A MICROCOMPUTER-BASED VOICE RECOGNITION SYSTEM

BY

THOMAS CHIATUNG LIU

B.S., University of Illinois, 1982

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1983

Urbana, Illinois

ACKNOWLEDGMENT

Many people assisted me in this thesis. Without their help, this voice recognition system would not have been debugged as swiftly and would probably not perform as well. I would like to thank my thesis advisor, Prof. Ricardo Uribe, for his advice and suggestions, and for the use of his lab, Advanced Digital Systems Laboratory. Thanks are also extended to his teaching assistant, Martin Eberhard, for demonstrating the operations of various equipment in the lab and for valuable suggestions on various aspects of this thesis. Finally, I'd especially like to thank these two students, Jim Kepler and Carl Steiner, who helped in constructing the hardware and in writing the first versions of the LISTEN program.

PREFACE

As this thesis was being completed, an article appeared in a local newspaper. It anounced the introduction of an isolated word voice recognition system, initially available for the IBM Personal Computer, that sells for $995. At this price, it is still expensive; but it is considerably less than earlier systems of comparable capabilities. Thus, the realization of a practical system may be nearer than what I had estimated.

It is interesting to note that in the past, in Advanced Digital Systems Laboratory, several projects in building a voice recognition system had been initiated. Although none of them were successful, they provided much incentive for building this one. As of now, this system is able to recognize words and, thus, represents the next step toward Prof. Uribe's eventual goal of placing such a system onto a robot.

There are many interesting aspects about this thesis. It encompasses both hardware and software. The hardware utilizes both analog and digital technologies. The software is written using both assembly language and Pascal. In addition, because of the protocol chosen for communication between the feature extractor and the MPT/100, the hardware and software portions were able to be debugged separately. At times, a standard CRT terminal was used to simulate the system hardware; at other times, it was simulating the MPT/100 software. Overall, this has been a very satisfying experience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

BACKGROUNDS

1.1 Introduction

As more and more automatic machines and robots are becoming involved in our lives, there are increasing needs for communication between people and machines. Traditionally, this has been accomplished by using such devices as keyboards and CRT displays. However, the most natural method of communication for humans is probably through speech. Thus, much research and development has been done in both speech synthesis and speech recognition, with the latter being significantly more difficult.

There are two main categories of speech recognition: discrete utterance and connected speech. Discrete utterance recognition concerns recognizing isolated words. The application here, as an example, may be for simple commands to a robot. Connected speech recognition extends the recognition to complete sentences. An example of its application may be an automatic dictation machine.

This thesis describes an implementation of a discrete utterance recognition system. There are many goals for this thesis. One is to design using standard components of the latest technology, thereby simultaneously reducing the part count and cost. A second goal is to keep the system relatively simple, for easy debugging and better reliability. A third goal is to make provisions for future expansions and improvements. A fourth, perhaps the most important goal, is to design a working system, thus proving the practicality of speech input to computers.

1.2 Problems in Voice Recognition

As with many other systems in science and engineering, there are problems to overcome in this system. These deal with the conversion of speech input into a form usable by a computer, and with the process of the computer in its attempt to recognize the input.

The solutions to the obstacles of the first category must answer many questions. One, what are the relevent components of speech signals? Two, how are these parameters to be collected by the voice recogniton system? And finally, in what forms should they be represented so the computer can process efficiently?

Similarly, many questions arise in solving problems of the second category. First, how can the computer associate a given set of input parameters to a word? Second, what are the algorithms necessary for recognizing the spoken word, since no person can speak any given word identically all the time? And finally, would the computer have problems recognizing words spoken by different persons, given that everyone has an unique voice?

All, or at least most, of these problems must be addressed and solved before a practical voice recognition system can be realized. This thesis describes this author's attempt in implementing such a system.

Chapter 2 gives an overview of this system, including discussions about decisions on the approach taken. Chapters 3 and 4 describe the hardware and software involved, respectively. Chapter 5 presents performance of the resulting system. Chapter 6 concludes with dicussions on possible improvements. The appendices include schematic diagrams, all the program listings, and the operating procedures for this system.

It should be emphasized here that, unlike many other theses, this one does not represent a final nor a best solution to a problem. Instead, this system should be considered as a starting point toward realizing an inexpensive and practical voice recognition system.

CHAPTER 2

SYSTEM OVERVIEW

This chapter presents overviews of the voice recognition system from two different viewpoints: functional and operational.

2.1 Functional Overview

Functionally, this system has two major components, as shown in Figure 1. The feature extractor detects the input voice signal. After parameterizing, its representation is passed to the host computer for processing.

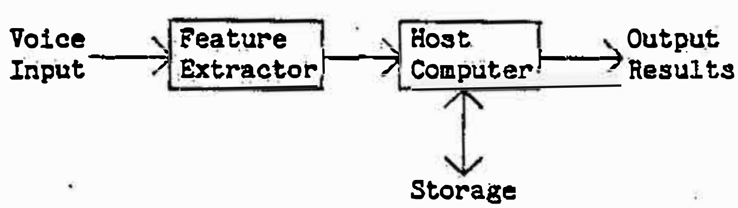


Figure 1 - System block diagram.

2.1.1 Feature extractor

The function of the feature extractor is to capture the "essential" parameters of the input voice signal. But first, some decisions must be made as to what these parameters are. There are two methods for deriving them, based on either time- or frequency-domain analysis.

Time-domain analysis is usually implemented in hardware, producing such parameters as zero-crossing density and fundamental frequency. This approach suffers from one difficulty. Since a speech signal is time-varying, any parameters representing it must vary with time. Therefore, these time-based parameters are necessarily representing short-time intervals. The difficulty arises, then, in distinguishing components of the signal between long-time and short-time. Also, past researches have shown that time-based parameters alone do not provide adequate information for successful voice recognition [12].

Frequency-domain analysis extracts parameters such as energies of spectral bands, gross spectral shape, and formant frequencies and trajectories [12]. This is accomplished by first converting the speech signal into short-time frequency spectrum. Three methods are available: bandpass filtering, Fourier transform, and linear predictive coding.

Bandpass filtering is straightforward but has many practical constraints, limiting its frequency resolving capacity. Fourier transform provides better resolution but requires large amount of computation. Linear predictive coding does not improve spectral resolution but is efficient in determining formant frequency; however, it also requires a large amount of computation.

Since one of the goals of this thesis is to keep the design simple, the bandpass filtering approach is chosen. It is also decided that the energies of each passband would provide sufficient information for voice recognition. Thus, the feature extractor has a simple job: convert the outputs from these filters into parameters proportional to the energy in each. This is accomplished by using peak detectors tracking the envelope, which relates to energy, of the signal in each band.

Since a digital computer is accepting the outputs from this device, the outputs of the peak detectors must be in digital forms. This is obtained using an analog-to-digital (A/D) converter.

For ease in development, this design makes the feature extractor, the hardware portion of the system, independent of the host computer. That is, it should be able to operate with most computers. Therefore, some standard interface must be used to transmit the collected data.

A serial port is the most appropriate medium, since most computers have such facility. This, however, introduces one problem. That is, this hardware must buffer the gathered parameters as the serial interface may not be fast enough to transmit them in real-time.

The design of the digital portion of this hardware incorporates a microprocessor-based controller. This controller has the responsibilities of controlling the data acquisition and transmission. In addition, the encoding of the parameters is performed by this controller.

As will be described in Chapter 3, the design of this controller is much simplified by the choice of the microprocessor, which is the Intel 8751 single-chip microcontroller. A block diagram of the resulting feature extractor is shown in Figure 2.

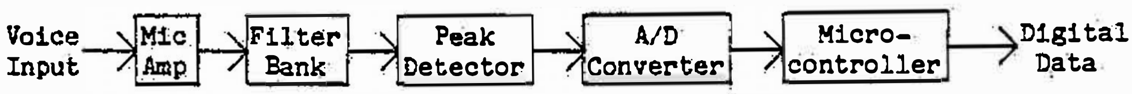


Figure 2 - Feature extractor.

2.1.2 Host computer

Because the feature-extracting hardware is designed to be compatible with most computers, the choice of the host computer for this thesis is quite arbitrary. In fact, almost any computer would be adequate, as long as it has a serial port and enough mass storage for storing the vocabulary.

The host computer chosen is a desk-top model, the Data General MPT/100. This computer has a 16-bit CPU with hardware multiply and divide. Also, it has two disk drives, each capable of storing 358 kilobytes [2].

To allow the possiblility of using a different host computer in the future, the system software should be written in an easily transportable language. This should also be a compiled language, to reduce computation time.

The software in this thesis is written in Pascal. Hopefully, only a minimum of modifications would be needed if a different host computer is used in the future.

2.2 Operational Overview

This voice recognition system is designed to respond reliably to the voice of only one person, for any one set of words. This simplifies the task of the system since it does not have to compensate for the differences in voices among different speakers. The person who is to operate this system, then, has to train the system to his/her voice. Therefore, operationally, this voice recognition system has two distinct phases, as shown in Figure 3. During the training session, the input voice pattern is associated with the text word and stored into the mass storage. During the recognition session, the input pattern is compared against the stored patterns while trying to recognize the input.

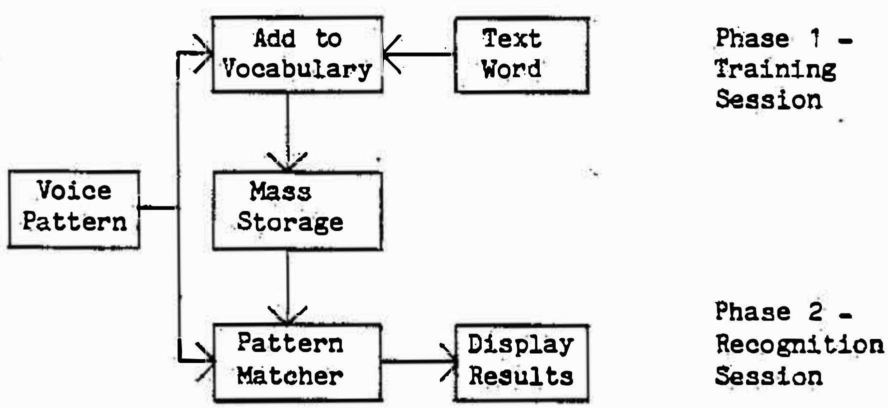


Figure 3 - Operational block diagram.

2.2.1 Training session

The purpose of the training session is to allow the system to associate a word to its parameterized voice pattern. This is under the control of two programs. One in the feature extractor, controlling the data collection, and the other in the host computer, controlling the storage of the parameters and the input of the text word.

The first program, named LISTEN, monitors the input for the possible beginning of a word. Once the start of a word is detected, LISTEN collects, encodes, and transmits the voice patterns to the host computer. When the end of the word is found, a special marker is sent to indicate such. Section 1 of Chapter 4 describes this program in detail. Note that, as far as LISTEN is concerned, there is no difference between training and recognition. That is, LISTEN is common to both sessions.

The second program, named TRAINING, builds the vocabulary by, first, prompting the user to enter each word at the keyboard. Then, it allows the user to vocalize that word, whose parameters are captured by the feature extractor and transmitted from LISTEN. This pattern, along with the word typed at the keyboard, is stored into a disk file named VOCAB, without any further processing.

The reason for storing the raw data into VOCAB is to facilitate program development. By not processing in TRAINING, the program in the recognition session can use various algorithms essentially independent of the training session. When a new algorithm is being developed, TRAINING would, hopefully, remain unchanged. Therefore, the program TRAINING is a simple one, as will be described in the second section of Chapter 4.

2.2.2 Recognition session

Similar to the training session, the recognition session is also under the control of two programs: one to collect the data and one to perform the task of recognition. As mentioned, the first program, LISTEN, is common to both sessions.

The second program, located in the host computer and named RECOGN, is the heart of this system. It is responsible for reading in the vocabulary file, allowing the user to say a test word, and attempting to recognize this input. The recognition is accomplished by comparing the test input against the vocabulary. After the voice pattern of a test word is received from LISTEN, RECOGN computes the similarity scores between this and the each of the patterns in the vocabulary. When a score meets the decision criteria, the input is recognized and the stored text word printed.

Section 3 of Chapter 4 will describe this program in full detail, including dicussions on the various algorithms for matching the unknown voice pattern to the stored patterns.

CHAPTER 3

SYSTEM HARDWARE

The hardware of this voice recognition system is composed of two major components: the analog and the controller boards. These two boards are physically located in one box and are connected by a simple bus system.

3.1 Analog Board

This board contains the components for the microphone amplifier, the bandpass filters, and the peak detectors.

3.1.1 Microphone amplifier

The microphone amplifier used in this thesis is designed for use with a low-impedance dynamic microphone; specifically, the Shure Brothers model 561. This amplifer is designed with an adjustable gain, to be calibrated such that when a fairly loud sound is picked up, the output of this amplifier is at about one volt amplitude.

This design uses two op amps from an LM324 quad op amp IC, as shown in Figure 4. The first stage is a simple microphone preamplifier. The second stage is an adjustable amplifier with a maximum gain of 200. The LM324 IC is chosen to minimize component count, since there are four amplifiers in one package, and it is quite readily available and inexpensive.

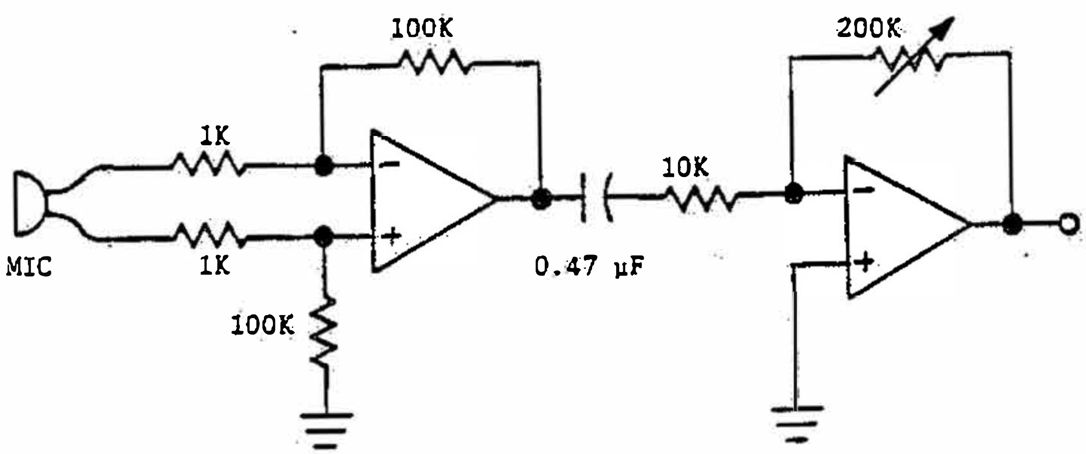


Figure 4 - Microphone amplifier.

3.1.2 Bandpass filters

Since the spectral information of the speech input will be derived using bandpass filters, several design decisions must be made. These concern the number of bands, the bandwidth of each, and the response of each filter. These factors are very much interrelated.

In order to gather reasonably accurate spectral information about the speech signal, a large number of bands should be employed. However, unless a large amount of overlap is tolerable, a large number of bands implies narrow bandwidths and steep cutoffs, with the associated requirement of precision components. On the contrary, if too few bands are used, subtleties in the frequency domain would not be detected. Therefore, some compromise must be found.

From speech research, it is known that human speech spectrum covers the range from just under 100 to about 3000 Hertz. This was confirmed experimentally during the course of this thesis. However, the speech energies are not uniformly distributed: vowels are rich in low frequencies and consonants in high frequencies. Because of this, one may be inclined to design the filters such that the more critical frequency ranges are covered more thoroughly; that is, to use filters of narrower bandwidths at strategic frequency ranges. This, however, encounters one problem. The pitch of different speakers varies. Thus, the critical frequency bands would vary from speaker to speaker. Therefore, nonuniform bandwidth may not be any better than uniform bandwidth.

Ideally, each passband should have zero transition bandwidth. This is not achievable in practice. Since it would be desirable to have a flat passband response and as sharp a transition as possible, some high-order filter would be necessary. Filters of very high order, however, are known to be difficult to design and very sensitive to component variations. Again, some sort of compromise must be achieved.

After some research into similar voice recognition systems of the past, where the number of bands varied from about five to about thirty, the decision is made at seven, somewhat arbitrarily. This number is chosen since the frequency range of 100 to 3000 covers about six octaves. Just in case there is information at either extreme of this range, the bandpass filters should cover a range slightly larger, say 60 to 4000, which is seven octaves, hence the number seven. In addition, to simplify design and since nonuniform bandwidth may not be advantageous, bandpass filters of equal widths, on a logarithmic scale, are used. The resulting filters have bands covering seven octaves, with center frequencies of 62, 125, 250, 500, 1000, 2000, and 4000 Hertz. Based on some of the data collected, there are significant energies in all seven bands.

To obtain the desired response of the filters, some prelimilary filters of various designs were built. The criterion for choosing the final design was a compromise between higher-order filter and component count. Based on these experiments, the final design used in this thesis is a fourth-order inverse Chebychev filter. This filter has the desirable features of fairly flat passband response and fairly steep cutoff. Also, the implementation of the filter is chosen to be of the infinite-gain multiple-feedback (MFB) circuit, as shown in Figure 5, because of its simplicity, good stability, and low output impedance.

Using reference [6], for Q=1 (corresponding to octave filters of one octave width), stopband rejection of 30 dB, and unity gain, the design parameters are:

$$B = 1.413164$$

$$C = 1.031123$$

For order of four, two cascaded stages are used. To design each stage using the MFB implementation, various auxiliary parameters are defined as follows:

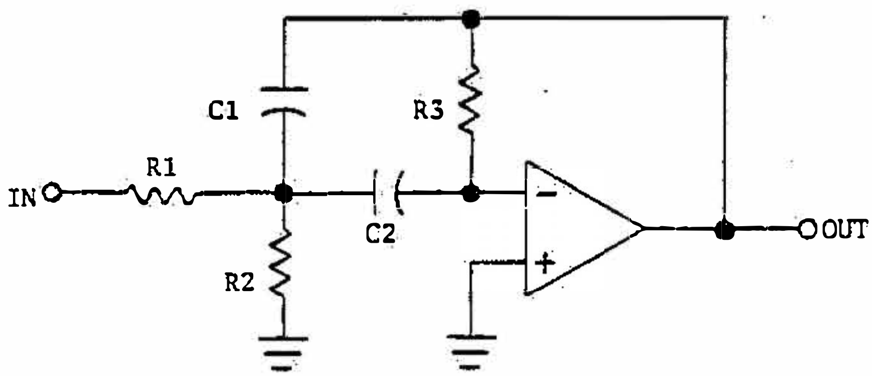


Figure 5 - MFB bandpass filter.

$$E = \frac{1}{B}\left[\frac{C+4Q^2+[(C+4Q^2)^2-(2BQ)^2]^{\frac{1}{2}}}{2}\right]^{\frac{1}{2}}$$

$$D = \frac{1}{2}\left[\frac{BE}{Q}+\left[\left(\frac{BE}{Q}\right)^2-4\right]^{\frac{1}{2}}\right]$$

$$p = \frac{C^{1/2}}{Q}$$

For the first stage,

$$b = \frac{D}{E}$$

$$r = D^2$$

For the second stage,

$$b = \frac{1}{DE}$$

$$r = \frac{1}{D^2}$$

The capacitors C1 and C2 are chosen using the criteria:

$$C1 = \frac{10}{f_o}$$

$$C2 > \frac{(pb-r)C1}{r}$$

The resistors R1, R2, and R3 are then calculated using

$$R1 = \frac{1}{pw_oC1}$$

$$R2 = \frac{b}{[(r-pb)C1+rC2]w_o}$$

$$R3 = \frac{1}{bw_o}\left[\frac{1}{C1}+\frac{1}{C2}\right]$$

where

$$w_o = 2\pi f_o$$

The calculated component values are rounded to the nearest standard values. Table 1 gives the actual values for the components used in this thesis, noting that C1's and C2's for both stages of each filter are of the same value.

TABLE 1 - COMPONENT VALUES FOR THE FILTER BANK.

		stage #1			stage #2		
Freq	C1=C2	R1	R2	R3	R1	R2	R3
62	0.1	24	7.5	51	24	24	110
125	0.1	12	3.9	27	12	12	56
250	0.1	6.2	1.8	13	6.2	6.2	30
500	0.1	.3	0.91	6.8	3	3	13
1000	0.01	16	4.7	33	16	16	68
2000	0.01	7.5	2.4	16	7.5	7.5	36
4000	0.01	3.9	1.2	8.2	3.9	3.9	18

Notes: Frequencies in Hertz.
Capacitances in microfarads.
Resistances in kilohms.

Since each stage of each filter requires one op amp and, as will be described shortly, each peak detector of each band needs two op amps, the LM324 quad op amp IC's are chosen for this thesis. By doing so, the components for each filter occupy a minimum area on the analog board.

