided system should provide a reliable method which reduces the labour-intensity of classifying normal and abnormal sleep records. The variability of the sleep record will be accommodated, as only like segments will be classified alike. If there is a high degree of variability within a given patient's record, the clinician will

have to identify more examplary segments in order to classify the record. If a patient presents a record which has no repetitious segment and high variability, the system will effectively reduce to the manual classification method commonly used now. Areas of the record which the system has been unable to classify with the provided definitions can be related to the clinician for further clarification.

Beyond the specific application of sleep signal classification, this method represents a powerful approach for a wide variety of problems involving extended duration signal characterization and feature extraction.

The objectives of this study are to:

1. Study the digital signal processing required to effect sleep signal spectral estimation;
2. Evaluate the use of recent low-cost microprocessor hardware for the task of such estimation;
3. Develop an interface between such estimation hardware and an existing microcomputer;
4. Study the systematic procedures for evaluating an aid to sleep classification system; and
5. Conduct a preliminary evaluation of the correlation between the characterizations of the sleep signals and the manually classified sleep record.

Chapter II

SLEEP CLASSIFICATION METHODS

2.1 CHARACTERISTIC FEATURES AND RULES

Sleep signal classification rules commonly used today are based primarily on the scoring criteria established by Dement and Kleitman in 1957 [16]. These rules were reaffirmed by the U.S. Department of Health in the Rechtschaffen and Kales (1968) standardization manual for sleep scoring [4]. These studies reflect the firmly established fact that sleep is not a homogeneous state, and that sleep stages follow fairly orderly cyclic patterns [4]. Sleep can be subdivided into two broad classes of REM (rapid eye movement) and non-REM sleep. Non-REM sleep is further subdivided into sleep stages numbered 1 through 4, or as awake sleep. These definitions are based on characteristics observed in the electroencephalogram (EEG), the electromyogram (EMG) and the electrooculogram (EOG). The EEG signal electrodes are commonly applied at positions C4/A1 and or C3/A2 according to the international 10-20 placement system [17]. It is often desirable to record 2 channels of EEG, having one as a backup in case a pair of electrodes cease to function or exhibit excessive amounts of artifact. The EOG is derived from electrodes placed around the eyes, and the EMG is often taken from an electrode placed under the chin. These signals are generally recorded with a polygraphic recorder with a minimum paper speed of 10 mm/second, and a minimum sensitivity of approximately 5 microvolts per mm. The classifi-

cation of sleep is based primarily on the spectral distribution of the EEG within established bands as a function of time [4]. The spectral bands which are considered most important are the low (0.25 to 2.0 Hz) and mixed (2.0 to 7.0 Hz) frequency bands. The appearance of sleep spindles (rhythmic bursts of at least 0.5 second duration in the 12 to 14 Hz range) and K complexes (well delineated negative sharp waves which are immediately followed by a positive component [4]) are also considered significant. The EOG is commonly considered only within the band of 2.0 to 10.0 Hz, and the EMG is considered primarily from the perspective of average signal power.

Just after a patient has fallen asleep, the EEG activity is quite similar to that present when the patient is awake. This period represents the highest neurological state during sleep, and is referred to as awake sleep. The awake sleep stage has the EEG characteristic of predominately high frequency activity with some low amplitude, mixed frequency activity. Most of a night is spent in non-REM sleep, with some intermixed periods of REM sleep. During REM sleep, the EEG activity reflects a fairly 'light' level of sleep (stage 1), yet there is only a low amplitude EMG in conjunction with episodic rapid eye movements (REMs). REM appears to correspond with a 'deep' level of sleep in that there is little EMG activity, while the EEG activity corresponds with that of light sleep. As a result, REM sleep is sometimes referred to as paradoxical sleep, and is commonly thought to be associated with dreaming. Sleep stages 1 and 2 can be grouped as mixed frequency sleep, and represent 'light' sleep. Sleep stages 3 and 4 can be grouped as slow wave sleep, and represent 'deep' sleep. Stage 1 is defined as an interval of rela-

tively low amplitude, mixed frequency EEG with absolute absence of REMs, K complexes, or spindles. Stage 2 is defined as relatively low amplitude, mixed frequency EEG background with the occurence of sleep spindles and/or K complexes. If more than 3 minutes of stage 2 sleep pass without a sleep spindle or K complex, the classification defaults back to stage 1. Stage 3 is defined by an EEG segment in which at least 20% but not more than 50% of the epoch consists of predominately low frequency activity. Stage 4 is defined by an EEG segment in which more than 50% of the epoch consists of primarily slow frequency activity. It is often difficult to differentiate between stages 3 and 4, and they are, therefore, sometimes classified together as slow wave sleep.

Normal sleep pattern distributions vary dependent on factors such as as patient age and sex [1]. Sleep patterns can also be altered by a variety of clinical disorders. An example of a disorder which can alter sleep is sleep apnea syndrome [2]. Sleep apnea is a disruption of respiration where breathing is diminished or stops for periods exceeding 10 seconds. During such events, the oxygen content of the patient's blood may drop. Events of this nature are typically terminated with an arousal, although the patient is generally not aware of the event. Frequent disruptions of sleep may lead to variety of symptoms or disorders, and can have a significant impact on the patient [2].

2.2 MANUAL CLASSIFICATION METHODS

The most commonly accepted method for sleep signal classification is a manual review of the polygraphic recording. Traditionally, the entire night's record is divided into fixed-length sections or epochs. Epoch lengths of 30 seconds to 2 minutes have commonly been used. Classification is based on the visually observed waveforms averaged over each epoch interval. A single classification is made for each epoch, and the record is classified on an epoch-by-epoch basis. The clinician will often scan through the entire record in order to 'calibrate' for the scoring, and must often refer to other portions of the record in order to classify a particular epoch. For the classification of abnormal sleep, a variable length epoch method has been used [18]. With this scheme, sleep stage transitions are identified wherever they may occur (with a minimum epoch length of 15 seconds). This method allows a more accurate tracking of disturbed sleep patterns. The classification data is then commonly entered into a computer for further analysis. The classification data has, for example, been correlated with respiration variables concurrently recorded during sleep [19].

2.3 AUTOMATED CLASSIFICATION ATTEMPTS

There have been many attempts to automate the classification of sleep signal records. These attempts have used a variety of signal processing techniques, and have considered many different decision algorithm methods. In general, these attempts have focused on trying to achieve total automation of the sleep classification process. In a review of attempts to automate sleep classification [15], it was noted that most studies

that have achieved some satisfactory results have had much in common. These systems have, in general, used REM detectors to discriminate stage 1 from REM, and have tried to establish some algorithm to mimic the heuristic decision making of a clinician on the basis of some wave features extracted from the EEG. The EMG, although valuable according to the Rechtschaffen standards, has often not been considered as it has been found to be unreliable [15].

Direct amplitude analysis, spectral analysis, Walsh expansions, and period analysis have all been considered in the problem of sleep classification [20,21,22]. Although different studies have used widely varied signal processing techniques and implementations, the more advanced attempts have achieved approximately the same level of controlled success. Frequently used methods for the wave feature extraction have been bandpass filtering, fast Fourier transforms (FFTs), matched filtering or correlation detection, zero crossing and pitch period detection, or the use of hybrid wave-form detection circuitry. The most frequently used technique for the wave feature extraction has been bandpass filtering. The filters have, in the past, commonly been implemented with analog circuitry. The low frequency nature of the passbands has made such equipment difficult and costly to implement. Many other studies have recorded the entire sleep record and have conducted post-analysis FFT studies. There has been much study of the detection of sleep spindles, K complexes, and rapid eye movements (REMs) [23,24,25,26,27,28,29,30,31]. K-complexes are frequently not considered as they are poorly defined and difficult to detect in both manual and computer-aided analysis. REMs are very valuable in distinguishing be-

tween stage 1 and REM, and sleep spindles are also generally considered. Overall, the primary factor considered in sleep classification is the amplitude of the EEG within spectral bands as distributed over time.

Independent of the wave features extracted, the first study to accurately classify all stages of sleep was a scheme which incorporated pattern recognition [15]. The early work with pattern recognition was criticized because it could not accommodate a patient group with a large variance in age without changing the recognition scheme. Also, these systems often consumed very large amounts of expensive computer time to effect the wave feature extraction and pattern recognition. Pattern recognition is a field which has received much study of late, however, and holds promise in its application to a number of areas including sleep classification [7,6,32]. Many of the other studies which achieved some success used decision tree algorithms to try to directly mimic the activity of the clinician. These studies were frustrated by the very heuristic nature of manual sleep classification, and as a result achieved success only within confined patient groups.

Chapter III

PROPOSED METHOD

The general method considered in this study is one of having the computer aid the clinician in the classification of sleep. The computer is used to extract the wave features which are considered significant in the classification of sleep, and the clinician identifies to the system examplary segments of the sleep stages. In this way, the experience of the clinician is used to effect the heuristic process of sleep classification, and the computer is used to reduce the effort expended in the classification of like segments. When a particular segment of a sleep record is classified, it is desired that like segments be classified in a like manner. Since sleep patterns tend to repeat themselves during the course of a night, a system which could identify repeated segments could reduce the labour-intensity of the activity. This is especially true for the classification of disturbed sleep in that, for patients with such disorders, the frequency of sleep stage transitions is greatly increased. In a more general sense, this method represents a powerful approach for a wide variety of problems involving extended duration signal characterization and classification.

The conjecture of this approach is supported by much of the work done on automated sleep classification. Gath and Bar-on [7] studied a system which extracted linear predictive coding (LPC) coefficients for the EEG, EOG, and EMG during sleep in order to characterize the entire nights ac-

tivity. The spectrum of the signals was then estimated on the basis of the LPC coefficients, and a scheme using fuzzy set theory was used to classify the sleep (fuzzy set theory is closely related to pattern recognition). This work reinforced the conjecture that the average spectrum presented in the EEG is 'somewhat' stationary during sleep. The term 'somewhat' in this case implies that the average spectrum is constant for some time, although varying over longer time intervals. In fact, this work suggested that the transitions in the average signal spectrums were quite distinct, and that an 8-hour recording could be reduced to about 3500 record segments with somewhat constant average spectral characteristics. Bourne [6] has extensively studied the use of syntactic pattern recognition for sleep classification, and has had promising results.

The primary reasons for considering a feature extraction/pattern recognition scheme are because the characteristics observed during sleep vary greatly between patients, and because the classification process effected by the clinician is so heuristic. This is not to criticize clinicians for their inability to describe completely the process which they effect. An analogy might be that an individual, on hearing two distinct pieces of music, is clearly able to distinguish between them although generally unable to say exactly why. What is implied is that the human brain is an extremely capable processing element which is able to conduct very heuristic decision making. Given the heuristic nature of the sleep classification process, and the fact that repeated attempts to establish decision-tree like structures to model the process have achieved only marginal success, a generalized pattern recognition scheme

which bases the classification on identified examplary segments seems like an appropriate avenue for consideration. The consideration of highly variable, disturbed sleep records further intensifies the need to move away from a fixed-model approach.

Some of the studies which have considered pattern recognition have been found to be very intensive in their computer usage. These costs have limited their use. The ever increasing complexity and dropping costs of microprocessing hardware provide strong motivation for their use in problems like sleep classification. Recent developments in microelectronics have led to microprocessors well suited for use in signal processing. The Intel 2920 is an example of such a microprocessor [33]. The use of such microprocessing hardware for the task of sleep classification may lead to the development of an inexpensive labour-saving aid. Therefore, this study has considered the use of inexpensive microprocessor hardware for sleep signal classification.

Chapter IV

SLEEP SIGNAL PROCESSING

4.1 INTRODUCTION TO DIGITAL SIGNAL PROCESSING

Analog signals are those which are continuous as a function of time. Digital signal processing is the general subject associated with sampled signals which are represented numerically. Sampling a signal and processing it numerically has a number of advantages over processing a signal with analog components. Numerical processing is frequently accomplished with a digital computer. This fact implies that there is a high degree of flexibility in the processing which can be accomplished by adjusting software algorithms. Achieving such flexibility with analog hardware is, in general, not economically feasible. When an analog circuit is specified, the components with which it is constructed are only approximations of the values desired. This implies that precise analog circuits require fine tuning after the components particular to the circuit have been installed. Also, after the components have been installed and fine tuned, the circuit function can drift as the analog components age and change in characteristics. This drift may require that the circuit be periodically fine tuned to reestablish its function. Independent of the approximation and drift problems, it may be difficult to fabricate the components required in an analog circuit. This problem is often true when working with low frequency signals. Also, whenever an analog signal is processed, there is almost always some noise added to

it. Systematic and expensive design and fabrication procedures can minimize (although not eliminate) these problems.

Digital signal processing does not solve the problems identified with analog processing, but rather replaces them with problems that can sometimes be more easily addressed. In general, digital components which are stable and drift free can be fabricated quite accurately and repeatably. Also, functions which may be difficult to implement with analog components are often straightforward with digital signal processing techniques. Once a signal is sampled, there is no additive electrical noise since the processing occurs through numerical operations. Like most things in life, however, there are tradeoffs to counter the benefits realized in using digital signal processing techniques. Today, most digital signal processing hardware implementations are more costly than their analog counterparts (unless very delicate, noise immune, or low frequency analog circuits are required). In place of additive electrical noise in processing a signal, finite register quantization error, round off and truncation errors, and register overflow errors introduce noise into the signal. Also, the cost of the digital hardware system increases sharply if real-time high frequency signal processing is required. Therefore, digital signal processing is not an automatic choice over its analog counterpart.

4.2 CONSIDERATIONS IN SAMPLING A SIGNAL

Sampling a signal, or establishing a digital representation for the value of a signal at a specific moment in time, is generally accomplished with an analog to digital (A/D) converter. Since A/D converters require a finite time to perform such a conversion, an analog sample and hold circuit is often used as a buffer between the input signal and the A/D converter. This circuit acquires the value of the continuous signal at the moment of sampling, and holds the value constant while the A/D conversion is completed. The digital signal processing algorithm which is applied to such a sampled signal depends on what processing function is desired. Whatever algorithm, the function of the process is implemented through a sequence of numerical processes, in general operating on the sampled or quantized input. Since the process is effected through numerical operations, a digital computer is ideally suited for the implementation of such algorithms. The processing of such an input signal may occur while the signal is occuring (in real-time), or the signal samples may be stored for later processing.

The type of result which is obtained from a digital signal processing algorithm depends on the algorithm being performed. The results may be processed further by other algorithms, or the result of the algorithm may be a stream of digital values which correspond to a transform of the analog input signal. It may, therefore, be desired to reconstruct an analog output signal from such a stream. This function is accomplished with a digital to analog (D/A) converter. A D/A converter produces an analog output which is proportional, at a given moment, to the digital value which is supplied to it.

The sampling of a continuous signal affects the frequency domain characteristic or spectrum of that signal, depending on the relationship of the sampling rate to the highest spectral component of the signal. Intuitively, it seems reasonable that if a signal is varying 10 times a second, sampling the signal once a second will not 'capture' the information in the signal. The Nyquist relation states that, for an infinite duration signal, sampling at twice the frequency of the highest spectral component of the input signal will result in a complete capture of the information contained in the signal. Sampling at rates below the Nyquist rate results in aliasing distortion or noise. This relationship is best explained by a diagram. In Fig. 4.1, it can be seen that the spectrum of the sampled signal is that of its analog counterpart, except that the spectrum repeats itself or is folded by the sampling frequency. If a sampling frequency of less than two times the highest spectral component of the input signal is used, aliasing noise is added into the sampled spectrum. In practice, sampling rates well in excess of twice the highest component are used. Bandlimiting lowpass or anti-aliasing filters are often used to bandlimit the input signal and ensure a minimum of aliasing noise.

Since the continuous signal is represented by digital values of finite length, quantization noise is also introduced. If an N bit representation is used, 2^N values can be represented. If a signal variation smaller than $1/(2^N)$ of the input signal range occurs, the variation will go 'unnoted'. This problem manifests itself as additive quantization noise which is a function of the number of bits of resolution used.

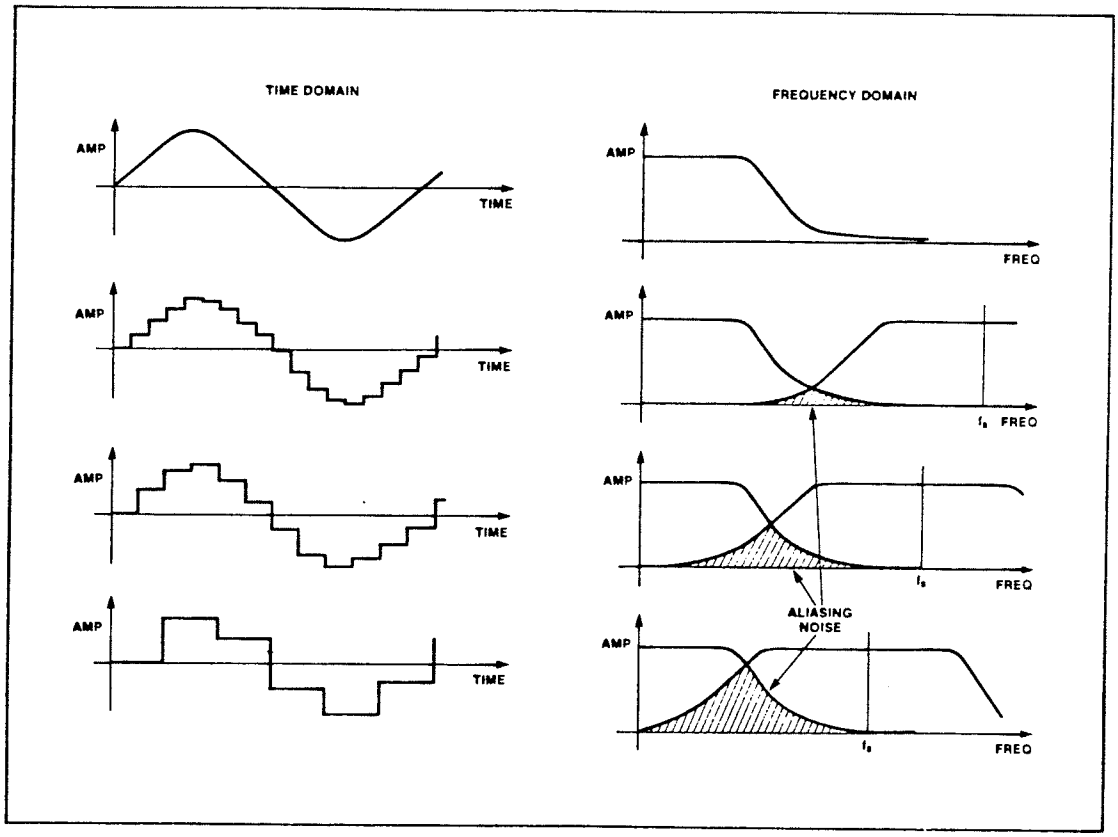


Figure 4.1: Spectrum aliasing with decreasing sampling frequency

Taken from Intel Corp., 2920 Analog Signal Processor Design Handbook. Santa Clara, 1980.

If an analog output is required from a digital system, this is accomplished with a D/A converter. The output of such a system remains constant until a new output is available. At that time, the output jumps to the new output value. These jumps introduce high frequency components into the output. A lowpass reconstruction filter is often used to smooth the output waveform.

4.3 SPECTRAL AMPLITUDE ESTIMATION FOR EEG AND EOG

The first step in any system which is to aid in the sleep signal classification process is the consideration of spectral amplitude estimation. The most frequently used method for such estimation has been analog bandpass filtering. The low frequency nature of these signals, however, makes analog solutions difficult and costly to implement. As a result, various digital signal processing techniques have been attempted with varying degrees of success. Pitch period detection algorithms have been used, and are based on a time domain analysis of a waveform. The algorithm detects the rise or fall of a signal, thereby determining when peaks and troughs have occured. With the time between peaks and troughs and the difference between the peak and the trough, a crude estimate of the amplitude spectrum is made. These algorithms are desirable because of their simplicity, but have had only limited success when applied to the problem of EEG amplitude spectrum estimation. Fast Fourier transforms and Walsh transforms have been applied to the problem, and both have achieved good success. Between the two methods, FFTs are more commonly accepted and used. Linear predictive coding has been used as a method of characterizing the EEG and EOG in real-time, with Fourier

transform techniques applied in post-analysis to evaluate the spectrum. Correlation techniques for spectral estimation of sleep signals have not commonly been used, although recent publications on spectral analysis have advocated their use [34].

The decision on which method to use in an implementation for the spectral amplitude estimation is influenced by several factors. A significant factor steering any such practical decision is the cost of the solution. Many of the attempts at this problem have been based in large and well funded laboratories which have been able to afford main-frame computers dedicated in real-time to the problem. Less expensive but still costly solutions have been to store all the samples from an entire night's sleep and analyze the signals after the fact. This alternative is costly in that a large capacity of bulk storage is required for the samples (at least 1.5 Megabytes formatted per EEG track).

Microprocessor based solutions are comparatively inexpensive. Most inexpensive microprocessors, however, have instruction sets and word sizes which are not well suited for digital signal processing applications. As a result, the use of microprocessors for real-time digital signal processing can be difficult. More advanced configurations of general purpose microprocessors can be used, but recent advances have also been made in the development of microprocessors which are oriented towards digital signal processing. The Intel 2920 is such a device, as it contains analog multiplexers, sample and hold circuitry, an A/D converter, random access memory (RAM), a programmable permanent memory (EPROM), a D/A converter, and output demultiplexer and hold circuitry on one integrated circuit (IC). The 2920 is able to implement a shift and

add/subtract in one instruction, and is able to conduct parallel digital and analog operations. All internal registers are 24 bits wide, with a 9 bit bipolar A/D and D/A converters. This single chip signal processor is quite suitable for real-time signal processing. Real-time signal processing reduces the need for a bulk storage device to save the samples, thereby significantly reducing the cost of the system.

Given the attempt to try to reduce the cost of the system through real-time signal processing with the Intel 2920, it is nessesary to review the processing requirements of the various algorithmic alternatives. The fast Fourier transform is an algorithm which performs a time domain to frequency domain transformation. Given a set of time domain samples, this algorithm provides a transformation to a set of values which correspond to estimates of the frequency domain characteristics presented in the time domain samples. The results of this algorithm are, in general, complex values which estimate the magnitude and phase of the spectral components. The larger the set of samples taken in such a calculation, the better the resolution or spacing between the values calculated in the frequency domain. The FFT , however, is subject to a phenomenon known as windowing. Briefly, there is distortion in the calculated spectrum which results from the fact that the input signal has not been considered for an infinite time. Any real FFT algorithm is of finite size, and therefore windowing distortion is always present in the calculated spectrum. Advanced techniques can be applied to reduce this problem, although it can not be totally removed. The FFT is also quite processing intensive. In order to perform the FFT, a large number of complex arithmetic calculations, and enough memory to store at least

the samples for the FFT and the results are required. Although the 2920 is limited in the amount of available memory, and does not implement complex arithmetic very easily, it is well suited for digital filtering. Digital filtering is a technique which facilitates a digital approximation of an analog filter. Primarily for this reason, together with the experience in EEG analysis based on spectral estimation within bands, digital filtering was chosen as the method for the spectral estimation.

4.4 SPECTRAL ESTIMATION USING DIGITAL FILTERS

The scheme used to estimate the amplitude spectrum of the EEG using digital filters can be described as follows. A set of bandpass filters is used to separate out the spectral bands of interest from the incoming EEG and EOG signals. The bands which have been found to be of significance are: 0.25 to 2.0 Hz, 2.0 to 7.0 Hz, and 12 to 14 Hz for the EEG, and 2.0 to 10.0 Hz in the EOG. In order to establish reasonably narrow passbands, fourth order Butterworth bandpass filters were chosen to pass the segments of interest. Since the passbands all exist above dc, the dc component of each of the output waveforms should equal zero. The output from each of the bandpass filters is then passed through a full wave rectifier. The output of the full wave rectifier is then passed through a low pass filter with a break frequency very near dc. The effect of the scheme is that the output of the low pass filter is a value which corresponds to the dc or average in the rectified bandpass signal. A block diagram of this process is shown in Fig. 4.2. In an attempt to minimize the ripple from the rectified bandpass in the low pass filter output, break frequencies for the low pass filters were chosen at least a factor

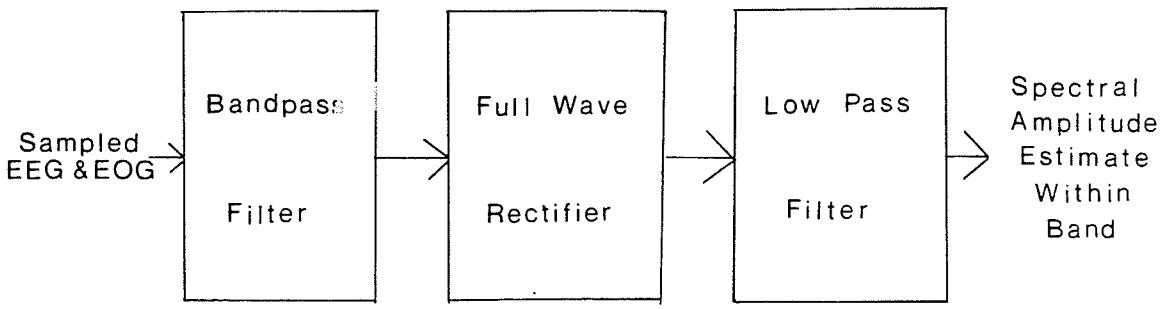


Figure 4.2: Process block diagram

of 10 less than the lower break frequency of the bandpass filter. Also, second order Butterworth lowpass filters were chosen to reduce the ripple in the output.

4.5 FILTER DESIGN CONSIDERATIONS

Infinite-Impulse Response (IIR) digital filters are those which which have a response of infinite duration to an impulse input. An IIR filter has a transfer function, H(z), of the form:

$$H(z)= \frac{b_0 z^N + b_1 z^{N-1} + b_2 z^{N-2} + \ldots + b_N}{z^N + a_1 z^{N-1} + a_2 z^{N-2} + \ldots + a_N} \qquad (4.1)$$

where all the a's are not zero. IIR filters are desirable because they provide good transfer characteristics from a comparatively simple structure. The feedback in these structures, however, introduce the possibility of the filter being unstable. The design of the digital filter is generally accomplished through a transformation of an analog filter design. The design objective is to arrive at a filter design which meets not only the specifications, but is also stable, causal, and is simple enough to be realized with available hardware.

Specification of a filter can include amplitude and phase characteristics, as well as transient performance specifications. There are basically three types of transformation methods which can be used to transform an analog filter design to a digital design. Each of these methods has specifications which are ensured in the digital design. The impulse-invariant and step-invariant methods ensure that the resulting digital filter will have the same impulse or step transient performance as their analog counterparts. Amplitude and/or phase characteristics are usually sacrificed for these results. The third common method is the bilinear transform method. This method provides a filter which approximates the amplitude characteristic of its analog counterpart quite well. For this reason, the bilinear technique was chosen for the design of the bandpass and lowpass filters.

The bilinear transform method maps the left-half of the s-plane into the interior of the unit circle of the z-plane. The jw axis of the s-plane is mapped onto the $|z|=1$ circle of the z-plane. For a stable filter in the z-plane, all poles must reside inside the unit circle. Since all the poles of a stable analog filter reside in the left-half of the s-plane, the digital filter obtained by the bilinear transformation of a stable analog filter is always stable. This mapping is obtained by substituting $z=(1+s)/(1-s)$. Since an infinite space is mapped into a finite space, the mapping cannot be linear. The analog frequencies are therefore prewarped with the tan function $\overline{w} = \tan(wT/2)$ where T is the sampling period of the digital filter. The derivations of the digital filter designs are included in the Appendix.

The choice of sampling rate is based on many factors. If the sampling rate is chosen too low, significant aliasing noise can be introduced into the spectrum. To counter this problem, a sampling rate much higher than the highest critical frequency is desirable. The sampling rate, however, also affects the positions of the poles of the filter on the z-plane. This can be seen in Fig. 4.3. For the example of the 12-14 Hz bandpass filter (4th order Butterworth), it can be seen that as the sampling rate increases, the poles are in general shifted towards the $|z|=1$ circle. Since the $|z|=1$ circle represents the limit of stability, the sensitivity of the system is increased as the poles approach that limit. Any real digital filter can only approximate a designed system to a finite accuracy. As the poles approach the $|z|=1$ limit, the possibility of the approximation residing outside that limit increases. For each of the filters used in this study, the difficulty of realizing each pole was considered. The tradeoffs between sampling rate, pole sensitivity and scaling were considered somewhat heuristically, and the resulting sampling rates are shown in Table 4.1. Complete data on filter coefficients and scaling are included in the Appendix.

TABLE 4.1

Filter characteristics and operating rates

Filter Characteristics (Butterworth)	Operating Rate
1/4 - 2.0 Hz 4th order bandpass	: 50 Hz
.025 Hz 2nd order lowpass	: 12.5 Hz
2.0 - 7.0 Hz 4th order bandpass	: 50 Hz
0.2 Hz 2nd order lowpass	: 50 Hz
2.0 - 10.0 Hz 4th order bandpass	: 50 Hz
12.0 - 14.0 Hz 4th order bandpass	: 200 Hz
1.0 Hz 2nd order lowpass	: 200 Hz

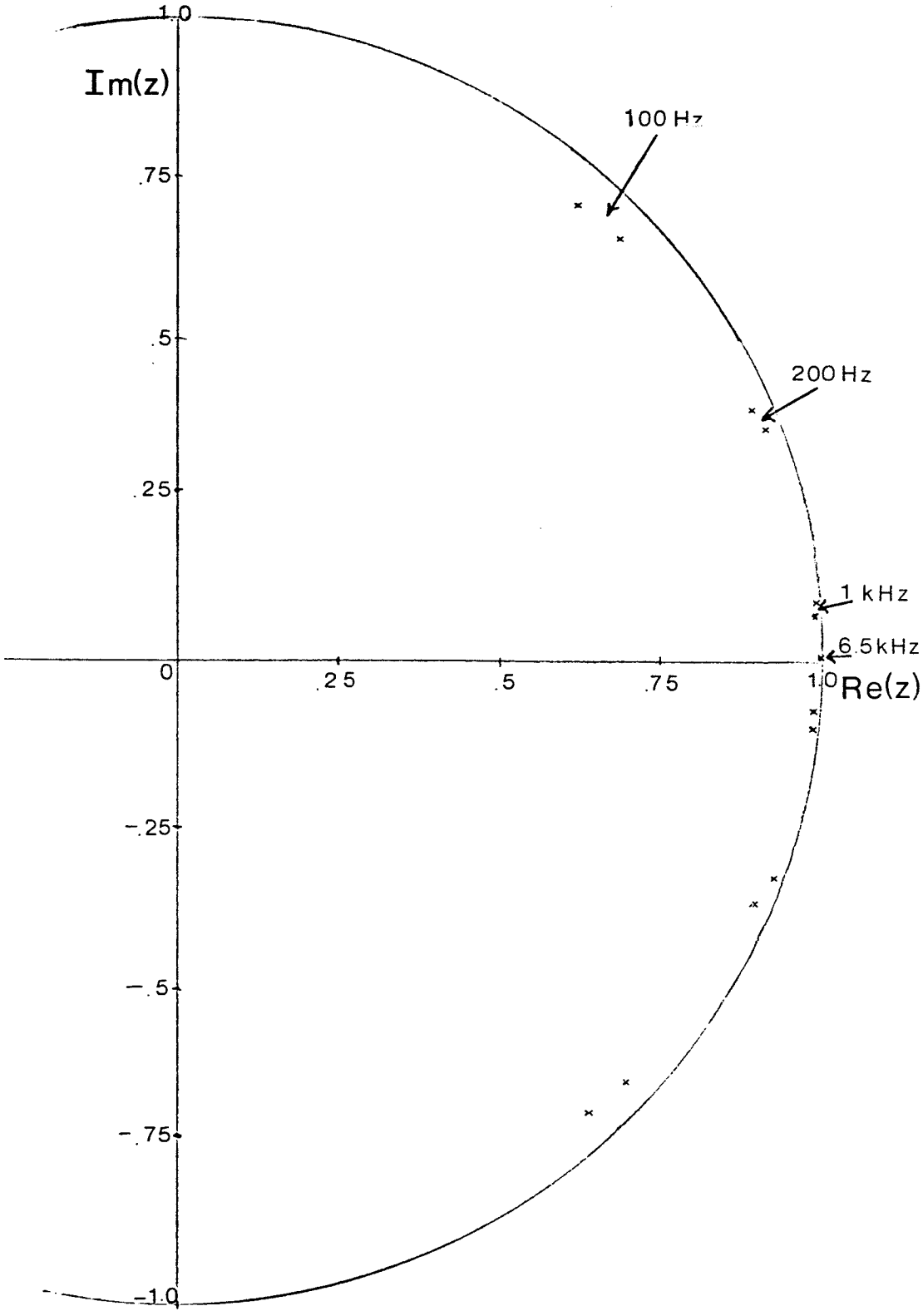


Figure 4.3: Shifting pole position with sampling rate

4.6 IMPLEMENTATION CONSIDERATIONS

The design considered in the previous section must be implemented with a structure of arithmetic operations and delays. The implementation is best described with a block diagram. These block diagrams use delays, multipliers, and adders. Transfer functions are most often realized in blocks of first or second order sections, since higher order sections are quite sensitive to parameter variations. Direct or canonical-forms are two common realization methods. The poles associated with the filters are neither close to the $|z|=0$ point or the $\mathrm{Im}(z)=0$ line. Given this fact, and the fact that there are complex conjugate poles, Chen [35] recommends the use of parallel (second order) canonical-form realization blocks. This choice is based on a comparison of the sensitivity of the poles, the number of arithmetic operations required, and the flexibility to realize complex conjugate poles with each form. The transfer functions for the filters are:

$$\text{Bandpass: } H(z)= \frac{K\,(z-1)(z+1)}{z^2 - Az + B} \cdot \frac{(z-1)(z+1)}{z^2 - Cz + D} \qquad (4.2)$$

$$\text{Lowpass : } H(z)= \frac{K(z^2 + 2z + 1)}{z^2 + \mathrm{alp}\ z + \mathrm{bet}} \qquad (4.3)$$

These transfer functions can be realized with the block structure seen in Fig. 4.4.

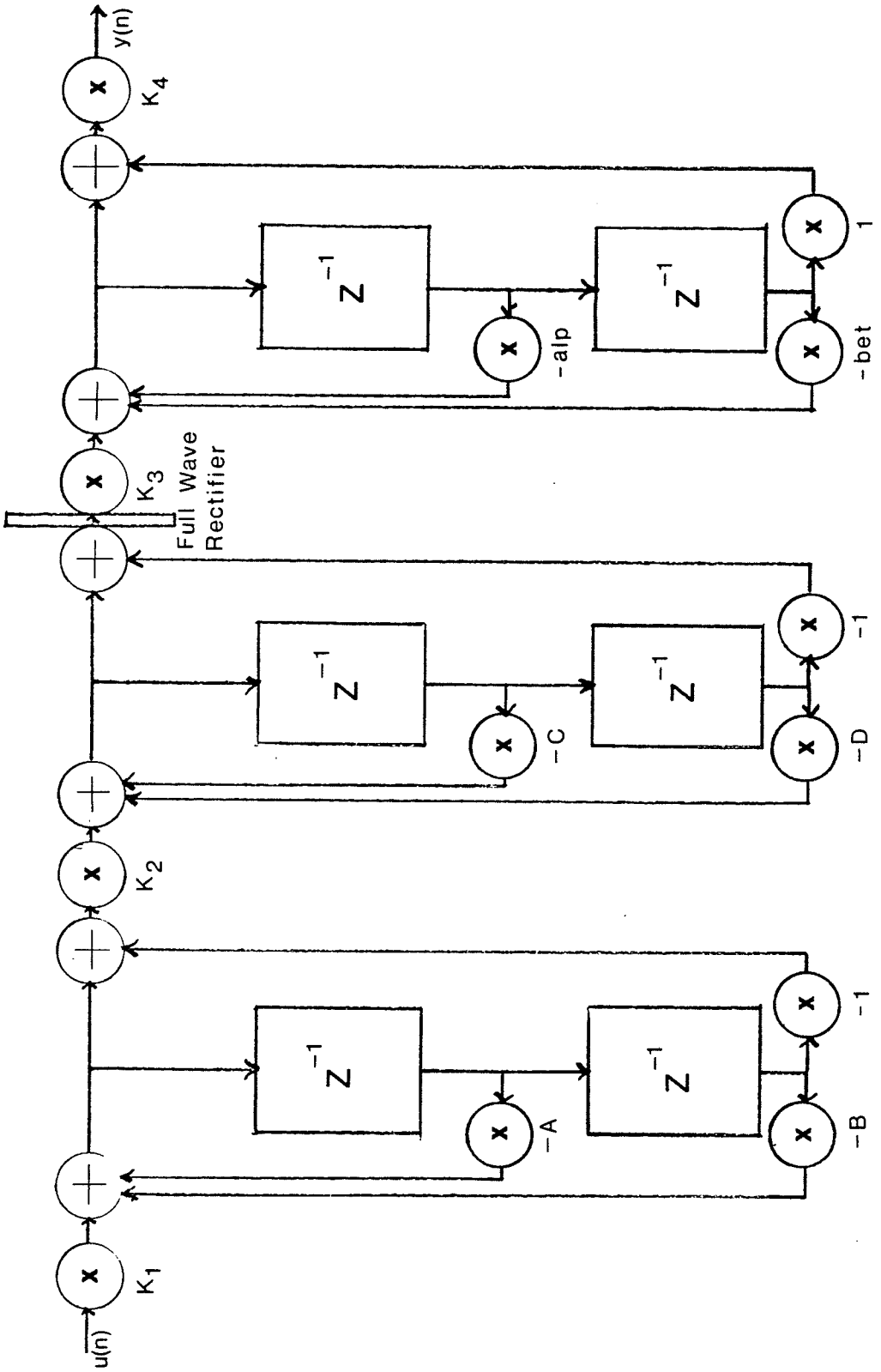


Figure 4.4: Block realization

4.7 INTRODUCTION TO THE 2920 SIGNAL PROCESSOR

The Intel 2920 signal processor is a microprocessor designed for signal processing. The device contains on a single chip: a 9 bit bipolar A/D converter with a 4 channel multiplexer and sample & hold, a D/A converter with 8 demultiplexed output channels with sample & holds, a 25 bit arithmetic logic unit, 40 - 24 bit dual port RAM locations, 192 - 24 bit program (EPROM) memory locations, a shifter unit, and an instruction set which is designed for doing digital signal processing. Each instruction has an arithmetic and an analog part. Arithmetic and analog processes are conducted in parallel. An arithmetic shift and add/subtract can be implemented with one instruction. No internal bus signals are available, and the device is simply supplied with input signals, power, and a crystal. A diagram of the architecture of the device is shown in Fig. 4.5.

