15561

RECOGNITION AND TRANSCRIPTION OF ONE PART MELODY WITH COMPUTER

A MASTER'S THESIS

in

Electrical and Electronics Engineering

Middle East Technical University

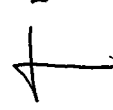
T. C.
Yükseköğretim Kurulu
Dokümantasyon Merkezi

by

Emin Germen

September 1991

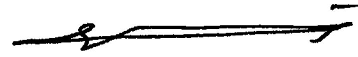
Approval of the Graduate School of Natural and Applied Sciences



Prof. Dr. Alpay ANKARA

Director

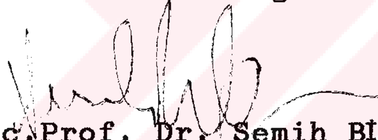
I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science in Electrical and Electronics Engineering.



Prof. Dr. Erol KOCAOĞLAN

Chairman of the Department

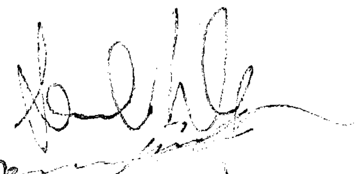
We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science in Electrical and Electronics Engineering.



Assoc.Prof. Dr. Semih BİLGEN — Supervisor

Assoc.Prof. Dr. Melek YÜCEL — Co-supervisor

Examining Comitee in Charge :



Assoc. Prof. Dr. Semih BİLGEN

Assoc. Prof. Dr. Melek YÜCEL

Prof. Dr. Kemal İNAN

Assoc. Prof. İstemihan TAVİLOĞLU

Assoc. Prof. Dr. Güney GÖNENÇ (Chairman)

ABSTRACT

RECOGNITION AND TRANSCRIPTION OF ONE PART MELODY WITH COMPUTER

GERMEN, Emin

M.S. in Electrical and Electronics Eng.
Supervisor : Assoc. Prof. Dr. Semih BİLGEN
Co-supervisor : Assoc. Prof. Dr. Melek YÜCEL
September 1991, 110 Pages

The presented work is an attempt to recognize a one part melodic form and to obtain the transcription of the input melody with computer. Melody recognition consists in comparing two input melodies on the basis of the squared distance criterion. The comparison is separately performed on the notes and their durations.

In order to obtain the frequency characteristic of the input melody, Hartley Transform is used. Also in order to avoid the unwanted harmonics some heuristics are proposed.

Key words : Melody recognition, computer analysis of musical sound, computer transcription of music.

SCIENCE COD:

ÖZET

TEK SESLİ MELODİNİN BİLGİSAYAR YARDIMI İLE TANINMASI VE NOTASININ ÇIKARTILMASI

GERMEN, Emin

Yüksek Lisans Tezi, Elektrik ve Elektronik Müh. Bl.
Danışman : Doç. Dr. Semih BİLGEN
Yardımcı Danışman : Doç. Dr. Melek YÜCEL
Eylül 1991, 110 Sayfa

Sunulan çalışma, tek sesli bir melodik biçimin tanınması ve girilen melodinin notasının bilgisayarla yazılmasına yöneliktir.

Melodi tanıma, iki melodinin uzaklık karesi ölçütüne göre karşılaştırılmasına dayalıdır. Karşılaştırma, notalar ve süreleri üzerinde ayrı ayrı yapılmaktadır.

Girilen melodinin frekans niteliğinin elde edilmesi için Hartley dönüşümü kullanılmaktadır. İstenmeyen doğuşkanların ayıklanması için bazı buluşsal yöntemler önerilmiştir.

Anahtar Kelimeler : Melodi tanıma, müzik seslerinin bilgisayarlı çözümlenmesi, bilgisayarlı müzik çevriyazımı.

BİLİM SAYISAL KOD: 609.03.04

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Assoc. Prof Dr. Semih BİLGEN for his guidance and encourragement in the formation of this thesis, and for his friendly supervision. I also like to thank to Assoc. Prof. Dr. Melek YÜCEL for her helps and guidance. I am grateful to Özgür İZMİRLİ and Osman ÖZTÜRK for their hardware support.