The actual responses of the filters are plotted on Figure 6. In addition, a composite response of all seven bands is also made, to

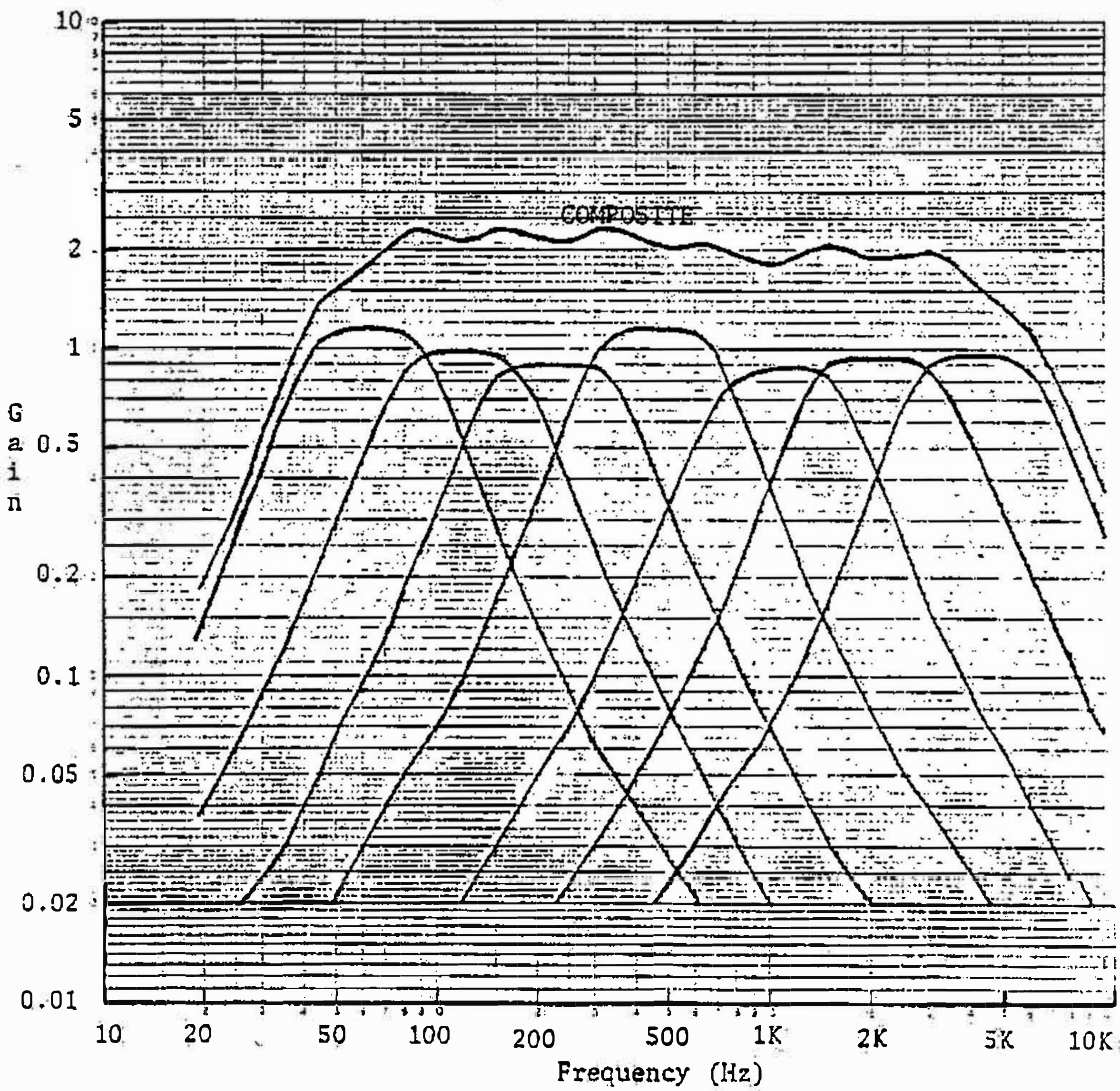


Figure 6 - Frequency response.

illustrate the overall response of the filter bank. Looking at Figure 6, it can be seen that, although the center frequencies and gains of each band are not quite at the precise designed values, they are reasonably close. Since the training and recognition phases would use the signals passing through the same filters, these irregularities should not cause problems. Also, the composite plot is fairly flat from about 60 to 4000 Hertz, indicating that this filter bank would not favor any portion of the usable speech spectrum.

3.1.3 Peak detectors

The output from each of the bandpass filters is an AC waveform with amplitude corresponding to that region of the spectrum at any instant. What the voice recognition system needs, however, is the energy level within each band. The function of the peak detectors is to convert the AC waveform into some time-varying DC level proportional to the energy level in each band.

Several considerations are relevent here. For one, since speech waveforms tend to be asymetric with respect to zero volt, as any simple experiment would confirm, a simple half-wave rectifier circuit is insufficient. Two solutions are available. One approach would be to have two peak detectors, one for positive and one for negative peaks. The other would be to use a full-wave peak detector. The design here utilizes the latter approach, though this solution may tend to lose some information concerning the speech input. However, this is deemed to be less important, as using one peak detector needs fewer components than using two.

A second consideration is that an ordinary diode-capacitor peak detector, such as that shown in Figure 7, has the disadvantage of one diode voltage drop. To overcome this, the op amp peak detector design

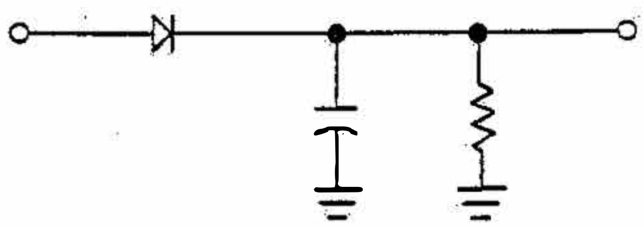


Figure 7 - Diode-capacitor peak detector.

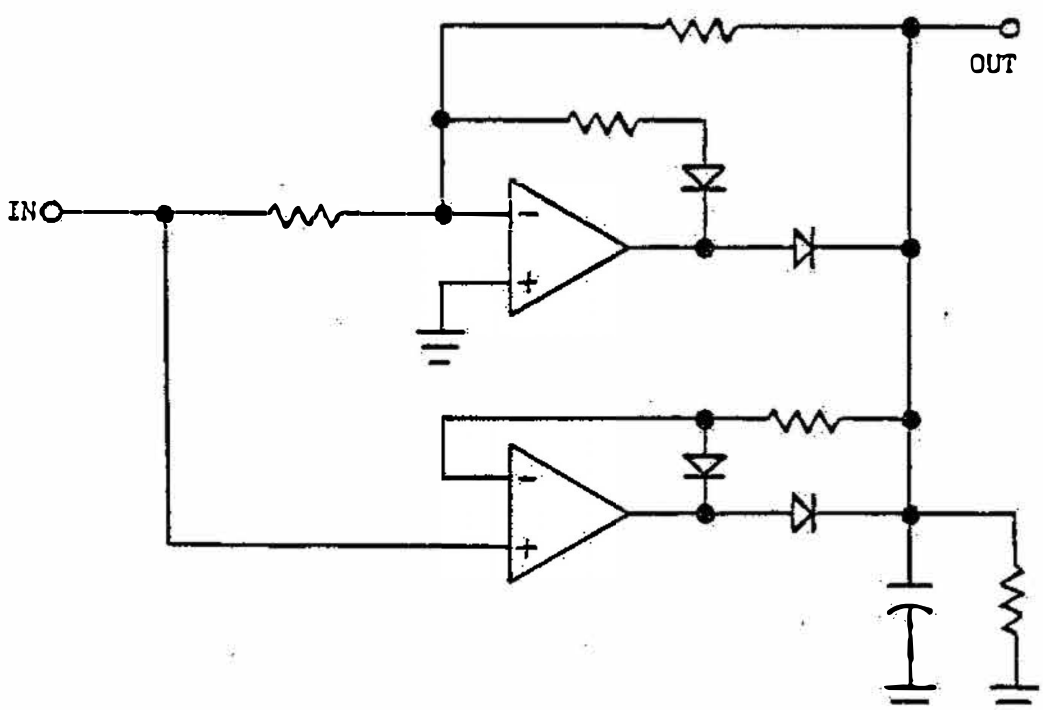
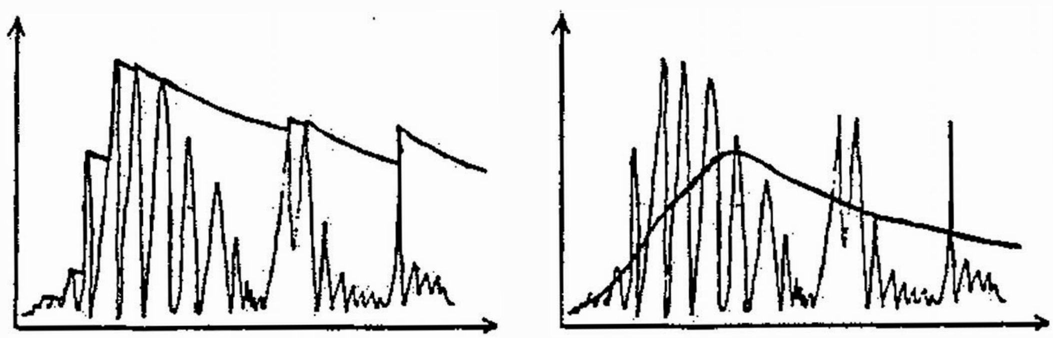


Figure 8 - Op amp peak detector.

is chosen. The design in this thesis uses that from reference [9], as shown in Figure 8, with some modifications to be described presently.

The third concern is that the circuit as it is in Figure 8 would blindly follow the peak waveform, as demonstrated in Figure 9(a). This has the unfortunate problem of tracking any noise spikes present in the input waveform. The solution is to place a series resistor in front of the capacitor to limit the rise time of the output waveform, with the resulting waveforms as in Figure 9(b).



(a) (b)

Figure 9 - Peak detection.

The final design of the peak detectors is shown in Figure 10. The resistors R1 and R2 are chosen such that the rise and fall times of the output would be fast enough to track any speech signal and yet slow enough such as not to track noise and not to track the AC waveform itself. The optimum values are determined experimentally.

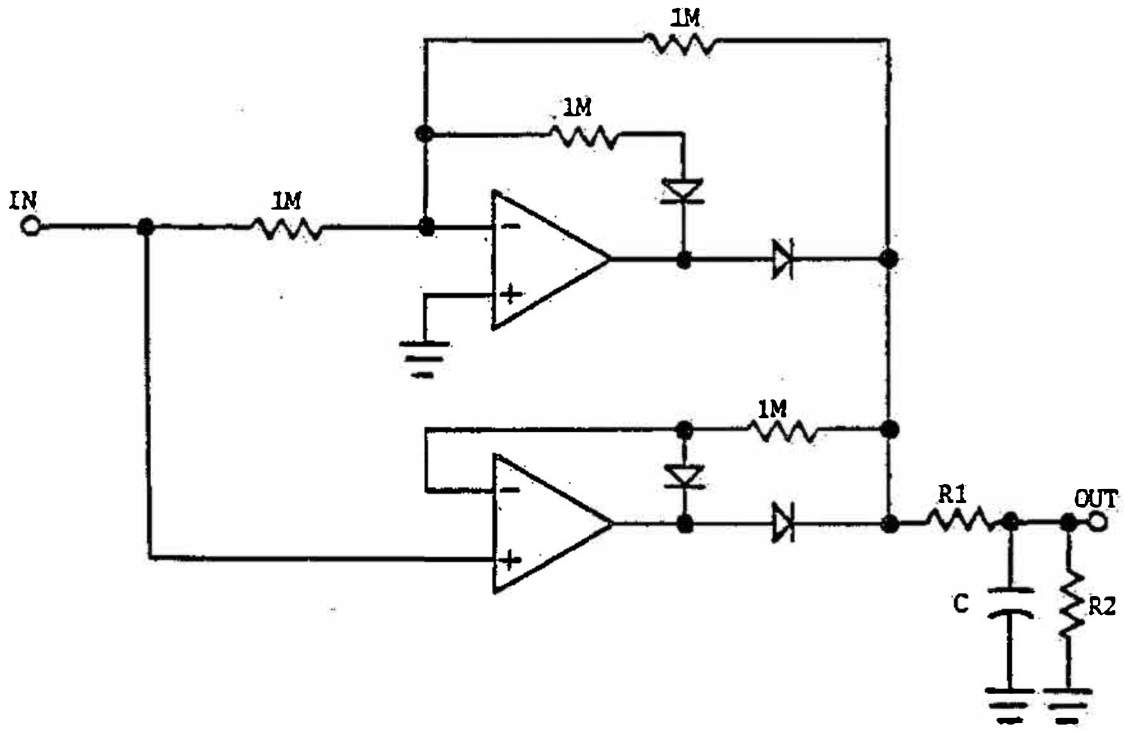


Figure 10 - Peak detector.

3.2 Controller Board

The function of the controller board is to convert the amplitudes at each band into digital form. The digitized information is then encoded and sent to the host computer for further processing. To accomplish this task, this board utilizes a microcontroller, an A/D converter, and a logarithmic amplifier. In addition, a clock circuit is responsible for generating all the clock signals required on this board.

3.2.1 Microcontroller

The microcontroller is designed around an Intel 8751 microprocessor chip. This IC contains 4 kilobytes of on-chip EPROM, 128 bytes of on-chip RAM, 2 programmable timers, one serial I/O port, and up to 32 I/O port pins. The 8751 is designed for controller applications with many bit-oriented instructions to manipulate the port pins [4].

In this thesis, some of the I/O pins are used to control the A/D converter while some others control the serial port for communication with the host computer. Due to the flexibility of this IC, the I/O interfacing is very simple, as will be described later.

Since the controller has to buffer the digitized input before sending it to the host computer, and since the 128 bytes of on-chip RAM is not sufficient to serve this purpose, additional RAM IC's are employed in this design. It is decided, somewhat arbitrarily, that 8 kilobytes would be the maximum need in the foreseeable future. Therefore, the external RAM decoding circuit is designed to handle a maximum of 8 kilobytes.

Because the 8751 uses multiplexed address and data lines, the lower 8 bits of the address sharing the same pins as the data lines, two possibilities present themselves as candidates for the RAM IC's. One way is to utilize a latch to store the lower 8 bits of the address and use conventional RAM chips. This has the advantage of low cost but has the disadvantage of higher component count. The second method is to use RAM IC's designed especially for this type of processor. This has the advantage of lower component count and simpler circuitry, but has a higher cost.

The design here is of the second approach. Using the Intel 8185 RAM chips, each IC contains 1 kilobytes of RAM, organized as 1024 by 8 bits, with built-in latch for the multiplexed address lines [5]. Since a maximum of 8 kilobytes of RAM is desired, corresponding to a maximum of eight 8185 IC's, the decoding circuitry needs to decode the upper 3 address bits to enable the chip select circuit, which decodes the next 3 address bits to enable one of upto eight RAM chips. The remaining 10 address bits are passed on to the selected chip. To simplify circuitry, a single IC is used to perform the above function, the 74LS138. As shown in Figure 11, this decoder decodes an address range of hexadecimal 2000 to 3FFF, enabling one of the eight possible 8185 RAM IC's. Currently, 2 kilobytes of external RAM are available on this board, with an address range of hexadecimal 2000 to 27FF.

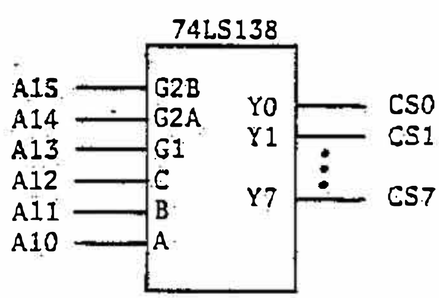


Figure 11 - Address decoder.

As mentioned before, the 8751 contains an on-chip serial port. This port has two alternatives for its clock input: from the internal clock or from an external source. It is decided that in order to leave the maximum number of port pins uncommitted, the internal clock is used, since using external clock would tie up one port pin. This decision has a major impact on the choice of clock frequency for the 8751, as will be described later.

It is also decided that the serial port is to transmit at as high a bit rate as the host computer can handle. This is determined to be 9600 bps (bits per second) for the Data General MPT/100. (Although the MPT/100 has a setting for 19200 bps, it could not handle serial input at this rate as such input would cause severe performance degradation, slowing the system to a very, very slow rate! Also, during different experiments, it was found that many CRT terminals are unable to handle 19200 bps witout occasionally losing characters.)

When using the internal clock input, the bit rate of the serial port is determined by the following equation

$$r = \frac{f}{384(256-x)}$$

where f is the 8751 clock input and x is the value loaded into timer 1. Additional contraints are that $f \leq 8$ MHz, for this version of 8751, and that x must be an integer. Following the above constraints, this design settles on a value of 254 for x. Thus, the clock frequency needs to be

$$f = 7.3728 \text{ MHz}$$

How this clock is generated will be described later in this chapter.

Since the on-chip serial I/O is done at TTL levels, some level translations must be made to drive the RS232C lines for communicating with the host computer. This is accomplished using standard line driver and receiver IC's, the MC1488 and the MC1489, respectively. The serial I/O lines are then brought to a DB25 connector.

Finally, the Intel 8751 IC has a built-in power-up reset circuitry. Although Intel recommends using a 10 microfarad capacitor [4], a 33 microfarad unit is used, for generating a long reset pulse. With this value, the actual measured reset duration is more than one second.

3.2.2 A/D converter

The inputs from the analog board to this board are 7 analog signals proportional to the energy densities in each of the passbands. In order for the host computer to act on these, they need to be converted to equivalent digital forms. This is accomplished by an A/D converter (ADC).

The design here uses an ADC0816 single-chip 8-bit converter. This IC has a built-in analog multiplexer of 16 channels, with 4 address input lines. It has a typical conversion time of 100 microseconds, when using a clock frequency of 640 KHz. The conversion is done using a successive approximation method, with all operations performed on-chip; that is, only a start signal needs to be sent to the IC and, when conversion is complete, it signals by asserting an end-of-conversion (EOC) signal. In addition, the digital output is through an 8-bit register having tri-state capability, allowing direct connection to a data bus [10].

In this design, the 4 address lines and the 8 data lines of the ADC0816 share the same 8 port pins of the 8751. This is possible since when conversion is in progress, the 8751 can send the proper address to the ADC0816, latching it using the ALE input of the converter IC. At the same time, it inhibits the output from this chip by sending a logic zero to OUTPUT ENABLE. When conversion is completed, the 8751 switches the port pins to input mode and enable the output of the ADC0816, thus allowing for the data to flow from the converter to the 8751.

Since the ALE and START signals of the ADC0816 are triggered on the leading and falling edge, respectively, they are tied together and driven by a common signal from the 8751. This further reduces the usage of the port pins. In addition, the EOC signal from the converter chip is sent to one port pin of the 8751, allowing the 8751 to sense the end of conversion, rather than waiting the maximum conversion time, making more efficient use of the CPU time. The design here does not use the EOC signal to interrupt the 8751, since it is of the wrong logic level for such a purpose. It is, however, possible to use it as an interrupt input to the 8751: just add an inverter. This is not done because the use of this additional inverter would add one more IC package to this board.

To determine the clock frequency, a few contraints must be observed. One is that, according to the specifications of the ADC0816, its permissible clock range is 10 KHz to 1.28 MHz. Another is that whatever clock frequency, it must be easily derivable from the system crystal. A further consideration is that to insure the best accuracy of conversion, the ADC0816 should not operate at its maximum speed. The designed clock frequency is 614.4 KHz, providing a maximum conversion time of 120 microseconds (74 clock cycles).

3.2.3 Clock circuit

The function of the clock circuit is to generate all the necessary clock signals for both the 8751 microcontroller and the ADC0816 converter. As mentioned above, the clock frequency for the 8751 IC is chosen at 7.3728 MHz. If a crystal of this value is readily available, then it would be easy: just connect this crystal to the on-chip crystal oscillator circuit. Unforturnately, this crystal is relatively rare. To generate 7.3728 MHz, then, requires a crystal whose frequency is some integer multiple of 7.3728 MHz. The lowest value where a crystal would readily be available is 22.1184 MHz, three times the desired frequency. Therefore, a crystal of this value is used as the primary clock.

Unfortunately, with frequency as high as this, the standard two-inverter oscillator, such as one shown in Figure 12, would not work. Thus, another oscillator circuit has to be used. Looking through

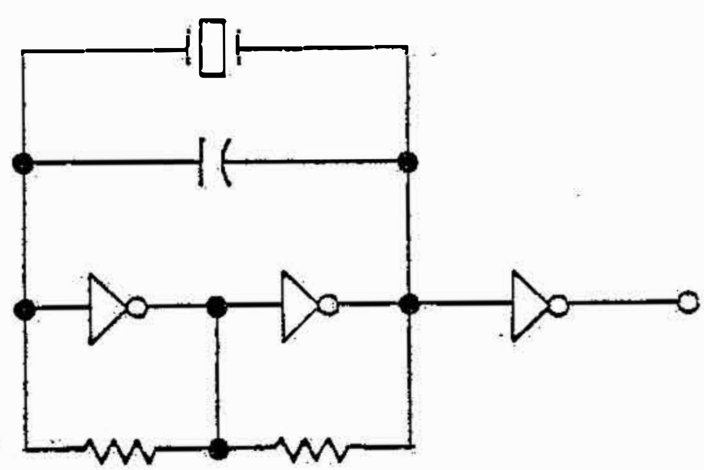


Figure 12 - Two-inverter oscillator.

reference [11] reveals that there are IC's designed for just such purpose. A 74LS629 IC is therefore employed as the oscillator circuit, with the connections as shown in Figure 13. It should be noted here that, although the oscillator as designed works beautifully, this IC is only rated to operate at a maximum frequency of 20 MHz. Therefore, this design is really only marginal, relying primarily on the conservative ratings generally given for TTL IC's. Consequently, if this IC needs replacement in the future, one would have to be careful to make sure the replacement would also operate at 22.1184 MHz.

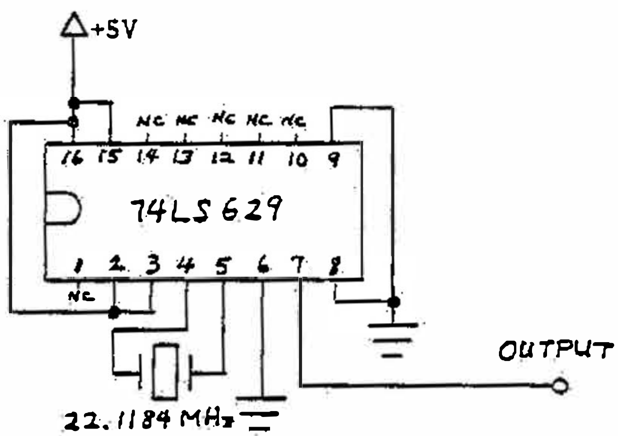


Figure 13 - Crystal oscillator.

