

FOR REFERENCE

NOT TO BE TAKEN FROM THIS ROOM

SPEECH SYNTHESIS
USING
REFLECTION COEFFICIENTS

by

Önder Bicioğlu

B.S. in E.E., Boğaziçi University, 1982

Bogazici University Library



14

39001100314411

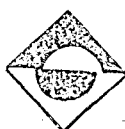
Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science
in
Electrical Engineering

Boğaziçi University

1985

to my mother

182894



ACKNOWLEDGEMENTS

I am grateful to my thesis supervisor Doç. Dr. Bülent Sankur . for his help, guidance and cooperation and especially acknowledge his encouraging supervision in this thesis.

I would also like to express my thanks to Y. Mün. Tanju Argun , Director of Research and Development Department Of Netaş, for his guidance and support.

I also thank to Hüsnü Özbek for his developing the speech synthesis kit and enabling me to subjectively judge the results.

SPEECH SYNTHESIS

USING

REFLECTION COEFFICIENTS

This study outlines a method for transmitting speech using a relatively low bit rate, in comparison to traditional methods. The method is based on the theory of Linear Prediction. In this approach speech is represented by reflection coefficients, energy and pitch parameters. The Reflection Coefficients are calculated using Gueguen - Le Ruex algorithm. Residual energy is encoded as the energy parameter. For pitch detection a modified form of SIFT algorithm is used.

The physical foundations of Reflection Coefficient approach and its theory are explained. The theory given in this study is based on inner product formulation.

The algorithms developed are applied to Turkish words, sounds and sentences. The results and observations are illustrated.

YANSIMA KATSAYILARI

KULLANARAK

SES SENTEZİ

Bu çalışmada . eski tekniklere oranla daha düşük ikil aktarımı gerektiren bir yöntem sunulmaktadır. Sunulan yöntem Doğrusal Öngörü kuramına dayanmaktadır. Bu yaklaşımda ses, yansima katsayıları, enerji ve temel uyarı sıklığı değerleri ile temsil edilmektedir. Yansima katsayıları Gueguen-Le Ruex yöntemi ile elde edilmektedir . Öngörmedeki nata, enerji katsayısını vermektedir. Temel uyarıma sıklığı ise SIFT yöntemine benzeyen bir yaklaşımla hesaplanmaktadır.

Yansima katsayılarının fiziksel anlamı ve dayanıkları kuram. çalışmada vurgulanmıştır. Kuram verilirken iç çarpım yaklaşımından yararlanılmıştır.

Geliştirilen yöntemler ve izlenceler Türkçe sözcük, tümce ve seslere uygulanmış ve elde edilen sonuçlar sunulmuştur.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	iii
ABSTRACT.....	iv
ÖZETCE.....	v
LIST OF FIGURES.....	ix
LIST OF TABLES.....	x
LIST OF SYMBOLS.....	xi
CHAPTER 1.....	1
I. INTRODUCTION.....	1
II. VOICE RESPONSE SYSTEMS.....	3
III. SPEECH STORAGE METHODS.....	6
A. Waveform Coding.....	6
B. Parametric Coding.....	9
C. Hybrid Coding.....	11
IV. DECISION CRITERIA.....	13
CHAPTER 2.....	17
I. NATURAL SPEECH PRODUCTION.....	17
II. LOSSLESS TUBE MODEL.....	21
CHAPTER 3.....	24
I. LINEAR PREDICTION MODEL.....	24
A. Least Squares.....	25
B. Maximum Likelihood.....	28
C. Prony's Method.....	30
D. Correlation Matching.....	30
E. Spectral Matching.....	32

II.	REFLECTION COEFFICIENT APPROACH.....	34
A.	Recursive Solution.....	37
B.	Gueuen Le-Ruex Algorithm.....	40
III.	FUNDAMENTAL FREQUENCY ESTIMATION.....	43
A.	Parallel Procesing Approach.....	44
B.	Short Time AMDF.....	45
C.	Autocorrelation Function.....	46
D.	SIFT Algorithm.....	47
IV.	GAIN MATCHING.....	50
V.	ENCODING.....	52
CHAPTER 4.....		54
I.	SPEECH ANALYSIS SYSTEM DECRIPTION.....	55
A.	External Hardware.....	57
B.	INTELLEC 800 MDS I/O Ports.....	61
C.	Software Modules.....	61
CHAPTER 5.....		75
I.	RESULTS.....	75
A.	Comoutational Aspects.....	75
B.	Subjective Aspects.....	78
CHAPTER 6.....		82
I.	CONCLUSION.....	82
A.	Future Applications.....	82

APPENDIX A.	32-Bit Arithmetic Operation Library.....	84
APPENDIX B.	INTELLEC 800 MDS Hardware.....	86
APPENDIX C.	ISIS-II Diskette Operating System.....	90
APPENDIX D.	A Law Companding.....	95
APPENDIX E.	Conversion Tables.....	97
APPENDIX F.	Program Listings.....	98
APPENDIX G.	Encoded LPC Coefficients.....	187
BIBLIOGRAPHY.....		209

LIST OF FIGURES

	Page
Figure 1.2.1 Block Diagram of a Voice Response System	4
Figure 1.3.1 Speech Generation Model	9
Figure 1.4.1 Quality versus Bit Rate	14
Figure 2.1.1 Human Speech Generation System	17
Figure 2.2.1 Lossless Tube Model	21
Figure 2.2.2 Signal Flow between Lossless Tubes	22
Figure 2.2.3 Relationship among z-transforms at a junction	23
Figure 3.2.1 Inner Product Formulation	34
Figure 3.2.2 Lattice Filter Model	36
Figure 3.2.3 Flowchart of Gueguen Le Ruex Algorithm	42
Figure 3.3.1 Parallel Process Pitch Estimation	44
Figure 3.3.2 SIFT Algorithm	47
Figure 4.1.1 Analyzing System Description	54
Figure 4.1.2 External Conversion Circuitry	58
Figure 4.1.3 Timing Diagrams	60
Figure 4.1.4 Flowchart of ANALPC	65
Figure 4.1.5 Hex File of "BIR"	73
Figure B.1 Block Diagram of INTELLEC 800 MDS	86
Figure C.1 Command Generation for ISIS-II	91
Figure C.2 Memory Map for ISIS-II	94

LIST OF TABLES

	Page
Table 1.4.1 Relative Complexities	13
Table 1.4.2 Bit Rates for Toll Quality	15
Table 1.4.3 Bit Rates for Communications Quality	15
Table 1.4.4 Bit Rates for Synthetic Quality	15
Table 1.4.5 Objective versus Subjective Ratings	16
Table 2.1.1 Phonemes Constituting Turkish Sounds	20
Table 4.1.1 Voltage versus 8-bit Data	67
Table D.1 A-law Expansion	96

LIST OF SYMBOLS

$A(z)$	Forward predictor
a_i	i th coefficient of forward predictor
$B(z)$	Backward predictor
b_i	i th coefficient of backward predictor
E_t	Total prediction error for one frame
$e(n)$	Prediction error at sample n
e_i	Prediction error for the whole frame at i th recursion (Levinson-Durbin)
$e_{j,i}$	Inner product term described for Gugen-Le Ruex
G	Gain or amplitude of input signal
J	Sum of squares of prediction errors
K_i	Encoded i th reflection coefficient
k_i	i th reflection coefficient
$P_i(x, t)$	Pressure in the i th tube at point x and at time t
$P(w)$	Power spectrum of the original signal
P	Predictor order
$R(i)$	i th autocorrelation coefficient
$R_e(i)$	i th autocorrelation coefficient of prediction error
$R_c(t_p)$	Cyclic pitch measure for frequency t_p
r_i	Reflection Coefficients for the lossless tube model
$S(e^{j\omega})$	Spectrum of original signal
$s(i)$	i th speech sample
$\hat{s}(i)$	Predicted value for i th sample

t_p	Pitch estimate
$u_i^+(t)$	positive going travelling volume velocity in the i th tube
$u_i^-(t)$	negative going travelling volume velocity in the i th tube
$w(i)$	Window function
T	Duration between consecutive samples
z^{-1}	Delay element
σ	Mean square error energy
γ_{ij}	Coefficient of $B_j(z)$ when generating $B_i(z)$ using Gram-Schmidt process
τ_i	Travelling time for volume velocity, until it reaches the junction

CHAPTER 1

I. INTRODUCTION

The purpose of this thesis is to outline a method for obtaining parametric representation of speech, and apply them to some commonly used Turkish words.

In the first chapter voice response systems are introduced. Their application areas and limitations are given. One of the most serious limitations is memory, where speech is stored (or alternatively bit rate at which speech can be transmitted). Therefore a brief summary of voice storage methods is given. Waveform coding, parametric and hybrid coding techniques are introduced. They are compared on the basis of commonly used criteria.

Second chapter is dedicated to natural speech production. Basic sounds and the way they are generated are described. Phonemes constituting Turkish sounds are listed. The lossless tube model for the human vocal tract and the related equations are quoted. The transition from the lossless tube model to the digital lattice filter presentation is emphasized.

In chapter three theory of linear prediction and its application to speech analysis are described. Reflection coefficient approach to Least Squares Estimation is outlined, based on the inner product formulation. Gueguen - Le Ruex algorithm and its advantages are illustrated. Methods for

extracting and encoding of other important features (energy and pitch) are also explained.

The algorithms developed during this work are introduced in chapter four. Each of the hardware and software modules used throughout the analysis are explained stage by stage.

In chapter five the observations and results are listed. Ideas to improve the quality of the output speech are introduced. Also computational aspects that might improve accuracy and speed up the analysis are mentioned.

In Conclusion the results are summarized and some future applications of this study mentioned. The results and tools developed during this study may be helpful for another stage that establishes the rules for creating messages and provides necessary hardware for speech generation.

II. VOICE RESPONSE SYSTEMS

There are generally recognized to be three major areas (modes of communication) within the general area of man-machine communication by voice. These areas include : [1]

1. Voice Response Systems
2. Speaker Recognition Systems
3. Speech Recognition Systems

Voice Response Systems are designed to respond to a request for information, using spoken messages. Thus Voice Response Systems communicate by voice in one direction only, i.e. from the machine to man. On the other hand, areas 2 and 3 deal with systems in which communication is by voice from man to machine. For Speaker Recognition Systems the task of the system is to either verify a speakers identity, i.e. a yes-no decision as to whether the speaker is who he claims to be, or to identify the speaker from some known ensemble. The basic tasks of a speech recognition system is either to recognize the entire spoken utterance exactly (e.g. a phonetic speech to text writer) or else to understand the spoken utterance (i.e. to respond in the correct manner)

The elements of a voice response system include :

1. Provision for storage of a vocabulary for the voice response system.
2. Rules for forming messages from elements of the vocabulary

3. A program for composing voice response messages

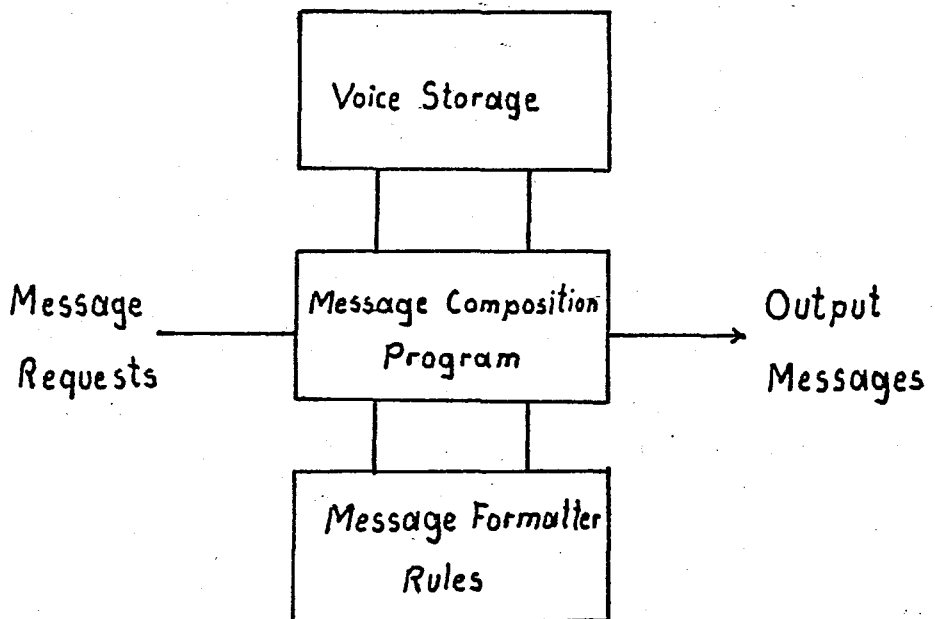


Figure 1.2.1 Block diagram of a voice response system

The input to the voice response system is in the form of a message request, which may be initiated by another information processing system or directly by a human seeking information from the voice response system. The output messages are in the form of speech utterances in response to the message requests.

Typical application areas of computer voice response systems are the following:
[2]

1. A system for producing vocal instructions for wiring communications equipment.
2. A directory assistance system
3. Talking toys and computers
4. A data set testing information system

5. A flight information system
6. A speaker verification system
7. Aids to handicapped persons

There are two main approaches to the implementations of a voice response system . One approach is the limited vocabulary system in which the output message is created by concatenating isolated natural speech elements in the vocabulary storage.

The second approach is to attempt to build a machine with powers of speech comparable to human . Such systems (often referred to as speech synthesis by-rule systems) utilize models to simulate human articulatory system. In this case the vocabulary storage is essentially a pronouncing dictionary and the message formation rules must generate the required control signals (e.g. pitch intensity , and vocal tract response parameters) to control a speech synthesizer based on the speech production model. Such systems are of interest when an unlimited vocabulary is required.

III. SPEECH STORAGE METHODS

The representation of speech signals in digital form is of fundamental concern . There are many possibilities for discrete representations of speech signals . These can be classified into three broad groups :

- A. Waveform representations
- B. Parametric representation
- C. Hybrid representation

A. Waveform Coding

In waveform coding the aim is to convert speech onto a bit stream that determines and preserves the shape of the original signal. In this regard one is guided by the well-known Sampling Theorem which states that a band-limited signal can be represented by samples taken periodically in time provided that the samples are taken at a high enough rate . The most popular coding methods available under this heading are :

- PCM (Pulse Code Modulation)
- DPCM (Differential PCM)
- DM (Delta Modulation)

In PCM , an analog speech signal , after band-limiting in a low-pass filter , is converted by a sampling circuit into a series of samples $s(n)$, separated on the time axis by an interval T . (the sampling rate) The value of

each sample $s(n)$ is then rounded off to $s(n)$ in a quantization circuit and converted into a code word of B bits an encoder.

According to the method used in adjusting the step-size of the quantizer, one distinguishes among linear PCM, log PCM and Adaptive PCM. In linear PCM step-size is fixed, in log PCM a logarithmic compansion is applied to the speech. The two log-quantization characteristics are μ -law and A-law. The idea in both methods is to quantize small signals more precisely in comparison to signals of large amplitude. They try to establish the maximum dynamic range, and at the same time exceed a given Signal-to-Noise Ratio (SNR). SNR is defined by

$$\text{SNR} = 10 \log(E_s / E_q) \quad (1.3.1)$$

where E_s and E_q are signal and quantization error energies respectively. For details about A-law companding refer to Appendix D.

If the step size is adjusted adaptively the coding scheme is called PCM with adaptive quantization.

Speech signals can be represented more efficiently by making use of the close correlation that exists between the value of successive samples $s(n)$. This is done in all systems based on differential coding. DPCM quantizes the difference between current and previous samples.

When the speech samples $s(n)$ are separated by a sufficiently small interval T , the quantization circuit need only have two levels. This is DM (Delta Modulation). By using a variable gain whose value is determined from the bit stream representing the difference, one arrives at ADM (Adaptive DM). ADM requires a less bit rate than DM.

Differential coders are actually predictors of the first order. They necessitate the difference signal to generate the current sample. The idea of predictive coding is generalized to predictive coders of orders greater than one. One of the most efficient predictive methods is APC (Adaptive Predictive Coding). Due to the nonstationary nature of speech signals, a fixed predictor cannot predict the signal values efficiently at all times. Thus the predictor must vary with time to cope with the changing spectral envelope of speech signal as well as with the changing periodicities in voiced speech.

B. Parametric Coding

In the parametric coding approach the aim is to derive a limited number (eg 10 to 20) of relatively slowly varying characteristic parameters of the speech signal and to represent these alone as a bit stream.

A widely used speech production model utilizes two signal sources P and N. The source P delivers a series of quasi-periodic pulses whose repetition frequency $1/T$ corresponds to the fundamental frequency or pitch of the speech and the source N delivers a noise like signal. The source P generates voiced sounds and N unvoiced sounds. One or the other source is selected on the basis of a voiced/unvoiced decision. The magnitude of the source signal selected can be varied by means of the gain G, and its frequency spectrum can be modified by the filter F. The model described in figure 1.3.2 can also be interpreted as the source of the sound and the resonant system that modifies (or modulates) the sound.

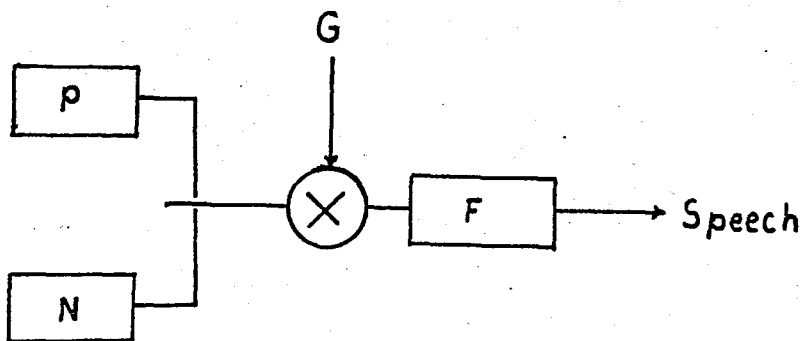


Figure 1.3.1 Speech generation model

In essence the filter performs a function comparable with that of the human vocal tract (Pharynx plus nasal or mouth cavities)

The parameters representing this model are:

- the voiced/unvoiced decision
- for voiced sounds the fundamental frequency $1/t_p$
- the magnitude of the gain G
- some form of description of filter F

This model has been the starting point for the design of many types of speech coding systems (vocoders) . The oldest type of vocoder is the channel vocoder . where the vocal tract is represented by a bank (10 to 20) of band-pass filters . At the transmitting end the average signal amplitude in each small frequency band is determined by means of a rectifier and a low-pass filter. The resultant information is transferred in coded form to the receiver . where it is used to adjust the gain in the branches of an identical bank of filters. The bandwidths of the individual filters in such a filter bank do not have to be equal: with a limited number of filters the best results are obtained when the filters for the lowest frequencies have the smallest bandwidths.

In DFT vocoders the filter bank function is performed by means of a Discrete Fourier Transform (DFT) module.

Another method of characterizing the vocal tract is based on a description of the envelope of the speech spectrum in terms of formants . resulting in a formant vocoder. The

vocal tract is now represented by four or five band-pass filters in which the bandwidths and the centre frequency vary as a function of time. The required values can be derived from speech by means of Linear Predictive Coding (LPC). A formant vocoder of this type is a member of the much larger family of LPC vocoders. The idea behind all these vocoders is again the assumption that any speech sample $s(n)$ can be well approximated by a weighted sum of preceeding speech samples:

$$s(n) = \sum_{i=1}^p a_i (s-i) \quad (3.3.2)$$

For details refer to Chapter 3.

C. Hybrid Coding

Hybrid coding techniques make use of the Waveform and Parametric coding methods at the same time. Among the many different approaches to be found, two techniques have become fairly general: Noise Shaping and Residual Coding

In Noise Shaping, the aim is to make optimum use of the noise-masking effect of human hearing (In human hearing strong sounds tend to mask weaker sounds of the same frequency). An application of this technique is Sub-band coding, where the speech signal is split into 4 to 16 frequency bands and the signals in each sub-band are separately coded.

A method that bears some resemblance to sub-band coding is Adaptive Transform Coding (ATC) in which the speech signal is first subjected to a frequency transformation and

the transformation results are then adaptively coded using optimal bit assignment.

Voice-excitation systems use a 'broad-band' excitation signal instead of the pitch and voiced/unvoiced decision information.

Two more complex forms of hybrid coding are based on Residual Coding. In the feed-forward configuration the residual signal is encoded and transmitted. In the feed-back system the excitation signal that minimizes error is obtained and transmitted. This approach is often referred to as "Analysis by Synthesis".

IV. DECISION CRITERIA

In considering the choice of digital coding schemes for Voice Response Application . it is helpful to consider three factors:

1. The information rate (bit rate) required for acceptable speech quality
2. The complexity of the coding and decoding schemes
3. The flexibility of the representation . i.e. the potential for modification of the vocabulary elements

1. The complexity is commonly given by the count of logic gates . Relative complexities of various techniques can be approximated as in table 1.1 :

Relative complexity	Coder
1	ADM
1	ADPCM
5	Sub-band coding
50	ATC
50	VEV (Voice Excited Vocoder)
100	LPC
500	Formant vocoder
1000	Articulatory (Vocal Tract synthesizer)

Table 1.4.1 Relative complexities

2. Any assesment of signal quality implies a fidelity measure . For most communication systems this measure is difficult to specify quantitatively because it involves human perception . Speech quality is traditionally assessed by the criterion that a listener understands i) What is being said

ii) Who said it

Objective measures that accurately reflect these factors are intensively sought but are difficult to establish with generality.

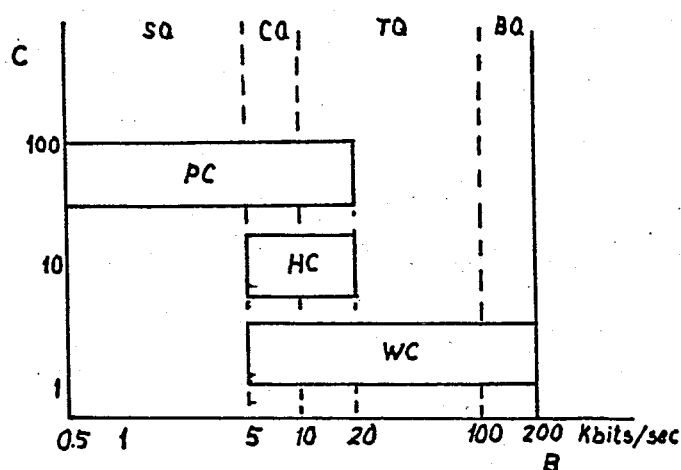


Figure 1.4.1 Complexity versus bit rate with quality as parameter.

Figure 1.4.1 indicates the quality of speech reproduction that can presently be attained at a prescribed bit rate. The quality characterizations are denoted as commentary, toll, communications and synthetic.

Toll quality loosely corresponds to the following properties:

frequency range	:	200 - 3200Hz
SNR	>	30dB
harmonic distortion	<	2-3%

Tables 1.4.2 to 1.4.4 illustrate the performance of several coders at given bit rates.

Coder	Bit rate (Kbits)
log PCM	56
ADM	40
ADPCM	32
Sub-band	24
ATC	16
Voice-excited vocoder	16

Table 1.4.2 Bit rate for some coders that can achieve toll quality

Coder	Bit rate (Kbits)
log PCM	36
ADM	24
ADPCM	16
Sub-band	9.6
ATC	7.2
Voice-excited vocoder	7.2

Table 1.4.3 Bit rate for some coders that can achieve communications quality

Coder	Bit rate (Kbits)
Channel vocoder	2.4
LPC	2.4
Formant vocoder	0.5

Table 1.4.4 Bit rate for some coders that can establish synthetic-quality transmission

Objective measures such as SNR . not always coincide with subjective measures . Table 1.4.5 compares SNR ratings with subjective preference . Although 6-bit PCM is expected to produce better results than 4-bit ADPCM . listeners preferred 4-bit ADPCM.

Objective rating (SNR)	Subjective rating (preference)
7-bit PCM	7-bit PCM
6-bit PCM	4-bit ADPCM
4-bit ADPCM	6-bit PCM
5-bit PCM	3-bit ADPCM
3-bit ADPCM	5-bit PCM
4-bit PCM	4-bit PCM

Table 1.4.5 Objective versus subjective ratings

CHAPTER 2

I. NATURAL SPEECH PRODUCTION

The acoustical speech waveform is an acoustic pressure wave which originates from voluntary physiological movements of the structure shown in figure 2.1.1:

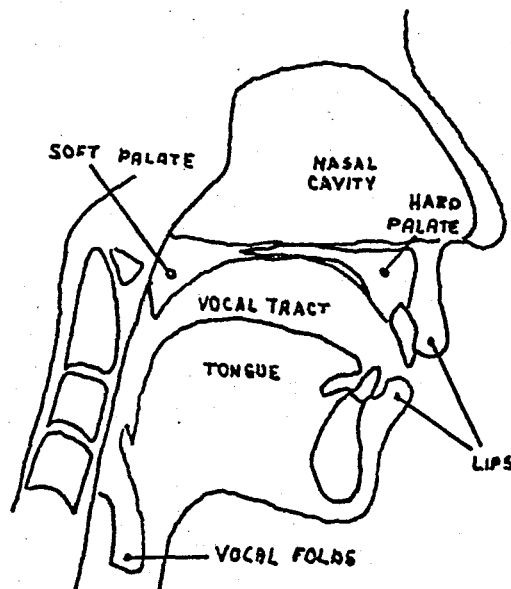


Figure 2.1.1 Human speech system

In the pulmonic air stream mechanism, which is the main mechanism to produce Turkish sounds, air is expelled from the lungs into the trachea and then forced between the vocal folds. During the generation of voiced sounds such as /l/ . /e/ the air pushed towards the lips from the lungs causes the vocal folds to open and close at a rate dependent upon the air pressure in the trachea and the physiological adjustment of the vocal folds. This adjustment includes changes in the length, thickness and tension of the vocal

folds . The greater the tension the higher the perceived pitch or acoustically measured fundamental frequency of the voice . The opening between the vocal folds is defined as the glottis . The subglottic pressure and the time variations in glottal area determine the volume velocity of glottal air flow (glottal volume velocity waveform) expelled into the vocal tract . The rate at which the glottis opens and closes can be approximately measured acoustically as the inverse of the time interval between observed periods of the acoustic pressure wave . It is the glottal volume velocity waveform that defines the acoustic energy input or driving function to the vocal tract.

Speech signals are composed of a sequence of sounds. These sounds and the transitions between them serve as a symbolic representation of information. The arrangement of these sounds (symbols) is governed by the rules of language. The study of these rules and their implication in human communication is the domain of linguistics and the study and classification of the sounds of speech is called phonetics.

The phonological system of languages is described in terms of a set of distinctive sounds or phonemes. Phonemes can be classified in the following manner:

Vowels

- 1) Simple Vowels are produced by exciting a fixed vocal tract with quasi-periodic pulses of air caused by vibration of the vocal cords. The period of this excitation is called pitch period. The range of this frequency amounts

from 33Hz for a low pitch male to 1500Hz for a high-pitch singer. (Usually sopranos should be able to produce sounds excited by a 1400Hz pitch, so they have to be able to generate higher pitch frequencies up to 1800Hz). For an average male speaker this frequency lies in the range of 100-110Hz.

As described in Section II the vocal tract is modeled as a concatenation of lossless tubes of various cross-sections. This structure determines the resonant frequencies of the tract. These are called the Formants.

ii) Diphthongs are gliding mono-syllabic speech items that start at or near the articulatory position for one vowel and moves to or towards the position for another. e.g. / ay /

Consonants

i) Stops are transient non-continuant sounds which are produced by building up pressure behind a total constriction somewhere in the oral tract and suddenly releasing the pressure. /p/, /t/, /k/ are voiceless stops. If during the sound production also the vocal cords vibrate a voiced stop is produced. Voiced stops are /b/, /d/ and /g/.

ii) Fricatives are produced by exciting the vocal tract by a steady air flow which becomes turbulent in the region of a constriction in the vocal tract. Voiceless fricatives are /f/, /s/ and /ʃ/. If additionally vocal cords vibrate during the air flow, the sound generated is a voiced fricative. /v/, /z/, /ʒ/ are such sounds.

iii) Affricatives are produced by first totally constricting air passage and then releasing it with friction.

/c/ is a voiceless affricative and /j/ is a voiced one.

iv) Nasals are produced with glottal excitation and the vocal tract totally constricted at some point along the oral passage way. /m/, /n/ are nasal sounds.

v) Lateral is produced by flow of air on both sides of the tongue. /l/ is an example for lateral sounds.

vi) Sounds that are voiced in initial and intervocalic positions and voiceless in final position are retroflex. /r/ is such a sound.

Semivowels are the group of sounds that cannot easily be distinguished from vowels. /y/ is such a sound.

An allophone is defined as a speech sound which is modified by the environment in which it occurs. The results of this study may be used to investigate the features of allophones, and speech synthesis rules using allophones can be developed.

VOWELS

i	ü	ı	u
e	ö	a	o

CONSONANTS

stops	p	t	k
	b	d	g
fricatives	f	s	h
	v	z	
affricatives			
nasals	m	n	
lateral		l	
retroflex		r	
semivowel			y

Table 2.1.1 Phonemes Constituting Turkish Sounds

II. LOSSLESS TUBE MODEL

A widely used model for speech production is based upon the assumption that the vocal tract can be represented as a concatenation of lossless acoustic tubes as shown in Figure 2.2.1 [1]

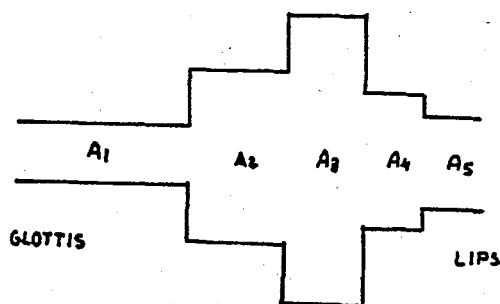


Figure 2.2.1 Lossless tube model

The propagation of waves in concatenated lossless tubes is governed by the following equations :

$$P_i(x,t) = (\rho_c / A_k) (u_i^+(t-x/c) + u_i^-(t+x/c)) \quad (2.2.1)$$

$$u_i(x,t) = u_i^+(t-x/c) - u_i^-(t-x/c) \quad (2.2.2)$$

where x is distance measured from left end of the i th tube and $u_i^+(\cdot)$ and $u_i^-(\cdot)$ are positive going and negative going travelling volume velocities in the i th tube. A_i gives cross-section of the i th tube and P_i gives pressure in i th tube. Solving these equations and using boundary conditions at the junctions results in:

$$u_{k+1}^+(t) = (2A_{k+1}/(A_{k+1}+A_k)) u_k^+(t-\tau_k) + r_k u_{k+1}^-(t) \quad (2.2.3)$$

$$u_k^-(t+\tau_k) = -r_k u_k^+(t-\tau_k) + (2A_k/(A_{k+1}+A_k)) u_{k+1}^-(t) \quad (2.2.4)$$

The quantity $r_i = (A_{i+1} - A_i) / (A_{i+1} + A_i)$ is the amount of $u_{i+1}^-(t)$ that is reflected at the junction. Thus the quantity r_i is called the reflection coefficient for the i th junction. Since all cross-section values are positive the value of r 's are between -1 and +1.

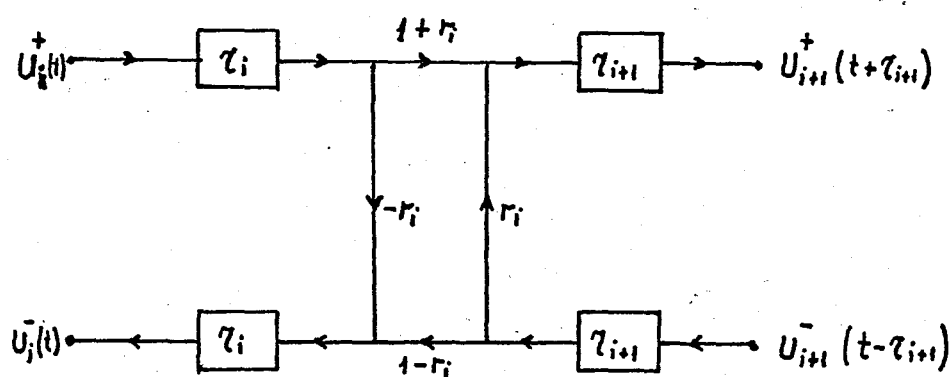


Figure 2.2.2 Signal flow representation of the junction between two lossless tubes

One can easily observe the isomorphism between the signal flow representation of the junction between two lossless tubes and the corresponding digital filter: r_i of figure 2.2.2 corresponds to k_i of figure 2.2.3, similarly, delays of figure 2.2.2 represented by the z -term of figure 2.2.3.

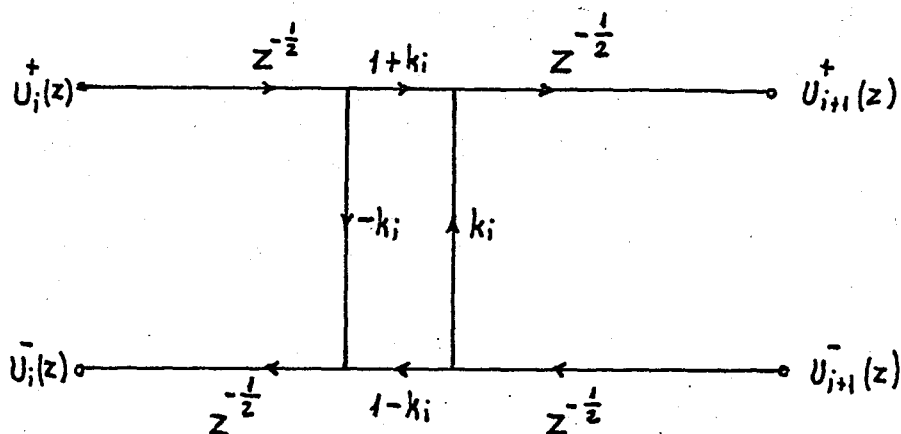


Figure 2.2.3 Flow graph representing relationship among z -transforms at a junction

When passing to the digital filter representation of the vocal tract, one considers a system composed of p lossless tubes of length $1/p$. In this modeling p gives the order of the filter. Length of various sections are selected to be equal in order to provide the same amount of delay between sections. This makes the usage of z as a common delay operator possible.

CHAPTER 3

I. LINEAR PREDICTION MODEL

Although linear least squares estimation (or prediction) dates from Gauss in 1795 . it appears that the first specific use of the term 'Linear Prediction' was in Wiener's 1949 book in chapter 2. 'The Linear Predictor for a Single Time Series'.

The first application of linear prediction to speech from a Maximum Likelihood formulation was in the work of Saito and Itakura in 1966. In 1967 and 1968 . Atal and Schroder published results on Predictive Coding of speech signals. Since then . several other equivalent Linear Prediction formulations have been developed . The most popular ones are the following:

- A. Least Square
- B. Maximum Likelihood
- C. Prony's Method
- D. Correlation Matching
- E. Spectral Estimation

Actually all of the mentioned techniques correspond to different perspectives of the model . but they all end up in the same or similar equations.

A. Least Squares

[8]

Least Squares approach is useful in the identification of the finite order Auto-regressive (AR) model for scalar observations $s(0), s(1), \dots, s(N-1)$. The linear model is represented by :

$$\hat{s}(n) = - \sum_{i=1}^p a_i s(n-i) + Gu(n) \quad (3.1.1)$$

Where a_i are the coefficients to be identified :

$s(k-1)$ corresponds to previous observations and $u(k)$ is a bounded variance, zero-mean and ergodic white noise sequence with

$$E\{u(k)\} = 0 \quad \text{for all } k \quad (3.1.2.a)$$

$$E\{u(k)u(l)\} = R \quad \text{for all } k, l \quad (3.1.2.b)$$

for some finite $R > 0$

The error between $s(k)$ and its predicted value $\hat{s}(k)$ is given by :

$$e(k) = s(k) - \hat{s}(k) = \sum_{i=0}^p a_i s(k-i) \quad (3.3)$$

where $a_0 = 1$

Least Squares method minimizes the sum of the squares of the prediction errors:

$$(3.1.4)$$

The 'normal equations' are obtained by setting the partial derivatives of J with respect to each a_i equal to zero:

$$\sum_{i=1}^p a_i c(i, j) = -c(0, j) \quad \text{for } j = 1, 2, \dots, p \quad (3.1.5)$$

where

$$c(i, j) = \sum_k s(k-i)s(k-j) \quad (3.1.6)$$

The Direct Least Squares (LS) method can be derived from the normal equations in matrix form as:

$$\hat{\underline{a}} = - (\underline{\underline{S}}^T \underline{\underline{S}})^{-1} \underline{\underline{S}}^T \underline{s} \quad (3.1.7)$$

where

$$\underline{\underline{S}} = \begin{bmatrix} s(p-1) & s(p-2) & \dots & \dots & \dots & \dots & s(0) \\ s(p) & \dots & \dots & \dots & \dots & \dots & s(1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ s(N-2) & s(N-3) & \dots & \dots & \dots & \dots & s(N-p-1) \end{bmatrix} \quad (3.1.8)$$

and

$$\underline{s} = \begin{bmatrix} s(p) \\ s(p+1) \\ \vdots \\ s(N-1) \end{bmatrix} \quad (3.2.9)$$

The Covariance method for a given zero mean data sequence can be stated as in equation 3.1.10 to 3.1.12. The matrix product $\underline{\underline{S}}^T \underline{\underline{S}}$ is indicated as \underline{W} and $\underline{\underline{S}}^T \underline{s}$ is called $\underline{R}(0)$.

$$\hat{\underline{a}} = \underline{\underline{W}}^{-1} \underline{R}(0) \quad (3.1.10)$$

$$\underline{\underline{W}} = \begin{bmatrix} c(1,1) & c(2,1) & \dots & c(p,1) \\ c(1,2) & c(2,2) & \dots & c(p,2) \\ \vdots & \vdots & \ddots & \vdots \\ c(1,p) & c(2,p) & \dots & c(p,p) \end{bmatrix} \quad (3.1.11)$$

and

$$\underline{R}(0) = \begin{bmatrix} c(0,1) \\ c(0,2) \\ \vdots \\ c(0,p) \end{bmatrix} \quad (3.1.12)$$

If the signal $s(k)$ is wide-sense stationary for large number of observations N . $R(i,j)$ is equivalent to the sample autocorrelation $R(|i-j|)$ of $s(k)$ i.e.

$$R(|i-j|) = \sum_{k=0}^{N-1} s(k)s(k-|i-j|) \quad \text{for } i,j=1,2,\dots,p \quad (3.1.13)$$

The following matrix equation represents the approach:

$$\underline{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} R(0) & \dots & R(p-1) \\ R(1) & \dots & R(p-2) \\ \vdots & \ddots & \vdots \\ R(p-1) & \dots & R(0) \end{bmatrix}^{-1} \begin{bmatrix} R(1) \\ R(2) \\ \vdots \\ R(p) \end{bmatrix} \quad (3.1.14)$$

The fact that Autocorrelation matrix is Toeplitz (all elements along each diagonal are equal) enables that various approaches have been developed for solving these equations.