TABLE OF CONTENTS

ABSTRACT iii

ÖZET iv

ACKNOWLEDGEMENTS v

LIST OF FIGURES viii

1. INTRODUCTION AND THE DEFINITIONS 1

1.1 Key Terms and Definitions 3

1.2 Musical Tones and Their Characteristics 6

1.3 The Temporal Structure of Music 9

1.4 Fourier Analysis and Synthesis of Tones ...10

1.4.1 Orthogonality and Orthogonal Functions 10

1.4.2 Fourier Representation of a Signal 12

2. HISTORICAL BACKGROUND 15

3. PROCESSING OF MUSICAL SOUND 21

3.1 Sampling the Analog Data 21

3.2 The Sampling System 23

3.3 Hartley Transform 24

3.4 Frequency Determination 26

4. RECOGNITION AND UNDERSTANDING OF A MELODIC FORM .28

4.1 Determination of Duration of a Note 28

4.2 Discarding the Harmonics 32

4.3 Melody Transcription 35

4.4 Melody Recognition 36

4.4.1 Coding of the Melodic Form with respect to Interval and Duration ...37
4.4.2 Comparison of Two Melodies 39
5. THE HARDWARE AND SOFTWARE IMPLEMENTATION 47
5.1 The Hardware Implementation 47
5.2 The Software Implementation 49
6. RESULTS AND DISCUSSION 56
REFERENCES 66
APPENDICES
A. SOFTWARE SOURCE LISTING 67
B. THE SCHEMATIC DIAGRAM OF THE INTERFACE CARD 105
C. DATA SHEETS OF ADC 106
D. EIGHT OCTAVES OF MUSICAL NOTE FREQUENCIES 110

LIST OF FIGURES

Figure 1.1 A typical waveform of a musical tone 4

Figure 1.2 A typical example of a noisy signal 4

Figure 1.3 The frequency spectrum of the flute 8

Figure 1.4 The frequency spectrum of the violin 8

Figure 4.1 The exact division of the melodic pattern into the windows 30

Figure 4.2 The stepwise division of the melodic pattern into the windows 32

Figure 4.3 The intervals between the notes 37

Figure 4.4 A note sequence 38

Figure 4.5 Coded form of the sequence 38

Figure 4.6 Melody 1 for melody comparison 40

Figure 4.7 Melody 2 for melody comparison 41

Figure 4.8 The distance matrix of the two melodies .41

Figure 4.9 The same melody in Figure 4.6 with temporal warped form 43

Figure 4.10 Difference between the temporal form of two melodies (Mul Factors 1,1) 44

Figure 4.11 Difference between the temporal form of two melodies (Mul Factors 0.5,1) 45

Figure 4.12 Difference between the temporal form of two melodies (Mul Factors 1,0.5) 45

Figure 5.1 The hardware model of the system 47
Figure 6.1 The score of the J.S.Bach's No 4 Flute Sonata Part 2 Allegro 58
Figure 6.2 The Transcription of J.S.Bach's 4th Flute Sonata 58
Figure 6.3 The Transcription of the melody of the synthesizer's memory 60
Figure 6.4 The score of Telemann's Fantasie No 6 ...61
Figure 6.5 The transcription of Telemann's Fantasie by the system 61
Figure 6.6 The score of Haendel's piece 62
Figure 6.7 The transcription of Haendel's piece 62
Figure 6.8 Two similar melodies for melody comparison 63
Figure 6.9 Two unsimilar melodies for melody comparison 64

CHAPTER 1

INTRODUCTION AND THE DEFINITIONS

Advances in the development of computers, the cognitive sciences, applied artificial intelligence, musical software and hardware have made it imperative to broaden the notion of musicology and open the gates of computer technology into those areas. As a result various techniques, algorithms, heuristics of music cognition, music transcription and melody storage have been proposed, new approaches are developed and introduced into the musical area [Roads 1985, Ebcioğlu 1988, Yavelow 1987, Pickover 1985, Carlos 1986, Loy, Abbott 1985].

The research on the computer music field can be classified mainly into two categories. The first category covers music composition and production with computer, including synthetic tone generation, while the second category covers the analytic perception and cognition of musical sounds, and their interpretation, classification and modeling. That is, the works on computer music have adopted either the analytic or the synthetic approaches.

The subject matter of this thesis is based on the analytical approach. One of the features of this work

is to obtain music transcription from live performed music. The purpose of the system is to ensure the recognition and understanding of the musical sound, which forms a melody pattern, and to obtain the transcription of this given melody both on graphic display and on printed output. Transcription is done on a single sound melody. The other feature is comparing two melodies and finding the correlation between them. These constitute a step in the solution of the macro problems such as, three or four part music transcription, music composition, style analyzing, and tonal model development. The first step which covers the fundamentals has resulted in a previous thesis in M.E.T.U. [İzmirli 1988].

Although the disciplines and the scopes of synthetic and analytic studies are different, their fundamentals and their mathematical bases are similar.

Discussion of the characteristic of the musical sound and explanation of the key terms are the subjects of the next section. A brief literature survey on the related area will be given in chapter 2. Sampling and processing of given musical signal and its mathematical background is the subject of chapter 3. The problem of recognition and perception of musical sound and timing of the given melody will be discussed in the forth chapter. Hardware and software implementation of the system will be dealt in the fifth chapter. In chapter 6, the results

of the work and outputs of the system will be given. Also the further research that can be done on this subject will be discussed in this chapter. The software and related diagrams are given in the appendices.