To drive the 8751, this clock of 22.1184 MHz needs to be divided down. This is accomplished using a 74LS92 IC, operating as a a divider by three. Note that the resulting clock has a duty cycle of 33%. This is permissible, however, as explained in reference [4].

To drive the ADC0816, the 7.3728 MHz clock is further divided by 12, by another 74LS92 IC, hence the reason for the seemingly odd frequency of 614.4 KHz. A block diagram of this clock circuit is shown in Figure 14.

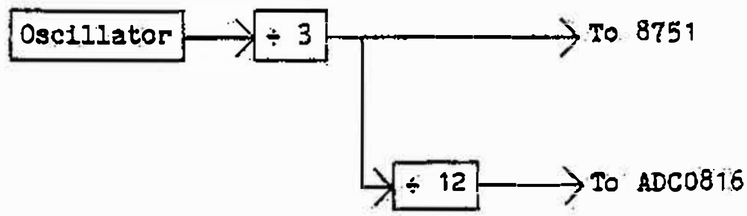


Figure 14 - Block diagram of clock circuit.

3.2.4 Logarithmic amplifier

In order to provide wider dynamic range for the speech signal, a logarithmic amplifier (log amp) is utilized in this voice recognition system. Besides compressing the signals, the log amp also has the side benefit of reducing any multiplication and division operations on the digitized data into addition and subtraction operations, respectively. This would speed up the processing.

There are two logical locations for a log amp. Either one log amp per band is used and placed in front of the multiplexer or only one log amp is used by placing it between the multiplexer and the ADC. This latter approach is chosen for the obvious reason of economy. This unfortunately introduces one problem. That is, after switching the multiplexer, the controller must wait for the log amp to settle before starting the conversion cycle. This, however, is not really that bad, as some settling time must already be allowed when the switching takes place.

The log amp design here relies on the fact that when the collector-base voltage is zero, the emitter-base voltage is proportional to the collector current [7]. Thus, by using the circuit in Figure 15,

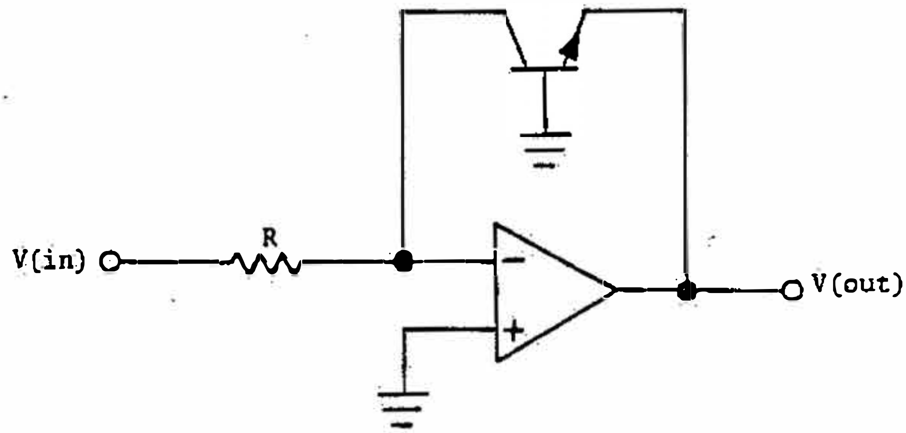


Figure 15 - Simple log amp.

V(out) becomes

$$V(out) = \frac{1}{Q} \ln \frac{V(in)}{RP}$$

where P and Q are constants dependent on the characterics of the transistor, and R is determined experimentally, so as to provide a maximum output swing when the input range is about 0 to 1 volt. There are two problems with this basic circuit. The first is that, if V(in) becomes negative, the op amp essentially has no feedback, causing the output voltage to saturate to the positive supply. This is overcome by connecting a diode in reverse parallel, as shown in Figure 16, introducing a small, but hopefully insignificant, error. A second problem is that at certain input voltage range, the circuit tends to oscillate at very high frequency. This is suppressed by the output capacitor of Figure 16, with the log amp having longer settling time as a side effect.

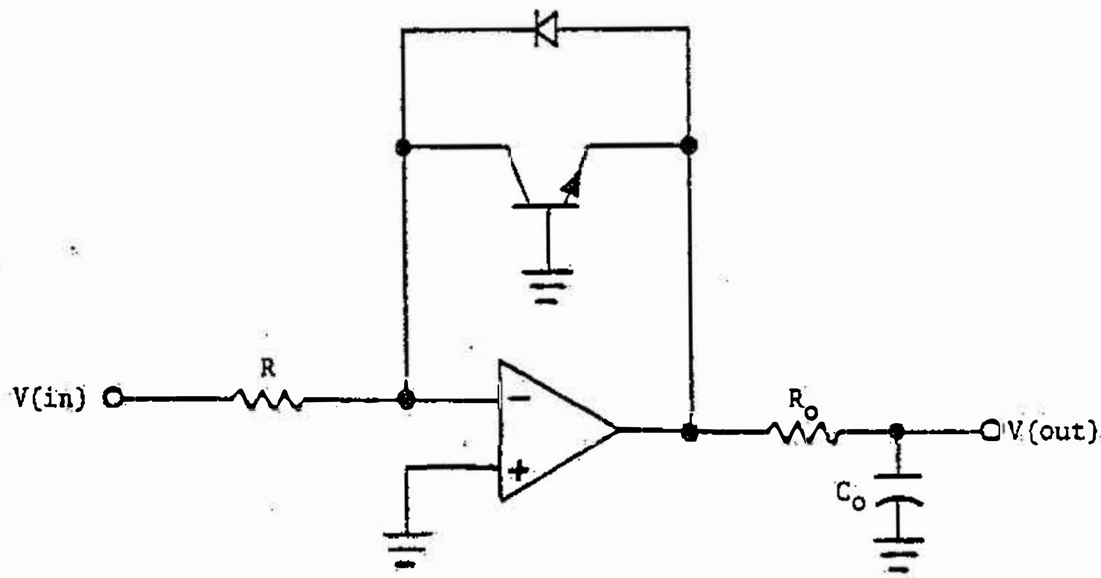


Figure 16 - Logarithmic amplifier.

Since the output voltage range of this log amp is only about -0.3 to -0.7 volt, as determined experimentally using a value of R chosen as described above, when the input range of the converter is 0 to 5 volts, a translation circuit needs to be introduced. The resulting log amp circuit is as shown in Figure 17. Notice that two diodes are added at the output of this log amp, since it is possible for voltages outside the zero to five volts range to appear. They serve as a protection for the input of the ADC, limiting the output of the log amp to one diode drop above 5 volts and one diode drop below ground.

The ZERO and GAIN adjustments are provided to obtain the optimum voltage output of 0 to 5 volts for normal volume levels. The calibration is made by placing the switch in the TEST position and placing the test voltage at the point marked TEST. For the ZERO adjustment, the test voltage is 0 volt and ZERO ADJ potentiometer is

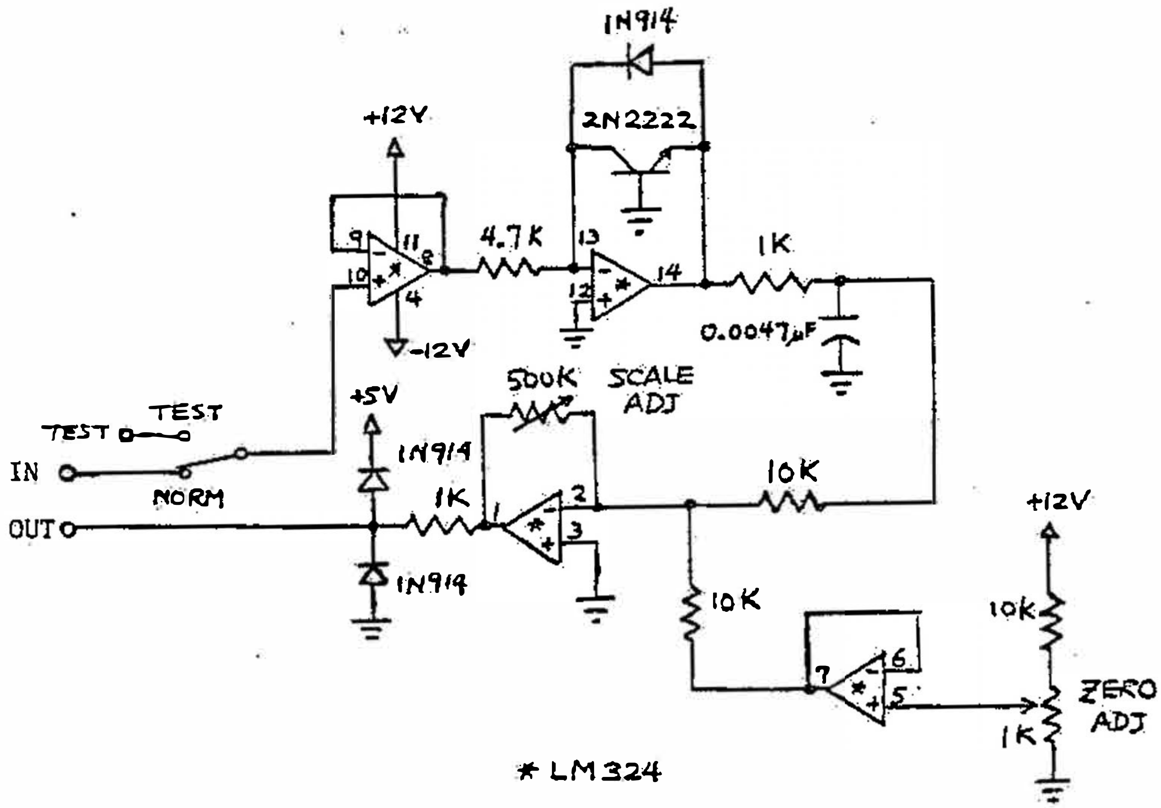


Figure 17 - The log amp circuit.

rotated until the output is barely above -0.7 volt, one diode drop below ground. For the GAIN adjustment, the test voltage is about 1 volt (the normal maximum peak level) and GAIN ADJ is rotated until output is at about 5 volts.

With all the calibrations made, this log amp has a response as shown in Figure 18. For comparison, the response of an ideal log amp is shown on the same plot.

The complete schematic diagrams of the entire voice recognition hardware is included in Appendix A.

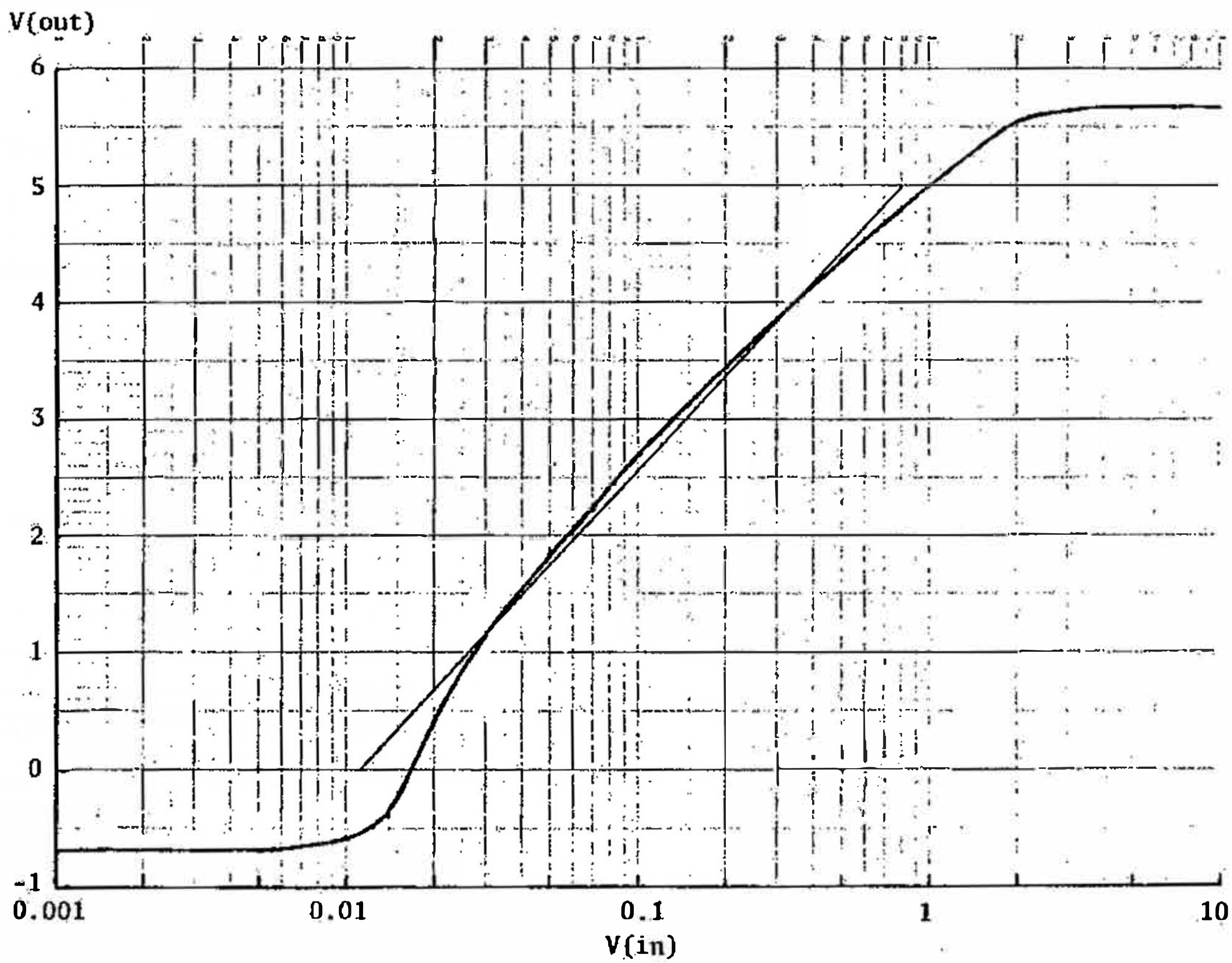


Figure 18 - Log amp response.

CHAPTER 4

SYSTEM SOFTWARE

There are three programs in this voice recognition system, controlling both the training and the recognition sessions. LISTEN runs on the Intel 8751 microcontroller and controls the sampling process. TRAINING runs on the MPT/100 and is in charge of the training session. RECOGN also runs on the MPT/100 and performs the task of matching during the recognition session.

4.1 Intel 8751 Software

The program LISTEN runs on the Intel 8751 microcontroller, in the feature extractor of this system. It has three major tasks: sampling the input voice, detecting the boundaries of spoken words, and communicating with the host computer.

4.1.1 Voice sampling

A voice signal can be modeled as a time-varying frequency spectrum. To track this varying spectrum, the input needs to be sampled at a sufficiently rapid rate. Experimentally, it is determined that a sampling interval of about 10 to 20 milliseconds is adequate for this system.

There are tradeoffs in deciding on the sampling rate. At higher rates, the need for buffering and the amount of computation increase; at lower rates, time-domain resolution may not be sufficient for reliable recogniton. This design uses a sampling period of 15 milliseconds, as a compromise, partly because with it, the system performance is somewhat better than when using 20 milliseconds. At this rate, the buffer RAM of 2 kilobytes could hold upto about 2 seconds of voice samples.

This sampling interval is timed using timer 0 of the 8751. This timer is set up at the start of a sampling period, to expire after 15 milliseconds. LISTEN then determines its status by examining bit TF0, which is set when the desired amount of time has elapsed.

Although other researchers have indicated that time-smoothing of voice signal has no benefits [3], it is found that averaging two adjacent samplings provides somewhat more consistent recognition in this system. Therefore, each frame of data produced by this program consists of 7 averaged values (for the 7 filters) computed from the corresponding bands of current and previous samples. The first set of samples in a word is suppressed and is used as the initial condition.

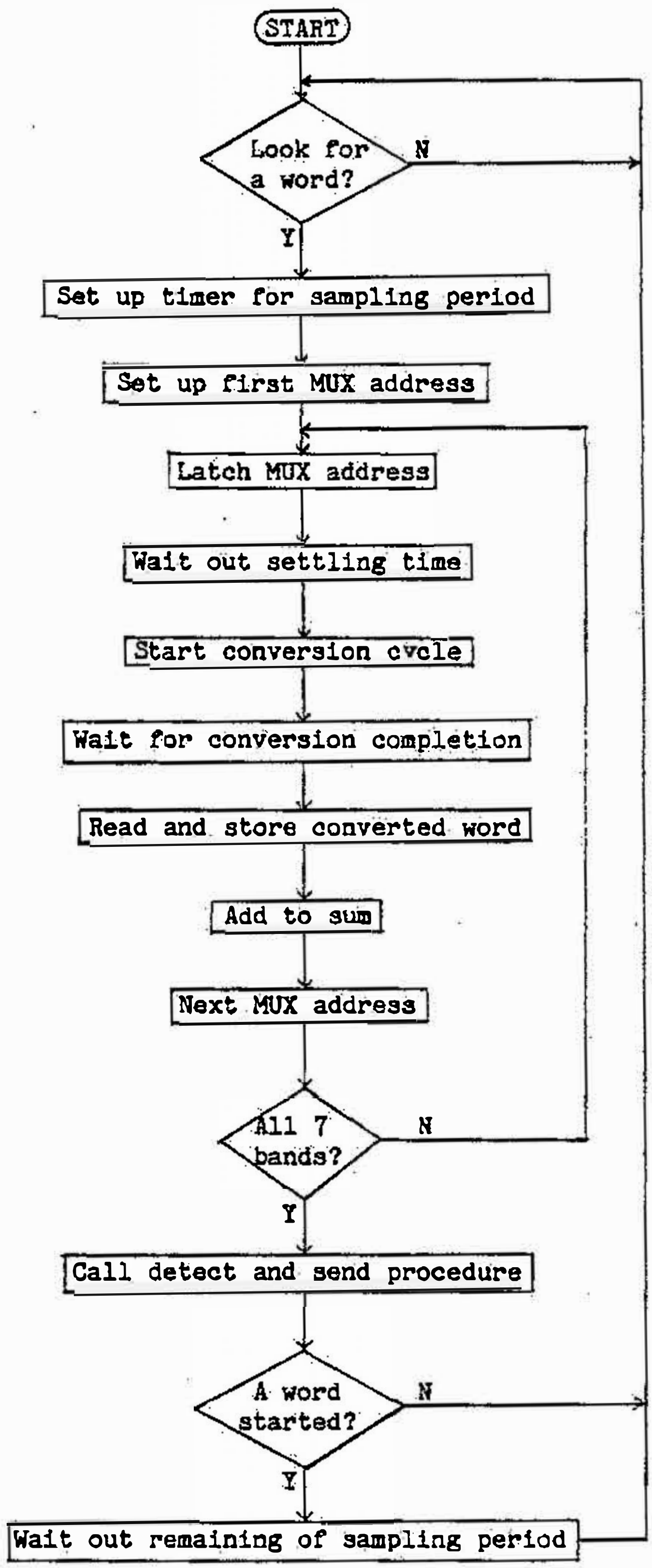


Figure 19 - Main loop of LISTEN.

This portion of LISTEN is implemented as indicated by the flowchart of Figure 19. When the host computer is ready for a word, it sends a command causing flag ST to be set, as will be described later. The sampling then takes place by scanning each of the 7 bands, converting the log amplitudes into digital data. Note that after switching the multiplexer and before starting the A/D conversion, a settling time of 100 microseconds is allowed. Also, as each channel is sampled, it is added to a running sum, for calculating the total energy of this set of samples. This sum is later used by the boundary detection algorithm. Furthermore, while not in the middle of a word, i.e., the start of a word has not yet been detected, the scanning takes place at a very rapid rate (at intervals equaling to the total conversion times for 7 channels) instead of at 15 millisecond intervals. When all 7 channels have been scanned, LISTEN invokes the detection and output procedure, then waits for timer 0 to expire before starting the next scan cycle.

4.1.2 Boundary detection

The algorithm for detecting the boundaries of a word is quite simple. When the input energy exceeds a preset threshold, a word has started. When several frames having total energy below the threshold are encountered, the word has ended. This latter is done to accomodate words with embedded silent intervals, such as the word "it" where there is a short pause between the "i" and the "t" sounds.

Various threshold values were tried in this thesis. The value currently used is determined from experimentation. Similarly, the silence interval allowed is also chosen empirically, compromising between false detection of word ending and noise pickup. These two values are defined at the start of the source program as THR and SPCC.

4.1.3 Communicating with the host computer

There are four aspects in communicating with the host computer: communication protocol, data encoding, buffer management, and interrupt service routine.

4.1.3.1 Communication protocol

For communicationg with the host computer, a simple protocol is used. When the host is ready to accept a word input, it informs this program by sending a STCMD character. Then, for each frame of data, the host requests the frame by sending a RQCMD character, upon which time, this program sends one complete set of data, terminating with a NL character. LISTEN then waits for the host to request data before further transmission. This continues until the end of the word, when an EOWM character and a NL are sent.

Currently, STCMD is "!" (exclamation mark), RQCMD and NL are linefeed codes, and EOWM is "*" (asterisk).

4.1.3.2 Data encoding

The 8-bit data from the ADC are encoded for two reasons. One, most high level languages could not handle pure binary data. And, two, if displayable codes are used, this program can be debugged using a terminal as a host.

Thus, each 8-bit word is encoded into 2 ASCII characters. The format used is simple hexadecimal digits: 0 through 9 and A through F. This encoding incurs a reduction of effective data rate, since each 8-bit word is now transmitted as 2 characters.

4.1.3.3 Buffer management

A circular buffering scheme is employed in this design. Two procedures are responsible for its management. One to put in data, PUTIN, and the other to pull out data, GETOUT. PUTIN sets a flag, XMT, to indicate the presence of data in the buffer. It also checks the TIF flag (see below); if it is set, a character is transmitted immediately, to start the transmitter. GETOUT, after obtaining one character out of the buffer, checks for existence of more data. If none is left, XMT is reset. Note that all data to the host are routed through the buffer.