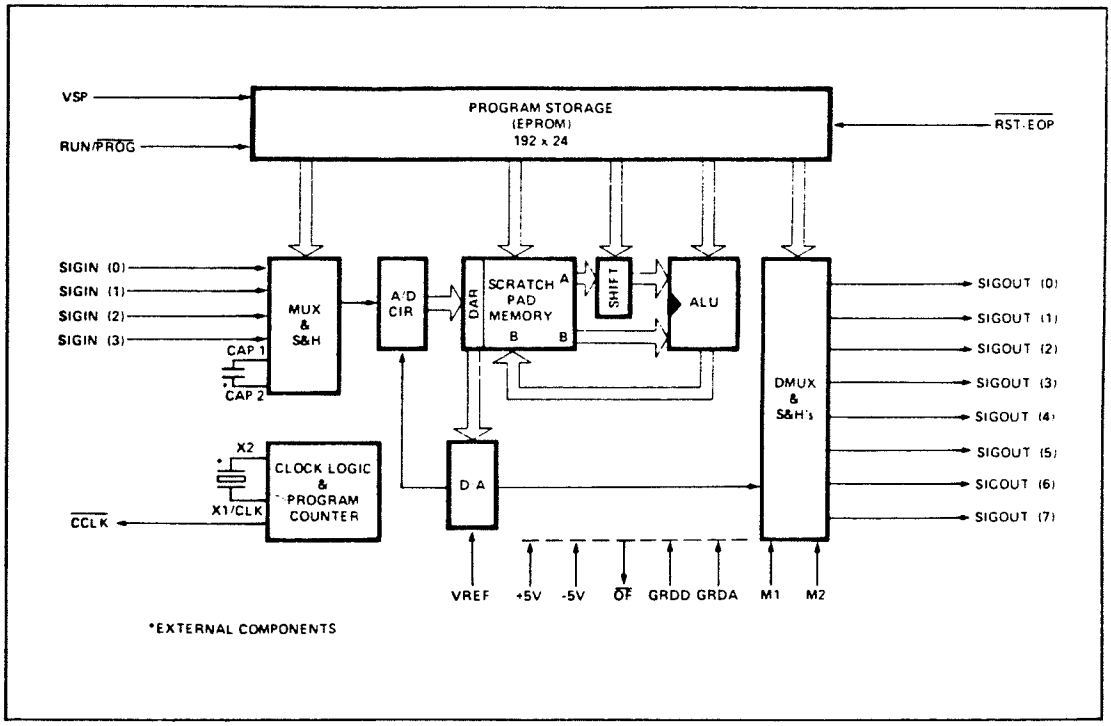


Figure 4.5: 2920 block diagram

Taken from Intel Corp., 2920 Analog Signal Processor Design Handbook. Santa Clara, 1980.

4.8 2920 PROGRAM DEVELOPMENT

The 2920 is programmed via an EPROM which resides in the device. This implies that an EPROM programming circuit must be available to program a 2920. Also, the 2920 instructions are 24 bits long and comprised of 5 operands. The bit string is also jumbled before it is programmed into the device. The effective result is that an assembler is essential to the development of 2920 code. The assembly/programming function can be purchased from Intel in two forms. The best solution is to buy an Intel development system. This option, however, costs approximately $50,000. A second alternative is to purchase a dedicated development board from Intel for a cost of approximately $1,400. (The University of Manitoba has purchased such a board). This board is, however, somewhat unreliable and less than optimal in terms of ease of use. Therefore, Mr. Andy Weirich and I designed and implemented a 2920 assembler and programmer for the Motorola EXORcisor laboratory computer. Programming is accomplished through a programmer board inserted in the EXORcisor system bus. The assembler was developed in Motorola MPL [36] (a PL/1-like compiler which compiles to 6800 assembly mnemonics), is 940 lines long, and operates in the EXORcisor. It allows free format entry of 2920 programs, with full error diagnostics and error recovery during assembly and programming. Programs are entered and edited with the EXORcisor editor, and are stored on floppy disk. An interface between the EXORcisor and a Heath H14 printer was also developed, to allow paper copy of disk files. A listing of the assembler source code is included in the Appendix.

4.9 FILTER PERFORMANCE CONSIDERATIONS

The coefficients for the digital filters were calculated with a FORTRAN program which is included in the Appendix. The 2920 is able to multiply with shifts and adds/subtracts, and therefore a sign digit canonic form breakdown of each of the coefficients was also evaluated by the program. The 2920 used in this implementation was supplied with a 5.0 MHz crystal. The 2920 requires 4 crystal cycles/ instruction, and therefore a maximum length program of 192 instructions results in a sampling rate of approximately 6.5 kHz (assuming only one sample is performed per program pass). At this sampling rate the poles of the digital filters are very sensitive and difficult to realize. Therefore, a submultiple sampling scheme was implemented. The submultiple sampling scheme is based on software counters and conditional delays. The calculations for all the filters are conducted at a rate of 6.5 kHz, with delays effected only when certain software conditions have been met with the counters. Through this scheme, software adjustable operating rates were established to effect each of the desired operating rates. The interested reader can refer to Intel's design handbook [33].

Anti-aliasing filters were not implemented as the EEG and EOG signals available for this processing were already bandlimited. The outputs of the digital filters were transferred to a second computer digitally, and therefore, reconstruction filters were not considered.

Overflow and scaling considerations were addressed as follows. Given a realization of a structure with a given machine, it is a nontrivial problem to establish which (bandlimited) input sequences will result in the highest probability of overflow [37,38]. In order to determine some

estimate of appropriate scaling factors, the 2920 operations were simulated on the Amdahl computer in FORTRAN. Practicing saturation arithmetic as effected by the 2920, a full scale input signal at the center frequency of each bandpass segment was passed through each of the filter setups. After operating each of the filters long enough to settle down, several cycles of the input waveform were passed to determine if overflows were occuring anywhere in the structure. If overflows did occur, the appropriate scaling factor was reduced by a power of 2 and the process repeated. In this way, a set of scaling factors was determined which provided a reasonable, though not optimal, use of the implementation hardware. A listing of the simulation software is included in the Appendix.

4.10 FILTER TESTING PROCEDURES

Each of the filter setups was tested with sinusoidal signals, with the outputs from the filters converted to analog by the 2920. The low frequency nature of the filters allowed for accurate paper plotting of the output. The performance of each of the setups was measured with standard frequency response techniques. This was accomplished by applying a sinusoidal input and measuring the amplitude of the output. The filters were found to perform very well, and the spectral amplitude characteristics for each of the filters matched that theoretically predicted almost exactly. This level of performance can be attributed, for the most part, to the large word (data) path provided by the 2920. As mentioned earlier, the 2920 provides a 9 bit A/D conversion, and uses/stores 24 bits for all calculations. Also, all calculations are conduct-

ed with 25 bits of accuracy, with the result rounded to 24 bits. The spectral amplitude characteristics for the bandpass filters is shown in Fig. 4.6.

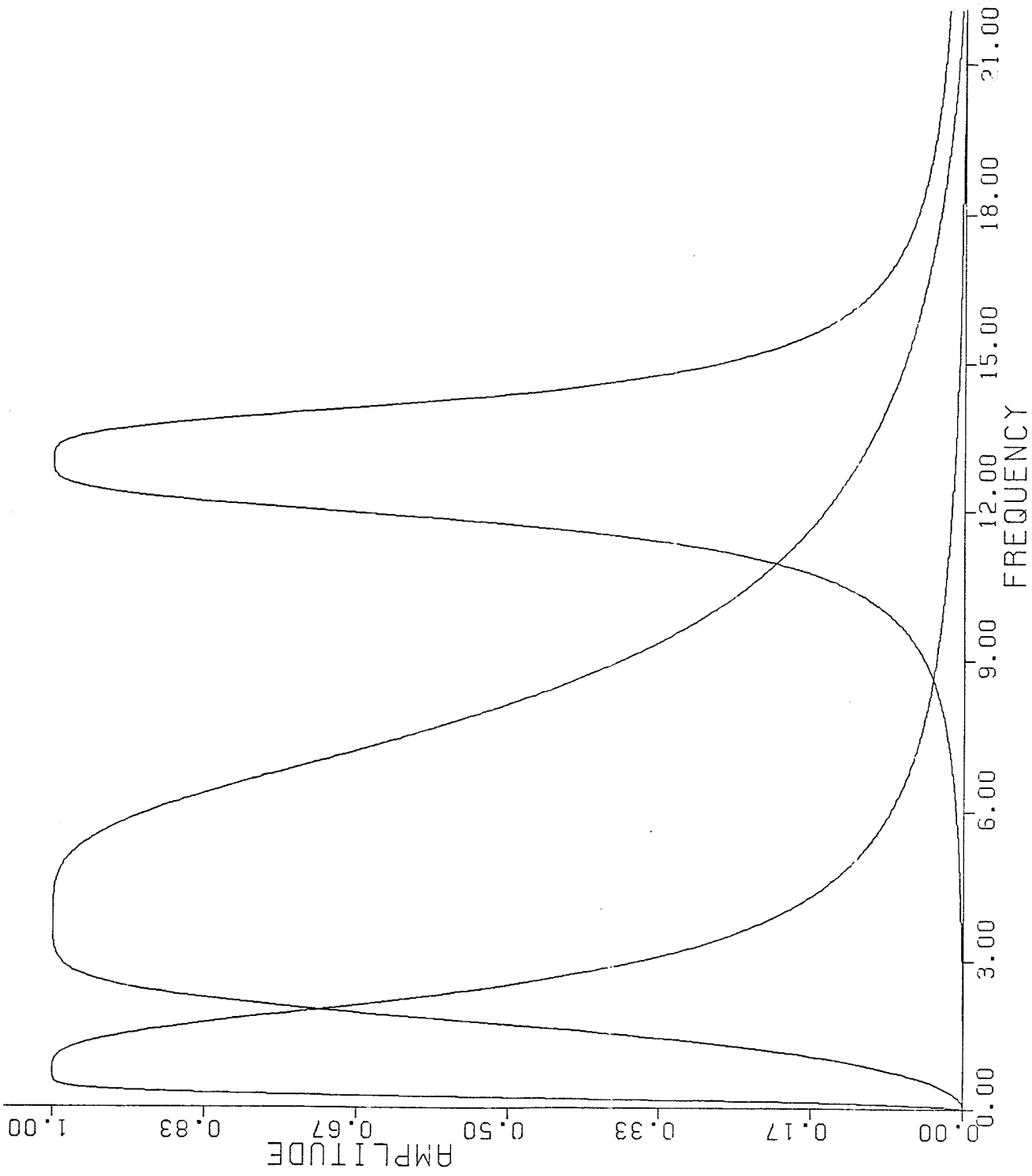


Figure 4.6: Bandpass filter amplitude characteristics

Chapter V

TEST SYSTEMS AND PROCEDURES

5.1 SYSTEM OVERVIEW

In order to implement the proposed aid for sleep classification, it is necessary to collect the spectral estimates extracted during the recording, and to store them for post-analysis. To accomplish this task, it is necessary to interface the 2920s to a second computer or intelligent storage system. With the extracted features stored, the post-processing can be conducted by the data collection processor, or the data can be transmitted to a more suitable computer.

Within the confines of this study, the objectives were to investigate the proposed method, and to study the use of inexpensive microprocessing hardware for the spectral estimation. Given those objectives and the budget constraints of the project, a system hardware scheme was devised using available equipment. For these reasons, the 2920s were interfaced to a Motorola EXORcisor laboratory microcomputer, and the collected data was transmitted to an Amdahl V7 main-frame computer. The EXORcisor is a 6800 based microcomputer with 56 Kbytes of RAM, 2 floppy disk drives, parallel and serial interface ports, and a disk operating system (Motorola EXORdisk II MDOS). The Amdahl is a large computer which supports many online users, and has a Versatec graphics plotter attached to it.

The EXORcisor does not use a dedicated floppy disk control device, and therefore the 6800 processor itself controls all disk operation ac-

tivities. This fact, combined with the time required to conduct floppy disk activities, imply that the 6800 may be occupied for periods of up to several seconds in a disk read or write. Since it was desired to sample the spectral estimates from the 2920s many times within the briefest sleep stage interval (approximately 15 seconds), a sampling rate was chosen to be 1 Hz. Given this desired sampling rate, it was not possible to conduct disk operations in real-time without missing some sample times. Therefore, it was decided to collect data only to the limit of the EXORcisor's RAM storage capacity, with storage on disk after collection. The collected data was then transmitted to the Amdahl through an RS-232 asynchronous communications link for post-analysis. A system data flow diagram of this configuration can be seen in Fig. 5.1.

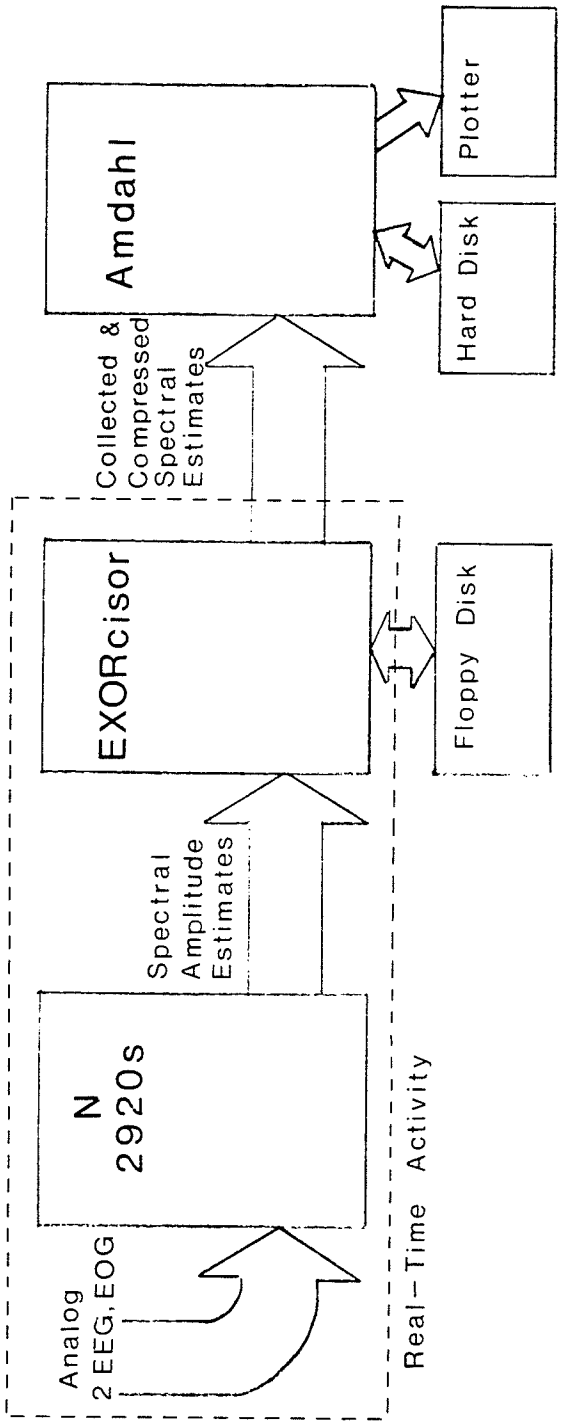


Figure 5.1: System data flow diagram

5.2 2920/EXORCISOR INTERFACE

5.2.1 Transfer Method

A 2920 is capable of producing analog or digital output from any of its 8 output lines. Since the outputs of the spectral estimation algorithms in the 2920s are numerical, it was decided to transfer the estimates digitally. Because of the architecture of the 2920 and the fact than several estimates had to be transfered out of each 2920, it was decided to transfer the estimates out of the 2920s serially. A parallel rather than serial interface to the EXORcisor was chosen in an attempt to maintain as universal as possible an interface. Serial in, parallel out shift registers (74164) were used to construct this interface. Because a submultiple sampling rate was used to effect the required filter sampling rates, a number of effectively wasted program passes were spent between actual filter iterations. These program passes were used to transfer out the bits of the digital spectral estimates. It was decided to pass an 8 bit (twos complement) spectral estimate, and therefore 8 successive program passes were used to shift out the values. The 2920 is capable of generating an end of program pulse (EOP) through the use of the EOP instruction. The EOP pulse was, therefore, used to control the shifting of the registers. Since it was desired to have the shift registers stop shifting after the 8 passes, a control output was produced by the 2920 (through software timers) to enable the shifting for only 8 program passes. Software timers in the 2920 were also used to establish one second intervals, and the operations of the filters were arranged so that no new filter output values were determined within the 8 program pass period. The shift control signal was also used to trigger an in-

terrupt in the EXORcisor, thereby signalling that a read of the estimate values should occur.

5.2.2 Interface Problems

There were a number of problems encountered in developing this interface. The two most serious problems stemmed from glitches which were present in the 2920 output signals. The first of these problems was that the EOP pulse generated by the 2920 contained 2 spikes, each distinct enough to trigger a TTL gate input. This waveform can be seen in Fig. 5.2. The second problem was related to the control signal used to enable the shifting. A digital output of the 2920 was used to produce a logical one, under software control, for the required 8 program pass period. This implies, given the architecture of the 2920, that the output was successively 'rewritten' with each program pass. The successive writing of a logical one with a 2920 is corrupted with a glitch, also large enough to trigger a TTL gate input. This waveform can be seen in Fig. 5.3.

These problems were addressed as follows. The end of program pulse was used to trigger a monostable multivibrator (one-shot) circuit (74LS221) in order to generate the desired single pulse. This circuit was not affected by the second spike of the pulse. The control line was deglitched through the use of a CMOS NOR gate, and a resistor-capacitor pair. A CMOS logic gate input threshold voltage is 50% of its supply voltage. Since the gate draws very little current, a resistor-capacitor pair can be arranged to ensure that one of the gate input lines does not reach the threshold voltage unless a change in the input voltage occurs

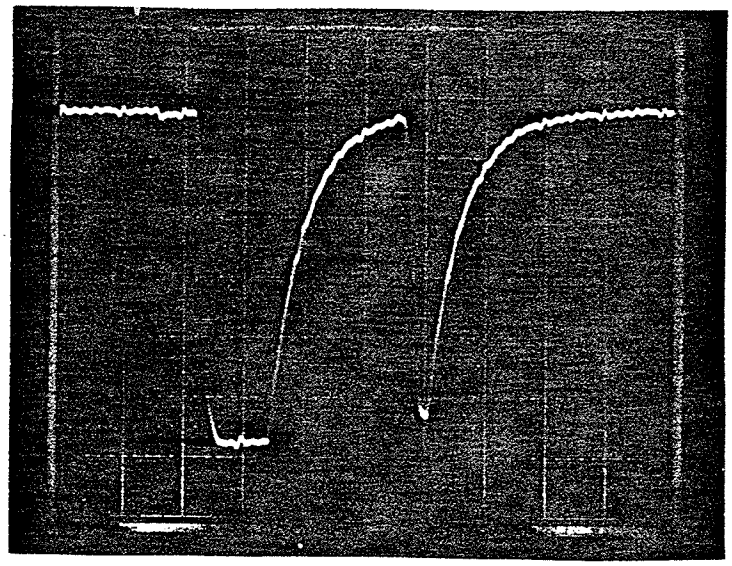


Figure 5.2: 2920 End of program pulse waveform

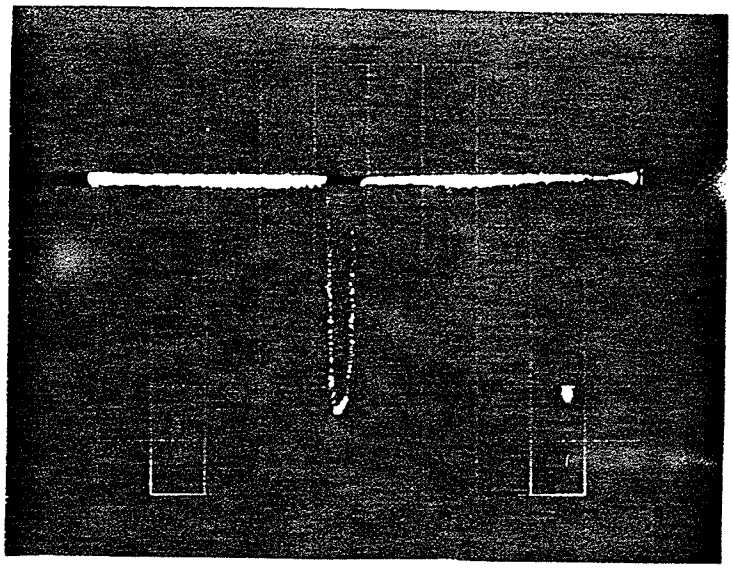


Figure 5.3: 2920 successive logic 1 output waveform

for longer than a selected time. In this way, the control line can be effectively deglitched. Complete circuit diagrams for the 2920/EXORCisor interface and the 2920 programs are included in the appendix.

5.2.3 2920/EXORcisor Synchronization

Since three 2920s were used to accomplish the signal processing, synchronization between the devices and the EXORcisor had to be considered. First of all, it was desired that the 2920s all have their estimates shifted into their shift registers synchronously. In order to accomplish this, one 2920 was chosen as the master. The end of program pin on the 2920 can act as an input or output, and so the EOP generated by the master 2920 (used to control the shift registers) was also used to synchronize the program passes of the 2920s. If the EOP lines of two 2920s are tied together, the devices will be assured to be operating within 6 program steps of each other (the difference arises from the fact that the 2920 prefetches instructions in order to increase its operating rate). On powerup, therefore, the 2920s will all be synchronized after the master 2920 reaches its end of program instruction the first time. In order to synchronize the software timers in each of the 2920s, a start control signal was generated using a CMOS NOR gate and a resistor-capacitor pair. This circuit provided a signal which remained high for several seconds after powerup, and then went low for the remainder of the powered on period. This signal, taken as input by each of the 2920s, provided a means of synchronizing all the software timers. The 2920s all derived their clocks from a common circuit, and therefore all operated

in complete synchronism. The interface and synchronization scheme is general enough to accomodate any number of 2920s, and additional elements can simply be added to the structure in parallel.

5.2.4 EXORcisor Data Acquisition

In order to accommodate the acquisition of the parallel data from the 2920/shift registers, four Motorola 6821 parallel interface adapters (PIAs) were used. This scheme provided parallel access to 8 bytes of input. The outputs of the shift registers were simply connected to the PIAs. A clock was added to the EXORcisor in order to allow correlation of the recorded information to the manually classified sleep record. A Motorola 6840 programmable timer was used to accomplish this function.

As mentioned earlier, the choice of realization structure for the implementation of a digital filter is based on many factors. The form chosen in this study was based on recommendations suggested by Chen [35]. One of the considerations in such a decision is whether the filter structure will always return, from a given state, to a stable zero output for a continuous zero input. For a stable filter design, the structure suggested by Chen assures that the implementation will always be stable, given an initial state of zero in all the filter delays and a bandlimited input signal. Recent studies have shown, however, that these structures may lead to unstable filters given a completely random initial state [39,40]. The memory in the 2920 is in a random state on powerup. Given the architecture of the 2920, it is not possible to effect a zero initial state for all the filter delays without sacrificing many program instruction steps. It was decided, therefore, to implement

the filters without establishing an initial zero state. The possiblity of having an unstable structure was realized only late in the study, as only a few of the many possible initial random states of the 2920 lead to unstable conditions. While the recent studies of this instability have suggested structures less susceptible to the problem, they were not implemented in this study. To accomodate the problem, therefore, a program was developed for the EXORcisor to display the 2920 outputs. This routine, which is included in the appendix, allows the user to see the outputs of the 2920s once a second. When no signals are being applied to the 2920s, the outputs should equal zero. After the 2920s are powered on, time is required for the outputs of the filters to stabilize to zero (given the initial random state). Should a structure become unstable, the output will not settle to zero. If this occurs, the 2920s can be powered down for a moment, and a different random state will be established in the 2920 RAM on powerup. In practice, it was found that the filters were almost always stable, and that a single powerdown/up generally cleared any problem which might be present. In order to stop the display process, the user simply enters any key on the keyboard.

5.3 EXORCISOR/AMDAHL INTERFACE

The interface between the EXORcisor and the Amdahl was implemented in the following manner. A program was developed for the EXORcisor to allow it to look like an intelligent terminal to the Amdahl. The user initiates the terminal emulation program (included in the Appendix), and receives a prompt from the program. The user can then logically establish communications between the EXORcisor system terminal and the Amdahl

through the use of an asynchronous communications interface adapter (ACIA) card in the EXORcisor bus. The EXORcisor program passes the messages between the two ports until an escape character is sent from the user's terminal. At that time, the EXORcisor program sends the user a prompt, and allows the user to either go back 'on line', upload an EXORcisor file, or end the procedure.

In order to conduct machine to machine communications, there must not exist the possibility of one machine sending information to the other more quickly than the second can accomodate. Most large, multi-user systems, however, are managed by complex operating systems and activity schedulers, and have difficulty assuring that absolutely no data 'over-runs' will occur with their online users. This can present problems for the user of a microprocessing system who wishes to transfer data to the main-frame. The University of Manitoba Computer Centre has addressed this problem in the following manner. The terminal control program operating in their Amdahl main-frame site has been modified to allow the user to specify a character which is to be sent to the users terminal just before input will be accepted. This feature, when used within a very limited range of main-frame activities, will assure that data over-runs will not occur for a single burst of input of up to 150 characters followed by a carriage return. The user then has to wait until the Amdahl responds with the specified character before the next burst can be sent. The software operating in the Amdahl which is accepting the incoming data, however, must not provide any input prompting (ie. line numbers), or the assurance is lost. Also, the user should disable the message receive capability of the userid in order to assure that nothing

disrupts the data transfer. For the University of Manitoba Amdahl site, this can be accomplished (with the MANTES monitor system) by locating to the file in which the data is to be placed, and entering: 't profile noopmsgs nointercom;t terminal write(xon);i la sup noi'. The data can then be sent in the manner described. This scheme is effected by the terminal emulation routine operating in the EXORcisor. If the EXORcisor waits for the start character from the Amdahl for more than 10 seconds, the routine assumes that there has been a breakdown in communications and terminates the procedure. (In practice it has been found that, when the Amdahl is heavily loaded, this period can be exceeded even though communications have not broken down). The EXORcisor routine provides a line count as the lines are being sent, and logically reconnects the user terminal to the Amdahl when the upload has been completed. (For the MANTES user, the the following should then be entered to reset the user-id: 't terminal nowrite;t profile opmsgs intercom').

5.4 SYSTEM TESTING

The system was tested in the following manner. As mentioned earlier, the filter performances were tested with standard frequency response techniques. The digital transfer scheme was tested with a 2920 and the EXORcisor programmed to simply sample an input signal, transfer the digital sample to the EXORcisor, and store the sample on the EXORcisor disk. Sinusoidal and dc test patterns were captured in this manner. These patterns were then transfered from the EXORcisor to the Amdahl, and were plotted on the Versatec plotter. The actual filter programs with the digital interface scheme were also checked by running the pow-

er-on routine in the EXORcisor, and observing the spectral estimates as passed from the 2920s. Various frequencies and amplitudes of a sinusoidal input were applied in this test.

For evaluation purposes, 2 hours of sleep signals (2 EEG and an EOG) were recorded with a Hewlett Packard 3960 FM instrumentation tape recorder. These signals were also plotted with a polygraphic recorder for manual classification. The beginning of the recorded signal was identified with a synchronization mark to allow correlation with the manually classified sleep record.

5.5 OPERATION OVERVIEW

In order to process a given sleep signal recording, the following procedure is used. Finding the synchronization mark on the tape is accomplished with a Hewlett Packard 7402A strip chart recorder. With time zero of the recording established, the 2920s are powered on and allowed to stabilize. This is checked with the power-on routine, as explained earlier. When the filters are stable, the power-on routine is terminated by entering any key on the system terminal.

Recording of the spectral characteristics is conducted as follows. The user initiates the record program in the EXORcisor (the program listing is included in the Appendix). The record command is appended with the name of an EXORcisor disk file into which the recorded data is to be stored. The routine checks for disk space and opens the file. The routine then prompts the user and waits for a carriage return to synchronize time zero. The user then starts the tape recorder and enters a carriage return at the same time. The real-time clock in the EXORcisor

is zeroed when the carriage return is received, and the interrupt from the 2920s is enabled. The EXORcisor is then interrupted once a second by the 2920s to read the current spectral estimates. The 2920s shift their most current estimates into the shift registers just before the interrupt, and the shift registers remain 'frozen' until just before the next interval arrives. The EXORcisor routine compares the values read with those last recorded, and determines if any of the newly read values differ beyond a threshold from those corresponding in the last record iteration. The difference allowed is adjustable for each of the estimates. If a difference is exceeded, the newly read values are added to the list of recorded values, and appended by the current reading of the clock. These values are then taken as the comparison values. This scheme effects some data compression in that no new entries are added to the list of recorded values if the incoming estimates are the same within the thresholds. The recording process is terminated if either the end of the EXORcisor RAM is reached, or the user enters any key on the system console. At that time, the recorded hexadecimal values are written into the specified disk file. While the recording is in progress, the current number of records written into the list is displayed on the system console.

Before the collected estimates can be transfered to the Amdahl, they must be converted into ASCII character images of the hexadecimal values. It is also desirable to have the times of the recording transitions placed in the frame of reference of the polygraphic recording (for comparision with the manually classified record). The polygraphic recording paper is numbered, with one page corresponding to 30 seconds of record-

ing. The MAP routine, which is run in the EXORcisor, is used to accomplish these functions (program is included in the Appendix). This routine prompts the user for the input hex file name, an output file name , the starting page number of the recording, and the offset of the starting pulse (in seconds) from the start of the page. The recorded values (2s complement 8 bit) are converted to an ASCII image of signed base 10 integers (ranging between -128 and +127), with one line used for each recorded transition. The mapped image of the estimates is then transmitted to the Amdahl with the scheme described earlier.

Chapter VI

RESULTS AND DISCUSSION

The described system was used to estimate the spectrum for a patient exhibiting normal and abnormal sleep waveforms. The sensitivity in detecting changes in the incoming spectral estimates was set at any change beyond the least significant bit of an estimate. With this sensitivity to change, a 90 minute recording was possible before a memory limit of 25 Kbytes of data was reached in the EXORcisor. The record which was chosen provided examples of sleep stages 1 through 4, and of awake sleep. Approximately 60 minutes into the recording, the patient began a period of frequently aroused or disturbed sleep. The EOG channel, although processed and recorded, was not considered in the preliminary evaluation of the recorded estimates. Likewise, the 12 to 14 Hz band estimate used to detect sleep spindles was not considered in this study. The primary concern in this preliminary evaluation was the correlation between the observed spectral estimates in the low and mixed bands, and the manually classified sleep record.

The direct plot of these estimates, as recorded at the end of each one second interval over approximately the first 36 minutes, can be seen in Fig. 6.1. The classifications of the record, as determined manually, have been superimposed on the plot. (The manual classification was done without the clinician seeing the recorded spectral estimates). It can be observed that the spectrum does not appear very stationary over that

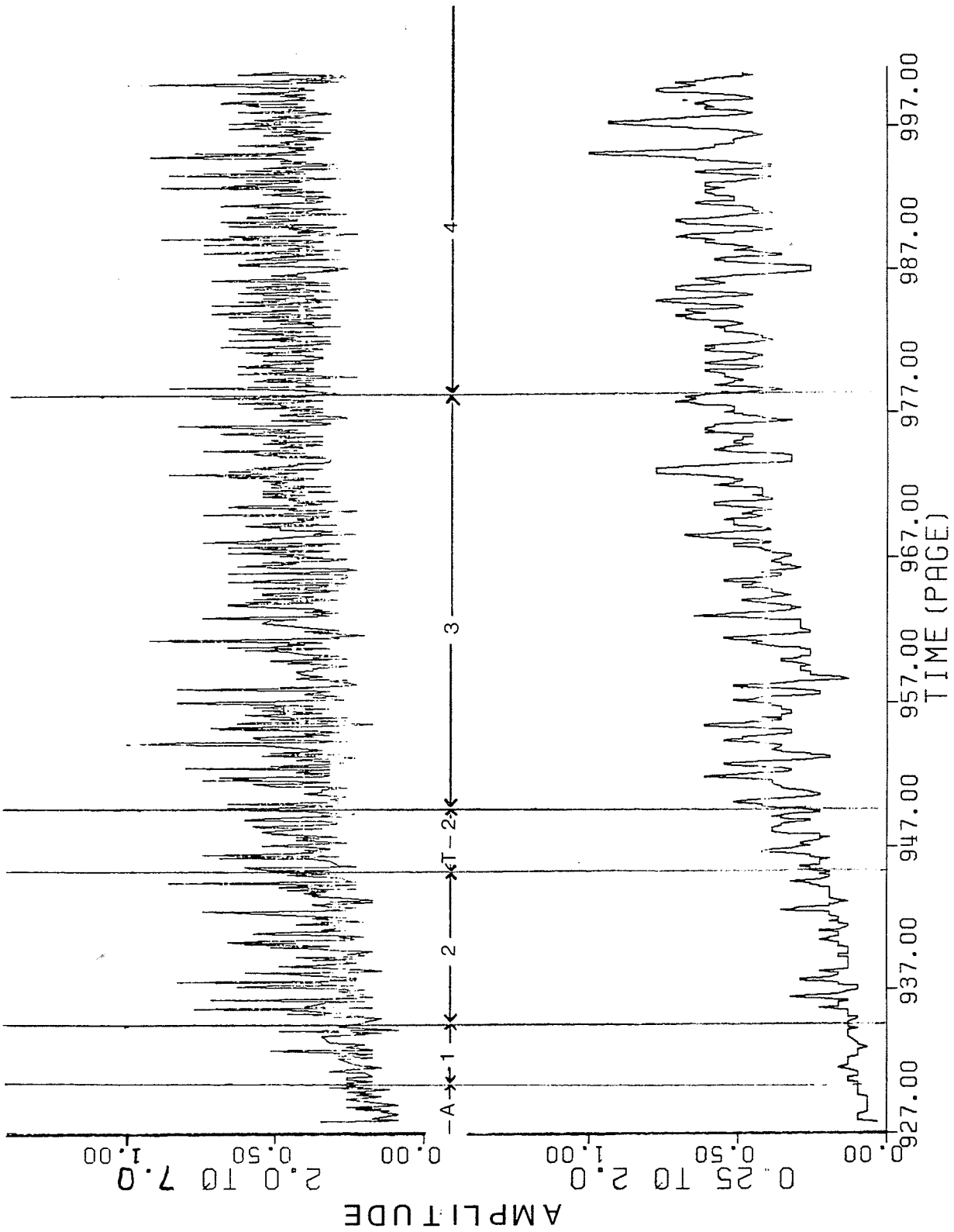


Figure 6.1: Spectral estimates sampled at 1 second intervals

interval, although general trends can be seen to correlate fairly well with the manually classified record. The clinician applies some averaging when manually assessing the plotted sleep signals. It is also clear, from Fig. 6.1, that it would be difficult to establish a pattern recognition scheme able to accomodate the detail presented in a long sleep segment without some averaging. In an attempt to establish what averaging or epoch interval would be most suitable for the classification of normal and abnormal sleep, various epoch rules were applied to the recorded estimates. Plots of the spectral estimates with the application of 5, 10, 30, and 60 second epoch rules can be seen in Figs. 6.2, 6.3, 6.4, and 6.5. The program generating these plots is included in the Appendix. From these plots it can be seen that the spectral estimates become less variable when considered with a longer averaging period. Past studies have commonly used epoch averaging periods of 30 to 60 seconds. These plots show that, with such an averaging period, there is a distinct gross correlation between the spectral estimates and the manually classified sleep stage. The 30 and 60 second epoch rule plots show how past studies have been able to use simple level detectors to establish sleep stage. The exact threshold levels to be applied, however, can vary even within a single recording.