1.1 Key Terms and Definitions

The first fundamental question about the sound stimulated by a sonorous body is, whether it is a noise or a musical tone. The nature of the difference between musical tones and noise is, the rapidity of alteration of the sound. With harsh and rapid variations, the characteristic of the sound becomes noise, whereas periodically or slowly altered sound corresponds to musical tone. That is to say, the sensation of a musical tone is due to a rapid periodic motion of the sonorous body, however, the sensation of noise is due to non periodic motion [Helmholtz 1954].

Figure 1.1 and Figure 1.2 show the timing characteristics of a musical tone and a noisy signal.

When the spectral plots of the two types of sound are compared, it is observed that, the energy of the noisy tone is widely scattered over the spectrum, however, the spectral energy of the musical tone is condensed around a definite frequency. Frequency is the most distinctive characteristic of various musical

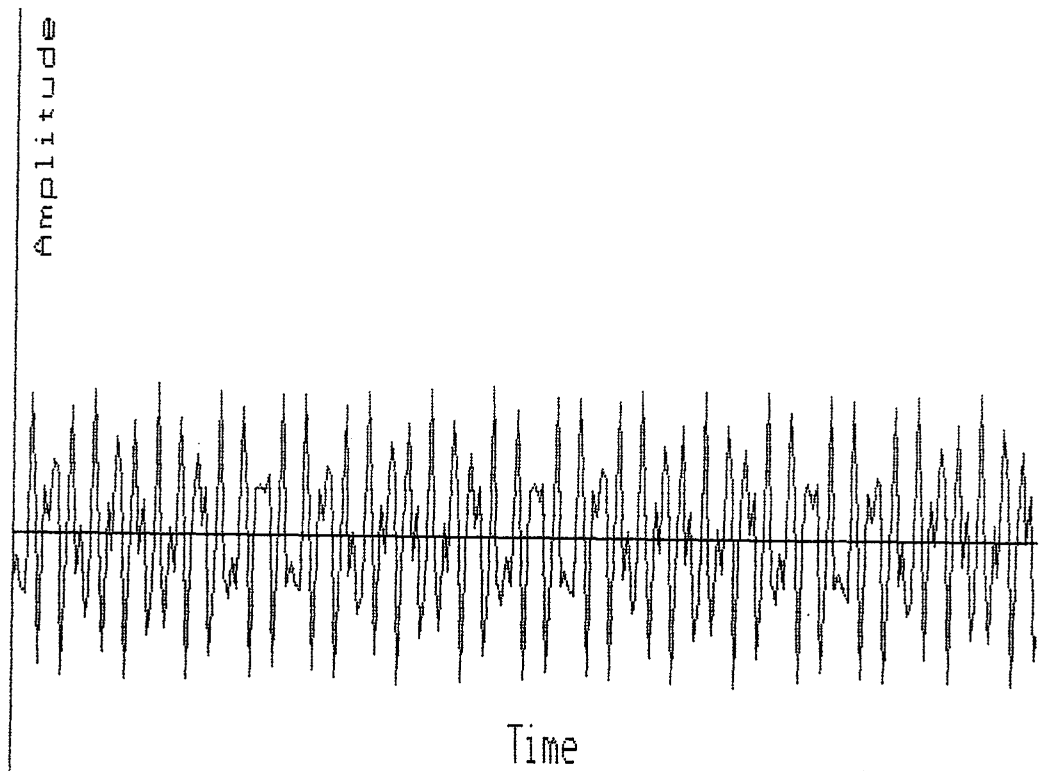


Figure 1.1 A typical waveform of a musical tone. The periodic nature is the main feature of the signal.