4.1.3.4 Interrupt service routine

In the 8751, interrupts from the serial transmitter and receiver are handled by a common service routine, as flowcharted in Figure 20. The

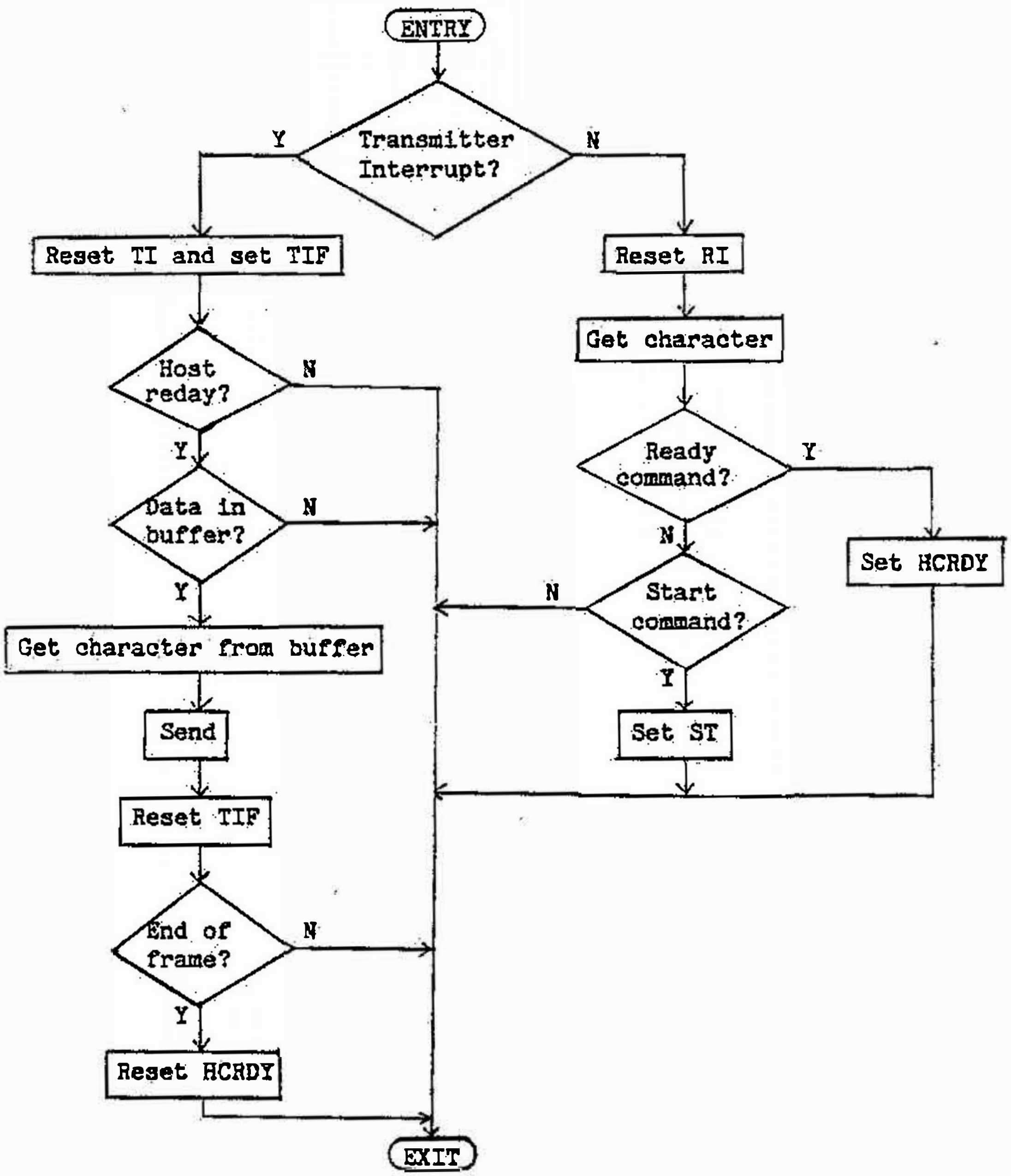


Figure 20 - Interrupt service routine.

source of interrupt is determined by the status of the TI flag, which is set when the transmitter interrupts.

When a transmitter interrupt occurs, the serial port is ready for the next output character. The transmit portion of the interrupt service routine first checks for both HCRDY (host computer ready) and XMT (data available in buffer) before sending the next data byte from the buffer. The flag TIF is used to indicate transmitter availability, if no transmission takes place during this interrupt. This is to cover the case where the transmitter is ready but the buffer contains no data (PUTIN routine is then responsible for restarting the transmitter). Also, if the data transmitted is the end of one frame, HCRDY flag is reset to indicate that the host is busy processing this frame.

When the receiver interrupts, there is data from the host. Normally, this can only be one of two commands: STCMD and RQCMD. The appropriate flag is set to reflect the command received: ST for STCMD and HCRDY for RQCMD.

4.2 Training Software

The program TRAINING, written in Pascal to run on the MPT/100, controls the training session of this system, in conjuction with LISTEN program above.

4.2.1 Vocabulary file

As explained in Chapter 2, the vocabulary file VOCAB contains essentially the raw data from the feature extractor. This file is made up of records representing each word in the vocabulary. Each record contains the text word itself, the encoded data frames, and a separator,

```
One
77575C48000044
2799B100000000
3FA1C2383A0000
3FB9D382830000
60CADEA6A50000
86D6E9BBBA0905
8EDAEDC1C1221C
8FD9ECC2C13A39
8BD7EACAC9574D
86D1E7E0E07B60
86CBEAEDEE8F6E
80C9EEE1E28F71
7AC7EECFD19275
7CC7EFCDCFA578
7CCAF1CCCDA76E
79CEF4C8C9A260
75D1F6C6C8A44F
7AD7FBC8C8AA41
87DFFFCBCBAE3B
8FE4FFC6C6971B
8FE8FFBDBD7600
87E8FFB5B45400
5FE0FA9C9B2000
3FD9F278790000
3FD6ED5A5C0000
3FD2E83B330000
3FCFE6160C0000
3FCEE600080000
3FCFE600080000
3FCDE400000000
58CCE300000000
58C7DB00000000
2FB9CA00000000
2F9FB000000000
*
```

Figure 21 - Sample vocabulary record.

as shown in the example of Figure 21. Each record is stored sequentially in this file.

4.2.2 Noise rejection

To prevent noise from being stored as a word, this program checks for minimum and maximum word length, in number of frames. When a received "word" contains too many or too few frames, it is rejected immediately. In addition, before storing into VOCAB, TRAINING seeks confirmation. This way, the user has a chance to ignore an entry, either because it was noise or because it was not vocalized properly.

4.2.3 Operation

When this program is invoked, it first prompts for the minimum sample length and the silence period count. The first value should be large enough to reject most noises, such as a bump on the microphone, but small enough not to ignore a legitimate word. A value of 15 seems to be a good compromise. The second corresponds to the maximum allowable silence period. It is used to truncate the last few received frames which correspond to silence. This value should agree with SPCC of the LISTEN program, currently set at 8.

After the above initialization, TRAINING prompts for the user to enter one vocabulary word or a session-ending command. If a word is to be added to the vocabulary, it is typed in; then TRAINING waits for its vocalization. A confirmation prompt is presented when the end of a valid word is received, as explained above. When the entire set of words have been trained, the user enters a "*" to exit this program.

4.3 Recognition Software

The program RECOGN is central to this voice recognition system. In conjuction with LISTEN and the file VOCAB, this program attempts to recognize any spoken word.

4.3.1 Voice pattern representation

The voice patterns of the vocabulary in this program are represented using an array of records, one record for each word. Each record contains three fields. TEXT contains the text of the word. N stores the number of frames in this word. And, DAT is a 7-by-N array of integers representing the amplitudes of each band in each frame.

Because the vocabulary file contains only raw, but encoded, data, some transformation is needed. Thus, each record is processed as follows. The text word is read in and stored in the TEXT field. Each frame of the pattern is converted into 7 integers, representing each of the 7 bands, and placed into the array of the DAT field. This continues until the separator is encountered. The actual number of frames found is then stored in the N field.

4.3.2 Energy normalization

Since no one can enunciate any given word with the same amplitude all the time, energy normalization is needed. There are many methods to accomplish this: average energy, peak energy, proportional, and equal-sum normalization.

Average energy normalization attempts to compensate for the energy variation at the frame level. The average energy of each frame is first calculated using

$$E(t) = \frac{\sum X(f,t)}{N}, \text{ for } f=1,\ldots,N$$

where $X(f,t)$ is the amplitude of band f at time t and N is the number of bands, 7 in this case. Each value in the frame is then adjusted using

$$X'(f,t) = X(f,t) - E(t)$$

The effect of this normalization is to adjust the total energy of each frame to zero. That is, each frame has zero sum.

Similarly, the peak energy normalization also adjusts at the frame level. This is accomplished by

$$E(t) = \text{MAX } X(f,t), \text{ for } f=1,\ldots,N$$

$$X'(f,t) = X(f,t) - E(t)$$

using the peak level of each frame. The theory is that now the peak amplitudes in each frame are equal: all are zeroes.

Instead of normalizing at the frame level, the proportional normalization attempts to preserve the frame-to-frame relationship. This algorithm first finds the maximum average energy, A, of the entire word. Then, each value of each frame is adjusted according to

$$X'(f,t) = X(f,t) + \frac{[E(t)-L][M-A]}{A-L}, \text{ if } E[t]>L$$

$$X'(f,t) = X(f,t), \text{ if } E[t]\leq L$$

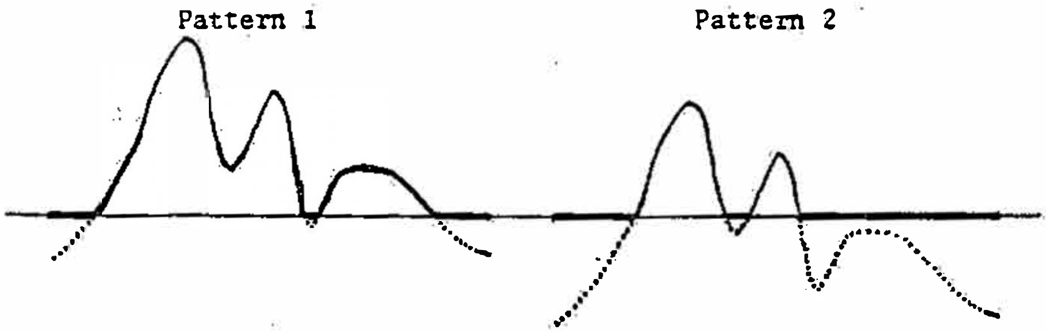
where $E(t)$ is the average energy of each frame, and M and L are preset constants to be determined as follows. M is chosen such that $M>E(t)$ for all t, representing the ceiling where all values are being scaled up. L is chosen to be a noise level, below which the input is considered as silent [1].

Two of the above algorithms were actually tried in the work of this thesis, namely average energy and proportional normalizations. Both of them, however, suffer from a threshold effect existing in this feature extractor: all input signals below a certain amplitude, the threshold, are always represented as zero after the A/D conversion. The normalization methods just described treat this zero value as any others. Referring to Figure 22(a), two spectra of identical shape but different energies are detected as shown by the solid lines, due to the threshold effect. After normalization, shown in Figure 22(b), the two do not look alike. Thus, unreliable operation results. Although the peak energy algorithm was not tried, it would also suffer from this same problem.

To overcome this, an alternate algorithm is used. The equal-sum normalization attempts to normalize, at the frame level, by shifting the spectrum of higher energy downward, clipping at the threshold, until both areas under the curve are equal, as shown in Figure 23. Note that equal area in the continuous case becomes equal sum in the discrete case, as is the situation here; hence the name equal-sum.

This algorithm is implemented as follows:

1. Calculate the sums of the two frames to be compared.



(a) Before normalization.

(b) After normalization.

Figure 22 - Effect of conventional normalization.

Figure 23 - Effect of equal-sum normalization.

2. Determine the smaller of the two and use that frame as the reference for the remainder of this algorithm; the other becomes the test frame.

3. Compute the absolute difference, D, of the two sums.

4. Find the number of non-zero bands, N, in the test frame.

5. If N=0, the normalizaton is done.

6. Calculate the adjustment factor, A=:D/N.

7. If A=0, the algorithm terminates.

8. For each non-zero band of value V in the test frame, if V>A, then V=:V-A and D=:D-A; otherwise, D=:D-A, V=:0, and N=:N-1.

9. Repeat steps 5 to 8.

Experiments, done as part of the work for this thesis, showed that this increases the correct recognition rate (to be defined later) from 31 to 78 percent. In RECOGN, the function NORM handles this algorithm.

4.3.3 Local distance function

The purpose of a local distance function is to generate a number representing the similarity of two frames of voice patterns. The smaller this value, the more similar they are. There are various ways of computing this: Euclidean distance, Chebychev norm, p-power distance, and maximum magnitude.

The Euclidean distance function uses the analogy of distance in n-dimensional space, where n=7 in this case. This is calculated by

$$d = \left(\frac{\sum [X(i)-Y(i)]^2}{N}\right)^{\frac{1}{2}}, \text{ for } i=1,\ldots,N$$

where X(i) and Y(i) are amplitudes of band i for the two frames, and N is the number of bands. This function has one significant drawback: square and square-root operations are time-consuming.

Chebychev norm distance function is computationally more efficient. It is calculated by

$$d = \frac{\sum |X(i)-Y(i)|}{N}, \text{ for } i=1,\ldots,N$$

This method was tried during the work of this thesis. However, the results were not too encouraging, when compared to the one actually implemented.

The preceding two functions are actually special cases of the p-power distance function, where the distance is determined by

$$d = \left[\frac{\sum [X(i)-Y(i)]^p}{N}\right]^{\frac{1}{p}}, \text{ for } i=1,\ldots,N$$

where p is a real number. The Euclidean distance is simply the case of p=2; the Chebychev norm, p=1. Again, this method is not computationally efficient. In addition, past researches have shown that p other than 1 did not yield better performance [1].

The local distance function employed in this program is the maximum magnitude function,

$$d = \text{MAX } |X(i)-Y(i)|, \text{ for } i=1,\ldots,N$$

This function tends to emphasize the difference between two frames, without using a p value other than 1. (This would seem to contradict the results of past research [1], however.) Computationally, this algorithm is about as efficient as the Chebychev norm function.

The local distance function is implemented in RECOGN by the function DIST.

4.3.4 Matching algorithm

The heart of the recognition software is the matching algorithm. Its function is to return a measure of similarity between two voice patterns. Again, there are several methods available. All of them take into consideration that the lengths of the patterns may not be the same. The common algorithms are: linear time normalization (LTN), LTN with boudary adjustment, dynamic time warping (DTW) using dynamic programming (DP), and DTW using band DP.

LTN is the simplest algorithm. One pattern is simply mapped onto the other. Then, the local distances between the corresponding frames are summed. The similarity score is the sum divided by the number of frames compared. Since the two patterns are usually of different lengths, the shorter one is "stretched" to match the length of the longer (this was found, from past research, to be better than the converse method). This is accomplished by either mapping some frames of the shorter pattern onto more than one frame of the reference or by using interpolation. The first method is computationally more efficient than the second; thus, it was tried in this thesis. The results, however, were not very good, especially for patterns other than the one used in training. The reason may be erroneous boundary determination.

Therefore, LTN with boundary adjustment algorithm was attempted. This was implemented by repeating LTN matching several times, truncating different number of frames from the beginning and from the ending of each of the two patterns. Figure 24 illustrates 3 possible matching paths, with the path 1 being the same as no boundary adjustment. (A and B are the lengths of the patterns.) But again, the result was not good.

The reason for the poor performance of LTN is that when words are vocalized at different times, a phenomenum known as time warping occurs.

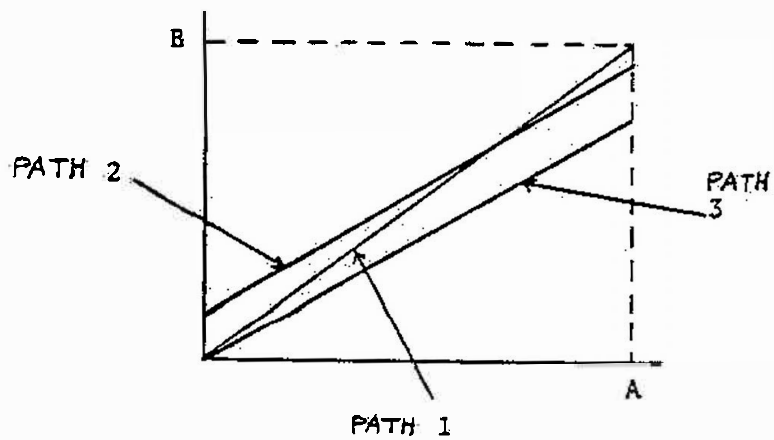


Figure 24 - Linear-time normalization with boundary adjustments.

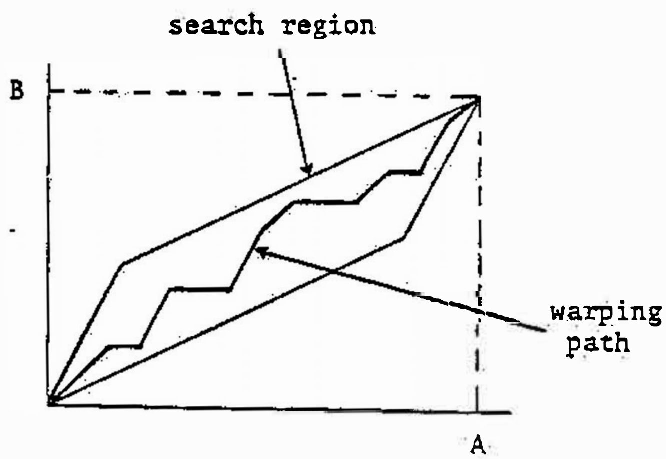


Figure 25 - Dynamic-time warping.

That is, part of the word is vocalized at faster speed relative to other parts. To compensate for this, the DTW algorithm is used. In DTW, the matching takes place in a warping path. Referring to Figure 25, the horizontal and vertical axes represent the time domains of the reference and the test patterns, respectively. The matching is on a path of minimum local distance, subject to slope constraints and bounded by the search region. The accumulated sum of the local distances at any point (x,y) is given by the DP equation

$$D(x,y) = d(x,y) + MIN[D(x-1,y),D(x-1,y-1),D(x-1,y-2)]$$

where D(x,y) is the accumulated distance and d(x,y) is the local distance [1]. The net effect of this method is to move along the two time axes at different rates, corresponding to the stretching and compressing of time. More details of DTW and DP are found in reference [8].

DTW as described above assumes accurate boundary determination, which may not be true. Therefore, a modified DTW algorithm is used in this thesis. This method is called band DP since the search region has

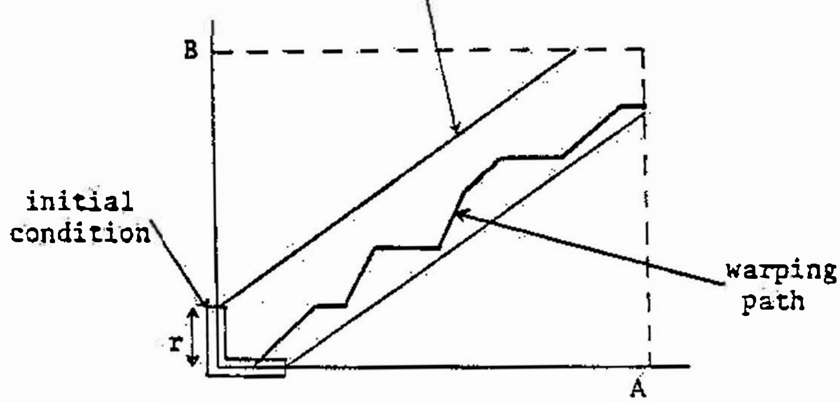


Figure 26 - Band DP.

been modified to resemble a band, as shown in Figure 26. The marked region near the origin is the initial condition. The value r is chosen to window the search region, which also has the effect of adjusting boundaries by a maximum of r frames. Past research indicates that this method is superior [1].

In this program, the matching takes place by using the longer pattern as the reference. The algorithm is implemented by the function TOTAL_DIST.

4.3.5 Recognition criteria

Once the scores between the test pattern and the reference patterns had been determined, some criteria must be used to decide on the matched word, if any. The criteria used in this thesis are twofold: the score must be less than a threshold, and it must also differ from the next best score by a differentiating factor.

This approach is taken mainly to provide protection from false recognition, when a test word not in the vocabulary is spoken. By adjusting these two values, it is possible for this program to be tailored, i.e., to achieve a higher correct recognition rate at the expense of higher error rate (these two terms will be defined in Chapter 5).

4.3.6 Operation

When RECOGN is invoked, it first prompts for the silence period count, which should be answered with the value of SPCC from LISTEN, currently set at 8. The vocabulary is then read into the array WORDS.

There are several commands available to the user: A to adjust parameters, L to list vocabulary, R to run a test input, and Q to quit. The adjustable parameters are: minimum sample length, threshold value for decision, differentiation factor, boundary adjustment (size of the search region of band DP), and normalization (whether it should be performed). These are provided for experimentation. The best settings seem to be that listed in Table 2.

The complete listings of all three programs is found in Appendices B, C, and D. Also, the complete operation instructions are outlined in Appendix E.

TABLE 2 - BEST PARAMETER SETTINGS OF RECOGN.

(1) Threshold factor:	35
(2) Differentiation factor:	5
(3) Boundary adjustment:	1
(4) Minimum sample count:	15
(5) Normalization:	Y

CHAPTER 5

PERFORMANCE

To find out the effectiveness of this voice recognition system, measurements of its performance are made. In this chapter, the results of these measurements are presented, using both recorded and live voices.

5.1 Measuring Performance

For measuring the performance, several terms are defined here. When a word from the vocabulary is spoken as the test input, a <u>correct recognition</u> occurs if it is correctly identified, an <u>error</u> if it is matched to the wrong word, and a <u>miss</u> if the system does not return a match. When a word not in the vocabulary is the test input and this system returns a match, a <u>false recognition</u> takes place. The performance measures used in this thesis, then, are the rates at which these events occur, expressed in percentages.