B. Maximum Likelihood

This approach is based on the question what choice of parameter values makes the probability of occurrence of the actual observations (the speech samples) most likely, i.e. maximizes the probability density of parameters?

Speech samples are modeled as elements of a Gaussian, stationary, random process generated by passing uncorrelated noise through an all-pole filter of the form:

$$1 / A(z) = 1 / \sum_{i=0}^M a_i z^{-i} \quad \text{where } a_0 = 1 \quad (3.1.15)$$

Knowing that the sequence $\{s(n)\}$ is Gaussian with zero mean one can define the joint probability density of the variables $s(0), s(1), s(2), \dots, s(N-1)$. It is simply the standard multivariate probability density:

$$p(n) = f(a_1, a_2, a_3, \dots, a_M, n) \quad (3.1.16)$$

The optimum values for a are to be found by differentiating $p(n)$ with respect to a and setting them equal to zero. But for M greater than 2 the problem becomes extremely non-linear.

Two different approximations have been implemented: Itakura and Saito utilized approximations based upon the assumption $N \gg M$ (the number of data points greatly exceeds the filter order) to show that the joint probability

density for the sequence $s(0), s(1), \dots, s(N-1)$ can be approximated by:

$$p(n) = (2\pi\sigma^2)^{-N/2} \cdot e^{-\alpha \frac{1}{2\sigma^2}} \quad (3.1.17)$$

where

$$\alpha = \sum_{n=-\infty}^{\infty} \left[\sum_{i=0}^p a_i s(n-i) \right]^2 \quad (3.1.18)$$

α is precisely the error energy in the Autocorrelation approach to LS Linear Prediction.

This approximation results in the same equation as in the Autocorrelation method, with the additional property that a gain term $\sigma^2 = \alpha/N$ is obtained for matching the energy of the model to that of the input signal.

The second approximation involves a conditional Maximum Likelihood approach, where the conditional probability density p is given by

$$p = (2\pi\sigma^2)^{-(N-M)/2} e^{-\alpha/2\sigma^2} \quad (3.1.19)$$

and

$$\alpha = \sum_{n=M}^{N-1} \left[\sum_{i=0}^M a_i s(n-i) \right]^2 \quad (3.1.20)$$

and results in the equations obtained in the Covariance method, with the additional property that

$$\sigma_e^2 = \alpha/(N-M)$$

C. Prony's Method

[4]

This approach models voiced speech during a pitch period as a sum of complex exponentials which add up to a real term. If speech is considered to be consisting of M such terms $2M$ speech samples are enough to correctly model the system.

A more general model taking care of transients from previous pitch period and zeroes due to nasal tract can be described as in equation (3.1.21)

$$X(z) = P(z) / A(z) = \frac{\sum_{i=0}^{M-1} p_i z^{-i}}{\sum_{i=0}^M a_i z^{-i}} \quad (3.1.21)$$

D. Correlation Matching

Let $H(z) = 1/A(z)$ be the speech synthesizer model and denote its impulse response by the sequence $\hat{h}(n)$

$$\hat{h}(n) = \sum_{i=1}^P a_i \hat{h}(n-i) + G \delta(n) \quad (3.1.22)$$

By multiplying both sides with $\hat{h}(n-k)$ and summing over n , one obtains the expression for autocorrelation coefficients of impulse response:

$$\hat{R}_n(k) = \sum_{i=1}^P a_i \hat{R}_n(k-i) + G^2 \delta(k) \quad (3.1.23)$$

The energies of $s(n)$ and $h(n)$ have to be equal :

$$R_n(0) = \hat{R}_n(0) \quad (3.1.24)$$

and similarly for other coefficients:

$$R_n(i) = \hat{R}_n(i) \quad (3.1.25)$$

These equalities follow from the similarity of equations (3.1.14) and (3.1.23) for $k > 0$.

This way the problem is reduced to obtaining the filter whose first $1+p$ values of autocorrelation of its impulse response are equal to the first $0+1$ values of the speech signal autocorrelation.

Similarly if the input $u(n)$ to the model of a sequence of uncorrelated samples (white noise) with zero mean and unit variance, the output $s(n)$ of the filter is given by:

$$\hat{s}(n) = - \sum_{i=1}^p a_i s(n-i) + G u(n) \quad (3.1.26)$$

By multiplying both sides with $s(n-i)$ and taking expected values, one obtains autocorrelation of the output $s(n)$:

$$\hat{R}(i) = - \sum_{k=1}^p a_k R(i-k) + G \delta(i) \quad (3.1.27)$$

Last term exists due to the fact that $u(n)$ and $s(n-i)$ are uncorrelated for $i > 0$. It is the dualism between the deterministic impulse and statistical white noise, that enables their usage in speech synthesis models.

D. Spectral Matching

In the frequency domain formulation of LPC the error $e(n)$ between the actual and predicted signal has a z -transform of the form:

$$E(z) = [1 + \sum_{i=1}^p a_i z^{-i}] S(z) = A(z)S(z) \quad (3.1.28)$$

The total error energy is given by (Parseval's theorem)

$$\bar{\epsilon}_t = \sum_{n=-\infty}^{\infty} e^2(n) = 1/2 \pi \int_{-\pi}^{\pi} |E(e^{-j\omega})|^2 d\omega \quad (3.1.29)$$

where $E(e^{-j\omega})$ is obtained by evaluating $E(z)$ on the unit circle $z=e^{j\omega}$. Equivalently:

$$\bar{\epsilon} = 1/2 \pi \int_{-\pi}^{\pi} P(\omega) A(e^{j\omega}) A(e^{-j\omega}) d\omega \quad (3.1.30)$$

$$\text{where } P(\omega) = |S(e^{j\omega})|^2$$

The power spectrum $P(\omega)$ of the all-pole model is given by:

$$P(\omega) = |H(e^{j\omega})|^2 = G^2 / |A(e^{j\omega})|^2 \quad (3.1.31)$$

and power spectrum $P(\omega)$ of the original signal is given by :

$$P(\omega) = |E(e^{j\omega})|^2 / |A(e^{j\omega})|^2 \quad (3.1.32)$$

By comparing these two equations, one can see that if $P(\omega)$ is being modeled by $\bar{P}(\omega)$, then the error power spectrum $|E(e^{j\omega})|^2$ is being modeled by a flat spectrum equal to G . This means that the actual error signal $e(n)$ is being approximated by another signal that has a flat spectrum.

such as a unit impulse, white noise or any other signal with a flat spectrum. Therefore filter $A(z)$ is sometimes known as a 'whitening filter'.

II REFLECTION COEFFICIENT APPROACH TO LEAST SQUARES

As outlined in section I Least Squares method tries to minimize the sum of the squares of prediction error.

If one assumes that the filter $A(z)$ establishes this minimum, the following progress is possible : [4]

Definition : Inner product of two filters $F(z)$ and $G(z)$ is defined as follows:

$$\langle F(z), G(z) \rangle = \sum_{n=n_0}^{n_1} u(n) v(n) \quad (3.2.1)$$

where $u(n)$ and $v(n)$ are the outputs of respective filters when a common input $x(n)$ is applied to them.

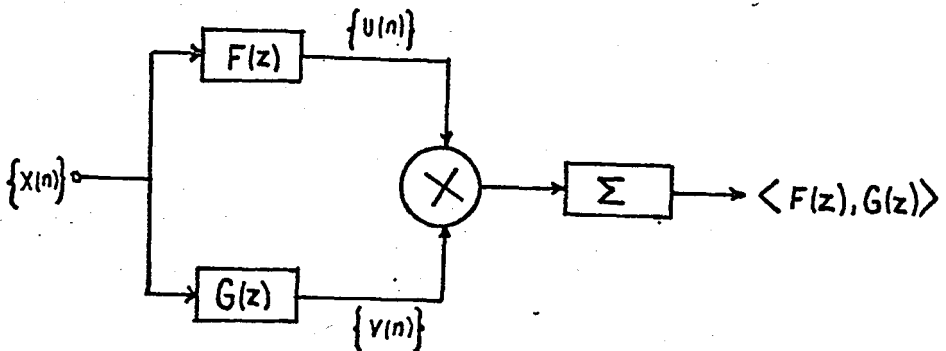


Figure 3.2.1 Inner product formulation

$$\text{If } F(z) = \sum_{i=0}^{\infty} f_i z^{-i} \text{ and } G(z) = \sum_{i=0}^{\infty} g_i z^{-i}$$

then

$$\langle F(z), G(z) \rangle = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} [f_i [\sum_{n=n_0}^{n_1} x(n-i) x(n-j)]] g_j \quad (3.2.2)$$

and

$$\langle z^{-i}, z^{-j} \rangle = \sum_{n=n_0}^{n_1} x(n-i) x(n-j) \quad (3.2.3)$$

It is easy to show that the filters together with this definition of inner product define a vector space.

With this definition of inner product, prediction error to be minimized is the norm squared of filter $A(z)$

$$e(n) = \langle A(z), A(z) \rangle \quad (3.2.4)$$

$$\text{where } A(z) = \sum_{i=0}^p a_i z^{-i}$$

If norm of $A(z)$ is minimum, then norm of $A(z) + cz^{-j}$ $j=1,2,\dots,p$ has to be greater than $A(z)$

$$\begin{aligned} \langle A(z) + cz^{-j}, A(z) + cz^{-j} \rangle &> \langle A(z), A(z) \rangle \\ \langle A(z), A(z) \rangle + 2c \langle A(z), z^{-j} \rangle + c \langle z^{-j}, z^{-j} \rangle &> \langle A(z), A(z) \rangle \\ 2c \langle A(z), z^{-j} \rangle + c \langle z^{-j}, z^{-j} \rangle &> 0 \end{aligned}$$

since c is nonzero and arbitrary one can select

$$c = -\langle A(z), z^{-j} \rangle / \langle z^{-j}, z^{-j} \rangle$$

and the following inequality follows :

$$-\langle A(z), z^{-j} \rangle / \langle z^{-j}, z^{-j} \rangle > 0 \quad (3.2.5)$$

since $\langle z^{-j}, z^{-j} \rangle$ is always positive, the only condition is :

$$\langle A(z), z^{-j} \rangle = 0 \quad (3.6)$$

i.e. filter $A(z)$ is orthogonal to z^{-j} $j=1,2,\dots,p$

This condition is actually identical to equation (3.1.14)

The structure of the filter can be modeled as described by figure 3.2.2.

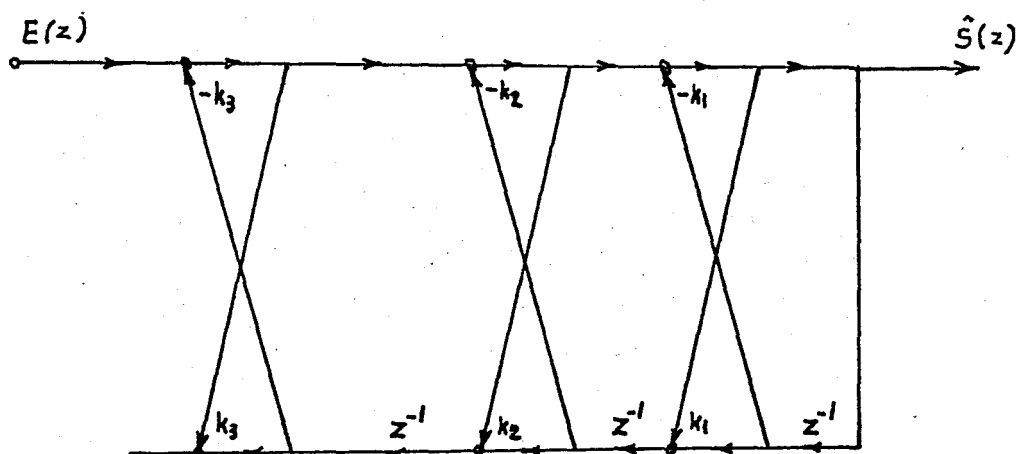


Figure 3.2.2. Lattice filter

As observed in the model there is another filter utilized in the model $B(z) = \sum_{i=1}^p b_i z^{-i}$. This filter tries to predict past samples utilizing following ones. It can be interpreted as a backward predictor. The purpose of the Least Squares approach is also to minimize the error due to this backward prediction.

Since in the forward predictor the coefficients corresponding to the last sample (sample to be predicted) is one, the backward predictor has one as the coefficient of its $(p+1)$ th term. (It tries to predict $s(n-p-1)$ as a linear combination of $s(n-1), \dots, s(n-p)$)

The same procedure can be applied to show that $B(z)$ is orthogonal to all powers of z^{-j} $j=1, 2, \dots, p+1$

A. Recursive Solution

The advantage of this approach is that one obtains the optimum prediction filters of smaller orders until one ends up in the final p th order prediction filter. There is one more important point in this procedure :

$B_i(z)$ and $B_j(z)$ are orthogonal to each other, unless $i=j$.

$$\langle B_i(z), B_j(z) \rangle = 0 \quad i \neq j \quad (3.2.7)$$

This condition follows from the fact that $B(z)$ does not have a z term. Same condition does not apply to $A(z)$, because $A(z)$ has a zeroth power term.

One starts with

$$A_0(z) = 1 \quad \text{and} \quad B_0(z) = z^{-1} \quad (3.2.8.a)$$

$$|A_0(z)| = R(0) \quad \text{and} \quad |B_0(z)| = R(0) \quad (3.2.8.b)$$

$$\langle A_0(z), B_0(z) \rangle = R(1) \quad (3.2.8.c)$$

where $R(1)$ are the autocorrelation coefficients.

$A_1(z)$ has to be orthogonal to z^{-1} ; so in the general form:

$$A_1(z) = A_0(z) + k_1 B_0(z) \quad (3.2.9)$$

by taking inner product with z^{-1} one gets:

$$k_1 = -\langle A_0(z), z^{-1} \rangle / \langle B_0(z), z^{-1} \rangle = -R(1)/R(0) \quad (3.2.10)$$

$B_1(z)$ has to be orthogonal to $B_0(z)$

$$B_1(z) = z^{-2} + \gamma_{10} B_0(z) \quad (3.2.11)$$

taking inner product with $B_0(z)$ results in

$$\gamma_{10} = -\langle B_0(z), z^{-2} \rangle / \langle B_0(z), B_0(z) \rangle \quad (3.2.12)$$

The procedure carries on until $A_p(z)$ and $B_p(z)$ are reached. Each time the following equations are utilized

$$A_{i+1}(z) = A_i(z) + \kappa_i B_i(z) \quad (3.2.13.a)$$

$$B_{i+1}(z) = z^{-(i+1)} + \sum_{k=0}^i \gamma_{i+1,k} B_k(z) \quad (3.2.13.b)$$

Equation (3.2.13.b) is actually the implementation of Gram-Schmidt orthogonalization process to filters $B_i(z)$.

The autocorrelation method has the property that $R(i) = R(-i)$. This enables one to obtain $B_{i+1}(z)$ in a much easier way:

$$B_{i+1}(z) = z^{-1} [\kappa_{i+1} A_i(z) + B_i(z)] \quad (3.2.14)$$

$B_i(z)$ is related to $A_i(z)$ in the following manner:

$$B_i(z) = z^{-(i+1)} A_i(z^{-1}) \quad (3.2.15.a)$$

or similarly

$$a_{i,k} = b_{i,i+1-k} \quad (3.2.15.b)$$

The algorithm can be summarized as follows:

$$a_{0,0} = 1 \quad b_{0,0} = 1 \quad e_0 = R(0) \quad (3.2.16.a)$$

$$k_1 = -R(1)/R(0) \quad (3.2.16.b)$$

$$a_{1,0} = 1 \quad b_{1,0} = 1 \quad e_1 = e_0 (1 - k_1^2) \quad (3.2.16.c)$$

$$a_{1,1} = k_1 \quad b_{1,1} = k_1 \quad (3.2.16.d)$$

$$k_2 = [a_{1,0} R(2) + a_{1,1} R(1)] / e_1 \quad (3.2.16.e)$$

$$a_{i,0} = 1 \quad b_{i,0} = 1 \quad e_i = e_{i-1} (1 - k_i^2) \quad (3.2.16.f)$$

$$a_{i,j} = a_{i-1,j} + k_i a_{i-1,i-j} \quad (3.2.16.g)$$

$$b_{i,j} = a_{i,i+1-j} \quad (3.2.16.h)$$

$$k_i = - \sum_{j=0}^{i-1} a_{i-1,j} R(i-j) / e(i-1) \quad (3.2.16.i)$$

$$a_{ij} = k_j \quad (3.2.16.j)$$

In literature this algorithm is known as the Levinson-Durbin algorithm.

B. Gueguen - Le Ruex Algorithm

Equation (3.2.16) describes how reflection coefficients can be evaluated recursively using the autocorrelation coefficients. Levinson-Durbin algorithm also produces and uses predictor coefficients of lower order predictors. But throughout the calculation a_i 's have no other usage except the evaluation of numerator term for the expression of k in equation (3.2.16.e) Numerator term can be interpreted as the projection of recently obtained predictor on the new added delay element: $\langle A_i(z) , z^{-(i+1)} \rangle$

Gueguen - Le Ruex algorithm makes use of this inner product as an intermediate variable but bypasses the calculation of lower order predictors and expression of this inner product in terms of lower order predictor coefficients. It can be summarized as in equations (3.2.17)

$$e_{0,i} = R(i) \quad i = 1, \dots, p \quad (3.2.17.a)$$

$$e_{0,i-1} = R(i-1) \quad i = 1, \dots, p \quad (3.2.17.b)$$

$$k_1 = -e_{0,1} / e_{0,0} \quad (3.2.17.c)$$

$$e_{1,0} = e_{0,0} + k_1 e_{0,1} \quad (3.2.17.d)$$

$$e_{1,p} = e_{0,p} + k_1 e_{0,p-1} \quad (3.2.17.e)$$

$$e_{1,i} = e_{0,i} + k_1 e_{0,i-1} \quad (3.2.18.f)$$

$$e_{1,i-1} = e_{0,i-1} + k_1 e_{0,i} \quad (3.2.18.g)$$

generally

$$k_j = -e_{j-1,j} / e_{j-1,0} \quad (3.2.18.h)$$

$$e_{j,0} = e_{j-1,0} + k_j e_{j-1,j} \quad (3.2.18.i)$$

$$e_{j,p} = e_{j-1,p} + k_j e_{j-1,1-p} \quad (3.2.18.j)$$

$$e_{j,i} = e_{j-1,i} + k_j e_{j-1,1-i} \quad (3.2.18.k)$$

$$e_{j,1-i} = e_{j-1,1-i} + k_j e_{j-1,i} \quad (3.2.18.l)$$

Instead of predictor coefficients, the projection of inverse filter on various delay elements ($\langle A_j(z) \cdot z^{-j} \rangle$; $j=1,2,\dots,m$) are calculated.

Also the update described in equation (3.2.16.i) is implemented in another way. Theoretically for infinite number accuracy the results should be the same.

Update described in equation (3.2.18.f) requires two multiplications and one division. While Gueguen-Le Ruex only requires one division and one multiplication for the corresponding update described by (3.2.18.i) Inner product terms are denoted by:

$$e_{j,i} = \langle A_j(z) \cdot z^{-i} \rangle = \sum_{m=0}^j a_{jm} R(i-m) \quad (3.2.19.a)$$

$$e_{j,j+1-i} = \langle B_j(z) \cdot z^{-i} \rangle = \langle A_j(z) \cdot z^{i-j-1} \rangle \quad (3.2.19.b)$$

The flow of algorithm is described in figure 3.2.3.

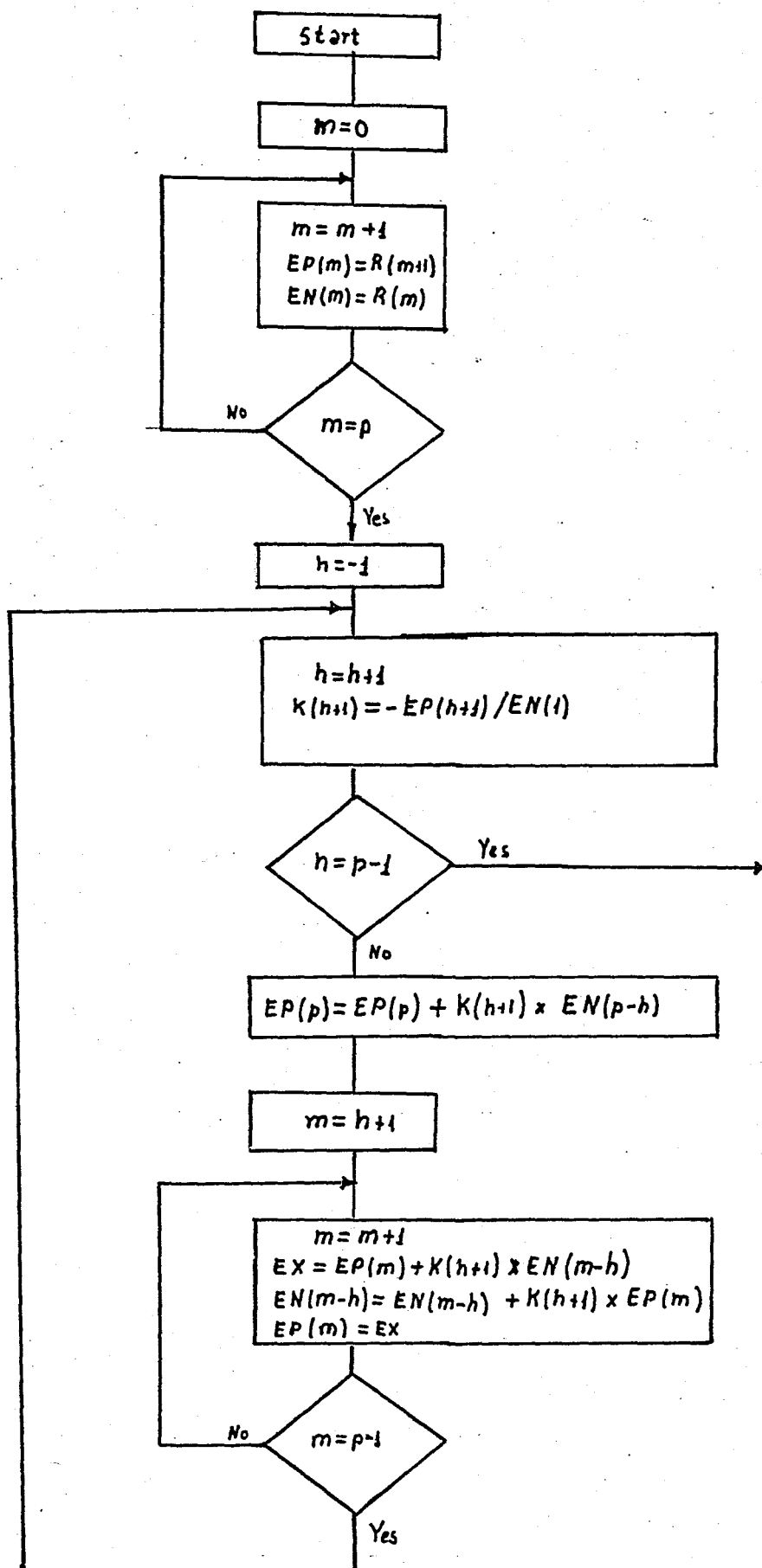


Figure 3.2.3 Flowchart of Gueguen-Le Ruex Algorithm

III. FUNDAMENTAL FREQUENCY ESTIMATION

The model of figure 1.3.1 includes a periodic impulse generator to simulate generation of voiced sounds. The repetition rate of these pulse is actually the fundamental frequency (or pitch frequency). Although pitch period is a personal characteristics, it may change throughout a sentence or even a long sound.

[10]

It ranges from 50 to 500 Hz for regular speech. For an average male voice it is about 100 Hz. This wide range makes it difficult to establish a general algorithm for pitch period estimation. But general rules may be stated as follows:

- i) A framelength that includes at least one full pitch period should be selected
- ii) In order to minimize interaction between pitch and lowest formant frequencies, preemphasis should be applied
- iii) An efficient measure of repetitiveness should be established
- iv) Thresholds should be adjusted to distinguish between silence, voiced and unvoiced sound.

Four methods are reviewed in this section. The one used in this study is modified SIFT algorithm.

A. Parallel Processing Approach

This pitch detection algorithm is based on purely time domain processing and is therefore popular for real time applications. [11], [12] Figure 3.3.1 describes its operation.

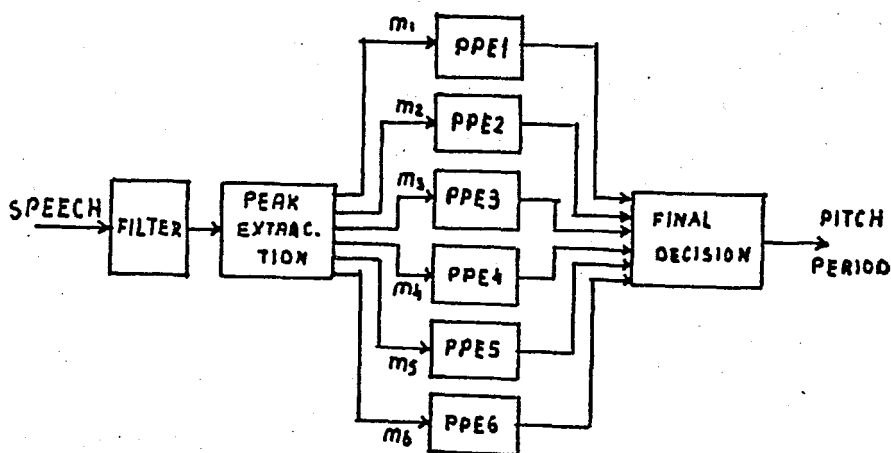


Figure 3.3.1 Parallel Process Pitch Estimation

m_1 Impulse of peak amplitude at each peak

m_2 Impulse of difference amplitude between temporary peak and preceeding valley at each peak

m_3 Impulse of difference amplitude between temporary peak and preceeding peak at each peak

m_4 Impulse of negative amplitude at each valley

m_5 Impulse of negative amplitude of temporary valley plus the preceeding peak value at each valley

m_6 Impulse of negative amplitude of temporary valley plus the preceeding amplitude of valley at each valley.

If any difference comes out as negative then the result is accepted as zero.

Each impulse train is processed by a time varying nonlinear system. When an impulse of sufficient amplitude is detected in the input, the output is reset to the value of that impulse and then held for a blanking interval, $t(n)$ during which no pulse can be detected. At the end of $t(n)$, the output begins to decay exponentially. When an impulse exceeds the level of the exponential decaying output, the process is repeated. This procedure results in six estimates of the pitch period (six quasi-periodic sequences of pulses are obtained). These six estimates are combined with two of the most recent estimates for each of the six pitch detectors. These estimates are then compared and the most frequently occurring is declared as the pitch period at that time. For unvoiced speech there is a distinct lack of consistency among estimates.

B. Short Time Average Magnitude Difference Function

This approach makes use of the fact that for periodic signals $d(n) = x(n) - x(n-k)$ approaches zero as k approaches the period t . ^[1] AMDF is defined as:

$$\text{AMDF}(k) = \sum_{m=0}^{N-1} |x(n+m)w(m) - x(n+m-k)w(m-k)| \quad (3.3.1)$$

With floating point arithmetic, where multiplies and adds take approximately the same time, about the same time is required for other methods. However, for special purpose hardware, or with fixed point arithmetic, AMDF appears to

have the advantage. In this case multiplies usually are more time consuming and furthermore either scaling or a double precision accumulator is required to hold the sum of lagged products.

C. Autocorrelation Function

The short-time autocorrelation function is defined as

$$R(k) = \sum_{m=-\infty}^{\infty} x(m)w(n-m)x(m+k)w(n-k-m) \quad (3.3.2)$$

The autocorrelation of voiced speech shows periodicities. One problem encountered is : If N (number of samples in a frame) is not chosen sufficiently large (less than or equal to pitch period) pitches cannot be detected.

The modified short-time autocorrelation function is defined as :

$$R_n(k) = \sum_{m=0}^{N-1} x(n+m) x(n+m+k) \quad (3.3.3)$$

To avoid the difficulty, due to the fact that autocorrelation function has many peaks: (because of damped oscillations of the vocal tract response) "Spectrum Flatteners" are used. Their function is to process the speech signal such that the periodicity is made more prominent by distracting features of the signal.

D. SIFT Algorithm

SIFT (Simplified Inverse Filter Tracking) is an implementation of the autocorrelation approach. The block diagram in figure 3.3.2 describes this model:

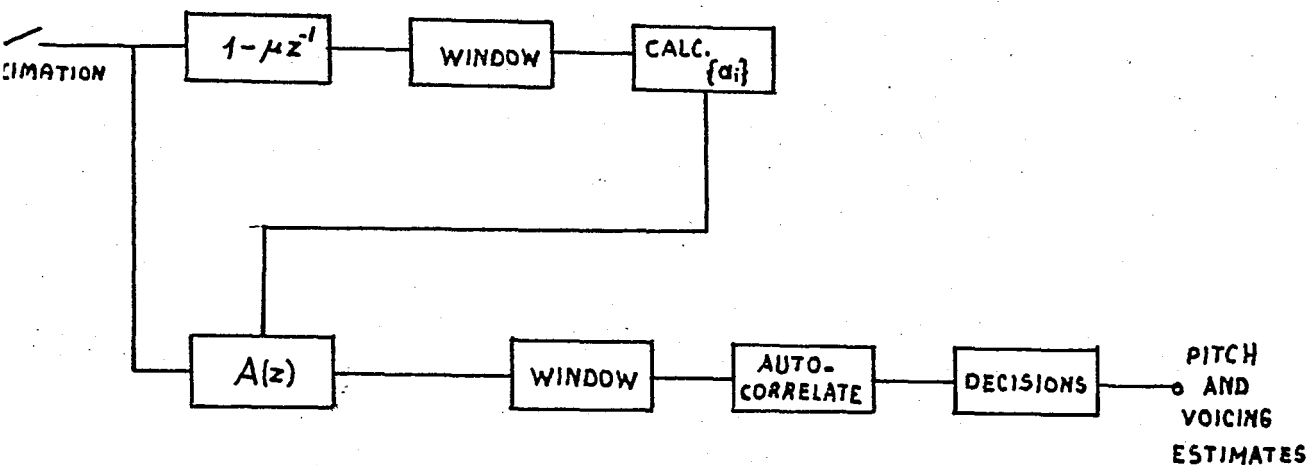


Figure 3.3.2 SIFT Algorithm

At the input the speech signal is decimated by taking one out of four samples. This does not cause any problems because frequencies of interest lie within 0-1kHz range.

$\{1 - \mu z^{-1}\}$ is a pre-emphasizer module that differentiates the input signal to accentuate the region of the second formant.

Spectrum flattening is implemented by calculating the predictor coefficients and passing the decimated speech samples through the whitening filter. The whitening filter is a fourth order predictor whose output is a sequence of prediction errors $e(n)$. These samples are passed through a window. This is necessary to deemphasize peaks at both ends

of a given frame. Peaks at the ends occur, because at the beginning one tries to predict nonzero samples from zeroed ones and at the end zeroed samples from nonzero ones.

After the autocorrelation functions have been calculated, the peaks are selected and normalized by $R(0)$. If this ratio exceeds a threshold, this value is declared as a candidate for the searched pitch period. For low pitch periods this threshold is large, for high pitch values low.

After the first pitch prediction rough errors are smoothed and parabolic interpolation is applied to the pitch estimates.

The method used in this work is essentially a modified form of SIFT algorithm. Instead of the autocorrelation function a new function is calculated. This is a cyclic autocorrelation function and is defined as:

$$R_C(t_p) = \sum_{n=0}^{N-t_p} x(n)x(n+t_p) + \sum_{n=t_p}^{N-1} x(n)x(n-t_p M) \quad (3.3.4)$$

M is restricted to be an integer greater than 2 and N is the number of samples in a frame. This definition is based on the assumption that the signal repeats itself with the fundamental frequency $1/t_p$. In this case, one cannot calculate autocorrelation coefficients for $N-p < n < N$. But with this definition the same number of terms are added together for any value of the pitch estimates. This also eliminates the need of defining different thresholds for

different pitch estimates while SIFT algorithm requires this. For a pitch range of 2.5 msec to 12.5 msec, a frame length of at least 37.5 msec is required. At 8 KHz sampling rate this corresponds to 300 samples. A typical threshold for voiced decision is 0.4 (i.e. ratio of the introduced pitch measure and total prediction error energy should exceed 0.4 for voiced signal frames)

IV. GAIN MATCHING

Following the reasoning of sections I.B and I.D , one can arrive at the conclusion that the excitation to the model has to be proportional to the prediction error . Actually the prediction error is proportional to the energy of input signal: (This equality follows from the recursion described by equation (3.2.16.f))

$$\bar{\epsilon}_t^2 = R(0) \prod_{i=1}^p (1 - k_i^2) \quad (3.4.1)$$

The excitation should be modeled by a function which has a flat spectrum. Two candidates for the excitation signal are unit impulse (deterministic input) and white noise (random signal) for voiced and unvoiced speech respectively.

If during analysis pre-emphasis has been applied, at the output of the prediction filter post emphasis is necessary. This integration together with non-zero-mean excitation sequences can cause a low frequency thumping sound as the fundamental frequency increases (because of accumulation of input at the integrator section) The solution to this problem is an input of the form :

$$u(n) = \begin{cases} \bar{\epsilon}_t & n=0, t_p, 2t_p, \dots \\ -\bar{\epsilon}_t / (t_p - 1) & \text{else} \end{cases} \quad (3.4.2)$$

such that a zero-mean excitation is obtained.

Another approach that experimentally proved to be better is "the Chiro Function". It does not have a flat spectrum . but components at all frequencies. The one used by TMS 5220 has a 5Khz tone at the beginning and then a growing 770Hz. together with accompanying components . It continues about 4 milliseconds.

V. ENCODING

The basic parameters representing speech are energy, pitch and reflection coefficients (or another set of coefficients derivable from them). In vocoder (voice coder) applications the main consideration is to reduce bit rate when transmitting speech.

It is a common approach to encode energy and pitch period on a logarithmic basis. Typical approach is using 4-6 bits for each of them.

An experimentally observed property of k_i 's is their skew distribution. i.e. if non-pre-emphasized speech is analyzed, k_1 lies very near to -1 and k_2 very near to +1. Higher order coefficients lie within -0.7 and +0.7. This is one of the reasons why reflection coefficients are not encoded linearly. Some of the proposed methods are the following:

$$K_i = \ln((F - k_i) / (F + k_i)) \quad (3.5.1.a)$$

$$K_i = \sin^{-1}(k_i) \quad (3.5.1.b)$$

Another approach is to encode last few coefficients with few bits. The encoding scheme utilized by TMS 5100A is the following:

```
pitch....5 bits
energy...4 bits
k1-k2....5 bits
k3-k7....4 bits
k8-k10...3 bits
```

Bit reduction is obtained by not using last six reflection coefficients for unvoiced sounds. (For unvoiced sounds, last six reflection coefficient are usually near to zero . i.e. they do not decrease the prediction error as heavily . as the first four coefficients do.) A further bit reduction means is the repetity factor , which indicates that only pitch and energy values can be changing , but reflection coefficients of the previous frame may be used.

Silence is coded with energy = 0 (four bits), and no further information.

This way bit rates of 1200 bits/second can be achieved. For details refer to Appendix E.

CHAPTER 4

I. SPEECH ANALYSIS SYTEM DESCRIPTION

The block diagram in figure 3.1 describes the flow of operations during analysis of speech.

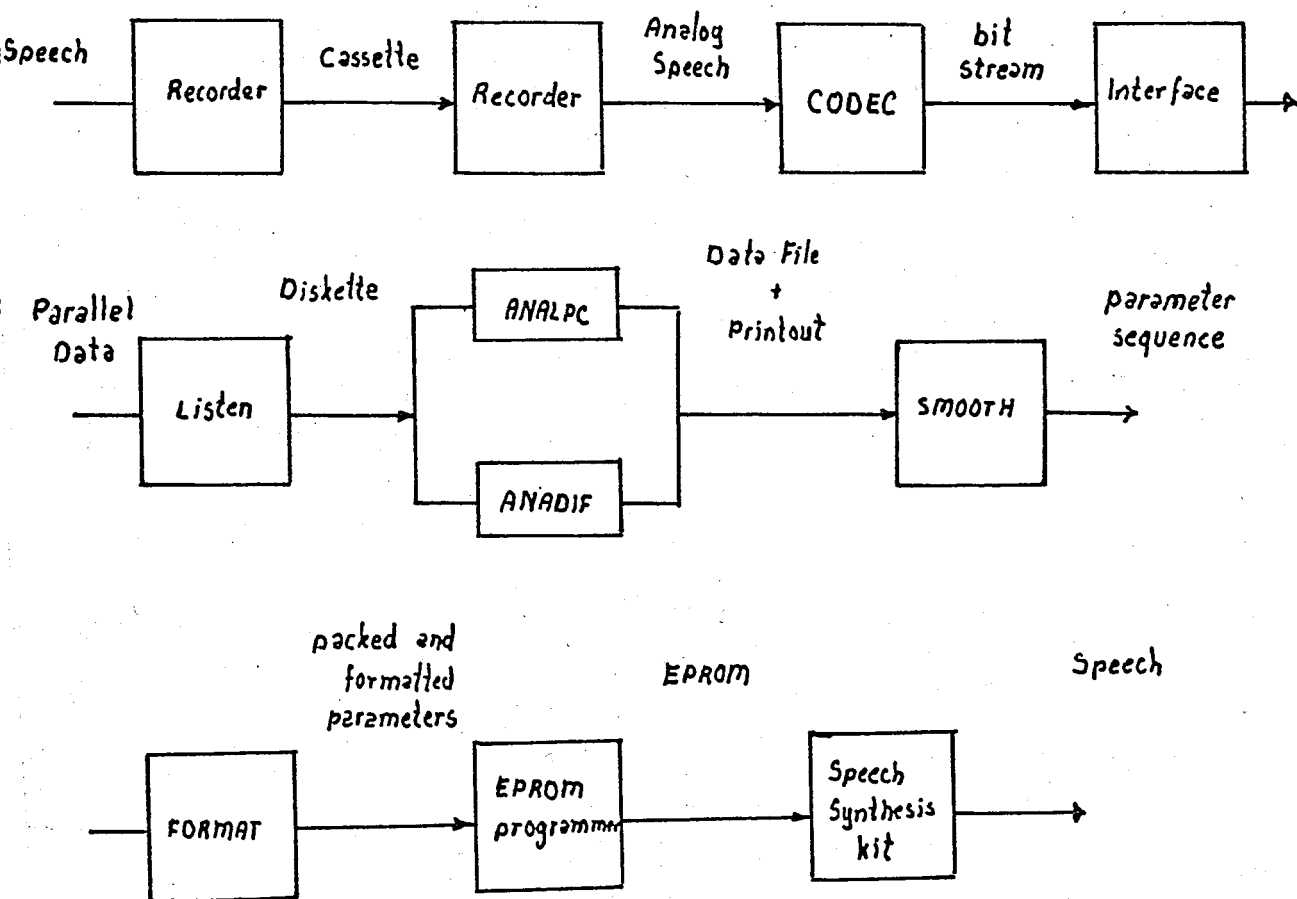


Figure 4.1.1 Analyzing System Description

For purposes of analysis, sentences consisting of various sounds are recorded to a cassette in a noiseless environment. In laboratory, analog output of cassette recorder is fed into a CODEC circuit whose output is an 8-bit serial A-law companded bit stream, representing the low-pass filtered speech samples.