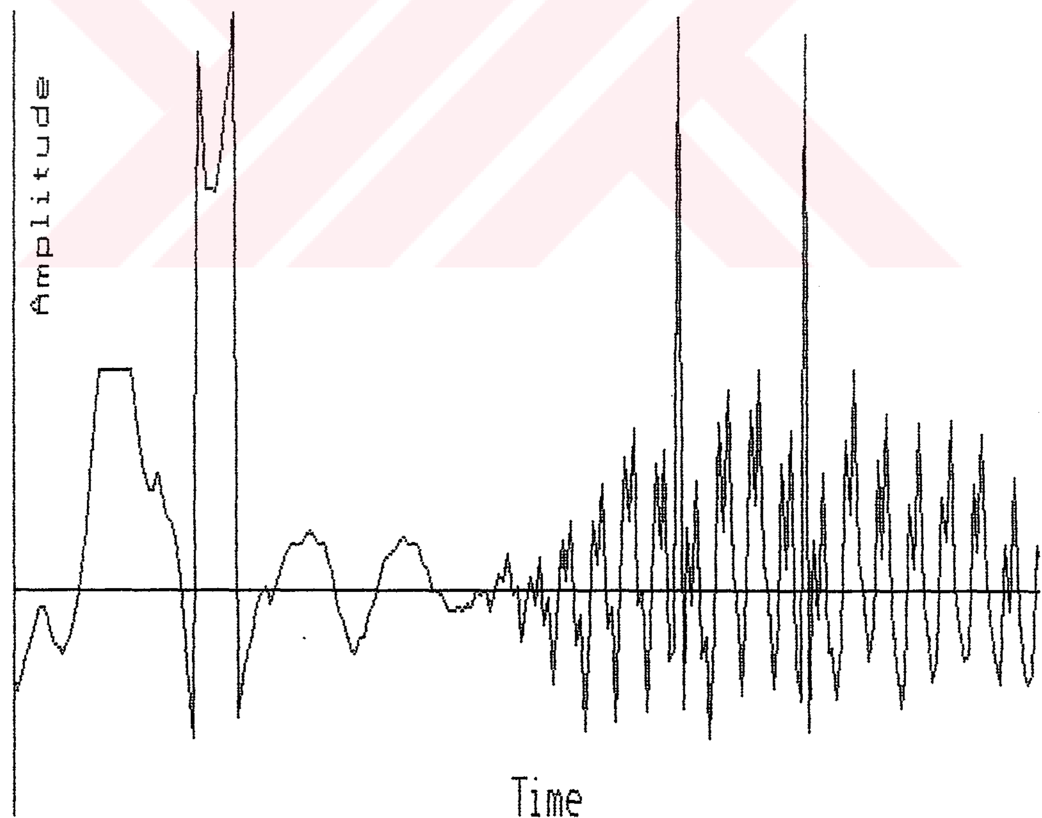


Figure 1.2 A typical example of a noisy signal. An aperiodic waveform, is the characteristic of the signal.

sounds. In musical terminology, the frequency of a tone is called as *pitch*. Musical tones are higher if their pitch number is high and lower otherwise. *Note* is the position of the pitch in the musical scale.

The actual pitch (the subjective sensation of frequency) of the tones the sonorous musical body produces is perceived normally as being a function of the fundamental frequency. The fundamental is usually the component of the complex tone into which most of energy is directed, and in most cases, the fundamental is also the lowest of the tone produced. The other tones present in addition to the fundamental are referred to as harmonics or partials. Precisely speaking, the term partial and harmonic are not synonymous, though for most intents and purposes they are interchangeable. In fact however, any component of a sound is a partial, including the fundamental. A harmonic, however, is a partial occurring at any frequency that is an integral multiple of the fundamental, but not including the fundamental. Consequently, the first harmonic can be referred as the second partial.

Octave is the first harmonic of the tone. In the well-tempered musical structure, an octave is divided into 12 equal intervals, each called a semitone. Notes in an octave, each seperated by a semitone are named as A, A flat, B, C, C flat, D, D flat, E, F, F flat, G, G flat.

The letters ranging from A to G correspond to La to Sol in general musical notation. The notes and their relative frequencies are given in Appendix D.

There are some special cases in which the ear does not perceive the fundamental as the lowest tone produced. This is the phenomenon known as 'missing fundamental'. It is proved that, the pitch of the tone is related to the frequency difference between partials, which does not have to be the same as the frequency of the lowest component, although it usually is [Donnigton 1982]. For example, when objects are excited that give the sound with pitch, equivalent 1200, 1400, and 1600 Hz's at the same instance of time, the tone that can be heard is 200 Hz. [Cardoso 1968]. Because the constant frequency difference between the tones is equivalent to 200 Hz. For complex sounds one faces the same problem. According to the fundamental extracted, to make a decision on whether the pitch related with this fundamental is correct or not is the crucial point of melody recognition.

1.2 Musical Tones And Their Characteristics

The quality of the musical tone depends only on the physical structure of the sound. It is possible to classify the musical sounds in two categories:

quasi-periodic (harmonic) and aperiodic (inharmonic) [Moorer 1977]. The tones in which all the power goes into the production of a particular, unique frequency are called simple tones (pure tones or sine waves). Most sounds we hear in everyday life and in particular, the sounds we encounter when listening to music are not of this simple type. Instead, they are a compound of a number of simple sounds occurring simultaneously in a phase related manner. In other words, the power that goes into producing the sound, is divided between a number of tones of different frequencies. We can define this type of sound as a complex tone. Musical instruments, for example, produce their characteristic sounds, because they produce complex tones which have more or less unique, and typical structure. In quasi-periodic nature, the tones are composed of the sum of integer multiples of the fundamental. However, aperiodic feature implies that the energy is distributed in the frequency spectrum without a regular pattern. Most of the orchestra instruments have quasi- periodic nature.

In Figures 1.3 and 1.4 frequency spectra of two orchestra instruments, flute and violin, are shown. Both instruments play the same note. Their frequency characteristics show that the violin has many partials, but the flute has greater of its energy in the fundamental, making it purer, that is close to a single sinusoid.