5.2 Performance of Recorded Voice

For this set of measurements, a recording of several words was made on a cassette tape. These are the digits 0 to 9 and the number 10. Also, to test the rejection of noise by this system, two "words" of noise were recorded, by blowing air into the microphone. Each word is vocalized 4 consecutive times, separated by time intervals of approximately 5 seconds. This recording was made in a fairly noisy room, where an airconditioner was operating. The voice was that of this author, a male voice of average pitch with moderate accent.

The training session employed the first replications of 7 of the 11 words:

ONE	FIVE
TWO	SEVEN
THREE	TEN
FOUR	

The playback of the tape was from a pocket cassette player, using a volume level of 4, as indicated on its dial. The microphone was placed approximately one half inch from the speaker of this player.

Three sets of measurements were made using this same vocabulary file, using all the recorded words. The first set (test A) used playback level of 4, identical to the training session. The second and third sets (tests B and C) had different playback levels, 5 and 3, respectively. All three used parameter settings of RECOGN from Table 2 of the preceding chapter.

The data taken from these 3 set of measurements are recorded in Tables 3, 4, and 5 (pp. 59-61). Notice that in addition to the returned results, the two highest scores for each test word are included for comparison. The resulting performance measures are summarized in Table 6 (p. 62).

Referring to Tables 3, 4, and 5, the number "ten" was never recognized correctly, except for the one utterance trained. This would seem to indicate a bad pronunciation of the first "ten." To test the effect of not using the recordings of "ten," the four test words "ten" are ignored; plus, any matches to it are removed, using the next best word, instead. The effect of this manipulation is a much improved performance, as shown in Table 7 (p. 62). Notice that, with this, the error rate is down to zero.

5.3 Performance of Live Voice

To get an idea of how this system would perform in a real-life situation, measurements were made using live voice. This was done by first training using the same set of words as above, in a quiet room where only the fan noises of the MPT/100 computer and of the room ventilation system were present. Again, the voice used was this author's. During training, the microphone was hand held at about one inch from the mouth (no attempt was made to precisely maintain this distance).

Several recognition sessions were conducted. One immediately after training (test D), a second one an hour later (test D′), and a third 2 days later (test E). Extensive measurements were made only for tests D and E; test D′ was just a quick check. The parameter settings for the program RECOGN were the same as above. And, attempts were made to pronounce the words as closely to the trained ones as possible; that is, a cooperative speaker.

The measurements are presented in Tables 8, 9, and 10 (pp. 63-65). The performance measures are summarized in Table 11 (p. 66). Comparing the results of recorded and live tests, it is seen that when recognition occurs immediately after training, the performance of live voice is almost as good as recorded voices. However, with elapsing time, the performance of live voice degrades, presumably because the user forgets how the words were pronounced during training.

TABLE 3 - RESULTS OF TEST A.

Test Word	Result	Best Score	Best Word	Second Score	Second Word
* ONE	C	10	=	34	FOUR
ONE	C	20	=	32	FOUR
ONE	C	23	=	32	FOUR
ONE	?	28	FOUR	31	=
* TWO	C	10	=	45	SEVEN
TWO	C	19	=	37	SEVEN
TWO	C	19	=	41	FOUR
TWO	C	22	=	39	FOUR
* THREE	C	10	=	50	FIVE
THREE	C	15	=	50	FIVE
THREE	C	33	=	47	FOUR
THREE	C	25	=	58	FIVE
* FOUR	C	5	=	33	ONE
FOUR	C	15	=	29	ONE
FOUR	X	37	=	41	ONE
FOUR	C	18	=	35	ONE
* FIVE	C	10	=	49	ONE
FIVE	X	36	=	46	ONE
FIVE	C	27	=	45	ONE
FIVE	C	24	=	46	ONE
# SIX	X	39	TEN	47	SEVEN
# SIX	F	35	TEN	45	SEVEN
# SIX	X	43	TEN	50	SEVEN
# SIX	X	47	SEVEN	55	TEN
* SEVEN	C	7	=	34	TEN
SEVEN	C	17	=	36	TEN
SEVEN	C	19	=	34	TEN
SEVEN	C	20	=	41	TWO
# EIGHT	F	32	TEN	42	FIVE
# EIGHT	X	37	TEN	40	FIVE
# EIGHT	X	36	FIVE	43	TEN
# EIGHT	X	38	FIVE	40	TEN
# NINE	F	34	ONE	46	TWO,FOUR
# NINE	X	38	ONE,TWO		
# NINE	F	32	ONE	44	TWO
# NINE	X	43	FIVE	47	ONE
# ZERO	?	31	TWO	34	FOUR
# ZERO	F	25	FOUR	37	ONE
# ZERO	X	45	FOUR	51	ONE
# ZERO	?	35	TWO	37	FOUR
* TEN	C	10	=	37	SEVEN
TEN	X	36	SEVEN	37	=
TEN	E	33	TWO	44	=
TEN	E	27	TWO	35	SEVEN
# noise	X	62	TEN	73	SEVEN
# noise	X	53	TEN	63	SEVEN

NOTES:
* - trained version
\# - not in vocabulary
= - same as test word
? - ambiguous result

C - correct recognition
F - false recognition
X - no match
E - error

TABLE 4 - RESULTS OF TEST B.

		Best		Second	
Test Word	Result	Score	Word	Score	Word
* ONE	C	14	=	37	FOUR
ONE	C	25	=	34	FOUR
ONE	C	33	=	39	FOUR
ONE	?	33	FOUR	35	=
* TWO	C	22	=	46	FOUR
TWO	C	22	=	44	FOUR
TWO	C	26	=	44	FOUR
TWO	C	27	=	43	FOUR
* THREE	C	18	=	45	FIVE
THREE	C	28	=	48	FOUR
THREE	X	37	=	45	FOUR
THREE	C	29	=	44	FOUR
* FOUR	C	18	=	46	ONE
FOUR	C	18	=	38	ONE
FOUR	X	39	=	50	ONE
FOUR	C	34	=	50	ONE
* FIVE	C	18	=	47	ONE
FIVE	X	36	=	44	ONE,FOUR
FIVE	C	23	=	46	ONE
FIVE	C	25	=	47	ONE
# SIX	X	41	SEVEN	49	TEN
# SIX	X	37	TEN	41	SEVEN
# SIX	X	42	TEN	46	SEVEN
# SIX	X	42	SEVEN	54	TEN
* SEVEN	C	17	=	39	TWO
SEVEN	C	17	=	37	TWO
SEVEN	C	24	=	40	TEN
SEVEN	C	24	=	35	TWO
# EIGHT	X	38	FIVE	44	ONE
# EIGHT	X	40	ONE,FIVE		
# EIGHT	F	35	FIVE	40	ONE
# EIGHT	X	36	FIVE	41	ONE
# NINE	F	31	ONE	41	FOUR
# NINE	?	35	FOUR	39	ONE,TWO
# NINE	?	35	ONE	36	FOUR
# NINE	X	37	FOUR	40	ONE
# ZERO	X	38	FOUR	39	ONE
# ZERO	X	37	FOUR	44	ONE
# ZERO	X	45	FOUR	51	ONE
# ZERO	X	36	TWO,FOUR		
* TEN	C	25	=	32	SEVEN
TEN	E	31	TWO	43	SEVEN
TEN	E	27	TWO	44	SEVEN
TEN	E	27	TWO	44	SEVEN
# noise	X	59	TEN	69	SEVEN
# noise	X	50	TEN	61	SEVEN

NOTES: * - trained version C - correct recognition
- not in vocabulary F - false recognition
= - same as test word X - no match
? - ambiguous result E - error

TABLE 5 - RESULTS OF TEST C.

		Best		Second	
Test Word	Result	Score	Word	Score	Word
* ONE	C	30	=	39	THREE,FOUR
ONE	X	54	FOUR	63	=
ONE	?	24	=	28	FOUR
ONE	?	22	FOUR	24	=
* TWO	C	14	=	28	FOUR
TWO	C	18	=	28	FOUR
TWO	C	16	=	37	FOUR
TWO	C	18	=	27	FOUR
* THREE	C	20	=	51	FOUR
THREE	C	25	=	46	FOUR
THREE	C	34	=	43	FOUR
THREE	C	30	=	50	FOUR
* FOUR	C	8	=	28	ONE
FOUR	X	46	=	56	ONE
FOUR	C	23	=	31	ONE
FOUR	C	14	=	28	ONE
* FIVE	C	27	=	39	ONE
FIVE	?	34	=	38	ONE
FIVE	C	24	=	42	ONE
FIVE	?	30	=	31	ONE
# SIX	X	38	FIVE	42	ONE
# SIX	X	36	FIVE	39	ONE
# SIX	F	28	TEN	45	SEVEN
# SIX	F	31	TEN	49	SEVEN
* SEVEN	C	10	=	26	TEN
SEVEN	C	15	=	34	TWO
SEVEN	C	15	=	33	TEN
SEVEN	?	22	=	26	TWO
# EIGHT	F	35	TEN	53	ONE
# EIGHT	F	35	TEN	49	ONE
# EIGHT	X	36	TEN	46	FIVE
# EIGHT	X	41	TEN	47	ONE
# NINE	F	33	ONE	43	SEVEN
# NINE	F	35	ONE	41	TWO
# NINE	F	34	ONE	43	FOUR,SEVEN
# NINE	X	42	ONE	45	FOUR
# ZERO	X	44	ONE	46	FOUR
# ZERO	F	27	FOUR	39	ONE
# ZERO	X	36	FOUR	41	ONE,TWO
# ZERO	?	35	FOUR	37	ONE,TWO
* TEN	C	28	=	41	SEVEN
TEN	X	38	TWO	40	SEVEN
TEN	E	30	TWO	37	SEVEN
TEN	E	30	TWO	43	SEVEN
# noise	X	63	TEN	71	SEVEN
# noise	X	53	TEN	64	SEVEN

NOTES: * - trained version C - correct recognition
- not in vocabulary F - false recognition
= - same as test word X - no match
? - ambiguous result E - error

TABLE 6 - PERFORMANCE OF RECORDED VOICE.

Performance Measure	Test A	Test B	Test C
Correct recognition	78.6	75.0	64.3
Miss ratio	14.3	14.3	28.6
Error rate	7.1	10.7	7.1
False recognition	24.8	11.1	44.4

TABLE 7 - PERFORMANCE OF RECORDED VOICE WITHOUT "TEN."

Performance Measure	Test A	Test B	Test C
Correct recognition	87.5	83.3	70.8
Miss ratio	12.5	16.7	29.2
Error rate	0.0	0.0	0.0
False recognition	16.6	11.1	22.2

TABLE 8 - RESULTS OF TEST D.

Test Word	Result	Best Score	Best Word	Second Score	Second Word
ONE	C	13	=	32	TWO
ONE	?	25	FOUR	26	=
ONE	C	18	=	28	FOUR
ONE	C	14	=	29	FOUR
TWO	C	17	=	25	ONE,FOUR
TWO	C	19	=	27	FOUR
TWO	C	16	=	25	FOUR
TWO	C	18	=	24	FOUR
THREE	C	28	=	38	FIVE
THREE	X	39	=	40	FIVE
THREE	C	27	=	36	FIVE
THREE	C	22	=	32	FIVE
FOUR	C	25	=	30	SEVEN
FOUR	C	20	=	25	TWO
FOUR	C	25	=	31	SEVEN
FOUR	?	22	=	23	SEVEN
FIVE	C	19	=	28	SEVEN
FIVE	C	18	=	27	SEVEN,TEN
FIVE	C	24	=	29	SEVEN
FIVE	?	18	=	21	TEN
# SIX	?	17	FIVE,SEVEN		
# SIX	?	12	TEN	15	SEVEN
# SIX	?	23	FIVE	29	SEVEN
# SIX	F	31	FIVE	36	SEVEN
SEVEN	C	12	=	26	FIVE
SEVEN	C	25	=	30	FIVE
SEVEN	C	17	=	30	FIVE
SEVEN	?	26	=	30	FIVE,TEN
# EIGHT	X	46	TEN	47	SEVEN
# EIGHT	X	38	FIVE	45	SEVEN
# EIGHT	X	47	SEVEN	48	TEN
# EIGHT	F	34	FIVE	41	SEVEN
# NINE	F	27	FIVE	37	FOUR
# NINE	F	19	FIVE	31	THREE,TEN
# NINE	?	28	TEN	31	FIVE
# NINE	?	32	FIVE	33	TEN
# ZERO	?	27	TEN	28	FIVE
# ZERO	?	23	SEVEN	24	TEN
# ZERO	?	30	TEN	34	FIVE
# ZERO	X	49	FIVE	50	TEN
TEN	C	25	=	38	SEVEN
TEN	X	36	=	47	SEVEN
TEN	C	35	=	42	SEVEN
TEN	C	21	=	30	FOUR
# noise	X	43	SEVEN	45	FIVE
# noise	?	33	SEVEN	34	FIVE

NOTES: # - not in vocabulary
= - same as test word
? - ambiguous result
C - correct recognition
F - false recognition
X - no match
E - error

TABLE 9 - RESULTS OF TEST D′.

Test Word	Result	Best Score	Best Word	Second Score	Second Word
ONE	C	10	=	26	TWO
TWO	C	20	=	31	FOUR
THREE	C	22	=	40	FIVE
FOUR	C	24	=	29	TWO
FIVE	C	22	=	35	SEVEN
# SIX	X	37	FIVE	38	SEVEN
SEVEN	?	26	FIVE	27	=
# EIGHT	X	40	FIVE	48	SEVEN
# NINE	?	23	FIVE	27	TEN
# ZERO	?	28	TEN	30	FIVE,SEVEN
TEN	X	40	SEVEN	47	=
# noise	?	32	SEVEN	34	TEN

NOTES: # - not in vocabulary
= - same as test word
? - ambiguous result
C - correct recognition
F - false recognition
X - no match
E - error

TABLE 10 - RESULTS OF TEST E.

Test Word	Result	Best Score	Best Word	Second Score	Second Word
ONE	X	38	=	43	FOUR
ONE	E	29	FIVE	38	TWO
ONE	X	38	TEN	41	FIVE
ONE	C	24	=	40	FOUR
TWO	C	23	=	32	FOUR
TWO	C	19	=	26	ONE
TWO	?	29	=	33	FOUR
TWO	C	21	=	32	FOUR
THREE	C	30	=	49	FIVE
THREE	C	33	=	41	FIVE
THREE	X	36	=	50	FIVE
THREE	C	20	=	40	FIVE
FOUR	?	31	ONE,=		
FOUR	?	34	=	36	ONE
FOUR	E	28	ONE	35	=
FOUR	X	41	ONE	46	=
FIVE	?	22	SEVEN	25	=
FIVE	?	22	=	23	SEVEN
FIVE	C	23	=	29	SEVEN
FIVE	C	19	=	27	SEVEN
# SIX	X	40	FIVE	49	SEVEN
# SIX	X	47	SEVEN	48	FIVE
# SIX	X	44	FIVE	55	SEVEN
# SIX	F	23	FIVE	34	SEVEN
SEVEN	C	23	=	28	FOUR
SEVEN	C	24	=	34	FOUR
SEVEN	C	25	=	34	FOUR
SEVEN	?	26	FIVE,=		
# EIGHT	X	39	FOUR,FIVE		
# EIGHT	X	53	SEVEN	64	FOUR,TEN
# EIGHT	X	50	SEVEN	52	FOUR
# EIGHT	?	34	FOUR	35	SEVEN
# NINE	F	20	FIVE	34	THREE
# NINE	X	36	FIVE	38	TEN
# NINE	F	31	FIVE	37	TEN
# NINE	F	22	FIVE	38	TEN
# ZERO	F	26	FOUR	31	FIVE,SEVEN
# ZERO	?	27	FIVE	31	FOUR
# ZERO	?	24	SEVEN	27	FIVE
# ZERO	X	37	FIVE,TEN		
TEN	X	46	SEVEN	47	=
TEN	?	22	FOUR	24	SEVEN
TEN	?	30	SEVEN	31	=
TEN	X	38	SEVEN	45	FOUR
# noise	X	39	SEVEN	44	FIVE
# noise	X	38	FIVE	41	SEVEN
# noise	?	8	ONE,TEN		

NOTES: # - not in vocabulary
= - same as test word
? - ambiguous result

C - correct recognition
F - false recognition
X - no match
E - error

TABLE 11 - PERFORMANCE OF LIVE VOICE.

Performance Measure	Test D	Test D'	Test E
Correct recognition	78.6	71.4	42.9
Miss ratio	21.4	28.6	50.0
Error rate	0.0	0.0	7.1
False recognition	22.2	0.0	26.3

CHAPTER 6

FUTURE OF THIS SYSTEM

6.1 Future Improvements

As is seen from the preceding chapter, the performance of this voice recognition system is less than perfect. Based on the experiences with the current system, many possibilities for improvements exist. For one, the number of filters may be increased to provide better frequency resolution. This may be accomplished by adding more analog filters, or, better still, using digital filters. A second possibility is to segment the input pattern into phoneme-like units. This would allow the system to discriminate using different sounds, rather than strictly on the pattern itself.

Possibilities also exist for reducing processing time. Using phoneme-like units, as mentioned above, can speed up processing. Also, other data reduction methods need to be investigated. Furthermore, it is possible to operate a recognition system with a network of microprocessors. Then, the processors can be searching in parallel, thereby significantly reduce the computation time.

At the present, this system is speaker-dependent. This limits the applications of this system. Investigations into improvements may achieve speaker independency. This would probably involve frequency-domain manipulations.

As stated before, this thesis represents one more step toward realizing a practical voice recognition system. By implementing some of the suggested improvements above and possibly many others, just such a system should become a reality in the near future.

6.2 Concluding Remarks

The subject of machine recognition of human speech has been investigated for a long time. Thus far, there is really no practical system. This thesis provides some insights into practical aspects of building such a system.

Although this system is only capable of recognizing isolated words spoken by the person who trained the system, it is the first step to realizing a more sophisticated system. The actual hardware of this system is relatively simple, thanks to the use of microprocessor-based input system. With increasing availability of microprocessors and single-chip digital signal processors, there is no doubt in this author's mind that a practical voice recognition system is not far in the future.

APPENDIX A

SCHEMATIC DIAGRAMS

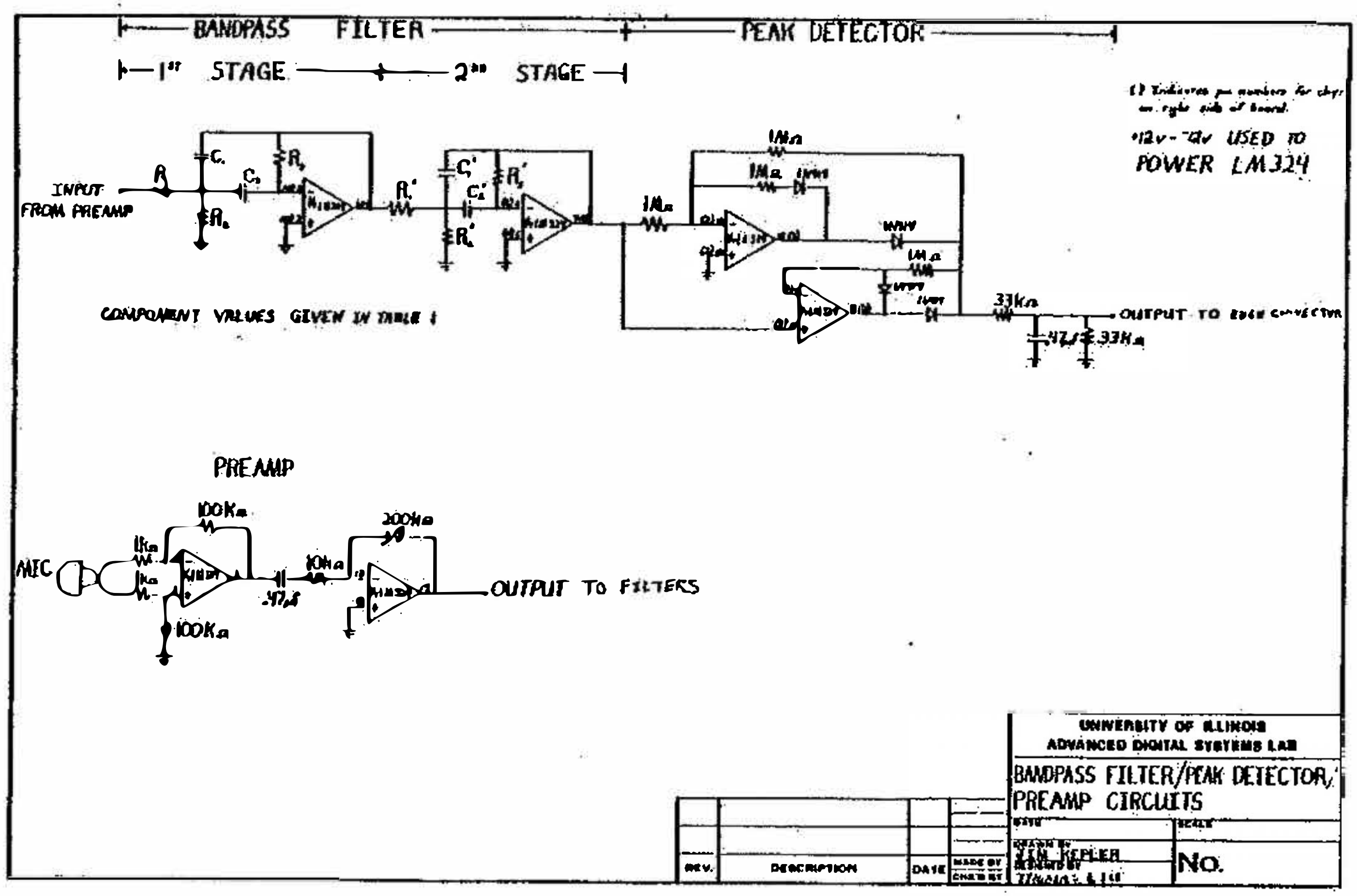


Figure A.1 - Analog board.