The difficulty in the application of a longer averaging period arises when sleep stage transitions occur in fairly rapid succession. An example of this problem can be seen in Fig. 4.5, where a 10 second epoch rule has been applied to a later portion of the same patient's record. During this period, the patient went through a number of sleep stages in a short time. The correlation between the superimposed manual classifi-

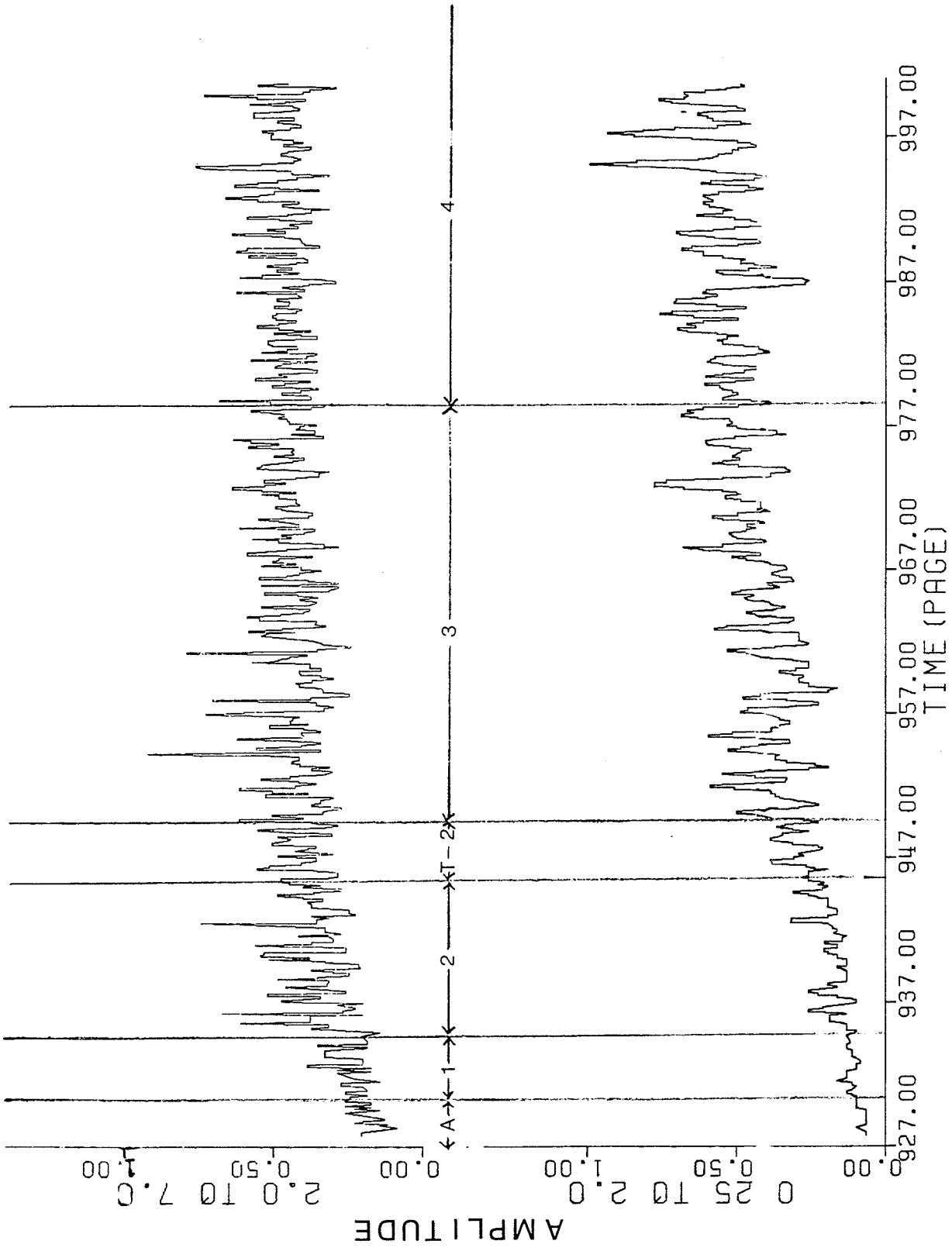


Figure 6.2: Spectral estimates with 5 second epoch rule

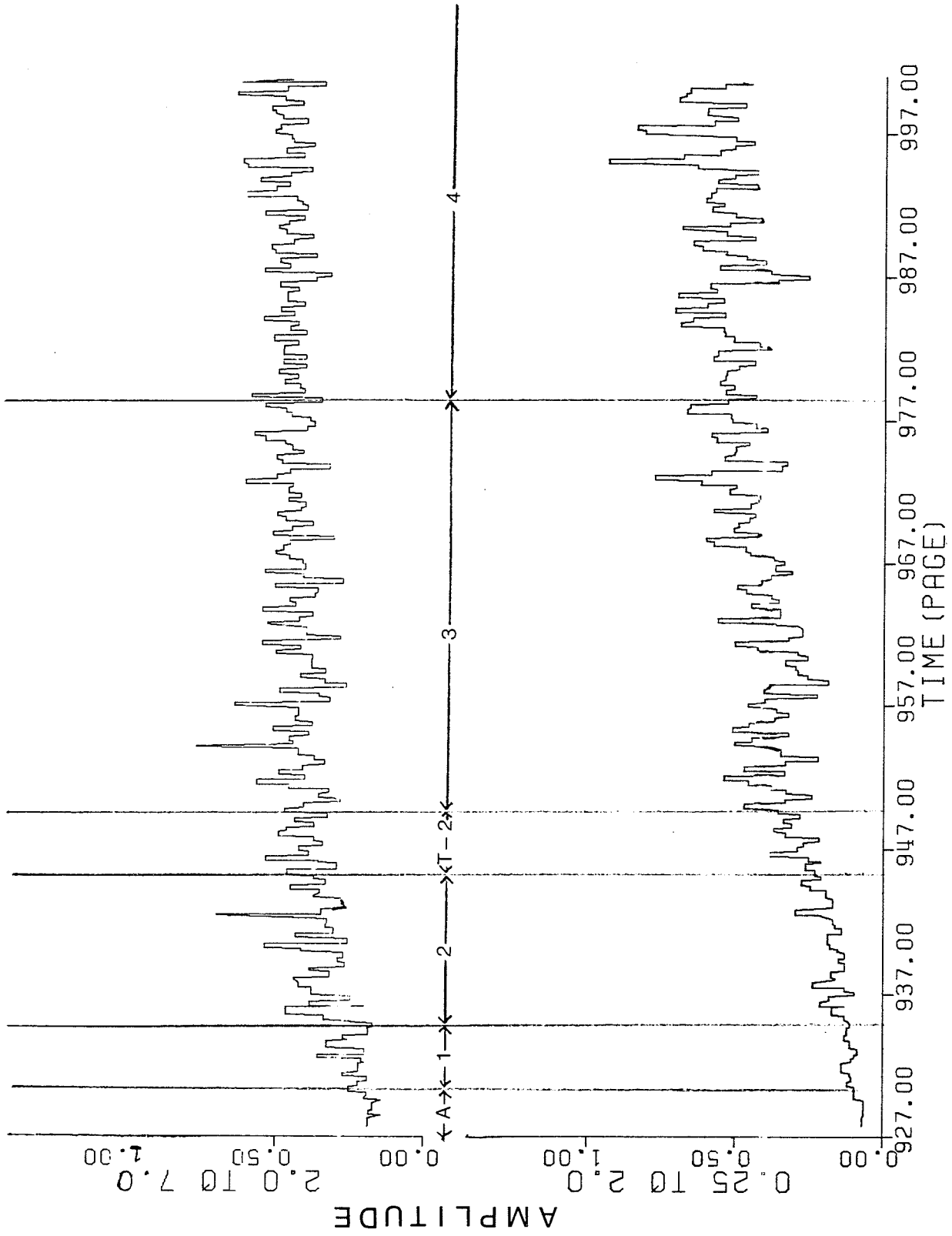


Figure 6.3: Spectral estimates with 10 second epoch rule

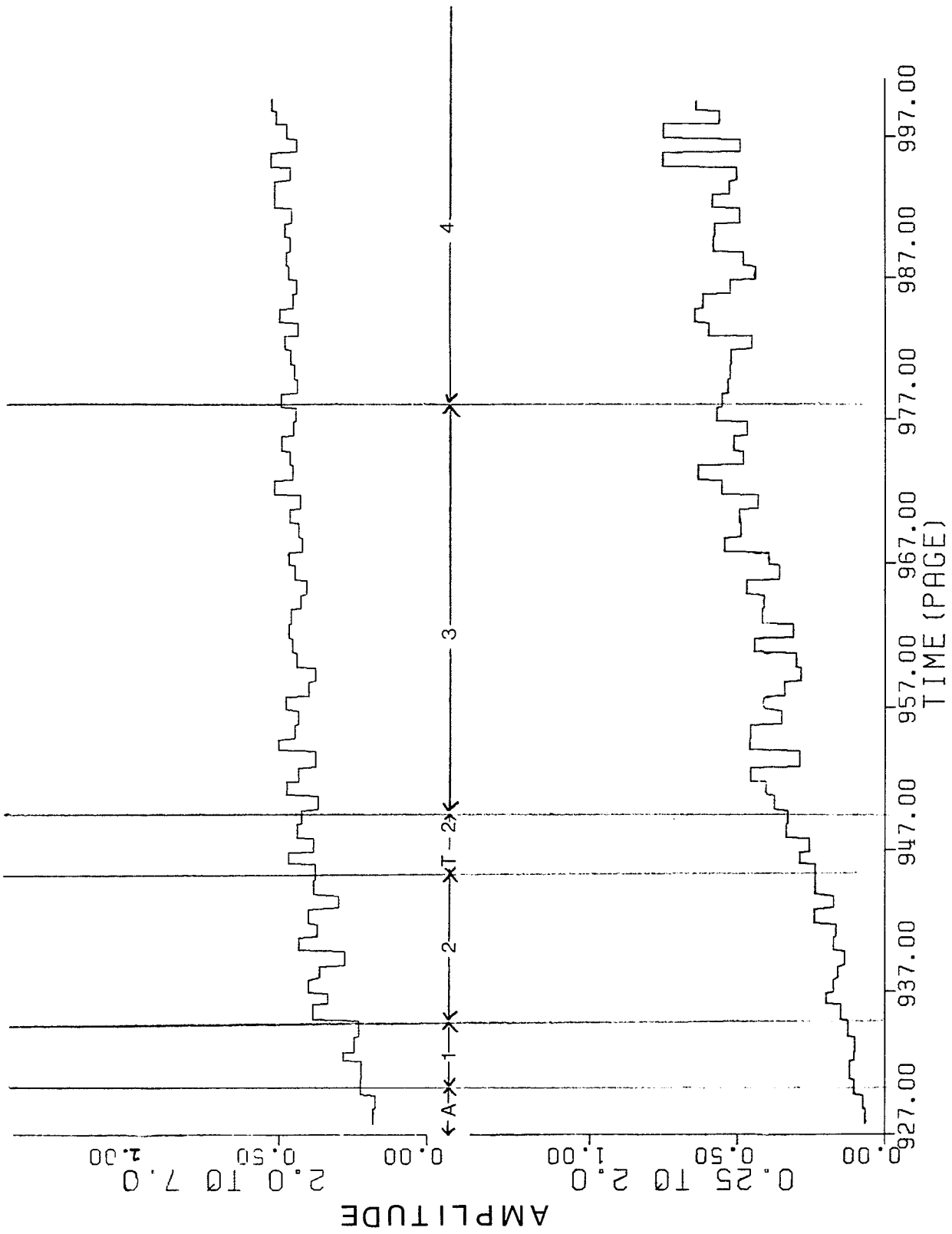


Figure 6.4: Spectral estimates with 30 second epoch rule

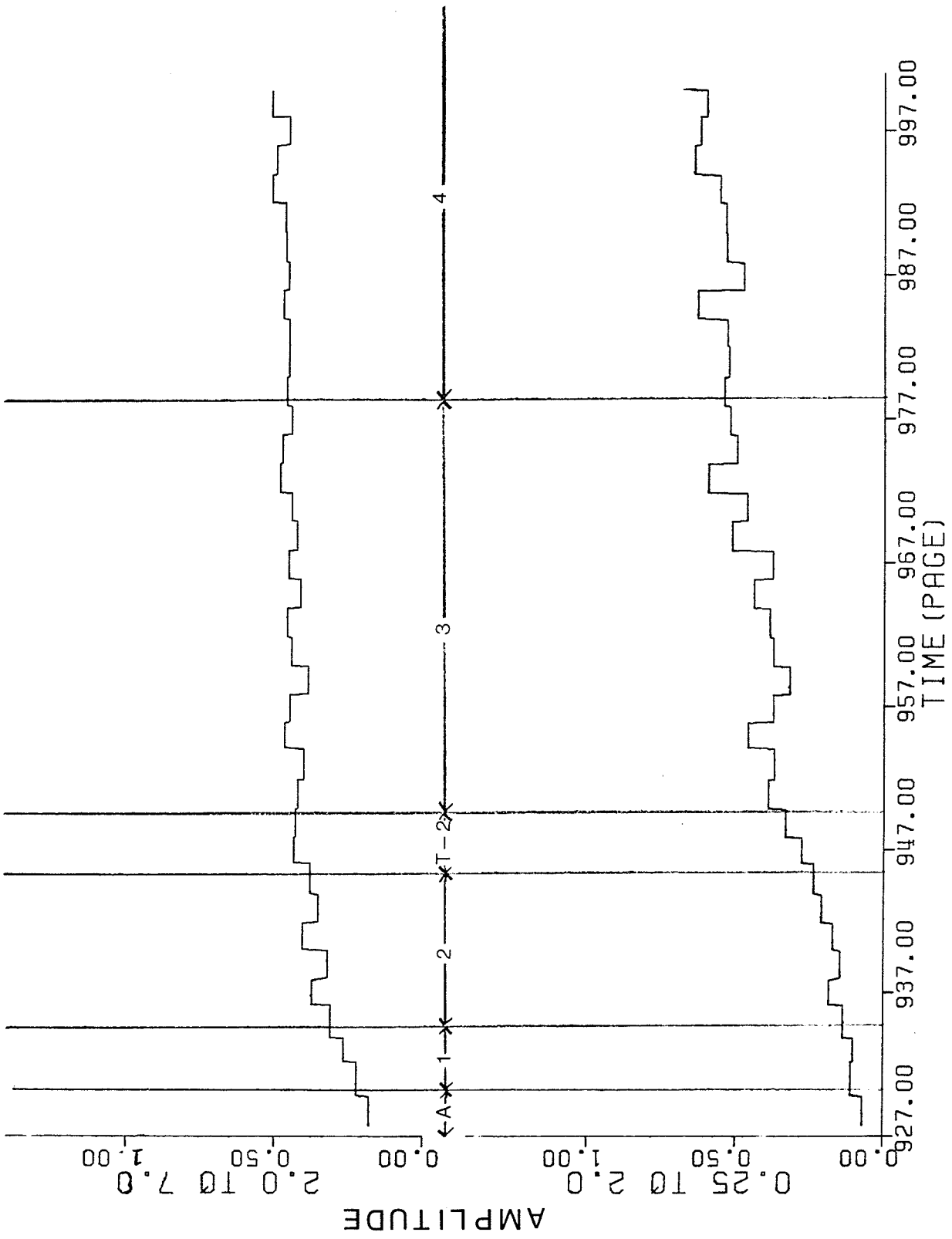


Figure 6.5: Spectral estimates with 60 second epoch rule

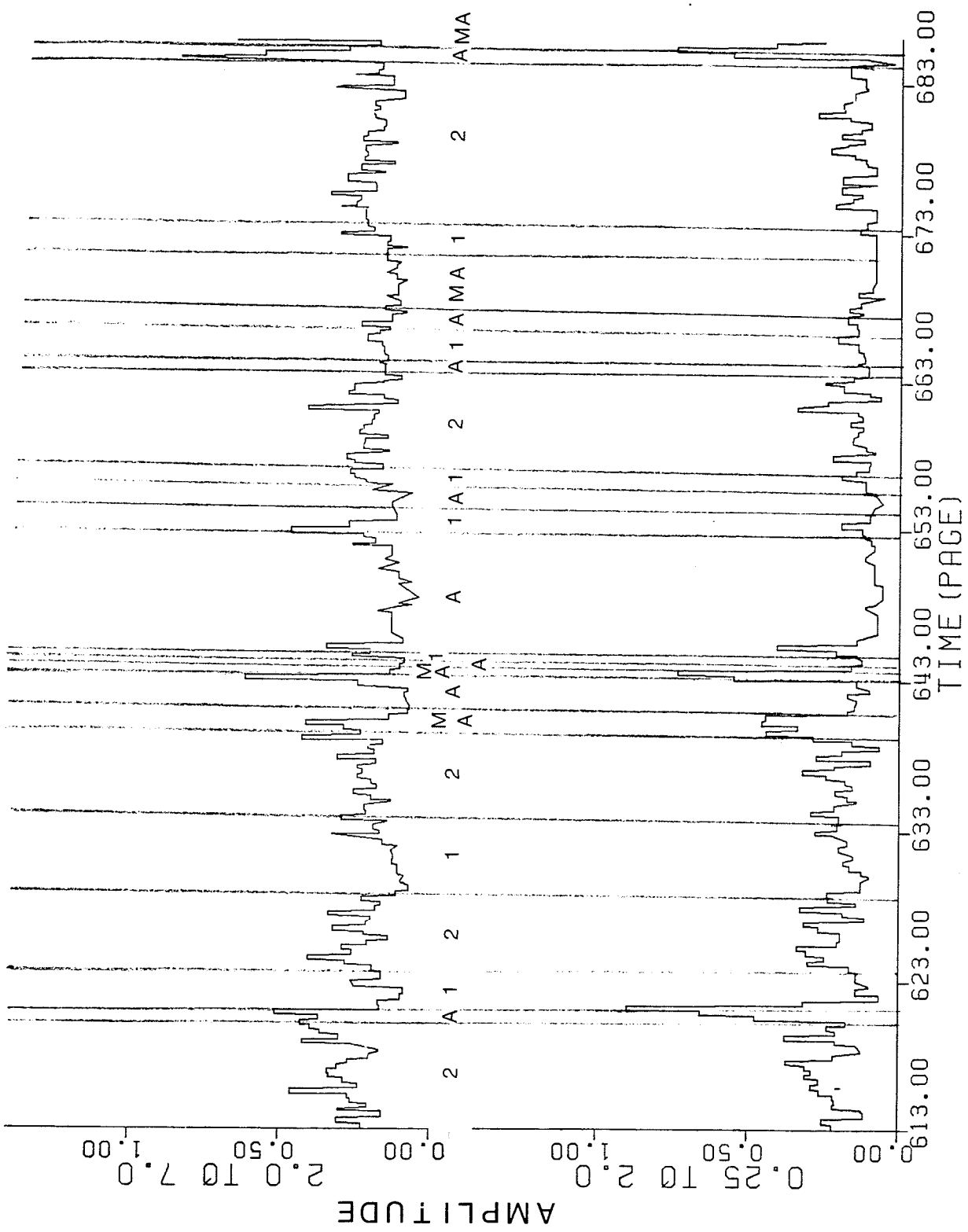


Figure 4.5: Spectral estimates during abnormal sleep (10 sec epoch)

cation and the spectral estimates is not at all clear. Past studies have justified the use of long averaging periods with the fact that normal sleep patterns see changes which occur only over long time intervals. The use of such an averaging rule may affect the ability of a system to identify a brief sleep segment. If a brief averaging rule is used however, the spectral estimates are not stationary. The close dependence on chosing an averaging period of 30 to 60 seconds in order to observe a stationary spectrum has not been reported and was somewhat unexpected.

It appears, therefore, that a tradeoff must occur in the use of an averaging period. For a given pattern recognition scheme, the use of a short averaging period will require the consideration of a large amount of detail in the classification of a normal, lengthy sleep stage interval. The application of a long averaging interval, however, may prevent the identification of a brief sleep stage segment (the averaging implied by the step responses of the filters estimating the spectrum may prevent such an identification). The evaluation of this problem must be integrated into a study considering a specific pattern recognition scheme. A variable averaging technique make be an avenue for consideration.

Chapter VII

CONCLUSIONS AND RECOMMENDATIONS

This study has focused on the development of a computer-aided method for sleep signal classification, and has considered the use of inexpensive microprocessing hardware for the task of sleep signal spectral estimation. It was found that, for normal sleep, the spectral estimates evaluated by the developed system correlated well with the manually classified sleep record. These estimates (as sampled once a second) were found to be non-stationary within the manually classified sleep stage segments, although a gross correlation between the estimates and the manual classification was still apparent. The application of averaging to the spectral estimates was examined, and it was found that the averaged spectral estimates became progressively more stationary with increasing averaging period. This averaging simplified the correlation between the manual classifications and the spectral estimates. An abnormal sleep record was also examined, and it was found that the application of even a brief averaging period could blur the correlation between the manual classification and the spectral estimates. This difficulty arose because disturbed sleep can present frequent sleep stage transitions. It appears, therefore, that a tradeoff must occur in the use of such an averaging or epoch rule.

As a result of this work, it is recommended that the use of syntactic pattern recognition for sleep signal classification be evaluated in a

scheme using the clinician as feedback to the system. This study will have to evaluate, for the specific recognition scheme, the use of a spectral averaging period. The use of a variable averaging period may be an avenue for consideration. Recent advances in pattern recognition, coupled with the approach developed in this study, may lead to an inexpensive scheme to aid in the classification of normal and abnormal sleep. In order to conduct this study, a clinical evaluation of the scheme will be required. It is strongly suggested that a library of recorded and manually classified sleep records be established for this purpose. Such a library will be invaluable to any further study of this subject.

Beyond the specific application considered, the developed method represents a general approach which should be applicable to a number of problems involving extended duration signal characterization and classification.

REFERENCES

1. Williams, R. L. et al, Electroencephalography (EEG) of Human Sleep: Clinical Applications. John Wiley & Sons, New York, 1974.

2. Guilleminault, C., and Demant, W. C., Sleep Apnea Syndromes. Alan R. Liss, Inc., New York, 1978.

3. Mezon et al, 'Sleep Breathing Abnormalities in Kyphoscololiosos', Am. Rev. Resp. Dis., 1980, 122:617.

4. Rechtschaffen, Allen, and Kales, Anthony, Editors, 'A Manual of Standardized Terminology, Techniques and Scoring System for Sleep Stages of Human Subjects', U. S. Department of Health, Bethesda, 1968.

5. Isaksson, Anders et al, 'Computer Analysis of EEG Signals with Parametric Models', Proc. IEEE, 1981, 69:451-461.

6. Bourne, J. R. et al, 'A Software System for Syntactic Analysis of the EEG', Computer Programs in Biomedicine, 1980, 11:190-200.

7. Gath, I. and Bar-on, E., 'Computerized Method for Scoring of Polygraphic Sleep Recordings', Computer Programs in Biomedicine, 1980, 11:217-223.

8. Smith, J. R. et al, 'Automated Sleep EEG Analysis Applied to the Evaluation of Drugs: Illustrated by Study of Chlorazepate Dipotassium', Electroenceph. clin. Neurophysiol., 1976, 41:587-594.

9. Smith, J. R. et al, 'Detection of Human Sleep EEG Waveforms', Electroenceph. clin. Neurophysiol., 1975, 38:435-437.

10. Su, Stanley Y. W. and Smith, J. R., 'Micro and Macro Analysis of Sleep Data Using Hybrid and Digital Computers', Computers and Biomedical Research, 1974, 7:432-488.

11. Martin, W. B. et al, 'Pattern Recognition of EEG-EOG as a Technique for All-Night Sleep Stage Scoring', Electroenceph. clin. Neurophysiol., 1972, 32:417-427.

12. Smith, J. R. and Karacan, Ismet, 'EEG Sleep Stage Scoring by an Automatic Hybrid System', Electroenceph. clin. Neurophysiol., 1971, 31:231-237.

13. Larsen, L. E. and Walter, D. O., 'On Automatic Methods of Sleep Staging of EEG Spectra', Electroenceph. clin. Neurophysiol., 1970, 28:459-467.

14. Itil, T. M. et al, 'Digital Computer Classification of EEG Sleep Stages', Electroenceph. clin. Neurophysiol., 1969, 27:76-83.

15. Smith, Jack R., 'Computers in Sleep Research', CRC Critical Reviews in Bioengineering, December, 1978.

16. Dement, W. and Kleitman, N., 'Cyclic variations in EEG during sleep and their relation to eye movements, body motility, and dreaming', Electroenceph. clin. Neurophysiol., 1957, 9:673-690.

17. Jasper, H. H., 'The ten-twenty electrode system', International Federation of Societies of Electroencephalography and Clinical Neurophysiology, Electroenceph. clin. Neurophysiol., 1958, 10:371-375.

18. Acres, John C. et al, 'Breathing During Sleep in Parents of Sudden Infant Death Syndrome Victims', in press: Am. Rev. Resp. Dis.

19. West, Peter, and Kryger, Meir H., 'Data Acquisition of Respiration Variables During Sleep by Microcomputer', in press.

20. Roessler, R. et al, 'A Period Analysis Classification of Sleep Stages', Electroenceph. clin. Neurophysiol., 1970, 29:358-362.

21. Agnew, Harman W. Jr. et al, 'Amplitude Measurement of the Sleep Electroencephalogram', Electroenceph. clin. Neurophysiol., 1967, 22:84-86.

22. Lubin, A. et al, 'Discrimination Among States of Consciousness using EEG Spectra', Psychophysiology, 1969, 6:122-132.

23. Silverstein, Louis D. and Levy, C. Michael, 'The Stability of the Sigma Spindle', Electroenceph. clin. Neurophysiol., 1976, 40:666-676.

24. Minard, James G. and Krausman, David, 'Rapid Eye Movement Definition and Count: An On-Line Detector', Electroenceph. clin. Neurophysiol., 1971, 31:99-102.

25. McPartland, Richard J. et al, 'Rapid Eye Movement Analyser', Electroenceph. clin. Neurophysiol., 1973, 34:317-320.

26. Gondeck, Allen R. and Smith, Jack R., 'Dynamics of Human Sleep Sigma Spindles', Electroenceph. clin. Neurophysiol., 1974, 37:293-297.

27. Giaquinto, S. and Marciano, F., 'Automatic Stimulation Triggered by EEG Spindles', Electroenceph. clin. Neurophysiol., 1971, 30:151-154.

28. Ktonas, Periklis Y., 'Semi-Automatic Analysis of Rapid Eye Movement (REM) Patterns: A Software Package', Computers and Biomedical Research, 1976, 9:109-124.

29. Degler, H. Edward Jr. et al, 'Automatic Detection and Resolution of Synchronous Rapid Eye Movements', Computers and Biomedical Re-

search, 1975, 8:393-404.

30. Smith, Jack R. et al, 'A Multichannel Hybrid System for Rapid Eye Movement Detection (REM Detection)', Computers and Biomedical Research, 1971, 4:275-290.

31. Bremer, Gordon et al, 'Automatic Detection of the K-Complex in Sleep Electroencephalogram', IEEE Trans. Biomed. Eng. 1970, 17:314-323.

32. Sanderson, Arthur C., 'Hierarchical Modeling of EEG Signals', IEEE Trans. PAMI, 1980, 2:405-415.

33. Intel Corporation, 2920 Analog Signal Processor Design Handbook. Santa Clara, 1980.

34. Kay, Steven M., and Marple, Lawrence Stanley Jr., 'Spectrum Analysis - A Modern Prospective', Proc. IEEE, 1981, 69:1380-1419.

35. Chen, Chi-Tsong, One-Dimensional Digital Signal Processing. Marcel Decker, Inc., New York, 1979.

36. Motorola Inc., M6800 Resident MPL Compiler Reference Manual. Austin, 1978.

37. Rabiner, Lawrence R. and Gold, Bernard, Theory and Application of Digital Signal Processing. Prentice-Hall, Englewood Cliffs, 1975.

38. Oppenheim, Alan V. and Schafer, Ronald W., Digital Signal Processing. Prentice-Hall, Englewood Cliffs, 1975.

39. Claasen, Theo A. C. et al, 'Effects of Quantization and Overflow in Recursive Digital Filters', IEEE Trans. Acoust., Speech, Signal Processing, 1976, 24:517-529.

40. Meerkotter, Klaus, and Wegener, Wilfried, 'A New Second-Order Digital Filter without Parasitic Oscillations', AEU, 1975, 29:312-314.

Appendix A

FILTER DERIVATIONS

BANDPASS FILTER

ω_1 = upper cutoff frequency [Hz]

ω_2 = lower cutoff frequency [Hz]

f = sampling frequency [Hz]

$$h = \frac{\cos[(\omega_1 + \omega_2)\ 2\pi/2(f)]}{\cos[(\omega_1 - \omega_2)\ 2\pi/2(f)]} \qquad (A.1)$$

$$\bar{\omega} = \frac{h - \cos[\omega_1(2\pi)/f]}{\sin[\omega_1(2\pi)/f]} \qquad (A.2)$$

Design a Butterworth Lowpass (2nd order) with cutoff at $\bar{\omega}$

$$H(s) = \frac{K}{s^2 + \sqrt{2}\,s + 1}$$ second order Butterworth with cutoff normalized (A.3)

$$H(s) = \frac{K'}{s^2 + \sqrt{2}\,\bar{\omega}\,s + \bar{\omega}^2}$$ second order Butterwork L.P. with cutoff at $\bar{\omega}$ (A.4)

$$= \frac{K'}{(s-m1)(s-m2)} \qquad (A.5)$$

where: $m1, m2 = \frac{-(\sqrt{2}\,\bar{\omega}) \pm \sqrt{(\sqrt{2}\,\bar{\omega})^2 - 4\bar{\omega}^2}}{2}$ (A.6)

TRANSFORMATION: Analog lowpass to digital bandpass

$$S = \frac{z^2 - 2hz + 1}{z^2 - 1} \qquad (A.7)$$

$$H(z) = \frac{K}{(\frac{z^2-2hz+1}{z^2-1} - m1)(\frac{z^2-2hz+1}{z^2-1} - m2)} \qquad (A.8)$$

$$H(z) = \frac{K(z^2-1)^2}{[(z^2-2h+1) - m1(z^2-1)]\cdot[(z^2-2hz+1)-m2(z^2-1)]} \qquad (A.9)$$

$$= \frac{K(z^2-1)^2}{[(1-m1)z^2 - 2h + (m1+1)]\cdot[(1-m2)z^2 - 2hz + (m2+1)]} \qquad (A.10)$$

$$= \frac{K(z^2-1)^2(\frac{1}{1-m1})(\frac{1}{1-m2})}{[z^2 - (\frac{2h}{1-m1})z + (\frac{1+m1}{1-m1})]\cdot[z^2 - (\frac{2h}{1-m2})z + (\frac{1+m2}{1-m2})]} \qquad (A.11)$$

$$= \frac{K'(z-1)(z+1)(z-1)(z+1)}{(z-\alpha_1)(z-\alpha_2)(z-\alpha_3)(z-\alpha_4)} \qquad (A.12)$$

where: $$\alpha_1, \alpha_2 = \frac{(\frac{-2h}{1-m1}) \pm \sqrt{(\frac{2h}{1-m1})^2 - 4(\frac{1+m1}{1-m1})}}{2} \qquad (A.13)$$

$$\alpha_3, \alpha_4 = \frac{(\frac{-2h}{1-m2}) \pm \sqrt{(\frac{2h}{1-m2})^2 - 4(\frac{1+m2}{1-m2})}}{2} \qquad (A.14)$$

Poles α_1 & α_3 and α_2 & α_4 are complex conjugates

BANDPASS FILTER TRANSFER FUNCTION:

$$H(z) = \frac{K(z-1)(z+1)}{z^2 - (\alpha_1+\alpha_3)z + \alpha_1\alpha_3} \cdot \frac{(z-1)(z+1)}{z^2 - (\alpha_2+\alpha_4)z + \alpha_2\alpha_4} \qquad (A.15)$$

LOW PASS FILTER

cutoff frequency = ωp [Hz]

sampling frequency = fs [Hz]

$$\bar{\omega p} = \tan \frac{\omega p(2\pi)}{2(fs)} = x \qquad (A.16)$$

$$\bar{H}(s) = \frac{1}{(\frac{s}{x})^2 + \frac{1.414}{x} s + 1} = \frac{x^2}{s^2 + \sqrt{2}\, x\, s + x^2} \qquad (A.17)$$