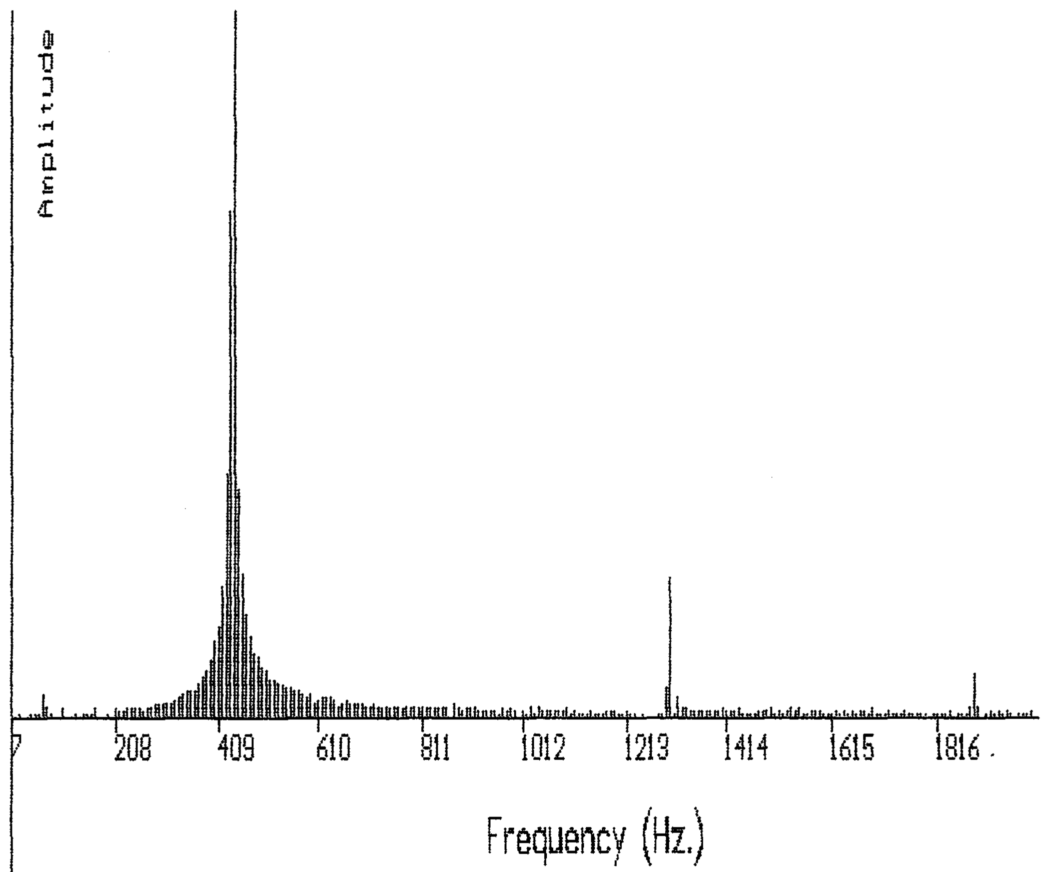


Figure 1.3 The frequency spectrum of the flute. Flute is playing A_4. The energy is concentrated mainly on the fundamental

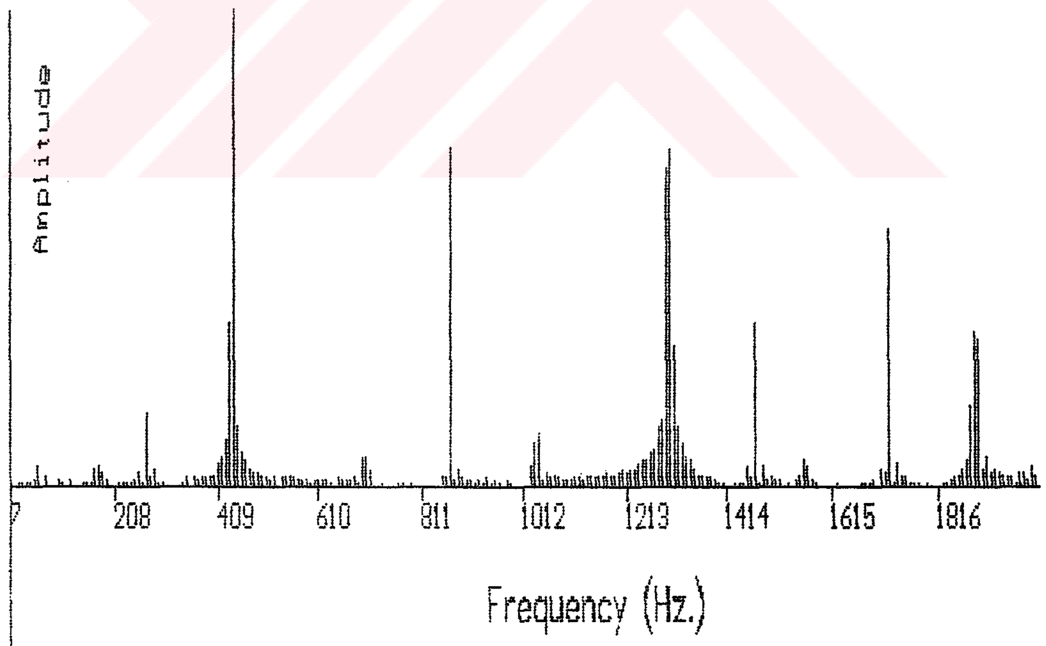


Figure 1.4 The frequency spectrum of the violin. Violin is playing A_4 note. The energy is divided into the partials.