The system used for analysing purposes is an INTELEC 800 MDS (Micro Development System) based on a 8080-microprocessor. Since the serial bit stream is too fast for MDS to receive, it is converted to parallel and sent to a parallel port of MDS.

A software module entitled LISTEN provides for reception of parallel speech data in synchronism with CODEC, enters received data to RAM and after collecting enough data (adjustable in multiples of 1 seconds up to 5 seconds) writes sampled data to diskette.

Another software module entitled ANALPC provides for silence/speech discrimination, pitch analysis and reflection coefficient estimation. This module reads sampled speech from the diskette to RAM, analyses it frame by frame, uses 32-bit arithmetic for coefficient estimation and writes the results to diskette. Also an optional printout is possible.

The output of this module is a data block consisting of 12 parameters (energy, pitch, 10 reflection coefficients) in encoded form. Residual energy (16bits) and $R(0)$ (in 24 bits) for each frame. If printout option is selected, a hardcopy, giving relative magnitudes of cyclic pitch measures

corresponding to pitch estimates is obtained. A similar output is obtained, when speech samples are processed by ANADIF. The only difference is, that, in the latter analysis reflection coefficients are obtained from pre-emphasized samples, when the frame is estimated as voiced.

The outputs of the mentioned modules are smoothed and scaled by SMOOTH module. a bytewise sequence of representing parameters is obtained.

FORMAT module provides for packing and formatting of the byte sequence into a compact data block, ready for transferring to EPROM.

After subjective judgement of the speech generated, the raw output of ANALPC or ANADIF modules is further smoothed and the procedure repeated.

A. External Hardware

The circuit developed for analog to digital conversion of natural speech and the interface to the minicomputer system consists of four parts:

1. Timing circuitry
2. CODEC
3. serial to parallel converter
4. parallel to serial converter

The diagram of this circuit is given in figure 4.1.2. Figure 4.1.3 summarizes timing requirements of the circuit.

1. Timing Circuitry

The oscillator circuit consisting of inverters provides a square wave at 6.144 MHz. U1 establishes a frequency division by 3. The 2.048 MHz output is fed into U2 which divides the frequency by 16. 128 kHz generated this way is divided in U3 by 16 and finally 8 kHz is obtained. One of the D-type flip-flops in 7474 is clocked by 128 kHz and generates an 16T shifted version of 8 kHz input ($T=1/2.048$ microseconds). The delayed and nondelayed versions of 8 kHz frequency are NORed to generate a pulse of 8T duration and 8 kHz frequency. This signal is inverted and applied to CE input of CODEC. The other timing input to CODEC is its Clk input, 2.048 MHz generated by U1.

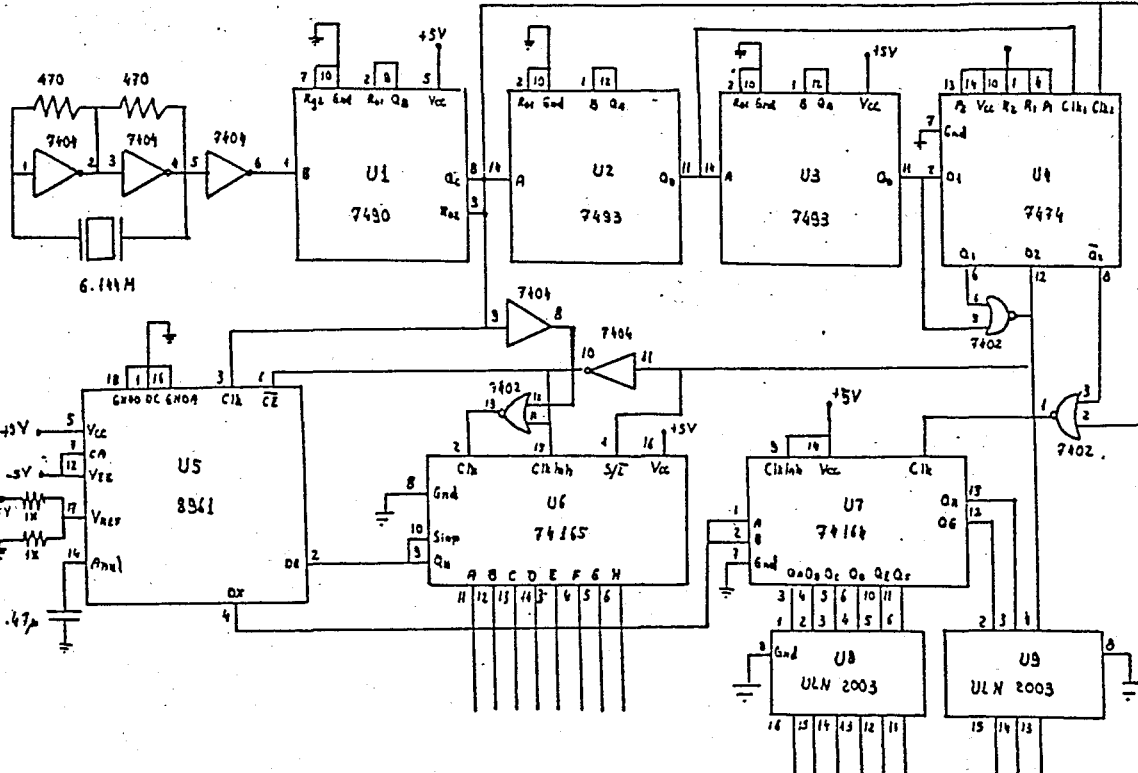


Figure 4.1.2 External Conversion Circuitry

2. 8961 CODEC

The Codec circuitry provides for low-pass filtering of sampled data (3.4 kHz bandwidth when clocked with 2.048 MHz). and also a notch filter for rejecting 50/60 Hz. A-law companding is applied to the filtered data. Starting with the sign bit an 8-bit serial bit stream is generated at every 125 microseconds, representing the analog input sampled at 8 kHz. Also the receive path includes a lowpass filter, and additionally a $\sin x/x$ correction. At the output there is a buffer amplifier capable of driving 5V peak to peak into a 10 kOhm load.

3. Serial to Parallel Conversion Circuit

This operation is accomplished by 74164. Serial bit stream is fed into 74164 inputs (both A and B inputs are ANDed in the chip). At each rising edge of its clock input received bit is shifted. After 8 clock periods 8-bit data is ready at the outputs QA-QH. Since 74164 cannot provide current necessary for input port of MDS, a driver circuit is used. ULN2003 provides for this action. In order to synchronize MDS with Codec operation a further signal is sent to MDS: a delayed version of \overline{CE} of CODEC.

4. Parallel to Serial Conversion Circuit

A 74165 provides for conversion of parallel data received from output port of MDS to a serial form suitable for 8961 CODEC. Most significant bit is connected to H. The first clock pulse shifts the bits in a circular fashion: H gives MSB and A gives LSB. This shift occurs before CODEC starts reading serial data. The same connection (delayed version of CE) provides for synchronization of CODEC and MDS.

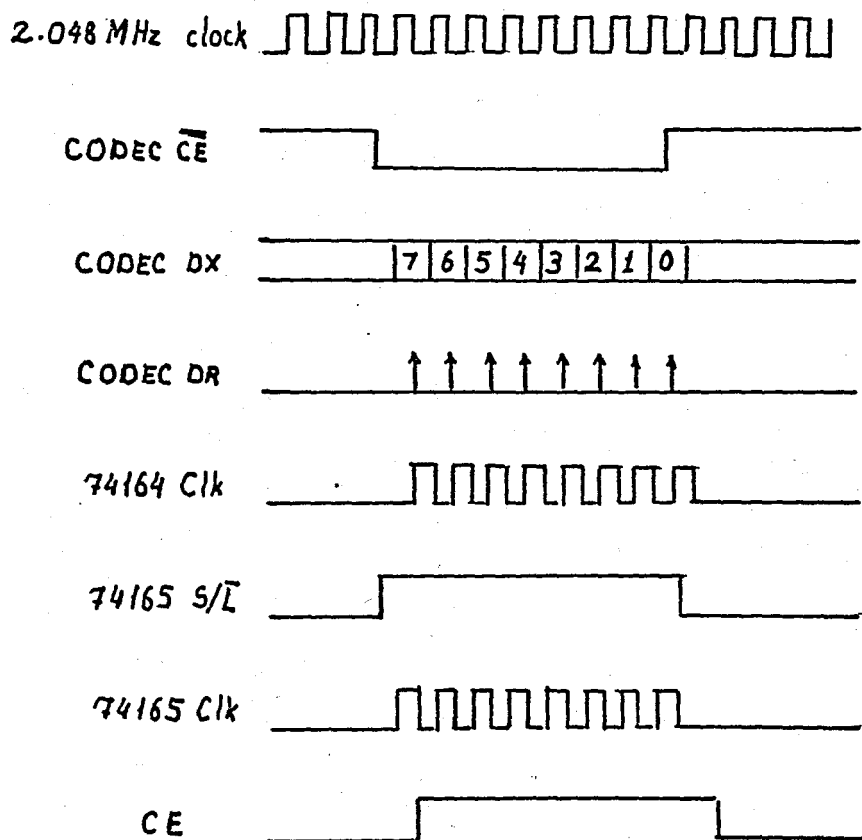


Figure 4.1.3 Timing requirements

B. INTELLEC 800 MDS I/O Ports

Monitor module of MDS provides several I/O interfaces: Those used during the work are the following:

-High speed paper tape reader/punch interface:

This port is utilized for reception of parallel data samples.

-The line printer interface

Codec-MDS synchronization is established by means of the interrupt pin of this port. For details refer to Appendix B.

C. Software Modules

1. LISTEN

This is an interactive utility program that establishes transfer of received digital data on the diskette under the file heading given by the user. A typical session is as follows:

```
>LISTEN
```

```
How many seconds ? 5
```

Numbers 1-5 are legal inputs at this stage.

Immediately after pressing 5 on the keyboard program starts receiving and placing data to RAM. When data corresponding to the duration required has been stored, reception stops and the following prompt appears:

Enter name of output file!

If the user wants to store received data on a diskette, the following is to be entered:

:FX:filename

X gives the drive number of the diskette; filename consists of 6 main-characters and 3 extension-characters part (this is a restriction of ISIS-II operating System) For example:

SPEECH.LPC

If no record is required, one has to enter :BB: in both cases carriage return should follow. After the diskette write operation is finished the following prompt appears:

Do You want to continue ?

If the user wants to continue Y should be pressed. Any other key exits the recording session.

This module consists of object files of LISTEN, SAMPLR, and USRITF files. Also PLM80, SYSTEM and 32BITS libraries should be linked together.

PLM80.LIB contains implicit functions of PLM80 language. SYSTEM.LIB contains I/O routines for the system in use and file management utilities. USRITF contains utilities that interface the user. LISTEN is the main part of the module: and SAMPLR provides for the interrupt driven, speech data reception and placement to RAM area process.

2. SHOW

This module enables the user to see the waveform of speech recorded on the diskette. Display is available as a 17x80 dot matrix. Since different monitors require different control characters to move the cursor. The user has to indicate which system he or she is using. A typical session is as follows:

Enter Filename to be displayed:

Speech.001

is this new MDS ? Y

The graph is displayed starting from the beginning of data. At the beginning each horizontal dot corresponds to 125 microseconds. and each vertical dot to 16 vertical units (vertical unit descriptive, but since data to be displayed is logarithmically companded one cannot talk about volts or millivolts). The conversion table is available in Appendix D. Acceptable commands for this module are listed below:

T is used to change time base

A is used to shift graph one time base to left
(default value of timebase is 1 milliseconds
and for amplitude base 20 units)

O is used to change the origin

< is used to shift graph one time base to left

> is used to shift graph one time base to right

E to exit the display session

Object files of SHOW, GUTILY, USRITF, SYSTEM.LIB and PLX80.LIB constitute this module. SHOW is the main part and GUTILY is the file containing display utilities.

3. ANALPC

This is the main module that is responsible for

- 1) Conversion of 8-bit A-law companded data to 12-bit linear data
- ii) Silence-speech discrimination
- iii) Voiced-unvoiced decision and pitch period calculation
- iv) Calculation of reflection coefficients
- v) Scaling pitch, energy and reflection coefficients

The flow of this module is summarized in figure 4.3.1.

- 1) A-law expansion: Since all algorithms implemented require linear data companded, speech is converted to linear data. A-law compansion has precisely the following form:

$$\begin{aligned}
 & \text{sgn}(x) (A|x| / (1+\ln(A))) \\
 & \text{for } 0 < |x| < 1/A \\
 F(x) = & \text{sgn}(x) (1+\ln(1/A|x|)) / (1+\ln(A)) \\
 & \text{for } 1/A < |x| < 1
 \end{aligned}
 \tag{4.3.1}$$

In order to facilitate conversion, a table can be used. The table for conversion between 8-bit to 12-bit samples is given in Appendix D.

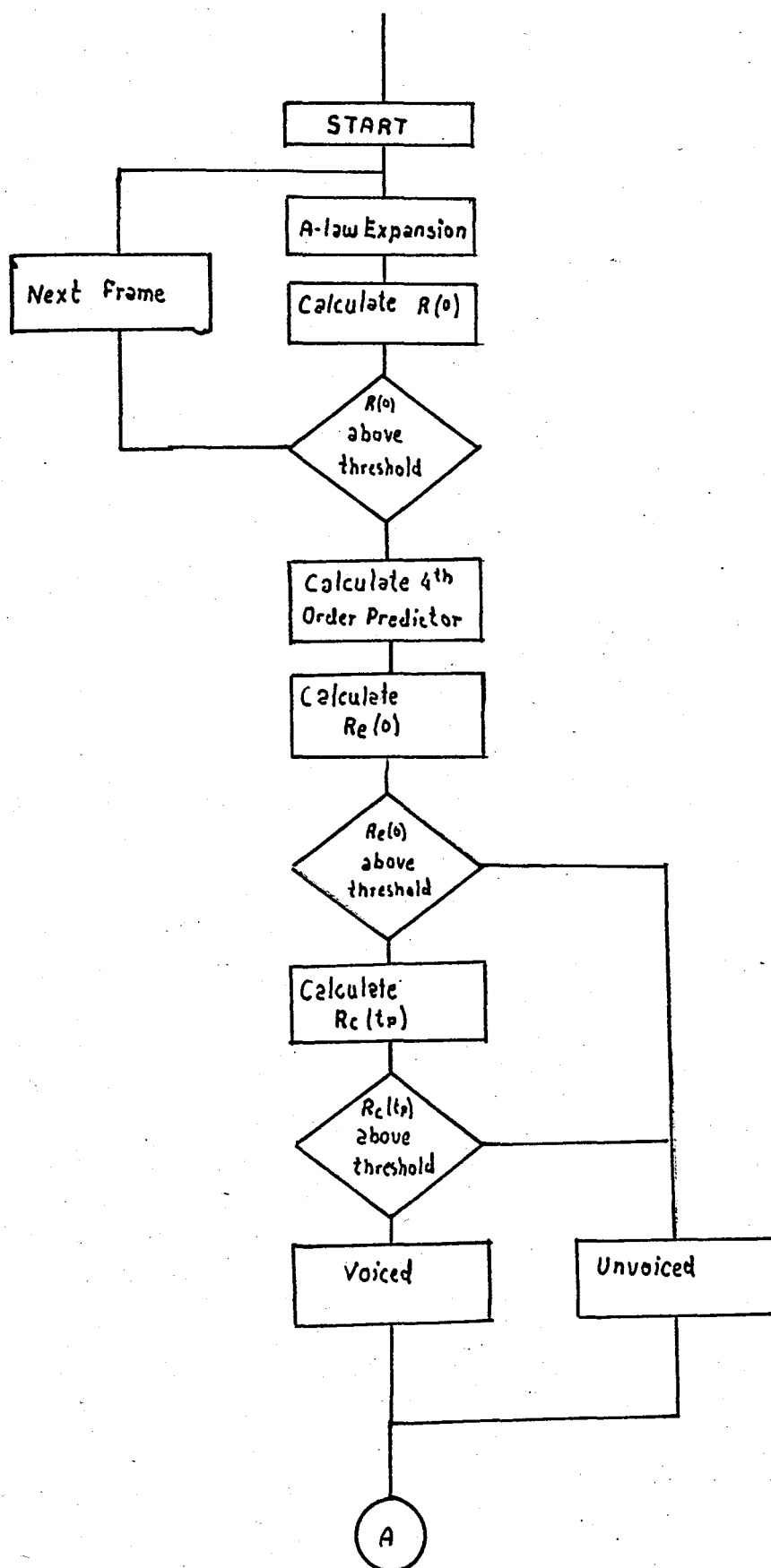
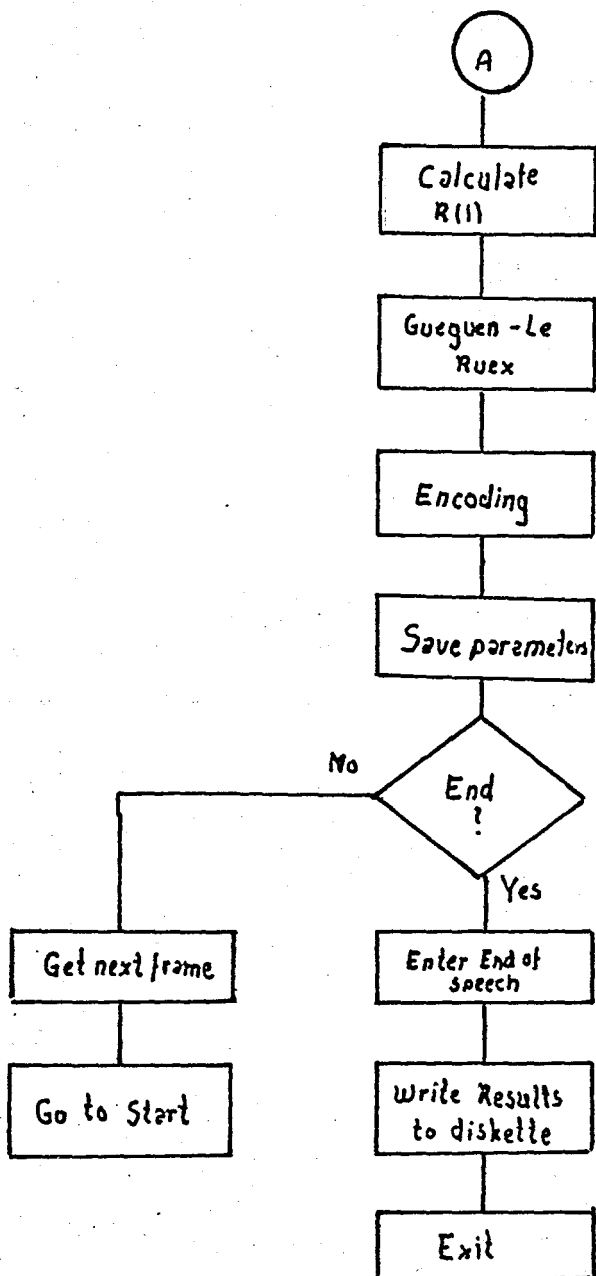


Figure 4.1.4 Flowchart of ANALPC



Maximum voltage used amounts to 2.5 Volts. An acceptable conversion table for voltage to 7-bit magnitude representation is given in table 4.3.1. During companding data is scaled such that 2.5V corresponds to 0FH=15 and 156 mV corresponds to 1. Although conversion to only 12-bits is required, the output of A-law expansion is a sequence of 32-bit samples .

1.25 - 2.50 V	70H-7FH
0.625- 1.25 V	60H-6FH
312 - 625 mV	50H-5FH
156 - 312 mV	40H-4FH
78 - 156 mV	30H-3FH
39 - 78 mV	20H-2FH
19.5 - 39 mV	10H-1FH
10 -19.5 mV	0H-0FH

Table 4.1 Voltage versus 8-bit A-law companded data

11) Silence Speech discrimination:

This decision is not made explicitly. For the ongoing analysis calculationsof $R(0)$ is required. This is done as follows:

A frame consists of 200 samples and corresponds to 25 milliseconds. But for pitch analysis 400 samples (200 samples belong to next frame) are utilized. One out of four samples are taken for the pitch analysis (speech signal is decimated). After passing decimated data through a Hamming window, sum of squares of samples is calculated ($R(0)$). If this value does not exceed 0.43, this frame is justified as silence. This ratio has been found empirically and is dependent on the environment where speech has been recorded

and where it is played back.

111) Voiced-Unvoiced decision and pitch period calculation:

The linearized speech samples are decimated by taking one out of four samples and preemphasized. Preemphasis is implemented by taking difference of consecutive data ($s(n) = s(n+1) - s(n)$). Preemphasized data is passed through a window and first five autocorrelation coefficients are calculated. These coefficients are passed to the Gueguen - Le Ruex algorithm and first four reflection coefficients are obtained. The predictor coefficients are calculated by the STEPS procedure. The spectrum flattener obtained this way is actually a fourth order predictor and a sequence of prediction errors comes out. This sequence is passed through a Hamming window and cyclic pitch measures corresponding to pitch frequencies in the range of 6 milliseconds to 11.5 milliseconds are calculated. For normalization purposes also $R_e(0)$ is calculated and $R_c(t_p)$ s are normalized. Negative values are truncated to zero. If more than 10 of 13 measure coefficients are nonzero, the frame is declared as unvoiced. Similarly, if the maximum normalized $R_c(t_p)$ is below 0.375, the frame is accepted as unvoiced. (Also this value is obtained by trial and error.) Otherwise t_p value corresponding to $\max\{R(t_p)\}$ determines the pitch period.

iv) Calculation of reflection coefficients:

For each frame first 11 autocorrelation coefficients are calculated using 200, windowed, linearized samples. These coefficients are passed to Gueguen Le Ruex algorithm, where 10 reflection coefficients calculated. Throughout the process sum of prediction errors is computed recursively. At the end also sum of squares of prediction error is implicitly obtained. If throughout the analysis the recently calculated reflection coefficient exceeds one in magnitude, no further values are calculated and those from the previous frame are used. This situation is marked by a * in the printout.

v) Energy Calculation and Encoding

Residual energy (sum of squares of prediction error) is used to calculate energy. It is normalized by 125 and encoded according to the conversion table in Appendix E. Similarly, pitch and reflection coefficient values are encoded using the same conversion table.

ANALPC module consists of ANALPC, ANALGR, USRITF, ENCODE, CONVAK, DCDTBL, WINDOW and ALAW object files. Additionally SYSTEM PLM80 and 32BITS libraries are linked together. ANALPC is the main module. ANALGR provides for pitch and ANALGR for Gueguen Le Ruex algorithm. ALAW file establishes A-law expansion of recorded speech samples. DCDTBL contains conversion tables for reflection coefficients. ENCODE file uses this table and provides for encoding.

ANADIF

This module differs from ANALPC only in the calculation of reflection coefficients. If frame has been judged as voiced during the pitch analysis, speech samples are preemphasized before calculating first 11 autocorrelation coefficients.

Instead of object file of ANALPC, object file of ANADIF is linked with those file mentioned for ANALPC.

5. SMOOTH

This module is used to correct errors made in predicting the parameters, such as energy, pitch or reflection coefficients. During a smoothing session the user can do the following :

FxxEy Change energy coefficient of frame xx to y

FxxPy Change pitch value of frame xx to yy

FxxkYzz Change k of frame xx to zz

FxxRy Change repeat bit of frame xx to y

Dxx Display frame xx. The following information is displayed:

frame number

energy

pitch

reflection coefficients

sum of squared prediction errors (2 bytes)

R(0) (3 bytes)

Sxxyy Scale energy , by dividing sum of squared error entry by the given number yy.xx (yy gives integer part and xx the fractional part)

Rxxyy Remove frames xx to yy (yy included)

ixx Insert a new sound , starting after frame xx. After having received this instruction , name of the file containing sound to be inserted is requested by the system:

File to be inserted:

The user has to enter the file name and press carriage return.

C Create a file , containing only a sequence of parameters (four reflection coefficients for unvoiced frames, only energy parameter for silence and only energy, repeat bit and pitch for repeated frames) The system first asks , whether a printout is required:

Do You want to take printout?

Y is pressed for positive answer. If any other key is pressed no printout is created , but the frames are displayed on the screen.

Then the system requests for for the output file name:

File name of smoothed data:

After the user has entered file name and pressed carriage return , the new file is created. For no file output :BB: should be entered as file name.

- EQ Exit smoothing session, without saving recently made modifications
- EX Save last modifications and exit to ISIS-II operating system. The system requests for the name of file . where modified data block has to be written into:

File name of smoothed data:

After the name is entered and carriage return is pressed, save operation follows and control returns to ISIS-II.

6. FORMAT

The output of SMMOTH module is a sequence of bytes each representing one of the parameters necessary. But TMS-5100 speech synthesiser chip, that has been utilized for subjective judgement, requires this information in a packed form. e.g.

Energy	4H = 0100B
Repeat	0H = 0B
Pitch	0H = 00000B
K1	13H = 10011B
K2	EH = 01110B
K3	9H = 1001B
K4	7H = 0111B

Should be combined to

```
00000010
01100100
10010111
  1110
```

i.e. the bits are interchanged such that MSB becomes LSB and vice versa and then packed, such that first byte fills the

least significant bits.

FORMAT module also formats packed LPC data suitable for transferring to EPROM programmer. The resultant data packet consists of lines each starting with the line indicator character " : ", number of bytes in the frame, frame number in multiples of 10H, bytes to be transmitted in this frame and finally the checksum. A sample output is given in figure 4.1.5. Since 0FH for energy means end of phrase program searches until a 0FH is encountered when reading energy. Up to end of phrase all data is packed.

```
:1000000000000006444215519A9D6EC9C32469D9EFF
:10001000D10467142739A309AE6C4E7266935CDE77
:10002000DAF4ACAA8BA22DD901CAD50410A9770177
:100030008D68259048B40393723A3039A7014388FC
:100040002B309904038D7000003CFFFFFFFFFFFFFFF82
:000000001FF
```

Figure 4.1.5 Hex file generated by the FORMAT program for the word "Bir".

A typical session is as follows. Program requests for input filename:

Filename for LPC data:

BIR.LPC

Then asks name of output file containing formatted data

Filename for formatted data:

BIR.HEX

Immediately after pressing carriage return , program starts to packing and formatting the data and returns to ISIS-II when finished.

This module consists of object files of FORMAT, PACK, USRITF and SYSTEM.LIB. FORMAT is the main file and PACK an ASM80 file establishing the required packing operation.

CHAPTER 5

I. RESULTS

The tools developed during this study have been applied to Turkish sounds. Phonemes alone cannot be generated during regular speech, so syllables have to be chosen. The syllables chosen are the ones constituting the alphabet (i.e. A, BE, CE.....) and numbers from one to ten. Also a sentence has been analysed. The results of the analysis are available in Appendix G.

Throughout this study different algorithms have been tested. In the following sections they are compared with each other and suggestions for further algorithms are developed.

After each analysis I had the opportunity to judge subjectively the results (speech output). These observations are summarized in part B.

A. Computational Aspects

Throughout the whole analysis all of the results are kept as 32 bit twos complement binary and fixed point arithmetic is used. During the analysis of a frame the following operations are performed:

- Four hundred samples are expanded to 32-bits according to A law.

- decimated by taking one out of four samples and
- reduced to 100 samples.
- windowed (100 multiplications) .
- preemphasized (100 multiplication and additions) .
- first five autocorrelation coefficients calculated (500 multiplication and 500 additions) .
- fourth order predictor's coefficients computed (80 additions and 40 multiplications) .
- prediction error sequence computed (400 addition and multiplications) .
- $R_e(0)$ calculated (100 multiplications and 100 additions).
- cyclic pitch measures for 13 different period estimates computed and compared (1300 multiplication and 1300 additions).
- 200 samples are expanded according to A law .
- preemphasized (optional) (200 additions) .
- windowed (200 multiplications) .
- first 11 autocorrelation coefficients computed (1100 multiplications and 1100 additions) .
- tenth order filter reflection coefficients computed (210 multiplications and 400 additions)

The operations add up to about 5000 multiplication and additions there are also other instructions such as decisions , search loops, move operations that occur during the analysis of a frame. The analysis of a sentence of about 2.5 seconds takes about half an hour (18 seconds for a frame) But this is an average value because analysis of silent

frames and unvoiced frames may take considerably less time.

During this work no effort has been spent , to speed up the computation. But the following changes would reduce the required time by a considerable amount:

Application of efficient autocorrelation calculation methods that reduce the number of multiplications.

Use of more memory such that results , once calculated, and later to be used again , can be saved and not calculated twice.

Use of 32-bit operations only during computation of autocorrelation coefficients. For other computations such as windowing, preemphasize and recursive solution 24-bit or less would suffice.

A further reduction of operation count (or increase in accuracy , while the number of operations remain the same) can be established by using a sequential lattice method. Sequential methods do not require windowing, and they generate prediction error sequence at the same time. So production of this sequence as a byproduct , eliminates the necessity of extra calculation required for pitch analysis . This way the number of multiplications would be highly reduced. Using a digital signal processor with a very low cycle time (e.g. 200 nanoseconds for 32-bit multiplication) real time applications can be developed. Such a device can manipulate 125 000 multiplications during a 25 milliseconds frame.

B. Subjective Aspects

Before Gueguen-Le Ruex , Levinson-Durbin algorithm has been used to extract reflection coefficients from input speech. Although the precision of operations are the same , Levinson - Durbin algorithm resulted in reflection coefficients, that are greater than one in magnitude. For voiced speech this occurred at predictors of order eight to ten , for unvoiced speech for orders four to five. But Gueguen - Le Ruex always ended up in stable outputs.

It is interesting to note that , even when the first reflection coefficient was calculated correctly , persons, who knew , what the coming utterance should be , have been able to recognize it. This case clearly outlines the importance of the first reflection parameter.

Excitation with a constant pitch value of 100Hz does not result in a monotone sound. Apparently it is the energy parameter, that provides for the intonation in a sentence . Pitch period variations add only naturalness to speech.

Preemphasis , that has been applied to speech samples is actually a form of differentiation. This enhances upper portion of spectrum. It is an accepted idea in literature , that upper part of spectrum contains the main intelligibility.

During pitch analysis several thresholds have been arranged by trial and error. A set of thresholds , that can produce very good results in pitch detection and voicing

decision for one case . can result in erroraneous voiced/unvoiced decisions for another. But since the autocorrelation function used in SIFT, requires various thresholds for pitch candidates, the pitch measure, introduced in this work , results in less erroraneous outputs. If voiced frames are judged as unvoiced and such frames alternate among voiced frames the sound is like a burbling.

Another problem that has been encountered during the work is the energy parameter. First $R(0)$ (sum of squares of speech samples) is scaled and applied as energy. This does not produce intelligible voice. Then residual energy (sum of squares of prediction errors) is scaled and applied as energy parameter. This resulted in a good speech for a set of records. But for a second set of records the scaling factor has been too small. The result was , that a large number of frames had the maximum allowable energy. The speech output in this case is a distorted one.

The solution to the energy scaling problem is the interactive SMOOTH module. The user can process the raw output data block of ANALPC and ANADIF modules. The outputs of these modules include the residual , that has been scaled by the default value.

Both analysis modules ANADIF and ANALPC are essentially similar. They only differ in the preprocessing before reflection coefficient computation. ANADIF module differentiates input speech, before windowing. This also is based on the idea to enhance the information hidden in the

high frequency part of speech.

Without preemphasis (differentiation) vowels such as /i/ , /o/ , /o/ , /u/ , /u/ resulted in very distorted and unrecognizable sounds when analyzed alone. Also /a/ sounded like an /e/. But when accompanied by consonant , in a word, they resulted in recognizable sounds. But even if followed by consonants , /a/ at the beginning of an utterance sounds like an /e/.

ANADIF module applies preemphasis to all samples in the frame . if the frame has been judged as voiced by the pitch estimation program. The speech parameters generated by this module resulted in a higher quality speech. The results outlined in Appendix F are the outputs evaluated by this module. But they are not raw data , but smoothed data.

The output of this module can be improved by first smoothing voicing decisions and then evaluating reflection coefficients. This requires, that, first a pitch detection algorithm process all frames , then voicing decisions are smoothed and later all frames are processed again to extract reflection coefficients.

An important property of reflection coefficients observed during this work is the following: Although there may be large variations between two sets of reflection parameters , the prediction coefficients corresponding to them differ very little. This effect has been seen, by once generating synthetic speech , and then analysing it using ANALPC.

An experimentally observed property of reflection coefficients is their skew distribution for the first two coefficients. First one lies very near to minus one, while the second one lies very near to one. This distribution is true for non-preemphasized speech. Preemphasis has the effect to remove them from one and minus one respectively. This result can be interpreted in the following way: Without preemphasize the prediction error is decreased quickly by the first two coefficients, but since in preemphasized signal higher frequencies are enhanced, the prediction at the second stage does not decrease prediction error by the same amount.

It has been expected that differentiating the signal would result in different signal and error energies. But the outcome of both ANALPC and ANADIF modules have very similar energy levels.

An interesting observation is that, vowels alone cannot be recognized by everyone when produced using the results of analysis. But in a single word they are recognized. Listeners mostly recognized single word utterances. The same utterance in a sentence is only seldom recognized.

CHAPTER 6

I. CONCLUSION

This work establishes a guideline for analysing connected or isolated speech. The algorithms developed can be improved by implementing one or more of the suggested techniques in Chapter 5. The listings of modules are given Appendix F and the analysis results in Appendix G.

With this study it has been shown that reflection coefficient representation is an efficient way of representing speech. But throughout this study success has been limited to single words. Connected speech, generated using this approach is not always intelligible. Further improvement of the algorithms, such that interactions between sounds are taken into consideration, can lead to better results.

A. Future Applications

A straightforward application of this study is to analyse messages and store them in memory for using them in future. One of the possible implementations is, to use messages instead of overflow tones in common communication equipment such as an EPABX (Electronic Private Automatic Branch Exchange). It is not easy for every subscriber to understand why his or her call did not terminate

successfully. just by listening to the tone sent.

This work can also be used to generate the allophones of Turkish language. The interactions between consecutive sounds can be deeply explored (some outputs outlining this interaction are given in Appendix F) and a text-to-speech algorithm can be developed.

Text-to-speech synthesis can also be implemented using syllables that constitute Turkish words, by first analyzing and storing them on a disc. Speech can be generated by either concatenating syllables or allophones.

APPENDIX A

32-BIT ARITHMETIC

Since Intel 8080A microprocessor can only perform 16-bit addition and no multiplication and division, these should be produced separately. Therefore a 32-bit fixed point arithmetic library is created.

32-bit data is stored in memory in the following way:

```
base address + 0 : B0
base address + 1 : B1
base address + 2 : B2
base address + 3 : B3
```

B0 is the least significant byte and B3 is the most significant byte. Most significant bit of B3 determines sign (0 means positive). Negative numbers are stored in 2's complement form. B0 and B1 correspond to the fractional part and B2 and B3 to integer part. For example 352.4 is represented as:

```
B0 = 66H
B1 = 66H
B2 = 60H
B3 = 01H
```

similarly -352.4 is represented as:

```
B0 = 9AH
B1 = 99H
B2 = 9FH
B3 = FEH
```

The operations are performed by the following subroutines:

```
For addition      : FBADD      (C = A + B)
For subtraction  : FBSUB      (C = B - A)
For division      : FBDIV      (C = A / B)
For multiplication: FBMUL      (C = A * B)
```


They have the following general format of usage:

CALL FBxxx(address of A, address of B, address of C)

The listings of these modules are given in Appendix F .

APPENDIX B

INTELLEC 800 MDS HARDWARE

The INTELLEC MDS is a complete microcomputer design center based on Intel's 8080 Microprocessor. It has a 2-microseconds instruction cycle time. The basic hardware configuration includes 16K bytes of Random Access Memory and six fully implemented I/O interfaces to :

- a Teletype (including its paper tape reader)
- a CRT terminal (or other compatible device)
- a high-speed paper tape reader
- a high-speed paper tape punch
- a line printer, and
- Intel's Universal PROM Programmer

Figure B.1 describes the block diagram of the basic configuration used for this work.

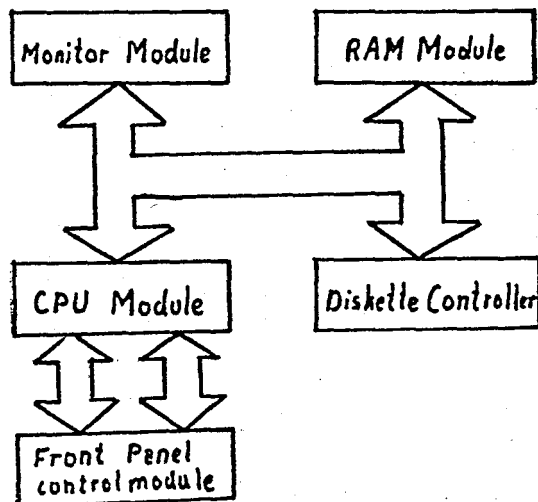


Figure B.1 Block diagram of INTELLEC 800

Monitor module provides for the communication of Intellec 800 MDS system with the external world. Speech storage is established using the interface circuits of this modules. Some of the interfaces are the following:

The high-speed paper tape reader/punch interface provides an 8-bit data input path, two command bit outputs and a single ready status input for a 200 cps paper tape reader, and an 8-bit data output path, two command bit outputs and a single ready status input for a 75 cps paper tape punch.

The PROM programmer interface provides an 8-bit path to PROM programmer peripheral for the transfer of data, address, control and status information. It also presents the necessary commands to the PROM programming peripheral specifying the direction and the type of transfer.

The line printer interface has been designed to operate with printers which are capable of receiving input commands as coded ASCII characters in the same manner as data.

The monitor interrupt logic provides a hardware scheme that can be utilized by a user-generated operating system or specialized application routine to service I/O devices.