BILINEAR TRANSFORM: $s = \frac{z-1}{z+1}$ (A.18)

$$= \frac{x^2}{(\frac{z-1}{z+1})^2 + \sqrt{2}\, x\, (\frac{z-1}{z+1}) + x^2} \qquad (A.19)$$

$$= \frac{x^2(z+1)^2}{(z-1)^2 + \sqrt{2}\, x\, (z-1)(z+1) + x^2(z+1)^2} \qquad (A.20)$$

$$= \frac{x^2(z+1)(z+1)}{(z-1)(z-1) + \sqrt{2}\, x\, (z-1)(z+1) + x^2(z+1)(z+1)} \qquad (A.21)$$

$$= \frac{x^2(z^2+2z+1)}{(z^2-2z+1) + \sqrt{2}\,x\,(z^2-1) + x^2(z^2+2z+1)} \qquad (A.22)$$

$$= \frac{x^2(z^2+2z+1)}{(x^2+\sqrt{2}\,x+1)z^2 + (2x^2-2)z + (x^2-\sqrt{2}\,x+1)} \qquad (A.23)$$

$$= \frac{\dfrac{x^2}{(x^2+\sqrt{2}\,x+1)} \cdot (z^2+2z+1)}{z^2 + \dfrac{2x^2-2}{(x^2+\sqrt{2}\,x+1)}\,z + \dfrac{x^2-\sqrt{2}\,x+1}{x^2+\sqrt{2}\,x+1}} \qquad (A.24)$$