The tones that are considered within the scope of this work are all compound and quasi-periodic tones. The starting point is to extract the simple tones of those complex tones. Examination of the perception of sound by ear can indicate how one should proceed. Helmholtz states that: [Helmholtz 1954]

> "Only that particular motion of the air which we have denominated a simple vibration for which the vibrating particles swing backwards and forwards according to the low of pendular motion, is capable of exciting in the air the sensation of a single simple tone. Every motion of the air, then which corresponds to a composite mass of musical tones, is capable of being analyzed into a sum of simple pendular vibrations, and to each such single simple vibration corresponds a simple tone, sensible to the ear, and having a pitch determined by the periodic time of the corresponding motion of the air."

With pendular motion Helmholtz refers to sinusoidal motion.

1.3 The Temporal Structure of Music

The psychological characteristic of music also depends on the temporal structure. It involves duration, tempo and rhythm. The tempo determines the rapidity or slowness, in other words, the speed of the musical piece. All the notes and rests determine the melodic progression of the piece. All durations are relative to each other and also mainly relative to the tempo. The tempo is given by metronome number. This signifies the number of beats in a minute. The actual speed of a melody pattern is

totally related with this metronome number.

1.4 Fourier Analysis and Synthesis of Tones

It has been stated before that the tones that enter our scope are complex and quasi-periodic tones. Analyzing this type of a sound signal requires to extract the simple sounds which can be considered as the fundamental of some other frequencies. If those extra components can be extracted, the frequency of the remaining simple sound will correspond to the note of the tone.

One of the most powerful and useful conceptual tools for determining the frequencies which build up the complex tone is the Fourier Transform. The French mathematician Joseph Fourier, had proved that, any periodic waveform can be represented as a linear combination of possibly an infinite number of simple periodic functions. Fourier theory uses a set of orthogonal sinusoids to expand a given periodic function.

1.4.1 Orthogonality and Orthogonal Functions

A set of continuous real valued functions $\{y_n(t)\} = \{y_0(t), y_1(t), \ldots y_n(t)\}$ is said to be *orthogonal* in the interval $(t, t+T)$ if

$$\int_T y_m(t)\, y_n(t)\, dt = \begin{cases} c & \text{if } m \neq n \\ 0 & \text{if } m = n \end{cases} \tag{1.1}$$

where $\int_T$ denotes $\int_t^{t+T}$. If $c = 1$ then the set is said to be an *orthonormal* set [Rao 1975].

Now let's consider a signal which can be represented as:

$$x(t) = \sum_{n=0}^{\infty} a_n y_n(t) \tag{1.2}$$

where a_n defines the nth coefficient of the expansion. If we can find the coefficients, then it is easy to represent $x(t)$ with the weighted sum of an orthogonal set.

since

$$\int_T x(t)\, y_m(t)\, dt = \int_T \sum_{n=0}^{\infty} a_n\, y_n(t)\, y_m(t)\, dt \tag{1.3}$$

using equation (1.1) it is found that;

$$a_m = \frac{1}{c} \int_T x(t)\, y_m(t)\, dt \quad m = 0,1,2,... \tag{1.4}$$

Thus, the signal x(t) can be represented by the weighted sum of orthogonal functions. The multiplication factors of those functions are constitude the infinite but countable set $\{a_0, a_1, ...\}$.

1.4.2 Fourier Representation of a Signal

If the orthogonal set $\{u_n(t)\}$ is chosen as $\{\cos(nw_0t), \sin(nw_0t)\}$ for $n = 0,1,2,...,\infty$, then the transformation is called the Fourier Transformation. The series expansion which corresponds to equation (1.2) can be written as follows;

$$x(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos(nw_0t) + \sum_{n=1}^{\infty} b_n \sin(nw_0t) \qquad (1.5)$$

where, w_0 (radian per second) is the fundamental angular frequency which is related to the period T of the function by the formula $T = 2\pi/w_0$.The frequencies nw_0 contribute to the harmonics of the given fundamental. The coefficients $\{a_n, b_n\}$ are the amplitudes of the harmonics, and a_o is the amplitude of the DC component of the signal.