Each time a device indicates that it is ready, the appropriate service request signal is generated. The service request lines form a 7-bit interrupt status word which can be read by CPU. In addition if Monitor module interrupts are enabled, an active service request results in an interrupt.

request to CPU on level 3.

Monitor Module Interrupts can be enabled or disabled, and device service requests can be reset individually or as a group by executing a single output instruction.

Addresses of I/O ports are 0F0H - 0FBH :

PTP DAY	0FBH
PT CTL	0F9H
PTR DAY	0FBH
PT STAT	0F9H

A low to high transition on the LPT ACK pin causes an interrupt of level 3. This corresponds to Line Printer Acknowledge information (used to establish synchronization between MDS and CODEC during speech storage)

Some of the important addresses related to the monitor module are the following:

0F3H.....Monitor Interrupt Mask
1 corresponds to enabled

I	I	I	I	I	I	I	I	I		
I	I	I	I	I	I	I	I	I		Monitor interrupts
I	I	I	I	I	I	I	I	I		Line printer
I	I	I	I	I	I	I	I	I		CRT input
I	I	I	I	I	I	I	I	I		CRT output
I	I	I	I	I	I	I	I	I		TTY output
I	I	I	I	I	I	I	I	I		TTY input
I	I	I	I	I	I	I	I	I		PTP output
I	I	I	I	I	I	I	I	I		PTR input

0FAH.....Interrupt Status Word
entries correspond to peripherals is explained above. 1 in an entry means :
respective interrupt has occurred.

OFCH..... Interrupt Mask
 1 corresponds to disabled

I	I	I	I	I	I	I	I	I	I		
I	I	I	I	I	I	I	I	I	I	Level 0	
I	I	I	I	I	I	I	I	I	I	Level 1	
I	I	I	I	I	I	I	I	I	I	Level 2	
I	I	I	I	I	I	I	I	I	I	Level 3	
I	I	I	I	I	I	I	I	I	I	Level 4	
I	I	I	I	I	I	I	I	I	I	Level 5	
I	I	I	I	I	I	I	I	I	I	Level 6	
I	I	I	I	I	I	I	I	I	I	Level 7	

I/O Device Driver/Receiver Requirements

Paper Tape Reader : 7437 TTL drivers capable of sinking 48mA
 8097 TTL receivers without pull-up

Paper Tape Punch : 8097 TTL drivers capable of sinking 32mA
 8097 TTL receivers with 470 Ohm pull-up
 resistors

PTR edge connector

Data	Pin
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
Gnd	11, 12, 13, 24, 25

PTP edge connector

Data	Pin
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
Gnd	18, 23, 25

LPT

Ack	16
Gnd	15, 25

APPENDIX C

ISIS-II DISKETTE OPERATING SYSTEM

ISIS-II is an operating system for the Intellec development systems:

Intellec Series II Microcomputer Development Systems
Intellec Microcomputer Development System

It supports all models of Intellec systems with one or more single- or double density disk drives and at least 32K of RAM.

An operating System is a group of programs that manages the resources of a system and frees the user to concentrate on other tasks. It provides a simple command language that allows the user to state what he or she wants done and what device or file is expected to do it.

The computer has to execute thousands of instructions to carry out each command, but the user has not to worry about the details.

A brief list of commands available in ISIS-II is given below:

IDISK format a new disk to a basic system or non-system disk

DEBUG load a program and give control to monitor

SUBMIT enter the file that contains commands to be executed

DIR output the names of and information about the files listed within the disk directory

DELETE remove references to a file from the directory and free disk storage space associated with that file

RENAME change the name of a disk file

OBJHEX convert a program from object module to hexadecimal format

CREDIT create and edit ISIS-II files

COPY copy a file from one device to another

LINK combine program files and resolve external addressing

LOCATE convert relocatable object to absolute addresses for execution

Several compilers and assemblers are also available:

ASM80	8080/85	Assembler
ASM86	8086/88	Assembler
PLM80	8080/85	PLM compiler
PLM86	8086/88	PLM compiler

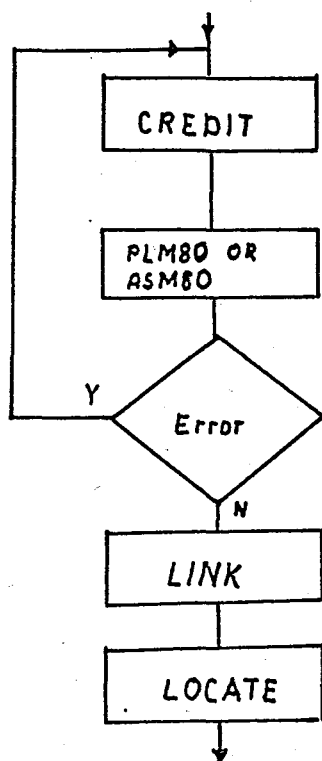


Figure C.1 Command generation

User Generated Commands

Actually commands of any operating system are linked and located object codes , that are suitable for the execution of the main processor. the system is based on. For Intellec 800 MDS. object codes of ASM80 and PLM80 compilers are suitable for execution. The Process can be summarized as in figure C.1.

Some of ISIS-II utilities necessary for file management and interactive command execution are given below. They reside in SYSTEM.LIB . During LINK operation only those, that are used are linked together.

OPEN(AFTNPTR, FILEPTR, ACCESS, ECHOAFTN, STATUS)

OPEN call initializes ISIS tables and allocates buffers that are prepared for input/output processing of the specified file.

AFTNPTR : Active File Table Number Pointer

FILEPTR : Address of the ASCII string that specifies the name of the file to be opened

ACCESS : indicates how the file is to be accessed when open.

- 1...open for input(READ) Marker=0 , Length not changed
- 2...open for output(WRITE) Marker=0 , Length=0
- 3...open for update

ECHOAFTN: contains AFTN of the echo-file , if line editing is desired , otherwise contains zero

STATUS : memory location for the return of non-fatal error number

READ(AFTN, BUFFER, COUNT, ACTUAL, STATUS)

The READ call transfers data from an open file to a specified memory location specified by the calling program.

AFTN : number assigned to the file

BUFFER : address of memory location to which data is to be transferred

COUNT : number of bytes to be transferred

ACTUAL : number of bytes actually transferred

STATUS : memory location for the return of non-fatal error number

WRITE(AFTN, BUFFER, COUNT, STATUS)

The WRITE call transfers data from a specified location in memory called a buffer to an open file.

AFTN : number assigned to the file
 BUFFER : address of memory location to which data is to be transferred
 COUNT : number of bytes to be transferred
 STATUS : memory location for the return of non-fatal error number

CLOSE(AFTN, STATUS)

The CLOSE call removes a file from the system input/output tables and releases the buffers allocated for it by OPEN.

AFTN : number assigned to the file
 STATUS : memory location for the return of non-fatal error number

EXIT

The EXIT call can be used by user programs in order to terminate their execution and return to ISIS-II

CO(CHARACTER)

Console Output Routine takes a single character and transmits it to the system console output device

CHARACTER: ASCII character to be displayed on the CRT

LO(CHARACTER)

List Output Routine takes a single character and transmits it to the system list device

CHARACTER: ASCII character to be printed

CI

The Console Input Routine reads a character entered at the Inteltec Console input device and returns it as a byte variable (in A-register)

Memory Allocation of ISIS-II

Total RAM requirement of the system is 32 Kbytes. The first 12Kbytes (starting at location 0) are reserved for use by that portion of ISIS-II, that must remain in memory during operation of disk-based functions of the system. An additional 318 bytes of higher order memory is reserved for Monitor workspace. The remaining memory is shared by the program being developed, by programs called by ISIS-II functions from diskette, and by the data used or generated during user program development.

Locations 0-23 : ISIS-II Interrupts 0,1,2

Locations 24-63 : User interrupts

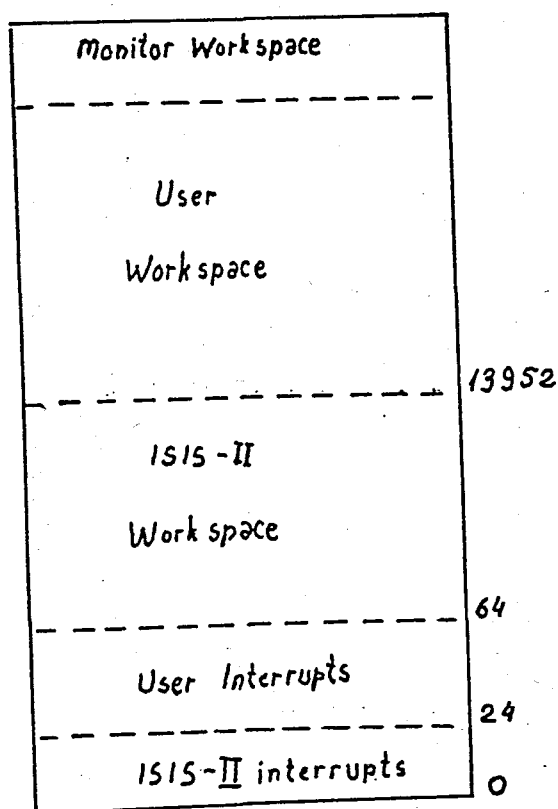


Figure C.2 Memory Map for ISIS-II

APPENDIX D

A-LAW EXPANSION

In order to provide for a wide dynamic range by using few bits to represent the signal companding is applied. The approach advised by CCITT is A law companding. The equations describing compansion and expansion of the signal are the following:

$$\begin{aligned} & \text{sgn}(x) (A|x| / (1+\ln(A))) \\ & \text{for } 0 < |x| < 1/A \\ F(x) = & \text{sgn}(x) (1+\ln(|Ax|)) / (1+\ln(A)) \\ & \text{for } 1/A < |x| < 1 \end{aligned} \quad (D.1)$$

$$\begin{aligned} & \text{sgn}(y) (|y| (1+\ln(A))) / A \\ & \text{for } 0 < |y| < 1/(1+\ln(A)) \\ F(y) = & \text{sgn}(y) (e^{|y|(1+\ln(A))} - 1) / A \\ & \text{for } 1/(1+\ln(A)) < |y| < 1 \end{aligned} \quad (D.2)$$

The segmented A law code is usually referred to as a 13 segments code owing to the existence of seven positive and seven negative segments with the two segments near the origin being colinear. In the given tables the first segment is divided into two parts to produce eight positive and eight negative segment. Thus a compressed code word consists of a sign bit P followed by 3 bits of a segment identifier S and 4 bits of quantizer level Q .

I	linear data	I	expanded data	I
I	0 0 0 w x y z	I	0 0 0 0 0 0 0 w x y z 1	I
I	0 0 1 w x y z	I	0 0 0 0 0 0 1 w x y z 1	I
I	0 0 0 w x y z	I	0 0 0 0 0 1 w x y z 1 0	I
I	0 0 1 w x y z	I	0 0 0 0 1 w x y z 1 0 0	I
I	0 0 0 w x y z	I	0 0 0 1 w x y z 1 0 0 0	I
I	0 0 1 w x y z	I	0 0 1 w x y z 1 0 0 0 0	I
I	0 0 0 w x y z	I	0 1 w x y z 1 0 0 0 0 0	I
I	0 0 1 w x y z	I	1 w x y z 1 0 0 0 0 0 0	I

Table D.1 A law expansion

APPENDIX E

CONVERSION TABLE

RMS	Pitch	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	Cod
0	0	-501	-328	-441	-328	-328	-256	-308	-256	-256	-205	0
52	25	-498	-303	-387	-273	-282	-212	-260	-161	-176	-132	1
87	27	-497	-274	-333	-217	-235	-168	-212	-66	-96	-59	2
123	28	-495	-244	-279	-161	-189	-123	-164	29	-15	14	3
174	30	-493	-211	-225	-106	-142	-79	-117	124	65	87	4
246	31	-491	-175	-171	-50	-96	-35	-69	219	146	160	5
348	32	-488	-138	-117	5	-50	10	-21	314	226	234	6
491	34	-482	-99	-63	61	-3	54	27	409	307	307	7
694	35	-478	-59	-9	116	43	98	75				8
981	36	-474	-18	45	172	90	143	122				9
1385	37	-469	24	98	228	136	187	170				10
1957	38	-464	64	152	283	182	232	218				11
2764	39	-459	105	206	339	229	276	266				12
3904	40	-452	143	260	394	275	320	314				13
5514	42	-445	180	314	450	322	365	361				14
7789	44	-437	215	368	506	368	408	409				15
	46	-412	248									16
	48	-380	278									17
	50	-339	306									18
	52	-288	331									19
	54	-227	354									20
	56	-158	374									21
	58	-81	392									22
	61	-1	408									23
	63	80	422									24
	66	157	435									25
	70	226	445									26
	75	287	455									27
	80	337	463									28
	85	379	470									29
	90	411	476									30
	95	436	506									31

Each table entry of K1 to K10 is expressed as a quotient of integers with the denominator value fixed at 512.

Pitch period table entry is the pitch period for voiced excitation expressed as number of samples. For 8KHz sampling rate and a pitch period of 100Hz pitch period corresponds to 80 samples i.e coded value is 128.

APPENDIX F

PROGRAM LISTINGS

The following files are included:

File name	Page
ANALPC.SRC.....	99
ALAW .SRC.....	104
ANALGR.SRC.....	107
ENCODE.SRC.....	111
DCDTBL.SRC.....	114
SAMPLR.SRC.....	118
LISTEN.SRC.....	121
FORMAT.SRC.....	123
PACK .SRC.....	127
GUTILY.SRC.....	131
CONVAK.SRC.....	137
USRITF.SRC.....	140
ANAPTC.SRC.....	144
ANADIF.SRC.....	149
SMOOTH.SRC.....	153
SHOW .SRC.....	162
WINDOW.SRC.....	166
SPALUO.SRC.....	171
SPALU1.SRC.....	173
SPALU2.SRC.....	179

PLS-II PL/M-80 V3.0 COMPILATION OF MODULE MAIN

JECT MODULE PLACED IN :F1:ANALPC.OBJ

MPILER INVOKED BY: PLM80 :F1:ANALPC.SRC WORKFILES(:FO:..:FO:)

\$PAGEWIDTH(80)

```

/*****
**
**          SPEECH SYNTHESIS SYSTEM
**          MAIN MODULE
**
**  FILE NAME      : ANALPC.SRC          DATE
**  MODULE DEF     : Calls LPC Analysis -----
**  AUTHOR         : Onder Bicioglu    22 06 85
**
*****/

```

MAIN: DO:

```

1
2  1  DISPLAY$MESSAGE: PROCEDURE(POINTER) EXTERNAL:
3  2    DECLARE POINTER ADDRESS:
4  2    END DISPLAY$MESSAGE:
5  1  PRINT$MESSAGE: PROCEDURE(POINTER) EXTERNAL:
6  2    DECLARE POINTER ADDRESS:
7  2    END PRINT$MESSAGE:
8  1  PRINT$NUMBER: PROCEDURE(NO) EXTERNAL:
9  2    DECLARE NO BYTE:
10 2    END PRINT$NUMBER:
11 1  PRINT$BLOCK: PROCEDURE(POINTER,MCOUNT) EXTERNAL:
12 2    DECLARE (POINTER,MCOUNT) ADDRESS:
13 2    END PRINT$BLOCK:
14 1  PRINT: PROCEDURE(POINTER) EXTERNAL:
15 2    DECLARE POINTER ADDRESS:
16 2    END PRINT:
17 1  GET$CHARACTERS: PROCEDURE EXTERNAL:
18 2    END GET$CHARACTERS:
19 1  FBDIV: PROCEDURE(OP1,OP2,OP3) EXTERNAL:
20 2    DECLARE (OP1,OP2,OP3) ADDRESS:
21 2    END FBDIV:
22 1  ANALYSER: PROCEDURE EXTERNAL:
23 2    END ANALYSER:
24 1  INVERT: PROCEDURE EXTERNAL:
25 2    END INVERT:
26 1  SIFT: PROCEDURE EXTERNAL:
27 2    END SIFT:
28 1  CI: PROCEDURE BYTE EXTERNAL:
29 2    END CI:
30 1  LO: PROCEDURE(MES) EXTERNAL:
31 2    DECLARE MES BYTE:
32 2    END LO:
33 1  EXIT: PROCEDURE EXTERNAL:
34 2    END EXIT:
35 1  IOSET: PROCEDURE(MASK) EXTERNAL:
36 2    DECLARE MASK BYTE:
37 2    END IOSET:
38 1  OPEN:
    PROCEDURE(AFT, FILE, ACCESS, MODE, STATUS) EXTERNAL:

```

```

2      DECLARE (AFT, FILE, ACCESS, MODE, STATUS) ADDRESS;
2      END OPEN;
1      READ:
2          PROCEDURE (AFT, BUF, CNT, ACT, STATUS) EXTERNAL;
2          DECLARE (AFT, BUF, CNT, ACT, STATUS) ADDRESS;
2          END READ;
1      WRITE:
2          PROCEDURE (AFT, BUF, CNT, STATUS) EXTERNAL;
2          DECLARE (AFT, BUF, CNT, STATUS) ADDRESS;
2          END WRITE;
1      CLOSE:
2          PROCEDURE (AFT, STATUS) EXTERNAL;
2          DECLARE (AFT, STATUS) ADDRESS;
2          END CLOSE;
1      DECLARE FILE(20) BYTE PUBLIC;
1      DECLARE FPOINTER ADDRESS PUBLIC;
1      DECLARE BUFFER(1) BYTE EXTERNAL;
1      DECLARE R(1) STRUCTURE(
2          RDIGIT(4) BYTE) EXTERNAL;
1      DECLARE EN(1) STRUCTURE(
2          NDIGIT(4) BYTE) EXTERNAL;
1      DECLARE K(1) STRUCTURE(
2          KDIGIT(4) BYTE) EXTERNAL;

1      DECLARE NSAMPLE ADDRESS EXTERNAL;
1      DECLARE SAMPLE(1) BYTE EXTERNAL;
1      DECLARE (ORDER, PITCH, OVER$FLOW$FLAG, ENERGY)
        BYTE EXTERNAL;

1      DECLARE (I, J, L) ADDRESS;
1      DECLARE (M, N, STATUS, NFRAME, COUNT) ADDRESS;
1      DECLARE (LPC$POINTER, RAM$POINTER) ADDRESS;
1      DECLARE RAM BASED RAM$POINTER BYTE;
1      DECLARE LPC BASED LPC$POINTER BYTE;
1      DECLARE C(2) ADDRESS;

1      DECLARE F(*) BYTE DATA(
        0, 0, 200, 0);

1      DECLARE ASK$SPEECH$FILE(*) BYTE DATA(OAH, ODH,
        'File name of speech data :', OAH, ODH, 00);
1      DECLARE ERROR(*) BYTE DATA(
        'Error', OAH, ODH, 00);
1      DECLARE ASK$PRINTER(*) BYTE DATA(
        'Do You want to take print-out?', OAH, ODH, 00);
1      DECLARE ASK$LPC$FILE(*) BYTE DATA(OAH, ODH,
        'File name for LPC coefficients :', OAH, ODH, 00);

1      DECLARE HEADING(*) BYTE DATA(OAH, ODH,
        'Frame -----Auto-corr',
        'relations-----',
        'RMS R P k1 k2 k3 k4 k5 k6',
        ' k7 k8 k9 k10 Error Energy', OAH, ODH, 00);

/*****
*   PROCEDURE : PLACE
*   FUNCTION : Places data to LPC data buffer

```



```

*   PLM CALL   : CALL PLACE(.POINTER);
*   INPUT      : POINTER = Address of first entry
*   OUTPUT     : transfer operation
*   GLOBALS    : LPC$POINTER
*   CALLS      : none
*/

1  PLACE: PROCEDURE(LPC$DATA) PUBLIC;
2  DECLARE LPC$DATA BYTE;

3  2      LPC = LPC$DATA;
4  2      LPC$POINTER = LPC$POINTER + 1;
5  2      COUNT = COUNT + 1;

6  2  END PLACE;
   /*****

7  1  DECLARE BEGIN LABEL PUBLIC;
8  1  BEGIN:
9  1  NSAMPLE = 200;
0  1  CALL DISPLAY$MESSAGE(.ASK$PRINTER(0));
1  1  IF (CI AND 7FH) = 'Y' THEN
2  1      CALL IOSET(81H);
   ELSE
3  1      CALL IOSET(41H);

4  1  CALL DISPLAY$MESSAGE(.ASK$SPEECH$FILE(0));
5  1  CALL GETCHARACTERS;
6  1  CALL OPEN(.M,.BUFFER(0),1,0,.STATUS);
7  1  IF STATUS <> 0 THEN
8  1      DO;
9  2      CALL DISPLAY$MESSAGE(.ERROR(0));
10 2      GO TO BEGIN;
112      END;
12 1      ELSE DO;
13 2      CALL READ(M,6000H,6000H,.C(1),.STATUS);
14 2      CALL CLOSE(M,.STATUS);
15 2      END;
16 1      CALL FBDIV(.C(0),.F(0),.C(0));
17 1      NFRAME = C(1);

18 1      RAM$POINTER=6000H;
19 1      LPC$POINTER=6000H;

20 1      CALL DISPLAY$MESSAGE(.ASK$LPC$FILE(0));
21 1      CALL GET$CHARACTERS;
22 1      CALL OPEN(.M,.BUFFER(0),2,0,.STATUS);

23 1      COUNT = 0;
24 1      C(1) = 0;

25 1      DO N = 1 TO 10;
26 2          DO I = 0 TO 3;
27 3              K(N).KDIGIT(I) = 0;
28 3          END;
29 2      END;

30 1      DO N=1 TO NFRAME-2;

```

```

0 2      IF C(1) = 0 THEN
1 2      DO:
2 3          CALL LO(OCH); /*--Form Feed--*/
3 3          CALL PRINT$MESSAGE(.HEADING(0));
4 3      END:
5 2      DO I=0 TO 399:
6 3          SAMPLE(I)=RAM:
7 3          RAM$POINTER = RAM$POINTER+1;
8 3      END:
9 2      RAM$POINTER = RAM$POINTER - 200;

10 2      CALL LO(' ');
11 2      CALL PRINT$NUMBER(N);
12 2      CALL LO(' ');

13 2      CALL SIFT;
14 2      CALL LO(' ');
15 2      IF ENERGY = 0 THEN
16 2          FILE(0) = 0;
17 2      ELSE DO:
18 3          CALL ANALYSER;
19 3          CALL INVERT;
20 3      END:

21 2      DO I = 0 TO 12;
22 3          CALL PRINT$NUMBER(FILE(I));
23 3          CALL PLACE(FILE(I));
24 3          CALL LO(' ');
25 3      END:

26 2      DO I = 0 TO 1;
27 3          CALL PLACE(EN(1).NDIGIT(I));
28 3          CALL PRINT$NUMBER(EN(1).NDIGIT(I));
29 3      END:
30 2      CALL LO(' ');
31 2      DO I = 0 TO 2;
32 3          CALL PLACE(R(0).RDIGIT(I));
33 3          CALL PRINT$NUMBER(R(0).RDIGIT(I));
34 3      END:
35 2      IF OVER$FLOW$FLAG = OFFH THEN
36 2          CALL LO('*');
37 2      CALL LO(OAH);
38 2      CALL LO(ODH);
39 2      C(1) = C(1) + 1;
40 2      IF C(1) = 60 THEN C(1) = 0;
41 2      END:
42 1      CALL WRITE(M,6000H,COUNT,.STATUS);
43 1      CALL CLOSE(M,.STATUS);

44 1      CALL EXIT;
45 1      END MAIN;

```

MODULE INFORMATION:

CODE AREA SIZE = 03AFH 943D
VARIABLE AREA SIZE = 002FH 47D
MAXIMUM STACK SIZE = 000BH 8D
217 LINES READ
0 PROGRAM ERROR(S)

OF PL/M-80 COMPILATION

S-II PL/M-80 V3.0 COMPILATION OF MODULE ALAW
 ECT MODULE PLACED IN :F1:ALAW.OBJ
 PILER INVOKED BY: PLM80 :F1:ALAW.SRC WORKFILES(:FO:,:FO:)

\$PAGEWIDTH(80)

```

/*****
**
**          SPEECH SYNTHESIS SYSTEM
**          A LAW CONVERSION
**
**  FILE NAME   : ALAW.SRC          DATE
**  MODULE DEF  : Decode Tables     -----
**  AUTHOR      : Onder Bicioglu    22 06 85
**
*****/

```

ALAW: DO;

```

1  DECLARE SIGNAL(1) STRUCTURE(
    SBYTE1  BYTE,
    SADDRESS ADDRESS,
    SBYTE2  BYTE) EXTERNAL;

```

```

3  1  DECLARE SAMPLE(400) BYTE PUBLIC;
4  1  DECLARE NSAMPLE ADDRESS EXTERNAL;
5  1  DECLARE I ADDRESS;
6  1  DECLARE J BYTE;

```

```

/*****
*  PROCEDURE : A$LAW$EXPANSION
*  FUNCTION  : Establishes conversion of A-law
*              companded data to linear data:
*  PLM CALL   : CALL ALAW;
*  INPUT      : SAMPLE = Companded data
*  OUTPUT     : SIGNAL.SDIGIT = Expanded data
*  GLOBALS    : SIGNAL, SAMPLE
*  CALLS      : none
*

```

I	encoded data	I	decoded data	I
I	0 0 0 w x y z	I	0 0 0 0 0 0 0 w x y z 1	I
I	0 0 1 w x y z	I	0 0 0 0 0 0 1 w x y z 1	I
I	0 1 0 w x y z	I	0 0 0 0 0 1 w x y z 1 0	I
I	0 1 1 w x y z	I	0 0 0 0 1 w x y z 1 0 0	I
I	1 0 0 w x y z	I	0 0 0 1 w x y z 1 0 0 0	I
I	1 0 1 w x y z	I	0 0 1 w x y z 1 0 0 0 0	I
I	1 1 0 w x y z	I	0 1 w x y z 1 0 0 0 0 0	I
I	1 1 1 w x y z	I	1 w x y z 1 0 0 0 0 0 0	I

```

1-----I-----I-----I
*/
7 1 A$LAWS$EXPANSION: PROCEDURE PUBLIC;
8 2 DO I=0 TO 399;
9 3   J = SHR(SAMPLE(I),4) AND 7;
0 3   SIGNAL(I).SBYTE1 = 0;
1 3   SIGNAL(I).SBYTE2 = 0;
2 3   SIGNAL(I).SADDRESS = SAMPLE(I) AND 0FH;
3 3   SIGNAL(I).SADDRESS = SHL(SIGNAL(I).SADDRESS,1) OR 1;
4 3   IF J <> 0 THEN
5 3     SIGNAL(I).SADDRESS = SIGNAL(I).SADDRESS OR 20H;
6 3   IF J > 1 THEN
7 3     SIGNAL(I).SADDRESS = SHL(SIGNAL(I).SADDRESS,J-1);
8 3   IF (SAMPLE(I) AND 80H) <> 0 THEN DO;
9 4     SIGNAL(I).SADDRESS = NOT SIGNAL(I).SADDRESS;
0 4     SIGNAL(I).SBYTE2 = 0FFH;
1 4   END;
2 4 END;
3 3
4 2 END A$LAWS$EXPANSION;

```

```

/*****
*  PROCEDURE : A$LAWS$COMPANSION
*  FUNCTION  : Converts calculated 32-bit linear data
*              to A-law companded form
*  PLM CALL  : CALL A$LAWS$COMPANSION;
*  INPUT     : SIGNAL.SDIGIT = 32 bit data
*  OUTPUT    : SAMPLE = 8 bit companded data
*  GLOBALS   : SIGNAL,SAMPLE
*  CALLS     : none

```

encoded data	I	decoded data	I
0 0 0 w x y z	I	0 0 0 0 0 0 0 w x y z a	I
0 0 1 w x y z	I	0 0 0 0 0 0 1 w x y z a	I
0 1 0 w x y z	I	0 0 0 0 0 1 w x y z a b	I
0 1 1 w x y z	I	0 0 0 0 1 w x y z a b c	I
1 0 0 w x y z	I	0 0 0 1 w x y z a b c c	I
1 0 1 w x y z	I	0 0 1 w x y z a b c c e	I
1 1 0 w x y z	I	0 1 w x y z a b c d e f	I
1 1 1 w x y z	I	1 w x y z a b c d e f g	I

```

*/
25 1 A$LAWS$COMPANSION: PROCEDURE PUBLIC;
26 2 DECLARE (DUMMY,MASK) ADDRESS;
27 2 DO I = 0 TO NSAMPLE - 1;
28 3   DUMMY = SIGNAL(I).SADDRESS;
29 3   IF (SIGNAL(I).SBYTE2 AND 80H) = 0 THEN

```

```
3      SAMPLE(I) = 0;
3      ELSE DO ;
4      SAMPLE(I) = 80H;
4      DUMMY = NOT DUMMY;
4      END;
3      MASK = 800H;
3      DO J = 0 TO 6;
4      IF (MASK AND DUMMY) (<) 0 THEN GO TO OKEY;
4      MASK = SHR(MASK, 1);
4      END;
3      DUMMY = SHR(DUMMY, 1);
3      OKEY:
3      J = NOT(J) AND 7;
3      IF J (<) 0 THEN
3      DUMMY = SHR(DUMMY, J) AND 0FH;
3      SAMPLE(I) = SAMPLE(I) OR SHL(J, 4) OR LOW(DUMMY);
3      END;
7  2      END A$LAWS$COMPANSION;
3  1      END ALAW;
```

FILE INFORMATION:

CODE AREA SIZE	= 01E2H	482D
VARIABLE AREA SIZE	= 0197H	407D
MAXIMUM STACK SIZE	= 0004H	4D
130 LINES READ		
0 PROGRAM ERROR(S)		

OF PL/M-80 COMPILATION

PL/M-80 V3.0 COMPILATION OF MODULE ANALYSIS

MODULE PLACED IN :F1:ANALGR.OBJ

COMPILER INVOKED BY: PLM80 :F1:ANALGR.SRC WORKFILES(:FO:, :FO:)

\$PAGEWIDTH(80)

```

/*****
**
**          SPEECH SYNTHESIS SYSTEM
**          REFLECTION COEFFICIENT ANALYSIS
**          USING GUEGUEN LE-RUEX ALGORITHM
**
** FILE NAME      : ANALGR.SRC          DATE
** MODULE DEF     : Analyses speech    -----
** AUTHOR        : Onder Bicioglu      22 06 85
**
*****/

```

ANALYSIS: DO;

```

1 1 DECLARE EP(17) STRUCTURE(
2   PWORD(2) ADDRESS) PUBLIC;
3 1 DECLARE R(25) STRUCTURE(
4   RWORD(2) ADDRESS) PUBLIC;
5 1 DECLARE EN(17) STRUCTURE(
6   NWORD(2) ADDRESS) PUBLIC;
7 1 DECLARE K(17) STRUCTURE(
8   KDIGIT(4) BYTE) PUBLIC;
9 1 DECLARE SIGNAL(400) STRUCTURE(
10  SDIGIT(4) BYTE) PUBLIC;
11 1 DECLARE W(1) STRUCTURE(
12  WDIGIT(4) BYTE) EXTERNAL;
13 1 DECLARE SAMPLE(1) BYTE EXTERNAL;
14
15 1 DECLARE (PITCH, ORDER) BYTE EXTERNAL;
16 1 DECLARE NSAMPLE ADDRESS PUBLIC;
17 1 DECLARE E(4) BYTE PUBLIC;
18 1 DECLARE EDUMMY(4) BYTE PUBLIC;
19 1 DECLARE BDUMMY(4) BYTE PUBLIC;
20 1 DECLARE TEMP(2) ADDRESS PUBLIC;
21 1 DECLARE OVER$FLOW$FLAG BYTE PUBLIC;
22
23 1 DECLARE (M, N, I, J) ADDRESS;
24 1 DECLARE ZERO(*) BYTE DATA(0, 0, 0, 0);
25
26 1 FBADD: PROCEDURE(OP1, OP2, OP3) EXTERNAL;
27 2   DECLARE (OP1, OP2, OP3) ADDRESS;
28 2   END FBADD;
29 1 FBSUB: PROCEDURE(OP1, OP2, OP3) EXTERNAL;
30 2   DECLARE (OP1, OP2, OP3) ADDRESS;
31 2   END FBSUB;
32 1 FBDIV: PROCEDURE(OP1, OP2, OP3) EXTERNAL;
33 2   DECLARE (OP1, OP2, OP3) ADDRESS;
34 2   END FBDIV;
35 1 FBMUL: PROCEDURE(OP1, OP2, OP3) EXTERNAL;
36 2   DECLARE (OP1, OP2, OP3) ADDRESS;
37 2   END FBMUL;

```

```

1  A$LAWE$EXPANSION: PROCEDURE EXTERNAL;
2  END A$LAWE$EXPANSION;

/*****
*  PROCEDURE : GUEGUEN$LE$RUEX
*  FUNCTION  : Analysis using Gueguen Le-Ruex
*              algorithm
*  PLM CALL   : CALL GUEGUEN$LE$RUEX;
*  INPUT      : PITCH =pitch period
*              SAMPLE = 8-bit samples
*  OUTPUT     : K=reflection coefficients
*  GLOBALS    : K, R, ORDER, NSAMPLE, SIGNAL
*  CALLS      : FBMUL, FBDIV, FBADD, A$LAWE$EXPANSION
*/

2  1  GUEGUEN$LE$RUEX: PROCEDURE PUBLIC;

/*-----INITIALIZATION-----*/

3  2  DO M = 1 TO 10;
4  3  DO J = 0 TO 1;
5  4  EP(M).PWORD(J) = R(M).RWORD(J);
6  4  EN(M).NWORD(J) = R(M-1).RWORD(J);
7  4  END;
8  3  END;
9  2  OVERFLOW$FLAG = 0;

/*----- G U E G U E N L E - R U E X -----*/

0  2  DO I = 0 TO 9;

/*-- K(I) = - (EP(I+1) / EN(1))-----*/

1  3  CALL FBDIV(.EP(I+1).PWORD(0), .EN(1).NWORD(0),
               .EDUMMY(0));

/*-----RANGE CHECK-----*/

2  3  IF (EDUMMY(2) > 0) AND (EDUMMY(2) < OFFH)
3  4  THEN DO;
4  4  OVER$FLOW$FLAG = OFFH;
5  4  RETURN;
6  4  END;

/*-----*/

7  3  CALL FBSUB(.EDUMMY(0), .ZERO(0),
               .K(I+1).KDITIT(0));
8  3  CALL FBMUL(.K(I+1).KDITIT(0), .EP(I+1).PWORD(0),
               .EDUMMY(0));
9  3  CALL FBADD(.EN(1).NWORD(0), .EDUMMY(0),
               .EN(1).NWORD(0));
0  3  CALL FBMUL(.EN(10-I).NWORD(0), .K(I+1).KDITIT(0),
               .EDUMMY(0));
1  3  CALL FBADD(.EP(10).PWORD(0), .EDUMMY(0),
               .EP(10).PWORD(0));

2  3  IF I = 9 THEN

```



```

3      RETURN;

3      DO M = I+2 TO 9;
4          CALL FBMUL(.K(I+1).KDIGIT(0),.EN(M-I).NWORD(0),
4              .EDUMMY(0));
4          CALL FBADD(.EDUMMY(0),.EP(M).PWORD(0),
4              .E(0));
4          CALL FBMUL(.EP(M).PWORD(0),.K(I+1).KDIGIT(0),
4              .EDUMMY(0));
4          CALL FBADD(.EDUMMY(0),.EN(M-I).NWORD(0),
4              .EN(M-I).NWORD(0));
4          CALL FBADD(.E(0),.ZERO(0),.EP(M).PWORD(0));
4      END;
3      END;

2      END GUEGUEN$LE$RUEX;
      /*****
      *   PROCEDURE : ANALYSER
      *   FUNCTION  : Analyses the signal and creates
      *               reflection coefficients using auto-
      *               correlation method and
      *               GUEGUEN LE-RUEX algorithm
      *   PLM CALL   : CALL ANALYSER;
      *   INPUT      : ORDER = order of the prediction
      *               NSAMPLE = number of samples to be used
      *   OUTPUT     : K(I).KDIGIT(J) = reflection coefficients
      *               OVERFLOW$FLAG = OFFH if any of the k's
      *                   exceeds 1 in magnitude
      *   GLOBALS     : SIGNAL structure
      *   CALLS       : FBMUL, FBDIV, FBADD, FBSUB, GUEGUEN$LE$RUEX
      */
3      1      ANALYSER: PROCEDURE PUBLIC;
4      2      CALL A$LAW$EXPANSION;

      /*-----W I N D O W I N G -----*/
5      2      DO I = 0 TO SHR(NSAMPLE,1)-1;
6      3          CALL FBMUL(.SIGNAL(I).SDIGIT(0),.W(I).WDIGIT(0),
7              .SIGNAL(I).SDIGIT(0));
7      3          CALL FBMUL(.SIGNAL(NSAMPLE-1-I).SDIGIT(0),.W(I).WDIGIT(0),
8              .SIGNAL(NSAMPLE-1-I).SDIGIT(0));
8      3      END;

      /*-----C A L C U L A T I O N   O F   R ( I )-----*/
9      2      DO I = 0 TO 10;
10     3          DO J = 0 TO 1;
11     4              R(I).RWORD(J) = 0;
12     4          END;
13     3          DO M = 0 TO NSAMPLE-1-I;
14     4              CALL FBMUL(.SIGNAL(M).SDIGIT(0),
15              .SIGNAL(M+I).SDIGIT(0),.TEMP(0));
16     4              CALL FBADD(.TEMP(0),.R(I).RWORD(0),
17              .R(I).RWORD(0));
18     4          END;
19     3      END;

```

```
2      CALL GUEGUEN$LE$RUEX;  
2      END ANALYSER;  
1      END ANALYSIS;
```

LE INFORMATION:

```
CODE AREA SIZE      = 0335H      821D  
VARIABLE AREA SIZE = 078BH      1931D  
MAXIMUM STACK SIZE = 0008H      8D  
171 LINES READ  
0 PROGRAM ERROR(S)
```

OF PL/M-80 COMPILATION

S-II PL/M-80 V3.0 COMPILATION OF MODULE ENCODE
 ECT MODULE PLACED IN :F1:ENCODE.OBJ
 PILER INVOKED BY: PLM80 :F1:ENCODE.SRC WORKFILES(:F0:,:F0:)

\$PAGEWIDTH(80)

```

/*****
**
**          SPEECH SYNTHESIS SYSTEM          **
**          ENCODING MODULE                  **
**
**  FILE NAME   : ENCODE.SRC                DATE   **
**  MODULE DEF  : Encode Tables             ----- **
**  AUTHOR      : Onder Bicioglu           22 06 85   **
**
*****/
ENCODE: DO;