$$= \frac{K(z^2 + 2z + 1)}{z^2 + \alpha\, z + \beta} \qquad (A.25)$$

where: $$\alpha = \frac{2x^2 - 2}{x^2 + \sqrt{2}\,x + 1} \qquad (A.26)$$

$$\beta = \frac{x^2 - \sqrt{2}\,x + 1}{x^2 = \sqrt{2}\,x + 1} \qquad (A.27)$$

Appendix B

SOFTWARE LISTINGS

```
      $JOB  WATFIV   REIMER,NOEXT
      C
      C ***************************************************************
      C *                                                             *
      C * PROGRAM TO CALCULATE BANDPASS AND LOWPASS FILTER COEFFICIENTS *
      C * AND EXPERIMENTALLY EVALUATE SCALING FACTORS TO THE NEAREST    *
      C * POWER OF 2 TO AVOID SATURATIONS IN THE FILTER CALCULATIONS    *
      C *                                                             *
      C ***************************************************************
      C
  1         REAL W1,W2,F,WP,FS,A,B,C,D,ALP,BET,FIN,K1,K2,K3
  2         INTEGER N,I
  3         READ,N
  4         DO 999 I=1,N
  5           READ,W2,W1,F,WP,FS,FIN
  6           CALL BAND(W1,W2,F,A,B,C,D)
  7           CALL LOW(WP,FS,ALP,BET)
  8           CALL SCALE(A,B,C,D,F,ALP,BET,FS,FIN,K1,K2,K3,K4)
  9           PRINT 401,IFIX(K1)
 10           PRINT 402,IFIX(K2)
 11           PRINT 403,IFIX(K3)
 12           PRINT 404,K4
 13   401     FORMAT(' ','K1=2**',I3)
 14   402     FORMAT(' ','K2=2**',I3)
 15   403     FORMAT(' ','K3=2**',I3)
 16   404     FORMAT(' ','LARGEST OBSERVED OUTPUT = ',F9.7)
 17   999   CONTINUE
 18         STOP
 19         END
      C
      C ***************************************************************
      C *                                                             *
      C * PROGRAM TO COMPUTE THE COEFFICIENTS FOR A FOURTH ORDER      *
      C * BUTTERWORTH DIGITAL BANDPASS FILTER USING THE BILINEAR      *
      C * ANALOG LOWPASS TO DIGITAL BANDPASS METHOD.                  *
      C *                                                             *
      C ***************************************************************
      C
 20         SUBROUTINE BAND(W1,W2,F,A,B,C,D)
 21         COMPLEX M1,M2,TEMP2,TEMP3,ALP1,ALP2,ALP3,ALP4
 22         REAL W1,W2,DIG,H,PI,F,TEMP0,TEMP1,A,B,C,D
 23         PRINT 100
 24   100   FORMAT('1')
 25         PRINT,'BANDPASS FILTER'
 26         PRINT 101,W2
 27   101   FORMAT(' ','LOWER CUTOFF FREQUENCY = ',F9.5)
 28         PRINT 102,W1
 29   102   FORMAT(' ','UPPER CUTOFF FREQUENCY = ',F9.5)
 30         PRINT 103,F
 31   103   FORMAT(' ','SAMPLING FREQUENCY = ',F6.1)
 32         PI=3.14159265
 33         TEMP0=((W1+W2)*2*PI)/(2*F)
 34         TEMP1=((W1-W2)*2*PI)/(2*F)
 35         H=(COS(TEMP0))/(COS(TEMP1))
 36         TEMP0=(W1*2*PI)/F
 37         DIG=(H-(COS(TEMP0)))/(SIN(TEMP0))
 38         TEMP0=(SQRT(2.0))*DIG
 39         TEMP1=(TEMP0**2)-(4*(DIG**2))
 40         TEMP3=CMPLX(TEMP1,0.0)
 41         M1=((-TEMP0)+CSQRT(TEMP3))/2.0
 42         M2=((-TEMP0)-CSQRT(TEMP3))/2.0
 43         TEMP2=(2*H)/(1-M1)
 44         TEMP3=(TEMP2**2)-(4*((1+M1)/(1-M1)))
 45         ALP1=(TEMP2+CSQRT(TEMP3))/2.0
 46         ALP2=(TEMP2-CSQRT(TEMP3))/2.0
 47         TEMP2=(2*H)/(1-M2)
 48         TEMP3=(TEMP2**2)-(4*((1+M2)/(1-M2)))
 49         ALP3=(TEMP2+CSQRT(TEMP3))/2.0
 50         ALP4=(TEMP2-CSQRT(TEMP3))/2.0
 51         TEMP0=REAL(ALP1)
 52         TEMP1=AIMAG(ALP1)
 53         A=-(2*TEMP0)
 54         B=(TEMP0**2)+(TEMP1**2)
 55         PRINT 104,A
 56   104   FORMAT('0','A = ',F10.7)
 57         TEMP9=-A
 58         CALL SIGN(TEMP9)
 59         PRINT 105,B
 60   105   FORMAT('0','B = ',F10.7)
 61         TEMP9=B
 62         CALL SIGN(TEMP9)
 63         TEMP0=REAL(ALP2)
 64         TEMP1=AIMAG(ALP2)
 65         C=-(2*TEMP0)
 66         D=(TEMP0**2)+(TEMP1**2)
 67         PRINT 106,C
 68   106   FORMAT('0','C = ',F10.7)
 69         TEMP9=-C
 70         CALL SIGN(TEMP9)
 71         PRINT 107,D
 72   107   FORMAT('0','D = ',F10.7)
 73         TEMP9=D
 74         CALL SIGN(TEMP9)
 75         RETURN
 76         END
```

```
      C
      C ***************************************************************
      C *                                                             *
      C * SUBROUTINE TO LIST THE SIGN DIGIT CANONIC FORM OF A NUMBER  *
      C *                                                             *
      C ***************************************************************
      C
 77         SUBROUTINE SIGN(A)
 78         REAL*8 TABLE(16),DIFF,DIFF1,DIFF2
 79         INTEGER I,J,NUM(16)
 80         PRINT 9, A
 81    9    FORMAT('0','SIGN DIGIT BREAKDOWN OF ',F11.7)
 82         NUM(1)=2
 83         TABLE(1)=4.0
 84         I=1
 85         DO 1 J=2,16
 86           TABLE(J)=TABLE(J-1)*0.5
 87           NUM(J)=I
 88           I=I-1
 89    1    CONTINUE
 90         DIFF=A
 91         WHILE(DABS(DIFF) .GT. .0001) DO
 92           I=16
 93           WHILE(TABLE(I) .LT. DABS(DIFF)) DO
 94             I=I-1
 95           END WHILE
 96           IF(I .EQ. 16) GO TO 2
 97           DIFF1=DABS(TABLE(I) - DABS(DIFF))
 98           DIFF2=DABS(TABLE(I+1) - DABS(DIFF))
 99           IF(DIFF2 .LT. DIFF1) I=I+1
100    2      IF(DIFF .GT. 0) THEN DO
101             PRINT 10,NUM(I)
102   10        FORMAT(' ',5X,'ADD 2 TO THE POWER ',I3)
103             DIFF=DIFF-TABLE(I)
104           ELSE DO
105             PRINT 30,NUM(I)
106   30        FORMAT(' ',5X,'SUBTRACT 2 TO THE POWER ',I3)
107             DIFF=TABLE(I)+DIFF
108           END IF
109         END WHILE
110         RETURN
111         END
      C
      C ***************************************************************
      C *                                                             *
      C * PROGRAM TO CALCULATE THE COEFFICIENTS FOR A SECOND ORDER    *
      C * BUTTERWORTH LOWPASS SECOND ORDER SECTION                    *
      C *                                                             *
      C ***************************************************************
      C
112         SUBROUTINE LOW(WP,FS,ALP,BET)
113         REAL X,ALP,BET,PI,FS,WP,TEMP
114         PRINT,' '
115         PRINT, 'LOWPASS FILTER'
116         PRINT 200, WP
117   200   FORMAT(' ','BREAK FREQUENCY  =',F10.4)
118         PRINT 201, FS
119   201   FORMAT(' ','SAMPLE FREQUENCY =',F10.4)
120         PI=3.14159
121         X=(WP*2*PI)/(2*FS)
122         X=TAN(X)
123         TEMP=1+(1.414*X)+(X**2)
124         ALP=((2*(X**2))-2)/TEMP
125         PRINT 202,ALP
126   202   FORMAT('0','ALPHA = ',F10.7)
127         TEMP1=-ALP
128         CALL SIGN(TEMP1)
129         BET=(1-(1.414*X)+(X**2))/TEMP
130         PRINT 203,BET
131   203   FORMAT('0','BETA = ',F10.7)
132         TEMP1=BET
133         CALL SIGN(TEMP1)
134         RETURN
135         END
      C
      C ***************************************************************
      C *                                                             *
      C * PROGRAM TO EXPERIMENTALLY EVALUATE THE SCALING FACTORS      *
      C * REQUIRED TO AVOID OVERFLOWS (SATURATIONS) IN THE BANDPASS   *
      C * LOWPASS FILTER SETS. THE SCALING FACTORS ARE EVALUATED TO THE *
      C * NEAREST POWER OF 2 WHICH RESULTS IN NO SATURATION.          *
      C *                                                             *
      C ***************************************************************
      C
136         SUBROUTINE SCALE(A,B,C,D,FREQ,ALP,BET,FS,FREQIN,
           *K1,K2,K3,K4)
137         REAL A,B,C,D,D1,D2,D3,D4,FREQIN,FREQ,X
138         REAL INCREM,K1,K2,K3,K4,DX,DY,ALP,BET
139         INTEGER NMAX,N,I,CNTL,LAST,CNTL1,NUM,NUMEND
140         K1=1
141         K2=1
142         K3=1
143         NMAX=IFIX(FREQ/FREQIN)
144         INCREM=(2*3.14159)/NMAX
145         NUMEND=IFIX(FREQ/FS)
146         CNTL1=1
147    1    N=0
148         X=0
149         NUM=0
150         D1=0
```

```
151           D2=0
152           D3=0
153           D4=0
154           DX=0
155           DY=0
156           CNTL=0
157           LAST=10*NMAX
158           K4=0
159           DO 10 I=1,LAST
160             CALL CYCLE(CNTL,A,B,C,D,K1,K2,K3,D1,D2,D3,D4,
              *  N,X,NMAX,INCREM,NUM,NUMEND,DX,DY,ALP,BET,K4)
161     10    CONTINUE
162           CNTL=CNTL1
163           K4=0
164           DO 20 I=1,LAST
165             CALL CYCLE(CNTL,A,B,C,D,K1,K2,K3,D1,D2,D3,D4,
              *  N,X,NMAX,INCREM,NUM,NUMEND,DX,DY,ALP,BET,K4)
166             IF (CNTL .EQ. 99) GO TO 99
167             IF (CNTL .EQ. 98) GO TO 98
168             IF (CNTL .EQ. 97) GO TO 97
169     20    CONTINUE
170           CNTL1=CNTL1+1
171           IF (CNTL1 .EQ. 4) RETURN
172           GO TO 1
173     99    K1=K1-1
174           IF (K1 .EQ. -14) THEN DO
175             PRINT,'K1 SCALING POWER LESS THAN 2**-14'
176             GO TO 96
177           END IF
178           GO TO 1
179     98    K2=K2-1
180           IF (K2 .EQ. -14) THEN DO
181             PRINT,'K2 SCALING POWER LESS THAN 2**-14'
182             GO TO 96
183               END IF
184               GO TO 1
185     97        K3=K3-1
186               IF (K3 .EQ. -14) THEN DO
187                 PRINT,'K3 SCALING POWER LESS THAN 2**-14'
188                 GO TO 96
189               END IF
190               GO TO 1
191     96        PRINT,'SCALING ABORTED'
192               RETURN
193               END
      C
      C ***************************************************************
      C *                                                             *
      C * SUBROUTINE TO SIMULATE A DELAY ITERATION OF THE DIGITAL FILTERS*
      C *                                                             *
      C ***************************************************************
      C
194               SUBROUTINE CYCLE(CNTL,A,B,C,D,K1,K2,K3,D1,D2,D3,D4,
                 *N,X,NMAX,INCREM,NUM,NUMEND,DX,DY,ALP,BET,K4)
195               INTEGER CNTL,NMAX,N,NUM,NUMEND
196               REAL A,B,C,D,VAL,K1,K2,K3,K4,D1,D2,D3,D4,X,INCREM
197               REAL TEMP1,TEMP2,TEMP3,TEMP4,TEMP5,DX,DY,ALP,BET
198               VAL=SIN(X)
199               N=N+1
200               X=X+INCREM
201           IF (N .EQ. NMAX) THEN DO
202             N=0
203             X=0
204           END IF
205           TEMP1=D1*A
206           IF (TEMP1 .GT.  1) TEMP1=1
207           IF (TEMP1 .LT. -1) TEMP1=-1
208           TEMP2=D2*B
209           IF (TEMP2 .GT.  1) TEMP2=1
210           IF (TEMP2 .LT. -1) TEMP2=-1
211           TEMP3=(-TEMP1)+(-TEMP2)
212           IF (TEMP3 .GT.  1) TEMP3=1
213           IF (TEMP3 .LT. -1) TEMP3=-1
214           TEMP4=(VAL*(2.0**K1))+TEMP3
215           IF (TEMP4 .GT.  1) TEMP4=1
216           IF (TEMP4 .LT. -1) TEMP4=-1
217           TEMP5=TEMP4-D2
218           IF (TEMP5 .GT.  1) TEMP5=1
219           IF (TEMP5 .LT. -1) TEMP5=-1
220           IF (CNTL .EQ. 1) THEN DO
221             IF (ABS(TEMP1) .GE. 1) CNTL=99
222             IF (ABS(TEMP2) .GE. 1) CNTL=99
223             IF (ABS(TEMP3) .GE. 1) CNTL=99
224             IF (ABS(TEMP4) .GE. 1) CNTL=99
225             IF (ABS(TEMP5) .GE. 1) CNTL=99
226           END IF
227           D2=D1
228           D1=TEMP4
229           TEMP1=D3*C
230           IF (TEMP1 .GT.  1 ) TEMP1=1
231           IF (TEMP1 .LT. -1) TEMP1=-1
232           TEMP2=D4*D
233           IF (TEMP2 .GT.  1 ) TEMP2=1
234           IF (TEMP2 .LT. -1) TEMP2=-1
235           TEMP3=(-TEMP1)+(-TEMP2)
236           IF (TEMP3 .GT.  1 ) TEMP3=1
237           IF (TEMP3 .LT. -1) TEMP3=-1
238           TEMP4=(TEMP5*(2.0**K2))+TEMP3
239           IF (TEMP4 .GT.  1 ) TEMP4=1
240           IF (TEMP4 .LT. -1) TEMP4=-1
```

```
241           TEMP5=TEMP4-D4
242           IF (TEMP5 .GT.  1 ) TEMP5=1
243           IF (TEMP5 .LT. -1) TEMP5=-1
244           IF (CNTL .EQ. 2) THEN DO
245             IF (ABS(TEMP1) .GE. 1) CNTL=98
246             IF (ABS(TEMP2) .GE. 1) CNTL=98
247             IF (ABS(TEMP3) .GE. 1) CNTL=98
248             IF (ABS(TEMP4) .GE. 1) CNTL=98
249             IF (ABS(TEMP5) .GE. 1) CNTL=98
250           END IF
251           D4=D3
252           D3=TEMP4
253           NUM=NUM+1
254           IF (NUM .EQ. NUMEND) THEN DO
255             NUM=0
256             TEMP1=ALP*DX
257             IF (TEMP1 .GT.  1 ) TEMP1=1
258             IF (TEMP1 .LT. -1) TEMP1=-1
259             TEMP2=BET*DY
260             IF (TEMP2 .GT.  1 ) TEMP2=1
261             IF (TEMP2 .LT. -1) TEMP2=-1
262             TEMP3=(-TEMP1)+(-TEMP2)
263             IF (TEMP3 .GT.  1 ) TEMP3=1
264             IF (TEMP3 .LT. -1) TEMP3=-1
265             TEMP5=ABS(TEMP5)
266             TEMP4=(TEMP5*(2.0**K3))+TEMP3
267             IF (TEMP4 .GT.  1 ) TEMP4=1
268             IF (TEMP4 .LT. -1) TEMP4=-1
269             TEMP5=(DX*2)+DY+TEMP4
270             IF (TEMP5 .GT.  1 ) TEMP5=1
271             IF (TEMP5 .LT. -1) TEMP5=-1
272             DY=DX
273             DX=TEMP4
274             IF (CNTL .EQ. 3) THEN DO
275               IF (ABS(TEMP1) .GE. 1) CNTL=97
276               IF (ABS(TEMP2) .GE. 1) CNTL=97
277               IF (ABS(TEMP3) .GE. 1) CNTL=97
278               IF (ABS(TEMP4) .GE. 1) CNTL=97
279               IF (ABS(TEMP5) .GE. 1) CNTL=97
280             END IF
281             IF (K4 .LT. TEMP5) K4=TEMP5
282           END IF
283           RETURN
284           END
          $ENTRY
```

```
BANDPASS FILTER
LOWER CUTOFF FREQUENCY =    0.25000
UPPER CUTOFF FREQUENCY =    2.00000
SAMPLING FREQUENCY =    50.0
A =  -1.9585290
SIGN DIGIT BREAKDOWN OF    1.9585290
     ADD 2 TO THE POWER    1
     SUBTRACT 2 TO THE POWER   -5
     SUBTRACT 2 TO THE POWER   -7
     SUBTRACT 2 TO THE POWER   -9
     SUBTRACT 2 TO THE POWER  -11
B =   0.9597065
SIGN DIGIT BREAKDOWN OF    0.9597065
     ADD 2 TO THE POWER    0
     SUBTRACT 2 TO THE POWER   -5
     SUBTRACT 2 TO THE POWER   -7
     SUBTRACT 2 TO THE POWER  -10
     SUBTRACT 2 TO THE POWER  -12
C =  -1.7178450
SIGN DIGIT BREAKDOWN OF    1.7178450
     ADD 2 TO THE POWER    1
     SUBTRACT 2 TO THE POWER   -2
     SUBTRACT 2 TO THE POWER   -5
     SUBTRACT 2 TO THE POWER  -10
D =   0.7634901
SIGN DIGIT BREAKDOWN OF    0.7634901
     ADD 2 TO THE POWER    0
     SUBTRACT 2 TO THE POWER   -2
     ADD 2 TO THE POWER   -6
     SUBTRACT 2 TO THE POWER   -9
     SUBTRACT 2 TO THE POWER  -13

LOWPASS FILTER
BREAK FREQUENCY  =     0.0250
SAMPLE FREQUENCY =    12.5000
ALPHA =  -1.9822320
SIGN DIGIT BREAKDOWN OF    1.9822320
     ADD 2 TO THE POWER    1
     SUBTRACT 2 TO THE POWER   -6
     SUBTRACT 2 TO THE POWER   -9
     SUBTRACT 2 TO THE POWER  -12
BETA =   0.9823886
SIGN DIGIT BREAKDOWN OF    0.9823886
     ADD 2 TO THE POWER    0
     SUBTRACT 2 TO THE POWER   -6
     SUBTRACT 2 TO THE POWER   -9
K1=2** -9
K2=2** -1
K3=2**-11
```

```
LARGEST OBSERVED OUTPUT = 0.6824680
BANDPASS FILTER
LOWER CUTOFF FREQUENCY =    2.00000
UPPER CUTOFF FREQUENCY =    7.00000
SAMPLING FREQUENCY =    50.0
A = -1.7093410
SIGN DIGIT BREAKDOWN OF    1.7093410
      ADD 2 TO THE POWER    1
      SUBTRACT 2 TO THE POWER   -2
      SUBTRACT 2 TO THE POWER   -5
      SUBTRACT 2 TO THE POWER   -7
      SUBTRACT 2 TO THE POWER   -9
      ADD 2 TO THE POWER -12
      ADD 2 TO THE POWER -13
B =  0.7812762
SIGN DIGIT BREAKDOWN OF    0.7812762
      ADD 2 TO THE POWER    0
      SUBTRACT 2 TO THE POWER   -2
      ADD 2 TO THE POWER   -5
C = -1.0809290
SIGN DIGIT BREAKDOWN OF    1.0809290
      ADD 2 TO THE POWER    0
      ADD 2 TO THE POWER   -4
      ADD 2 TO THE POWER   -6
      ADD 2 TO THE POWER   -9
      ADD 2 TO THE POWER -10
      SUBTRACT 2 TO THE POWER -13
D =  0.5283684
SIGN DIGIT BREAKDOWN OF    0.5283684
      ADD 2 TO THE POWER   -1
      ADD 2 TO THE POWER   -5
      SUBTRACT 2 TO THE POWER   -9
      SUBTRACT 2 TO THE POWER -10

LOWPASS FILTER
BREAK FREQUENCY  =     0.2000
SAMPLE FREQUENCY =    50.0000
ALPHA = -1.9644670
SIGN DIGIT BREAKDOWN OF    1.9644670
      ADD 2 TO THE POWER    1
      SUBTRACT 2 TO THE POWER   -5
      SUBTRACT 2 TO THE POWER   -8
      SUBTRACT 2 TO THE POWER -11
      ADD 2 TO THE POWER -13
BETA =  0.9650878
SIGN DIGIT BREAKDOWN OF    0.9650878
      ADD 2 TO THE POWER    0
      SUBTRACT 2 TO THE POWER   -5
      SUBTRACT 2 TO THE POWER   -8
      ADD 2 TO THE POWER -12
K1=2** -4
K2=2**  0
K3=2**-12
LARGEST OBSERVED OUTPUT = 0.9679918
BANDPASS FILTER
LOWER CUTOFF FREQUENCY =   12.00000
UPPER CUTOFF FREQUENCY =   14.00000
SAMPLING FREQUENCY =   200.0
A = -1.8149230
SIGN DIGIT BREAKDOWN OF    1.8149230
      ADD 2 TO THE POWER    1
      SUBTRACT 2 TO THE POWER   -3
      SUBTRACT 2 TO THE POWER   -4
      ADD 2 TO THE POWER   -9
      ADD 2 TO THE POWER -11
B =  0.9587363
SIGN DIGIT BREAKDOWN OF    0.9587363
      ADD 2 TO THE POWER    0
      SUBTRACT 2 TO THE POWER   -5
      SUBTRACT 2 TO THE POWER   -7
      SUBTRACT 2 TO THE POWER   -9
      SUBTRACT 2 TO THE POWER -12
C = -1.7763690
SIGN DIGIT BREAKDOWN OF    1.7763690
      ADD 2 TO THE POWER    1
      SUBTRACT 2 TO THE POWER   -2
      ADD 2 TO THE POWER   -5
      SUBTRACT 2 TO THE POWER   -8
      SUBTRACT 2 TO THE POWER -10
D =  0.9543562
SIGN DIGIT BREAKDOWN OF    0.9543562
      ADD 2 TO THE POWER    0
      SUBTRACT 2 TO THE POWER   -5
      SUBTRACT 2 TO THE POWER   -6
      ADD 2 TO THE POWER -10
      ADD 2 TO THE POWER -12

LOWPASS FILTER
BREAK FREQUENCY  =     1.0000
SAMPLE FREQUENCY =   200.0000
ALPHA = -1.9555860
SIGN DIGIT BREAKDOWN OF    1.9555860
      ADD 2 TO THE POWER    1
      SUBTRACT 2 TO THE POWER   -5
      SUBTRACT 2 TO THE POWER   -6
      ADD 2 TO THE POWER   -9
      ADD 2 TO THE POWER -11
BETA =  0.9565516
SIGN DIGIT BREAKDOWN OF    0.9565516
      ADD 2 TO THE POWER    0
```

```
      SUBTRACT 2 TO THE POWER  -5
      SUBTRACT 2 TO THE POWER  -6
      ADD 2 TO THE POWER  -8
      SUBTRACT 2 TO THE POWER -11
K1=2** -7
K2=2** -5
K3=2**-10
LARGEST OBSERVED OUTPUT = 0.6799301
BANDPASS FILTER
LOWER CUTOFF FREQUENCY =    2.00000
UPPER CUTOFF FREQUENCY =   10.00000
SAMPLING FREQUENCY =    50.0
A = -1.6701810
SIGN DIGIT BREAKDOWN OF    1.6701810
      ADD 2 TO THE POWER   1
      SUBTRACT 2 TO THE POWER  -2
      SUBTRACT 2 TO THE POWER  -4
      SUBTRACT 2 TO THE POWER  -6
      SUBTRACT 2 TO THE POWER  -9
      ADD 2 TO THE POWER -12
B =   0.7390358
SIGN DIGIT BREAKDOWN OF    0.7390358
      ADD 2 TO THE POWER  -1
      ADD 2 TO THE POWER  -2
      SUBTRACT 2 TO THE POWER  -7
      SUBTRACT 2 TO THE POWER  -8
      ADD 2 TO THE POWER -10
      SUBTRACT 2 TO THE POWER -12
C = -0.5517535
SIGN DIGIT BREAKDOWN OF    0.5517535
      ADD 2 TO THE POWER  -1
      ADD 2 TO THE POWER  -4
      SUBTRACT 2 TO THE POWER  -7
      SUBTRACT 2 TO THE POWER  -8
      ADD 2 TO THE POWER -10
D =   0.3414242
SIGN DIGIT BREAKDOWN OF    0.3414242
      ADD 2 TO THE POWER  -2
      ADD 2 TO THE POWER  -4
      ADD 2 TO THE POWER  -5
      SUBTRACT 2 TO THE POWER  -9
      SUBTRACT 2 TO THE POWER -11
      ADD 2 TO THE POWER -13

LOWPASS FILTER
BREAK FREQUENCY  =     0.2000
SAMPLE FREQUENCY =    50.0000
ALPHA = -1.9644670
SIGN DIGIT BREAKDOWN OF    1.9644670
      ADD 2 TO THE POWER   1
      SUBTRACT 2 TO THE POWER  -5
      SUBTRACT 2 TO THE POWER  -8
      SUBTRACT 2 TO THE POWER -11
      ADD 2 TO THE POWER -13
BETA =   0.9650878
SIGN DIGIT BREAKDOWN OF    0.9650878
      ADD 2 TO THE POWER   0
      SUBTRACT 2 TO THE POWER  -5
      SUBTRACT 2 TO THE POWER  -8
      ADD 2 TO THE POWER -12
K1=2** -3
K2=2**  0
K3=2**-12
LARGEST OBSERVED OUTPUT = 0.9086537
```

```
*************************************************************
*                                                           *
* PROGRAM OPERATING IN 2920 NUMBER 1- EEG TAKEN AS INPUT AT *
* A RATE OF 6.5 KHZ, WITH BANDPASS/FW RECT/LOWPASS FILTERS  *
* OPERATING AT 200, 50, & 12.5 HZ.  THIS 2920 ALSO PRODUCES *
* THE SHIFT CONTROL PULSE WHICH PERMITS THE SHIFT IN OF THE *
* SERIAL DATA.  THIS SHIFT PULSE OCCURS ONCE A SECOND,      *
* PERMITTING A BURST OF SHIFTS CONTROLLED BY THE EOP PULSE. *
* THE 8 MOST SIGNIFICANT BITS OF A VALUE ARE SHIFTED.       *
*                                                           *
*************************************************************
LDA  DAR      KP2            ; LOAD DAR WITH FS/4 FOR DIGITAL INPUT
LDA  WORK1    F1D1     L01 IN1 ; OF CONTROL SIGNAL TO SYNCH TIMERS
SUB  WORK1    F1D1     R05 IN1 ; ALSO, 0.25 TO 2.0 HZ BANDPASS 'A' CALC-
SUB  WORK1    F1D1     R07 IN1 ; ULATION.  THE 0.25 TO 2.0 HZ FILTER IS
SUB  WORK1    F1D1     R09 IN1 ; OPERATING AT 50 HZ.
SUB  WORK1    F1D1     R11 IN1 ;
LDA  WORK2    F1D2     R00 IN1 ; 0.25 TO 2.0 HZ BANDPASS 'B' CALCULATION
SUB  WORK2    F1D2     R05 IN1 ;
SUB  WORK2    F1D2     R07 IN1 ;
SUB  WORK2    F1D2     R10     ; SAMPLE DONE-ALLOW TO SETTLE
SUB  WORK2    F1D2     R12 CVT3; CNVRT BIT 3 TO DETERMINE IF 1 OR 0
LDA  WORK3    F1D3     L01     ; 0.25 TO 2.0 HZ BANDPASS 'C' CALCULATION
SUB  WORK3    F1D3     R02     ;
LDA  FIFTY    KP4          CND3; RESET 50 HZ TIMER IF CNTL LINE ='1'
LDA  ONE      KP3          CND3; RESET 1 HZ TIMER IF CNTL LINE='1'
SUB  DAR      DAR           ; CLEAR DAR FOR EEG1 A/D CONVERSION
SUB  WORK3    F1D3     R05 IN0 ; SAMPLE EEG1
SUB  WORK3    F1D3     R10 IN0 ;
LDA  WORK4    F1D4     R00 IN0 ; 0.25 TO 2.0 HZ BANDPASS 'D' CALCULATION
SUB  WORK4    F1D4     R02 IN0 ;
ADD  WORK4    F1D4     R06 IN0 ;
SUB  WORK4    F1D4     R09 IN0 ;
SUB  WORK4    F1D4     R13 IN0 ;
SUB  WORK1    WORK2        IN0 ; 0.25 TO 2.0 HZ BANDPASS CALCULATION
ADD  WORK1    EEG      R09     ; ADD IN EEG SAMPLE MULT BY K1
LDA  WORK2    WORK1        CVTS; BP CALCULATION, START A/D CONVERSION
ADD  DAR      KM2      R00 CND6; A/D FIXUP - 2920 PROBLEM
SUB  WORK2    F1D2          ; BP CALCULATION
SUB  WORK3    WORK4         ; BP CALCULATION
ADD  WORK3    WORK2    R01 CVT7; ADD RESULT OF FIRST SECTION MULT BY K2
LDA  WORK4    WORK3         ; BP CALCULATION
SUB  WORK4    F1D4          ; BP CALCULATION
LDA  WORK2    F2D1     L01 CVT6; LOWPASS FILTER FOR 0.25-2.0 HZ BP 'ALPHA'
SUB  WORK2    F2D1     R06     ; CALCULATION
SUB  WORK2    F2D1     R09     ;
SUB  WORK2    F2D1     R12 CVT5;
                       NOP     ;
ABA  WORK2    WORK4    R11     ; ADD IN RECT BP OUTPUT MULT BY K3
LDA  WORK4    F2D2     R00 CVT4; 'BETA' CALCULATION
SUB  WORK4    F2D2     R06     ;
SUB  WORK4    F2D2     R09     ;
SUB  WORK2    WORK4        CVT3; LP CALCULATION
                           CND4; A/D FIXUP - 2920 PROBLEM
LDA  WORK4    WORK2         ; LP CALCULATION
ADD  WORK4    F2D1     L01 CVT2; LP CALCULATION
                           CND4; A/D FIXUP - 2920 PROBLEM
ADD  WORK4    F2D2          ; LP CALCULATION
LDA  CNTL     KM7           CVT1; SET DEFAULT FOR DIGITAL OUTPUT->0
                           CND4; A/D FIXUP - 2920 PROBLEM
SUB  FIFTY    KP1      R05     ; DECREMENT 50 HZ TIMER
SUB  TWOHUND  KP1      R05 CVT0; DECREMENT 200 HZ TIMER
SUB  TWELVE   KP1      R07     ; DECREMENT 12.5 HZ TIMER
LDA  EEG      DAR           ; SAVE THE A/D CONVERTED EEG1 SAMPLE
LDA  DAR      FIFTY         ; LOAD 50 HZ TIMER FOR CONDITIONAL DELAYS
LDA  F1D2     F1D1          CND5; CONDITIONAL DELAY
LDA  F1D1     WORK1         CND5; CONDITIONAL DELAY
LDA  F1D4     F1D3          CND5; CONDITIONAL DELAY
LDA  F1D3     WORK3         CND5; CONDITIONAL DELAY
ADD  ONE      KM1      R04 CND5; DECREMENT 1 HZ TIMER
LDA  DAR      TWELVE        ; LOAD 12.5 HZ TIMER FOR CONDITIONAL DELAYS
LDA  F2D2     F2D1          CND5; CONDITIONAL DELAY
LDA  F2D1     WORK2         CND5; CONDITIONAL DELAY
LDA  F2OUT    WORK4         CND5; CONDITIONAL DELAY
LDA  TWELVE   KP4           CND5; RESET 12.5 HZ TIMER
LDA  DAR      ONE           ; LOAD 1 HZ TIMER FOR SHIFT CONTROL
LDA  EIGHT    KM8           CND5; RESET THE COUNT 8 TIMER FOR SHIFTING
LDA  TEMP2    F2OUT         CND5; CATCH FILTER OUTPUTS FOR (DESTRUCTIVE)
LDA  TEMP6    F6OUT         CND5; SHIFT OUT OF VALUES (50 HZ IN SYNCH)
LDA  ONE      KP3           CND5; RESET 1 HZ TIMER
LDA  DAR      EIGHT         ; LOAD COUNT 8 TIMER TO SET CONTROL OUTPUT
ADD  EIGHT    KP1           CND5; 'DECREMENT' COUNT 8 TIMER
LDA  CNTL     KP7           CND5; SET CONTROL OUTPUT HIGH IF IN COUNTDOWN
LDA  DAR      CNTL          ; LOAD CONTROL IN DAR FOR OUTPUT
LDA  WORK1    F3D1     L01     ; 2.0 TO 7.0 HZ BANDPASS FILTER 'A' CALC
SUB  WORK1    F3D1     R02     ;
SUB  WORK1    F3D1     R05     ;
SUB  WORK1    F3D1     R09 OUT7; SHIFT CONTROL OUTPUT
ADD  WORK1    F3D1     R12 OUT7; SHIFT CONTROL OUTPUT
LDA  CNTL     KM7           ; SET DEFAULT DIGITAL OUTPUT TO '0'
LDA  DAR      TEMP2         ; LOAD CAPTURED OUTPUT FOR SERIAL OUTPUT
LDA  CNTL     KP7           CND1; SEND OUT BIT 1 (LEAST SIG OF UPPER 8 BITS
LDA  TEMP2    DAR      R01     ; SAVE BACK SHIFTED VALUE FOR NEXT SHIFT
LDA  DAR      CNTL          ; LOAD DAR WITH CONTROL VALUE FOR OUTPUT
LDA  WORK2    F3D2     R00     ; BANDPASS FILTER 'B' CALCULATION
ADD  WORK3    F3D3     R04 OUT6; AND BANDPASS FILTER 'C' CALCULATION
ADD  WORK3    F3D3     R06     ;
ADD  WORK3    F3D3     R09     ;
ADD  WORK3    F3D3     R10     ;
SUB  WORK3    F3D3     R13     ;
```

```
LDA  WORK4    F3D4     R01       ;  BANDPASS FILTER 'D' CALCULATION
ADD  WORK4    F3D4     R05       ;
SUB  WORK4    F3D4     R09       ;
SUB  WORK4    F3D4     R10       ;  2 TO 7 HZ AMP ESTIMATION OUTPUT ****
SUB  WORK1    WORK2              ;  BP CALCULATION
LDA  CNTL     KM7                ;  SET DEFAULT DIGITAL OUTPUT TO '0'
LDA  DAR      TEMP6              ;  LOAD CAPTURED OUTPUT FOR SERIAL OUTPUT
LDA  CNTL     KP7          CND1; ;  SEND OUT BIT 1 (LEAST SIG OF UPPER 8 BITS
LDA  TEMP6    DAR        R01     ;  SAVE BACK SHIFTED VALUE FOR NEXT SHIFT
LDA  DAR      CNTL               ;  LOAD DAR WITH CONTROL VALUE FOR OUTPUT
ADD  WORK1    EEG        R04     ;  ADD IN EEG SAMPLE MULT BY K1
SUB  WORK2    F3D2               ;  BP CALCULATION
LDA  WORK2    WORK1              ;  BP CALCULATION
SUB  WORK3    WORK4        OUT4; ;  12 TO 14 HZ AMP ESTIMATION OUTPUT ****
ADD  WORK3    WORK2      R00 OUT4; ;  ADD FIRST SECTION OUTPUT MULT BY K2
LDA  WORK4    WORK3              ;  BP CALCULATION
SUB  WORK4    F3D4               ;  BP CALCULATION
LDA  WORK2    F4D1       L01     ;  2 - 7 HZ BP/FW RECT/ LOWPASS 'ALPHA' CALC
SUB  WORK2    F4D1       R05     ;
SUB  WORK2    F4D1       R08     ;
SUB  WORK2    F4D1       R11     ;
ADD  WORK2    F4D1       R13     ;
ABA  WORK2    WORK4      R12     ;  ADD RECT BP OUTPUT MULT BY K3
LDA  WORK4    F4D2       R00     ;  LOWPASS FILTER 'BETA' CALCULATION
SUB  WORK4    F4D2       R05     ;
SUB  WORK4    F4D2       R08     ;
ADD  WORK4    F4D2       R12     ;
SUB  WORK2    WORK4              ;  LP CALCULATION
LDA  WORK4    WORK2              ;  LP CALCULATION
ADD  WORK4    F4D1       L01     ;  LP CALCULATION
ADD  WORK4    F4D2       R00     ;  LP CALCULATION
LDA  DAR      FIFTY              ;  LOAD 50 HZ TIMER FOR CONDITIONAL DELAYS
LDA  F3D2     F3D1           CNDS; ;  CONDITIONAL DELAY
LDA  F3D1     WORK1          CNDS; ;  CONDITIONAL DELAY
LDA  F3D4     F3D3           CNDS; ;  CONDITIONAL DELAY
LDA  F3D3     WORK3          CNDS; ;  CONDITIONAL DELAY
LDA  F4D2     F4D1           CNDS; ;  CONDITIONAL DELAY
LDA  F4D1     WORK2          CNDS; ;  CONDITIONAL DELAY
LDA  F4OUT    WORK4          CNDS; ;  CONDITIONAL DELAY
LDA  FIFTY    KP4            CNDS; ;  RESET 50 HZ TIMER
LDA  CNTL     KM7                ;  SET DEFAULT DIGITAL OUTPUT TO '0'
LDA  DAR      F4OUT              ;  LOAD CAPTURED OUTPUT FOR SERIAL OUTPUT
LDA  CNTL     KP7          CND1  ;  SEND OUT BIT 1 (LEAST SIG OF UPPER 8 BITS
LDA  F4OUT    DAR        R01     ;  SAVE BACK SHIFTED VALUE FOR NEXT SHIFT
LDA  DAR      CNTL               ;  LOAD DAR WITH CONTROL VALUE FOR OUTPUT
LDA  WORK1    F5D1       L01     ;  12 TO 14 HZ BANDPASS FILTER 'A' CALC
SUB  WORK1    F5D1       R03     ;
SUB  WORK1    F5D1       R04     ;
ADD  WORK1    F5D1       R09 OUT5;
ADD  WORK1    F5D1       R11 OUT5;
LDA  WORK2    F5D2       R00     ;  BANDPASS FILTER 'B' CALCULATION
SUB  WORK2    F5D2       R05     ;
SUB  WORK2    F5D2       R07     ;
SUB  WORK2    F5D2       R09     ;
SUB  WORK2    F5D2       R12     ;
LDA  WORK3    F5D3       L01     ;  BANDPASS FILTER 'C' CALCULATION
SUB  WORK3    F5D3       R02     ;
ADD  WORK3    F5D3       R05     ;
SUB  WORK3    F5D3       R08     ;
SUB  WORK3    F5D3       R10     ;
LDA  WORK4    F5D4       R00     ;  BANDPASS FILTER 'D' CALCULATION
SUB  WORK4    F5D4       R05     ;
SUB  WORK4    F5D4       R06     ;
ADD  WORK4    F5D4       R10     ;
ADD  WORK4    F5D4       R12     ;
SUB  WORK1    WORK2              ;  BP CALCULATION
ADD  WORK1    EEG        R07     ;  ADD IN EEG1 SAMPLE MULT BY K1
LDA  WORK2    WORK1              ;  BP CALCULATION
SUB  WORK2    F5D2               ;  BP CALCULATION
SUB  WORK3    WORK4              ;  BP CALCULATION
ADD  WORK3    WORK2      R05     ;  ADD FIRST SECTION OUTPUT MULT BY K2
LDA  WORK4    WORK3              ;  BP CALCULATION
SUB  WORK4    F5D4               ;  BP CALCULATION
LDA  WORK2    F6D1       L01     ;  12 - 14 HZ BP/RECT/LOWPASS 'ALPHA' CALC
SUB  WORK2    F6D1       R05     ;
SUB  WORK2    F6D1       R06     ;
ADD  WORK2    F6D1       R09     ;
ADD  WORK2    F6D1       R11     ;
ABA  WORK2    WORK4      R10     ;  ADD IN THE FW RECTIFIED MULT BY K3
LDA  WORK4    F6D2       R00     ;  'BETA' CALCULATION
SUB  WORK4    F6D2       R05     ;
SUB  WORK4    F6D2       R06     ;
ADD  WORK4    F6D2       R08     ;
SUB  WORK4    F6D2       R11     ;
SUB  WORK2    WORK4              ;  LP CALCULATION
LDA  WORK4    WORK2              ;  LP CALCULATION
ADD  WORK4    F6D1       L01     ;  LP CALCULATION
ADD  WORK4    F6D2       R00     ;  LP CALCULATION
LDA  DAR      TWOHUND            ;  LOAD 200 HZ TIMER FOR CONDITIONAL DELAYS
LDA  F5D2     F5D1           CNDS; ;  CONDITIONAL DELAY
LDA  F5D1     WORK1          CNDS; ;  CONDITIONAL DELAY
LDA  F5D4     F5D3           CNDS; ;  CONDITIONAL DELAY
LDA  F5D3     WORK3          CNDS; ;  CONDITIONAL DELAY
LDA  F6D2     F6D1           CNDS; ;  CONDITIONAL DELAY
LDA  F6D1     WORK2          CNDS; ;  CONDITIONAL DELAY
LDA  F6OUT    WORK4          CNDS; ;  CONDITIONAL DELAY
                             EOP ;  SIGNAL END OF PROGRAM (PREFETCHED 4 INST)
LDA  TWOHUND  KP1            CNDS; ;  RESET 200 HZ TIMER
                             NOP ;
                             NOP ;
```

```
*************************************************************
*                                                           *
* PROGRAM OPERATING IN 2920 NUMBER 2- EEG TAKEN AS INPUT AT*
* A RATE OF 6.5 KHZ, WITH BANDPASS/FW RECT/LOWPASS FILTERS *
* OPERATING AT 200, 50, & 12.5 HZ. THIS 2920 USES THE      *
* SHIFT CONTROL PULSE WHICH IS PROVIDED BY 2920 NUMBER 1.   *
* THIS SHIFT PULSE OCCURS ONCE A SECOND, PERMITTING A BURST*
* OF SHIFTS CONTROLLED BY THE EOP PULSE. THE 8 MOST SIGNIF *
* BITS OF A VALUE ARE SHIFTED. THE 2920S ARE ALL SYNCED     *
* WITH A CONTROL INPUT AND WITH THEIR EOPS TIED TOGETHER.   *
*                                                           *
*************************************************************
LDA DAR       KP2                ; LOAD DAR WITH FS/4 FOR DIGITAL INPUT
LDA WORK1     F1D1      L01 IN1  ; OF CONTROL SIGNAL TO SYNCH TIMERS
SUB WORK1     F1D1      R05 IN1  ; ALSO, 0.25 TO 2.0 HZ BANDPASS 'A' CALC-
SUB WORK1     F1D1      R07 IN1  ; ULATION. THE 0.25 TO 2.0 HZ FILTER IS
SUB WORK1     F1D1      R09 IN1  ; OPERATING AT 50 HZ.
SUB WORK1     F1D1      R11 IN1  ;
LDA WORK2     F1D2      R00 IN1  ; 0.25 TO 2.0 HZ BANDPASS 'B' CALCULATION
SUB WORK2     F1D2      R05 IN1  ;
SUB WORK2     F1D2      R07 IN1  ;
SUB WORK2     F1D2      R10      ; SAMPLE DONE-ALLOW TO SETTLE
SUB WORK2     F1D2      R12 CVT3; CNVRT BIT 3 TO DETERMINE IF 1 OR 0
LDA WORK3     F1D3      L01      ; 0.25 TO 2.0 HZ BANDPASS 'C' CALCULATION
SUB WORK3     F1D3      R02      ;
LDA FIFTY     KP4            CND3; RESET 50 HZ TIMER IF CNTL LINE ='1'
LDA ONE       KP3            CND3; RESET 1 HZ TIMER IF CNTL LINE='1'
SUB DAR       DAR                ; CLEAR DAR FOR EEG1 A/D CONVERSION
SUB WORK3     F1D3      R05 IN0  ; SAMPLE EEG1
SUB WORK3     F1D3      R10 IN0  ;
LDA WORK4     F1D4      R00 IN0  ; 0.25 TO 2.0 HZ BANDPASS 'D' CALCULATION
SUB WORK4     F1D4      R02 IN0  ;
ADD WORK4     F1D4      R06 IN0  ;
SUB WORK4     F1D4      R09 IN0  ;
SUB WORK4     F1D4      R13 IN0  ;
SUB WORK1     WORK2          IN0  ; 0.25 TO 2.0 HZ BANDPASS CALCULATION
ADD WORK1     EEG       R09      ; ADD IN EEG SAMPLE MULT BY K1
LDA WORK2     WORK1          CVT5; BP CALCULATION, START A/D CONVERSION
ADD DAR       KM2       R00 CND6; A/D FIXUP - 2920 PROBLEM
SUB WORK2     F1D2               ; BP CALCULATION
SUB WORK3     WORK4              ; BP CALCULATION
ADD WORK3     WORK2     R01 CVT7; ADD RESULT OF FIRST SECTION MULT BY K2
LDA WORK4     WORK3              ; BP CALCULATION
SUB WORK4     F1D4               ; BP CALCULATION
LDA WORK2     F2D1      L01 CVT6; LOWPASS FILTER FOR 0.25-2.0 HZ BP 'ALPHA'
SUB WORK2     F2D1      R06      ; CALCULATION
SUB WORK2     F2D1      R09      ;
                        NOP      ;
SUB WORK2     F2D1      R12 CVT5;
ABA WORK2     WORK4     R11      ; ADD IN RECT BP OUTPUT MULT BY K3
LDA WORK4     F2D2      R00 CVT4; 'BETA' CALCULATION
SUB WORK4     F2D2      R06      ;
SUB WORK4     F2D2      R09      ;
SUB WORK2     WORK4          CVT3; LP CALCULATION
                             CND4; A/D FIXUP - 2920 PROBLEM
LDA WORK4     WORK2              ; LP CALCULATION
ADD WORK4     F2D1      L01 CVT2; LP CALCULATION
                             CND4; A/D FIXUP - 2920 PROBLEM
ADD WORK4     F2D2               ; LP CALCULATION
LDA CNTL      KM7                ; SET DEFAULT FOR DIGITAL OUTPUT->0
                             CND4; A/D FIXUP - 2920 PROBLEM
SUB FIFTY     KP1       R05      ; DECREMENT 50 HZ TIMER
SUB TWOHUND   KP1       R05 CVT0;DECREMENT 200 HZ TIMER
SUB TWELVE    KP1       R07      ; DECREMENT 12.5 HZ TIMER
LDA EEG       DAR                ; SAVE THE A/D CONVERTED EEG1 SAMPLE
LDA DAR       FIFTY              ; LOAD 50 HZ TIMER FOR CONDITIONAL DELAYS
LDA F1D2      F1D1           CND5; CONDITIONAL DELAY
LDA F1D1      WORK1          CND5; CONDITIONAL DELAY
LDA F1D4      F1D3           CND5; CONDITIONAL DELAY
LDA F1D3      WORK3          CND5; CONDITIONAL DELAY
ADD ONE       KM1       R04 CND5; DECREMENT 1 HZ TIMER
LDA DAR       TWELVE             ; LOAD 12.5 HZ TIMER FOR CONDITIONAL DELAYS
LDA F2D2      F2D1           CND5; CONDITIONAL DELAY
LDA F2D1      WORK2          CND5; CONDITIONAL DELAY
LDA F2OUT     WORK4          CND5; CONDITIONAL DELAY
LDA TWELVE    KP4            CND5; RESET 12.5 HZ TIMER
LDA DAR       ONE                ; LOAD 1 HZ TIMER FOR SHIFT CONTROL
LDA EIGHT     KM8            CND5; RESET THE COUNT 8 TIMER FOR SHIFTING
LDA TEMP2     F2OUT          CND5; CATCH FILTER OUTPUTS FOR (DESTRUCTIVE)
LDA TEMP6     F6OUT          CND5; SHIFT OUT OF VALUES (50 HZ IN SYNCH)
LDA ONE       KP3            CND5; RESET 1 HZ TIMER
LDA DAR       EIGHT              ; LOAD COUNT 8 TIMER TO SET CONTROL OUTPUT
ADD EIGHT     KP1            CND5; 'DECREMENT' COUNT 8 TIMER
LDA WORK1     F3D1      L01      ; 2.0 TO 7.0 HZ BANDPASS FILTER 'A' CALC
SUB WORK1     F3D1      R02      ;
SUB WORK1     F3D1      R05      ;
SUB WORK1     F3D1      R09      ;
ADD WORK1     F3D1      R12      ;
LDA DAR       TEMP2              ; LOAD CAPTURED OUTPUT FOR SERIAL OUTPUT
LDA CNTL      KP7            CND1; SEND OUT BIT 1 (LEAST SIG OF UPPER 8 BITS
LDA TEMP2     DAR       R01      ; SAVE BACK SHIFTED VALUE FOR NEXT SHIFT
LDA DAR       CNTL               ; LOAD DAR WITH CONTROL VALUE FOR OUTPUT
LDA WORK2     F3D2      R00      ; BANDPASS FILTER 'B' CALCULATION
SUB WORK2     F3D2      R02      ;
ADD WORK2     F3D2      R05      ;
LDA WORK3     F3D3      R00 OUT6; .25 TO 2 HZ AMP ESTIMATION OUTPUT ****
ADD WORK3     F3D3      R04 OUT6; AND BANDPASS FILTER 'C' CALCULATION
ADD WORK3     F3D3      R06      ;
ADD WORK3     F3D3      R09      ;
ADD WORK3     F3D3      R10      ;
```

```
SUB  WORK3    F3D3     R13       ;
LDA  WORK4    F3D4     R01       ; BANDPASS FILTER 'D' CALCULATION
ADD  WORK4    F3D4     R05       ;
SUB  WORK4    F3D4     R09       ;
SUB  WORK4    F3D4     R10       ; 2 TO 7 HZ AMP ESTIMATION OUTPUT ****
SUB  WORK1    WORK2              ; BP CALCULATION
LDA  CNTL     KM7                ; SET DEFAULT DIGITAL OUTPUT TO '0'
LDA  DAR      TEMP6              ; LOAD CAPTURED OUTPUT FOR SERIAL OUTPUT
LDA  CNTL     KP7         CND1;  SEND OUT BIT 1 (LEAST SIG OF UPPER 8 BITS
LDA  TEMP6    DAR       R01      ; SAVE BACK SHIFTED VALUE FOR NEXT SHIFT
LDA  DAR      CNTL               ; LOAD DAR WITH CONTROL VALUE FOR OUTPUT
ADD  WORK1    EEG       R04      ; ADD IN EEG SAMPLE MULT BY K1
LDA  WORK2    WORK1              ; BP CALCULATION
SUB  WORK2    F3D2               ; BP CALCULATION
SUB  WORK3    WORK4       OUT4;  12 TO 14 HZ AMP ESTIMATION OUTPUT ****
ADD  WORK3    WORK2     R00 OUT4;  ADD FIRST SECTION OUTPUT MULT BY K2
LDA  WORK4    WORK3              ; BP CALCULATION
SUB  WORK4    F3D4               ; BP CALCULATION
LDA  WORK2    F4D1      L01      ; 2 - 7 HZ BP/FW RECT/ LOWPASS 'ALPHA' CALC
SUB  WORK2    F4D1      R05      ;
SUB  WORK2    F4D1      R08      ;
SUB  WORK2    F4D1      R11      ;
ADD  WORK2    F4D1      R13      ;
ABA  WORK2    WORK4     R12      ; ADD RECT BP OUTPUT MULT BY K3
LDA  WORK4    F4D2      R00      ; LOWPASS FILTER 'BETA' CALCULATION
SUB  WORK4    F4D2      R05      ;
SUB  WORK4    F4D2      R08      ;
ADD  WORK4    F4D2      R12      ;
SUB  WORK2    WORK4              ; LP CALCULATION
LDA  WORK4    WORK2              ; LP CALCULATION
ADD  WORK4    F4D1      L01      ; LP CALCULATION
ADD  WORK4    F4D2      R00      ; LP CALCULATION
LDA  DAR      FIFTY              ; LOAD 50 HZ TIMER FOR CONDITIONAL DELAYS
LDA  F3D2     F3D1         CNDS; CONDITIONAL DELAY
LDA  F3D1     WORK1        CNDS; CONDITIONAL DELAY
LDA  F3D4     F3D3         CNDS; CONDITIONAL DELAY
LDA  F3D3     WORK3        CNDS; CONDITIONAL DELAY
LDA  F4D2     F4D1         CNDS; CONDITIONAL DELAY
LDA  F4D1     WORK2        CNDS; CONDITIONAL DELAY
LDA  F4OUT    WORK4        CNDS; CONDITIONAL DELAY
LDA  FIFTY    KP4          CNDS; RESET 50 HZ TIMER
LDA  CNTL     KM7                ; SET DEFAULT DIGITAL OUTPUT TO '0'
LDA  DAR      F4OUT              ; LOAD CAPTURED OUTPUT FOR SERIAL OUTPUT
LDA  CNTL     KP7       CND1     ; SEND OUT BIT 1 (LEAST SIG OF UPPER 8 BITS
LDA  F4OUT    DAR       R01      ; SAVE BACK SHIFTED VALUE FOR NEXT SHIFT
LDA  DAR      CNTL               ; LOAD DAR WITH CONTROL VALUE FOR OUTPUT
LDA  WORK1    F5D1      L01      ; 12 TO 14 HZ BANDPASS FILTER 'A' CALC
SUB  WORK1    F5D1      R03      ;
SUB  WORK1    F5D1      R04      ;
ADD  WORK1    F5D1      R09 OUT5;
ADD  WORK1    F5D1      R11 OUT5;
LDA  WORK2    F5D2      R00      ; BANDPASS FILTER 'B' CALCULATION
SUB  WORK2    F5D2      R05      ;
SUB  WORK2    F5D2      R07      ;
SUB  WORK2    F5D2      R09      ;
SUB  WORK2    F5D2      R12      ;
LDA  WORK3    F5D3      L01      ; BANDPASS FILTER 'C' CALCULATION
SUB  WORK3    F5D3      R02      ;
ADD  WORK3    F5D3      R05      ;
SUB  WORK3    F5D3      R08      ;
SUB  WORK3    F5D3      R10      ;
LDA  WORK4    F5D4      R00      ; BANDPASS FILTER 'D' CALCULATION
SUB  WORK4    F5D4      R05      ;
SUB  WORK4    F5D4      R06      ;
ADD  WORK4    F5D4      R10      ;
ADD  WORK4    F5D4      R12      ;
SUB  WORK1    WORK2              ; BP CALCULATION
ADD  WORK1    EEG       R07      ; ADD IN EEG1 SAMPLE MULT BY K1
LDA  WORK2    WORK1              ; BP CALCULATION
SUB  WORK2    F5D2               ; BP CALCULATION
SUB  WORK3    WORK4              ; BP CALCULATION
ADD  WORK3    WORK2     R05      ; ADD FIRST SECTION OUTPUT MULT BY K2
LDA  WORK4    WORK3              ; BP CALCULATION
SUB  WORK4    F5D4               ; BP CALCULATION
LDA  WORK2    F6D1      L01      ; 12 - 14 HZ BP/RECT/LOWPASS 'ALPHA' CALC
SUB  WORK2    F6D1      R05      ;
SUB  WORK2    F6D1      R06      ;
ADD  WORK2    F6D1      R09      ;
ADD  WORK2    F6D1      R11      ;
ABA  WORK2    WORK4     R10      ; ADD IN THE FW RECTIFIED MULT BY K3
LDA  WORK4    F6D2      R00      ; 'BETA' CALCULATION
SUB  WORK4    F6D2      R05      ;
SUB  WORK4    F6D2      R06      ;
ADD  WORK4    F6D2      R08      ;
SUB  WORK4    F6D2      R11      ;
SUB  WORK2    WORK4              ; LP CALCULATION
LDA  WORK4    WORK2              ; LP CALCULATION
ADD  WORK4    F6D1      L01      ; LP CALCULATION
ADD  WORK4    F6D2      R00      ; LP CALCULATION
LDA  DAR      TWOHUND            ; LOAD 200 HZ TIMER FOR CONDITIONAL DELAYS
LDA  F5D2     F5D1         CNDS; CONDITIONAL DELAY
LDA  F5D1     WORK1        CNDS; CONDITIONAL DELAY
LDA  F5D4     F5D3         CNDS; CONDITIONAL DELAY
LDA  F5D3     WORK3        CNDS; CONDITIONAL DELAY
LDA  F6D2     F6D1         CNDS; CONDITIONAL DELAY
LDA  F6D1     WORK2        CNDS; CONDITIONAL DELAY
LDA  F6OUT    WORK4        CNDS; CONDITIONAL DELAY
LDA  TWOHUND  KP1          CNDS; RESET 200 HZ TIMER
                           NOP ; NOPS FOR REMAINDER OF
                           NOP ; FULL 192 PROGRAM STEPS
```

```
*************************************************************
*                                                           *
* PROGRAM OPERATING IN 2920 NUMBER 3- EOG TAKEN AS INPUT AT*
* A RATE OF 6.5 KHZ, WITH BANDPASS/FW RECT/LOWPASS FILTER   *
* OPERATING AT 50 HZ. THIS 2920 USES THE SHIFT CONTROL      *
* PULSE WHICH IS PROVIDED BY 2920 NUMBER 1.                 *
* THIS SHIFT PULSE OCCURS ONCE A SECOND, PERMITTING A BURST*
* OF SHIFTS CONTROLLED BY THE EOP PULSE. THE 8 MOST SIGNIF  *
* BITS OF A VALUE ARE SHIFTED. THE 2920S ARE ALL SYNCED     *
* WITH A CONTROL INPUT AND WITH THEIR EOPS TIED TOGETHER.   *
*                                                           *
*************************************************************
LDA DAR      KP2                ; LOAD DAR WITH FS/4 FOR DIGITAL INPUT
LDA WORK1    F1D1     L01 IN1   ; OF CONTROL SIGNAL TO SYNCH TIMERS
SUB WORK1    F1D1     R02 IN1   ; ALSO, 2.0 TO 10 HZ BANDPASS 'A' CALC-
SUB WORK1    F1D1     R04 IN1   ; ULATION. THE 2.0 TO 10 HZ FILTER IS
SUB WORK1    F1D1     R06 IN1   ; OPERATING AT 50 HZ.
SUB WORK1    F1D1     R09 IN1   ;
ADD WORK1    F1D1     R12 IN1   ;
LDA WORK2    F1D2     R01 IN1   ; 2.0 TO 10 HZ BANDPASS 'B' CALCULATION
ADD WORK2    F1D2     R02 IN1   ;
SUB WORK2    F1D2     R07       ; SAMPLE DONE-ALLOW TO SETTLE
SUB WORK2    F1D2     R08 CVT3; CNVRT BIT 3 TO DETERMINE IF 1 OR 0
ADD WORK2    F1D2     R10       ;
SUB WORK2    F1D2     R12       ;
LDA FIFTY    KP4           CND3; RESET 50 HZ TIMER IF CNTL LINE ='1'
LDA ONE      KP3           CND3; RESET 1 HZ TIMER IF CNTL LINE='1'
SUB DAR      DAR                 ; CLEAR DAR FOR EOG A/D CONVERSION
LDA WORK3    F1D3     R01 IN0   ; SAMPLE EOG, BANDPASS 'C' CALCULATION
ADD WORK3    F1D3     R04 IN0   ;
SUB WORK3    F1D3     R07 IN0   ;
SUB WORK3    F1D3     R08 IN0   ;
ADD WORK3    F1D3     R10 IN0   ;
LDA WORK4    F1D4     R02 IN0   ; BANDPASS 'D' CALCULATION
ADD WORK4    F1D4     R04 IN0   ;
ADD WORK4    F1D4     R05 IN0   ;
SUB WORK4    F1D4     R09       ;
SUB WORK4    F1D4     R11 CVT5; START A/D CONVERSION
ADD DAR      KM2      R00 CND6; A/D FIXUP - 2920 PROBLEM
ADD WORK4    F1D4     R13       ;
SUB WORK1    WORK2               ; BP CALCULATION
ADD WORK1    EOG      R03 CVT7; ADD IN SAMPLE MULT BY K1
LDA WORK2    WORK1               ; BP CALCULATION
SUB WORK2    F1D2                ; BP CALCULATION
SUB WORK3    WORK4          CVT6; BP CALCULATION
ADD WORK3    WORK2    R00       ; ADD RESULT OF FIRST SECTION MULT BY K2
LDA WORK4    WORK3               ; BP CALCULATION
SUB WORK4    F1D4           CVT5; BP CALCULATION
LDA WORK2    F2D1     L01       ; LOWPASS FILTER 'ALPHA' CALCULATION
SUB WORK2    F2D1     R05       ;
SUB WORK2    F2D1     R08 CVT4;
SUB WORK2    F2D1     R11       ;
ADD WORK2    F2D1     R13       ;
ABS WORK4    WORK4          CVT3; FULL WAVE RECTIFY THE BP OUTPUT
                            CND4; A/D FIXUP - 2920 PROBLEM
ADD WORK2    WORK4    R12       ; ADD IN SECOND SECTION MULT BY K3
LDA WORK4    F2D2     R00 CVT2; LOWPASS FILTER 'BETA' CALCULATION
                            CND4; A/D FIXUP - 2920 PROBLEM
SUB WORK4    F2D2     R05       ;
SUB WORK4    F2D2     R08 CVT1;
                            CND4; A/D FIXUP - 2920 PROBLEM
ADD WORK4    F2D2     R12       ;
SUB WORK2    WORK4          CVT0; LP CALCULATION
LDA WORK4    WORK2               ; LP CALCULATION
ADD WORK4    F2D1     L01       ; LP CALCULATION
ADD WORK4    F2D2               ; LP CALCULATION
LDA EOG      DAR                 ; SAVE THE A/D CONVERTED EOG SAMPLE
LDA CNTL     KM7                 ; SET DEFAULT FOR DIGITAL OUTPUT->0
SUB FIFTY    KP1      R05       ; DECREMENT 50 HZ TIMER
LDA DAR      FIFTY               ; LOAD 50 HZ TIMER FOR CONDITIONAL DELAYS
LDA F1D2     F1D1           CND5; CONDITIONAL DELAY
LDA F1D1     WORK1          CND5; CONDITIONAL DELAY
LDA F1D4     F1D3           CND5; CONDITIONAL DELAY
LDA F1D3     WORK3          CND5; CONDITIONAL DELAY
ADD ONE      KM1      R04 CND5; DECREMENT 1 HZ TIMER
LDA F2D2     F2D1           CND5; CONDITIONAL DELAY
LDA F2D1     WORK2          CND5; CONDITIONAL DELAY
LDA F2OUT    WORK4          CND5; CONDITIONAL DELAY
LDA FIFTY    KP4            CND5; RESET THE 50 HZ TIMER
LDA DAR      ONE                 ; LOAD 1 HZ TIMER FOR SHIFT CONTROL
LDA EIGHT    KM8            CND5; RESET THE COUNT 8 TIMER FOR SHIFTING
LDA ONE      KP3            CND5; RESET 1 HZ TIMER
LDA DAR      EIGHT               ; LOAD COUNT 8 TIMER
ADD EIGHT    KP1            CND5; DECREMENT COUNT 8 TIMER
LDA DAR      F2OUT               ; LOAD LP OUTPUT FOR SHIFTOUT
LDA CNTL     KP7            CND1; SET OUT BIT 1(LEAST SIG OF UPPER 8 BITS)
LDA F2OUT    DAR      R01       ; SAVE BACK SHIFTED VALUE FOR NEXT SHIFT
LDA DAR      CNTL                ; LOAD DAR WITH CONTROL VALUE FOR OUTPUT
                            NOP ;
                            NOP ;
SUB WORK2    F3D2     R02       ;
ADD WORK2    F3D2     R05       ;
LDA WORK3    F3D3     R00 OUT6; .25 TO 2 HZ AMP ESTIMATION OUTPUT ****
                            NOP ;
                            OUT6; 2.0 TO 10 HZ AMP ESTIMATION OUTPUT ****
                            OUT6;
                            NOP ; NOPS FOR REMAINDER OF FULL  192
                            NOP ; PROGRAM STEPS
```

```
//REIMER JOB ',,,T=15,R=256,I=30,C=0,L=5','REIMER'
/*D800 VPLOT
// EXEC FORTHCLG,USERLIB='SYS2.VPLOTLIB'
//FORT.SYSIN DD *
C ***************************************************************
C *                                                             *
C * PROGRAM TO APPLY A GIVEN EPOCH RULE (ADJUSTABLE LENGTH) AND *
C * PLOT THE RESULTING IMAGE ON THE VERSATEC PLOTTER            *
C *                                                             *
C ***************************************************************
      INTEGER IBUF(4000),INDEX,INDEX1,N,NOW,LAST
      REAL INCR,EPOCH,AVER1,AVER2,AVER3
      REAL BPM1(3000),PAGE(3000),SECS(3000),TIME(3000),PGS
      REAL DUMMY,LARGE,Y,X,BPL1(3000),BPH1(3000),BPL,BPM,BPH
61    FORMAT(F4.0,4(F5.0))
      BPL=0
      BPM=0
      BPH=0
      INDEX=1
1       READ(5,61,END=10) PAGE(INDEX),SECS(INDEX),BPL1(INDEX),
     *  BPM1(INDEX),BPH1(INDEX)
        TIME(INDEX)=PAGE(INDEX)+(SECS(INDEX)/30.0)
        IF(BPL1(INDEX) .GT. BPL) BPL=BPL1(INDEX)
        IF(BPM1(INDEX) .GT. BPM) BPM=BPM1(INDEX)
        IF(BPH1(INDEX) .GT. BPH) BPH=BPH1(INDEX)
        INDEX=INDEX+1
      GO TO 1
10    LAST=INDEX-1
      DO 20 INDEX=1,LAST
        BPL1(INDEX)=BPL1(INDEX)/BPL
        BPM1(INDEX)=BPM1(INDEX)/BPM
        BPH1(INDEX)=BPH1(INDEX)/BPH
20    CONTINUE
      N=0
      AVER1=0
      AVER2=0
      AVER3=0
      INCR=10.0/30.0
      PGS=PAGE(LAST)-PAGE(1)+1
      PGS=0.25*PGS
      CALL SIZE(PGS)
      CALL PLOTS(IBUF,4000)
      EPOCH=TIME(1)+INCR
      DO 6 INDEX=1,LAST
        N=N+1
        AVER1=AVER1+BPL1(INDEX)
        AVER2=AVER2+BPM1(INDEX)
        AVER3=AVER3+BPH1(INDEX)
        IF(TIME(INDEX) .LT. EPOCH) GO TO 6
          AVER1=AVER1/N
          AVER2=AVER2/N
          AVER3=AVER3/N
          DO 5 INDEX1=1,N
            NOW=INDEX-INDEX1+1
            BPL1(NOW)=AVER1
            BPM1(NOW)=AVER2
            BPH1(NOW)=AVER3
5         CONTINUE
          N=0
          AVER1=0
          AVER2=0
          AVER3=0
          EPOCH=EPOCH+INCR
6     CONTINUE
      Y=1.0/4.0
      X=4.0
      CALL PLOT(0.0,-PGS,-3)
      CALL PLOT(0.0,0.5,-3)
      TIME(LAST+1)=PAGE(1)
      BPM1(LAST+1)=0.0
      TIME(LAST+2)=X
      BPL1(LAST+2)=Y
      CALL AXIS(0.0,0.0,'0.25 TO 2.0',11,4.0,90.0,0.0,Y)
      CALL AXIS(0.0,0.0,'TIME(PAGE)',-10,PGS,0.0,PAGE(1),X)
      CALL LINE(TIME,BPL1,LAST,1,0,0)
      CALL PLOT(0.0,4.1,-3)
      BPM1(LAST+1)=0.0
      BPM1(LAST+2)=Y
      CALL AXIS(0.0,0.0,'2.0 TO 7.0',10,4.0,90.0,0.0,Y)
      CALL LINE(TIME,BPM1,LAST,1,0,0)
      CALL PLOT(0.0,4.1,-3)
      BPH1(LAST+1)=0.0
      BPH1(LAST+2)=0.5
      CALL AXIS(0.0,0.0,'12 TO 14',8,2.0,90.0,0.0,0.5)
      CALL LINE(TIME,BPH1,LAST,1,0,0)
      CALL PLOT((PGS+4.0),0.0,999)
      STOP
      END
/*
//GO.FT01F001 DD DSN=&&FT01F001,UNIT=SYSDA,SPACE=(CYL,(2,2)),
//     DISP=(NEW,PASS)
//GO.VWORK DD DSN=&&VWORK,UNIT=SYSDA,SPACE=(CYL,(2,2)),
//     DISP=(NEW,PASS)
//GO.SYSIN DD DSN=JREIMER.PLOT.DATA,VOL=SER=USER02,
// DCB=(LRECL=80,BLKSIZE=6080),UNIT=SYSDA,DISP=(OLD,KEEP)
// EXEC VPLOT,COND=(0,NE)
```

```
*****************************************************************
*                                                               *
* PROGRAM TO ALLOW THE EXORCISOR TO APPEAR AS AN INTELLEGENT    *
* TERMINAL TO THE AMDAHL COMPUTER                               *
*                                                               *
*****************************************************************
0010 TERM:
0020  PROC OPTIONS(MAIN)
0030 $ NAM TERM
0040  DCL
0050    1 IOCB,
0060      2 STATUS BIN(1),
0070      2 MODE   BIN(1),
0080      2 BUFFP BIN(2),
0090      2 BUFFS BIN(2),
0100      2 BUFFE BIN(2),
0110      2 TYPE CHAR(2),
0120      2 UNIT CHAR(1),
0130      2 NAME CHAR(8),
0140      2 SUF   CHAR(2),
0150      2 DUM0   BIN(2),
0160      2 FORMAT BIN(1),
0170      2 DUM1   BIN(3),
0180      2 DUM2   BIN(2),
0190      2 ALLOC BIN(2),
0200      2 SECTS BIN(2),
0210      2 SECTE BIN(2),
0220      2 SECTP BIN(2)
0230  DCL INPUT CHAR(8)
0240  DCL DUMMY CHAR(1)
0250  DCL INPUT1 CHAR(13)
0260  DCL INDEX1 BIN(1)
0270  DCL PRMPT CHAR(12) INIT('EXORCISOR=*>')
0280  DCL EOTCH CHAR(1) INIT(¢4)
0290  DCL CMD1 CHAR(8) INIT('ONLINE')
0300  DCL CMD2 CHAR(8) INIT('END')
0310  DCL CMD3 CHAR(8) INIT('UPLOAD')
0320  DCL CMD5 CHAR(8) INIT('HELP')
0330  DCL BUFFER CHAR(60)
0340  DCL BUFEND CHAR(1) INIT(¢D)
0350  DECLARE
0360    1  BUF,
0370       2  BUFF(133) CHAR(1)
0380  DCL LAST BIN(2)
0390  DCL HOLD CHAR(2)
0400  DCL SECT0 CHAR(256)
0410  DCL SECT1 CHAR(256)
0420  DCL SECT2 CHAR(256)
0430  DCL SECT3 CHAR(256)
0440  DCL SECT4 CHAR(256)
0450  DCL SECT5 CHAR(256)
0460  DCL SECT6 CHAR(256)
0470  DCL SECT7 CHAR(256)
0480  DCL SECT8 CHAR(256)
0490  DCL SECT9 CHAR(256)
0500  DCL INDEX BIN(1)
0510  DCL PACK1 BIN(2)
0520  DCL PACK2 BIN(2)
0530  DCL 1 OUT4,
0540        2 XX7 CHAR(19) INIT('ENTER NAME OF FILE'),
0550        2 XX8 CHAR(15) INIT(' TO BE UPLOADED'),
0560        2 XX9 CHAR(1)  INIT(¢D)
0570  DCL FEED CHAR(1) INIT(¢C)
0580  DCL LINE CHAR(1) INIT(¢A)
0590  DCL LCU  CHAR(1) INIT(¢75)
0600  DCL CNTT CHAR(1) INIT(¢14)
0610  DCL CNTQ CHAR(1) INIT(¢11)
0620  DCL ESCP CHAR(1) INIT(¢1B)
0630  DCL CNTO CHAR(1) INIT(¢0F)
0640  DCL CNTN CHAR(1) INIT(¢0E)
0650  DCL HOLD0 BIN(1)
0660  DCL HOLD1 BIN(2)
0670  DCL HOLD2 CHAR(1)
0680  DCL NUM BIN(1)
0690  DCL ONE CHAR(1)
0700  DCL LNUM BIN(2)
0710  DCL COUNT2 BIN(2) DEF ¢E03C
0720  DCL LATCH3 BIN(2) DEF ¢E03E
0730  DCL LATCH2 BIN(2) DEF ¢E03C
0740  DCL CNTL3  BIN(1) DEF ¢E038
0750  DCL CNTL2  BIN(1) DEF ¢E039
0760  DCL CNTL1  BIN(1) DEF ¢E038
0770  DCL TIME   BIN(2)
0780  DCL TERMCT BIN(1) DEF ¢FCF4
0790  DCL TERMTR BIN(1) DEF ¢FCF5
0800  DCL CARDCT BIN(1) DEF ¢E000
0810  DCL CARDTR BIN(1) DEF ¢E001
0820  DCL STATE BIN(1)
0830  DCL 1 OUT6,
0840        2 XX10 CHAR(18) INIT('SENDING RECORD #:  '),
0850        2 XX11 CHAR(1)  INIT(¢04)
0860  DCL 1 OUT7,
0870        2 XX12 CHAR(4),
0880        2 XX13 CHAR(1) INIT(¢08),
0890        2 XX14 CHAR(1) INIT(¢08),
0900        2 XX15 CHAR(1) INIT(¢08),
0910        2 XX16 CHAR(1) INIT(¢08),
0920        2 XX17 CHAR(1) INIT(¢04)
0930  CARDCT=3
0940  TERMCT=3
```

```
0950   CARDCT=$49
0960   TERMCT=$49
0970 COMMND:
0980     CALL DSPLY<, , ADDR(BUFEND)>
0990 $ LDX #PRMPT
1000 $ SCALL .DSPLZ
1010     CALL KEYIN<,8,ADDR(INPUT)> GIVING<,NUM>
1020     IF NUM EQ 0 THEN GO TO HELP
1030     PACK1=ADDR(INPUT)
1040     PACK1=PACK1+NUM
1050     DO WHILE NUM LE 7
1060       PACK1->ONE=' '
1070       PACK1=PACK1+1
1080       NUM=NUM+1
1090     END
1100     IF INPUT EQ CMD1 THEN GO TO ONLIN
1110     IF INPUT EQ CMD2 THEN GO TO ENDER
1120     IF INPUT EQ CMD3 THEN GO TO UPLOAD
1130     IF INPUT EQ CMD5 THEN GO TO HELP
1140     BUFFER='ERROR, ENTER "HELP" FOR HELP'
1150     CALL DSPLY<, , ADDR(BUFFER)>
1160     GO TO COMMND
1170 HELP: BUFFER='SUPPORTED COMMANDS ARE:'
1180     CALL DSPLY<, , ADDR(BUFFER)>
1190     BUFFER='ONLINE   END   UPLOAD'
1200     CALL DSPLY<, , ADDR(BUFFER)>
1210     GO TO COMMND
1220 ONLIN:
1230     BUFFER='GOING ONLINE THROUGH ACIA CARD'
1240     CALL DSPLY<, , ADDR(BUFFER)>
1250     BUFFER='ENTER "ESC" TO END ONLINE'
1260     CALL DSPLY<, , ADDR(BUFFER)>
1270     STATE=0
1280 TOP:
1290     CALL TERMIN
1300     IF NUM EQ 0 THEN GO TO CONT1
1310     IF ONE EQ ESCP THEN GO TO COMMND
1320     CALL CARDOU
1330 CONT1: CALL CARDIN
1340     IF NUM EQ 0 THEN GO TO TOP
1350     IF ONE EQ CNTN THEN STATE=1
1360     IF ONE EQ CNTO THEN STATE=0
1370     IF STATE EQ 1 THEN GO TO TOP
1380     CALL TERMOU
1390     GO TO TOP
1400 ENDER: CALL MDOS
1410 UPLOAD:
1420     BUFFER=OUT4
1430     CALL DSPLY<, , ADDR(BUFFER)>
1440     INPUT1=' '
1450     CALL KEYIN<,14,ADDR(INPUT1)> GIVING<,NUM>
1460     IF NUM EQ 0 THEN GO TO NOGO
1470     DUMMY=',
1480 $ LDX #DUMMY
1490 $ STX PACK1
1500     PACK2=ADDR(UNIT)
1510 $ LDX #PACK1
1520 $ SCALL .PFNAM
1530 $ STAB STATUS
1540     IF STATUS NE 0
1550       THEN DO
1560         STATUS=7
1570         GO TO ERRUP
1580       END
1590     MODE=1
1600     TYPE='DK'
1610 $ LDX #IOCB
1620 $ SCALL .RESRV
1630     IF STATUS NE 0 THEN GO TO ERRUP
1640     BUFFS = ADDR(BUFF)
1650     BUFFE = 132+BUFFS
1660     FORMAT = 5
1670     ALLOC = 0
1680     SECTS = ADDR(SECT0)
1690     SECTE = 2559+SECTS
1700 $ LDX #IOCB
1710 $ SCALL .OPEN
1720     IF STATUS NE 0 THEN GO TO ERRUP
1730     CALL TERMIN
1740     CALL CARDIN
1750 $ LDX #OUT6
1760 $ SCALL .DSPLZ
1770     LNUM=1
1780 GO9:
1790     XX12=LNUM
1800     LNUM=LNUM+1
1801 GOX:
1810 $ LDX #IOCB
1820 $ SCALL .GETRC
1830     IF STATUS NE 0 THEN GO TO CLOSE
1840 $ LDX #OUT7
1850 $ SCALL .DSPLZ
1860 GO10:  DO INDEX=1 TO 129
1870     ONE=BUFF(INDEX)
1871     IF ONE EQ BUFEND AND INDEX EQ 1 THEN GO TO GOX
1880     CALL CARDOU
1890     IF ONE EQ BUFEND THEN GO TO ENDRC
1900 END
1910 ENDRC:CALL INSYS
1920     IF NUM EQ 101 THEN GO TO CLOSE
```

```
1930    GO TO GO9
1940  CLOSE:
1950    CALL DSPLY(, , ADDR(BUFEND))
1960  $ LDX #IOCB
1970  $ SCALL .CLOSE
1980    IF STATUS NE 0 THEN GO TO ERRUP
1990  $ LDX #IOCB
2000  $ SCALL .RELES
2010    IF STATUS NE 0 THEN GO TO ERRUP
2020    IF NUM EQ 101 THEN GO TO NOGO
2030    BUFFER='UPLOAD COMPLETED'
2040    CALL DSPLY(, , ADDR(BUFFER))
2050    STATE=0
2060    CALL CARDIN
2070    CALL TERMIN
2080    ONE=BUFEND
2090    CALL CARDOU
2100    GO TO TOP
2110  ERRUP:
2120  $ CLRB
2130  $ LDX #IOCB
2140  $ SCALL .MDERR
2150  NOGO:  BUFFER='UPLOAD ABORTED'
2160    CALL DSPLY(, , ADDR(BUFFER))
2170    GO TO TOP
2180  END
2190  TERMIN:  PROC
2200  $ LDA A TERMCT
2210  $ ASR A
2220  $ BCC ENDTRM
2230  $ LDA A TERMTR
2240  $ STA A ONE
2250    NUM=1
2260    RETURN
2270  ENDTRM:  NUM=0
2280    RETURN
2290  END
2300  CARDOU:  PROC
2310  $        LDA A ONE
2320  $TRANS1 LDA B CARDCT
2330  $        BIT B #02H
2340  $        BEQ TRANS1
2350  $        STA A CARDTR
2360    RETURN
2370  END
2380  CARDIN:  PROC
2390  $        LDA A CARDCT
2400  $        ASR A
2410  $        BCC ENDCRD
2420  $        LDA A CARDTR
2430  $        STA A ONE
2440    NUM=1
2450    RETURN
2460  ENDCRD:  NUM=0
2470    RETURN
2480  END
2490  TERMOU:  PROC
2500  $        LDA A ONE
2510  $TRANS2 LDA B TERMCT
2520  $        BIT B #02H
2530  $        BEQ TRANS2
2540  $        STA A TERMTR
2550    RETURN
2560  END
2570  INSYS: PROC
2580    LATCH3=60282
2590    LATCH2=65535
2600    CNTL3=$83
2610    CNTL2=$01
2620    CNTL1=$00
2630    PACK1=COUNT2
2640    TIME=0
2650    INDEX1=1
2660    BUFFER=' '
2670    DO WHILE TIME LE 10
2680      CALL CARDIN
2690      IF NUM NE 0 THEN DO
2700        IF ONE EQ CNTQ THEN GO TO GO1
2710        INDEX1=INDEX1+1
2720        IF INDEX1 EQ 60 THEN GO TO GO2
2730      END
2740      TIME=65535-COUNT2
2750    END
2760  GO2: BUFFER='COMMUNICATIONS TIMEOUT'
2770    CALL DSPLY(, , ADDR(BUFFER))
2780    NUM=101
2790    RETURN
2800  GO1:  NUM=0
2810    RETURN
2820  END
```

```
*****************************************************************
*                                                               *
* PROGRAM TO MAP THE RECORDED SPECTRAL CHARACTERISTICS TO THE   *
* FRAME OF REFERENCE OF THE POLYGRAPHIC PAPER RECORDING FOR     *
* CORRELATION WITH THE MANUALLY CLASSIFIED RECORD               *
*                                                               *
*****************************************************************
0010 CONVRT:
0020   PROC OPTIONS(MAIN)
0030 $ NAM CONVRT
0040   DCL
0050     1 IOCB(2),
0060       2 STATUS BIN(1),
0070       2 MODE BIN(1),
0080       2 BUFFP BIN(2),
0090       2 BUFFS BIN(2),
0100       2 BUFFE BIN(2),
0110       2 TYPE CHAR(2),
0120       2 FILE,
0130         3 UNIT CHAR(1),
0140         3 NAME CHAR(8),
0150         3 SUF CHAR(2),
0160       2 DUM0 BIN(2),
0170       2 FORMAT BIN(1),
0180       2 DUM1 BIN(3),
0190       2 DUM2 BIN(2),
0200       2 ALLOC BIN(2),
0210       2 SECTS BIN(2),
0220       2 SECTE BIN(2),
0230       2 SECTP BIN(2)
0240   DCL 1 TFILE,
0250         2 TUNIT CHAR(1),
0260         2 TNAME CHAR(8),
0270         2 TSUF  CHAR(2)
0280   DCL DUMMY CHAR(1)
0290   DCL INPUT1 CHAR(13)
0300   DCL BUFFER CHAR(40)
0310   DCL BUFEND CHAR(1) INIT($D)
0320   DCL SECT0 CHAR(256)
0330   DCL SECT1 CHAR(256)
0340   DCL SECT2 CHAR(256)
0350   DCL SECT3 CHAR(256)
0360   DCL SECT4 CHAR(256)
0370   DCL SECT5 CHAR(256)
0380   DCL SECT6 CHAR(256)
0390   DCL SECT7 CHAR(256)
0400   DCL SECT8 CHAR(256)
0410   DCL SECT9 CHAR(256)
0420   DCL SECT10 CHAR(256)
0430   DCL SECT11 CHAR(256)
0440   DCL SECT12 CHAR(256)
0450   DCL SECT13 CHAR(256)
0460   DCL SECT14 CHAR(256)
0470   DCL SECT15 CHAR(256)
0480   DCL SECT16 CHAR(256)
0490   DCL SECT17 CHAR(256)
0500   DCL SECT18 CHAR(256)
0510   DCL SECT19 CHAR(256)
0520   DCL PAGE BIN(2)
0530   DCL SECS BIN(1)
0540   DCL NOWVAL BIN(2)
0550   DCL NUM1 BIN(1)
0560   DCL NUM2 BIN(1)
0570   DCL NUM10 BIN(1)
0580   DCL INDEX1 BIN(1)
0590   DCL PACK1 BIN(2)
0600   DCL PACK2 BIN(2)
0610   DCL ONE SIGNED BIN(1)
0620   DCL POINT BIN(2)
0630   DCL POINT1 BIN(2)
0640   DCL BUFF1(256)
0650   DCL NUM BIN(1)
0660   DCL INDEX2 BIN(1)
0670   DCL BUFOUT CHAR(45) INIT(' ')
0680   DCL HOLD CHAR(4)
0690   DCL PAGMAX BIN(2)
0700   DCL PAGMIN BIN(2)
0710   DCL PAGREF BIN(2)
0720   DCL NOGOC BIN(1)
0730    BUFFER='ENTER INPUT HEX FILE NAME'
0740    CALL DSPLY<, , ADDR(BUFFER)>
0750    INPUT1=' '
0760    CALL KEYIN<,13,ADDR(INPUT1)> GIVING<,NUM>
0770    IF NUM EQ 0 THEN GO TO NOGO
0780    DUMMY=','
0790 $ LDX #DUMMY
0800 $ STX PACK1
0810    PACK2=ADDR(TFILE)
0820 $ LDX #PACK1
0830 $ SCALL .PFNAM
0840 $ STAB NUM
0850    IF NUM NE 0
0860      THEN DO
0870        STATUS(1)=7
0880        GO TO ERRUP
0890      END
0900    FILE(1)=TFILE
0910    TYPE(1)='DK'
0920    TYPE(2)='DK'
0930    MODE(1)=1
```

```
0940    MODE(2)=2
0950 $  LDX #IOCB
0960 $  SCALL .RESRV
0970    IF STATUS(1) NE 0 THEN GO TO ERRUP
0980    BUFFS(1)=ADDR(BUFF1)
0990    BUFFS(2)=ADDR(BUFOUT)
1000    BUFFE(1)=255+BUFFS(1)
1010    BUFFE(2)=44+BUFFS(2)
1020    FORMAT(1)=3
1030    FORMAT(2)=5
1040    ALLOC(1)=0
1050    ALLOC(2)=200
1060    SECTS(1)=ADDR(SECT0)
1070    SECTS(2)=ADDR(SECT10)
1080    SECTE(1)=2559+SECTS(1)
1090    SECTE(2)=2559+SECTS(2)
1100 $  LDX #IOCB
1110 $  SCALL .OPEN
1120    IF STATUS(1) NE 0 THEN GO TO ERRUP
1130    BUFFER='ENTER OUTPUT FILE NAME'
1140    CALL DSPLY<, , ADDR(BUFFER)>
1150    INPUT1=' '
1160    CALL KEYIN<,13,ADDR(INPUT1)> GIVING<,NUM>
1170    IF NUM EQ 0 THEN GO TO NOGO
1180    DUMMY=',*'
1190 $  LDX #DUMMY
1200 $  STX PACK1
1210    PACK2=ADDR(TFILE)
1220 $  LDX #PACK1
1230 $  SCALL .PFNAM
1240 $  STA B NUM
1250    IF NUM NE 0 THEN DO
1260      STATUS(1)=7
1270      GO TO ERRUP
1280    END
1290    FILE(2)=TFILE
1300 $  LDX #IOCB+37
1310 $  SCALL .RESRV
1320    IF STATUS(2) NE 0 THEN DO
1330      STATUS(1)=STATUS(2)
1340      GO TO ERRUP
1350    END
1360 $  LDX #IOCB+37
1370 $  SCALL .OPEN
1380    IF STATUS(2) NE 0 THEN DO
1390      STATUS(1)=STATUS(2)
1400      GO TO ERRUP
1410    END
1420    PACK1=ADDR(BUFOUT)
1430    INDEX2=1
1440    NOGOC=0
1450    NOWVAL=0
1460    NUM10=0
1470    NUM1=0
1480    BUFFER='ENTER STARTING PAGE #'
1490    CALL DSPLY<, , ADDR(BUFFER)>
1500    CALL KEYIN<,3,ADDR(INPUT1)> GIVING<,NUM>
1510    IF NUM EQ 0 THEN GO TO NOGO
1520    PACK1=ADDR(INPUT1)
1530    DO WHILE NUM GE 1
1540      CALL HEXIN
1550      IF NOGOC EQ 1 THEN GO TO NOGO
1560    END
1570    PAGE=NOWVAL
1580    NOWVAL=0
1590    BUFFER='ENTER STARTING SEC ON PAGE'
1600    CALL DSPLY<, , ADDR(BUFFER)>
1610    CALL KEYIN<,2,ADDR(INPUT1)> GIVING<,NUM>
1620    IF NUM EQ 0 THEN GO TO NOGO
1630    PACK1=ADDR(INPUT1)
1640    DO WHILE NUM GE 1
1650      CALL HEXIN
1660      IF NOGOC EQ 1 THEN GO TO NOGO
1670    END
1680    SECS=NOWVAL
1690    PAGMIN=0
1700    PAGMAX=30-SECS
1710    PAGREF=PAGMAX
1720 LOOP:
1730    POINT1=ADDR(BUFF1)
1740 $  LDX #IOCB
1750 $  SCALL .GETRC
1760    IF STATUS(1) EQ 9 THEN GO TO CLOSE
1770    IF STATUS(1) NE 0 THEN GO TO ERRUP
1780    CALL ASCII
1790    IF INDEX1 NE 0 THEN GO TO ERRUP
1800    GO TO LOOP
1810 CLOSE:
1820    IF INDEX2 NE 1 THEN DO
1830 $    LDX #IOCB+37
1840 $    SCALL .PUTRC
1850      IF STATUS(2) NE 0 THEN DO
1860        STATUS(1)=STATUS(2)
1870        GO TO ERRUP
1880      END
1890    END
1900 $  LDX #IOCB
1910 $  SCALL .CLOSE
1920    IF STATUS(1) NE 0 THEN GO TO ERRUP
1930 $  LDX #IOCB
```

```
1940 $ SCALL .RELES
1950   IF STATUS(1) NE 0 THEN GO TO ERRUP
1960 $ LDX #IOCB+37
1970 $ SCALL .CLOSE
1980   IF STATUS(2) NE 0 THEN DO
1990     STATUS(1)=STATUS(2)
2000     GO TO ERRUP
2010   END
2020 $ LDX #IOCB+37
2030 $ SCALL .RELES
2040   IF STATUS(2) NE 0 THEN DO
2050     STATUS(1)=STATUS(2)
2060     GO TO ERRUP
2070   END
2080   BUFFER='CONVERSION COMPLETE'
2090   CALL DSPLY(, , ADDR(BUFFER))
2100   CALL MDOS
2110 ERRUP:
2120 $ CLRB
2130 $ LDX #IOCB
2140 $ SCALL .MDERR
2150 NOGO: BUFFER='CONVERT ABORTED'
2160   CALL DSPLY(, , ADDR(BUFFER))
2170   CALL MDOS
2180 END
2190 ASCII: PROC
2200   DO POINT=POINT1 TO BUFFP(1)
2210     IF INDEX2 EQ 1 THEN DO
2220       NUM1=POINT->NUM1
2230       INDEX2=INDEX2+1
2240       GO TO CONT1
2250     END
2260     IF INDEX2 EQ 2 THEN DO
2270       NUM2=POINT->NUM2
2280       IF NUM2 GT 150 AND NUM1 NE NUM10 THEN NUM1=NUM10
2290       NUM10=NUM1
2300       PACK2=ADDR(NUM1)
2310       PACK2=PACK2->PACK2
2320       DO WHILE PACK2 GT PAGMAX
2330         PAGE=PAGE+1
2340         PAGMAX=PAGMAX+30
2350         PAGMIN=PAGMAX-30
2360       END
2370       PACK2=PACK2-PAGMIN
2380       IF PAGMAX EQ PAGREF THEN PACK2=PACK2+SECS
2390       PACK1->HOLD=PAGE
2400       PACK1=PACK1+5
2410       PACK1->HOLD=PACK2
2420       PACK1=PACK1+5
2430       INDEX2=INDEX2+1
2440       GO TO CONT1
2450     END
2460     PACK1->HOLD=POINT->ONE
2470     PACK1=PACK1+5
2480     INDEX2=INDEX2+1
2490     IF INDEX2 EQ 10 THEN DO
2500       PACK1=ADDR(BUFOUT)
2510       INDEX2=1
2520 $     LDX #IOCB+37
2530 $     SCALL .PUTRC
2540       BUFOUT=' '
2550       IF STATUS(2) NE 0 THEN DO
2560         STATUS(1)=STATUS(2)
2570         INDEX1=99
2580         RETURN
2590       END
2600     END
2610 CONT1:
2620   END