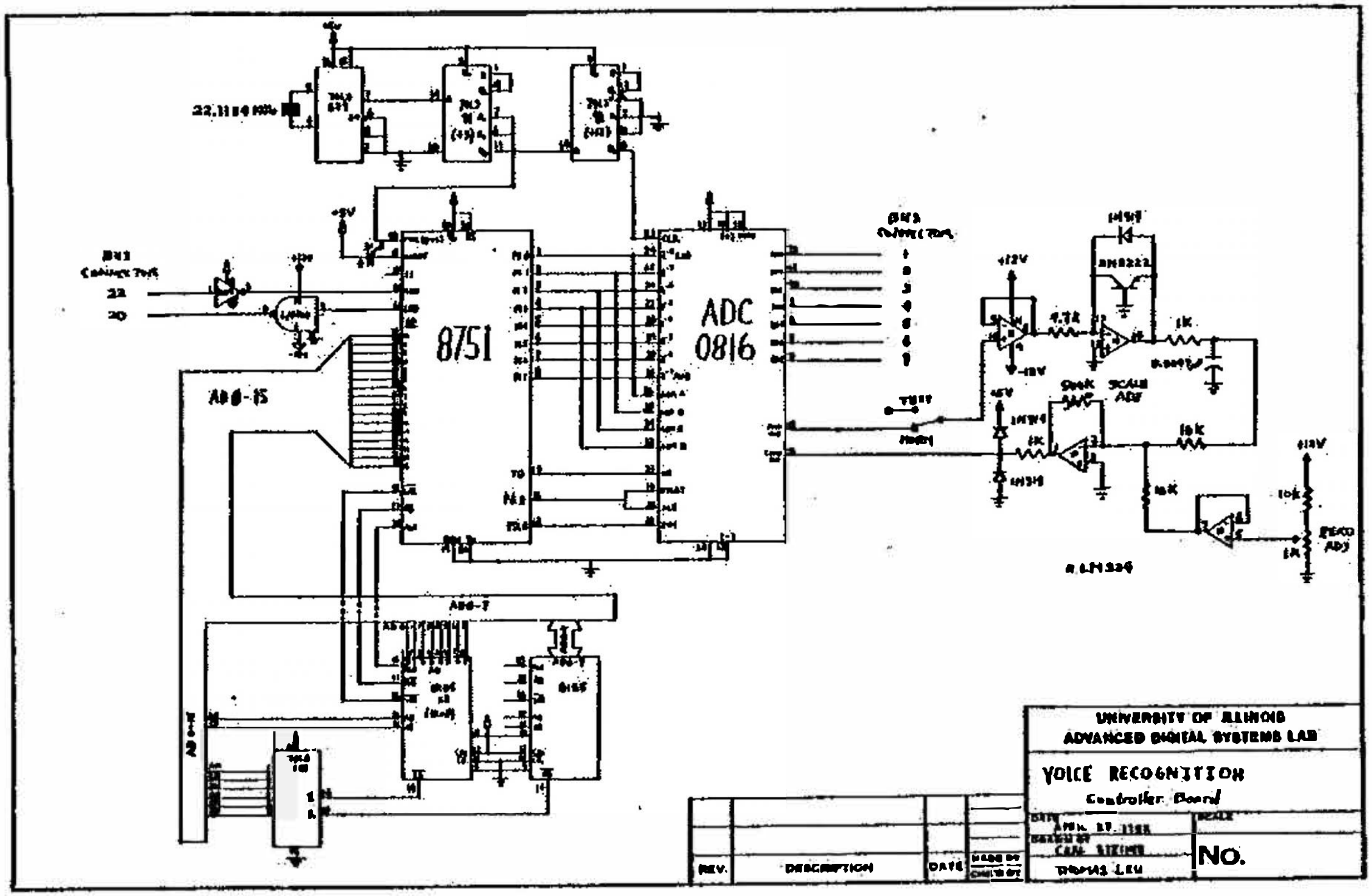


Figure A.2 - Controller board.

TABLE A.1 - BUS DEFINITION.

Connector Number	Controller Board	Analog Board
1	Channel 0 -	62 Hz
2	Channel 1 -	125 Hz
3	Channel 2 -	250 Hz
4	Channel 3 -	500 Hz
5	Channel 4 -	1000 Hz
6	Channel 5 -	2000 Hz
7	Channel 6 -	4000 Hz
9	-	Mic 1
10	-	Mic 2
19	-12 V	-12 V
20	Serial out	-
21	+12 V	+12 V
22	Serial in	-
A	+5 V	+5 V
Z	GND	GND

Note: All other connections unused.

APPENDIX B

LISTEN PROGRAM LISTING

ISIS-II MCS-51 MACRO ASSEMBLER V2.0
NO OBJECT MODULE REQUESTED
ASSEMBLER INVOKED BY: ASM51 LISTEN.SRC NOOBJECT NOPAGING PRINT(:TO:)

```
LOC  OBJ            LINE     SOURCE

                         ;
                         ;
                         ; *****************************************************************
                         ; *********************                           *********************
                         ; *********************  Voice Recognition System  *********************
                         ; *********************                           *********************
                         ; *********************          LISTEN           *********************
                         ; *********************                           *********************
                         ; *****************************************************************
                         ;
                         ;
                         ;----------------------------------------------------------------------
                         ;
                         ;
                         ;       Date:   May 5, 1983
                         ;       Update: July 6, 1983 - Revision 3.21
                         ;
                         ;       Author: Thomas Liu
                         ;
                         ;       Source: LISTEN.SRC
                         ;       Object: LISTEN.OBJ
                         ;
                         ;       Purpose:
                         ;           This program is responsible for scanning the outputs of
                         ;           the filter bank, detecting the boundaries of words, and
                         ;           communicating with the host computer.
                         ;
                         ;       Note:
                         ;           This version of this program is based on and modified from
                         ;           earlier test programs:
                         ;               TEST5.SRC       - Jim Kepler (EE 246 - Spring 1983)
                         ;               VOICE1.SRC      - Carl Steiner (EE 498 - Spring 1983)
                         ;
                         ;
                         ;----------------------------------------------------------------------
                         ;
                         ;
                         ; ***** CONSTANTS *****
                         ;
                         ;
                         ; SPER  =Sampling period value, for loading into timer 0 (2 bytes).
                         ;
                         ; BPF   =Number of bandpass filters.
                         ;
```

```
        ; STLTM =Settling time constant for log amp, use 100 us (1 byte).
        ;
        ; THR   =Threshold value for for detecting boundaries (2 bytes).
        ;
        ; SPCC  =Silence period count constant, in units of sampling period.
        ;
        ; STCMD =Command from host to start looking for a word.
        ;
        ; RQCMD =Command from host to transmit one frame of data.
        ;
        ; NL    =Character to host to indicate end of record.
        ;
        ; EOWM  =End of word marker.
        ;
        ; XMBEG =Beginning of external memory.
        ;
        ; XMEND =End of external memory+1.
        ;
        ; MUXST =Address of the first channel of the multiplexer.
        ;
        ;
        ;------------------------------------------------------------------------
        ;
        ;
        ; ***** EQAUTES *****
        ;
        ;
0C00    SPER    EQU     0DC00H          ; Sampling period for 15 ms rate
0007    BPF     EQU     7               ; Number of bands
0020    STLTM   EQU     20H             ; Settling time for log amp (100 us)
        ;
        ; Parameters for boundary detection
        ;
00C0    THR     EQU     00C0H           ; Threshold level (Max = 255*BPF!)
0008    SPCC    EQU     8               ; Silence period count constant
        ;
        ; Commands from the host
        ;
0021    STCMD   EQU     '!'             ; Start searching for a word
000A    RQCMD   EQU     10              ; Request for next frame of data
        ;
        ; Special characters sent by this program
        ;
000A    NL      EQU     10              ; Marks end of frame or end of word marker
002A    EOWM    EQU     '*'             ; End of word marker to host computer
        ;
        ; Buffer RAM space
        ;
2000    XMBEG   EQU     2000H           ; Start of external memory
2800    XMEND   EQU     27FFH+1         ; End of external memory + 1
        ;
        ; Start address for the multiplexer
        ;
0000    MUXST   EQU     0               ; First channel of MUX
```

```
        ;
        ;
        ;-----------------------------------------------------------------
        ;
        ;
        ; ***** FLAGS *****
        ;
        ;
        ; START =Indicates the beginning of a word has been found.  It is
        ;        initialized to zero, and controled by the subroutine DETECT.
        ;
        ; ST    =Indicates the program is looking for a word.  Initially it is
        ;        zero.  It is set by the interupt routine INTR.  It is cleared
        ;        by the subroutine DETECT.
        ;
        ; HCRDY =Indicates the host computer is ready for a frame of data.
        ;        Initially it is zero.  It is controled by the interupt routine.
        ;
        ; XMT   =Indicates there is data in the buffer ready for transmission.
        ;        Initially it is zero.  It is set by the subroutine PUTIN and
        ;        cleared by the subroutine GETOUT.
        ;
        ; TIF   =Indicates the transmitter is ready to transmit data.
        ;        Initially it is set.  It is cleared by the subroutine
        ;        DATAOUT, and set by the interupt routine INTR.
        ;
        ;
        ;-----------------------------------------------------------------
        ;
        ;
        ; ***** BIT ADDRESS SPACE *****
        ;
        ;
                BSEG
        ;
        ;
0000    START:  DBIT    1               ; Flags a word has been detected
0001    TIF:    DBIT    1               ; Transmitter ready
0002    HCRDY:  DBIT    1               ; Host computer ready to receive
0003    XMT:    DBIT    1               ; Data in buffer awaiting transmission
0004    ST:     DBIT    1               ; Look for start of a word
        ;
        ;
        ;-----------------------------------------------------------------
        ;
        ;
        ; ***** VARIABLES *****
        ;
        ;
        ; TBNEW =Temporary buffer for current samples.
        ;
        ; TBOLD =Temporary buffer for previous samples.
        ;
        ; SUM   =Sum of bandpass filters (2 bytes).
```

```
                ;
                ; COUNT =Used by DETECT to count the number of times SUM<THR
                ;       when in the middle of a word.
                ;
                ;
                ;-----------------------------------------------------------------
                ;
                ;
                ; ***** DATA ADDRESS SPACE *****
                ;
                ;
----                    DSEG    AT 30H          ; Start data addresses at hex 30
                ;
                ;
0030            TBNEW:  DS      BPF             ; Current samples
0037            TBOLD:  DS      BPF             ; Previous samples
003E            SUM:    DS      2               ; Sum of band, (low, high) bytes
0040            COUNT:  DS      1               ; Number of silent frames
0041            STACK:  DS      1               ; Start of stack space
                ;
                ;
                ;-----------------------------------------------------------------
                ;
                ;
                ; ***** REGISTER USAGE (BANK 0) *****
                ;
                ;
                ; DPTR  -Points to the next available space in the buffer.
                ;
                ; R7,R6 -Points to the next character to be pulled out of the buffer
                ;        (R7=high order byte, R6=low order byte).
                ;
                ; R0    -Pointer into temporary buffer.
                ;
                ; R2    -Multiplexer address for the ADC0816.
                ;
                ; R1,R5 -Scratch registers.
                ;
                ;
                ; ***** (BANK 1) *****
                ;
                ;
                ; R0,R1 -Scratch registers.
                ;
                ;
                ;-----------------------------------------------------------------
                ;
                ;
                ; ***** POWER-UP RESET *****
                ;
                ;
----                    CSEG
0000                    ORG     0               ; Power-up reset starts here
0000 020100             JMP     BGN             ; Jump to start of program
```

```
                ;
0100                    ORG     100H
                ;
0100 758141     BGN:    MOV     SP,#STACK       ; Initialize stack pointer
                ;
                ; Configure timers 0 and 1:
                ;       timer 0 -> 16-bit counter
                ;       timer 1 -> 8-bit auto-reload
                ;
0103 758921             MOV     TMOD,#00100001B ; Set up timers
                ;
                ; Set up serial port clock rate
                ;
0106 758DFE             MOV     TH1,#0FEH       ; Baud rate=9600
0109 758BFE             MOV     TL1,#0FEH
010C 758840             MOV     TCON,#40H       ; Start timer 1
                ;
                ; Set up serial port mode (8-bit UART)
                ;
010F 759850             MOV     SCON,#01010000B ; =>mode 1, enable serial reception
                ;
                ; Enable interrupts from serial port only
                ;
0112 75A890             MOV     IE,#90H         ; Enable interrupt from serial port
0115 75B810             MOV     IP,#10H         ; Serial port has high priority
                ;
                ; Initialize buffer pointers
                ;
0118 902000             MOV     DPTR,#XMBEG     ; Put pointer
011B 7F20               MOV     R7,#(HIGH XMBEG); Get pointer (R7=high order)
011D 7E00               MOV     R6,#(LOW XMBEG) ;             (R6=low order)
                ;
                ; Initialize flags
                ;
011F C202               CLR     HCRDY           ; Initialize to host not ready
0121 C203               CLR     XMT             ; Initialize to no data in buffer
0123 C200               CLR     START           ; Initialize to word not yet started
0125 D201               SETB    TIF             ; Initialize to transmitter ready
0127 C204               CLR     ST              ; Initialize to not looking for word
                ;
                ;
                ;-----------------------------------------------------------------
                ;
                ;
                ; ***** MAIN LOOP *****
                ;
                ;
                ; This is the main loop.  At every sampling interval, all 3PF channels
                ; are scanned, placed into temporary buffer, and calculated the sum of
                ; these samples.
                ;
0129 3004FD     LOOP0:  JNB     ST,$            ; Wait untill host computer says GO
                ;
                ; Set up timer for sampling period
```

```
                ;
012C C28C               CLR     TR0             ; Disable timer 0 while it's being reset
012E C28D               CLR     TF0             ; Clear timer 0 overflow
0130 758CDC             MOV     TH0,#(HIGH SPER); Load timer 0 so it overflows after
0133 758A00             MOV     TL0,#(LOW SPER) ; one sampling period
0136 D28C               SETB    TR0             ; Start timer 0
                ;
                ; Initialize for each outer loop
                ;
0138 7830               MOV     R0,#TBNEW       ; Init temporary buffer pointer
013A 753E00             MOV     SUM+0,#0        ; Init sum
013D 753F00             MOV     SUM+1,#0
0140 7A00               MOV     R2,#MUXST       ; Initial MUX address
                ;
                ; Inner loop to read each band
                ;
0142 8A90       LOOP1:  MOV     P1,R2           ; Send MUX ADDRESS
0144 D2B3               SETB    P3.3
0146 7D20               MOV     R5,#STLTM       ; Log amp settling
0148 DDFE               DJNZ    R5,$
014A C2B3               CLR     P3.3            ; Start conversion
014C 7590FF             MOV     P1,#0FFH        ; Port 1 for input
014F D2B2               SETB    P3.2            ; P3.2 is input
0151 30B2FD             JNB     P3.2,$          ; Wait until end of conversion
0154 D2B4               SETB    P3.4            ; Enable output from ADC
0156 E590               MOV     A,P1            ; Read data
0158 C2B4               CLR     P3.4            ; Disable output from ADC
015A F6                 MOV     @R0,A           ; Store into temporary buffer
015B 253E               ADD     A,SUM+0         ; Add to sum
015D F53E               MOV     SUM+0,A
015F E53F               MOV     A,SUM+1
0161 3400               ADDC    A,#0
0163 F53F               MOV     SUM+1,A
0165 0A                 INC     R2              ; Next channel
0166 08                 INC     R0
0167 BA07D8             CJNE    R2,#BPF,LOOP1   ; Until all bands sampled
                ;
                ; Call the detect and output procedure
                ;
016A 3174               ACALL   DETECT          ; Word or no word??
016C 30008A             JNB     START,LOOP0     ; Loop fast until start of word
016F 308DFD             JNB     TF0,$           ; Wait for timer overflow
0172 2129               AJMP    LOOP0
                ;
                ;
                ;------------------------------------------------------------------
                ;
                ;
                ; ***** DETECT *****
                ;
                ;
                ; This routine detects the start and end of a word.  A word has begun
                ; when the total energy exceeds threshold, that is, SUM0 > THR, START
                ; is then set and COUNT is reset.  When START=1, data in temporary buffer
```

```
                ; SRO is normalized, converted to ASCII and then put into the buffer in
                ; external RAM (ACALL SEND).  When START=1 but SUMO < THR, COUNT is
                ; incremented.  COUNT counts the number of consecutive times SUMO < THR.
                ; If COUNT < SPCC, silence paroid count, then the data SRO is normalized
                ; converted and put into the buffer (ACALL SEND).  If COUNT=SPCC, clear START
                ; to indicate end of a word.  The end of word marker is placed into the
                ; buffer (ACALL WORM).
                ;
0174 C0E0       DETECT: PUSH    ACC
                ;
                ; First decide if SUM < THR
                ;
0176 E53F               MOV     A,SUM+1         ; Check high byte first
0178 C3                 CLR     C
0179 9400               SUBB    A,#(HIGH THR)   ; C=1 when SUM < THR
017B 7004               JNZ     DL0             ; If not equal, comparison result known
017D E53E               MOV     A,SUM+0         ; Note C=0 here
017F 94C0               SUBB    A,#(LOW THR)    ; C=1 when SUM < THR
0181 4000       DL0:    JC      DL1             ; Jump if C=1
                ;
                ; Here SUM >= THR
                ;
0183 754000             MOV     COUNT,#00H      ; Initialize count
0186 300002             JNB     START,DL1ST     ; Skip if this is first sample of word
0189 31A8               ACALL   SEND            ; Put data into buffer
018B D200       DL1ST:  SETB    START           ; Indicate start or middle word
018D D0E0               POP     ACC
018F 22                 RET
                ;
                ; Here SUM < THR
                ;
0190 200003     DL1:    JB      START,DL2       ; Jump if START=1
0193 D0E0               POP     ACC
0195 22                 RET                     ; Return since START=0 and SUM < THR
                ;
                ; Here START=1 and SUM < THR
                ;
0196 0540       DL2:    INC     COUNT
0198 7408               MOV     A,#SPCC
019A B54009             CJNE    A,COUNT,DL3     ; COUNT <> SPCC then jump
                ;
                ; Here START=1, SUM < THR, and COUNT=SPCC
                ;
019D C200               CLR     START
019F 31CE               ACALL   WORM            ; Put end of word marker in buffer
01A1 C204               CLR     ST              ; No longer at start of word
01A3 D0E0               POP     ACC
01A5 22                 RET
                ;
                ; Here START=1, SUM < THR, and COUNT < SPCC
                ;
01A6 31A8       DL3:    ACALL   SEND            ; Put data in buffer
01A8 D0E0               POP     ACC
01AA 22                 RET
```

```
          ;
          ;
          ;------------------------------------------------------------------
          ;
          ;
          ; ***** SEND *****
          ;
          ;
          ; This routine convert to ascii using CONVERT BPF consecutive bytes of data.
          ; The first byte is pointed to by STTRO.  The data is then put into external
          ; RAM by PUTIN.
          ;
01AB C0E0 SEND:  PUSH  ACC
01AD D2D3        SETB  RS0           ; Bank 1
01AF 7830        MOV   R0,#TBNEW     ; Initialize for loop
01B1 7937        MOV   R1,#TBOLD
01B3 7A07        MOV   R2,#BPF       ; Number of bands
          ;
          ; Loop converts and then puts data into buffer
          ;
01B5 E6   NL2:   MOV   A,@R0         ; A has data
01B6 C7          XCH   A,@R1         ; Stores as previous sample
01B7 26          ADD   A,@R0         ; Average the two samples
01B8 13          RRC   A             ; Divides by 2
01B9 3400        ADDC  A,#0          ; Round off
01BB C2D3        CLR   RS0           ; Reset to reg bank 0 for calls
01BD 31F0        ACALL CONVERT       ; Data is now ascii, and in A
01BF D2D3        SETB  RS0           ; Return to reg. bank 1
          ;
          ; Data is put into buffer (ext. RAM)
          ;
01C1 08          INC   R0            ; Inc to next data byte
01C2 09          INC   R1
01C3 DAF0        DJNZ  R2,NL2        ; Loop until all bands
01C5 740A        MOV   A,#NL
01C7 5109        ACALL PUTIN         ; Put a line feed in buff.
01C9 D0E0        POP   ACC
01CB C2D3        CLR   RS0           ; Return to reg. bank 0
01CD 22          RET
          ;
          ;
          ;------------------------------------------------------------------
          ;
          ;
          ; ***** WORM *****
          ;
          ;
          ; This routine puts an end of word marker into the buffer
          ;
01CE C0E0 WORM:  PUSH  ACC
01D0 742A        MOV   A,#EOWM       ; Put in end of word mark
01D2 5109        ACALL PUTIN
01D4 740A        MOV   A,#NL         ; NL character
01D6 5109        ACALL PUTIN
```

```
01D8 D0E0              POP     ACC
01DA 22                RET
                 ;
                 ;
                 ;----------------------------------------------------------------
                 ;
                 ;
                 ; ***** MAYBE *****
                 ;
                 ;
                 ; This routine checks to see if a data can be transmitted
                 ;
01DB 30010A      MAYBE:  JNB     TIF,M1          ; Transmitter ready?
01DE 300207              JNB     HCRDY,M1        ; Host ready?
01E1 300304              JNB     XMT,M1          ; Ddat to transmit?
01E4 5121                ACALL   GETOUT          ; Yes, get data from buffer
01E6 31E9                ACALL   DATAOUT         ; Send it
01E8 22          M1:     RET
                 ;
                 ;
                 ;----------------------------------------------------------------
                 ;
                 ;
                 ; ***** DATAOUT *****
                 ;
                 ;
                 ; Output data from the ACC through serial i/o
                 ;
01E9 C299        DATAOUT:CLR     TI
01EB C201                CLR     TIF
01ED F599                MOV     SBUF,A
01EF 22                  RET
                 ;
                 ;
                 ;----------------------------------------------------------------
                 ;
                 ;
                 ; ***** CONVERT *****
                 ;
                 ;
                 ; Converts data byte in ACC into two ASCII characters and puts them
                 ; into the buffer
                 ;
01F0 C0E0        CONVERT:PUSH    ACC             ; Save byte
01F2 C4                  SWAP    A               ; Shift to right 4 bits
01F3 31FE                ACALL   OUTHEX          ; Output high order hex digit
01F5 D0E0                POP     ACC             ; Get original byte
01F7 C0E0                PUSH    ACC             ; Save it again
01F9 31FE                ACALL   OUTHEX          ; Lowe order digit
01FB D0E0                POP     ACC             ; Get original byte back
01FD 22                  RET
                 ;
                 ;
                 ;----------------------------------------------------------------
```

```
                    ;
                    ;
                    ; ***** OUTHEX *****
                    ;
                    ;
                    ; Convert and send the lower 4 bits as hex digit in ASCII
                    ;
01FE 540F           OUTHEX: ANL     A,#0FH          ; Retain lower 4 bits
0200 2490                   ADD     A,#90H          ; Convert to ASCII hex digit
0202 D4                     DA      A
0203 3440                   ADDC    A,#40H
0205 D4                     DA      A
0206 5109                   ACALL   PUTIN
0208 22                     RET
                    ;
                    ;
                    ;----------------------------------------------------------------
                    ;
                    ;
                    ; ***** PUTIN *****
                    ;
                    ;
                    ; Moves byte in ACC to buffer RAM's, DPTR holds the 16bit address which
                    ; must remain inside our memory limits.
                    ;
0209 F0             PUTIN:  MOVX    @DPTR,A                 ; Move the data into RAM
020A D203                   SETB    XMT                     ; Something is now in buffer
020C A3                     INC     DPTR
020D C0E0                   PUSH    ACC                     ; Save
020F E583                   MOV     A,DPH                   ; Check that DPTR is still within
0211 B42808                 CJNE    A,#(HIGH XMEND),DONEP   ; limits
0214 E582                   MOV     A,DPL
0216 B40003                 CJNE    A,#(LOW XMEND),DONEP
0219 902000                 MOV     DPTR,#XMBEG             ; Reset DPTR if outside limits
021C D0E0           DONEP:  POP     ACC                     ; Restore
021E 31DB                   ACALL   MAYBE                   ; See if can send something
0220 22                     RET
                    ;
                    ;
                    ;----------------------------------------------------------------
                    ;
                    ;
                    ; ***** GETOUT *****
                    ;
                    ;
                    ; Transfers one data byte from the buffer to the ACC.  Uses R7, R6 as
                    ; pointer to the data byte
                    ;
0221 C083           GETOUT: PUSH    DPH
0223 C082                   PUSH    DPL                     ; Save DPTR
0225 8F83                   MOV     DPH,R7                  ; Set up DPTR for data transfer
0227 8E82                   MOV     DPL,R6
0229 E0                     MOVX    A,@DPTR                 ; Get data from RAM
022A A3                     INC     DPTR
```

```
022B AF83              MOV     R7,DPH                   ; Load incremented pointer back
022D AE82              MOV     R6,DPL                   ; into R7, R6
022F D082              POP     DPL
0231 D083              POP     DPH                      ; Restore
0233 BF2807            CJNE    R7,#(HIGH XMEND),DONEB   ; Check that R7, R6 are within
0236 BE0004            CJNE    R6,#(LOW XMEND),DONEB    ; limits
0239 7F20              MOV     R7,#(HIGH XMBEG)         ; Else must reset
023B 7E00              MOV     R6,#(LOW XMBEG)
023D C0E0      DONEB:  PUSH    ACC                      ; Save data
023F EF                MOV     A,R7                     ; See if any more data
0240 B58306            CJNE    A,DPH,CONT1
0243 EE                MOV     A,R6
0244 B58202            CJNE    A,DPL,CONT1
0247 C203              CLR     XMT
0249 D0E0      CONT1:  POP     ACC                      ; Restore data byte
024B 22                RET
               ;
               ;
               ;--------------------------------------------------------------------------------
               ;7/6/83-12:24-tcl
               ;
               ;
                       END
```