```

```

1
2  1  DECLARE P(*) BYTE DATA(0,
      25,27,28,30,31,32,34,35,
      36,37,38,40,42,44,46,48,
      50,52,54,56,58,61,63,66,
      70,75,80,85,90,95,100);
3  1  DECLARE RMS(*) ADDRESS DATA(0,52,87,123,
      174,246,348,491,694,981,1385,1957,
      2764,3904,5514,7789);
4  1  DECLARE T1(32) ADDRESS EXTERNAL ;
5  1  DECLARE T2(32) ADDRESS EXTERNAL ;
6  1  DECLARE T3(16) ADDRESS EXTERNAL ;
7  1  DECLARE T4(16) ADDRESS EXTERNAL ;
8  1  DECLARE T5(16) ADDRESS EXTERNAL ;
9  1  DECLARE T6(16) ADDRESS EXTERNAL ;
10 1  DECLARE T7(16) ADDRESS EXTERNAL ;
11 1  DECLARE T8(8) ADDRESS EXTERNAL ;
12 1  DECLARE T9(8) ADDRESS EXTERNAL ;
13 1  DECLARE T10(8) ADDRESS EXTERNAL ;
14 1  DECLARE SIZE(*) BYTE DATA(0,31,31,15,15,
      15,15,15,7,7,7);
15 1  DECLARE BEGIN(*) ADDRESS DATA(0,0,64,128,160,192,
      224,256,288,304,320,336);
16 1  DECLARE FACTOR(*) ADDRESS DATA(0,512);
17 1  DECLARE ENERGY#FACTOR(*) ADDRESS DATA(0,128);
18 1  DECLARE K(1) STRUCTURE(
      KWORD(2) ADDRESS) EXTERNAL;
19 1  DECLARE R(1) STRUCTURE(
      R1    BYTE,
      R23   ADDRESS,
      R4    BYTE) EXTERNAL;
20 1  DECLARE EN(1) STRUCTURE(

```

NWORD(2) ADDRESS) EXTERNAL;

```

1  DECLARE (I,J) BYTE;
1  DECLARE ORDER BYTE EXTERNAL;
1  DECLARE PITCH BYTE EXTERNAL;
1  DECLARE FILE(1) BYTE EXTERNAL;
1  DECLARE FPOINTER ADDRESS EXTERNAL;
1  DECLARE NORM(2) ADDRESS;

1  FBDIV: PROCEDURE(O1,O2,O3) EXTERNAL;
2      DECLARE (O1,O2,O3) ADDRESS;
2      END FBDIV;

1  FBMUL: PROCEDURE(O1,O2,O3) EXTERNAL;
2      DECLARE (O1,O2,O3) ADDRESS;
2      END FBMUL;

/*****
*  PROCEDURE : INVERT
*  FUNCTION   : Inverts 32-bit reflection coefficients
*                to encoded format for TMS
*  PLM CALL   : CALL INVERT;
*  INPUT      : K = 32-bit reflection coefficients
*                PITCH = pitch period
*                U = Energy
*  OUTPUT     : FILE = Inverted parameters
*  GLOBALS    : E,K,FILE,PITCH
*  CALLS      : FBMUL,FBDIV
*/

3  1  INVERT: PROCEDURE PUBLIC;
4  2  DECLARE POINTER ADDRESS;
5  2  DECLARE TABLE BASED POINTER ADDRESS;

6  2  CALL FBDIV(.EN(1).NWORD(0),.ENERGY$FACTOR(0),
7      .EN(1).NWORD(0));
8  2  DO I = 0 TO 14;
9  3      IF EN(1).NWORD(0) < RMS(I) THEN GO TO RMS$FOUND;
10 3  END;
11 2  I = 14;
12 2  RMS$FOUND:
13 2  FILE(0) = I;
14 2  FILE(1) = 0;

15 2  DO I = 0 TO 31;
16 3      IF PITCH < P(I) THEN
17 3          GO TO PITCH$FOUND;
18 3  END;
19 2  PITCH$FOUND:
20 2  FILE(2) = I - 1;

21 2  DO I = 1 TO 10;
22 3      POINTER = .T1(0) + BEGIN(I);
23 3      CALL FBMUL(.K(I).KWORD(0),.FACTOR(0),.NORM(0));
24 3      J = 0;
25 3      DO WHILE 1;
```

```

4      IF NORM(1) > 8000H THEN
/*-----k(i) is negative-----*/
4      DO:
5          IF TABLE < 8000H THEN GO TO FOUND;
/*-----table is positive-----*/
5          IF NORM(1) < TABLE THEN GO TO FOUND;
/*-----table is negative -----*/
/*-----and greater than k(i)-----*/
5      END;
4      ELSE DO:
/*-----k(i) is positive-----*/
5          IF TABLE < 8000H THEN
/*-----table is positive-----*/
5          DO:
6              IF TABLE > NORM(1) THEN
/*-----table is greater than k(i)-----*/
6              GO TO FOUND;
6              END;
5          END;
4          IF J = SIZE(1) THEN GO TO FOUND;
4          J = J + 1;
4          POINTER = POINTER + 2;
2          END;
3          FOUND:
3              FILE(I+2) = J;
4          3      END;
5          2      END INVERT:
6          1      END ENCODE:

```

ULE INFORMATION:

CODE AREA SIZE	= 019DH	413D
VARIABLE AREA SIZE	= 0008H	8D
MAXIMUM STACK SIZE	= 0004H	4D
138 LINES READ		
0 PROGRAM ERROR(S)		

OF PL/M-80 COMPILATION

OBJ

LINE

SOURCE STATEMENT

```

1 $PAGEWIDTH(80)
2 ;*****
3 ;*
4 ;*          SPEECH SYNTHESIS SYSTEM
5 ;*          TMS CONVERSION TABLES
6 ;*
7 ;*  FILE NAME   : DCDTBL.SRC          DATE
8 ;*  MODULE DEF  : K-conversion table -----
9 ;*  AUTHOR      : Onder Bicioglu     22 06 85
10 ;*
11 ;*****/
12 ;
13      NAME      DCDTBL
14 ;
15      PUBLIC    T1,T2,T3,T4
16      PUBLIC    T5,T6,T7,T8
17      PUBLIC    T9,T10
18 ;
19      CSEG
20 ;
21 T1:      DW      -501,-498,-497,-495
22
23
24
25
26 ;
27 T2:      DW      -493,-491,-488,-482,-478,-474,-469,-464
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

0BFE

0EFF

0FFE

11FE

13FE

15FE

18FE

1EFE

22FE

26FE

2BFE

30FE

35FE

3CFE

43FE

4BFE

64FE

84FE

ADFE

E0FE

1DFF

62FF

AFFF

FFFF

5000

9D00

E200

1F01

5101

7B01

9B01

B401

B8FE

IC	OBJ	LINE	SOURCE STATEMENT
042	D1FE		
044	EEFE		
046	OCFF		
048	2DFF	28	DW -211, -175, -138, -99, -59, -18, 24, 64
04A	51FF		
04C	76FF		
04E	9DFF		
050	CSFF		
052	EEFF		
054	1800		
056	4000		
058	6900	29	DW 105, 143, 180, 215, 248, 278, 306, 331
05A	8F00		
05C	B400		
05E	D700		
060	F800		
062	1601		
064	3201		
066	4B01		
068	6201	30	DW 354, 374, 392, 408, 422, 435, 445, 455
06A	7601		
06C	8801		
06E	9801		
070	A601		
072	B301		
074	BD01		
076	C701		
078	CF01	31	DW 463, 470, 476, 506
07A	D601		
07C	DC01		
07E	FA01		
		32 ;	
080	47FE	33 T3:	DW -441, -387, -333, -279
082	7DFE		
084	B3FE		
086	E9FE		
088	1FFF	34	DW -225, -171, -117, -63, -9, 45, 98, 152
08A	55FF		
08C	8BFF		
08E	C1FF		
090	F7FF		
092	2D00		
094	6200		
096	9800		
098	CE00	35	DW 206, 260, 314, 368
09A	0401		
09C	3A01		
09E	7001		
		36 ;	
0A0	B8FE	37 T4:	DW -328, -273, -217, -161
0A2	EFEE		
0A4	27FF		
0A6	5FFF		
0A8	96FF	38	DW -106, -50, 5, 61, 116, 172, 228, 283
0AA	CEFF		

C	OBJ	LINE	SOURCE STATEMENT
AC	0500		
AE	3D00		
B0	7400		
B2	AC00		
B4	E400		
B6	1B01		
B8	5301	39	DW 339, 394, 450, 506
BA	8A01		
BC	C201		
BE	FA01		
		40 ;	
CD	B8FE	41 T5:	DW -328, -282, -235, -189
CE	E6FF		
CF	15FF		
D0	43FF		
D2	72FF	42	DW -142, -96, -50, -3, 43, 90, 136, 182
DA	A0FF		
DC	CEFF		
DE	FDFF		
E0	2B00		
E2	5A00		
E4	8800		
E6	B600		
E8	E500	43	DW 229, 275, 322, 368
EA	1301		
EC	4201		
EE	7001		
		44 ;	
EO	00FF	45 T6:	DW -256, -212, -168, -123
EE	2CFF		
E4	58FF		
E6	85FF		
E8	B1FF	46	DW -79, -35, 10, 54, 98, 143, 187, 232
EA	DDFF		
EC	0A00		
EE	3600		
EO	6200		
E2	8F00		
E4	BB00		
E6	E800		
E8	1401	47	DW 276, 320, 365, 408
EA	4001		
EC	6D01		
EE	9801		
		48 ;	
100	CCFE	49 T7:	DW -308, -260, -212, -164
102	FCFE		
104	2CFF		
106	5CFF		
108	8BFF	50	DW -117, -69, -21, 27, 75, 122, 170, 218
10A	BBFF		
10C	EBFF		
10E	1B00		
110	4B00		
112	7A00		

OBJ	LINE	SOURCE STATEMENT
14 AA00		
16 DA00		
18 0A01	51	DW 266, 314, 361, 40
1A 3A01		
1C 6901		
1E 2B00		
	52 ;	
20 00FF	53 T8:	DW -256, -161, -66, 29
22 5FFF		
24 BEFF		
26 1D00		
28 7C00	54	DW 124, 219, 314, 409
2A DB00		
2C 3A01		
2E 9901		
	55 ;	
30 00FF	56 T9:	DW -256, -176, -96, -15
32 50FF		
34 A0FF		
36 F1FF		
38 4100	57	DW 65, 146, 226, 307
3A 9200		
3C E200		
3E 3301		
	58 ;	
40 33FF	59 T10:	DW -205, -132, -59, 14
42 7CFF		
44 C5FF		
46 0E00		
48 5700	60	DW 87, 160, 234, 307
4A A000		
4C EA00		
4E 3301		
	61 ;	
	62 ;	*****
	63 ;	
	64	END

IC SYMBOLS

C 0000	T10	C 0140	T2	C 0040	T3	C 0080
C 00A0	T5	C 00C0	T6	C 00E0	T7	C 0100
C 0120	T9	C 0130				

RNAL SYMBOLS

SYMBOLS

C 0000	T10	C 0140	T2	C 0040	T3	C 0080
C 00A0	T5	C 00C0	T6	C 00E0	T7	C 0100
C 0120	T9	C 0130				

MBLY COMPLETE, NO ERRORS

OBJ	LINE	SOURCE STATEMENT
	55 ;	
11 FB	56 LOOP1: EI	
12 BC	57 CMP H	; Remain in loop if
13 FB	58 EI	; end of buffer is
14 D21100 C	59 JNZ LOOP1	; not reached yet
	60 ;	
17 F3	61 DI	
	62 ;	
18 3E12	63 MVI A,12H	; initialize interrupt
1A D3FD	64 OUT LSTST	; levels
	65 ;	
1C 3EFC	66 MVI A,0FCH	; enable only level
1E D3FC	67 OUT INTMSK	; 0,1,2 and 3
	68 ;	
20 3ED0	69 MVI A,0COH	; Enable monitor interrupts
22 D3F3	70 OUT INTRST	
	71 ;	
24 FB	72 EI	
	73 ;	
25 C9	74 RET	
	75 ;	
	76 ;*****	
	77 ; PROCEDURE : SAMPLE	
	78 ; FUNCTION : takes data via paper-tape reader input	
	79 ; PLM CALL : none	
	80 ; INPUT : (HL) = current destination	
	81 ; OUTPUT : increments (HL) and places data to RAM	
	82 ; GLOBALS : none	
	83 ; CALLS : none	
	84 ; DESTROYS : HL	
	85 ;	
26 F5	86 SAMPLE: PUSH PSW	; Starting point of interrupt
	87	; service routine
27 DBFA	88 WAIT: IN STATUS	; if interrupt is not the
29 E640	89 ANI 40H	; desired one (Line Printer)
2B DA2700 C	90 JZ WAIT	; then wait
	91 ;	
2E 3ED0	92 MVI A,0COH	; else reset printer
30 D3F3	93 OUT INTRST	; flip-flop
	94 ;	
32 DBFB	95 IN PRTPTR	; read data from
	96	; paper tape reader
34 77	97 MOV M,A	; place it to buffer
35 23	98 INX H	; point to next location
36 3E20	99 MVI A,20H	; restore last status
38 D3FD	100 OUT LSTST	
3A F1	101 POP PSW	
3B C9	102 RET	
	103 ;	
	104 ;*****	
	105 ;	
	106 END	

CORD C 0000 SAMPLE C 0026

INTERNAL SYMBOLS
 ATBUF E 0000

ER SYMBOLS

ATBUF E 0000	INTMSK A 00FC	INTRST A 00F3	INTSER A 0018
OP1 C 0011	LSTSTT A 00FD	PRTPTR A 00FB	RECORD C 0000
MPLE C 0026	STATUS A 00FA	WAIT C 0027	

SEMBLY COMPLETE, NO ERRORS

3-II PL/M-80 V3.0 COMPILATION OF MODULE LISTENER
 ECT MODULE PLACED IN :F1:LISTEN.OBJ
 PILER INVOKED BY: PLM80 :F1:LISTEN.SRC WORKFILES(:FO:,:FO:)

\$PAGEWIDTH(80)

```

/*****
**
**          SPEECH SYNTHESIS SYSTEM          **
**          LISTENER MAIN MODULE             **
**
**  FILE NAME   : LISTEN.SRC                DATE   **
**  MODULE DEF  : Listener                  ----- **
**  AUTHOR      : Onder Bicioglu           22 06 85   **
**
**          *****/

```

```

1  LISTENER: DO;
2  1  DECLARE DATBUF(0B000H) BYTE PUBLIC;
3  1  DECLARE BUFFER(1) BYTE EXTERNAL;
4  1  DECLARE (FILE,STATUS,END$ADDRESS) ADDRESS PUBLIC;
5  1  DECLARE KEY BYTE;
6  1  DECLARE SAMPLE ADDRESS EXTERNAL;
7  1  DECLARE INT$POINTER ADDRESS;
8  1  DECLARE INT$DATA BASED INT$POINTER BYTE;
9  1  GET$CHARACTERS: PROCEDURE EXTERNAL;
10 2  END GET$CHARACTERS;
11 1  DISPLAY$MESSAGE: PROCEDURE(POINTER) EXTERNAL;
12 2  DECLARE POINTER ADDRESS;
13 2  END DISPLAY$MESSAGE;
14 1  RECORD: PROCEDURE(DATA$END) EXTERNAL;
15 2  DECLARE DATA$END ADDRESS;
16 2  END RECORD;
17 1  OPEN: PROCEDURE(OP1,OP2,OP3,OP4,OP5) EXTERNAL;
18 2  DECLARE (OP1,OP2,OP3,OP4,OP5) ADDRESS;
19 2  END OPEN;
20 1  WRITE: PROCEDURE(OP1,OP2,OP3,OP4) EXTERNAL;
21 2  DECLARE (OP1,OP2,OP3,OP4) ADDRESS;
22 2  END WRITE;
23 1  CLOSE: PROCEDURE(OP1,OP2) EXTERNAL;
24 2  DECLARE (OP1,OP2) ADDRESS;
25 2  END CLOSE;
26 1  EXIT: PROCEDURE EXTERNAL;
27 2  END EXIT;
28 1  CI: PROCEDURE BYTE EXTERNAL;
29 2  END CI;
30 1  CO: PROCEDURE(MES) EXTERNAL;
31 2  DECLARE MES BYTE;
32 2  END CO;
33 1  DECLARE ASK$SECONDS(*) BYTE DATA(OAH,ODH,
    'How many seconds?',OAH,ODH,00);
34 1  DECLARE ASK$WHICH$WORD(*) BYTE DATA(OAH,ODH,
    'Enter name of output file!',OAH,ODH,00);
35 1  DECLARE ASK$GOON(*) BYTE DATA(OAH,ODH,
    'Do You want to continue?',OAH,ODH,00);
36 1  DECLARE OFFSET(*) ADDRESS DATA(8000,8000,16000,
    24000,32000,40000,40000,40000);

```

```

1      MAIN:
      DO WHILE 1;
2      CALL DISPLAY$MESSAGE(.ASK$SECONDS(0));
2      KEY = CI AND 7FH;
2      CALL CO(KEY);
2      END$ADDRESS = .DAT$BUF(0)+OFFSET((KEY AND 7));
2      INT$POINTER = 18H;
2      INT$DATA = 0C3H;
2      INT$POINTER = INT$POINTER + 1;
2      INT$DATA = LOW(.SAMPLE);
2      INT$POINTER = INT$POINTER + 1;
2      INT$DATA = HIGH(.SAMPLE);
2      CALL RECORD(END$ADDRESS);
2      CALL DISPLAY$MESSAGE(.ASK$WHICH$WORD(0));
2      CALL GET$CHARACTERS;
2      CALL OPEN(.FILE,.BUFFER(0),2,0,.STATUS);
2      CALL WRITE(FILE,.DAT$BUF(0),OFFSET(KEY AND 7),
                .STATUS);
2      CALL CLOSE(FILE,.STATUS);
2      CALL DISPLAY$MESSAGE(.ASK$GOON(0));
2      IF (CI AND 5FH) <> 'Y' THEN
2          CALL EXIT;
2      END;
1      END LISTENER;

```

LE INFORMATION:

```

CODE AREA SIZE      = 0114H    276D
VARIABLE AREA SIZE = B009H    45065D
MAXIMUM STACK SIZE = 0008H     8D
75 LINES READ
0 PROGRAM ERROR(S)

```

OF PL/M-80 COMPILATION

II PL/M-80 V3.0 COMPILATION OF MODULE MAIN

MODULE PLACED IN :F1:FORMAT.OBJ

PL/M-80 V3.0 COMPILER INVOKED BY: PLM80 :F1:FORMAT.SRC WORKFILES(:FO:,:FO:)

\$PAGEWIDTH(80)

```

/*****
**
**          SPEECH SYNTHESIS SYSTEM          **
**          FORMATTER MODULE                 **
**
**  FILE NAME      : FORMAT.SRC              DATE      **
**  MODULE DEF     : Formats LPC             ----- **
**                  Coefficients            25 05 1985 **
**  AUTHOR         : Onder Bicioglu         **
**
**
**
*****/
MAIN: DO;

```

```

1  DISPLAY$MESSAGE: PROCEDURE(POINTER) EXTERNAL;
2  DECLARE POINTER ADDRESS;
2  END DISPLAY$MESSAGE;
1  PRINT$MESSAGE: PROCEDURE(POINTER) EXTERNAL;
2  DECLARE POINTER ADDRESS;
2  END PRINT$MESSAGE;
1  GET$CHARACTERS: PROCEDURE EXTERNAL;
2  END GET$CHARACTERS;
1  CI: PROCEDURE BYTE EXTERNAL;
2  END CI;
1  LO: PROCEDURE(MES) EXTERNAL;
2  DECLARE MES BYTE;
2  END LO;
1  EXIT: PROCEDURE EXTERNAL;
2  END EXIT;
1  IOSET: PROCEDURE(MASK) EXTERNAL;
2  DECLARE MASK BYTE;
2  END IOSET;
1  OPEN:
    PROCEDURE(AFT, FILE, ACCESS, MODE, STATUS) EXTERNAL;
2  DECLARE (AFT, FILE, ACCESS, MODE, STATUS) ADDRESS;
2  END OPEN;
1  READ:
    PROCEDURE(AFT, BUF, CNT, ACT, STATUS) EXTERNAL;
2  DECLARE(AFT, BUF, CNT, ACT, STATUS) ADDRESS;
2  END READ;
1  WRITE:
    PROCEDURE(AFT, BUF, CNT, STATUS) EXTERNAL;
2  DECLARE(AFT, BUF, CNT, STATUS) ADDRESS;
2  END WRITE;
1  CLOSE:
    PROCEDURE(AFT, STATUS) EXTERNAL;
2  DECLARE (AFT, STATUS) ADDRESS;
2  END CLOSE;

1  PACK: PROCEDURE(UNPACK$ADDRESS, PACK$ADDRESS)
    ADDRESS EXTERNAL;

```

```

2      DECLARE (UNPACK$ADDRESS, PACK$ADDRESS) ADDRESS;
2      END PACK;

1      DECLARE LPC$DATA(2000H) BYTE PUBLIC;
1      DECLARE FORMAT(6000H) BYTE PUBLIC;
1      DECLARE BUFFER(1) BYTE EXTERNAL;
1      DECLARE (INDEX, READ$COUNT, COUNT) ADDRESS PUBLIC;
1      DECLARE (I, FILE, STATUS, FORMAT$POINTER) ADDRESS;
1      DECLARE CHECK$SUM BYTE;

1      DECLARE ASK$LPC$FILE(*) BYTE DATA(OAH, ODH,
      'File name of LPC data :', OAH, ODH, OO);
1      DECLARE ERROR(*) BYTE DATA(
      'Error', OAH, ODH, OO);
1      DECLARE ASK$HEX$FILE(*) BYTE DATA(OAH, ODH,
      'File name for HEX file :', OAH, ODH, OO);

/*****
*   PROCEDURE : PLACE
*   FUNCTION   : Places data to LPC data buffer
*   PLM CALL   : CALL PLACE(.POINTER);
*   INPUT      : POINTER = Address of first entry
*   OUTPUT     : transfer operation
*   GLOBALS    : FORMAT$POINTER
*   CALLS      : none
*/
1      PLACE: PROCEDURE(PACK$DATA) PUBLIC;
2      DECLARE PACK$DATA BYTE;

2      FORMAT(COUNT) = PACK$DATA;
2      FORMAT$POINTER = FORMAT$POINTER + 1;
2      COUNT = COUNT + 1;

2      END PLACE;

/*****
*   PROCEDURE : ENTER
*   FUNCTION   : Enters data to FORMATDATA
*   PLM CALL   : CALL ENTER(DATA);
*   INPUT      : DATA = number to be entered
*   OUTPUT     : ASCII code of corresponding to number
*               is generated and placed to memory
*   GLOBALS    : CHECKSUM
*   CALLS      : PLACE
*/
1      ENTER: PROCEDURE(PACK$DATA) PUBLIC;
2      DECLARE PACK$DATA ADDRESS;

2      DECLARE CONVERTTABLE(*) BYTE DATA(
      '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A',
      'B', 'C', 'D', 'E', 'F');

2      CALL PLACE(CONVERT$TABLE(SHR(PACK$DATA, 4)));
2      CALL PLACE(CONVERT$TABLE(PACK$DATA AND OFH));
2      CHECKSUM = CHECKSUM + PACK$DATA;

2      END ENTER;

```



```

/*****/
07 1 DECLARE BEGIN LABEL PUBLIC;
08 1 BEGIN:
09 1 CALL DISPLAY$MESSAGE(.ASK$LPC$FILE(0));
10 1 CALL GETCHARACTERS;
11 1 CALL OPEN(.FILE,.BUFFER(0),1,0,.STATUS);
12 1 IF STATUS <> 0 THEN
13 1 DO;
14 2 CALL DISPLAY$MESSAGE(.ERROR(0));
15 2 GO TO BEGIN;
16 2 END;
17 1 ELSE DO;
18 2 CALL READ(FILE,.FORMAT(0),3000H,.READ$COUNT,
19 2 .STATUS);
20 2 CALL CLOSE(FILE,.STATUS);
21 2 END;
22 1 CALL DISPLAY$MESSAGE(.ASK$HEX$FILE(0));
23 1 CALL GET$CHARACTERS;
24 1 CALL OPEN(.FILE,.BUFFER(0),2,0,.STATUS);
25 1 READ$COUNT = PACK(.FORMAT(0),.LPC$DATA(0)) -
26 1 .LPC$DATA(0);
27 1 COUNT = 0;
28 1 DO INDEX = 0 TO READCOUNT;
29 2 IF (INDEX AND OFH) = 0 THEN
30 2 DO;
31 3 CALL PLACE(' ');
32 3 CHECKSUM = 0;
33 3 CALL ENTER(10H);
34 3 CALL ENTER(HIGH(INDEX));
35 3 CALL ENTER(LOW(INDEX));
36 3 CALL ENTER(00);
37 3 END;
38 2 CALL ENTER(LPCDATA(INDEX));
39 2 IF (INDEX AND OFH) = OFH THEN
40 2 DO;
41 3 CALL ENTER(-CHECKSUM);
42 3 CALL PLACE(0DH);
43 3 CALL PLACE(0AH);
44 3 END;
45 2 END;
46 1 IF (READCOUNT AND OFH) <> OFH THEN
47 1 DO;
48 2 DO I=READ$COUNT + 1
49 2 TO (READ$COUNT AND OFFFOH) +OFH;
50 3 CALL ENTER(OFFH);
51 3 END;
52 2 CALL ENTER(-CHECKSUM);
53 2 CALL PLACE(0DH);
54 2 CALL PLACE(0AH);
55 2 END;

```

```
2 1 CALL PLACE(' ');
3 1 CALL ENTER(0);
4 1 CALL ENTER(0);
5 1 CALL ENTER(0);
6 1 CALL ENTER(1);
7 1 CALL ENTER(OFFH);
8 1 CALL PLACE(ODH);
9 1 CALL PLACE(OAH);
0 1 CALL WRITE(FILE, .FORMAT(0), COUNT, .STATUS);
1 1 CALL CLOSE(FILE, .STATUS);

2 1 CALL EXIT;

3 1 END MAIN;
```

RULE INFORMATION:

```
CODE AREA SIZE      = 0268H      616D
VARIABLE AREA SIZE = 8012H      32786D
MAXIMUM STACK SIZE = 0008H      8D
179 LINES READ
0 PROGRAM ERROR(S)
```

OF PL/M-80 COMPILATION

OBJ

LINE SOURCE STATEMENT

```

1 $PAGEWIDTH(80)
2 ;*****
3 ;*
4 ;*          SPEECH SYNTHESIS SYSTEM
5 ;*          COEFFICIENT PACKING
6 ;*
7 ;* FILE NAME      : PACK.SRC          DATE
8 ;* MODULE DEF     : PACKER          -----
9 ;* AUTHOR         : Onder Bicioglu   22 06 85
10 ;*
11 ;*****/
12 ;
13          NAME      PACKER
14 ;
15          DSEG
16 ;
01 17 SETBIT: DB      1
01 18 VOICED: DB      1
01 19 SAVE:   DB      1
20 ;
21          CSEG
22 ;
23          PUBLIC   PACK
24 ;
25 ;*****
26 ; PROCEDURE : PACK
27 ; FUNCTION  : Packs byte values less than 8-bits
28 ;             converts a sequence of 3,4,5,6 bit
29 ;             coefficients to full bytes
30 ; PLM CALL   : CALL PACK(UNPACKED$ADDRESS,PACK$ADDRESS);
31 ; INPUT      : UNPACKED$ADDRESS = start address of
32 ;             unpacked coefficients
33 ;             PACK$ADDRESS = start address of packed
34 ;             coefficients
35 ; OUTPUT     : packed coefficients
36 ; GLOBALS    : VOICED,SETBIT
37 ; CALLS      : PUT
38 ; DESTROYS   : all
39 ;
EB 40 PACK:     XCHG          ;(HL)=start address of
41                                     ;packed coefficients
42          MOV      D,B
43          MOV      E,C      ;(DE)=start address of
44                                     ;unpacked coefficients
3E01 45 MVI      A,01H
320000 46 STA     SETBIT      ;start filling bits from LSB
3600 47 MVI      M,0
48 ;
49          DCX      D
1B 50 ENERGY: INX      D
1A 51          LDAX    D
0604 52 MVI      B,4
07 53          RLC
07 54          RLC

```

CC	OBJ	LINE	SOURCE STATEMENT
011	07	55	RLC
012	07	56	RLC
013	CD9E00	57	LOOP1: CALL PUT
016	05	58	DCR B
017	C21300	59	JNZ LOOP1
01A	1A	60	LDAX D
01B	B7	61	ORA A
01C	CA0B00	62	JZ ENERGY
01F	FE0F	63	CPI OFH
021	C8	64	RZ
		65	;
022	13	66	REPEAT: INX D
023	1A	67	LDAX D
024	0F	68	RRC
025	CD9E00	69	CALL PUT
		70	;
028	13	71	PITCH: INX D
029	1A	72	LDAX D
02A	07	73	RLC
02B	07	74	RLC
02C	07	75	RLC
02D	320100	76	STA VOICED
030	0605	77	MVI B, 5
032	CD9E00	78	LOOP2: CALL PUT
035	05	79	DCR B
036	C23200	80	JNZ LOOP2
		81	;
039	1B	82	DCX D
03A	1A	83	LDAX D
03B	13	84	INX D
03C	B7	85	ORA A
03D	C20B00	86	JNZ ENERGY
		87	;
040	0E02	88	K1K2: MVI C, 2
042	13	89	LOOP3: INX D
043	1A	90	LDAX D
044	07	91	RLC
045	07	92	RLC
046	07	93	RLC
047	0605	94	MVI B, 5
049	CD9E00	95	LOOP4: CALL PUT
04C	05	96	DCR B
04D	C24900	97	JNZ LOOP4
050	0D	98	DCR C
051	C24200	99	JNZ LOOP3
		100	;
054	0E02	101	K3K4: MVI C, 2
056	13	102	LOOP5: INX D
057	1A	103	LDAX D
058	07	104	RLC
059	07	105	RLC
05A	07	106	RLC
05B	07	107	RLC
05C	0604	108	MVI B, 4
05E	CD9E00	109	LOOP6: CALL PUT

OBJ	LINE	SOURCE STATEMENT
01 05	110	
02 C25E00	111	DCR B
05 0D	112	JNZ LOOP6
06 C25600	113	DCR C
	114	JNZ LOOP5
09 3A0100	115	LDA VOICED
0C B7	116	DRA A
0D CA0B00	117	JZ ENERGY
	118	
00 0E03	119	KSK6K7: MVI C,3
02 13	120	LOOP7: INX D
03 1A	121	LDAX D
04 07	122	RLC
05 07	123	RLC
06 07	124	RLC
07 07	125	RLC
08 0604	126	MVI B,4
0A CD9E00	127	LOOP8: CALL PUT
0D 05	128	DCR B
0E C27A00	129	JNZ LOOP8
01 0D	130	DCR C
02 C27200	131	JNZ LOOP7
	132	
05 0E03	133	K8910: MVI C,3
07 13	134	LOOP9: INX D
08 1A	135	LDAX D
09 07	136	RLC
0A 07	137	RLC
0B 07	138	RLC
0C 07	139	RLC
0D 07	140	RLC
0E 0603	141	MVI B,3
00 CD9E00	142	LOOP10: CALL PUT
03 05	143	DCR B
04 C29000	144	JNZ LOOP10
07 0D	145	DCR C
08 C28700	146	JNZ LOOP9
	147	
0B C30B00	148	JMP ENERGY
	149	
	150	*****
	151	PROCEDURE : PUT
	152	FUNCTION : sets current bit of packed coefficient
	153	byte, or skips it, according to MSB
	154	of content of (A). If full byte is
	155	filled next byte is accessed
	156	PLM CALL : none
	157	INPUT : MSB (A) = 1 or 0
	158	OUTPUT : shifts content of SETBIT by 1 to right,
	159	and sets or skips current bit of packed
	160	byte
	161	GLOBALS : SETBIT
	162	CALLS : none
	163	DESTROYS : (A),
	164	

CC	OBJ	LINE	SOURCE STATEMENT
09E	17	165	PUT: RAL
09F	320200	D 166	STA SAVE
0A2	D2AA00	C 167	JNC NOSET
		168	:
0A5	3A0000	D 169	LDA SETBIT
0A8	B6	170	ORA M
0A9	77	171	MOV M, A
		172	:
0AA	3A0000	D 173	NOSET: LDA SETBIT
0AD	17	174	RAL
0AE	B7	175	ORA A
0AF	C2B700	C 176	JNZ NOFIN
0B2	3E01	177	MVI A, 01H
0B4	23	178	INX H
0B5	3600	179	MVI M, 0
		180	:
0B7	320000	D 181	NOFIN: STA SETBIT
0BA	3A0200	D 182	LDA SAVE
0BD	C9	183	RET
		184	:
		185	:*****
		186	:
		187	END

LIC SYMBOLS

K C 0000

ERNAL SYMBOLS

R SYMBOLS

RGY C 000B	K1K2 C 0040	K3K4 C 0054	K5K6K7 C 0070
10 C 0085	LOOP1 C 0013	LOOP10 C 0090	LOOP2 C 0032
P3 C 0042	LOOP4 C 0049	LOOP5 C 0056	LOOP6 C 005E
P7 C 0072	LOOP8 C 007A	LOOP9 C 0087	NOFIN C 00B7
ET C 00AA	PACK C 0000	PITCH C 0028	PUT C 009E
EAT C 0022	SAVE D 0002	SETBIT D 0000	VOICED D 0001

SEMBLY COMPLETE, NO ERRORS

S-II PL/M-80 V3.0 COMPILATION OF MODULE GRAPHUTILITIES
 ECT MODULE PLACED IN :F1:GUTILY.OBJ
 PILER INVOKED BY: PLM80 :F1:GUTILY.SRC WORKFILES(:FO:, :FO:)

\$PAGEWIDTH(80)

```

/*****
**
**          SPEECH SYNTHESIS SYSTEM          **
**          DISPLAY UTILITIES                **
**
**  FILE NAME   : GUTILY.SRC                DATE
**  MODULE DEF  : Display utilities         -----
**  AUTHOR      : Onder Bicioglu           22 06 85
**
*****/

```

GRAPH\$UTILITIES:

DO:

```

1  DECLARE (TIMEBASE,AMPLITUDEBASE,START)
2      ADDRESS PUBLIC;
3  1  DECLARE GRAPHDATA(80) BYTE PUBLIC;
4  1  DECLARE (LEFT,RIGHT,UP,DOWN,
5      HOME,CLEAR,LI) BYTE EXTERNAL;
6  1  DECLARE MDS BYTE EXTERNAL;
7  1  DECLARE (I,J) ADDRESS;
8  1  DECLARE DATABUF(4000H) BYTE PUBLIC;
9      DECLARE CR          LITERALLY '13',
10         HORIZONTALUNITS LITERALLY '8',
11         VERTICALUNITS   LITERALLY '8';
12  1  DECLARE KEY BYTE EXTERNAL;
13  1  DECLARE TEMP BYTE;
14  1  DECLARE CONVERT(*) BYTE DATA('0','1','2','3',
15         '4','5','6','7','8','9','A','B','C','D','E','F');
16
17  CO: PROCEDURE(CHARACTER) EXTERNAL;
18      DECLARE CHARACTER BYTE;
19      END CO;
20  CI: PROCEDURE BYTE EXTERNAL;
21      END CI;
22  DISPLAY$MESSAGE: PROCEDURE(MESPOINTER) EXTERNAL;
23      DECLARE MESPOINTER ADDRESS;
24      END DISPLAY$MESSAGE;

```

```

/*****
*  PROCEDURE : MOVE$CURSOR
*  FUNCTION  : moves cursor in the given direction
*              in the given amount of characters
*  PLM CALL  : CALL MOVE$CURSOR(DIRECTION,COUNT);
*  INPUT     : DIRECTION = direction
*              COUNT      = count
*  OUTPUT    : cursor movement
*  GLOBALS   : none
*  CALLS     : CO
*/

```

```

0  1  MOVECURSOR: PROCEDURE(DIRECTION,COUNT) PUBLIC;
1  2  DECLARE (P,DIRECTION,COUNT) BYTE;

```

```

2      DO P=1 TO COUNT;
3      IF (DIRECTION=UP) OR (DIRECTION=DOWN) OR
        (MDS=1) THEN
3          CALL CO(LI);
3          CALL CO (DIRECTION);
3      END;

7  2  END MOVECURSOR;

/*****
*  PROCEDURE : GETHEXDIGIT
*  FUNCTION  : Receives a 1-digit hex number from
*              keyboard
*  PLM CALL  : J = GETHEXDIGIT
*  INPUT     : key from console
*  OUTPUT    : returns hex digit
*  GLOBALS   : none
*  CALLS     : CI,CO
*/
3  1  GETHEXDIGIT: PROCEDURE BYTE;

9  2  DO WHILE 1;
0  3      KEY=CI AND 7FH;
1  3      DO J=0 TO 15;
2  4          IF KEY=CONVERT(J) THEN
3  4              DO;
4  5                  CALL CO(KEY);
5  5                  RETURN J;
6  5              END;
7  4          END;
8  3      END;
9  2  END GETHEXDIGIT;

/*****
*  PROCEDURE : GETFULL$NUMBER
*  FUNCTION  : Receives 2-digit hex digit from
*              keyboard
*  PLM CALL  : CALL GET$FULL$NUMBER;
*  INPUT     : none
*  OUTPUT    : TEMP = 2-digit hex number
*  GLOBALS   : TEMP
*  CALLS     : GET$HEX$DIGIT
*/
0  1  GETFULLNUMBER: PROCEDURE PUBLIC;
1  2      TEMP=GETHEXDIGIT;
2  2      TEMP=SHL(TEMP,4) OR GETHEXDIGIT;
3  2  END GETFULLNUMBER;

/*****
*  PROCEDURE : DISPLAY$START
*  FUNCTION  : Displays time counterpart of origin
*  PLM CALL  : CALL DISPLAY$START
*  INPUT     : START = address of memory where
*              display operation has to start
*  OUTPUT    : displays origin
*  GLOBALS   : START

```



```

* CALLS      : CO, MOVECURSOR
*/
44 1  DISPLAYSTART: PROCEDURE PUBLIC;
45 2  DECLARE DUMY ADDRESS ;

46 2  CALL MOVECURSOR(DOWN, 2*VERTICALUNITS+5);
47 2  CALL MOVECURSOR(RIGHT, 65);
48 2  DUMY=START-.DATABUF(0);
49 2  DUMY=SHR(DUMY, 3);
50 2  TEMP = HIGH(DUMY);
51 2  CALL CO(CONVERT(SHR(TEMP, 4)));
52 2  CALL CO(CONVERT(TEMP AND 0FH));
53 2  TEMP = LOW(DUMY);
54 2  CALL CO(CONVERT(SHR(TEMP, 4)));
55 2  CALL CO(CONVERT(TEMP AND 0FH));
56 2  CALL CO(LI);
57 2  CALL CO(HOME);