2630   INDEX1=0
2640   RETURN
2650   END
2660 HEXIN: PROC
2670   PACK2=NOWVAL
2680   DO INDEX1=1 TO 9
2690     NOWVAL=NOWVAL+PACK2
2700   END
2710   DUMMY=PACK1->DUMMY
2720 $ LDA A DUMMY
2730 $ SCALL .NUMD
2740 $ BCS NO
2750 $ STA A NUM1
2760   NOWVAL=NOWVAL + NUM1
2770   PACK1=PACK1+1
2780   NUM=NUM-1
2790   RETURN
2800 NO: NOGOC=1
2810   RETURN
2820 END
```

```
*****************************************************************
*                                                               *
* PROGRAM TO CONTROL THE RECORDING OF THE SPECTRAL ESTIMATES    *
* GENERATED BY THE 2920S                                        *
*                                                               *
*****************************************************************
0010 RECORD: PROC OPTIONS(MAIN)
0020 $ NAM RECORD
0030  DCL 1 IOCB,
0040          2 STATUS BIN(1),
0050          2 MODE   BIN(1),
0060          2 BUFFP  BIN(2),
0070          2 BUFFS  BIN(2),
0080          2 BUFFE  BIN(2),
0090          2 TYPE   CHAR(2) INIT('DK'),
0100          2 UNIT   CHAR(1),
0110          2 NAME   CHAR(8),
0120          2 SUF    CHAR(2),
0130          2 DUM0   BIN(2),
0140          2 FORMAT BIN(1),
0150          2 DUM1   BIN(3),
0160          2 DUM2   BIN(2),
0170          2 ALLOC  BIN(2),
0180          2 SECTS  BIN(2),
0190          2 SECTE  BIN(2),
0200          2 SECTP  BIN(2)
0210    DCL 1 BUFF0(100),
0220          2 BUFF1(256) BIN(1)
0230    DCL CNTL   BIN(1)
0240    DCL IRQST  LABEL INIT(INDATA)
0250    DCL SECTOR BIN(128)
0260    DCL BUFFER CHAR(40)
0270    DCL BUFEND CHAR(1) INIT(‡D)
0280    DCL 1 OUT,
0290          2 PRT1 CHAR(22) INIT('TRANSITIONS DETECTED:  '),
0300          2 PRT2 CHAR(5),
0310          2 PRT3 CHAR(1) INIT(‡04)
0320    DCL 1 BACK,
0330          2 PRT4 CHAR(1) INIT(‡08),
0340          2 PRT5 CHAR(1) INIT(‡08),
0350          2 PRT6 CHAR(1) INIT(‡08),
0360          2 PRT7 CHAR(1) INIT(‡08),
0370          2 PRT8 CHAR(1) INIT(‡08),
0380          2 PRT9 CHAR(5),
0390          2 PRT0 CHAR(1) INIT(‡04)
0400    DCL INPUT  CHAR(1)
0410    DCL PACK1  BIN(2)
0420    DCL PACK2  BIN(2)
0430    DCL INDEX1 BIN(1)
0440    DCL NUM    BIN(1)
0450    DCL NUM1   SIGNED BIN(1)
0460    DCL ONE    CHAR(1)
0470    DCL PAST(7) BIN(1)
0480    DCL NOW(7)  BIN(1)
0490    DCL SENSE(7) SIGNED BIN(1)
0500    DCL POINT3 BIN(2)
0510    DCL TRANS  BIN(2)
0520    DCL DIFF   SIGNED BIN(1)
0530    DCL XX     SIGNED BIN(1)
0540    DCL YY     SIGNED BIN(1)
0550    DCL ZZ     SIGNED BIN(1)
0560    DCL POINT  BIN(2)
0570    DCL POINT0 BIN(2)
0580    DCL POINT1 BIN(2)
0590    DCL POINT2 BIN(2)
0600    DCL COUNT2 BIN(2) DEF ‡E03C
0610    DCL LATCH3 BIN(2) DEF ‡E03E
0620    DCL LATCH2 BIN(2) DEF ‡E03C
0630    DCL CNTL3  BIN(1) DEF ‡E038
0640    DCL CNTL2  BIN(1) DEF ‡E039
0650    DCL CNTL1  BIN(1) DEF ‡E038
0660    DCL TERMCT BIN(1) DEF ‡FCF4
0670    DCL TERMTR BIN(1) DEF ‡FCF5
0680    DCL PIA1AD BIN(1) DEF ‡E010
0690    DCL PIA1AC BIN(1) DEF ‡E011
0700    DCL PIA1BD BIN(1) DEF ‡E012
0710    DCL PIA1BC BIN(1) DEF ‡E013
0720    DCL PIA2AD BIN(1) DEF ‡E020
0730    DCL PIA2AC BIN(1) DEF ‡E021
0740    DCL PIA2BD BIN(1) DEF ‡E022
0750    DCL PIA2BC BIN(1) DEF ‡E023
0760    DCL PIA3AD BIN(1) DEF ‡E030
0770    DCL PIA3AC BIN(1) DEF ‡E031
0780    DCL PIA3BD BIN(1) DEF ‡E032
0790    DCL PIA3BC BIN(1) DEF ‡E033
0800    DCL PIA4AD BIN(1) DEF ‡E034
0810    DCL PIA4AC BIN(1) DEF ‡E035
0820    DCL PIA4BD BIN(1) DEF ‡E036
0830    DCL PIA4BC BIN(1) DEF ‡E037
0840 $ LDX CBUFP$
0850 $ STX PACK1
0860    PACK2=ADDR(UNIT)
0870 $ LDX #PACK1
0880 $ SCALL .PFNAM
0890 $ STA B STATUS
0900    IF STATUS GT 1 THEN DO
0910      STATUS=7
0920      GO TO ERR
0930    END
0940    PIA1AD=0
```

```
0950    PIA1BD=0
0960    PIA2AD=0
0970    PIA2BD=0
0980    PIA3AD=0
0990    PIA3BD=0
1000    PIA4AD=0
1010    PIA4BD=0
1020    PIA1AC=4
1030    PIA1BC=4
1040    PIA2AC=4
1050    PIA2BC=4
1060    PIA3AC=4
1070    PIA3BC=4
1080    PIA4AC=4
1090    PIA4BC=4
1100    TERMCT=$03
1110    TERMCT=$49
1120 $  LDX IRQST
1130 $  STX IRQ$UV
1140    MODE=2
1150    BUFFS=ADDR(BUFF0)
1160    BUFFE=99+BUFFS
1170    FORMAT=3
1180    ALLOC=100
1190    SECTS=ADDR(SECTOR)
1200    SECTE=127+SECTS
1210 $  LDX #IOCB
1220 $  SCALL .RESRV
1230    IF STATUS NE 0 THEN GO TO ERR
1240 $  LDX #IOCB
1250 $  SCALL .OPEN
1260    IF STATUS NE 0 THEN GO TO ERR
1270    POINT1=ADDR(BUFF0)
1280    POINT2=POINT1+25599
1290    CNTL=0
1300    TRANS=0
1310    POINT0=ADDR(PAST)
1320    DO INDEX1=1 TO 7
1330      POINT0->NUM=0
1340      POINT0=POINT0+1
1350    END
1360    SENSE(1)=3
1370    SENSE(2)=3
1380    SENSE(3)=3
1390    SENSE(4)=3
1400    SENSE(5)=3
1410    SENSE(6)=3
1420    SENSE(7)=25
1430    BUFFER='ONCE RECORDING HAS BEGUN,'
1440    CALL DSPLY(, , ADDR(BUFFER))
1450    BUFFER='ENTER ESC TO TERMINATE'
1460    CALL DSPLY(, , ADDR(BUFFER))
1470    BUFFER='RETURN TO START RECORDING'
1480    CALL DSPLY(, , ADDR(BUFFER))
1490 $  LDA B #00H
1500 $  LDX #INPUT
1510 $  SCALL .KEYIN
1520    LATCH3=60282
1530    LATCH2=65535
1540    CNTL3=$83
1550    CNTL2=$01
1560    CNTL1=$00
1570    PACK1=COUNT2
1580    PRT2=0
1590 $  LDX #OUT
1600 $  SCALL .DSPLZ
1610    PIA1AC=12
1620 LOOP:
1630 $  LDA A TERMCT
1640 $  ASR A
1650 $  BCC NOIN
1660    PIA1AC=4
1670    BUFFER=' '
1680    CALL DSPLY(, , ADDR(BUFFER))
1690    GO TO OUTREQ
1700 NOIN: IF CNTL NE 0 THEN GO TO OUTNOW
1710    GO TO LOOP
1720 OUTNOW: PIA1AC=4
1730    BUFFER=' '
1740    CALL DSPLY(, , ADDR(BUFFER))
1750    BUFFER='MEMORY LIMIT REACHED'
1760    CALL DSPLY(, , ADDR(BUFFER))
1770 OUTREQ: BUFFER='RECORDING TERMINATED'
1780    CALL DSPLY(, , ADDR(BUFFER))
1790    POINT0=BUFFS
1800    DO WHILE BUFFE LE POINT1
1810 $    LDX #IOCB
1820 $    SCALL .PUTRC
1830      IF STATUS NE 0 THEN GO TO ERR
1840      BUFFS=BUFFS+100
1850      BUFFE=BUFFS+99
1860    END
1870    IF BUFFE NE POINT1 THEN DO
1880      BUFFE=POINT1
1890 $    LDX #IOCB
1900 $    SCALL .PUTRC
1910      IF STATUS NE 0 THEN GO TO ERR
1920    END
1930 $  LDX #IOCB
1940 $  SCALL .CLOSE
```

```
1950    IF STATUS NE 0 THEN GO TO ERR
1960 $  LDX #IOCB
1970 $  SCALL .RELES
1980    IF STATUS NE 0 THEN GO TO ERR
1990    CALL MDOS
2000 ERR:
2010 $  CLRB
2020 $  LDX #IOCB
2030 $  SCALL .MDERR
2040    CALL MDOS
2050 END
2060 INDATA: PROC
2070    POINT=ADDR(NOW)
2080    POINT0=ADDR(PAST)
2090    POINT3=ADDR(SENSE)
2100    NOW(1)=PIA1AD
2110    NOW(2)=PIA1BD
2120    NOW(3)=PIA2AD
2130    NOW(4)=PIA2BD
2140    NOW(5)=PIA3AD
2150    NOW(6)=PIA3BD
2160    NOW(7)=PIA4AD
2170    DO INDEX1=1 TO 7
2180      XX=POINT->NUM1
2190      YY=POINT0->NUM1
2200      ZZ=POINT3->NUM1
2210      DIFF=0
2220      IF XX GT YY THEN DIFF=XX-YY
2230      IF YY GT XX THEN DIFF=YY-XX
2240      IF DIFF GE ZZ THEN GO TO CHANGE
2250      POINT=POINT+1
2260      POINT0=POINT0+1
2270      POINT3=POINT3+1
2280    END
2290 $  RTI
2300 CHANGE: POINT=ADDR(NOW)
2310    POINT0=ADDR(PAST)
2320    POINT1->PACK1=65535-COUNT2
2330    POINT1=POINT1+2
2340    DO INDEX1=1 TO 7
2350      POINT0->NUM=POINT->NUM
2360      POINT1->NUM=POINT->NUM
2370      POINT=POINT+1
2380      POINT0=POINT0+1
2390      POINT1=POINT1+1
2400    END
2410    TRANS=TRANS+1
2420    PRT9=TRANS
2430 $  LDX #BACK
2440 $  SCALL .DSPLZ
2450    PACK1=POINT2-POINT1
2460    IF PACK1 LT 9 THEN CNTL=1
2470 $  RTI
2480 END
```

```
*****************************************************************
*                                                               *
* PROGRAM TO VERIFY CORRECT OPERATION OF 2920S ON POWERUP       *
*                                                               *
*****************************************************************
0010 POWER: PROC OPTIONS(MAIN)
0020 $ NAM POWER
0030   DCL IRQST   LABEL INIT(INDATA)
0040   DCL OUT     CHAR(45)
0050   DCL OUTEND CHAR(1) INIT(0D)
0060   DCL INDEX1 BIN(1)
0070   DCL NUM1    SIGNED BIN(
0080   DCL ONE1    CHAR(5)
0090   DCL NOW(7) BIN(1)
0100   DCL POINT   BIN(2)
0110   DCL POINT1 BIN(2)
0120   DCL TERMCT BIN(1) DEF $FCF4
0130   DCL TERMTR BIN(1) DEF $FCF5
0140   DCL PIA1AD BIN(1) DEF $E010
0150   DCL PIA1AC BIN(1) DEF $E011
0160   DCL PIA1BD BIN(1) DEF $E012
0170   DCL PIA1BC BIN(1) DEF $E013
0180   DCL PIA2AD BIN(1) DEF $E020
0190   DCL PIA2AC BIN(1) DEF $E021
0200   DCL PIA2BD BIN(1) DEF $E022
0210   DCL PIA2BC BIN(1) DEF $E023
0220   DCL PIA3AD BIN(1) DEF $E030
0230   DCL PIA3AC BIN(1) DEF $E031
0240   DCL PIA3BD BIN(1) DEF $E032
0250   DCL PIA3BC BIN(1) DEF $E033
0260   DCL PIA4AD BIN(1) DEF $E034
0270   DCL PIA4AC BIN(1) DEF $E035
0280   DCL PIA4BD BIN(1) DEF $E036
0290   DCL PIA4BC BIN(1) DEF $E037
0300    PIA1AD=0
0310    PIA1BD=0
0320    PIA2AD=0
0330    PIA2BD=0
0340    PIA3AD=0
0350    PIA3BD=0
0360    PIA4AD=0
0370    PIA4BD=0
0380    PIA1AC=4
0390    PIA1BC=4
0400    PIA2AC=4
0410    PIA2BC=4
0420    PIA3AC=4
0430    PIA3BC=4
0440    PIA4AC=4
0450    PIA4BC=4
0460    TERMCT=$03
0470    TERMCT=$49
0480    OUT=' '
0490 $ LDX IRQST
0500 $ STX IRQ+UV
0510    PIA1AC=12
0520 LOOP:
0530 $ LDA A TERMCT
0540 $ ASR A
0550 $ BCC LOOP
0560    PIA1AC=4
0570    CALL MDOS
0580 END
0590 INDATA: PROC
0600    POINT=ADDR(NOW)
0610    NOW(1)=PIA1AD
0620    NOW(2)=PIA1BD
0630    NOW(3)=PIA2AD
0640    NOW(4)=PIA2BD
0650    NOW(5)=PIA3AD
0660    NOW(6)=PIA3BD
0670    NOW(7)=PIA4AD
0680    POINT1=ADDR(OUT)
0690    DO INDEX1=1 TO 7
0700      POINT1->ONE1=POINT->NUM1
0710      POINT1=POINT1+5
0720      POINT=POINT+1
0730    END
0740    CALL DSPLY<, , ADDR(OUT)>
0750 $ RTI
0760 END
```

```
/*****************************************************************
*                                                                *
* PROGRAM TO ASSEMBLE 2920 PROGRAMS                              *
*                                                                *
*****************************************************************
0010 AS2920: PROC OPTIONS(MAIN)                  /* MAINLINE ROUTINE          */
0020 $ NAM AS2920                                /*                           */
0030 DECLARE                                     /*                           */
0040     1   IOCB(3),                           /* FILE I/O CONTROL BLOCKS   */
0050         2   STATUS BIN (1),               /* RETURN CODE               */
0060         2   MODE   BIN (1),               /* INPUT, OUTPUT,OR UPDATE   */
0070         2   BUFFP  BIN (2),               /* RECORD BUFFER POINTER     */
0080         2   BUFFS  BIN (2),               /* RECORD BUFFER START ADDR  */
0090         2   BUFFE  BIN (2),               /* RECORD BUFFER END ADDRESS */
0100         2   TYPE   CHAR(2),               /* DEVICE TYPE               */
0110         2   FILE,                         /* FILE NAME STRUCTURE       */
0120             3   UNIT  CHAR(1),           /* LOGICAL UNIT NUMBER       */
0130             3   NAME  CHAR(8),           /* FILE NAME                 */
0140             3   SUF   CHAR(2),           /* FILE SUFFIX               */
0150         2   DUM0   BIN (2),               /* RESERVED                  */
0160         2   FORMAT BIN (1),               /* BIN, ASCII, OR ASCII CONV */
0170         2   DUM1   BIN (5),               /* RESERVED                  */
0180         2   ALLOC  BIN (2),               /* ALLOCATION INCREMENT      */
0190         2   SECTS  BIN (2),               /* SECTOR BUFFER START ADDR  */
0200         2   SECTE  BIN (2),               /* SECTOR BUFFER END ADDRESS */
0210         2   SECTP  BIN (2)                /* SECTOR BUFFER POINTER     */
0220 DECLARE                                     /*                           */
0230     1   TFILE,                            /* TEMP FILE NAME STRUCTURE  */
0240         2   TUNIT  CHAR(1),               /* LOGICAL UNIT NUMBER       */
0250         2   TNAME  CHAR(8),               /* FILE NAME                 */
0260         2   TSUF   CHAR(2)                /* FILE SUFFIX               */
0270 DECLARE                                     /*                           */
0280     1   CNTL,                             /* ASSEMBLY CONTROL          */
0290         2   PRINT  BIN (1),               /* LISTING PRINT CONTROL     */
0300         2   OBJECT BIN (1),               /* OBJECT FILE CONTROL       */
0310         2   PAGEL  BIN (1),               /* PAGE LENGTH CONTROL       */
0320         2   INST   BIN (1) INIT(0),       /* EPROM INSTRUCTION COUNTER */
0330         2   LINE   BIN (1),               /* LISTING LINE COUNTER      */
0340         2   PAGE   BIN (1) INIT(1),       /* PAGE COUNTER              */
0350         2   ERROR  BIN (1) INIT(0),       /* ERROR COUNT               */
0360         2   RTRN   BIN (1),               /* RETURN CODE               */
0370         2   KIND   BIN (1),               /* TYPE OF TOKEN DECODED     */
0380         2   RTRNT  BIN (1),               /* TOKEN PARSER RETURN CODE  */
0390         2   VAR    BIN (1) INIT(0),       /* SYMBOL TABLE POINTER      */
0400         2   STATE  BIN (1),               /* STATE INDICATOR           */
0410         2   ENDYET BIN (1)                /* SCAN REMAINDER OF RECORD? */
0420 DCL     BUFFER CHAR(78)                   /* RECORD I/O BUFFER         */
0430 DCL     BUFEND CHAR(1) INIT($D)           /* CR MARKER                 */
0440 DCL     BUFEOT CHAR(1) INIT($4)           /* EOT CHARACTER             */
0450 DCL     SECT1  CHAR(128)                  /* INPUT FILE SECTOR BUFFER  */
0460 DCL     SECT2  CHAR(256)                  /* LISTING FILE SECTOR BUFFER*/
0470 DCL     SECT3  CHAR(128)                  /* OBJECT FILE SECTOR BUFFER */
0480 DCL     PARSE1 BIN (2)                    /* MDOS PARSER BUFFER        */
0490 DCL     PARSE2 BIN (2)                    /* MDOS PARSER BUFFER        */
0500 DCL     EPROMP BIN (2)                    /* EPROM POINTER             */
0510 DECLARE                                     /*                           */
0520     1   EPROM(192),                       /* EPROM MEMORY SPACE        */
0530         2  SEGE1  BIN(1),                 /* MOST SIGNIFICANT BYTE     */
0540         2  SEGE2  BIN(1),                 /* MIDDLE BYTE               */
0550         2  SEGE3  BIN(1)                  /* LEAST SIGNIFICANT BYTE    */
0560 DECLARE                                     /*                           */
0570     1   BUFFO,                            /* OBJECT FILE I/O STRUCTURE */
0580         2  SEG1   BIN(1),                 /* MOST SIGNIFICANT BUTE     */
0590         2  SEG2   BIN(1),                 /* MIDDLE BYTE               */
0600         2  SEG3   BIN(1)                  /* LEAST SIGNIFICANT BYTE    */
0610 DCL     RAMNAM (40) CHAR(8) INIT('  ')    /* RAM SYMBOL TABLE          */
0620 DCL     ENDRAM CHAR(1) INIT($D)
0630 DCL     INDEX1 BIN (1)                    /* GENERAL UTILITY COUNTER   */
0640 DCL     INDEX2 BIN (1)                    /* GENERAL UTILITY COUNTER   */
0650 DCL     INDEX3 BIN (1)                    /* GENERAL UTILITY COUNTER   */
0660 DCL     NUM    BIN (1)                    /* GENERAL UTILITY BIN(1)    */
0670 DCL     BUFF(80) CHAR(1)                  /* GENERAL CHAR BY 1 BUFFER  */
0680 DCL     BUFFI  CHAR(80)                   /* INPUT FILE BUFFER         */
0690 DCL     BUFFL  CHAR(132)                  /* LISTING FILE I/O BUFFER   */
0700 DCL     LAST   BIN (2)                    /* GENERAL UTILITY POINTER   */
0710 DCL     ONE    CHAR(1)                    /* GENERAL UTILITY CHAR      */
0720 DECLARE                                     /*                           */
0730     1   FOUR,                             /* FOUR CHAR STRUCTURE       */
0740         2  FOURX (4) CHAR(1)              /* ADDRESSABLE BY ONE        */
0750 DECLARE                                     /*                           */
0760     1   HEADER,                           /* ASSEMBLER TITLES          */
0770         2  CPT0   CHAR(1) INIT($C),       /* FORM FEED CHARACTER       */
0780         2  HEAD1,                         /* TOP OF PAGE MESSAGE       */
0790            3   CPT1   CHAR(31) INIT('MDOS 2920 ASSEMBLER VER 1.00'),
0800            3   CPT20  CHAR(12) INIT('  '),
0810            3   CPT21  CHAR(8),
0820            3   CPT22  CHAR(1) INIT('.'),
0830            3   CPT23  CHAR(2),
0840            3   CPT24  CHAR(1) INIT(':'),
0850            3   CPT25  CHAR(1),
0860            3   CPT26  CHAR(16) INIT('  '),
0870            3   CPT3   CHAR(5) INIT('PAGE '),
0880            3   CPT4   CHAR(2),
0890            3   CPT5   CHAR(1) INIT($D)
0900 DECLARE                                     /*                           */
0910     1   HEAD2,                            /* SECOND LINE OF HEADER     */
0920         2  CPT14  CHAR(27) INIT('LOC OBJECT SOURCE STATEMENT'),
0930         2  CPT15  CHAR(1) INIT($D),
0940         2  CPT16  CHAR(1) INIT($D)
0950 DECLARE                                     /*                           */
```

```
0960     1   OUT,                                    /* GENERAL OUTPUT LINE SHELL   */
0970         2  LOC     CHAR(3),                    /* MEMORY LOCATION             */
0980         2  CPT12   CHAR(1) INIT(' '),          /*                             */
0990         2  INSTL   CHAR(6) INIT(' '),          /* INSTRUCTION (HEX) SPACE     */
1000         2  CPT13   CHAR(1) INIT(' '),          /*                             */
1010         2  ININST  CHAR(80)                    /* SUPPLIED SOURCE STATEMENT   */
1020 DCL THREE BIN(3)                                /* UTILITY 3 BYTE VARIABLE     */
1030 DECLARE                                         /*                             */
1040     1   MESSGE,                                 /* GENERAL MESSAGE STRUCTURE   */
1050         2  P0    CHAR(40),                     /* MESSAGE PART                */
1060         2  P1    CHAR(1) INIT(¢D)              /* LINE FEED/CR                */
1070 DECLARE                                         /*                             */
1080     1   TOTALS,                                 /* END OD ASSEM-TOTAL ERRORS   */
1090         2  MESS0 CHAR(14) INIT('TOTAL ERRORS: '),
1100         2  NUMPRT CHAR(3),                     /* NUMBER OF ERRORS            */
1110         2  ENDPRT CHAR(1) INIT(¢D)             /* CR                          */
1120 DCL  MESS1 CHAR(40) INIT('* ERROR:  TOKEN OVER 8 CHAR')
1130 DCL  MESS2 CHAR(40) INIT('* ERROR:  INVALID SYNTAX')
1140 DCL  MESS3 CHAR(40) INIT('* ERROR:  OUT AFTER DAR AS DEST')
1150 DCL  MESS4 CHAR(40) INIT('* ERROR:  OUT AFTER COND SUB')
1160 DCL  MESS5 CHAR(40) INIT('* ERROR:  CVT AFTER DAR AS DEST')
1170 DCL  MESS6 CHAR(40) INIT('* ERROR:  CVT AFTER CVT')
1180 DCL  MESS7 CHAR(40) INIT('* ERROR:  CVT AFTER COND SUB')
1190 DCL  MESS8 CHAR(40) INIT('* ERROR:  CVT WITH DAR AS DEST')
1200 DCL  MESS9 CHAR(40) INIT('* ERROR:  CND USED WITH AND')
1210 DCL  MESS10 CHAR(40) INIT('* ERROR:  CND USED WITH LIM')
1220 DCL  MESS11 CHAR(40) INIT('* ERROR:  CND USED WITH ABS')
1230 DCL  MESS12 CHAR(40) INIT('* ERROR:  CND/SUB & DAR AS DEST')
1240 DCL  MESS13 CHAR(40) INIT('* ERROR:  LIM WITH SHIFT NOT R0')
1250 DCL  MESS14 CHAR(40) INIT('* ERROR:  MORE THAN 40 VAR USED')
1260 DCL  MESS15 CHAR(40) INIT('* ERROR:  END-EQU NOT ALLOWED')
1270 DCL  MESS16 CHAR(40) INIT('* ERROR:  INVALID VAR NAME')
1280 DCL  DAROUT(2) BIN(1)
1290 DCL  CONSUB(2) BIN(1)
1300 DCL  CVTX(2)   BIN(1)
1310 DCL  ANDX BIN(1)
1320 DCL  LIMX BIN(1)
1330 DCL  ABSX BIN(1)
1340 DCL  SUBX BIN(1)
1350 DECLARE                                         /*                             */
1360     1   SCREEN,                                 /*                             */
1370         2  CPT6    CHAR(30) INIT('MDOS 2920 ASSEMBLER  VER 1.00'),
1380         2  CPT7    CHAR(1)  INIT(¢D),
1390         2  CPT10   CHAR(1)  INIT(¢D),
1400         2  CPT11   CHAR(1)  INIT(¢4)
1410 DECLARE                                         /*                             */
1420     1   TOKENA,                                 /* TOKEN STRUCTURE             */
1430         2  TOKEN (8) CHAR(1)                    /* ADDRESSABLE BY 1            */
1440 DECLARE                                         /*                             */
1450     1   TOP,                                    /* 24 BIT INSTRUCTION STRUCT   */
1460         2  ADF0    BIT (1),                    /* I/O CODE                    */
1470         2  ADK2    BIT (1),                    /* I/O CODE                    */
1480         2  ADK1    BIT (1),                    /* I/O CODE                    */
1490         2  ADK0    BIT (1),                    /* I/O CODE                    */
1500         2  A2      BIT (1),                    /* SOURCE FIELD OPERAND        */
1510         2  B1      BIT (1),                    /* DESTINATION FIELD OPERAND   */
1520         2  A1      BIT (1),                    /* SOURCE FIELD OPERAND        */
1530         2  ADF1    BIT (1),                    /* I/O CODE                    */
1540         2  A4      BIT (1),                    /* SOURCE FIELD OPERAND        */
1550         2  B3      BIT (1),                    /* DESTINATION FIELD OPERAND   */
1560         2  A3      BIT (1),                    /* SOURCE FIELD OPERAND        */
1570       , 2  B2      BIT (1),                    /* DESTINATION FIELD OPERAND   */
1580         2  A0      BIT (1),                    /* SOURCE FIELD OPERAND        */
1590         2  B5      BIT (1),                    /* DESTINATION FIELD OPERAND   */
1600         2  A5      BIT (1),                    /* SOURCE FIELD OPERAND        */
1610         2  B4      BIT (1),                    /* DESTINATION FIELD OPERAND   */
1620         2  S2      BIT (1),                    /* SHIFT CODE                  */
1630         2  S1      BIT (1),                    /* SHIFT CODE                  */
1640         2  S0      BIT (1),                    /* SHIFT CODE                  */
1650         2  B0      BIT (1),                    /* DESTINATION FIELD OPERAND   */
1660         2  L2      BIT (1),                    /* OP CODE                     */
1670         2  L1      BIT (1),                    /* OP CODE                     */
1680         2  L0      BIT (1),                    /* OP CODE                     */
1690         2  S3      BIT (1)                     /* SHIFT CODE                  */
1700 DECLARE                                         /*                             */
1710     1   BITUP,                                  /* BYTE ADDRESSABLE BY BIT     */
1720         2  BIT7    BIT (1),                    /* MS BIT                      */
1730         2  BIT6    BIT (1),                    /*                             */
1740         2  BIT5    BIT (1),                    /*                             */
1750         2  BIT4    BIT (1),                    /*                             */
1760         2  BIT3    BIT (1),                    /*                             */
1770         2  BIT2    BIT (1),                    /*                             */
1780         2  BIT1    BIT (1),                    /*                             */
1790         2  BIT0    BIT (1)                     /* LS BIT                      */
1800 DECLARE                                         /*                             */
1810     1   BREAK,                                  /* BY ONE CHAR ADDRESSABLE     */
1820         2  CHAR6    CHAR (1),                  /* MOST SIGNIFICANT CHAR       */
1830         2  CHAR5    CHAR (1),                  /*                             */
1840         2  CHAR4    CHAR (1),                  /*                             */
1850         2  CHAR3    CHAR (1),                  /*                             */
1860         2  CHAR2    CHAR (1),                  /*                             */
1870         2  CHAR1    CHAR (1)                   /*                             */
1880 DCL OPCODE (87) CHAR(4)                         /* TABLE OF OPCODES            */
1890    INIT('XOR','AND','LIM','ABS','ABA','SUB','ADD','LDA',
1900         'IN0','IN1','IN2','IN3','NOP','EOP','CVTS','CNDS',
1910         'OUT0','OUT1','OUT2','OUT3','OUT4','OUT5','OUT6','OUT7',
1920         'CVT0','CVT1','CVT2','CVT3','CVT4','CVT5','CVT6','CVT7',
1930         'CND0','CND1','CND2','CND3','CND4','CND5','CND6','CND7',
1940         'R01','R02','R03','R04','R05','R06','R07','R08','R09','R10',
1950         'R11','R12','R13','L02','L01','R00','R0','R1','R2','R3','R4',
```

```
1960          'R5','R6','R7','R8','R9','L2','L1','KM1','KM2','KM3',
1970          'KM4','KM5','KM6','KM7','KM8','KP0','KP1','KP2','KP3',
1980          'KP4','KP5','KP6','KP7','DAR','EQU','END')
1990 DCL     CODE  (87) BIN (1)                    /* OPCODE BIT STRINGS          */
2000      INIT(0,1,2,3,4,5,6,7,0,4,8,12,16,20,24,28,1,5,9,13,17,21,25,
2010           29,2,6,10,14,18,22,26,30,3,7,11,15,19,23,27,31,0,1,2,3,
2020           4,5,6,7,8,9,10,11,12,13,14,15,15,0,1,2,3,4,5,6,7,8,13,14,
2030           63,62,61,60,59,58,57,56,48,49,50,51,52,53,54,55,40,0,0)
2040 DCL     CODET (87) BIN (1)                    /* OPCODE TYPES                */
2050      INIT(1,1,1,1,1,1,1,1,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
2060           5,5,5,5,5,5,5,5,5,5,5,5,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
2070           4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,3,3,3,3,3,3,3,3,3,3,3,
2080           3,3,3,3,3,3,2,6,6)
2090 DCL     CBUFP  BIN (2)                      /* MDOS COMMAND LINE POINTER   */
2100 DCL     OFFSET BIN (2)                      /* OFFSET FOR IOCB             */
2110 DCL     INAME  CHAR(8)                      /* DEFAULT FILE NAME HOLD      */
2120 DCL     IUNIT  CHAR(1)                      /* DEFAULT FILE UNIT HOLD      */
2130 DECODE:                                     /* DECODE COMMAND INPUT        */
2140   DAROUT(1)=0
2150   DAROUT(2)=0
2160   CONSUB(1)=0
2170   CONSUB(2)=0
2180   CVTX(1)=0
2190   CVTX(2)=0
2200   ANDX=0
2210   LIMX=0
2220   PRINT=0                                   /* SET DEFAULTS: NO LISTING    */
2230   ABSX=0
2240   SUBX=0
2250   OBJECT=0                                  /* NO OBJECT CODE              */
2260   PAGEL=62                                  /* PAGELENGTH = 66 LINES       */
2270   TUNIT='0'                                 /* INPUT FILE DEFAULT UNIT 0   */
2280   TSUF='SA'                                 /* INPUT FILE SUFFIX 'SA'      */
2290 $ LDX CBUFP$                                /* LOAD MDOS POINTER           */
2300 $ STX CBUFP                                 /* SAVE FOR MPL                */
2310   LAST=CBUFP                                /* POINT TO CURRENT MDOS END   */
2320 LOOP: IF LAST->ONE EQ BUFEND THEN GO TO CONT0
2330   LAST=LAST+1                               /* INCREMENT END OF LINE CNTR  */
2340   GO TO LOOP                                /* AND REPEAT CHECK            */
2350 CONT0: PARSE1=CBUFP                         /* POINT TO START OF COMMAND   */
2360   PARSE2=ADDR(TFILE)                        /* SET UP FILE NAME CNTL BLK   */
2370 $ LDX #PARSE1                               /* LOAD X WITH ADDR OF CNTL B  */
2380 $ SCALL .PFNAM                              /* PARSE OUT NAME              */
2390 $ STA B INDEX2                              /* STORE RETURN CODE           */
2400   IF INDEX2 LE 1 THEN GO TO CONT1           /* OF OK, CONTINUE             */
2410   INDEX1=7                                  /* ELSE: SET ERROR STATE       */
2420   GO TO ERR1                                /* AND MOP UP-END              */
2430 CONT1: FILE(1)=TFILE                        /* MOVE TEMP STRUCT TO IOCB    */
2440   CPT21=TNAME                               /* MOVE INPUT NAME TO PAGE HDR */
2450   CPT23=TSUF                                /* MOVE INPUT NAME TO PAGE HDR */
2460   CPT25=TUNIT                               /* MOVE INPUT UNIT TO PAGE HDR */
2470   INAME=TNAME                               /* SAVE DEFAULT FILE NAME      */
2480   IUNIT=TUNIT                               /* SAVE DEFAULT UNIT NUMBER    */
2490   PARSE1=PARSE1+1                           /* MOVE POINTER TO NEXT CHAR   */
2500   TYPE(1)='DK'                              /* DEVICE TYPE DISK            */
2510   TYPE(2)='DK'                              /*                             */
2520   TYPE(3)='DK'                              /*                             */
2530   MODE(1)=1                                 /* INPUT FILE: OLD FILE INPUT  */
2540   MODE(2)=2                                 /* LISTING FILE: NEW FILE OUTPU*/
2550   MODE(3)=2                                 /* OBJECT FILE: NEW FILE OUTPUT*/
2560   BUFFS(1)=ADDR(BUFFI)                      /* POINT TO START OF RECORD    */
2570   BUFFS(2)=ADDR(BUFFL)                      /* I/O BUFFERS                 */
2580   BUFFS(3)=ADDR(EPROM)                      /*                             */
2590   EPROMP=ADDR(EPROM)                        /* SAVE START ADDRESS OF EPROM */
2600   BUFFE(1)=BUFFS(1)+79                      /* POINT TO END OF RECORD      */
2610   BUFFE(2)=BUFFS(2)+131                     /* I/O BUFFERS                 */
2620   BUFFE(3)=BUFFS(3)+71                      /*                             */
2630   FORMAT(1)=5                               /* ASCII FILES                 */
2640   FORMAT(2)=5                               /*                             */
2650   FORMAT(3)=3                               /*                             */
2660   SECTS(1)=ADDR(SECT1)                      /* POINT TO START OF SECTOR    */
2670   SECTS(2)=ADDR(SECT2)                      /* I/O BUFFERS                 */
2680   SECTS(3)=ADDR(SECT3)                      /*                             */
2690   SECTE(1)=SECTS(1)+127                     /* POINT TO END OF SECTOR      */
2700   SECTE(2)=SECTS(2)+255                     /* I/O BUFFERS                 */
2710   SECTE(3)=SECTS(3)+127                     /*                             */
2720   ALLOC(1)=0                                /* NO NEW SPACE REQUIRED       */
2730   ALLOC(2)=100                              /* 100 SECTORS FOR LISTING     */
2740   ALLOC(3)=10                               /* 10 SECTORS FOR OBJECT CODE  */
2750 $ LDX #IOCB                                 /* LOAD IOCB ADDRESS FOR SCALL */
2760 $ SCALL .RESRV                              /* RESERVE DISK : INPUT FILE   */
2770   IF STATUS(1) NE 0 THEN DO                 /* IF ERROR IN RESERVING, SET  */
2780     INDEX1=STATUS(1)                        /* ERROR CODE AND GO TO ERROR  */
2790     GO TO ERR1                              /* ROUTINE                     */
2800   END                                       /*                             */
2810 $ LDX #IOCB                                 /* POINT TO INPUT IOCB         */
2820 $ SCALL .OPEN                               /* OPEN INPUT FILE             */
2830   IF STATUS(1) NE 0 THEN DO                 /* IF ERROR OPENING, SET ERROR */
2840     INDEX1=STATUS(1)                        /* CODE AND GO TO ERROR        */
2850     GO TO ERR2                              /* ROUTINE                     */
2860   END                                       /*                             */
2870 LOOP1: DO WHILE PARSE1 LT LAST              /* SCAN OPTIONS IN COMMAND LIN */
2880   IF PARSE1->ONE EQ ' ' THEN DO             /* IF A BLANK, PASS OVER       */
2890     PARSE1=PARSE1+1                         /* INCREMENT POINTER           */
2900     GO TO LOOP1                             /* AND REPEAT SCAN             */
2910   END                                       /*                             */
2920   IF PARSE1-> ONE EQ 'O' THEN DO            /* IF OPTION IS OBJECT FILE    */
2930     INDEX3=3                                /* PROC : INDICATE OBJECT FILE */
2940     OFFSET=74                               /* SET IOCB OFFSET             */
2950     TSUF='LX'                               /* SET DEFAULT FILE TYPE       */
```

```
2960      CALL OPTION GIVING<OBJECT, ,>          /* PROCESS OPTIONS              */
2970      GO TO (LOOP1,ERR,ERR7),RTRN            /* CHECK RETURN CODE            */
2980    END                                      /* END OBJECT FILE PROCESSING */
2990    IF PARSE1-> ONE EQ 'L' THEN DO           /* LISTING FILE OPTION ?        */
3000      INDEX3=2                               /* INDICATE PRINT FILE          */
3010      OFFSET=37                              /* SET IOCB OFFSET              */
3020      TSUF='AL'                              /* SET DEFAULT FILE TYPE        */
3030      CALL OPTION GIVING<PRINT, ,>           /* PROCESS OPTIONS              */
3040      GO TO (LOOP1,ERR,ERR7), RTRN           /* CHECK RETURN CODE             /
3050    END                                      /* END PRINT FILE PROCESSING   */
3060    GO TO ERR                                /* IF NOT O,L,CR,OR ' '-ERRO* */
3070  END                                        /* END WHILE                    */
3080  GO TO CONT2                                /* IF DROPPED THROUGH WHILE-OK*/
3090  ERR: BUFFER='*** INVALID OPTIONS ***'      /* SET MESSAGE FOR OPTIONS NOT*/
3100    CALL DSPLY<, , ADDR(BUFFER)>             /* UNDERSTOOD AND DISPLAY       */
3110    INDEX1=0                                 /* SET MSG NUMBER FOR MDERR     */
3120  ERR7: IF OBJECT NE 1 THEN GO TO ERR5       /* IF OBJECT FILE NOT OPENED    */
3130  $   LDX #IOCB+74                           /* POINT TO OBJECT IOCB         */
3140  $   SCALL .CLOSE                           /* AND CLOSE FILE               */
3150  $   LDX #IOCB+74                           /* POINT TO OBJECT IOCB         */
3160  $   SCALL .RELES                           /* RELEASE FILE                 */
3170  ERR5: IF PRINT NE 1 THEN GO TO ERR3        /* IF NO PRINT FILE - BYPASS    */
3180  $   LDX #IOCB+37                           /* ELSE : POINT TO PRINT IOCB */
3190  $   SCALL .CLOSE                           /* CLOSE PRINT FILE             */
3200  $   LDX #IOCB+37                           /* POINT TO PRINT IOCB          */
3210  $   SCALL .RELES                           /* RELEASE FILE                 */
3220  ERR3:                                      /* ENTRY TO CLOSE INPUT FILE    */
3230  $   LDX #IOCB                              /* POINT TO INPUT FILE IOCB     */
3240  $   SCALL .CLOSE                           /* AND CLOSE FILE               */
3250  ERR2:                                      /* ENTRY POINT TO RELES INPUT */
3260  $   LDX #IOCB                              /* POINT TO INPUT FILE IOCB     */
3270  $   SCALL .RELES                           /* AND RELEASE                  */
3280  ERR1: STATUS(1)=INDEX1                     /* LOAD ERROR CODE              */
3290    IF STATUS(1) EQ 0 THEN CALL MDOS         /* IF NO MDERR ERROR - STOP     */
3300  $   CLRB                                   /* CLEAR ACCB FOR MDERR         */
3310  $   LDX #IOCB                              /* POINT TO RETURN CODE         */
3320  $   SCALL .MDERR                           /* DISPLAY ERROR MESSAGE        */
3330    CALL MDOS                                /* AND STOP PROCESSING          */
3340  CONT2:                                     /* CONTINUATION POINT           */
3350  $ LDX #SCREEN                              /* LOAD X WITH START OF MESSAG*/
3360  $ SCALL .DSPLZ                             /* AND DISPLAY ON CONSOLE       */
3370    SEG1=$40                                 /* SET DEFAULT EPROM VALUES     */
3380    SEG2=$00                                 /*                              */
3390    SEG3=$EF                                 /*                              */
3400    DO INDEX1=1 TO 192                       /*                              */
3410      EPROM(INDEX1)=BUFFO                    /*                              */
3420    END                                      /*                              */
3430    CALL TOPPG                               /* PRINT TOP PAGE HEADING       */
3440    IF RTRN NE 0 THEN GO TO ERR7             /* IF I/O PROBLEM-MOPUP&END     */
3450  LOOP2:                                     /*                              */
3460    SEG1=$40                                 /* SET DEFAULT OBJECT CODE      */
3470    SEG2=$00                                 /*                              */
3480    SEG3=$EF                                 /*                              */
3490    ENDYET=0                                 /* NOT END OF RECORD YET        */
3500    STATE=1                                  /* SET STARTING STATE           */
3510    RTRNT=1                                  /* CLEAR RETURN CODE            */
3520    PARSE1=ADDR(TOP)                         /* POINT TO TOP OF BIT STRUC    */
3530    PARSE1->NUM=SEG1                         /* SAVE DEFAULT IN BIT STRUCT   */
3540    PARSE1=PARSE1+1                          /* INCREMENT POINTER            */
3550    PARSE1->NUM=SEG2                         /* SAVE DEFAULT IN BIT STRUCT   */
3560    PARSE1=PARSE1+1                          /* INCREMENT POINTER            */
3570    PARSE1->NUM=SEG3                         /* SAVE DEFAULT IN BIT STRUCT   */
3580    DAROUT(2)=DAROUT(1)
3590    DAROUT(1)=0
3600    CONSUB(2)=CONSUB(1)
3610    CONSUB(1)=0
3620    CVTX(2)=CVTX(1)
3630    CVTX(1)=0
3640    ANDX=0
3650    LIMX=0
3660  $ LDX #IOCB                                /* POINT IOCB TO INPUT FILE     */
3670  $ SCALL .GETRC                             /* AND GET NEXT INPUT RECORD    */
3680    ABSX=0
3690    SUBX=0
3700    IF STATUS(1) EQ 0 THEN GO TO CONT3       /* IF NO PROBLEM - CONTINUE     */
3710    IF STATUS(1) EQ 9 THEN GO TO EOF         /* IF END OF FILE DETECTED-     */
3720    INDEX1=STATUS(1)                         /* IF NEITHER- OTHER ERROR      */
3730    BUFFER='ERROR IN READING INPUT FILE'     /* DISPLAY MESSAGE              */
3740    CALL DSPLY<, , ADDR(BUFFER)>             /*                              */
3750    GO TO ERR7                               /* AND GO TO MOP UP ROUTINE     */
3760  CONT3:                                     /* CONTINUE                     */
3770    PARSE1=ADDR(BUFFI)                       /* POINT TO START OF RECORD     */
3780    ININST=BUFFI                             /* MOVE SOURCE STATEMENT FOR L*/
3790    IF PARSE1->ONE EQ '*' THEN GO TO COMMND
3800    NUM=PARSE1->NUM                          /* SAVE CHAR TO CHECK FOR LINE*/
3810    IF NUM GE 48 AND NUM LE 57 THEN DO       /* IF FIRST CHAR A NUMERIC      */
3820  LOOP9:  PARSE1=PARSE1+1                    /* CHECK NEXT CHARACTER         */
3830      ONE=PARSE1->ONE                        /* GET NEXT CHARACTER           */
3840      IF ONE EQ ' ' THEN GO TO CONT5         /* IF SPACE, END OF LINE NUMBE*/
3850      IF ONE EQ BUFEND THEN GO TO EOR        /* IF CR FOUND, END OF RECORD */
3860      GO TO LOOP9                            /* IF NEITHER, CONTINUE TO LO */
3870    END                                      /* END OF LINE NUMBER CHECK     */
3880    GO TO LOOP3                              /* IF NO LINE NUMBERS CONTINUE*/
3890  CONT5:  PARSE1=PARSE1+1                    /* POINT TO FIRST CHAR AFTER L*/
3900  LOOP3:  CALL PARSE                         /* PARSE OUT NEXT TOKEN         */
3910  GO TO (EOR,CALPRO,CALPRO,CALPRO,EORF,CALPRO,EORF),RTRNT
3920  CALPRO:  CALL PROCES                       /* PROCESS TOKEN                */
3930    IF STATE EQ 8 THEN GO TO EORF            /* LOGIC ERROR, SKIP REMAINDER*/
3940    IF ENDYET EQ 1 THEN GO TO EOR            /* IF CR AT END, END OF RECORD*/
3950    IF ENDYET EQ 2 THEN GO TO EORF           /* SKIP REST OF RECORD          */
```

```
3960     GO TO LOOP3                                  /* GET NEXT TOKEN              */
3970  COMMND: IF PARSE1->ONE EQ BUFEND THEN GO TO EORC /* SCAN TO END OF CM*/
3980     PARSE1=PARSE1+1                              /* MOVE POINTER TO NEXT CHAR   */
3990     GO TO COMMND                                 /* REPEAT                      */
4000  EORC: LOC=' '                                   /* SET COMMAND PORTION TO BLAN*/
4010     INSTL=' '                                    /*                             */
4020     RTRNT=0                                      /*                             */
4030     GO TO CONT4                                  /* MOVE TO PRINT TO OUTPUT     */
4040  EORF: IF PARSE1->ONE EQ BUFEND THEN GO TO EOR
4050     PARSE1=PARSE1+1                              /* SKIP OVER COMMNT AT END OF */
4060     GO TO EORF                                   /* RECORD- THEN END OF RECORD */
4070  EOR:                                            /* END OF RECORD               */
4080     PARSE2=ADDR(TOP)                             /* GET ADDRESS OF BIT STRUCT   */
4090     NUM=PARSE2->NUM                              /* GET MS EIGHT BITS OF STRUCT*/
4100  $  LDA A NUM                                    /* HEX-ASCII OF MS NIBBLE      */
4110  $  JSR CNVERM                                   /*                             */
4120  $  STA A CHAR6                                  /* STORE RESULT                */
4130  $  LDA A NUM                                    /* HEX-ASCII OF LS NIBBLE      */
4140  $  JSR CNVERL                                   /*                             */
4150  $  STA A CHAR5                                  /* STORE RESULT                */
4160     PARSE2=PARSE2+1                              /* POINT TO NEXT HEX IN BIT    */
4170     NUM=PARSE2->NUM                              /* GET NEXT EIGHT BITS OF STRI*/
4180  $  LDA A NUM                                    /* HEX-ASCII OF MS NIBBLE      */
4190  $  JSR CNVERM                                   /*                             */
4200  $  STA A CHAR4                                  /* STORE RESULT                */
4210  $  LDA A NUM                                    /* HEX-ASCII OF LS NIBBLE      */
4220  $  JSR CNVERL                                   /*                             */
4230  $  STA A CHAR3                                  /* STORE RESULT                */
4240     PARSE2=PARSE2+1                              /* GET NEXT EIGHT BITS OF STRI*/
4250     NUM=PARSE2->NUM                              /*                             */
4260  $  LDA A NUM                                    /* HEX-ASCII OF MS NIBBLE      */
4270  $  JSR CNVERM                                   /*                             */
4280  $  STA A CHAR2                                  /* STORE RESULT                */
4290  $  LDA A NUM                                    /* HEX-ASCII OF LS NIBBLE      */
4300  $  JSR CNVERL                                   /*                             */
4310  $  STA A CHAR1                                  /* STORE RESULT                */
4320     INSTL=BREAK                                  /* WRITE HEX-ASCII OBJECT      */
4330     LOC=INST                                     /* INSTRUCTION COUNT TO OUT    */
4340     INST=INST+1                                  /* INCREMENT INSTRUCTION PTR   */
4350     IF INST EQ 193 THEN DO
4360        BUFFER='ASSEMBLY ABORTED'
4370        CALL DSPLY(, , ADDR(BUFFER))
4380        BUFFER='MORE THAN 192 INSTRUCTIONS'
4390        CALL DSPLY(, , ADDR(BUFFER))
4400        GO TO EOF
4410     END
4420     PARSE2=ADDR(TOP)                             /* POINT TO TOP OF BIT STRUCT */
4430     BUFFO=PARSE2->BUFFO                          /* LOAD OVER BIT STRING        */
4440     EPROMP->BUFFO=BUFFO                          /* COPY 3 BYTES INTO EPROM     */
4450     EPROMP=EPROMP+3                              /* INCREMENT POINTER FOR NEXT */
4460  CONT4: LINE=LINE+1                              /* INCREMENT LINE COUNTER      */
4470     ININST=BUFFI                                 /* MOVE INPUTTED INSTRUCTION  */
4480     IF PRINT EQ 1 THEN DO                        /* IF FILE LISTING REQUESTED  */
4490        BUFFL=OUT                                 /* MOVE OUT TO RECORD BUFFER  */
4500        BUFFE(2)=BUFFS(2)+(PARSE1-ADDR(BUFFI))+11       /* SET END POINT */
4510        CALL WRITE(1,,)                           /* WRITE LINE                  */
4520        IF RTRN NE 0 THEN GO TO ERR7              /* IF I/O PROBLEM-MOPUP&END    */
4530     END                                          /*                             */
4540     IF PRINT EQ 2 THEN CALL DSPLY(, , ADDR(OUT))
4550  PAGER:                                          /* IF END OF PAGE              */
4560     IF LINE GT PAGEL THEN DO                     /*                             */
4570        CALL TOPPG                                /* WRITE PAGE HEADER           */
4580        IF RTRN NE 0 THEN GO TO ERR7              /* IF I/O ERROR-MOPUP&END      */
4590     END                                          /* END OF PAGE BREAK           */
4600     IF RTRNT GE 7 THEN DO                        /* IF ERROR IN ASSEMBLY        */
4610        LINE=LINE+2                               /* INCREMENT LINE COUNTER      */
4620        RTRNT=0                                   /* CLEAR STATE                 */
4630        IF PRINT EQ 1 THEN DO                     /* IF FILE LISTING             */
4640           BUFFL=MESSGE                           /* MOVE MESSAGE TO RECORD BUFF*/
4650           BUFFE(2)=BUFFS(2)+41                   /* SET END POINTER             */
4660           CALL WRITE(1, ,)                       /* WRITE RECORD                */
4670           IF RTRN NE 0 THEN GO TO ERR7           /* IF I/O PROBLEM-MOPUP&END    */
4680        END                                       /*                             */
4690        IF PRINT EQ 2 THEN CALL DSPLY(, , ADDR(MESSGE))
4700        IF LINE GT PAGEL THEN GO TO PAGER         /* IF PAGE BREAK REQUIRED-GO  */
4710     END                                          /* END OF RTRNT GT 8 IF        */
4720     GO TO LOOP2                                  /* PICK UP NEXT INPUT RECORD  */
4730  EOF:                                            /* ALL RECORDS OF INPUT DONE  */
4740     NUMPRT=ERROR                                 /* SAVE TO CHAR NUMBER OF ERRS*/
4750     CALL DSPLY(, , ADDR(TOTALS))                 /* DISPLAY NUMBER OF ERRS      */
4760     IF PRINT EQ 1 THEN DO                        /* IF FILE LISTING             */
4770        BUFFL=TOTALS                              /* MOVE ERRORS MESS TO BUFFL  */
4780        BUFFE(2)=BUFFS(2)+18                      /* SET END OF MESSAGE POINTER */
4790        CALL WRITE(1,,)                           /* WRITE MESSAGE TO FILE       */
4800        IF RTRN NE 0 THEN GO TO ERR7              /* IF PROBLEMS, MOPUP&END      */
4810     PARSE1=ADDR(RAMNAM)
4820     BUFFE(2)=BUFFS(2)+81
4830     DO INDEX1=1 TO 4
4840        BUFFL=PARSE1->BUFFER
4850        CALL WRITE(1,,)
4860        IF RTRN NE 0 THEN GO TO ERR7
4870        PARSE1=PARSE1+80
4880     END
4890  $     LDX #IOCB+37                              /* POINT TO LISTING IOCB       */
4900  $     SCALL .CLOSE                              /* CLOSE LISTING FILE          */
4910        IF STATUS(2) NE 0 THEN DO                 /* IF I/O PROBLEM-MOPUP&END    */
4920           INDEX1=STATUS(2)                       /* SAVE ERROR CODE             */
4930           PRINT=0                                /* DON'T TRY CLOSING AGAIN     */
4940           GO TO ERR7                             /* AND GO TO MOPUP             */
4950        END                                       /*                             */
```

```
4960 $    LDX #IOCB+37                          /* POINT TO LISTING IOCB      */
4970 $    SCALL .RELES                          /* RELEASE FILE               */
4980      IF STATUS(2) NE 0 THEN DO             /* IF I/O PROBLEM-MOPUP&END   */
4990        INDEX1=STATUS(2)                    /* SAVE ERROR CODE            */
5000        PRINT=0                             /* DON'T TRY RELEASING AGAIN  */
5010        GO TO ERR7                          /* GO TO MOP UP               */
5020      END                                   /*                            */
5030    END                                     /*                            */
5040    IF OBJECT EQ 1 THEN DO                  /* IF FILE OBJECT             */
5050      DO INDEX1=1 TO 8                      /* WRITE OUT EPROM            */
5060        CALL WRITE(0, ,)                    /* WRITE OUT RECORD           */
5070        IF RTRN NE 0 THEN GO TO ERR7        /* IF I/O PROBLEM-MOPUP&END   */
5080        BUFFS(3)=BUFFS(3)+72                /* ADJUST POINTERS FOR NEXT   */
5090        BUFFE(3)=BUFFS(3)+71                /* RECORD                     */
5100      END                                   /* CONTINUE DO                */
5110 $    LDX #IOCB+74                          /* POINT TO OBJECT IOCB       */
5120 $    SCALL .CLOSE                          /* CLOSE FILE                 */
5130      IF STATUS(3) NE 0 THEN DO             /* IF I/O PROBLEM-MOPUP&END   */
5140        INDEX1=STATUS(3)                    /* SAVE ERROR CODE            */
5150        OBJECT=0                            /* DON'T TRY TO CLOSE AGAIN   */
5160        GO TO ERR7                          /* GO TO MOPUP                */
5170      END                                   /*                            */
5180 $    LDX #IOCB+74                          /* POINT TO OBJECT IOCB       */
5190 $    SCALL .RELES                          /* RELEASE FILE               */
5200      IF STATUS(3) NE 0 THEN DO             /* IF PROBLEMS-MOPUP & END    */
5210        INDEX1=STATUS(3)                    /* SAVE ERROR CODE            */
5220        OBJECT=0                            /* DON'T TRY RELEASING AGAIN  */
5230        GO TO ERR7                          /* GO TO MOP UP               */
5240      END                                   /*                            */
5250    END                                     /*                            */
5260 $ LDX #IOCB                                /* POINT TO INPUT IOCB        */
5270 $ SCALL .CLOSE                             /* CLOSE INPUT FILE           */
5280    IF STATUS(1) NE 0 THEN DO               /* IF I/O PROBLEM-MOPUP&END   */
5290      INDEX1=STATUS(1)                      /* SAVE ERROR CODE            */
5300      GO TO ERR1                            /* GO TO MOP UP               */
5310    END                                     /*                            */
5320 $ LDX #IOCB                                /* POINT TO INPUT IOCB        */
5330 $ SCALL .RELES                             /* AND RELEASE INPUT FILE     */
5340    IF STATUS(1) NE 0 THEN DO               /* IF I/O PROBLEM-MOPUP&END   */
5350      INDEX1=STATUS(1)                      /* SAVE ERROR CODE            */
5360      GO TO ERR1                            /* AND GO TO MOP UP           */
5370    END                                     /*                            */
5380    CALL MDOS                               /* PASS CONTROL BACK TO MDOS  */
5390 END                                        /* END OF MAIN PROC           */
5400 $CNVERM AND A #0F0H                        /* HEX-ASCII OF MS NIBBLE     */
5410 $       LSR A                              /* SHIFT MS NIBBLE TO LS NIBBL*/
5420 $       LSR A                              /*                            */
5430 $       LSR A                              /*                            */
5440 $       LSR A                              /*                            */
5450 $       ADD A #90H                         /*                            */
5460 $       DAA                                /*                            */
5470 $       ADC A #40H                         /*                            */
5480 $       DAA                                /*                            */
5490 $       RTS                                /* RETURN TO MAINLINE         */
5500 $CNVERL AND A #0FH                         /*HEX-ASCII OF LS NIBBLE      */
5510 $       ADD A #90H                         /*                            */
5520 $       DAA                                /*                            */
5530 $       ADC A #40H                         /*                            */
5540 $       DAA                                /*                            */
5550 $       RTS                                /* RETURN TO MAINLINE         */
5560 PARSE:  PROC                               /* TOKEN PARSER ROUTINE       */
5570 LEAD1:                                     /* REMOVE LEADING SPACES      */
5580    IF PARSE1->ONE  EQ ' ' THEN DO          /* IF A BLANK-IGNORE          */
5590      PARSE1=PARSE1+1                       /* INCREMENT POINTER          */
5600      GO TO LEAD1                           /* AND REPEAT SCAN            */
5610    END                                     /*                            */
5620    TOKENA=' '                              /* ASSIGN CURRENT TOKEN BLANK */
5630    INDEX3=0                                /* ZERO CURRENT CHARACTER COUN*/
5640 SCAN:  ONE=PARSE1->ONE                     /* SINGLE OUT CURRENT CHARCTER*/
5650    IF ONE EQ BUFEND THEN DO                /* IF A CR                    */
5660      IF INDEX3 EQ 0 THEN DO                /* AND TOKEN LENGTH EQ 0      */
5670        RTRNT=1                             /* SET RETURN CODE,CR NO TOKEN*/
5680        RETURN                              /* AND RETURN TO MAIN PROGRAM */
5690      END                                   /*                            */
5700      RTRNT=2                               /* OTHERWISE, SET RETURN CODE */
5710      ENDYET=1                              /* DON'T SEARCH RECORD ANY MOR*/
5720      GO TO SEARCH                          /* AND SEARCH FOR OPCODE      */
5730    END                                     /*                            */
5740    IF ONE EQ ';' THEN DO                   /* IF A COMMENT DELIMITER     */
5750      IF INDEX3 EQ 0 THEN DO                /* AND ZERO LENGTH TOKEN      */
5760        RTRNT=5                             /* SET RTRNT CODE             */
5770        RETURN                              /* AND RETURN FOR NEXT RECORD */
5780      END                                   /*                            */
5790      RTRNT=4                               /* OTHERWISE, SET RETURN CODE */
5800      ENDYET=2                              /* PASS OVER REMAINDER OF RECO*/
5810      GO TO SEARCH                          /* AND LOOK FOR OPCODE        */
5820    END                                     /*                            */
5830    IF ONE EQ ':' THEN DO                   /* IF LABEL DELIMITER         */
5840      RTRNT=3                               /* NORMALLY, IGNORE LABEL     */
5850      PARSE1=PARSE1+1                       /* POINT TO NEXT CHAR FOR PARS*/
5860      IF INDEX3 EQ 0 THEN DO                /* IF ZERO LENGTH,SYNTAX ERROR*/
5870        RTRNT=7                             /* ZERO LENGTH SYNTAX ERROR   */
5880        ERROR=ERROR+1                       /* INCREMENT ERROR COUNTER    */
5890        P0=MESS2                            /* SET MESSAGE                */
5900      END                                   /*                            */
5910      RETURN                                /* RETURN TO MAINLINE,NEXT TOK*/
5920    END                                     /*                            */
5930    IF ONE EQ ',' OR ONE EQ ' '             /* IF NORMAL DELIMITER        */
5940      THEN DO                               /* THEN CHECK FOR ZERO LENGTH */
5950        IF INDEX3 EQ 0                      /* IF SO, SYNTAX ERROR        */
```

```
5960            THEN DO                                 /* SET RETURN CODE AND RETURN */
5970              RTRNT=7                               /*                            */
5980              ERROR=ERROR+1                         /* INCREMENT ERROR COUNT      */
5990              P0=MESS2                              /* SET MESSAGE                */
6000              RETURN                                /* RETURN FOR NEXT TOKEN      */
6010            END                                     /*                            */
6020          RTRNT=6                                   /* ELSE, NORMAL RETURN CODE   */
6030          PARSE1=PARSE1+1                           /* POINT TO NEXT CHAR FOR PARS*/
6040          GO TO SEARCH                              /* AND SEARCH FOR OPCODE      */
6050        END                                         /*                            */
6060      INDEX3=INDEX3+1                               /* IF NONE OF ABOVE, VALID CHA*/
6070      IF INDEX3 GT 8 THEN DO                        /* IF TOO MANY CHAR IN TOKEN  */
6080        RTRNT=7                                     /* SET RETURN CODE            */
6090        ERROR=ERROR+1                               /* INCREMENT ERROR COUNT      */
6100        P0=MESS1                                    /* SET MESSAGE                */
6110        RETURN                                      /*                            */
6120      END                                           /*                            */
6130      TOKEN(INDEX3)=ONE                             /* ASSIGN VALID CHAR TO TOKEN */
6140      PARSE1=PARSE1+1                               /* INCREMENT PARSE POINTER    */
6150      GO TO SCAN                                    /* AND CONTINUE SCAN          */
6160  SEARCH:
6170      IF INDEX3 LE 4 THEN DO                        /* IF POSSIBLE INSTRUCTION    */
6180        PARSE2=ADDR(TOKENA)                         /* POINT TO TOKEN DECODED     */
6190        FOUR=PARSE2->FOUR                           /* SAVE FIRST 4 IN 4 LONG STRU*/
6200        DO INDEX2=1 TO 87                           /* SCAN INSTRUCTION TABLE     */
6210          IF OPCODE(INDEX2) EQ FOUR                 /* IF TOKEN AND INSTRUCTION * */
6220            THEN DO                                 /* THEN SET FLAGS AND RETURN  */
6230              KIND=CODET(INDEX2)                    /* INDEX IN INDEX2            */
6240              RETURN                                /*                            */
6250            END                                     /* END IF                     */
6260        END                                         /* CONTINUE DO                */
6270      END                                           /* END LE 4 IF                */
6280      PARSE2=ADDR(TOKENA)
6290      KIND=0                                        /* IF NOT INSTRUCTION-VARIABLE*/
6300      NUM=PARSE2->NUM                               /* GET FIRST CHAR OF TOKEN    */
6310      IF NUM GE 65 AND NUM LE 90 THEN GO TO OK99  /* IF CHAR ALPHA              */
6320      ERROR=ERROR+1                                 /* IF FIRST NOT ALPHA,ERROR   */
6330      P0=MESS16                                     /* LOAD ERROR MESSAGE         */
6340      RTRNT=7                                       /* RELAY MESS-ERR             */
6350      RETURN                                        /* AND RETURN TO MAINLINE     */
6360  OK99:                                             /*                            */
6370      IF VAR EQ 0 THEN DO                           /* CHECK IF NO VARS USED YET  */
6380        RAMNAM(1)=TOKENA                            /* SAVE TOKEN AS VARIABLE NAME*/
6390        VAR=1                                       /* SET RAMNAM POINTER         */
6400        INDEX2=1                                    /* SET POINTER TO FIRST VAR   */
6410        RETURN                                      /* AND RETURN TO MAINLINE     */
6420      END                                           /*                            */
6430      ELSE DO                                       /* ELSE LOOK FOR NAME IN TABLE*/
6440        DO INDEX2=1 TO VAR                          /* LOOK THROUGH NAMES IN TABLE*/
6450          IF RAMNAM(INDEX2) EQ TOKENA THEN RETURN  /* FOUND NAME                 */
6460        END                                         /* INDEX TO RAMNAM IN INDEX2  */
6470        IF VAR EQ 40 THEN DO                        /* CHECK IF TABLE FULL        */
6480          ERROR=ERROR+1                             /* NEW NAME AND TABLE FULL-ERR*/
6490          P0=MESS14                                 /* INCREM ERR COUNTER&LOAD MES*/
6500          RTRNT=7                                   /* RTRN CODE TO SKIP REST&ERR */
6510          RETURN                                    /* AND GO BACK TO MAINLINE    */
6520        END                                         /* END VAR EQ 40 IF           */
6530        VAR=VAR+1                                   /* IF NO ERR, NEW NAME FOR TAB*/
6540        INDEX2=INDEX2+1                             /* POINT TO VARIABLE          */
6550        PARSE2=ADDR(RAMNAM)                         /* CALCULATE ADDRESS FOR TOKEN*/
6560        PARSE2=(PARSE2+(8*VAR))-8                   /* OFFSET IN RAMNAM           */
6570        PARSE2->TOKENA=TOKENA                       /* STORE TOKENA IN RAMNAM     */
6580        RETURN                                      /* RETURN TO MAINLINE         */
6590      END                                           /* END VAR EQ 0 ELSE DO CLAUSE*/
6600  END                                               /* END OF PARSE PROC          */
6610  WRITE: PROC(WTYPE, ,)                             /* LISTING/OBJECT FILE WRITE  */
6620    DCL WTYPE BIN(1)                                /* LISTING OR OBJECT PARM     */
6630    RTRN=0                                          /* SET NORMAL RETURN CODE     */
6640    IF WTYPE EQ 1                                   /* IF LISTING FILE WRITE      */
6650      THEN DO                                       /* THEN PUT OUT RECORD        */
6660 $       LDX #IOCB*37                               /* POINT IOCB TO LISTING FILE */
6670 $       SCALL .PUTRC                               /* WRITE RECORD TO DISKETTE   */
6680         IF STATUS(2) NE 0                          /* IF PROBLEMS WRITING        */
6690           THEN DO                                  /* THEN DISPLAY MESSAGE       */
6700             BUFFER='LISTING FILE PROBLEM'
6710             CALL DSPLY(, , ADDR(BUFFER))           /*                            */
6720             INDEX1=STATUS(2)                       /* SAVE RETURN CODE FOR MDERR */
6730             RTRN=1                                 /* SET RETURN CODE            */
6740           END                                      /*                            */
6750         RETURN                                     /* AND RETURN                 */
6760       END                                          /*                            */
6770       ELSE DO                                      /* ELSE OBJECT FILE           */
6780 $       LDX #IOCB*74                               /* POINT IOCB TO OBJECT FILE  */
6790 $       SCALL .PUTRC                               /* AND WRITE OUT RECORD       */
6800         IF STATUS(3) NE 0                          /* IF PROBLEM, DISPLAY MESSAGE*/
6810           THEN DO                                  /*                            */
6820             BUFFER='OBJECT FILE PROBLEM'           /*                            */
6830             CALL DSPLY(, , ADDR(BUFFER))           /*                            */
6840             INDEX1=STATUS(3)                       /* SAVE RETURN CODE FOR MDERR */
6850             RTRN=1                                 /* SET RETURN CODE            */
6860           END                                      /*                            */
6870         RETURN                                     /* AND RETURN                 */
6880       END                                          /*                            */
6890  END
6900  OPTION: PROC                                      /* OPTION PROCESSING ROUTINE  */
6910    RTRN=1                                          /* SET DEFAULT RETURN CODE    */
6920    TUNIT=IUNIT                                     /* SET DEFAULT UNIT NUMBER    */
6930    TNAME=INAME                                     /* SET DEFAULT FILE NAME      */
6940    PARSE1=PARSE1+1                                 /* POINT TO NEXT CHAR         */
6950    PARSE2=PARSE1+1                                 /* POINT TO CHAR AFTER        */
```