SYMBOL TABLE LISTING
------ ----- -------

```
N A M E     T Y P E   V A L U E     A T T R I B U T E S

ACC. . . .  D ADDR    00E0H    A
BGN. . . .  C ADDR    0100H    A
BPF. . . .    NUMB    0007H    A
CONT1. . .  C ADDR    0249H    A
CONVERT. .  C ADDR    01F0H    A
COUNT. . .  D ADDR    0040H    A
DATAOUT. .  C ADDR    01E9H    A
DETECT . .  C ADDR    0174H    A
DL0. . . .  C ADDR    0151H    A
DL1. . . .  C ADDR    0190H    A
DLIST. . .  C ADDR    016BH    A
DL2. . . .  C ADDR    0196H    A
DL3. . . .  C ADDR    01A6H    A
DONEG. . .  C ADDR    023DH    A
DONEP. . .  C ADDR    021CH    A
DPH. . . .  D ADDR    0083H    A
DPL. . . .  D ADDR    0082H    A
EOWM . . .    NUMB    002AH    A
GETOUT . .  C ADDR    0221H    A
HCRDY. . .  B ADDR    0020H.2  A
I1 . . . .  C ADDR    003EH    A
I2 . . . .  C ADDR    0037H    A
I4 . . . .  C ADDR    0051H    A
IE . . . .  D ADDR    00A8H    A
INTR . . .  C ADDR    0023H    A
IP . . . .  D ADDR    00B8H    A
LOOP0. . .  C ADDR    0129H    A
LOOP1. . .  C ADDR    0142H    A
M1 . . . .  C ADDR    01E3H    A
MAYBE. . .  C ADDR    010BH    A
MUXST. . .    NUMB    0000H    A
NL . . . .    NUMB    000AH    A
NL2. . . .  C ADDR    01B5H    A
OUTHEX . .  C ADDR    01FEH    A
P1 . . . .  D ADDR    0090H    A
P3 . . . .  D ADDR    00B0H    A
PSW. . . .  D ADDR    00D0H    A
PUTIN. . .  C ADDR    0209H    A
RI . . . .  B ADDR    0098H.0  A
RQCMD. . .    NUMB    000AH    A
RS0. . . .  B ADDR    00D0H.3  A
SBUF . . .  D ADDR    0099H    A
SCON . . .  D ADDR    0098H    A
SEND . . .  C ADDR    01ABH    A
SP . . . .  D ADDR    0081H    A
SPCC . . .    NUMB    0008H    A
SPER . . .    NUMB    0C00H    A
ST . . . .  B ADDR    0020H.4  A
```

```
STACK. . .  D ADDR   0041H   A
START. . .  B ADDR   0020H.0 A
STCMD. . .    NUMB   0021H   A
STLTM. . .    NUMB   0020H   A
SUM. . . .  D ADDR   003EH   A
TBNEW. . .  D ADDR   0030H   A
TBOLD. . .  D ADDR   0037H   A
TCON . . .  D ADDR   0088H   A
TF0. . . .  B ADDR   0088H.5 A
TH0. . . .  D ADDR   008CH   A
TH1. . . .  D ADDR   008DH   A
THR. . . .    NUMB   00C0H   A
TI . . . .  B ADDR   0098H.1 A
TIF. . . .  B ADDR   0020H.1 A
TL0. . . .  D ADDR   008AH   A
TL1. . . .  D ADDR   008BH   A
TMOD . . .  D ADDR   0089H   A
TR0. . . .  B ADDR   0088H.4 A
WORM . . .  C ADDR   01CEH   A
XMBEG. . .    NUMB   2000H   A
XMEND. . .    NUMB   2800H   A
XMT. . . .  B ADDR   0020H.3 A


REGISTER BANK(S) USED: 0, TARGET MACHINE(S): 8051

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

APPENDIX C

TRAINING PROGRAM LISTING

```

{ $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ }
{ $$$$$$$$$$$$$$$$$$$$$$                                   $$$$$$$$$$$$$$$$$$$$$$ }
{ $$$$$$$$$$$$$$$$$$$$$$  Voice Recognition System  $$$$$$$$$$$$$$$$$$$$$$ }
{ $$$$$$$$$$$$$$$$$$$$$$                                   $$$$$$$$$$$$$$$$$$$$$$ }
{ $$$$$$$$$$$$$$$$$$$$$$          TRAINING                 $$$$$$$$$$$$$$$$$$$$$$ }
{ $$$$$$$$$$$$$$$$$$$$$$                                   $$$$$$$$$$$$$$$$$$$$$$ }
{ $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ }

PROGRAM TRAINING ( INPUT, OUTPUT );

{ Date: April 22, 1983 }

{ Updated: July 6, 1983 - Revision 2.12 }

{ Author: Thomas Liu }

{ This is the TRAINING phase of the Voice Recognition System. }

{ ------------------------------------------------------------------------- }

CONST   VOCAB_FILE = 'VOCAB';        { File of prototypes for vocabulary }
        IN_PORT_FILE = '@TTI1';      { Input device from F-E box }
        OUT_PORT_FILE = '@TTO1';     { Output device to F-E box }
        END_CMD = '$<12>';           { End of input command, including NL }
        END_WORD = '$';              { End of word marker }
        FE_START = '!';              { Start command to F-E box }
        FE_RESET = '$';              { Reset command to F-E box }
        MAX_SAMPLE = 60;             { Maximum number of samples per word }
        NL = '<12>';                 { NL character }

TYPE    LINE_TYPE = ARRAY [1..20] OF CHAR;
                                     { Input line buffer type }

VAR     VOCAB : TEXT;                { Vocabulary file }
        IN_PORT : TEXT;              { Input data from F-E box }
        OUT_PORT : TEXT;             { Output control to F-E box }
        WORD : STRING 80;            { Input word or end command }
        SAMPLE : ARRAY [1..MAX_SAMPLE] OF LINE_TYPE;
                                     { Data for each sample }
        DUMMY : LINE_TYPE;           { Used when sample size too large }
        TOO_LARGE : BOOLEAN;         { Indicates sample size too long }
        N : INTEGER;                 { Number of samples in this word }
        I, J : INTEGER;              { Loop count }
        ACTIVE : BOOLEAN;            { Indicates activity from F-E box }
        CH : CHAR;                   { Answer to yes/no question }
        MIN_SAMPLE : INTEGER;        { Minimum number of samples per word }
        SPC : INTEGER;               { Silence period count from FE box }

{ ------------------------------------------------------------------------- }

PROCEDURE INITIALIZE;
```

```
  { Initialization procedure }
  BEGIN
    RESET(IN_PORT,IN_PORT_FILE);          { Reset input port }
    REWRITE(OUT_PORT,OUT_PORT_FILE);      { Reset output port }
    FILEAPPEND(VOCAB,VOCAB_FILE);         { Append to vocab file }
    WRITE(OUTPUT,'Enter minimum sample count: ');
    READLN(INPUT,MIN_SAMPLE);
    WRITE(OUTPUT,'Enter silence period count: ');
    READLN(INPUT,SPC)
  END;

{ ---------------------------------------------------------------- }

PROCEDURE READLINE ( VAR F : TEXT; VAR LINE : LINE_TYPE );
{ Reads in one line into array of char, terminating with <NL> }
{ Null characters are ignored, since for some reason, the MPT/100 will  }
{   receive them even though they weren't transmitted, or weren't they? }
  CONST NULL = '(0)';             { Null character }
  VAR I : INTEGER;                { Index into array }
      E_O_L : BOOLEAN;            { Indicates NL received }
  BEGIN
    I := 0;                       { Init }
    REPEAT
      I := I + 1;                 { Keep tracks of chars }
      READ(F,LINE[I]);            { Get one char }
      E_O_L := (LINE[I]=NL);      { Check for NL character }
      IF LINE[I] = NULL           { Ignore null chars }
        THEN I := I - 1
    UNTIL E_O_L                   { To end of line }
  END;

{ ---------------------------------------------------------------- }

BEGIN  { Main program }
  WRITELN(OUTPUT,'Training session started...');
  INITIALIZE;                             { Initialize this program }
  REPEAT                                  { Command loop }
    WRITE(OUTPUT,'Input word ("*" to exit): ');
    READLN(INPUT,WORD);                   { Get input word }
    IF WORD <> END_CMD THEN               { Only if not end of command }
      BEGIN
        N := 0;                           { Initialize count }
        WRITELN(OUTPUT,'Say the word typed above into the microphone.');
        WRITE(OUT_PORT,FE_START);         { Tell F-E box to start sampling }
        TOO_LARGE := FALSE;               { Init flags }
        ACTIVE := FALSE;
        DUMMY[1] := ' ';                  { Init dummy }
        REPEAT
          WRITELN(OUT_PORT);              { Send a <NL> to signify ready }
          IF N < MAX_SAMPLE THEN          { Checks sample size }
            BEGIN
              N := N + 1;                 { Keep track of count }
              READLINE(IN_PORT,SAMPLE[N]);{ Input sample from F-E box }
              IF NOT ACTIVE THEN          { Only when activity begins }
```

```
                  BEGIN
                    WRITELN(OUTPUT,'Input active!');
                    ACTIVE := TRUE
                  END
              END
            ELSE
              BEGIN
                READLINE(IN_PORT,DUMMY);  { Throw away this reading }
                TOO_LARGE := TRUE         { Set flag }
              END
          UNTIL (SAMPLE[N,1]=END_WORD) OR { Until end of word prototype }
                (DUMMY[1]=END_WORD);
          IF N < MIN_SAMPLE THEN          { Too few samples }
            WRITELN(OUTPUT,'Word too short!  Input ignored.')
          ELSE IF TOO_LARGE THEN          { Too many samples }
            WRITELN(OUTPUT,'Word too long!  Input ignored.')
          ELSE
            BEGIN
              WRITE(OUTPUT,'Store this prototype? ');
              READLN(INPUT,CH);           { Get answer }
              IF (CH='Y') OR (CH='y')
                THEN
                  BEGIN
                    WRITE(OUTPUT,'Storing ',WORD);
                    WRITE(VOCAB,WORD);            { Store into vocabulary }
                    FOR I := 1 TO N-SPC-1 DO      { Store the prototype }
                      BEGIN
                        J := 0;
                        REPEAT
                          J := J + 1;
                          WRITE(VOCAB,SAMPLE[I,J])
                        UNTIL SAMPLE[I,J]=NL
                      END;
                    WRITELN(VOCAB,END_WORD)       { End-of-word marker }
                  END
                ELSE WRITELN(OUTPUT,'Input ignored!')
            END
        END
    UNTIL WORD = END_CMD;                 { Until end command detected }
    WRITELN(OUTPUT,'Training ended.')
  END { 7/6/83-16:27-tcl }.
```

147 source lines were compiled in 2 minutes 30 seconds

Program area	Size in words
Program code	471
Program literals	203
Global initialized variables	422
Global non-initialized variables	618

No Compilation Errors

APPENDIX D

RECOGN PROGRAM LISTING

```

{ ******************************************************************** }
{ *********************                            ******************** }
{ *********************  Voice Recognition System  ******************** }
{ *********************                            ******************** }
{ *********************          RECOGN            ******************** }
{ *********************                            ******************** }
{ ******************************************************************** }

PROGRAM RECOGN ( INPUT, OUTPUT );

{ Date: April 28, 1983 }

{ Update: July 6, 1983 - Revision 6.01 }

{ Author: Thomas Liu   }

{ This is the recognition program of the Voice Recognition System. }

{ ------------------------------------------------------------------ }

CONST   INF_DIST = 32767;          { Largest value, this is infinity! }
        MAX_SAMPLE = 60;           { Maximum samples per word }
        N_BPF = 7;                 { Number of bands }
        MAX_VOCAB = 10;            { Maximum vocabulary size }
        VOCAB_FILE = 'VOCAB';      { Vocabulary file name }
        IN_PORT_FILE='@TTI1';      { Input port name }
        OUT_PORT_FILE='@TTO1';     { Output port name }
        FE_START = '!';            { Starts F-E conversion }
        END_WORD = '*';            { End of word marker }
        NL = '<12>';               { New line character }

TYPE    DISTANCE = INTEGER;        { distance measurement }
        TIME_SAMPLE = ARRAY [1..N_BPF] OF INTEGER;
                                   { Each time sample }
        SAMPLE = ARRAY [1..MAX_SAMPLE] OF TIME_SAMPLE;
                                   { Sampled values }
        LINE_TYPE = ARRAY [1..20] OF CHAR;
                                   { Input line buffer type }

VAR     VOCAB : TEXT;              { Vocabulary file }
        IN_PORT : TEXT;            { Input port }
        OUT_PORT : TEXT;           { Output port }
        N_WORDS : INTEGER;         { Number of words in vocabulary }
        WORDS : ARRAY [1..MAX_VOCAB] OF
          RECORD                   { Representation for each word }
            N   : INTEGER;         { Number of samples }
            TXT : STRING 20;       { Text string of this word }
            DAT : SAMPLE           { Prototype of vocabulary }
          END;
        CMD_CH : CHAR;             { Command character }
        RECOGN_THR : DISTANCE;     { Maximum score for recognition }
```

```
          RECOGN_DIF : DISTANCE;  { Minimum difference to next best }
          M_ADJ : INTEGER;        { Maximum boundary adjustment }
          MIN_SAMPLE : INTEGER;   { Minimum number of samples per word }
          SPC : INTEGER;          { Silence period count from FE box }
          NORM_FLAG : BOOLEAN;    { Indicates normalization desired }

{ ------------------------------------------------------------------ }

PROCEDURE READLINE ( VAR F : TEXT; VAR LINE : LINE_TYPE );
{ Reads in one line into array of char, terminating with <NL> }
{ Null characters are ignored, since for some reason, the MPT/100 will }
{   receive them even though they weren't transmitted, or weren't they? }
  CONST NULL = '<0>';             { Null character }
  VAR I : INTEGER;                { Index into array }
      E_O_L : BOOLEAN;            { Indicates NL received }
  BEGIN
    I := 0;                       { Init }
    REPEAT
      I := I + 1;                 { Keep tracks of chars }
      READ(F,LINE[I]);            { Get one char }
      E_O_L := (LINE[I]=NL);      { Check for NL character }
      IF LINE[I] = NULL           { Ignore null chars }
        THEN I := I - 1
    UNTIL E_O_L                   { To end of line }
  END;

{ ------------------------------------------------------------------ }

PROCEDURE CONVERT ( LINE : LINE_TYPE; VAR ARY : SAMPLE; S : INTEGER );
{ Convert the input line into internal representation }
  VAR I : INTEGER;                { Loop variable }
  FUNCTION HEX ( CH : CHAR ) : INTEGER;
    { Convert hex digit in ASCII to integer }
      BEGIN
        IF (CH>='0') AND (CH<='9')
          THEN HEX := ORD(CH) - ORD('0')          { 0..9 }
          ELSE HEX := ORD(CH) - ORD('A') + 10;    { A..F }
      END;
  BEGIN
    FOR I := 1 TO N_BPF DO               { For each band }
      ARY[S,I] := HEX(LINE[I+I-1]) * 16 + HEX(LINE[I+I])
  END;

{ ------------------------------------------------------------------ }

PROCEDURE INITIALIZE;
{ Reset I/O ports and read in prototypes from vocabulary file }

  VAR I : INTEGER;                { Counting samples per word }
      LINE : LINE_TYPE;           { Each line representing samples }

  BEGIN
    RESET(IN_PORT,IN_PORT_FILE);
    REWRITE(OUT_PORT,OUT_PORT_FILE);
```

```
    RESET(VOCAB,VOCAB_FILE);
    RECOGN_THR := 35;                   { Default threshold }
    RECOGN_DIF := 5;                    { Default differentiation factor }
    M_ADJ := 1;                         { Default boundary adjustment }
    MIN_SAMPLE := 15;                   { Default minimum sample per word }
    NORM_FLAG := TRUE;                  { Default is to normalize }
    N_WORDS := 0;                       { Initialize word count }
    WRITE(OUTPUT,'Enter silence period count: ');
    READLN(INPUT,SPC);
    WRITELN(OUTPUT,'Reading in vocabulary.  Please wait...');
    REPEAT                              { Loop to read in vocabulary }
      N_WORDS := N_WORDS + 1;           { Keeps tracks of number of words }
      READLN(VOCAB,WORDS[N_WORDS].TXT); { Read in one word }
      WRITE(OUTPUT,WORDS[N_WORDS].TXT); { debug }
      I := 0;                           { Initialize sample count }
      REPEAT                            { Loop for each sample }
        READLINE(VOCAB,LINE);           { Read in each set of samples }
        IF LINE[1] <> END_WORD THEN     { Check for end of word }
          BEGIN
            I := I + 1;                 { Sample count }
            CONVERT(LINE,WORDS[N_WORDS].DAT,I)
                                        { Change to internal representation }
          END
      UNTIL LINE[1] = END_WORD;         { Until end of word }
      WORDS[N_WORDS].N := I             { Read in the word and store size }
    UNTIL EOF(VOCAB)                    { Until end of all vocab words }
  END;

{ ------------------------------------------------------------------------ }

FUNCTION TOTAL_DIST ( ARY1 : SAMPLE; A : INTEGER;
                      ARY2 : SAMPLE; B : INTEGER ) : DISTANCE;
{ procedure to calculate the total distance                }
{ NOTE: A shoud be greater than or equal to B.             }
{       Some samples from the smaller array are duplicated. }

  TYPE  SUM_OF_BANDS = ARRAY [1..MAX_SAMPLE] OF INTEGER;
                             { Sum of bands for each time sample }
        NON_ZERO_BANDS = ARRAY [1..MAX_SAMPLE] OF INTEGER;
                             { Counts number of non-zero bands }