58 2  END DISPLAYSTART;

/*****
* PROCEDURE : DRAW$GRAPH
* FUNCTION  : Draws the required graph
* PLM CALL  : CALL DRAW$GRAPH;
* INPUT     : START = begin address of display buffer
*            TIME$BASE = time base
*            AMPLITUDE$BASE = amplitude base
* OUTPUT    : graph
* GLOBALS   : START, TIMEBASE, AMPLITUDEBASE
* CALLS     : MOVE$CURSOR, CO
*/

59 1  DRAWGRAPH: PROCEDURE PUBLIC;

60 2  DECLARE POINTER ADDRESS;
61 2  DECLARE DATABUFFER BASED POINTER BYTE;
62 2  DECLARE (DIRECTION, FACTOR) BYTE;
63 2  DECLARE (SUM, TEMPWORD) ADDRESS;

64 2  POINTER=START;
65 2  CALL MOVE$CURSOR(DOWN, 2);
66 2  CALL MOVE$CURSOR(RIGHT, 3);
67 2  DO I=1 TO HORIZONTALUNITS*8;
68 3  SUM = 0;
69 3  DO J=1 TO TIMEBASE;
70 4  TEMPWORD = DATABUFFER;
71 4  IF (DATABUFFER AND 80H)=0 THEN
72 4  SUM = SUM + TEMPWORD;
  ELSE
73 4  SUM = SUM - (TEMP$WORD AND 7FH);
74 4  POINTER=POINTER+1;
75 4  END;
76 3  TEMPWORD = SUM / TIME$BASE;
77 3  FACTOR = LOW(TEMP$WORD);
78 3  IF (FACTOR AND 80H)=0 THEN
79 3  DO;
80 4  TEMP = VERTICALUNITS-(FACTOR/AMPLITUDEBASE);

```

```

01 4      END;
02 3      ELSE DO;
03 4          TEMP=OFFH-FACTOR+1;
04 4          TEMP=(TEMP/AMPLITUDEBASE)+VERTICALUNITS;
05 4      END;
06 3      IF (I AND 1)=1 THEN DIRECTION=DOWN;
08 3      ELSE DO;
09 4          DIRECTION=UP;
10 4          TEMP=(2*VERTICAL$UNITS)-TEMP;
11 4      END;
12 3      DO J=0 TO VERTICALUNITS*2;
13 4          IF J<> (VERTICALUNITS) THEN
14 4              DO;
15 5              IF TEMP=J THEN
16 5                  DO;
17 6                  CALL CO('*');
18 6                  CALL MOVE$CURSOR(LEFT,1);
19 6              END;
20 5              ELSE DO;
21 6                  IF GRAPHDATA(I)=J THEN
22 6                      DO;
23 7                      CALL CO(' ');
24 7                      CALL MOVE$CURSOR(LEFT,1);
25 7                  END;
26 6              END;
27 5          END;

08 4          IF J=VERTICALUNITS*2 THEN
09 4              CALL MOVE$CURSOR(RIGHT,1);
10 4          ELSE DO;
11 5              CALL CO(LI);
12 5              CALL CO(DIRECTION);
13 5          END;
14 4      END;
15 3      GRAPHDATA(I)=TEMP;
16 3  END;

17 2  CALL CO(LI);
18 2  CALL CO(HOME);

19 2  END DRAWGRAPH;

/*****
*  PROCEDURE : CHANGE$ORIGIN
*  FUNCTION   : Changes origin
*  PLM CALL   : CALL CHANGE$ORIGIN;
*  INPUT      : new start address from the keyboard
*  OUTPUT     : START = new begin address
*  GLOBALS    : START
*  CALLS      : CO, DRAW$GRAPH
*/
0 1  CHANGE$ORIGIN: PROCEDURE PUBLIC;

1 2  DECLARE DUMY ADDRESS;

2 2  CALL MOVE$CURSOR(DOWN, 2*VERTICALUNITS+5);
3 2  CALL MOVE$CURSOR(RIGHT, 65);

```

```

4 2      DUMY = GET$HEX$DIGIT;
5 2      DUMY = SHL(DUMY,4) OR GET$HEX$DIGIT;
6 2      DUMY = SHL(DUMY,4) OR GET$HEX$DIGIT;
7 2      DUMY = SHL(DUMY,4) OR GET$HEX$DIGIT;
8 2      START = .DATA$BUF(0) + SHL(DUMY,3);
9 2      CALL CO(LI);
0 2      CALL CO(HOME);
1 2      CALL DRAW$GRAPH;

2 2      END CHANGE$ORIGIN;

/*****
*  PROCEDURE : CHANGE$TIME$BASE
*  FUNCTION  : Changes time base
*  PLM CALL  : CALL CHANGE$TIME$BASE;
*  INPUT     : New time base from keyboard
*  OUTPUT    : TIME$BASE = new time base
*  GLOBALS   : TIME$BASE, TEMP
*  CALLS     : GET$FULL$NUMBER, CO, MOVE$CURSOR,
*              DRAW$GRAPH
*/

3 1      CHANGETIMEBASE: PROCEDURE PUBLIC;

4 2      CALL MOVECURSOR(DOWN, 2*VERTICALUNITS+5);
5 2      CALL MOVECURSOR(RIGHT, 17);
6 2      CALL GETFULLNUMBER;
7 2      TIMEBASE=TEMP;
8 2      CALL CO(LI);
9 2      CALL CO(HOME);
0 2      CALL DRAWGRAPH;

1 2      END CHANGETIMEBASE;

/*****
*  PROCEDURE : CHANGE$AMPLITUDE$BASE
*  FUNCTION  : Changes amplitude base
*  PLM CALL  : CALL CHANGE$AMPLITUDE$BASE;
*  INPUT     : new amolitude base from keyboard
*  OUTPUT    : AMPLITUDE$BASE = new base
*  GLOBALS   : AMPLITUDE$BASE , TEMP
*  CALLS     : MOVE$CURSOR, CO, DRAW$GRAPH,
*              GET$FULL$NUMBER
*/

2 1      CHANGEAMPLITUDEBASE: PROCEDURE PUBLIC;

3 2      CALL MOVECURSOR(DOWN, 2*VERTICALUNITS+5);
4 2      CALL MOVECURSOR(RIGHT, 44);
5 2      CALL GETFULLNUMBER;
6 2      AMPLITUDEBASE=TEMP;
7 2      CALL CO(LI);
8 2      CALL CO(HOME);
9 2      CALL DRAWGRAPH;

0 2      END CHANGEAMPLITUDEBASE;

```

```

/*****
*  PROCEDURE : SHIFT$RIGHT
*  FUNCTION  : Shifts graph one time unit to right
*  PLM CALL  : CALL SHIFT$RIGHT;
*  INPUT     : START = begin of display buffer
*  OUTPUT    : START is updated
*  GLOBALS   : START
*  CALLS     : DISPLAYSTART, DRAWGRAPH
*/

```

```

1  SHIFTRIGHT: PROCEDURE PUBLIC;
2      IF START >= (.DATA$BUF(0) + SHL(TIMEBASE,3))
3      THEN DO;
3          START=START-8*TIMEBASE;
3          CALL DISPLAYSTART;
3          CALL DRAWGRAPH;
3      END;
2  END SHIFTRIGHT;

```

```

/*****
*  PROCEDURE : SHIFT$LEFTT
*  FUNCTION  : Shifts graph one time unit to leftt
*  PLM CALL  : CALL SHIFT$RIGHT;
*  INPUT     : START = begin of display buffer
*  OUTPUT    : START is updated
*  GLOBALS   : START
*  CALLS     : DISPLAYSTART, DRAWGRAPH
*/

```

```

1  SHIFTLLEFT: PROCEDURE PUBLIC;
2      START=START+8*TIMEBASE;
2      CALL DISPLAYSTART;
2      CALL DRAWGRAPH;
2  END SHIFTLLEFT;

```

```

/*****/

```

```

1  END GRAPHUTILITIES;

```

LE INFORMATION:

```

CODE AREA SIZE      = 0414H    1044D
VARIABLE AREA SIZE  = 406AH    16490D
MAXIMUM STACK SIZE  = 0008H     8D
321 LINES READ
0 PROGRAM ERROR(S)

```

OF PL/M-80 COMPILATION

PLM-80 V3.0 COMPILATION OF MODULE CONVERTATOK
 OBJECT MODULE PLACED IN :F1:CONVAK.OBJ
 COMPILER INVOKED BY: PLM80 :F1:CONVAK.SRC WORKFILES(:F0:,:F0:)

\$PAGEWIDTH(80)

```

/*****
**
**          SPEECH ANALYSIS SYSTEM
**          CONVERSION OF Ks TO As
**
** FILE NAME      : CONVAK.SRC          DATE
** MODULE DEF     : Conversion K to A  -----
** AUTHOR        : Onder Bicioglu      22 06 85
**
**
*****/
CONVERT$A$TO$K: DO;

```

```

1 1  DECLARE A(1) STRUCTURE(
2     ADIGIT(4) BYTE) EXTERNAL;
3 1  DECLARE K(1) STRUCTURE(
4     KDIGIT(4) BYTE) EXTERNAL;
5 1  DECLARE MIGNAL(150) STRUCTURE(
6     MDIGIT(4) BYTE) PUBLIC;
7 1  DECLARE E(4) BYTE EXTERNAL;
8 1  DECLARE BDUMMY(4) BYTE EXTERNAL;
9 1  DECLARE (M,I,J,L) BYTE;

8 1  FB MUL: PROCEDURE(O1,O2,O3) EXTERNAL;
9 2     DECLARE (O1,O2,O3) ADDRESS;
0 2     END FB MUL;
1 1  FBSUB: PROCEDURE(O1,O2,O3) EXTERNAL;
2 2     DECLARE (O1,O2,O3) ADDRESS;
3 2     END FBSUB;
4 1  FBADD: PROCEDURE(O1,O2,O3) EXTERNAL;
5 2     DECLARE (O1,O2,O3) ADDRESS;
6 2     END FBADD;

7 1  DECLARE ZERO(*) ADDRESS DATA(0,0);
8 1  DECLARE ACCESS$TABLE(*) BYTE DATA(
    0,1,3,6,10,15,21,28,
    36,45,55,66,78,91,105,120,
    136);

```

```

/*****
*  PROCEDURE : GET
*  FUNCTION  : Accesses to the matrix element and
*              returns it in BDUMMY
*  PLM CALL  : CALL GET(P1,P2);
*  INPUT     : Index values
*  OUTPUT    : BDUMMY = element required
*  GLOBALS   : SIGNAL,M
*  CALLS     : none
*/
9 1  GET: PROCEDURE(P1,P2) PUBLIC;
0 2  DECLARE (P1,P2) BYTE;

```

```

2      M = ACCESS$TABLE(P1) + P2;
2      DO J=0 TO 3;
3          BDUMMY(J) = MIGNAL(M).MDIGIT(J);
3      END;
2      END GET;

```

```

/*****
*   PROCEDURE : PUT
*   FUNCTION   : Puts content of E to the desired entry
*   PLM CALL   : CALL PUT(P1,P2);
*   INPUT      : indexes
*   OUTPUT     : none
*   GLOBALS    : SIGNAL,M
*   CALLS      : none
*/

```

```

5 1      PUT: PROCEDURE(P1,P2) PUBLIC;
7 2      DECLARE (P1,P2) BYTE;
3 2      M = ACCESS$TABLE(P1) + P2;
9 2      DO J=0 TO 3;
0 3          MIGNAL(M).MDIGIT(J)=E(J);
1 3      END;
2 2      END PUT;

```

```

/*****
*   PROCEDURE : STEPUP
*   FUNCTION   : Converts K coefficients to A
*                  coefficient
*   PLM CALL   : CALL STEPUP(ORDER);
*   INPUT      : K(I).KDIGIT=K coefficients
*                  ORDER = order of filter
*   OUTPUT     : A(I).ADIGIT = A coefficients
*   GLOBALS    : A,K,E,BDUMMY
*   CALLS      : GET, PUT, FB MUL, FBADD, PRINT$MESSAGE
*/

```

```

3 1      STEPUP: PROCEDURE(ORD) PUBLIC;
4 2      DECLARE ORD BYTE;
5 2      DO J=0 TO 3;
6 3          K(0).KDIGIT(J)=0;
7 3      END;
8 2      K(0).KDIGIT(2)=1;

9 2      DO L=0 TO ORD;
0 3          DO J = 0 TO 3;
1 4              E(J)=0;
2 4          END;
3 3          E(2)=1;
4 3          CALL PUT(L,0);

5 3          IF L < 0 THEN DO;
7 4              DO I=1 TO L-1;
8 5                  CALL GET(L-1,L-I);
9 5                  CALL FB MUL(.BDUMMY(0),.K(L).KDIGIT(0),.E(0));
0 5                  CALL GET(L-1,I);
1 5                  CALL FBADD(.E(0),.BDUMMY(0),.E(0));
2 5                  CALL PUT(L,I);
3 5          END;

```

```
4      4      END;
5      3      DO J=0 TO 3;
6      4          E(J) = K(L).KDIGIT(J);
7      4      END;
8      3      CALL PUT(L,L);
9      3      END;
0      2      DO L = 1 TO ORD;
1      3          CALL GET(ORD,L);
2      3          CALL FBSUB(.BDUMMY(0),.ZERO(0),.A(L).ADIGIT(0));
3      3      END;
4      2      END STEPUP;
5      1      END CONVERT$A$TO$K;
```

MODULE INFORMATION:

CODE AREA SIZE	= 01FDH	509D
VARIABLE AREA SIZE	= 0261H	609D
MAXIMUM STACK SIZE	= 0006H	6D
124 LINES READ		
0 PROGRAM ERROR(S)		

OF PL/M-80 COMPILATION

II PL/M-80 V3.0 COMPILATION OF MODULE USERINTERFACE
 MODULE PLACED IN :F1:USRITF.OBJ
 PLER INVOKED BY: PLM80 :F1:USRITF.SRC WORKFILES(:FO:,:FO:)

\$PAGEWIDTH(80)

```

/*****
**
**          SPEECH SYNTHESIS SYSTEM
**          USER INTERFACE
**
**  FILE NAME      : USRITF.SRC          DATE
**  MODULE DEF     : User interface     -----
**  AUTHOR         : Onder Bicioglu     22 06 85
**
*****/
USER$INTER$FACE: DO;
  
```

```

1  LO: PROCEDURE(MES) EXTERNAL;
2  DECLARE MES BYTE;
2  END LO;
  
```

```

1  CO: PROCEDURE(MES) EXTERNAL;
2  DECLARE MES BYTE;
2  END CO;
  
```

```

1  CI: PROCEDURE BYTE EXTERNAL;
2  END CI;
  
```

```

1  DECLARE BUFFER(40) BYTE PUBLIC;
1  DECLARE EDM      LITERALLY  '0',
    BACK           LITERALLY  '8',
    BLANK          LITERALLY  '20H',
    RUBOUT         LITERALLY  '7FH',
    CR             LITERALLY  '0DH',
    LF             LITERALLY  '0AH';
  
```

```

/*****
*  PROCEDURE : PRINT$NUMBER
*  FUNCTION  : Prints out the given number
*  PLM CALL  : CALL PRINT$NUMBER(NO);
*  INPUT     : NO = number to be printed
*  OUTPUT    : printout
*  GLOBALS   : none
*  CALLS     : LO
*/
  
```

```

1  PRINT$NUMBER: PROCEDURE(NO) PUBLIC;
2  DECLARE NO BYTE;
2  DECLARE L BYTE;
2  DECLARE ASCII$TABLE(*) BYTE DATA(
    '0','1','2','3','4','5','6','7','8','9','A',
    'B','C','D','E','F');
  
```

```

2  CALL LO(' ');
2  L = SHR(NO,4);
2  CALL LO(ASCII$TABLE(L));
2  L = NO AND 0FH;
2  CALL LO(ASCII$TABLE(L));
  
```



```

1 2      END PRINT$NUMBER;

      /*****
      *  PROCEDURE : PRINT
      *  FUNCTION  : Prints out the 4 byte entries
      *  PLM CALL  : CALL PRINT(.BASEADDRESS);
      *  INPUT     : BASEADDRESS= points to first of the 4
      *                byte structure
      *  OUTPUT    : printout
      *  GLOBALS   : none
      *  CALLS     : LO
      */

2 1      PRINT: PROCEDURE(BASE$ADDRESS) PUBLIC;
3 2          DECLARE BASE$ADDRESS ADDRESS;
4 2          DECLARE BUF BASED BASE$ADDRESS BYTE;
5 2          DECLARE N BYTE;
6 2          DO N=0 TO 3;
7 3              CALL PRINT$NUMBER(BUF);
8 3              BASE$ADDRESS = BASE$ADDRESS + 1;
9 3          END;
0 2          CALL LO(' ');
1 2          CALL LO(' ');

2 2      END PRINT;

      /*****
      *  PROCEDURE : PRINT$MESSAGE
      *  FUNCTION  : Prints indicated message
      *  PLM CALL  : CALL PRINT$MESSAGE(.MESSAGE);
      *  INPUT     : address of message
      *  OUTPUT    : printout of message
      *  GLOBALS   : none
      *  CALLS     : LO
      */

3 1      PRINT$MESSAGE: PROCEDURE(POINTER) PUBLIC;
4 2          DECLARE POINTER ADDRESS;
5 2          DECLARE OUTBUFFER BASED POINTER BYTE;

6 2          DO WHILE 1;
7 3              IF OUTBUFFER = EOM THEN RETURN;
8 3              CALL LO(OUTBUFFER);
9 3              POINTER = POINTER+1;
0 3          END;
1 3

2 2      END PRINT$MESSAGE;

      /*****
      *  PROCEDURE : PRINT$BLOCK
      *  FUNCTION  : Prints indicated amount of numbers
      *                from the given address on
      *  PLM CALL  : CALL PRINT$BLOCK(.START,COUNT);
      *  INPUT     : START = start address
      *                COUNT = number of numbers to be printed
      *  OUTPUT    : printout of message
      *  GLOBALS   : none
      *  CALLS     : LO

```

*/

```
1 PRINT$BLOCK: PROCEDURE(START$ADDRESS,COUNT) PUBLIC;
2   DECLARE (N,START$ADDRESS,COUNT) ADDRESS;
```

```
2   DO N = 0 TO COUNT-1;
3     IF (N AND 3) = 0 THEN
```

```
3     DO;
```

```
4       CALL LO(OAH);
```

```
4       CALL LO(ODH);
```

```
4     END;
```

```
3     CALL PRINT(START$ADDRESS);
```

```
3     START$ADDRESS=START$ADDRESS + 4;
```

```
3   END;
```

```
2   CALL LO(OAH);
```

```
2   CALL LO(ODH);
```

```
2 END PRINT$BLOCK;
```

```
/******
```

```
* PROCEDURE : DISPLAY$MESSAGE
```

```
* FUNCTION : Displays indicated message
```

```
* PLM CALL : CALL DISPLAY$MESSAGE(.MESSAGE);
```

```
* INPUT : address of message
```

```
* OUTPUT : display of message
```

```
* GLOBALS : none
```

```
* CALLS : CO
```

```
*/
```

```
1 DISPLAY$MESSAGE: PROCEDURE(POINTER) PUBLIC;
```

```
2   DECLARE POINTER ADDRESS;
```

```
2   DECLARE OUTBUFFER BASED POINTER BYTE;
```

```
2   DO WHILE 1;
```

```
3     IF OUTBUFFER = EDM THEN RETURN;
```

```
3     CALL CO(OUTBUFFER);
```

```
3     POINTER = POINTER+1;
```

```
3   END;
```

```
2 END DISPLAY$MESSAGE;
```

```
/******
```

```
* PROCEDURE : GETCHARACTERS
```

```
* FUNCTION : Receives characters from terminal
```

```
* PLM CALL : CALL GETFILENAME;
```

```
* INPUT : none
```

```
* OUTPUT : returns received character string in
```

```
* BUFFER
```

```
* GLOBALS : BUFFER
```

```
* CALLS : CI,CO
```

```
*/
```

```
1 GET$CHARACTERS: PROCEDURE PUBLIC;
```

```
2   DECLARE (KEY,J) BYTE;
```

```
2   J=0;
```

```
2   DO WHILE 1;
```

```
3     KEY = (CI AND 7FH);
```

```
3     IF KEY = RUBOUT THEN DO;
```

```

4         IF J <> 0 THEN DO;
5             CALL CO(BACK);
5             CALL CO(BLANK);
5             CALL CO(BACK);
5             J=J-1;
5         END;
4         END;
3         ELSE DO;
4             BUFFER(J)=KEY;
4             CALL CO(KEY);
4             IF KEY=CR THEN DO;
5                 CALL CO(LF);
5                 RETURN;
5             END;
4             J= J+1;
4         END;
3     END;

2     END GET$CHARACTERS;

1     END USER$INTERFACE;

```

LE INFORMATION:

```

CODE AREA SIZE      = 0180H      384D
VARIABLE AREA SIZE = 0039H      57D
MAXIMUM STACK SIZE = 0006H      6D
187 LINES READ
0 PROGRAM ERROR(S)

```

OF PL/M-80 COMPILATION

-II PL/M-80 V3.0 COMPILATION OF MODULE PITCHANALYSIS
 IT MODULE PLACED IN :F1:ANAPTC.OBJ
 ILER INVOKED BY: PLM80 :F1:ANAPTC.SRC WORKFILES(:FO:,:FO:)

\$PAGEWIDTH(80)

```

/*****
**
**          SPEECH SYNTHESIS SYSTEM
**          PITCH ANALYSIS
**
** FILE NAME   : ANAPTC.SRC          DATE
** MODULE DEF  : Analyses speech     -----
** AUTHOR     : Onder Bicioglu      22 06 85
**
*****/
PITCH$ANALYSIS: DO;

```

```

1  DECLARE R(20) STRUCTURE(
      RWORD(2) ADDRESS) EXTERNAL;
1  DECLARE K(1) STRUCTURE(
      KDIGIT(4) BYTE) EXTERNAL;
1  DECLARE SIGNAL(1) STRUCTURE(
      SDIGIT(4) BYTE) EXTERNAL;
1  DECLARE W(1) STRUCTURE(
      WDIGIT(4) BYTE) EXTERNAL;
1  DECLARE SAMPLE(1) BYTE EXTERNAL;

1  DECLARE (ORDER,PITCH) BYTE PUBLIC;
1  DECLARE A(5) STRUCTURE(
      ADIGIT(4) BYTE) PUBLIC;

1  DECLARE OVER$FLOW$FLAG BYTE EXTERNAL;
1  DECLARE E(1) BYTE EXTERNAL;
1  DECLARE EDUMMY(1) BYTE EXTERNAL;
1  DECLARE BDUMMY(1) BYTE EXTERNAL;
1  DECLARE TEMP(1) ADDRESS EXTERNAL;

1  DECLARE ENERGY BYTE PUBLIC;

1  DECLARE RATIO STRUCTURE(
      R1      BYTE,
      R23     ADDRESS,
      R4      BYTE);

1  DECLARE NON$ZERO$COUNT BYTE;
1  DECLARE MAXIMUM$RATIO ADDRESS;
1  DECLARE (M,N,I,J) ADDRESS;
1  DECLARE L BYTE;

1  DECLARE VOICED$ENERGY LITERALLY '2';
1  DECLARE VOICING$RATIO LITERALLY '6000H';
1  DECLARE LOWEST$PITCH LITERALLY '48';
1  DECLARE PLOSIVE$LIMIT LITERALLY '11';

1  FBADD: PROCEDURE(OP1,OP2,OP3) EXTERNAL;

```

```

2      DECLARE (OP1,OP2,OP3) ADDRESS;
2      END FBADD;
1      FBSUB: PROCEDURE(OP1,OP2,OP3) EXTERNAL;
2      DECLARE (OP1,OP2,OP3) ADDRESS;
2      END FBSUB;
1      FBDIV: PROCEDURE(OP1,OP2,OP3) EXTERNAL;
2      DECLARE (OP1,OP2,OP3) ADDRESS;
2      END FBDIV;
1      FBMUL: PROCEDURE(OP1,OP2,OP3) EXTERNAL;
2      DECLARE (OP1,OP2,OP3) ADDRESS;
2      END FBMUL;
1      STEPUP: PROCEDURE(ORD) EXTERNAL;
2      DECLARE ORD BYTE;
2      END STEPUP;
1      GUEGUEN$LE$RUEX: PROCEDURE EXTERNAL;
2      END GUEGUEN$LE$RUEX;
1      A$LAW$EXPANSION: PROCEDURE EXTERNAL;
2      END A$LAW$EXPANSION;
1      PRINT: PROCEDURE(POINTER) EXTERNAL;
2      DECLARE POINTER ADDRESS;
2      END PRINT;
1      PRINTNUMBER: PROCEDURE(NUMBER) EXTERNAL;
2      DECLARE NUMBER BYTE;
2      END PRINTNUMBER;

/*****
*   PROCEDURE : SIFT
*   FUNCTION   : Pitch analysis
*   PLM CALL   : CALL SIFT
*   INPUT      : SIGNAL.SDIGIT = 32-bit speech samples
*   OUTPUT     : ENERGY = R(0)
*   GLOBALS    : R,PITCH,ENERGY
*   CALLS      : FBMUL,FBADD
*/
1      SIFT: PROCEDURE PUBLIC;
2
2      CALL A$LAW$EXPANSION;

/*-----PRE-EMPHASIS-----*/
2      DO M = 1 TO 99;
3          I = SHL(M,2);
3          CALL FBSUB(.SIGNAL(I-4).SDIGIT(0),.SIGNAL(I).SDIGIT(0),
3              .SIGNAL(I-4).SDIGIT(0));
3      END;

/*-----WINDOWING-----*/
2      DO M = 0 TO 49;
3          N = SHL(M,1);
3          I = SHL(M,2);
3          CALL FBMUL(.SIGNAL(I).SDIGIT(0),.W(N).WDIGIT(0),
3              .SIGNAL(I).SDIGIT(0));
3          CALL FBMUL(.SIGNAL(399-I).SDIGIT(0),.W(N).WDIGIT(0),
3              .SIGNAL(399-I).SDIGIT(0));
3      END;

```

/*-----*/

```

2 DO I = 0 TO 4;
3   DO J = 0 TO 1;
4     R(I).RWORD(J) = 0;
4   END;
3   J = SHL(I,2);
3   DO M = 0 TO 99-I;
4     N = SHL(M,2);
4     CALL FBMUL(.SIGNAL(N).SDIGIT(0),
4               .SIGNAL(J+N).SDIGIT(0),.E(0));
4     CALL FBADD(.E(0),.R(I).RWORD(0),.R(I).RWORD(0));
4   END;
3 END;

```

```

2 ORDER = 4;
2 IF R(0).RWORD(1) = 0 THEN
2 DO;
3   IF R(0).RWORD(0) < 7000H THEN
3   DO;
4     ENERGY = 0;
4     RETURN;
4   END;
3 END;
2 ENERGY = OFFH;

```

```

2 CALL GUEGUEN$LE$RUEX;
2 CALL STEPUP(4);
2 CALL A$LAW$EXPANSION;

```

/*-----CALCULATION OF e(n)-----*/

```

2 DO I = 1 TO 3;
3   J = SHL(I,2);
3   DO L = 0 TO 3;
4     SIGNAL(J+1).SDIGIT(L) = SIGNAL(J).SDIGIT(L);
4   END;
3   DO M = 1 TO I;
4     N = SHL(M,2);
4     CALL FBMUL(.A(M).ADIGIT(0),.SIGNAL(J-N).SDIGIT(0),
4               .TEMP(0));
4     CALL FBSUB(.TEMP(0),.SIGNAL(J+1).SDIGIT(0),
4               .SIGNAL(J+1).SDIGIT(0));
4   END;
3 END;

2 DO I = 4 TO 99;
3   J = SHL(I,2);
3   DO L = 0 TO 3;
4     SIGNAL(J+1).SDIGIT(L) = SIGNAL(J).SDIGIT(L);
4   END;
3   DO M = 1 TO 4;
4     N = SHL(M,2);
4     CALL FBMUL(.A(M).ADIGIT(0),.SIGNAL(J-N).SDIGIT(0),
4               .TEMP(0));
4     CALL FBSUB(.TEMP(0),.SIGNAL(J+1).SDIGIT(0),
4               .SIGNAL(J+1).SDIGIT(0));

```

```

4      END;
3      END;

/*-----WINDOW-----*/

2      DO M = 0 TO 49;
3          N = SHL(M, 1);
3          I = SHL(M, 2);
3          CALL FBMUL(.SIGNAL(I+1).SDIGIT(0), .W(N).WDIGIT(0),
3              .SIGNAL(I).SDIGIT(0));
3          CALL FBMUL(.SIGNAL(397-I).SDIGIT(0), .W(N).WDIGIT(0),
3              .SIGNAL(396-I).SDIGIT(0));
3      END;

/*-----CALCULATION OF R(N)-----*/

2      TEMP(0) = 0;
2      TEMP(1) = 0;
2      DO M = 0 TO 99;
3          N = SHL(M, 2);
3          CALL FBMUL(.SIGNAL(N).SDIGIT(0), .SIGNAL(N).SDIGIT(0),
3              .E(0));
3          CALL FBADD(.TEMP(0), .E(0), .TEMP(0));
3      END;
2      CALL PRINT$NUMBER(LOW(TEMP(1)));

2      NON$ZERO$COUNT = 0;

2      DO I = 0 TO 13;
3          DO J = 0 TO 1;
4              R(I).RWORD(J) = 0;
4          END;
3          J = SHL(I, 2) + LOWEST$PITCH;
3          DO M = 0 TO 87-I;
4              N = SHL(M, 2);
4              CALL FBMUL(.SIGNAL(N).SDIGIT(0),
4                  .SIGNAL(J+N).SDIGIT(0), .E(0));
4              CALL FBADD(.E(0), .R(I).RWORD(0), .R(I).RWORD(0));
4          END;
3          J = SHL(J, 1);
3          DO M = (88-I) TO 99;
4              N = SHL(M, 2);
4              CALL FBMUL(.SIGNAL(N).SDIGIT(0),
4                  .SIGNAL(N-J).SDIGIT(0), .E(0));
4              CALL FBADD(.E(0), .R(I).RWORD(0), .R(I).RWORD(0));
4          END;
3          IF (R(I).RWORD(1) AND 8000H) = 0 THEN
3              DO;
4                  NON$ZERO$COUNT = NON$ZERO$COUNT + 1;
4                  CALL FBDIV(.R(I).RWORD(0), .TEMP(0), .R(I).RWORD(0));
4              END;
3              ELSE
3                  R(I).RWORD(0) = 0;
4                  CALL PRINT$NUMBER(HIGH(R(I).RWORD(0)));
3              END;

2      PITCH = 0;

```

```

2      IF NON$ZERO$COUNT > PLOSIVE$LIMIT THEN
2          RETURN;
2      IF (TEMP(1) < VOICED$ENERGY) THEN RETURN;
2      MAXIMUM$RATIO = VOICING$RATIO;
2      DO I = 0 TO 13;
3          IF R(I).RWORD(0) > MAXIMUM$RATIO THEN
3              DO;
4                  MAXIMUM$RATIO = R(I).RWORD(0);
4                  PITCH = SHL(I,2) + LOWEST$PITCH;
4                  END;
3              END;
2      END SIFT;
1      END PITCH$ANALYSIS;

```

FILE INFORMATION:

```

CODE AREA SIZE      = 05FCH   1532D
VARIABLE AREA SIZE = 0027H    39D
MAXIMUM STACK SIZE = 0006H    6D
237 LINES READ
0 PROGRAM ERROR(S)

```

OF PL/M-80 COMPILATION

PL/M-80 V3.0 COMPILATION OF MODULE ANALYSIS
 OBJECT MODULE PLACED IN :F1:ANADIF.OBJ
 COMPILER INVOKED BY: PL/M80 :F1:ANADIF.SRC WORKFILES(:FO:,:FO:)

```

$PAGewidth(80)
/*****
**
**          SPEECH SYNTHESIS SYSTEM          **
**      REFLECTION COEFFICIENT ANALYSIS      **
**      USING GUEGUEN LE-RUEX ALGORITHM      **
**          (WITH PRE-EMPHASIZE)             **
**
**      FILE NAME      : ANADIF.SRC          DATE
**      MODULE DEF     : Analyses soeech    -----
**      AUTHOR         : Onder Bicioglu     22 06 85
**
**      *****/
ANALYSIS: DO;

1  1  DECLARE EP(17) STRUCTURE(
2      PWORD(2) ADDRESS) PUBLIC;
3  1  DECLARE R(25) STRUCTURE(
4      RWORD(2) ADDRESS) PUBLIC;
5  1  DECLARE EN(17) STRUCTURE(
6      NWORD(2) ADDRESS) PUBLIC;
7  1  DECLARE K(17) STRUCTURE(
8      KDIGIT(4) BYTE) PUBLIC;
9  1  DECLARE SIGNAL(400) STRUCTURE(
10     SDIGIT(4) BYTE) PUBLIC;
11 1  DECLARE W(1) STRUCTURE(
12     WDIGIT(4) BYTE) EXTERNAL;
13 1  DECLARE SAMPLE(1) BYTE EXTERNAL;

14 1  DECLARE (PITCH,ORDER) BYTE EXTERNAL;
15 1  DECLARE NSAMPLE ADDRESS PUBLIC;
16 1  DECLARE E(4) BYTE PUBLIC;
17 1  DECLARE EDUMMY(4) BYTE PUBLIC;
18 1  DECLARE BDUMMY(4) BYTE PUBLIC;
19 1  DECLARE TEMP(2) ADDRESS PUBLIC;
20 1  DECLARE OVER$FLOW$FLAG BYTE PUBLIC;

21 1  DECLARE (M,N,I,J) ADDRESS;
22 1  DECLARE ZERO(*) BYTE DATA(0,0,0,0);

23 1  FBADD: PROCEDURE(OP1,OP2,OP3) EXTERNAL;
24 2  DECLARE (OP1,OP2,OP3) ADDRESS;
25 2  END FBADD;
26 1  FBSUB: PROCEDURE(OP1,OP2,OP3) EXTERNAL;
27 2  DECLARE (OP1,OP2,OP3) ADDRESS;
28 2  END FBSUB;
29 1  FBDIV: PROCEDURE(OP1,OP2,OP3) EXTERNAL;
30 2  DECLARE (OP1,OP2,OP3) ADDRESS;
31 2  END FBDIV;
32 1  FBMUL: PROCEDURE(OP1,OP2,OP3) EXTERNAL;
33 2  DECLARE (OP1,OP2,OP3) ADDRESS;

```

```

09 2      END FBMUL;
10 1      A$LAW$EXPANSION: PROCEDURE EXTERNAL;
11 2      END A$LAW$EXPANSION;

/*****
*  PROCEDURE : GUEGUEN$LE$RUEX
*  FUNCTION  : Analysis using Gueguen Le-Ruex
*              algorithm
*  PLM CALL   : CALL GUEGUEN$LE$RUEX;
*  INPUT      : PITCH =pitch period
*              SAMPLE = 8-bit samples
*  OUTPUT     : K=reflection coefficients
*  GLOBALS    : K, R, ORDER, NSAMPLE, SIGNAL
*  CALLS      : FBMUL, FBDIV, FBADD, A$LAW$EXPANSION
*/

12 1      GUEGUEN$LE$RUEX: PROCEDURE PUBLIC;

/*-----INITIALIZATION----- */

13 2          DO M = 1 TO 10;
14 3              DO J = 0 TO 1;
15 4                  EP(M).PWORD(J) = R(M).RWORD(J);
16 4                  EN(M).NWORD(J) = R(M-1).RWORD(J);
17 4              END;
18 3          END;
19 2          OVERFLOW$FLAG = 0;

/*----- G U E G U E N   L E - R U E X -----*/

20 2          DO I = 0 TO 9;

/*-- K(I) = - (EP(I+1) / EN(1))----- */

21 3              CALL FBDIV(.EP(I+1).PWORD(0), .EN(1).NWORD(0),
                           .EDUMMY(0));

/*-----RANGE CHECK-----*/

22 3              IF (EDUMMY(2) > 0) AND (EDUMMY(2) < OFFH)
23 4                  THEN DO;
24 4                      OVERFLOW$FLAG = OFFH;
25 4                      RETURN;
26 4                  END;

/*-----*/

27 3              CALL FBSUB(.EDUMMY(0), .ZERO(0),
                           .K(I+1).KDIGIT(0));
28 3              CALL FBMUL(.K(I+1).KDIGIT(0), .EP(I+1).PWORD(0),
                           .EDUMMY(0));
29 3              CALL FBADD(.EN(1).NWORD(0), .EDUMMY(0),
                           .EN(1).NWORD(0));
30 3              CALL FBMUL(.EN(10-I).NWORD(0), .K(I+1).KDIGIT(0),
                           .EDUMMY(0));
31 3              CALL FBADD(.EP(10).PWORD(0), .EDUMMY(0),
                           .EP(10).PWORD(0));

```

```

3      IF I = 9 THEN
3      RETURN;

3      DO M = I+2 TO 9;
4      CALL FBMUL(.K(I+1).KDIGIT(0),.EN(M-I).NWORD(0),
4      .EDUMMY(0));
4      CALL FBADD(.EDUMMY(0),.EP(M).PWORD(0),
4      .E(0));
4      CALL FBMUL(.EP(M).PWORD(0),.K(I+1).KDIGIT(0),
4      .EDUMMY(0));
4      CALL FBADD(.EDUMMY(0),.EN(M-I).NWORD(0),
4      .EN(M-I).NWORD(0));
4      CALL FBADD(.E(0),.ZERO(0),.EP(M).PWORD(0));
4      END;
3      END;

2      END GUEGUEN$LE$RUEX;
/******
*   PROCEDURE : ANALYSER
*   FUNCTION   : Analyses the signal and creates
*                 reflection coefficients using auto-
*                 correlation method and
*                 GUEGUEN LE-RUEX algorithm
*   PLM CALL   : CALL ANALYSER;
*   INPUT      : ORDER = order of the prediction
*                 NSAMPLE = number of samples to be used
*   OUTPUT     : K(I).KDIGIT(J) = reflection coefficients
*                 OVERFLOW$FLAG = OFFH if any of the k's
*                 exceeds 1 in magnitude
*   GLOBALS    : SIGNAL structure
*   CALLS      : FBMUL, FBDIV, FBADD, FBSUB, GUEGUEN$LE$RUEX
*/
53     1      ANALYSER: PROCEDURE PUBLIC;

64     2      CALL A$LAW$EXPANSION;

65     2      IF PITCH <> 0 THEN
66     2      DO;
67     3      DO I = 1 TO NSAMPLE;
68     4      CALL FBSUB(.SIGNAL(I-1).SDIGIT(0),.SIGNAL(I).SDIGIT(0),
68     4      .SIGNAL(I-1).SDIGIT(0));
69     4      END;
70     3      END;