The coefficients $\{a_n, b_n)$ can be computed by using the orthogonal properties of set of functions $\{\cos(nw_0t), \sin(nw_0t)\}$ over the period T such as:

$$\int_T \cos(nw_0t)\cos(mw_0t) = \begin{cases} T/2, & m=n \\ 0, & m\neq n \end{cases}$$

$$\int_T \cos(nw_0t)\sin(mw_0t) = 0 \text{ for all m and n} \qquad (1.6)$$

$$\int_T \sin(nw_0t)\sin(mw_0t) = \begin{cases} T/2, & m=n \\ 0, & m\neq n \end{cases}$$

Using equation (1.6) and orthogonality properties it is found that

$$a_0 = \frac{1}{T}\int_T x(t)\,dt$$

$$a_n = \frac{2}{T}\int_T x(t)\cos(nw_0t)\,dt \qquad (1.7)$$

$$b_n = \frac{2}{T}\int_T x(t)\sin(nw_0t)\,dt$$

The Hartley Transform used in this work to analyze the music signals is developed using the same orthogonality principle; its orthogonal basis functions are defined as

$$cas(2\pi ft) = \cos(2\pi ft) + \sin(2\pi ft) \qquad (1.8)$$

The reason for preferring the Hartley Transform

is that, rather a complex domain representation, it generates a real transform sequence. A close overview of the Hartley Transform is given in chapter 3.

CHAPTER 2

HISTORICAL BACKGROUND

The extraordinary rapid and creative development in the computer music field, was made possible by the technological development in electronics and by the requirements of the artists of our age to find new styles and new forms. In the early ages, Greek philosophers and mathematicians had worked on tonal patterns and proposed their theories on sound and sensation, from both psychological and scientific points of view. The tones and the relationship between sounds, stimulated from different sonorous bodies, had attracted so many scientists through the ages since those times [The Larousse 1984]. In 1877, Helmholtz had collected all the knowledge about sound and hearing, and written the book "On The Sensation Of Tone" [Helmholtz 1954]. Even today, this book is a significant reference, and the terminology and theory it proposes keep their validity.

After the computer has come into existence, a new discipline has opened its gates to musicians and scientists. At the beginning, the most widely used application areas of computer music involved the synthesis and processing of digital sound because the

sounds generated and synthesized by computers, had not been heard before. This feature has since then drawn the attraction of musicians and the commercial market.

The notable contribution of computer to the musical field is the capability of running of sound experiments in a very precise manner. Its significance is that the mathematical models can be turned into the musical structures, by the help of computer. Because with a digital to analog converter and digital sound models , it is possible to obtain a real sound. One more significance of the computer in music is, that musicians can make their own music without needing performers and they can design their own sounds. This opens the gates of science to music and the concept of scientific musicology comes into existence.

High fidelity sound synthesizing requires highly developed signal processing techniques, as well as improved hardware and sophisticated software. It means that, too much computing power was necessary. In order to meet those requirements, complicated hardware and software has been developed, and with this developed technology the scope of computer music has been broadened.

The programmable structure of computers with predetermined forms, attracts many composers into the

computer music field. The first composer who used the computer first in music composition was Lejaren Hiller [Mackay 1981] at the University of Illinois in 1957. He produced the "Illiac Suite". The well known Greek composer, scientist and architect, Xenakis also used the computer in composition. He has proposed a method of composition with probabilistic theories, and his music has been called stochastic music. After John Cage [1911-] and Karlheinz Stockhausen [1918-] the computer was accepted as one of the instruments of the Avant-Garde music style [Mackay 1981]. Several branches of computer science now support computer music, including discrete time signal processing, multitasking, interactive programming, expert systems, and user friendly programming algorithms.

With the standardization of musical data communication protocols, commercial synthesizers have been widely spread in the markets, and desk-top music studios have come into existence. The standardization has been done by the MIDI (Musical Instruments Digital Interface) association and it involves the protocol in a serial, current-loop data transmission. The MIDI standard specifies a coding for note-on, note-off, note velocity, key pressure, and other parameters related to the musical keyboard playing. With MIDI it is possible to obtain music with 16 channels capacity [Gualtieri 1986].

Artificial intelligence (AI) also has many contributions to the computer music field. The survey by Curtis Roads "Researches in Music and Artificial Intelligence" [Roads 1985] covers the technical aspects of AI in four areas of musical research. Those are: composition, performance, music theory and digital sound processing.

A major area of AI research is the knowledge based systems. In order to construct the theoretical basis on the musical structure, it is desired to make connections between the small elements of the musical structure. Those minimal structures construct the knowledge basis. This knowledge basis is used to organize the parsing and searching of the formal patterns in the musical domain in order to understand it [Roads 1985].