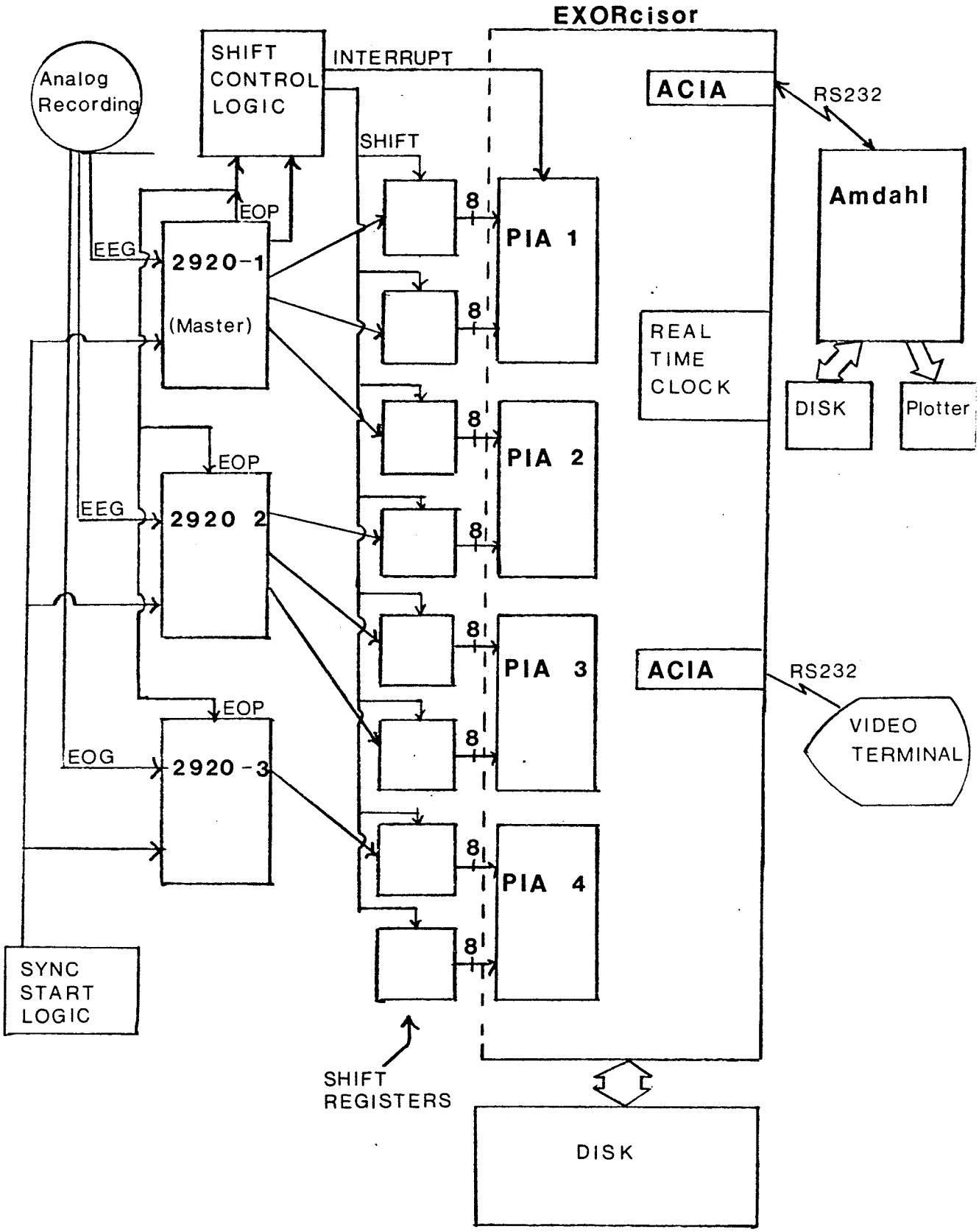
```
6960    IF PARSE1->ONE EQ '=' AND PARSE2->ONE EQ '#'
6970      THEN DO                                  /* IF CONSOLE INDICATED THEN  */
6980        PARSE1=PARSE1+5                        /* MOVE POINTER PAST =#CN AND */
6990        RETURN<2, ,>                           /* AND RETURN INDICATING CONSO*/
7000      END                                      /*                            */
7010    IF PARSE1->ONE EQ '='                      /* SPECIFIED FILE NAME        */
7020      THEN DO                                  /* PFNAM FILE NAME            */
7030        PARSE1->ONE=','                        /* SET CURRENT CHAR AS DEL    */
7040        PARSE2=ADDR(TFILE)                     /* SET PFNAM CONTROL BLOCK    */
7050 $      LDX #PARSE1                            /* POINT TO CNTL BLOCK        */
7060 $      SCALL .PFNAM                           /* PARSE NAME                 */
7070 $      STA B INDEX2                           /* STORE RETURN CODE          */
7080        IF INDEX2 LE 1                         /* IF NAME OK                 */
7090          THEN DO                              /* THEN LOAD NAME INO IOCB    */
7100            PARSE1=PARSE1+1                    /* INCREMENT CHAR POINTER     */
7110            GO TO OK0                          /* AND ATTEMPT TO RESERV&OPEN */
7120          END                                  /*                            */
7130          ELSE DO                              /* IF NAME NOT OK             */
7140            RTRN=2                             /* SET RETURN CODE            */
7150            RETURN<0, , >                      /* AND RETURN                 */
7160          END                                  /*                            */
7170      END                                      /* END = IF CLAUSE            */
7180 OK0:  FILE(INDEX3)=TFILE                       /* LOAD DECODED FILE NAME     */
7190 $ LDX #IOCB                                    /*                            */
7200 $ STX PARSE2                                   /* AND STORE FOR MPL          */
7210    PARSE2=PARSE2+OFFSET                        /* CALCLATE RELATIVE ADDRESS  */
7220 $ LDX PARSE2                                   /* POINT TO IOCB              */
7230 $ SCALL .RESRV                                 /* RESERVE FILE               */
7240    IF STATUS(INDEX3) NE 0                      /* IF AN ERROR OCCURED IN RESE*/
7250      THEN DO                                   /* THEN SET RETURN CODES      */
7260        INDEX1=STATUS(INDEX3)                   /* SAVE STATUS CODE           */
7270        RTRN=3                                  /* SET RETURN CODE            */
7280        RETURN<0, ,>                            /* AND RETURN                 */
7290      END                                       /*                            */
7300 $ LDX PARSE2                                   /* SET IOCB FOR OPEN          */
7310 $ SCALL .OPEN                                  /* OPEN FILE                  */
7320    IF STATUS(INDEX3) NE 0                      /* IF ERROR IN OPENING FILE   */
7330      THEN DO                                   /* THEN RELES IOCB AND MOP UP */
7340        INDEX1=STATUS(INDEX3)                   /* SAVE STATUS                */
7350 $      LDX PARSE2                              /* POINT TO IOCB              */
7360 $      SCALL .RELES                            /* RELEASE FILE               */
7370        RTRN=3                                  /* SET RETURN CODE            */
7380        RETURN<0, ,>                            /* AND RETURN                 */
7390      END                                       /*                            */
7400    RETURN<1, ,>                                /* IF NORMAL, RETURN OK       */
7410 END                                            /* END OF OPTION PROC         */
7420 TOPPG:  PROC                                   /* PRINT PAGE HEADER          */
7430    CPT4=PAGE                                   /* PAGE NUMBER TO HEADER      */
7440    PAGE=PAGE+1                                 /* INCREMENT PAGE COUNTER     */
7450    LINE=3                                      /* SET LINE COUNTER           */
7460    RTRN=0                                      /* CLEAR RETURN CODE          */
7470    IF PRINT EQ 1 THEN DO                       /* IF LISTING FILE REQUESTED  */
7480      BUFFL=HEADER                              /* HEADER TO RECORD BUFFER    */
7490      BUFFE(2)=BUFFS(2)+79                      /* SET END OF RECORD POINTER  */
7500      CALL WRITE<1, ,>                          /* WRITE RECORD TO LISTING FIL*/
7510      IF RTRN NE 0 THEN GO TO ENDTOP            /* IF I/O PROBLEM - MOPUP/END */
7520      BUFFL=HEAD2                               /* ELSE WRITE SECOND LINE HDR */
7530      BUFFE(2)=BUFFS(2)+29                      /* SET END OF RECORD POINTER  */
7540      CALL WRITE<1, ,>                          /* WRITE RECORD TO FILE       */
7550      IF RTRN NE 0 THEN GO TO ENDTOP            /* IF I/O PROBLEM-MOPUP & END */
7560    END                                         /*                            */
7570    IF PRINT EQ 2 THEN DO                       /* IF LISTING TO CONSOLE      */
7580      CALL DSPLY<, , ADDR(BUFFL)>               /* DISPLAY FIRST LINE         */
7590      CALL DSPLY<, , ADDR(HEAD2)>               /* DISPLAY SECOND LINE        */
7600      BUFFER=' '                                /* ADVANCE BLANK LINE         */
7610      CALL DSPLY<, , ADDR(BUFFER)>              /*                            */
7620    END                                         /*                            */
7630 ENDTOP:                                        /*                            */
7640 RETURN                                         /* RETURN TO MAINLINE         */
7650 END                                            /* END OF TOPPG PROC          */
7660 PROCES:  PROC                                  /* STATE MACHINE, LOGIC PROC  */
7670    PARSE2=ADDR(BITUP)                          /* GET POINTER TO BIT WISE BYT*/
7680    IF KIND EQ 0 THEN NUM=INDEX2-1              /* IF VAR, USE BASE 0 ADDRESS */
7690      ELSE NUM=CODE(INDEX2)                     /* ELSE GET BIT CODE          */
7700    PARSE2->NUM=NUM                             /* LOAD ADDR/BITS TO BITUP    */
7710    IF KIND EQ 6 THEN GO TO NONSUP              /* OUT FOR NONSUPPORTED CODE  */
7720    GO TO (ST1,ST2,ST3,ST4,ST5,ST6,STAERR),STATE
7730 ST1:  IF RTRNT EQ 3 THEN DO                    /* IF LABEL, NO CODE WRITTEN  */
7740         STATE=2                                /* MOVE TO STATE 2            */
7750         GO TO ENDSTA                           /* RETURN TO MAINLINE         */
7760       END                                      /*                            */
7770    IF KIND EQ 1 THEN GO TO ARITH               /* IF ARITH OP CODE, PROCESS  */
7780    IF KIND EQ 5 THEN GO TO IOCODE              /* IF I/O CODE, PROCESS       */
7790    GO TO STAERR                                /* OTHERWISE, SYNTAX ERROR    */
7800 ST2:  IF KIND EQ 1 THEN GO TO ARITH            /* IF ARITH OP CODE, PROCESS  */
7810    IF KIND EQ 5 THEN GO TO IOCODE              /* IF I/O CODE, PROCESS       */
7820    GO TO STAERR                                /* OTHERWISE, SYNTAX ERROR    */
7830 ST3:  IF KIND EQ 0 THEN GO TO VARDES           /* VARIABLE AS A DESTINATION  */
7840    IF KIND EQ 2 THEN GO TO DARDES              /* DAR AS DESTINATION         */
7850    GO TO STAERR                                /* OTHERWISE, SYNTAX ERROR    */
7860 ST4:  IF KIND EQ 0 THEN GO TO VARSOU           /* VARIABLE AS SOURCE         */
7870    IF KIND EQ 2 THEN GO TO DARSOU              /* DAR AS THE SOURCE          */
7880    IF KIND EQ 3 THEN GO TO CONXXX              /* CONSTANT AS SOURCE         */
7890    GO TO STAERR                                /* OTHERWISE, SYNTAX ERROR    */
7900 ST5:  IF KIND EQ 5 THEN GO TO IOCODE           /* I/O CODE, PROCESS          */
7910    IF KIND EQ 4 THEN GO TO SHXXX               /* SHIFT CODE, PROCESS        */
7920    GO TO STAERR                                /* OTHERWISE, SYNTAX ERROR    */
7930 ST6:  IF KIND EQ 5 THEN GO TO IOCODE           /* I/O CODE, PROCESS          */
7940 STAERR:  ERROR=ERROR+1                         /* INCREMENT ERROR COUNTER    */
7950    P0=MESS2                                    /* LOAD ERROR MESSAGE         */
```

```
7960 PNT90:  STATE=8                                  /* INDICATE ERROR TO MAINLINE */
7970    RTRNT=7                                       /*                            */
7980    RETURN                                        /* RETURN TO MAINLINE         */
7990 ARITH:  STATE=3                                  /* SET NEW STATE CODE         */
8000    IF INDEX2 EQ 2 THEN ANDX=1
8010    IF INDEX2 EQ 3 THEN LIMX=1
8020    IF INDEX2 EQ 4 THEN ABSX=1
8030    IF INDEX2 EQ 6 THEN SUBX=1
8040    L0=0
8050    L1=0
 060    L2=0
8070    IF BIT0 EQ 1 THEN L0=1
8080    IF BIT1 EQ 1 THEN L1=1
8090    IF BIT2 EQ 1 THEN L2=1
8100    GO TO ENDSTA                                  /* RETURN TO MAINLINE         */
8110 IOCODE:  STATE=7                                 /* SET NEW STATE CODE         */
8120    IF INDEX2 EQ 16 THEN GO TO PNT97
8130    IF INDEX2 GE 33 AND INDEX2 LE 40 THEN DO
8140 PNT97:  IF SUBX NE 1 THEN GO TO PNT98
8150           GO TO PNT96
8160    END
8170    GO TO PNT99
8180 PNT96:  CONSUB(1)=1
8190    IF DAROUT(1) EQ 1 THEN DO
8200      ERROR=ERROR+1
8210      P0=MESS12
8220      GO TO PNT90
8230    END
8240    GO TO PNT99
8250 PNT98:  IF ANDX EQ 1 THEN DO
8260      ERROR=ERROR+1
8270    * P0=MESS9
8280      GO TO PNT90
8290    END
8300    IF LIMX EQ 1 THEN DO
8310      ERROR=ERROR+1
8320      P0=MESS10
8330      GO TO PNT90
8340    END
8350    IF ABSX EQ 1 THEN DO
8360      ERROR=ERROR+1
8370      P0=MESS11
8380      GO TO PNT90
8390    END
8400 PNT99:  IF INDEX2 EQ 15 THEN GO TO PNT100
8410    IF INDEX2 GE 25 AND INDEX2 LE 32 THEN DO
8420 PNT100:  CVTX(1)=1
8430      IF DAROUT(2) EQ 1 THEN DO
8440        ERROR=ERROR+1
8450        P0=MESS5
8460        GO TO PNT90
8470      END
8480      IF CVTX(2) EQ 1 THEN DO
8490        ERROR=ERROR+1
8500        P0=MESS6
8510        GO TO PNT90
8520      END
8530      IF CONSUB(2) EQ 1 THEN DO
8540        ERROR=ERROR+1
8550        P0=MESS7
8560        GO TO PNT90
8570      END
8580      IF DAROUT(1) EQ 1 THEN DO
8590        ERROR=ERROR+1
8600        P0=MESS8
8610        GO TO PNT90
8620      END
8630    END
8640    IF INDEX2 GE 17 AND INDEX2 LE 24 THEN DO
8650      IF DAROUT(2) EQ 1 THEN DO
8660        ERROR=ERROR+1
8670        P0=MESS3
8680        GO TO PNT90
8690      END
8700      IF CONSUB(2) EQ 1 THEN DO
8710        ERROR=ERROR+1
8720        P0=MESS4
8730        GO TO PNT90
8740      END
8750    END
8760    ADF0=0
8770    ADF1=0                                        /*                            */
8780    ADK0=0                                        /*                            */
8790    ADK1=0                                        /*                            */
8800    ADK2=0                                        /*                            */
8810    IF BIT0 EQ 1 THEN ADF0=1
8820    IF BIT1 EQ 1 THEN ADF1=1
8830    IF BIT2 EQ 1 THEN ADK0=1
8840    IF BIT3 EQ 1 THEN ADK1=1
8850    IF BIT4 EQ 1 THEN ADK2=1
8860    GO TO ENDSTA                                  /* RETURN TO MAINLINE         */
8870 DARDES:                                          /*                            */
8880    DAROUT(1)=1                                   /* FLAG DAR AS DES IN CURRENT */
8890 VARDES:  STATE=4                                 /* SET NEW STATE CODE         */
8900    B0=0                                          /* COPY IN BIT ADDRESS        */
8910    B1=0                                          /*                            */
8920    B2=0                                          /*                            */
8930    B3=0                                          /*                            */
8940    B4=0                                          /*                            */
8950    B5=0                                          /*                            */
```