/*-----W I N D O W I N G -----*/

71     2      DO I = 0 TO SHR(NSAMPLE,1)-1;
72     3      CALL FBMUL(.SIGNAL(I).SDIGIT(0),.W(I).WDIGIT(0),
72     3      .SIGNAL(I).SDIGIT(0));
73     3      CALL FBMUL(.SIGNAL(NSAMPLE-1-I).SDIGIT(0),.W(I).WDIGIT(0),
73     3      .SIGNAL(NSAMPLE-1-I).SDIGIT(0));
74     3      END;

/*-----C A L C U L A T I O N   O F   R ( I )-----*/

75     2      DO I = 0 TO 10;
76     3      DO J = 0 TO 1;

```

```
77 4      R(I).RWORD(J) = 0;  
78 4      END;  
79 3      DO M = 0 TO NSAMPLE-1-I;  
80 4          CALL FBMDL(.SIGNAL(M).SDIGIT(0),  
81 4              .SIGNAL(M+1).SDIGIT(0), .TEMP(0));  
82 4          CALL FBADD(.TEMP(0), .R(I).RWORD(0),  
83 4              .R(I).RWORD(0));  
84 4      END;  
85 3      END;  
86 2      CALL GUEGUEN$LE$RUEX;  
87 2      END ANALYSER;  
88 1      END ANALYSIS;
```

MODULE INFORMATION:

```
CODE AREA SIZE      = 037AH      890D  
VARIABLE AREA SIZE = 078BH      1931D  
MAXIMUM STACK SIZE = 0008H      8D  
180 LINES READ  
0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

B-II PL/M-80 V3.0 COMPILATION OF MODULE MAIN
 OBJECT MODULES PLACED IN :F1:SMOOTH.OBJ
 COMPILER INVOKED BY: PL/M80 :F1:SMOOTH.SRC

\$PAGEWIDTH(80)

```

/*****
**
**          SPEECH SYNTHESIS SYSTEM          **
**          SMOOTHER MODULE                  **
**
**  FILE NAME      : SMOOTH.SRC              DATE          **
**  MODULE DEF     : Enables user to        -----      **
**                  smooth results          16 06 1985      **
**  AUTHOR         : Onder Bicioglu         **
**
*****/

```

```

1  MAIN: DO;

2  1  DISPLAY$MESSAGE: PROCEDURE(POINTER) EXTERNAL;
3  2  DECLARE POINTER ADDRESS;
4  2  END DISPLAY$MESSAGE;
5  1  PRINT$MESSAGE: PROCEDURE(POINTER) EXTERNAL;
6  2  DECLARE POINTER ADDRESS;
7  2  END PRINT$MESSAGE;
8  1  PRINT$NUMBER: PROCEDURE(ND) EXTERNAL;
9  2  DECLARE ND BYTE;
10 2  END PRINT$NUMBER;
11 1  GET$CHARACTERS: PROCEDURE EXTERNAL;
12 2  END GET$CHARACTERS;
13 1  CI: PROCEDURE BYTE EXTERNAL;
14 2  END CI;
15 1  LO: PROCEDURE(MES) EXTERNAL;
16 2  DECLARE MES BYTE;
17 2  END LO;
18 1  EXIT: PROCEDURE EXTERNAL;
19 2  END EXIT;
20 1  IOSET: PROCEDURE(MASK) EXTERNAL;
21 2  DECLARE MASK BYTE;
22 2  END IOSET;
23 1  OPEN:
    PROCEDURE(AFT, FILE, ACCESS, MODE, STATUS) EXTERNAL;
24 2  DECLARE (AFT, FILE, ACCESS, MODE, STATUS) ADDRESS;
25 2  END OPEN;
26 1  READ:
    PROCEDURE(AFT, BUF, CNT, ACT, STATUS) EXTERNAL;
27 2  DECLARE (AFT, BUF, CNT, ACT, STATUS) ADDRESS;
28 2  END READ;
29 1  WRITE:
    PROCEDURE(AFT, BUF, CNT, STATUS) EXTERNAL;
30 2  DECLARE (AFT, BUF, CNT, STATUS) ADDRESS;
31 2  END WRITE;
32 1  CLOSE:
    PROCEDURE(AFT, STATUS) EXTERNAL;
33 2  DECLARE (AFT, STATUS) ADDRESS;
34 2  END CLOSE;

```

```

5 1 FBDIV: PROCEDURE(OP1,OP2,OP3) EXTERNAL;
6 2   DECLARE (OP1,OP2,OP3) ADDRESS;
7 2   END FBDIV;

8 1   DECLARE BUFFER(1) BYTE EXTERNAL;
9 1   DECLARE (I,J,L,M,N) ADDRESS;
0 1   DECLARE (FILE,STATUS,NFRAME,CQUNT) ADDRESS;
1 1   DECLARE (LPC$POINTER, RAM$POINTER) ADDRESS;
2 1   DECLARE END$OF$BUFFER ADDRESS;
3 1   DECLARE RAM BASED RAM$POINTER BYTE;
4 1   DECLARE LPC BASED LPC$POINTER (20) BYTE;

5 1   DECLARE ASK$SMOOTH$FILE(*) BYTE DATA(OAH,ODH,
6 1   'File name of smoothed data :',OAH,ODH,00);
7 1   DECLARE ERROR(*) BYTE DATA(
8 1   'Error',OAH,ODH,00);
9 1   DECLARE ASK$PRINTER(*) BYTE DATA(
0 1   'Do You want to take print-out?',OAH,ODH,00);
1 1   DECLARE ASK$LPC$FILE(*) BYTE DATA(OAH,ODH,
2 1   'File name for LPC coefficients :',00);
3 1   DECLARE ASK$INSERT$FILE(*) BYTE DATA(
4 1   'File to be inserted :',00);
5 1   DECLARE PROMPT(*) BYTE DATA(OAH,ODH,')',00);
6 1   DECLARE HEADING(*) BYTE DATA(OAH,ODH,
7 1   'Frame RMS R P k1 k2 k3 k4 k5 k6',
8 1   ' k7 k8 k9 k10 ',OAH,ODH,00);

```

```

/*****
*   PROCEDURE : CONVERT
*   FUNCTION   : converts ASCII string to number
*   PLM CALL   : I = CONVERT(ASCII);
*   INPUT      : ASCII = ASCII character
*   OUTPUT     : returns number
*   GLOBALS    : none
*   CALLS      : none
*/

```

```

52 1   CONVERT: PROCEDURE(ASCII) BYTE PUBLIC;

53 2   DECLARE ASCII BYTE;
54 2   DECLARE CONVERT$TABLE(*) BYTE DATA(
55 2   '0','1','2','3','4','5','6','7',
56 2   '8','9','A','B','C','D','E','F');

57 2   DO I = 0 TO OFH;
58 3   IF ASCII = CONVERT$TABLE(I)
59 3   THEN RETURN I;
60 3   END;

61 2   END CONVERT;

```

```

/*****
*   PROCEDURE : CHANGE$FRAME
*   FUNCTION   : changes pointed entry of selected
*               : frame to the given value
*   PLM CALL   : CALL CHANGE$FRAME;
*   INPUT      : BUFFER = ABCDEFG
*               : A = 'F'
*

```

```

*           B = number
*           C = number
*           D = 'P', 'E', 'R' or 'K'
*           E = number
*           F = number
*           G = number
*   OUTPUT      : changed frame
*   GLOBALS     : LPC$POINTER, LPC, BUFFER
*   CALLS       : none
*/
50  1  CHANGE$FRAME: PROCEDURE PUBLIC;
51  2      N = CONVERT(BUFFER(1));
52  2      N = SHL(N,4) OR CONVERT(BUFFER(2));
53  2      DO I = 0 TO N-1;
54  3          LPC$POINTER = LPC$POINTER + 18;
55  3      END;
56  2      LPC$POINTER = LPC$POINTER - 18;
57  2      IF BUFFER(3) = 'P' THEN
58  2          DO;
59  3              LPC(2) = CONVERT(BUFFER(4));
60  3              LPC(2) = SHL(LPC(2),4) OR CONVERT(BUFFER(5));
61  3              RETURN;
62  3          END;
63  2      IF BUFFER(3) = 'E' THEN
64  2          DO;
65  3              LPC(0) = CONVERT(BUFFER(4));
66  3              RETURN;
67  3          END;
68  2      IF BUFFER(3) = 'R' THEN
69  2          DO;
70  3              LPC(1) = CONVERT(BUFFER(4));
71  3              RETURN;
72  3          END;
73  2      IF BUFFER(3) = 'K' THEN
74  2          DO;
75  3              N = 2 + CONVERT(BUFFER(4));
76  3              LPC(N) = CONVERT(BUFFER(5));
77  3              LPC(N) = SHL(LPC(N),4) OR CONVERT(BUFFER(6));
78  3              RETURN;
79  3          END;
80  2      CALL DISPLAY$MESSAGE(.ERROR(0));
81  2
82  2  END CHANGE$FRAME;

/*****
*   PROCEDURE : CREATE$FILE
*   FUNCTION  : Creates a file containing smoothed LPC
*               data
*   ALM CALL  : CALL CREATE$FILE;
*   INPUT     : 6000H-7FFFH .....LPC data
*   OUTPUT    : A new file
*****/

```

```
* GLOBALS      : LPC$POINTER, RAM$POINTER
* CALLS        : OPEN, WRITE, CLOSE
*/
```

```
2  1  CREATE$FILE: PROCEDURE PUBLIC;
3  2      CALL DISPLAY$MESSAGE(.ASK$PRINTER(0));
4  2      IF (CI AND 7FH) = 'Y' THEN
5  2          CALL IOSET(81H);
6  2      ELSE
7  2          CALL IOSET(41H);
8  2
9  2      CALL DISPLAY$MESSAGE(.ASK$SMOOTH$FILE(0));
10 2      CALL GET$CHARACTERS;
11 2      CALL OPEN(.M, .BUFFER(0), 2, 0, .STATUS);
12 2      RAM$POINTER = 8000H;
13 2      J = 1;
14 2      L = 0;
15 2      DO WHILE LPC$POINTER < END$OF$BUFFER;
16 3          IF L = 0 THEN
17 3              DO;
18 4                  CALL LD(0CH);
19 4                  CALL PRINT$MESSAGE(.HEADING(0));
20 4              END;
21 3              CALL LD(' ');
22 3              CALL PRINT$NUMBER(J);
23 3              CALL LD(' ');
24 3              J = J + 1;
25 3              IF LPC(0) = 0 THEN
26 3                  N = 0;
27 3              ELSE DO;
28 4                  IF LPC(1) = 1 THEN
29 4                      N = 2;
30 4                  ELSE DO;
31 5                      IF LPC(2) = 0 THEN
32 5                          N = 6;
33 5                      ELSE
34 5                          N = 12;
35 5                  END;
36 4              END;
37 3              DO I = 0 TO N;
38 4                  RAM = LPC(I);
39 4                  CALL PRINT$NUMBER(RAM);
40 4                  CALL LD(' ');
41 4                  RAM$POINTER = RAM$POINTER + 1;
42 4              END;
43 3              LPC$POINTER = LPC$POINTER + 18;
44 3              CALL LD(0AH);
45 3              CALL LD(0DH);
46 3              L = L + 1;
47 3              IF L = 60 THEN L = 0;
48 3          END;
49 2
50 2      RAM = 0FH;
51 2
52 2      COUNT = RAM$POINTER - 7FFFH;
53 2      CALL WRITE(M, 8000H, COUNT, .STATUS);
```



```

2      CALL CLOSE(M,.STATUS);
2      CALL IOSET(41H);
2      END CREATE$FILE;
/*****
*   PROCEDURE : SAVE
*   FUNCTION   : Creates a file containing smoothed row
*                 LPC data
*   PLM CALL   : CALL SAVE;
*   INPUT      : 6000H-7FFFH .....LPC data
*   OUTPUT     : A new file
*   GLOBALS    : LPC$POINTER, RAM$POINTER
*   CALLS      : OPEN,WRITE,CLOSE
*/
1      SAVE: PROCEDURE PUBLIC;
2          IF BUFFER(1) = 'X' THEN
2              DO;
3                  CALL DISPLAY$MESSAGE(.ASK$SMOOTH$FILE(0));
3                  CALL GET$CHARACTERS;
3                  CALL OPEN(.FILE,.BUFFER(0),2,0,.STATUS);
3                  COUNT = END$OF$BUFFER - 6000H;
3                  CALL WRITE(FILE,6000H,COUNT,.STATUS);
3                  CALL CLOSE(FILE,.STATUS);
3              END;
2          CALL EXIT;
2      END SAVE;

/*****
*   PROCEDURE : DISPLAY$FRAME
*   FUNCTION   : Displays frame data
*   PLM CALL   : CALL DISPLAY$FRAME
*   INPUT      : BUFFER = ABC
*                 A = 'D'
*                 B = number
*                 C = number
*   OUTPUT     : Displays frame data
*   GLOBALS    : LPC$POINTER
*   CALLS      : DISPLAY$MESSAGE, PRINT$NUMBER
*/
1      DISPLAY$FRAME: PROCEDURE PUBLIC;
2          N = CONVERT(BUFFER(1));
2          N = SHL(N,4) OR CONVERT(BUFFER(2));
2
2          DO I = 0 TO N-1;
3              LPC$POINTER = LPC$POINTER + 18;
3          END;
2          LPC$POINTER = LPC$POINTER - 18;
2          CALL PRINT$MESSAGE(.HEADING(0));
2          CALL LO(' ');
2          CALL PRINT$NUMBER(N);
2          CALL LO(' ');
2          DO I = 0 TO 17;
3              CALL PRINT$NUMBER(LPC(I));

```

```

3      CALL LO(' ');
3      END;

2  END DISPLAY$FRAME;
  /*****
  *  PROCEDURE : CHANGE$ENERGY
  *  FUNCTION  : Changes energy levels
  *  PLM CALL  : CALL CHANGE$ENERGY;
  *  INPUT     : LPC$POINTER points to data block
  *  OUTPUT    : Changed energy levels
  *  GLOBALS   : LPC$POINTER, BUFFER
  *  CALLS     : CONVERT
  */
1  1  CHANGE$ENERGY: PROCEDURE PUBLIC;

2  2  DECLARE RMS(*) ADDRESS DATA(
      0,52,87,123,174,246,348,491,
      694,981,1385,1957,2764,3904,5514,7789);
3  2  DECLARE E(4) BYTE;
4  2  DECLARE DUMMY ADDRESS;
5  2  DECLARE TEMP(2) ADDRESS;

6  2  E(1) = CONVERT(BUFFER(1));
7  2  E(1) = SHL(E(1),4) OR CONVERT(BUFFER(2));
8  2  E(2) = CONVERT(BUFFER(3));
9  2  E(2) = SHL(E(2),4) OR CONVERT(BUFFER(4));
0  2  E(3) = 0;

1  2  DO WHILE LPC$POINTER < END$OF$BUFFER;

2  3  DUMMY = LPC(14);
3  3  TEMP(0) = LPC(13);
4  3  TEMP(0) = TEMP(0) OR SHL(DUMMY,8);
5  3  TEMP(1) = 0;

6  3  CALL FBDIV(.TEMP(0),.E(0),.TEMP(0));
7  3  DO I = 0 TO 14;
8  4  IF TEMP(0) < RMS(I) THEN GO TO RMS$FOUND;
0  4  END;
1  3  I = 15;
2  3  RMS$FOUND:
4  3  IF (LPC(0) = 0) AND (I = 15) THEN;
5  3  ELSE LPC(0) = I-1;
6  3  LPC$POINTER = LPC$POINTER + 16;

7  3  END;

2  END CHANGE$ENERGY;

  /*****
  *  PROCEDURE : REMOVE
  *  FUNCTION  : Removes indicated frames from block
  *  PLM CALL  : CALL REMOVE;
  *  INPUT     : LPC$POINTER points to data block
  *             BUFFER(1),BUFFER(2) = Starting frame
  *             BUFFER(3),BUFFER(4) = End frame
  *  OUTPUT    : New block does not contain removed
  */

```

```

*               frames
* GLOBALS      : LPC$POINTER, BUFFER, RAMPOINTER
* CALLS        : CONVERT
*/
1 REMOVE: PROCEDURE PUBLIC;
2
2     N = CONVERT(BUFFER(1));
2     N = SHL(N,4) OR CONVERT(BUFFER(2));
2     I = CONVERT(BUFFER(3));
2     I = SHL(I,4) OR CONVERT(BUFFER(4));
2
2     DO J = 1 TO N;
3         LPC$POINTER = LPC$POINTER + 18;
3     END;
2     RAM$POINTER = LPC$POINTER;
2     DO J = N+1 TO I;
3         RAM$POINTER = RAM$POINTER + 18;
3     END;
2
2     DO WHILE RAM$POINTER < END$OF$BUFFER;
3         DO J = 0 TO 17;
4             LPC(J) = RAM;
4             RAM$POINTER = RAM$POINTER + 1;
4         END;
3         LPC$POINTER = LPC$POINTER + 18;
3     END;
2     END$OF$BUFFER = LPC$POINTER;
2
2 END REMOVE;

/*****
* PROCEDURE : INSERT
* FUNCTION  : Inserts selected file after the
*             indicated frame
* PLM CALL  : CALL INSERT;
* INPUT     : LPC$POINTER points to data block
*             BUFFER(1), BUFFER(2) = Starting frame
* OUTPUT    : New block contains inserted frames
* GLOBALS   : LPC$POINTER, BUFFER, RAMPOINTER
* CALLS     : CONVERT
*/
1 INSERT: PROCEDURE PUBLIC;
2
2     N = CONVERT(BUFFER(1));
2     N = SHL(N,4) OR CONVERT(BUFFER(2));
2
2     RAM$POINTER = 8000H;
2     DO J = 1 TO N;
3         DO I = 0 TO 17;
4             RAM = LPC(I);
4             RAM$POINTER = RAM$POINTER + 1;
4         END;
3         LPC$POINTER = LPC$POINTER + 18;
3     END;
2 ASK$AGAIN:
2     CALL DISPLAY$MESSAGE(.ASK$INSERT$FILE(0));
2     CALL GET$CHARACTERS;

```

```

2      CALL OPEN(.FILE,.BUFFER(0),1,0,.STATUS);
3      IF STATUS <> 0 THEN
4      DO;
5      3      CALL DISPLAY$MESSAGE(.ERROR(0));
6      3      GO TO ASK$AGAIN;
7      3      END;
8      2      CALL READ(FILE, RAM$POINTER, 2000H, .COUNT, .STATUS);
9      2      CALL CLOSE(FILE, .STATUS);
10     2      END$OF$BUFFER = END$OF$BUFFER + COUNT;
11     2      IF END$OF$BUFFER > 7FFFH THEN
12     2      DO;
13     3      CALL DISPLAYMESSAGE(.ERROR(0));
14     3      END$OF$BUFFER = END$OF$BUFFER - COUNT;
15     3      RETURN;
16     3      END;

7     2      RAM$POINTER = RAM$POINTER + COUNT;
8     2      DO WHILE LPC$POINTER < (END$OF$BUFFER-COUNT);
9     3      DO J = 0 TO 17;
10    4      RAM = LPC(J);
11    4      RAM$POINTER = RAM$POINTER + 1;
12    4      END;
13    3      LPC$POINTER = LPC$POINTER + 18;
14    3      END;
15    2      RAM$POINTER = 8000H;
16    2      LPC$POINTER = 6000H;

7     2      DO WHILE LPC$POINTER < END$OF$BUFFER;
8     3      DO J = 0 TO 17;
9     4      LPC(J) = RAM;
10    4      RAM$POINTER = RAM$POINTER + 1;
11    4      END;
12    3      LPC$POINTER = LPC$POINTER + 18;
13    3      END;

4     2      END INSERT;

/*****/

5     1      DECLARE BEGIN LABEL PUBLIC;
6     1      BEGIN:
7     1      CALL IOSET(41H);
8     1      CALL DISPLAY$MESSAGE(.ASK$LPC$FILE(0));
9     1      CALL GETCHARACTERS;
10    1      CALL OPEN(.FILE,.BUFFER(0),1,0,.STATUS);
11    1      IF STATUS <> 0 THEN
12    1      DO;
13    2      CALL DISPLAY$MESSAGE(.ERROR(0));
14    2      GO TO BEGIN;
15    2      END;
16    1      ELSE DO;
17    2      CALL READ(FILE, 6000H, 2000H, .COUNT, .STATUS);
18    2      END$OF$BUFFER = 6000H + COUNT;
19    2      CALL CLOSE(FILE, .STATUS);
20    2      END;
21    1      DO WHILE 1;
22    2      LOOP:

```

```

2      CALL DISPLAY$MESSAGE(, PROMPT(0));
2      CALL GET$CHARACTERS;
2      LDC$POINTER = 6000H;
2      IF BUFFER(0) = 'S' THEN
2          DO;
3              CALL CHANGE$ENERGY;
3              GO TO LOOP;
2          END;
2      IF BUFFER(0) = 'F' THEN
2          DO;
3              CALL CHANGE$FRAME;
3              GO TO LOOP;
2          END;
2      IF BUFFER(0) = 'D' THEN
2          DO;
3              CALL DISPLAY$FRAME;
3              GO TO LOOP;
2          END;
2      IF BUFFER(0) = 'E' THEN
2          DO;
3              CALL SAVE;
3              GO TO LOOP;
2          END;
2      IF BUFFER(0) = 'C' THEN
2          DO;
3              CALL CREATE$FILE;
3              GO TO LOOP;
2          END;
2      IF BUFFER(0) = 'R' THEN
2          DO;
3              CALL REMOVE;
3              GO TO LOOP;
2          END;
2      IF BUFFER(0) = 'I' THEN
2          DO;
3              CALL INSERT;
3              GO TO LOOP;
2          END;
2      END;
1      END MAIN;

```

LE INFORMATION:

```

CODE AREA SIZE      = 0916H   2326D
VARIABLE AREA SIZE  = 0023H    35D
MAXIMUM STACK SIZE  = 000AH    10D
491 LINES READ
0 PROGRAM ERROR(S)

```

OF PL/M-80 COMPILATION

II PL/M-80 V3.0 COMPILATION OF MODULE GRAPHMODULE
 AT MODULE PLACED IN :F1:SHOW.OBJ
 LER INVOKED BY: PLM80 :F1:SHOW.SRC WORKFILES(:FO:,:FO:)

\$PAGEWIDTH(80)

```

/*****
**
**          SPEECH SYNTHESIS SYSTEM
**          GRAPH DISPLAY MODULE
**
**  FILE NAME      : SHOW.SRC          DATE
**  MODULE DEF     : Main Display     -----
**  AUTHOR         : Onder Bicioglu   22 06 85
**
*****/

```

GRAPH\$MODULE:

DO;

```

1  DECLARE I BYTE;
1  DECLARE (FILE,COUNT,STATUS) ADDRESS;

1  DECLARE KEY BYTE PUBLIC;
1  DECLARE (LEFT,RIGHT,UP,DOWN,
           HOME,CLEAR,LI) BYTE PUBLIC;
1  DECLARE (TIME$BASE,AMPLITUDE$BASE,START)
           ADDRESS EXTERNAL;
1  DECLARE DATABUF(1) BYTE EXTERNAL;
1  DECLARE BUFFER(1) BYTE EXTERNAL;
1  DECLARE MDS BYTE PUBLIC;

1  DECLARE CR          LITERALLY '13',
           HORIZONTALUNITS LITERALLY '8',
           VERTICALUNITS  LITERALLY '8',
           VCLEFT         LITERALLY '8',
           VCRIGHT        LITERALLY '10H',
           VCU            LITERALLY '0CH',
           VCDOWN         LITERALLY '0BH',
           VCHOME         LITERALLY '12H',
           VCCLEAR        LITERALLY '1CH',
           VCLI           LITERALLY '1BH',
           INTELEFT       LITERALLY '44H',
           INTELRIGHT     LITERALLY '43H',
           INTELUP        LITERALLY '41H',
           INTELDOWN      LITERALLY '42H',
           INTELHOME      LITERALLY '48H',
           INTELCLEAR     LITERALLY '45H',
           INTELLI        LITERALLY '1BH';

1  DECLARE (DESPTR,SRCPTR) ADDRESS;
1  DECLARE DES BASED DESPTR BYTE;
1  DECLARE SRC BASED SRCPTR BYTE;
1  DECLARE UPPER(*) BYTE DATA(VCLI,VCCLEAR,VCLI,
           VCDOWN,VCLI,VCDOWN,CR,' ',' ',0);
1  DECLARE YAXIS(*) BYTE DATA('-',VCLI,VCDOWN,
           VCLEFT,'I',VCLI,VCDOWN,VCLEFT,00,00,00);

```

```

16 1 DECLARE XAXIS(*) BYTE DATA('-----I',00);
17 1 DECLARE BASEDATA(*) BYTE DATA(' Timebase :',
    ' 01 msec Amplitudebase : 20 Units ',
    ' Origin : 0000 msec',00);

18 1 DECLARE ASK$SYSTEM(*) BYTE DATA(OAH,ODH,
    'Is this new MDS ? ',OAH,ODH,00);
19 1 DECLARE ASK$SHOW$FILE(*) BYTE DATA(OAH,ODH,
    'Enter file name to be displayed',OAH,ODH,00);
20 1 DECLARE ERROR(*) BYTE DATA(OAH,ODH,
    'ERROR',OAH,ODH,00);
21 1 DECLARE VC414DATA(*) BYTE DATA(
    VCLEFT,VCRIGHT,VCUP,VCDOWN,VCHOME,
    VCCLEAR,VCLI);
22 1 DECLARE INTEL$DATA(*) BYTE DATA(
    INTELEFT,INTELRIGHT,INTELUP,INTELDOWN,
    INTELHOME,INTELCLEAR,INTELLI);
23 1 DECLARE INTELTABLE(*) BYTE DATA(
    INTELLI,INTELCLEAR,INTELLI,INTELDOWN,
    INTELLI,INTELDOWN,CR,' ',' ',0,
    '-',INTELLI,INTELDOWN,INTELLI,INTELEFT,'I',
    INTELLI,INTELDOWN,INTELLI,INTELEFT,00);

    /*****/

24 1 CO: PROCEDURE (CHARACTER) EXTERNAL;
25 2 DECLARE CHARACTER BYTE;
26 2 END CO;
27 1 CI: PROCEDURE BYTE EXTERNAL;
28 2 END CI;
29 1 MOVECURSOR: PROCEDURE (DIRECTION,COUNT) EXTERNAL;
30 2 DECLARE (DIRECTION,COUNT) BYTE;
31 2 END MOVECURSOR;
32 1 CHANGETIMEBASE: PROCEDURE EXTERNAL;
33 2 END CHANGETIMEBASE;
34 1 CHANGEAMPLITUDEBASE: PROCEDURE EXTERNAL;
35 2 END CHANGEAMPLITUDEBASE;
36 1 SHIFTLEFT: PROCEDURE EXTERNAL;
37 2 END SHIFTLEFT;
38 1 SHIFTRIGHT: PROCEDURE EXTERNAL;
39 2 END SHIFTRIGHT;
40 1 CHANGE$ORIGIN: PROCEDURE EXTERNAL;
41 2 END CHANGE$ORIGIN;
42 1 DRAWGRAPH: PROCEDURE EXTERNAL;
43 2 END DRAWGRAPH;
44 1 DISPLAY$MESSAGE: PROCEDURE (POINTER) EXTERNAL;
45 2 DECLARE POINTER ADDRESS;
46 2 END DISPLAY$MESSAGE;
47 1 EXIT: PROCEDURE EXTERNAL;
48 2 END EXIT;
49 1 OPEN: PROCEDURE (P1,P2,P3,P4,P5) EXTERNAL;
50 2 DECLARE (P1,P2,P3,P4,P5) ADDRESS;
51 2 END OPEN;
52 1 READ: PROCEDURE (P1,P2,P3,P4,P5) EXTERNAL;
53 2 DECLARE (P1,P2,P3,P4,P5) ADDRESS;
54 2 END READ;
55 1 CLOSE: PROCEDURE (P1,P2) EXTERNAL;

```

```

2      DECLARE (P1,P2) ADDRESS;
2      END CLOSE;
1      GETCHARACTERS: PROCEDURE EXTERNAL;
2      END GET$CHARACTERS;
/*****
0      1      BEGIN;
1      CALL DISPLAY$MESSAGE(.ASK$SHOW$FILE(0));
1      CALL GET$CHARACTERS;
2      1      CALL OPEN(.FILE,.BUFFER(0),1,0,.STATUS);
3      1      IF STATUS <> 0 THEN
4      1      DO;
5      2      CALL DISPLAY$MESSAGE(.ERROR(0));
6      2      GO TO BEGIN;
7      2      END;
8      1      CALL READ(FILE,.DATABUF(0),4000H,.COUNT,.STATUS);
9      1      CALL CLOSE(FILE,.STATUS);

0      1      CALL DISPLAY$MESSAGE(.ASK$SYSTEM(0));
1      1      IF (CI AND 7FH) = 'Y' THEN
2      1      DO;
3      2      SRCPTR = .INTEL$TABLE(0);
4      2      DESPTR = .UPPER(0);
5      2      DO I=1 TO 21;
6      3      DES = SRC;
7      3      DESPTR = DESPTR + 1;
8      3      SRCPTR = SRCPTR + 1;
9      3      END;
0      2      MDS = 1;
1      2      SRCPTR = .INTEL$DATA(0);
2      2      END;
3      1      ELSE DO;
4      2      MDS = 0;
5      2      SRCPTR = .VC414DATA(0);
6      2      END;
7      1      DESPTR = .LEFT;
8      1      DO I = 1 TO 7;
9      2      DES = SRC;
0      2      DESPTR = DESPTR + 1;
1      2      SRCPTR = SRCPTR + 1;
2      2      END;

3      1      START=.DATABUF(0);
4      1      TIMEBASE=1;
5      1      AMPLITUDEBASE=32;

6      1      CALL DISPLAY$MESSAGE(.UPPER(00));
7      1      DO I=1 TO VERTICALUNITS;
8      2      CALL DISPLAY$MESSAGE(.YAXIS(0));
9      2      END;
0      1      CALL CO(' ');
1      1      CALL MOVECURSOR(UP,VERTICALUNITS);
2      1      DO I=1 TO HORIZONTALUNITS;
3      2      CALL DISPLAY$MESSAGE(.XAXIS(0));
4      2      END;

5      1      CALL MOVECURSOR(LEFT,8*HORIZONTALUNITS);
6      1      CALL MOVECURSOR(DOWN,VERTICALUNITS+3);

```



```
1      CALL DISPLAY$MESSAGE(.,BASEDATA(0));
1      CALL CO(LI);
1      CALL CO(HOME);
1      CALL DRAWGRAPH;

1      DO WHILE 1;
2          KEY=CI AND 7FH;
2          IF KEY='E' THEN CALL EXIT;
2          IF KEY='T' THEN CALL CHANGE$TIME$BASE;
2          IF KEY='A' THEN CALL CHANGE$AMPLITUDE$BASE;
2          IF KEY=RIGHT THEN CALL SHIFTRIGHT;
2          IF KEY=LEFT THEN CALL SHIFTLEFT;
2          IF KEY='D' THEN CALL CHANGE$ORIGIN;
2      END;

1      END GRAPHMODULE;
```

MODULE INFORMATION:

CODE AREA SIZE	= 0291H	657D
VARIABLE AREA SIZE	= 0014H	20D
MAXIMUM STACK SIZE	= 0008H	8D
181 LINES READ		
0 PROGRAM ERROR(S)		

OF PL/M-80 COMPILATION

OBJ	LINE	SOURCE STATEMENT
	1	#PAGEWIDTH(80)
	2	*****
	3	;
	4	;
	5	SPEECH SYNTHESIS SYSTEM
	6	HAMMING WINDOW
	7	FILE NAME : WINDOW.SRC DATE
	8	MODULE DEF : Hamming Window
	9	AUTHOR : Onder Bicioglu 22 06 85
	10	*****
	11	*****
	12	;
	13	NAME WINDOW
	14	;
	15	PUBLIC W
	16	;
	17	CSEG
	18	;
00 7B14	19	W: DW 5243, 0, 5258, 0, 5303, 0, 5377, 0
02 0000		
04 8A14		
06 0000		
08 B714		
0A 0000		
0C 0115		
0E 0000		
10 6915	20	DW 5481, 0, 5614, 0, 5777, 0, 5969, 0
12 0000		
14 EE15		
16 0000		
18 9116		
1A 0000		
1C 5117		
1E 0000		
20 2E18	21	DW 6190, 0, 6440, 0, 6718, 0, 7025, 0
22 0000		
24 2819		
26 0000		
28 3E1A		
2A 0000		
2C 711B		
2E 0000		
30 C01C	22	DW 7360, 0, 7722, 0, 8112, 0, 8529, 0
32 0000		
34 2A1E		
36 0000		
38 B01F		
3A 0000		
3C 5121		
3E 0000		
40 0C23	23	DW 8972, 0, 9441, 0, 9936, 0, 10456, 0
42 0000		
44 E124		
46 0000		

LDC	OBJ	LINE	SOURCE STATEMENT
004B	D026		
004A	0000		
004C	D828		
004E	0000		
0050	F82A	24	DW 11000, 0, 11568, 0, 12161, 0, 12776, 0
0052	0000		
0054	302D		
0056	0000		
0058	B12F		
005A	0000		
005C	E831		
005E	0000		
0060	6534	25	DW 13413, 0, 14073, 0, 14753, 0, 15453, 0
0062	0000		
0064	F936		
0066	0000		
0068	A139		
006A	0000		
006C	5D3C		
006E	0000		
0070	2D3F	26	DW 16173, 0, 16912, 0, 16670, 0, 18444, 0
0072	0000		
0074	1042		
0076	0000		
0078	1E41		
007A	0000		
007C	0C48		
007E	0000		
0080	244B	27 W8:	DW 19236, 0, 20043, 0, 20892, 0, 21703, 0
0082	0000		
0084	4B4E		
0086	0000		
0088	9C51		
008A	0000		
008C	C754		
008E	0000		
0090	1A5B	28	DW 22554, 0, 23417, 0, 24291, 0, 25178, 0
0092	0000		
0094	795B		
0096	0000		
0098	E35E		
009A	0000		
009C	5A62		
009E	0000		
00A0	D965	29	DW 26073, 0, 26979, 0, 27892, 0, 28813, 0
00A2	0000		
00A4	6369		
00A6	0000		
00A8	F46C		
00AA	0000		
00AC	8D70		
00AE	0000		
00B0	2F74	30	DW 29740, 0, 30673, 0, 31611, 0, 32552, 0
00B2	0000		
00B4	D177		

LOC	OBJ	LINE	SOURCE STATEMENT
00B6	0000		
00B8	7E7B		
00BA	0000		
00BC	287F		
00BE	0000		
00C0	D882	31	DW 33496, 0, 34442, 0, 35389, 0, 36336, 0
00C2	0000		
00C4	8AB6		
00C6	0000		
00C8	3DBA		
00CA	0000		
00CC	F08D		
00CE	0000		
00D0	A291	32	DW 37282, 0, 38226, 0, 39168, 0, 40105, 0
00D2	0000		
00D4	5295		
00D6	0000		
00D8	0099		
00DA	0000		
00DC	A99C		
00DE	0000		
00E0	4EAD	33	DW 41038, 0, 41965, 0, 42887, 0, 43800, 0
00E2	0000		
00E4	EDA3		
00E6	0000		
00E8	87A7		
00EA	0000		
00EC	1BAB		
00EE	0000		
00F0	A1AE	34	DW 44705, 0, 45601, 0, 46487, 0, 47362, 0
00F2	0000		
00F4	21B2		
00F6	0000		
00F8	97B5		
00FA	0000		
00FC	02B9		
00FE	0000		
0100	61FD	35 W16:	DW 48225, 0, 49076, 0, 49912, 0, 50735, 0
0102	0000		
0104	B4BF		
0106	0000		
0108	F8C2		
010A	0000		
010C	2FCE		
010E	0000		
0110	5AF9	36	DW 51542, 0, 52334, 0, 53109, 0, 53866, 0
0112	0000		
0114	6ECC		
0116	0000		
0118	75CF		
011A	0000		
011C	6AD2		
011E	0000		
0120	4DD5	37	DW 54605, 0, 55325, 0, 56026, 0, 56706, 0
0122	0000		

LOC	OBJ	LINE	SOURCE STATEMENT
0124	1DD8		
0126	0000		
0128	DADP		
012A	0000		
012C	82DD		
012E	0000		
0130	15E0	38	DW 57365, 0, 58003, 0, 58617, 0, 59210, 0
0132	0000		
0134	93E2		
0136	0000		
0138	F9E4		
013A	0000		
013C	4AE7		
013E	0000		
0140	83FA	39	DW 59779, 0, 60323, 0, 60842, 0, 61338, 0
0142	0000		
0144	A3EB		
0146	0000		
0148	AAED		
014A	0000		
014C	9AEF		
014E	0000		
0150	6FF1	40	DW 61807, 0, 62191, 0, 62669, 0, 63057, 0
0152	0000		
0154	EFF2		
0156	0000		
0158	CDF4		
015A	0000		
015C	51F6		
015E	0000		
0160	BBF7	41	DW 63419, 0, 63753, 0, 64061, 0, 64339, 0
0162	0000		
0164	09F9		
0166	0000		
0168	3DFA		
016A	0000		
016C	53FA		
016E	0000		
0170	4DFC	42	DW 64589, 0, 64810, 0, 65001, 0, 65165, 0
0172	0000		
0174	2AFD		
0176	0000		
0178	E9FD		
017A	0000		
017C	8DFE		
017E	0000		
0180	12FF	43	DW 65298, 0, 65402, 0, 65476, 0, 65521, 0
0182	0000		
0184	7AFF		
0186	0000		
0188	C4FF		
018A	0000		
018C	F1FF		
018E	0000		

LOC	OBJ	LINE	SOURCE STATEMENT
-----	-----	------	------------------

		45	END
--	--	----	-----

PUBLIC SYMBOLS

C 0000

INTERNAL SYMBOLS

USER SYMBOLS

C 0000

W16

C 0100

W8

C 0080

ASSEMBLY COMPLETE, NO ERRORS

```

OC OBJ          LINE      SOURCE STATEMENT
1  $PAGEWIDTH(80)
2  ;*****
3  ;*
4  ;*          SPEECH SYNTHESIS SYSTEM
5  ;*          32-BIT LIBRARY DATA
6  ;*
7  ;*  FILE NAME      : ALU001.SRC          DATE
8  ;*  MODULE DEF    : Data definitions    -----
9  ;*  AUTHOR       : Onder Bicioglu      22 06 85
10 ;*
11 ;*****
12 ;
13          NAME      FBDATA
14 ;
15          PUBLIC  OP1,OP1END,OP2,OP2END
16          PUBLIC  OP3,OP3END,OP4,OP4END
17          PUBLIC  SIGN,INDEX,I,J
18          PUBLIC  ACOUNT,BCOUNT,LINDEX
19          PUBLIC  RESULT
20 ;
21          DSEG
22 ;*****
23 ;
24 OP1:      DS      3
25 OP1END:   DS      1
26 OP2:      DS      3
27 OP2END:   DS      1
28 OP3:      DS      3
29 OP3END:   DS      1
30 OP4:      DS      3
31 OP4END:   DS      1
32 SIGN:     DS      1
33 INDEX:    DS      1
34 ACOUNT:  DS      1
35 BCOUNT:  DS      1
36 I:        DS      1
37 J:        DS      1
38 RESULT:   DW      1
39 LINDEX:   DS      1
40 SIGNAL:   DS      1
41 ;
42 ;*****
43 ;
44          END