  VAR   VAL : DISTANCE;      { holds distance calculation result }
        X, Y : INTEGER;      { array indices }
        S : REAL;            { slope of band limits }
        D : ARRAY [1..MAX_SAMPLE,1..MAX_SAMPLE] OF DISTANCE;
                             { holds total distance thus far }
        Y_MIN, Y_MAX : ARRAY [1..MAX_SAMPLE] OF DISTANCE;
                             { Bounds of y as function of x }
        SUM1, SUM2 : SUM_OF_BANDS;
        NZ1, NZ2 : NON_ZERO_BANDS;

        { -------------------------------- }

  PROCEDURE SUMMARIZE ( SMPL : SAMPLE;
```

```
                          N : INTEGER;
                          VAR SUM : SUM_OF_BANDS;
                          VAR NZ : NON_ZERO_BANDS );
{ Sum all bands of each sample, also count number of nonzero bands }
  VAR I, J : INTEGER;          { Loop variables }
  BEGIN
    FOR I := 1 TO N DO         { For each sample }
      BEGIN
        SUM[I] := 0;           { Initialize }
        NZ[I] := N_BPF;
        FOR J := 1 TO N_BPF DO
          IF SMPL[I,J]=0
            THEN NZ[I] := NZ[I]-1
            ELSE SUM[I] := SUM[I]+SMPL[I,J]
      END
  END;

      { ------------------------------------ }

PROCEDURE NORMALIZE ( VAR SMPL : TIME_SAMPLE; SUM, NZ : INTEGER );
{ This algorithm is an equal-sum normalization. }
{ Normalize each time sample so the sum is approximatlely equals to SUM }
  VAR I : INTEGER;              { Index into time sample }
      ADJ : INTEGER;            { Reduction factor }
      S : INTEGER;              { Holds value of SUM passed }
      N : INTEGER;              { Holds value of NZ passed }
  BEGIN
    S := SUM;                   { So S can be changed }
    N := NZ;                    { So N can be modified }
    IF N>0 THEN                 { Only for non-zero N }
      ADJ := S DIV N;           { Calculate reduction factor }
    WHILE (ADJ>0)AND(N>0) DO    { Only if factor is non-zero }
      BEGIN
        FOR I := 1 TO N_BPF DO
          IF SMPL[I]>0 THEN     { Only if level in this band is non-zero }
            IF SMPL[I]>ADJ      { Want to leave result non-negative and    }
              THEN              {    reducing from S only actual reduction }
                BEGIN
                  SMPL[I] := SMPL[I]-ADJ;
                  S := S-ADJ
                END
              ELSE
                BEGIN
                  S := S-SMPL[I];
                  SMPL[I] := 0;
                  N := N-1
                END;
        IF N>0 THEN ADJ := S DIV N
      END
  END;

      { ------------------------------------ }

FUNCTION DIST ( I1, I2 : INTEGER ) : DISTANCE;
```

```
{ Calculates the distance between two points indexed by I1 and I2 }
{ New method: return the maximum of absolute differences. }
  VAR VAL : DISTANCE;         { Holds distance value so far }
      DIF : DISTANCE;         { Holds the distance calculated }
      I : INTEGER;            { Loop variable, index into bands }
      T1, T2 : TIME_SAMPLE;   { Samples at this instant in time }
  BEGIN
    FOR I := 1 TO N_BPF DO    { Copy time samples from both arrays }
      BEGIN
        T1[I] := ARY1[I1,I];
        T2[I] := ARY2[I2,I]
      END;
    DIF := SUM1[I1]-SUM2[I2]; { Difference of sum }
    IF NORM_FLAG THEN         { Only if normalization desired }
      IF DIF>0                { Normalize the larger }
        THEN NORMALIZE(T1,DIF,NZ1[I1])
        ELSE NORMALIZE(T2,-DIF,NZ2[I2]);
    VAL := 0;                 { Init }
    FOR I := 1 TO N_BPF DO    { For each band }
      BEGIN
        DIF := ABS(T1[I]-T2[I]);
        IF DIF > VAL THEN VAL := DIF
      END;
    DIST := VAL               { Return the result }
  END;

      { ------------------------------- }

FUNCTION GET_D ( X, Y : INTEGER ) : DISTANCE;
{ Controlled fetch from the array D }
  BEGIN
    IF (Y<=Y_MAX[X]) AND (Y>=Y_MIN[X])
      THEN GET_D := D[X,Y]
      ELSE GET_D := INF_DIST
  END;

      { ------------------------------- }

FUNCTION DP_FCN ( X, Y : INTEGER ) : DISTANCE;
{ This is the dynamic programming function }
  VAR VAL : DISTANCE;         { Holds the minimum of function }
      T : DISTANCE;           { Temporarily holds distance }
  BEGIN
    VAL := GET_D(X-1,Y);      { Find the minimum }
    T := GET_D(X-1,Y-1);
    IF T < VAL THEN VAL := T;
    T := GET_D(X-1,Y-2);
    IF T < VAL THEN VAL := T;
    IF VAL < INF_DIST
      THEN DP_FCN := DIST(X,Y) + VAL
      ELSE DP_FCN := INF_DIST
  END;

      { ------------------------------- }
```

```

  BEGIN  { TOTAL_DIST }
  { use band dynamic programming normalization }
    SUMMARIZE(ARY1,A,SUM1,NZ1);          { First find sums and non-zeros }
    SUMMARIZE(ARY2,B,SUM2,NZ2);
  { Calculate slope }
    S := FLOAT(B-M_ADJ-1)/FLOAT(A-M_ADJ-1);
  { Calculate limits on y-axis }
    FOR X := 1 TO M_ADJ+1 DO
      BEGIN
        Y_MIN[X] := 1;
        Y_MAX[X] := ROUND(S*FLOAT(X-1))+1+M_ADJ
      END;
    FOR X := M_ADJ+2 TO A-M_ADJ-1 DO
      BEGIN
        Y_MIN[X] := ROUND(S*FLOAT(X-M_ADJ-1))+1;
        Y_MAX[X] := ROUND(S*FLOAT(X-1))+1+M_ADJ
      END;
    FOR X := A-M_ADJ TO A DO
      BEGIN
        Y_MIN[X] := ROUND(S*FLOAT(X-M_ADJ-1))+1;
        Y_MAX[X] := B
      END;
  { Calculate initial points }
    FOR X := 1 TO M_ADJ+1 DO
      D[X,1] := DIST(X,1);
    FOR Y := 2 TO M_ADJ+1 DO
      D[1,Y] := DIST(1,Y);
  { Main calculations }
    FOR X := 2 TO A DO
      FOR Y := Y_MIN[X] TO Y_MAX[X] DO
        D[X,Y] := DP_FCN(X,Y);
  { Find minimum }
    VAL := D[A,B];
    FOR X := A-M_ADJ TO A-1 DO
      IF VAL > D[X,B] THEN VAL := D[X,B];
    FOR Y := B-M_ADJ TO B-1 DO
      IF VAL > D[A,Y] THEN VAL := D[A,Y];
    IF VAL < INF_DIST                    { Check for "infinity" }
      THEN TOTAL_DIST := (VAL + A DIV 2) DIV A
      ELSE TOTAL_DIST := INF_DIST
  END;

{ ---------------------------------------------------------------- }

PROCEDURE GO_RECOGN;
{ The actual recognition routine }

  VAR TEST : SAMPLE;                     { Test word array }
      N : INTEGER;                       { Number of samples in test word }
      LINE : LINE_TYPE;                  { Input line buffer }
      BEST : INTEGER;                    { Index of best score found }
      SECOND : INTEGER;                  { Index of second best score }
      I : INTEGER;                       { Index into vocabulary }
```

```
      TOO_LARGE : BOOLEAN;                { Indicates if too many samples }
      ACTIVE : BOOLEAN;                   { Indicates active input }
      DIST : ARRAY[1..MAX_VOCAB] OF DISTANCE;   { Temporary stores distance }

        { ------------------------------------ }

  FUNCTION SCORE ( I : INTEGER ) : DISTANCE;
  { Computes the score for each word in the vocabulary }
    VAR VAL : DISTANCE;          { Holds value }
    BEGIN
      IF WORDS[I].N < N          { Check the longer axis }
        THEN VAL := TOTAL_DIST(TEST,N,WORDS[I].DAT,WORDS[I].N)
        ELSE VAL := TOTAL_DIST(WORDS[I].DAT,WORDS[I].N,TEST,N);
      WRITE(OUTPUT,'Word: ',WORDS[I].TXT);       { debug }
      WRITELN(OUTPUT,'  Distance: ',VAL);        { debug }
      SCORE := VAL
    END;

        { ------------------------------------ }

  BEGIN  { GO_RECOGN }
    WRITELN(OUTPUT,'Say the test word.');
    WRITE(OUT_PORT,FE_START);              { Tell F-E box to start sampling }
    N := 0;                                { Initialize }
    TOO_LARGE := FALSE;                    { Init }
    ACTIVE := FALSE;                       { Init }
    REPEAT
      WRITELN(OUT_PORT);                   { Send a <NL> to signify ready }
      READLINE(IN_PORT,LINE);              { Get one sample }
      IF NOT ACTIVE THEN                   { Only if no input received yet }
        BEGIN
          WRITELN(OUTPUT,'Input active!');
          ACTIVE := TRUE
        END;
      IF LINE[1] <> END_WORD THEN          { Check for end of word }
        IF N < MAX_SAMPLE THEN             { Make sure not too large }
          BEGIN
            N := N + 1;                    { Keep track of count }
            CONVERT(LINE,TEST,N)           { Convert to internal representation }
          END
        ELSE TOO_LARGE := TRUE             { If too many samples }
    UNTIL (LINE[1]=END_WORD);              { Until end of word prototype }
    IF N < MIN_SAMPLE THEN                 { Too few samples }
      WRITELN(OUTPUT,'Word too short!  Input ignored.')
    ELSE IF TOO_LARGE THEN                 { Too many samples }
      WRITELN(OUTPUT,'Word too long!  Input ignored.')
    ELSE
      BEGIN                                { Try to match }
        WRITELN(OUTPUT,'End-of-word detected.');
        N := N - SPC;                      { Chop off silence period }
        FOR I := 1 TO N_WORDS DO
          DIST[I] := SCORE(I);             { For each reference word }
        IF DIST[1] > DIST[2]               { Init BEST and SECOND }
          THEN
```

```
          BEGIN
            BEST := 2;
            SECOND := 1
          END
        ELSE
          BEGIN
            BEST := 1;
            SECOND := 2
          END;
      FOR I := 3 TO N_WORDS DO          { Assumes N_WORDS > 2 }
        IF DIST[I] < DIST[BEST]
          THEN
            BEGIN                       { New BEST value }
              SECOND := BEST;
              BEST := I
            END
        ELSE IF DIST[I] < DIST[SECOND]
          THEN
            SECOND := I;                { New SECOND value }
      WRITELN(OUTPUT);
      IF DIST[BEST] > RECOGN_THR        { Check for match }
        THEN WRITELN(OUTPUT,'No match.')
        ELSE
          IF DIST[SECOND]-DIST[BEST] < RECOGN_DIF
            THEN WRITELN(OUTPUT,'Ambiguous input!')
            ELSE WRITELN(OUTPUT,'Word matched is: ',WORDS[BEST].TXT,
                         'Score is: ',DIST[BEST])
    END
  END;

{ ------------------------------------------------------------------ }

PROCEDURE LIST_VOCAB;
{ List currently defined vocabulary }

  VAR I : INTEGER;                      { Index into vocabulary }

  BEGIN
    WRITELN(OUTPUT);
    WRITELN(OUTPUT,'Current vocabulary:');
    FOR I := 1 TO N_WORDS DO            { For each word in the vocab... }
      WRITE(OUTPUT,'  ',WORDS[I].TXT);
    WRITELN(OUTPUT)
  END;

{ ------------------------------------------------------------------ }

PROCEDURE DISPLAY_PARAMS;
{ Displays the adjustable parameters }
  BEGIN
    WRITELN(OUTPUT,'(1) Threshold factor:         ',RECOGN_THR);
    WRITELN(OUTPUT,'(2) Differentiation factor: ',RECOGN_DIF);
    WRITELN(OUTPUT,'(3) Boundary adjustment:      ',M_ADJ);
    WRITELN(OUTPUT,'(4) Minimum sample count:     ',MIN_SAMPLE);
```

```
      WRITE(OUTPUT,'(5) Normalization:          ');
      IF NORM_FLAG
        THEN WRITELN(OUTPUT,'Yes')
        ELSE WRITELN(OUTPUT,'No');
      WRITELN(OUTPUT);
    END;

{ ----------------------------------------------------------------------- }

PROCEDURE ADJUST;
{ Adjust threshold value }
  VAR SELECT : INTEGER; { Selects option }
      CH : CHAR;
  BEGIN
    REPEAT
      WRITELN(OUTPUT);
      WRITELN(OUTPUT,'(0) No change');
      DISPLAY_PARAMS;
      WRITE(OUTPUT,'Select parameter to adjust: ');
      READLN(INPUT,SELECT);
      CASE SELECT OF
        0 : ;            { Nothing done here }
        1 : BEGIN
              WRITE(OUTPUT,'New threshold: ');
              READLN(INPUT,RECOGN_THR)
            END;
        2 : BEGIN
              WRITE(OUTPUT,'New differentiation factor: ');
              READLN(INPUT,RECOGN_DIF)
            END;
        3 : BEGIN
              WRITE(OUTPUT,'New boundary adjustment: ');
              READLN(INPUT,M_ADJ)
            END;
        4 : BEGIN
              WRITE(OUTPUT,'New minimum sample count: ');
              READLN(INPUT,MIN_SAMPLE)
            END;
        5 : BEGIN
              WRITE(OUTPUT,'Do you want normalization? ');
              READLN(INPUT,CH);
              NORM_FLAG := (CH='Y') OR (CH='y')
            END;
      OTHERWISE WRITELN(OUTPUT,'What?')
      END
    UNTIL SELECT = 0;            { Look for exit option }
    WRITELN(OUTPUT)
  END;

{ ----------------------------------------------------------------------- }

BEGIN  { Main program }
  WRITELN(OUTPUT,'Recognition phase started...');
  INITIALIZE;                            { initialize }
```

```
    WRITELN(OUTPUT);
    WRITELN(OUTPUT,'Default parameters are:');
    WRITELN(OUTPUT);
    DISPLAY_PARAMS;                       { Displays paramter settings }
    REPEAT                                { Command loop }
      WRITE(OUTPUT,'Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): ');
      READLN(INPUT,CMD_CH);
      CASE CMD_CH OF                      { Check command }
        'R', 'r' :  GO_RECOGN;            { Go try recogninition }
        'L', 'l' :  LIST_VOCAB;           { List vocabulary }
        'A', 'a' :  ADJUST;               { Adjust parameters }
        'Q', 'q' :  WRITELN(OUTPUT,'Ending recognition phase.');
      OTHERWISE WRITELN(OUTPUT,'What?')   { Illegal command }
      END
    UNTIL (CMD_CH='Q') OR (CMD_CH='q')    { Until quitting time }
  END { 7/6/83-16:30-tcl }.
```

```
500 source lines were compiled in 7 minutes 1 second

Program area                        Size in words
--------------------------------------------------------
Program code                          2345
Program literals                      479
Global initialized variables          4710
Global non-initialized variables      8

No Compilation Errors
```

APPENDIX E

OPERATING INSTRUCTIONS

This appendix contains the operating instructions for this voice recognition system. The instructions and sample seesions are for the current versions of the programs:

1. LISTEN revision 3.21
2. TRAINING revision 2.12
3. RECOGN revision 6.01

Also, it is assumed that the user is familar with the operations of the MPT/100 using MP/OS.

The instructions are divided into two parts, for training and recognition sessions.

Training Session

1. If the VOCAB file already exists and new words are not to append, then remove the old vocabulary by

   DELETE VOCAB

2. Invoke the training program by

   XEQ TRAINING

3. Answer the minimum sample length question, typically with 15 (see Table 2).
4. Answer the silence period count question with the value of SPCC defined in LISTEN, currently 8.
5. At each command prompt, enter the text of the vocabulary word.
6. Say the word into the microphone.
7. If the word is too long or too short, TRAINING will ignore; go back to step 5.
8. If the word is of valid length, TRAINING asks for confirmation before storing the prototype pattern. To store it, answer with "y".
9. Repeat steps 5 to 8.

10. When all words are entered, type "*" at command prompt.

Note that TRAINING does not limit the size of the vocabulary, even though there is a limit of 10 for the current version of RECOGN. A sample training session is included in Figure E.1.

Recognition Session

1. Make sure the file VOCAB exists.
2. Commence recognition session by

   XEQ RECOGN

3. Answer the silence period count question as in training session.
4. As VOCAB is being read in, each word is printed.
5. Default parameters are displayed.
6. At the command prompt, commands may be entered in lower or upper case. The available commands are:
   1. R - to run a test input
   2. A - to adjust parameter settings.
   3. L - to list vocabulary
   4. Q - to exit session
7. The commands are self-explanatory.

A sample recognition session is included in Figure E.2.

```
xes training
Training session started...
Enter minimum sample count: 15
Enter silence period count: 2
Input word ("*" to exit): One
Say the word typed above into the microphone.
Input active!
Store this prototype? y
Storing One
Input word ("*" to exit): Two
Say the word typed above into the microphone.
Input active!
Word too short!  Input ignored.
Input word ("*" to exit): Two
Say the word typed above into the microphone.
Input active!
Store this prototype? y
Storing Two
Input word ("*" to exit): Three
Say the word typed above into the microphone.
Input active!
Store this prototype? y
Storing Three
Input word ("*" to exit): don't store this
Say the word typed above into the microphone.
Input active!
Store this prototype? n
Input ignored!
Input word ("*" to exit): too long
Say the word typed above into the microphone.
Input active!
Word too long!  Input ignored.
Input word ("*" to exit): *
Training ended.
```

Figure E.1 - Sample training session.

```
xeq recogn
Recognition phase started...
Enter silence period count: 8
Reading in vocabulary.  Please wait...
One
Two
Three

Default parameters are:

(1) Threshold factor:          35
(2) Differentiation factor: 5
(3) Boundary adjustment:     1
(4) Minimum sample count:    15
(5) Normalization:           Yes

Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): r
Say the test word.
Input active!
End-of-word detected.
Word: One
  Distance: 68
Word: Two
  Distance: 64
Word: Three
  Distance: 29

Word matched is: Three
Score is: 29
Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): r
Say the test word.
Input active!
End-of-word detected.
Word: One
  Distance: 33
Word: Two
  Distance: 58
Word: Three
  Distance: 54

Word matched is: One
Score is: 33
Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): r
Say the test word.
Input active!
End-of-word detected.
Word: One
  Distance: 44
Word: Two
  Distance: 50
Word: Three
  Distance: 72

No match.
```

Figure E.2 - Sample recognition session.

```
Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): r
Say the test word.
Input active!
End-of-word detected.
Word: One
  Distance: 64
Word: Two
  Distance: 66
Word: Three
  Distance: 75

No match.
Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): r
Say the test word.
Input active!
Word too short!  Input ignored.
Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): r
Say the test word.
Input active!
Word too long!  Input ignored.
Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): r
Say the test word.
Input active!
End-of-word detected.
Word: One
  Distance: 52
Word: Two
  Distance: 59
Word: Three
  Distance: 62

No match.
Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): l
Current vocabulary:
  One
  Two
  Three

Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): not a
What?
Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): a

(0) No change
(1) Threshold factor:          35
(2) Differentiation factor:  5
(3) Boundary adjustment:       1
(4) Minimum sample count:      15
(5) Normalization:             Yes

Select parameter to adjust: 2
New differentiation factor: 100

(0) No change
(1) Threshold factor:          35
```

Figure E.2 (continued).

```
(2) Differentiation factor: 100
(3) Boundary adjustment:    1
(4) Minimum sample count:   15
(5) Normalization:          Yes

Select parameter to adjust: 0

Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): r
Say the test word.
Input active!
End-of-word detected.
Word: One
  Distance: 48
Word: Two
  Distance: 34
Word: Three
  Distance: 63

Ambiguous input!
Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): a

(0) No change
(1) Threshold factor:       35
(2) Differentiation factor: 100
(3) Boundary adjustment:    1
(4) Minimum sample count:   15
(5) Normalization:          Yes

Select parameter to adjust: 5
Do you want normalization? n

(0) No change
(1) Threshold factor:       35
(2) Differentiation factor: 100
(3) Boundary adjustment:    1
(4) Minimum sample count:   15
(5) Normalization:          No

Select parameter to adjust: 0

Command (R=Run, L=List vocab, A=Adjust params, Q=Quit): q
Ending recognition phase.
```

Figure E.2 (continued).

REFERENCES

1. S. K. Das, "Some Experiments in Discrete Utterance Recognition," IEEE Proc. Int. Conf. Accoustics, Speech & Signal Processing, pp. 178-181, 1980.

2. Data General Corporation, MPT/100 Computer System, 1981.

3. B. A. Dautrich, L. R. Rabiner, and T. B. Martin, "The Effects of Selected Signal Processing Techniques on the Performance of a Filter-Bank-Based Isolated Word Recognizer," Bell Syst. Tech. J., Vol. 62, No. 5, Part 1, May-June 1983.

4. Intel Corporation, Component Data Catalog, 1980.

5. Intel Corporation, Microcontroller User's Manual, 1982.

6. D. E. Johnson, J. R. Johnson, and H. P. Moore, A Handbook of Active Filters. Englewood Cliffs, N.J.: Prentice-Hall, 1980.

7. G. M. Miller, Linear Circuits for Electronics Technology. Englewood Cliffs, N.J.: Prentice-Hall, 1974.

8. H. Sakoe and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition," Automatic Speech & Speaker Recognition, N. R. Dixon and T. B. Martin, editors. New York: IEEE Press, 1979.

9. National Semiconductor Corporation, Linear Applications Handbook, Vol. 2, 1977.

10. National Semiconductor Corporation, Linear Data Book, 1980.

11. Texas Instruments Incorporated, The TTL Data Book, 2nd edition, 1981.

12. V. W. Zue and R. M. Schwartz, "Acoustic Processing and Phonetic Analysis," Trends In Speech Recognition, W. A. Lea, editor. Englewood Cliffs, N.J.: Prentice-Hall, 1980.