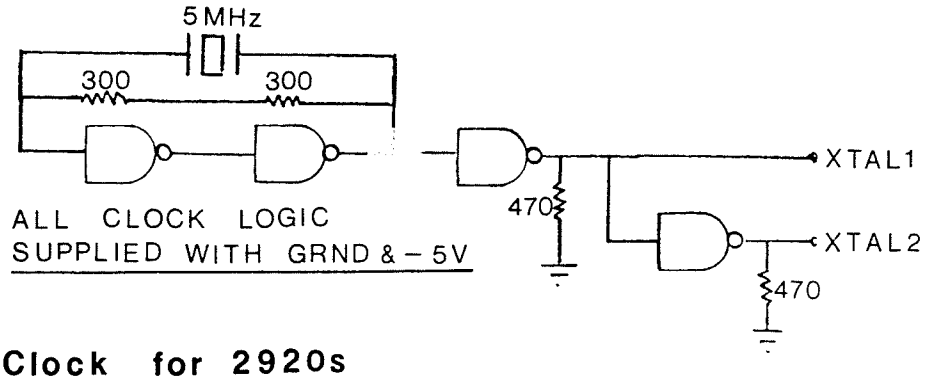
```
8960    IF BIT0 EQ 1 THEN B0=1
8970    IF BIT1 EQ 1 THEN B1=1
8980    IF BIT2 EQ 1 THEN B2=1
8990    IF BIT3 EQ 1 THEN B3=1
9000    IF BIT4 EQ 1 THEN B4=1
9010    IF BIT5 EQ 1 THEN B5=1
9020    GO TO ENDSTA                                    /* RETURN TO MAINLINE          */
9030  DARSOU:                                           /*                             */
9040  VARSOU:                                           /*                             */
9050  CONY    STATE=5                                   /* SET NEW STATE CODE          */
9060    A[illegible]                                    /* COPY IN BIT ADDRESS         */
9070    A1=0                                            /*                             */
9080    A2=0                                            /*                             */
9090    A3=0                                            /*                             */
9100    A4=0                                            /*                             */
9110    A5=0                                            /*                             */
9120    IF BIT0 EQ 1 THEN A0=1
9130    IF BIT1 EQ 1 THEN A1=1
9140    IF BIT2 EQ 1 THEN A2=1
9150    IF BIT3 EQ 1 THEN A3=1
9160    IF BIT4 EQ 1 THEN A4=1
9170    IF BIT5 EQ 1 THEN A5=1
9180    GO TO ENDSTA                                    /* RETURN TO MAINLINE          */
9190  SHXX: STATE=6                                     /* SET NEW STATE CODE          */
9200    IF LIMX EQ 1 AND CODE(INDEX2) NE $0F THEN DO
9210      ERROR=ERROR+1
9220      P0=MESS13
9230      GO TO PNT90
9240    END
9250    S0=0                                            /* COPY IN SHIFT BIT CODE      */
9260    S1=0                                            /*                             */
9270    S2=0                                            /*                             */
9280    S3=0                                            /*                             */
9290    IF BIT0 EQ 1 THEN S0=1
9300    IF BIT1 EQ 1 THEN S1=1
9310    IF BIT2 EQ 1 THEN S2=1
9320    IF BIT3 EQ 1 THEN S3=1
9330    GO TO ENDSTA                                    /*                             */
9340  NONSUP: ERROR=ERROR+1                             /* NON-SUPPORTED INSTRUCTION   */
9350    P0=MESS15                                       /*                             */
9360    STATE=8                                         /* FLAG PROBLEM TO MAINLINE    */
9370    RTRNT=7                                         /*                             */
9380    RETURN                                          /* RETURN TO MAINLINE          */
9390  ENDSTA: RETURN                                    /*                             */
9400  END
```

Appendix C

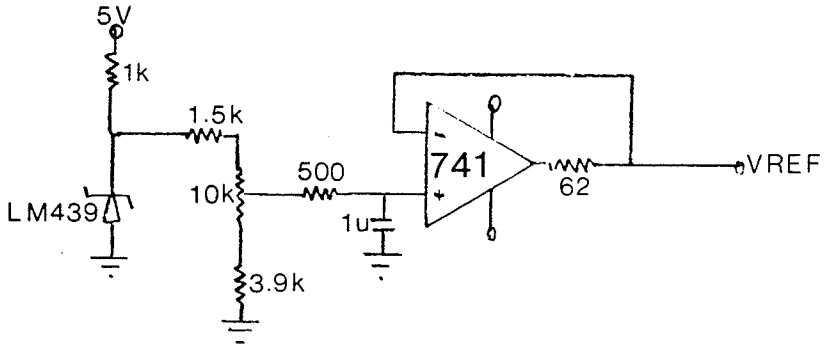
CIRCUIT DIAGRAMS



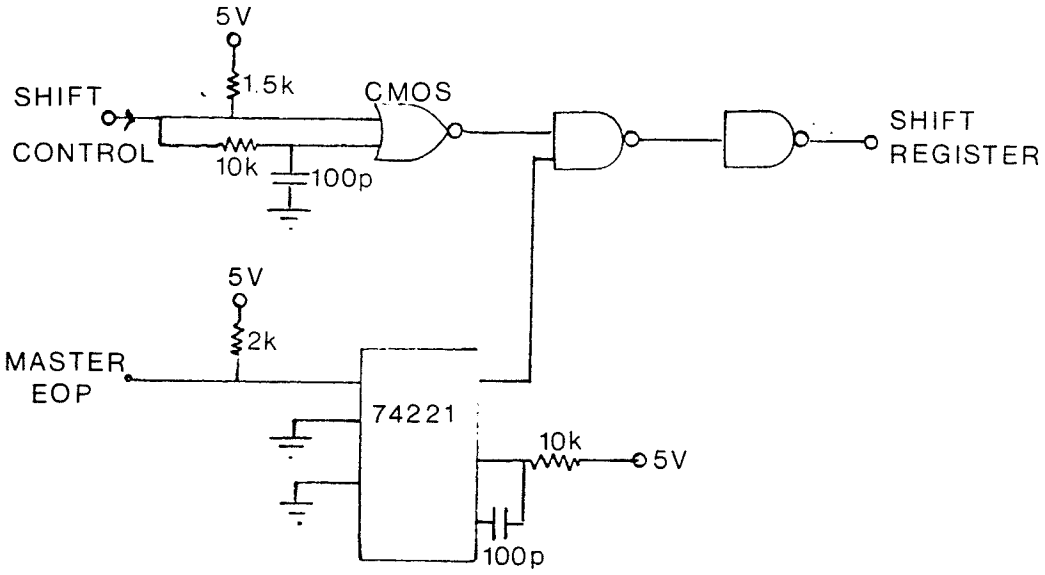
SYSTEM CONFIGURATION



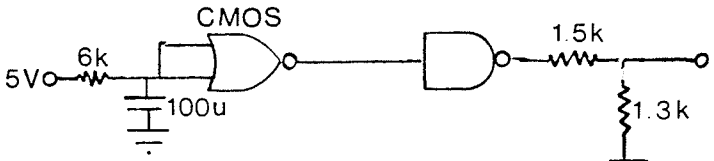
Clock for 2920s



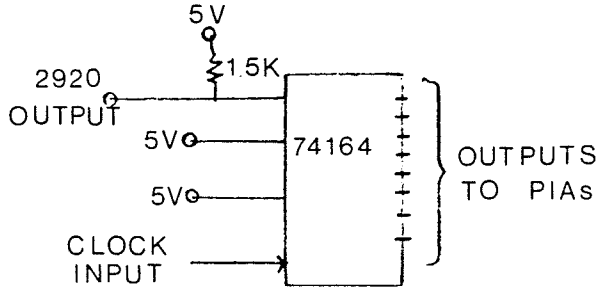
Reference Voltage



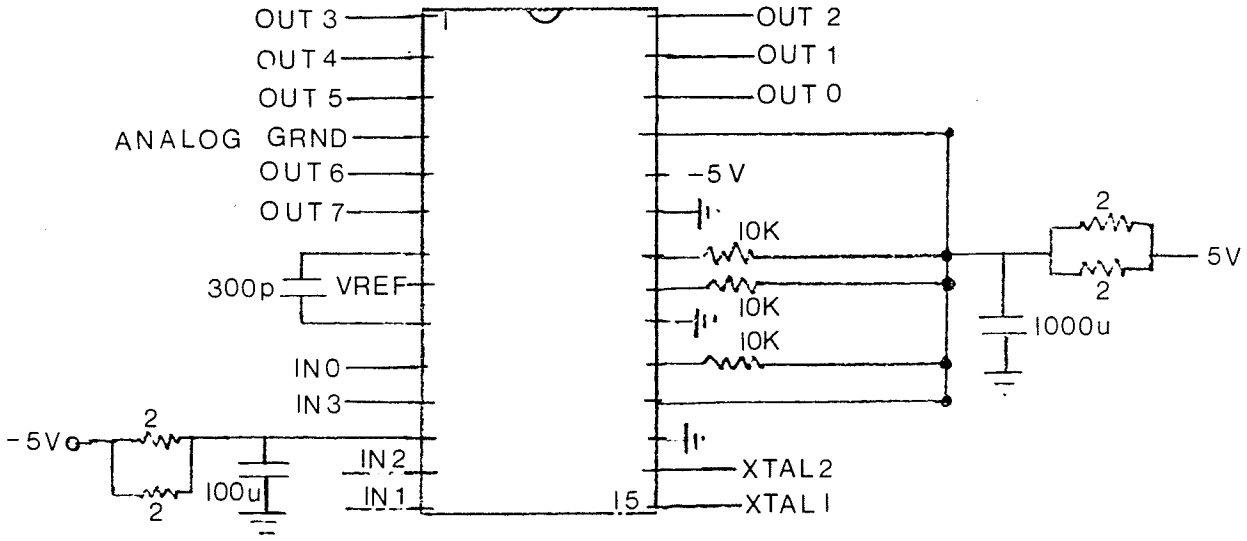
Shift Control



Synch Start



Shift Register



2920 Run Socket