```

PUBLIC SYMBOLS

ACOUNT D 0012	BCOUNT D 0013	I	D 0014	INDEX D 0011
D 0015	LINDEX D 0018	OP1	D 0000	OP1END D 0003
D 0004	OP2END D 0007	OP3	D 0008	OP3END D 000B
D 000C	OP4END D 000F	RESULT	D 0016	SIGN D 0010

INTERNAL SYMBOLS

R SYMBOLS

UNT D 0012	BCOUNT D 0013	I D 0014	INDEX D 0011
D 0015	LINDEX D 0018	OP1 D 0000	OP1END D 0003
D 0004	OP2END D 0007	OP3 D 0008	OP3END D 000B
D 000C	OP4END D 000F	RESULT D 0016	SIGN D 0010
NAL D 0019			

SEMBLY COMPLETE, NO ERRORS

OBJ

LINE

SOURCE STATEMENT

```
1 $PAGEWIDTH(80)
2 ;*****
3 ;*
4 ;*          SPEECH SYNTHESIS SYSTEM
5 ;*          ADDITION, SUBTRACTION, MULTIPLICATION
6 ;*
7 ;* FILE NAME      : ALU002.SRC          DATE
8 ;* MODULE DEF     : Four byte arith.   -----
9 ;* AUTHOR         : Onder Bicioglu     22 06 85
10 ;*
11 ;*****
12 ;
13          NAME      FBARIT
14 ;
15          EXTRN      OP1,OP1END,OP2,OP2END
16          EXTRN      OP3,OP3END,OP4,OP4END
17          EXTRN      SIGN,INDEX,I,J
18          EXTRN      LINDEX,RESULT
19 ;
20          EXTRN      BY2,TIMES2
21 ;
22          PUBLIC     FBADD,FBSUB,FBMUL
23 ;
24          CSEG
25 ;
26 ;*****
27 ; PROCEDURE : FBADD
28 ; FUNCTION  : Adds two four byte operands pointed by
29 ;             the first two parameters and placed to
30 ;             result to the memory pointed by the last
31 ;             operand
32 ; PLM CALL   : CALL FBADD(OPERAND1,OPERAND2,RESULT
33 ; INPUT      : OPERAND1....address of the first byte
34 ;             of first operand
35 ;             OPERAND2....address of the first byte
36 ;             of the second operand
37 ;             RESULT.....address of the first byte
38 ;             of memory to store the result
39 ; OUTPUT     : RESULT=result
40 ; GLOBALS    : none
41 ; CALLS      : none
42 ; DESTROYS   : all
43 ;
44 FBADD:      POP      H
45             XTHL
46 ;
47             LDAX     B
48             ADD      M
49             STAX     D
50 ;
51             INX      H
52             INX      D
53             INX      B
54             LDAX     B
```

OBJ	LINE	SOURCE STATEMENT
9 8E	55	ADC M
4 12	56	STAX D
	57 ;	
3 23	58	INX H
C 13	59	INX D
0 03	60	INX B
E 0A	61	LDAX B
F 8E	62	ADC M
0 12	63	STAX D
	64 ;	
1 23	65	INX H
2 13	66	INX D
3 03	67	INX B
4 0A	68	LDAX B
5 8E	69	ADC M
6 12	70	STAX D
	71 ;	
7 C9	72	RET
	73 ;	
	74 ;	*****
	75 ;	PROCEDURE : FBSUB
	76 ;	FUNCTION : Subtracts the four byte operand pointed
	77 ;	by the first parameter from the second
	78 ;	four byte parameter and puts the result
	79 ;	to the memory location pointed by third
	80 ;	parameter
	81 ;	PLM CALL : CALL FBSUB(OPERAND1, OPERAND2, RESULT
	82 ;	INPUT : OPERAND1....address of the first byte
	83 ;	of first operand
	84 ;	OPERAND2....address of the first byte
	85 ;	of the second operand
	86 ;	RESULT.....address of the first byte
	87 ;	of memory to store the result
	88 ;	OUTPUT : RESULT=result
	89 ;	GLOBALS : none
	90 ;	CALLS : none
	91 ;	DESTROYS : all
	92 ;	
8 E1	93	FBSUB: POP H
9 E3	94	XTHL
	95 ;	
A 0A	96	LDAX B
B 96	97	SUB M
C 12	98	STAX D
	99 ;	
D 23	100	INX H
E 13	101	INX D
F 03	102	INX B
0 0A	103	LDAX B
1 9E	104	SBB M
2 12	105	STAX D
	106 ;	
3 23	107	INX H
4 13	108	INX D
5 03	109	INX B

OBJ	LINE	SOURCE STATEMENT
0A	110	LDAX B
9E	111	SBB M
12	112	STAX D
	113 ;	
23	114	INX H
13	115	INX D
03	116	INX B
0A	117	LDAX B
9E	118	SBB M
12	119	STAX D
	120 ;	
C9	121	RET
	122 ;	
	123 ;	*****
	124 ;	PROCEDURE : FBMUL
	125 ;	FUNCTION : Multiplies two four byte operands and
	126 ;	places the result to the memory
	127 ;	location pointed by the third parameter
	128 ;	PLM CALL : CALL SBMUL(OPERAND1, OPERAND2, RESULT);
	129 ;	INPUT : OPERAND1.....First multiplier
	130 ;	OPERAND2.....Second multiplier
	131 ;	RESULT.....points to the memory to
	132 ;	store the result
	133 ;	OUTPUT : RESULT=result
	134 ;	GLOBALS : OP1, OP2, OP1END, OP2END, INDEX, SIGN
	135 ;	CALLS : FBADD, BY2
	136 ;	DESTROYS : all
	137 ;	
0 210000 E	138	FBMUL: LXI H, OP1
3 0A	139	LDAX B
4 77	140	MOV M, A
5 23	141	INX H
5 03	142	INX B
7 0A	143	LDAX B
8 77	144	MOV M, A
9 23	145	INX H
A 03	146	INX B
B 0A	147	LDAX B
C 77	148	MOV M, A
D 23	149	INX H
E 03	150	INX B
F 0A	151	LDAX B
0 77	152	MOV M, A ;OP1 loaded
	153	;
1 E1	154	POP H
2 E3	155	XTHL
3 010000 E	156	LXI B, OP2
6 7E	157	MOV A, M
7 02	158	STAX B
8 23	159	INX H
9 03	160	INX B
A 7E	161	MOV A, M
B 02	162	STAX B
C 23	163	INX H
D 03	164	INX B

C	OBJ	LINE	SOURCE STATEMENT
4E	7E	165	MOV A,M
4F	02	166	STAX B
50	23	167	INX H
51	03	168	INX B
52	7E	169	MOV A,M
53	02	170	STAX B
		171	
54	EB	172	XCHG ;OP2 loaded
55	220000	173	E SHLD RESULT
58	AF	174	XRA A
59	77	175	MOV M,A
5A	23	176	INX H
5B	77	177	MOV M,A
5C	23	178	INX H
5D	77	179	MOV M,A
5E	23	180	INX H
5F	77	181	MOV M,A
		182	
60	320000	183	E STA SIGN ;RESULT zeroed
		184	
			;SIGN = +
63	3E1F	185	MVI A,31
65	320000	186	E STA LINDEX
		187	
			;LINDEX = 31
68	3E11	188	MVI A,17
6A	320000	189	E STA INDEX
		190	
			;INDEX = 17
6D	210000	191	E LXI H,OP1END
70	CDEB00	192	C CALL SGNCHK
		193	
			;conversion if necessary
73	3A0000	194	E GOON: LDA OP1
76	E601	195	ANI 1
78	C28C00	196	C JNZ GOON1
		197	
			;
7B	210000	198	E LXI H,OP1END
7E	CD0000	199	E CALL BY2
		200	
			;OP1 = OP1 / 2
81	210000	201	E LXI H,LINDEX
84	35	202	DCR M
85	210000	203	E LXI H,INDEX
88	35	204	DCR M
89	C27300	205	C JNZ GOON
		206	
			;
8C	210000	207	E GOON1: LXI H,OP2END
8F	CDEB00	208	C CALL SGNCHK
		209	
			;conversion if necessary
		210	;
92	3A0000	211	E DOMULT: LDA INDEX
95	B7	212	ORA A
96	C9A900	213	C JZ SIFT02
99	3D	214	DCR A
9A	320000	215	E STA INDEX
		216	
			;If INDEX<>0 then
		217	;decrement INDEX
9D	2A0000	218	E LHLD RESULT
9A0	23	219	INX H

OC	OBJ	LINE	SOURCE STATEMENT
0A1	23	220	INX H
0A2	23	221	INX H
0A3	CD0000	222	CALL BY2
		223	
0A6	D3AF00	224	JMP SIFT01
		225	
0A9	210000	226	SIFT02: LXI H, OP2
0AC	CD0000	227	CALL TIMES2
		228	
0AF	210000	229	SIFT01: LXI H, OP1END
0B2	CD0000	230	CALL BY2
		231	
0B5	D2C300	232	JNC NOADD
0B8	2A0000	233	LHLD RESULT
0BB	E5	234	PUSH H
0BC	EB	235	XCHG
0BD	010000	236	LXI B, OP2
0C0	CD0000	237	CALL FBADD
		238	
0C3	210000	239	NOADD: LXI H, LINDE
0C6	35	240	DCR M
0C7	D29200	241	JNZ DOMULT
		242	
0CA	3A0000	243	LDA SIGN
0CD	B7	244	ORA A
0CE	CB	245	RZ
		246	
0CF	2A0000	247	LHLD RESULT
0D2	AF	248	XRA A
0D3	7E	249	MOV A, M
0D4	D601	250	SUI 1
0D6	2F	251	CMA
0D7	77	252	MOV M, A
0D8	23	253	INX H
0D9	7E	254	MOV A, M
0DA	DE00	255	SBI 0
0DC	2F	256	CMA
0DD	77	257	MOV M, A
0DE	23	258	INX H
0DF	7E	259	MOV A, M
0E0	DE00	260	SBI 0
0E2	2F	261	CMA
0E3	77	262	MOV M, A
0E4	23	263	INX H
0E5	7E	264	MOV A, M
0E6	DE00	265	SBI 0
0E8	2F	266	CMA
0E9	77	267	MOV M, A
0EA	C9	268	EXIT: RET
		269	
		270	*****
		271	
0EB	7E	272	SGNCHK: MOV A, M
0EC	B7	273	ORA A
0ED	F0	274	RP

LOC	OBJ	LINE	SOURCE STATEMENT
		275 ;	
00EE	2B	276	DCX H
00EF	2B	277	DCX H
00F0	2B	278	DCX H
		279 ;	
00F1	7E	280	MOV A, M
00F2	2F	281	CMA
00F3	DE01	282	ADI 1
00F5	77	283	MOV M, A
		284 ;	
00F6	23	285	INX H
00F7	7E	286	MOV A, M
00F8	2F	287	CMA
00F9	DE00	288	ACI 0
00FB	77	289	MOV M, A
		290 ;	
00FC	23	291	INX H
00FD	7E	292	MOV A, M
00FE	2F	293	CMA
00FF	DE00	294	ACI 0
0101	77	295	MOV M, A
		296 ;	
0102	23	297	INX H
0103	7E	298	MOV A, M
0104	2F	299	CMA
0105	DE00	300	ACI 0
0107	77	301	MOV M, A
		302 ;	
0108	3A0000	E 303	LDA SIGN
010B	2F	304	CMA
010C	320000	E 305	STA SIGN
010F	C9	306	RET
		307 ;	
		308 ;	*****
		309 ;	
		310	END

BLIC SYMBOLS

ADD	C 0000	FBMUL	C 0030	FBSUB	C 0018
-----	--------	-------	--------	-------	--------

TERNAL SYMBOLS

2	E 0000	I	E 0000	INDEX	E 0000	J	E 0000
NDEX	E 0000	OP1	E 0000	OP1END	E 0000	OP2	E 0000
2END	E 0000	OP3	E 0000	OP3END	E 0000	OP4	E 0000
4END	E 0000	RESULT	E 0000	SIGN	E 0000	TIMES2	E 0000

ER SYMBOLS

2	E 0000	DDMULT	C 0092	EXIT	C 00EA	FBADD	C 0000
MUL	C 0030	FBSUB	C 0018	GOON	C 0073	GOON1	C 008C
	E 0000	INDEX	E 0000	J	E 0000	LINDEX	E 0000
ADD	C 00C3	OP1	E 0000	OP1END	E 0000	OP2	E 0000
2END	E 0000	OP3	E 0000	OP3END	E 0000	OP4	E 0000
4END	E 0000	RESULT	E 0000	SGNCHK	C 00EB	SIFT01	C 00AF
FT02	C 00A9	SIGN	E 0000	TIMES2	E 0000		

OBJ

LINE

SOURCE STATEMENT

```

1  $PAGEWIDTH(80)
2  ;*****
3  ;*
4  ;*          SPEECH SYNTHESIS SYSTEM
5  ;*          DIVISION
6  ;*
7  ;*  FILE NAME      : ALU003.SRC          DATE
8  ;*  MODULE DEF     : Four byte arith.   -----
9  ;*  AUTHOR         : Onder Bicioglu     22 06 85
10 ;*
11 ;*****
12 ;
13      NAME      DIVIDE
14 ;
15      PUBLIC    FBDIV,BY2,TIMES2
16      PUBLIC    LOOP1,LOOP,CHKSGN
17 ;
18      EXTRN     ACOUNT,BCOUNT,INDEX,SIGN
19      EXTRN     I,J,RESULT
20      EXTRN     OP1,OP1END,OP2,OP2END
21      EXTRN     OP3,OP3END,OP4,OP4END
22      EXTRN     FBSUB
23 ;
24      CSEG
25 ;
26 ;*****
27 ;  PROCEDURE : FBDIV
28 ;  FUNCTION  : Divides first operand by the second one
29 ;              places the result to the memory
30 ;              location pointed by the third parameter
31 ;  PLM CALL  : CALL FBDIV(OPERAND1,OPERAND2,RESULT);
32 ;  INPUT     : OPERAND1.....dividend
33 ;              OPERAND2.....divisor
34 ;              RESULT.....result
35 ;  OUTPUT    : RESULT=result
36 ;  GLOBALS   : OP1,OP2,OP1END,OP2END,INDEX,SIGN
37 ;  CALLS     : FBADD
38 ;  DESTROYS  : all
39 ;
40 FBDIV:  XCHG
EB 220000  E 41      SHLD      RESULT
42 ;
AF 43      XRA      A
320000  E 44      STA      SIGN      ;sign=+
320000  E 45      STA      BCOUNT    ;BCOUNT=0
320000  E 46      STA      I        ;I=0
320000  E 47      STA      ACOUNT  ;ACOUNT=0
48 ;
210000  E 49      LXI      H,OP2
AF 50      XRA      A
OA 51      LDAX     B
2F 52      CMA
C601 53      ADI      1
77 54      MOV      M,A

```

LOC	OBJ	LINE	SOURCE STATEMENT
		55 ;	
001A	03	56	INX B
001B	23	57	INX H
001C	0A	58	LDAX B
001D	2F	59	CMA
001E	DE00	60	ACI 0
0020	77	61	MOV M, A
		62 ;	
0021	03	63	INX B
0022	23	64	INX H
0023	0A	65	LDAX B
0024	2F	66	CMA
0025	DE00	67	ACI 0
0027	77	68	MOV M, A
		69 ;	
0028	03	70	INX B
0029	23	71	INX H
002A	0A	72	LDAX B
002B	B7	73	DRA A
002C	F23B00	74	JP PLUS
		75 ;	
002F	2F	76	CMA
0030	DE00	77	ACI 0
0032	77	78	MOV M, A
0033	3EFF	79	MVI A, OFFH
0035	320000	80	STA SIGN
003B	D34A00	81	JMP GOON
		82 ;	
003B	77	83 PLUS:	MOV M, A
		84 ;	
003C	2B	85	DCX H
003D	0B	86	DCX B
003E	0A	87	LDAX B
003F	77	88	MOV M, A
		89 ;	
0040	2B	90	DCX H
0041	0B	91	DCX B
0042	0A	92	LDAX B
0043	77	93	MOV M, A
		94 ;	
0044	2B	95	DCX H
0045	0B	96	DCX B
0046	0A	97	LDAX B
0047	77	98	MOV M, A
		99 ;	
0048	E1	100 GOON:	POP H
0049	E3	101	XTHL
		102 ;	
004A	010000	103	LXI B, OP1
004D	AF	104	XRA A
		105 ;	
004E	7E	106	MOV A, M
004F	2F	107	CMA
0050	C601	108	ADI 1
0052	02	109	STAX B

;get first parameter

CC	OBJ	LINE	SOURCE STATEMENT
		110 ;	
053	03	111	INX B
054	23	112	INX H
055	7E	113	MOV A, M
056	2F	114	CMA
057	CE00	115	ACI 0
059	02	116	STAX B
		117 ;	
05A	03	118	INX B
05B	23	119	INX H
05C	7E	120	MOV A, M
05D	2F	121	CMA
05E	CE00	122	ACI 0
060	02	123	STAX B
		124 ;	
061	03	125	INX B
062	23	126	INX H
063	7E	127	MOV A, M
064	B7	128	DRA A
065	F27600	129	JP PLUS1
		130 ;	
068	2F	131	CMA
069	CE00	132	ACI 0
06B	02	133	STAX B
06C	3A0000	134	LDA SIGN
06F	2F	135	CMA
070	320000	136	STA SIGN
073	C3B300	137	JMP DODIV
		138 ;	
076	02	139	PLUS1: STAX B
077	2B	140	DCX H
078	0B	141	DCX B
079	7E	142	MOV A, M
07A	02	143	STAX B
07B	2B	144	DCX H
07C	0B	145	DCX B
07D	7E	146	MOV A, M
07E	02	147	STAX B
07F	2B	148	DCX H
080	0B	149	DCX B
081	7E	150	MOV A, M
082	02	151	STAX B
		152 ;	
083	2A0000	153	DODIV: LHLD RESULT
086	AF	154	XRA A
087	77	155	MOV M, A
088	23	156	INX H
089	77	157	MOV M, A
08A	23	158	INX H
08B	77	159	MOV M, A
08C	23	160	INX H
08D	77	161	MOV M, A
		162	;RESULT is made zero
08E	3A0000	163	LDA OPIEND
091	E640	164	ANI 40H

BC	OBJ	LINE	SOURCE STATEMENT
093	C2A900	C 165	JNZ OP10K
		166 ;	
096	210000	E 167	LXI H, DP1
099	CDA901	C 168	CALL TIMES2
09C	210000	E 169	LXI H, ACOUNT
09F	34	170	INR M
0A0	3E1F	171	MVI A, 31
0A2	BE	172	CMP M
0A3	CA9701	C 173	JZ EXIT
		174 ;	
0A6	C38300	C 175	JMP DODIV
		176 ;	
0A9	3A0000	E 177	OP10K: LDA OP2END
0AC	E640	178	ANI 40H
0AE	C2BE00	C 179	JNZ OP20K
		180 ;	
0B1	210000	E 181	LXI H, DP2
0B4	CDA901	C 182	CALL TIMES2
0B7	210000	E 183	LXI H, BCOUNT
0BA	34	184	INR M
0BB	C3A900	C 185	JMP OP10K
		186 ;	
0BE	210000	E 187	OP20K: LXI H, BCOUNT
0C1	3E10	188	MVI A, 16
0C3	86	189	ADD M
0C4	210000	E 190	LXI H, ACOUNT
0C7	96	191	SUB M
0C8	FE1F	192	CPI 31
0CA	DADF00	C 193	JC LDIDX ;load index
		194 ;	
0CD	2A0000	E 195	LHLD RESULT
0D0	36FF	196	MVI M, OFFH
0D2	23	197	INX H
0D3	36FF	198	MVI M, OFFH
0D5	23	199	INX H
0D6	36FF	200	MVI M, OFFH
0D8	23	201	INX H
0D9	367F	202	MVI M, 07FH
0DB	C7A701	C 203	JMP EXIT
		204 ;	
0DE	320000	E 205	LDIDX: STA INDEX
		206 ;	
0E1	210000	E 207	LOOP: LXI H, DP2
0E4	ES	208	PUSH H
0E5	010000	E 209	LXI B, DP1
0E8	110000	E 210	LXI D, DP4
0EB	CD0000	E 211	CALL FBSUB
		212 ;	
0EE	3A0000	E 213	LDA OP4END
0F1	E680	214	ANI 80H
0F3	C25B01	C 215	JNZ ALTB
		216	;A Less Than B
0FE	2A0000	E 217	LHLD RESULT
0F9	7E	218	MOV A, M
0FA	FE01	219	ORI 1

LOC	OBJ		LINE	SOURCE STATEMENT
00FC	77		220	MOV M, A
			221	
00FD	210000	E	222	LXI H, OP4 ; RESULT=RESULT+1
0100	CDA901	C	223	CALL TIMES2
			224	
0103	210000	E	225	LXI H, OP4 ; OP4=(A-B)*2
0106	110000	E	226	LXI D, OP1
0109	7E		227	MOV A, M
010A	12		228	STAX D
010B	13		229	INX D
010C	23		230	INX H
010D	7E		231	MOV A, M
010E	12		232	STAX D
010F	23		233	INX H
0110	13		234	INX D
0111	7E		235	MOV A, M
0112	12		236	STAX D
0113	13		237	INX D
0114	23		238	INX H
0115	7E		239	MOV A, M
0116	12		240	STAX D
			241	
			242	; A=OP4
0117	3A0000	E	243	LDA OP2END
011A	E640		244	ANI 40H
011C	C22801	C	245	JNZ RSLTX2
			246	
011F	210000	E	247	LXI H, OP2
0122	CDA901	C	248	CALL TIMES2
			249	; B=2*B
0125	D33901	C	250	JMP LOOP1
			251	
0128	210000	E	252	RSLTX2: LXI H, I
012B	3A0000	E	253	LDA INDEX
012E	BE		254	CMP M
012F	CA7501	C	255	JZ CHKSGN
			256	; if I=INDEX then exit
0132	34		257	INR M
			258	; I=I+1
0133	2A0000	E	259	LHLD RESULT
0136	CDA901	C	260	CALL TIMES2
			261	; RESULT=RESULT*2
0139	3A0000	E	262	LOOP1: LDA OP1END
013C	E640		263	ANI 40H
013E	C2E100	C	264	JNZ LOOP
			265	
0141	210000	E	266	LXI H, I
0144	3A0000	E	267	LDA INDEX
0147	BE		268	CMP M
0148	CA7501	C	269	JZ CHKSGN
			270	; if I=INDEX then exit
014B	34		271	INR M
			272	; I=I+1
014C	210000	E	273	LXI H, OP1
014F	CDA901	C	274	CALL TIMES2

LOC	OBJ	LINE	SOURCE STATEMENT
		275	
0152	2A0000	E 276	LHLD RESULT ;A=2*A
0155	CDA901	C 277	CALL TIMES2
		278	
0158	C33901	C 279	JMP LOOP1 ;RESULT=RESULT*2
		280 ;	
		281 ;	
015B	210000	E 282	ALTB: LXI H, I
015E	3A0000	E 283	LDA INDEX
0161	BE	284	CMP M
0162	CA7501	C 285	JZ CHKSGN
		286	
0165	34	287	INR M ;if I=INDEX then exit
		288	
			;I=I+1
0166	2A0000	E 289	LHLD RESULT
0169	CDA901	C 290	CALL TIMES2
		291	
			;RESULT=RESULT*2
016C	210000	E 292	LXI H, OP2END
016F	CD9801	C 293	CALL BY2
		294	
			;B=B/2
		295 ;	
0172	C3E100	C 296	JMP LOOP
		297 ;	
0175	3A0000	E 298	CHKSGN: LDA SIGN
0178	B7	299	DRA A
0179	CA9701	C 300	JZ EXIT
		301 ;	
017C	AF	302	XRA A
017D	2A0000	E 303	LHLD RESULT
0180	7E	304	MOV A, M
0181	D601	305	SUI 1
0183	2F	306	CMA
0184	77	307	MOV M, A
0185	23	308	INX H
0186	7E	309	MOV A, M
0187	DE00	310	SBI 0
0189	2F	311	CMA
018A	77	312	MOV M, A
018B	23	313	INX H
018C	7E	314	MOV A, M
018D	DE00	315	SBI 0
018F	2F	316	CMA
0190	77	317	MOV M, A
0191	23	318	INX H
0192	7E	319	MOV A, M
0193	DE00	320	SBI 0
0195	2F	321	CMA
0196	77	322	MOV M, A
0197	C9	323	EXIT: RET
		324 ;	
		325 ;*****	
		326 ;	
0198	AF	327	BY2: XRA A
0199	7E	328	MOV A, M
019A	1F	329	RAR

LOC	OBJ	LINE	SOURCE STATEMENT
019B	77	330	MOV M, A
019C	2B	331	DCX H
019D	7E	332	MOV A, M
019E	1F	333	RAR
019F	77	334	MOV M, A
01A0	2B	335	DCX H
01A1	7E	336	MOV A, M
01A2	1F	337	RAR
01A3	77	338	MOV M, A
01A4	2B	339	DCX H
01A5	7E	340	MOV A, M
01A6	1F	341	RAR
01A7	77	342	MOV M, A
01A8	C9	343	RET
		344	:
		345	*****
		346	:
01A9	AF	347	TIMES2: XRA A
01AA	7E	348	MOV A, M
01AB	17	349	RAL
01AC	77	350	MOV M, A
01AD	23	351	INX H
01AE	7E	352	MOV A, M
01AF	17	353	RAL
01B0	77	354	MOV M, A
01B1	23	355	INX H
01B2	7E	356	MOV A, M
01B3	17	357	RAL
01B4	77	358	MOV M, A
01B5	23	359	INX H
01B6	7E	360	MOV A, M
01B7	17	361	RAL
01B8	77	362	MOV M, A
01B9	C9	363	RET
		364	:
		365	*****
		366	:
		367	END

PUBLIC SYMBOLS

BY2	C 0198	CHKSGN	C 0175	FBDIV	C 0000	LOOP	C 00E1
LOOP1	C 0139	TIMES2	C 01A9				

EXTERNAL SYMBOLS

ACOUNT	E 0000	BCDUNT	E 0000	FBSUB	E 0000	I	E 0000
INDEX	E 0000	J	E 0000	OP1	E 0000	OP1END	E 0000
OP2	E 0000	OP2END	E 0000	OP3	E 0000	OP3END	E 0000
OP4	E 0000	OP4END	E 0000	RESULT	E 0000	SIGN	E 0000

USER SYMBOLS

ACOUNT	E 0000	ALTB	C 015B	BCOUNT	E 0000	BY2	C 0198
CHKSGN	C 0175	DODIV	C 0083	EXIT	C 0197	FBDIV	C 0000
FBSUB	E 0000	GOON	C 0048	I	E 0000	INDEX	E 0000
	E 0000	INDIX	C 00DE	LOOP	C 00E1	LOOP1	C 0139

E 0000	OP1END E 0000	OP1OK C 00A9	OP2 E 0000
END E 0000	OP2OK C 00BE	OP3 E 0000	OP3END E 0000
E 0000	OP4END E 0000	PLUS C 003B	PLUS1 C 0076
ULT E 0000	RSLTX2 C 0128	SIGN E 0000	TIMES2 C 01A9

SEMBLY COMPLETE, NO ERRORS

PARAMETERS FOR SOUND " U "

ame	RMS	R	P	k1	k2	k3	k4	k5	k6	k7	k8	k9	k10
01	00												
02	00												
03	01	00	00	00	14	0E	08						
04	01	00	00	05	18	0D	04						
05	0C	00	15	0A	12	0A	04	07	08	09	03	05	05
06	0C	00	17	0A	14	08	06	06	08	0A	04	05	04
07	0A	00	17	0A	15	0B	05	08	08	0A	04	03	03
08	0B	00	17	0A	14	0A	05	08	08	0A	04	05	04
09	0B	00	18	0A	16	0B	05	09	07	09	04	04	04
0A	0B	00	18	0A	15	0C	06	07	09	07	04	05	04
0B	0C	00	18	0A	15	0B	06	08	07	08	04	05	04
0C	0B	00	18	09	15	0B	06	08	07	07	04	05	05
0D	0B	00	17	09	14	0C	06	06	08	08	03	05	04
0E	0B	00	17	07	15	0E	07	07	05	06	05	05	02
0F	0B	00	15	07	11	0D	08	06	07	0A	05	04	03
10	09	00	14	07	10	0D	06	08	07	09	04	06	06
11	06	00	14	05	12	0D	0A	08	07	08	04	05	05
12	02	00	12	06	0D	0E	0B	0A	08	07	03	03	03
13	03	00	00	01	1F	0D	09						
14	01	00	00	01	1F	0E	09						
15	01	00	00	01	1F	0E	08						
16	01	00	00	00	1F	0F	0B						

PARAMETERS FOR SOUND " DORT "

Frame	RMS	R	P	k1	k2	k3	k4	k5	k6	k7	k8	k9	k10
01	00												
02	00												
03	00												
04	00												
05	00												
06	00												
07	00												
08	00												
09	0D	00	1A	17	0A	06	07	05	09	06	05	03	04
0A	0E	00	1B	11	0F	08	08	07	0A	08	05	04	06
0B	0E	00	1B	12	10	07	08	08	0B	0A	05	04	05
0C	0E	00	1B	13	0D	09	08	07	0A	09	05	04	06
0D	0E	00	1B	13	10	09	07	06	0A	09	06	04	05
0E	0E	00	1B	13	10	09	08	07	0A	09	06	05	05
0F	0E	00	19	14	11	09	09	04	0B	09	06	04	05
10	0E	00	18	12	10	08	07	06	0B	07	05	05	06
11	0B	01	17										
12	06	01	16										
13	05	00	14	14	07	07	0C	07	0A	07	06	05	04
14	02	00	00	0C	0E	0B	0B						
15	00												
16	00												
17	00												
18	04	00	00	13	0B	04	0B						
19	01	00	00	12	07	03	09						
1A	01	00	00	12	07	03	09						
1B	01	00	00	12	07	03	09						
1C	01	00	00	12	07	03	09						
1D	01	00	00	12	07	03	09						

PARAMETERS FOR SOUND " BES "

Frame	RMS	R	P	k1	k2	k3	k4	k5	k6	k7	k8	k9	k10
01	00												
02	00												
03	00												
04	00												
05	00												
06	00												
07	00												
08	00												
09	00												
0A	00												
0B	0E	00	1B	13	0C	06	0A	07	0A	09	04	04	06
0C	0E	00	1C	15	0B	05	0A	06	0C	09	04	05	06
0D	0E	00	1D	15	0B	05	0C	07	0B	07	04	05	06
0E	0E	00	1C	17	0B	04	0B	06	0C	07	04	05	05
0F	0E	00	1C	15	0A	05	0B	07	0B	07	04	05	05
10	0E	00	00	0B	14	0B	0C						
11	0E	00	1B	15	09	04	09	0A	0B	07	04	03	05
12	0B	00	15	16	0B	02	0B	0C	0D	07	04	01	04
13	0B	00	00	14	0B	03	0B						
14	0D	00	1B	1A	1B	09	0B	04	0B	06	03	04	06
15	0D	00	00	1B	0E	03	06						
16	0B	00	00	1B	0D	03	06						
17	0B	00	00	19	13	04	05						
18	0A	00	00	19	15	04	05						
19	0A	00	00	1B	12	04	06						
1A	0A	00	00	19	13	03	06						
1B	06	00	00	19	11	02	06						
1C	06	00	00	19	11	02	06						
1D	06	00	00	19	11	02	06						
1E	06	00	00	19	11	02	06						
1F	06	00	00	19	11	02	06						

name	RMS	R	P	k1	k2	k3	k4	k5	k6	k7	k8	k9	k10
01	00												
02	00												
03	00												
04	00												
05	01	00	00	09	02	0C	0B						
06	04	00	00	11	14	0A	07						
07	0E	00	1B	14	11	0C	0B	07	06	07	04	04	06
08	0E	00	1C	14	0E	0B	0C	06	09	05	04	04	05
09	0E	00	1C	14	0D	0A	0B	08	0A	04	05	04	04
0A	0C	00	1C	12	10	0A	09	07	0B	07	03	04	04
0B	06	00	1D	0F	0F	0C	0B	05	09	09	03	04	04
0C	02	00	00	03	13	0C	06						
0D	02	00	00	07	11	0A	0A						
0E	00												
0F	01	00	00	04	0A	0E	0A						
10	0B	00	00	12	0C	07	0B						
11	02	00	14	19	0B	0A	0B	06	05	06	04	04	06
12	0B	00	15	0F	10	07	07	08	08	07	03	04	04
13	0B	00	1B	11	0B	07	0C	08	09	06	04	04	04
14	09	00	1B	11	0A	07	0B	08	09	07	05	05	04
15	06	00	1C	0D	0D	09	0C	08	08	08	04	03	03
16	06	00	1C	0E	0C	09	0B	09	08	09	03	05	03
17	05	00	1C	0B	0D	0B	0B	09	08	08	05	04	03
18	05	00	1C	0B	0F	0B	09	08	06	09	04	05	03
19	09	00	1B	11	10	09	06	08	08	09	04	04	04
1A	0B	00	1B	12	0F	07	09	06	08	08	04	05	03
1B	0B	00	19	13	11	09	09	04	09	08	04	03	03
1C	0A	00	1B	11	10	0B	06	08	08	0A	05	03	04
1D	0B	00	15	10	0C	07	0A	0B	08	08	04	04	03
1E	0B	00	15	10	0B	07	0C	0B	09	07	03	04	05
1F	07	00	15	10	0A	06	0C	0C	09	06	03	03	05
20	06	00	15	0C	0B	09	0B	0E	06	0B	03	04	03
21	02	00	00	02	1F	0B	09						
22	01	00	15	0E	07	0C	0A	0D	0A	07	04	04	02
23	0B	00	00	12	15	03	06						
24	03	00	00	07	0F	0A	07						
25	06	00	1B	14	09	0B	07	05	05	0B	05	02	03
26	0B	00	1B	12	0C	0B	0B	06	03	09	04	03	05
27	0A	00	1C	12	12	07	09	07	07	06	05	05	04
28	0A	00	1C	11	12	07	0B	08	08	07	04	05	04
29	05	00	1C	0D	13	0B	06	0B	0B	09	04	05	04
2A	01	00	00	00	11	0F	07						
2B	00												
2C	01	00	00	00	09	0D	09						
2D	05	00	00	10	14	02	0B						
2E	0A	00	00	0A	16	06	09						
2F	0E	00	1C	14	0E	07	07	06	09	0A	06	03	03
30	0B	00	10	13	0B	0B	09	07	0A	07	06	04	04
31	03	00	00	02	0E	0C	0A						
32	01	00	00	01	10	0E	0B						
33	0A	00	1C	16	0B	03	05	0B	0C	09	05	03	05
34	0A	00	1D	13	0B	04	0B	0A	0A	07	03	04	06
35	04	00	1D	12	06	04	0B	0D	0B	07	04	02	02
36	05	00	1E	12	04	07	0C	0C	0A	06	03	02	04
37	04	00	1E	11	0B	06	09	0C	0A	0B	04	02	03
38	05	00	1D	12	09	05	0B	0D	0A	09	03	04	02
39	05	00	1B	12	0A	04	09	0B	0A	06	06	04	02
3A	0E	00	1B	15	09	06	0B	07	0B	0B	06	03	05
3B	0E	00	15	17	0A	05	07	09	0B	0A	06	03	04
3C	0C	00	15	13	07	05	0A	0A	09	05	05	04	04

" ALTINDAN KALKABILMEK ICIN

[illegible]

BIBLIOGRAPHY

1. L.R.Rabiner. R.W.Schafer . Digital Processing of Speech Signals. Prentice Hall Inc. 1978
2. L.R.Rabiner. R.W.Schafer . "Digital Techniques for Computer Voice Response Implementations and Applications." IEEE Proc pp. 416-433 April 1976
3. Slyter R.J.. "Digitization of Speech." Philips Technical Review. Volume 41. No 7/8 . pp. 201-223 1983/84
4. J.A.Kuecken . Talking Computers and Telecommunications. Van Nostrand Reinhold Company 1983.
5. J.D.Markel.A.H.Gray,Jr. Linear Prediction of Speech " Springer Verlag Berlin 1976
6. O.Demircan. Türkiye Türkçesinin Ses Düzeni, Türkiye Türkçesinde Sesler. Turk Dil Kurumu Yayinlari 1979
7. Lin K.S.. Goudie K.M.. Frants G.A.,Brantingham G.L., "Text-to-Speech Using Allophone Stringing," IEEE Trans. on Consumer Electronics Vol CE-27 pp. 144-152 May 1981
8. J.Makhoul . "Linear Prediction: A Tutorial Review," IEEE Proc. Vol.63 No4 pp. 561-580 April 1975
9. J.L.Roux.C.Gueguen. "A Fixed Point Computation of Partial Correlation Coefficients." IEEE Trans. on ASSP pp. 257-259 June 77
10. W.Hess. Pitch Determination of Speech Signals. Springer Verlag Berlin 1983
11. J.A.Feldman. E.M.Hofstetter.M.L.Maloss. "A compact Flexible LPC Vocoder Based on a Commercial Signal Processing Microcomputer." IEEE Trans. on ASSP Vol.31 No 1. pp. 252-258 February 1983
12. K.Dietz. "Tragbares Sprachanalyse-System erzeugt LPC-Parameter." Elektronik pp.63-67. 23/18.11.1983
13. J.Bellamy . Digital Telephony. Wiley 1982
14. TMS 5110A Voice Synthesis Processor Data Manual .Texas Instruments June 1983

15. TMS 5220C Voice Synthesis Processor Data Manual. Texas Instruments Nov. 1983
16. ISIS-II User's Guide. Intel Corporation
17. ISIS-II Credit (CRT-Based Text Editor) User's Guide. Intel Corporation
18. PLM80 Programming Manual. Intel Corporation
19. 8080/8085 Assembly Language Programming Manual. Intel Corporation