The "Workshop for Musical Notation by Computer" held in Switzerland in 1987, [Yavelow 1987] has been related with the topic, "The Graphical Interaction of Computer and User". Music scoring systems and data encoding schemes, were dealt with in the mentioned workshop. The system introduced by Giovanni Muller is an appreciable example of musical transcription hardware and software. In this system: data can be input from a MIDI device and input data are converted to a transcribed notation via an "Intelligent Translator". All the parameters and control code generations to convert the

data, depend on the user.

Another example of automatic music transcription, is the Moorer's research [Roads 1985]. In this work; a bank of heterodyne filters are used and the data are interpreted by those filters. At the end, digitally recorded two part guitar music is transcribed by the computer in two part notation.

There are considerably many applications of artificial intelligence on the recognition and understanding of a musical tone and on the modeling of melody patterns. With the effect of speech processing applications on analyzing techniques, a remarkable progress can be observed. Before those applications only a few research projects had been carried out in the area of analyzing and recognition of music. Knowledge-based signal processing, and signal to symbol transformation, are the fields of artificial intelligence related with listening and understanding of music.

A notable research on melodic pattern description and on artificial intelligence for music composition has been done by Kemal Ebcioğlu [Ebcioğlu 1981]. That study consists of a LISP program for counterpoint generation by computer. Ebcioğlu has also presented a remarkable study on artificial intelligence related with harmonizing a melody with a predetermined

style. He has proposed an expert system called CHORAL. This system creates a harmonization of a given melody with Bach style [Ebcioğlu 1988].

In M.E.T.U. there is ongoing research on analyses of tones. An earlier work covers an attempt to extract all notes sounded simultaneously in the form of a chord by analyzing the excitations of different tones and on the energy distribution of the musical sound patterns [İzmirli 1988].

CHAPTER 3

PROCESSING OF MUSICAL SOUND

All the sound signals that we hear in our natural environment have the analog signal characteristics. If they are converted to electrical signals via a transducer, it is observed that, the signal are continuous in the time domain. However in order to process these signals by computer, it is necessary to convert them into the discrete form. In other words, the analog data has to be sampled, digitized and stored in the computer memory before being processed.

3.1 Sampling the Analog Data

Analog to digital data conversion is essentially a three stage process. At the beginning, the analog data is lowpass-filtered. The filter removes the high frequency components of the input signal over the half of the sampling frequency R, to prevent the aliasing effect. The filtered signal is sampled with a sample rate R in the second stage of the conversion. At the end of the process, the sampled discrete signals are quantized and converted to the numeric values.

The analog signal and the sample values are related as

$$x^{*}(t) \simeq \Delta t \sum_{m=0}^{N-1} x(m\Delta t)\, \delta(t-m\Delta t) \qquad (3.1)$$

Where $x(t)$ represents the analog signal that is a continuous function of time t. $x^{*}(t)$ is the discrete (or sampled) signal obtained from $x(t)$. It is sampled at a specific rate of R samples per second. Δt is the sampling interval which is equal to 1/R. $\delta(t)$ denotes the dirac delta function. The result of the sampling is the discrete data sequences $\{x(m)\}$, which m = 0,1,2,...N-1, where $x(m) = x(mt)$ also [Rao 1975]. After the sampling process, $x(mt)$ is quantized and coded in digital form.

The signal at the beginning is passed from a lowpass filter. If a signal $x(t)$ is band limited with bandwidth B Hz, then this signal can be uniquely determined from its sampled form, obtained by sampling it at R samples per second if R ≥ 2*B. This rate is called as Nyquist Rate [Rao 1975].

If it is attempted to sample the analog data with sampling frequency R less than the Nyquist Rate as R<2*B, to reconstruct the original signal is impossible

because of the Aliasing effect. So in order to obtain the true characteristic of the analog data in digital data the musical signal is passed from the lowpass filter which has a filter characteristic pass the frequencies lower than R/2.

3.2 The Sampling System

Although the range of the sensation of the human ear is extended in frequency range from 20 Hz. to 20 kHz, the fundamental frequencies of musical sounds do not approach those limits. For example the most treble instrument in an orchestra is the piccolo, and the most treble note that a piccolo can produce is the eight octave C note which corresponds to the frequency value 4186 Hz. However, when we listen to music, we hear the frequencies more than 10 kHz. which are totally related with the upper harmonics of the musical instruments. Those upper harmonics determine the timbrel characteristic of the instrument.

The frequency range of the system implemented in this work is from 0 Hz. to 2048 Hz. There is a filter in the analog stage before the sampling stage, which cuts the frequencies above 2048 Hz. The sampling frequency of the system is 4096 Hz., that enables one to analyze data having a bandwidth below 2048 Hz. This is the half of the

sampling rate. Since in this work the attention is concentrated on the frequencies that determines the musical notes the sampling rate is well enough for melody recognition. The upper frequency that can be analyzed corresponds to the B_6 note, which is about 3 octaves above middle C. (Middle C is the note C which resides in the middle of the piano keyboard and it is a convention to take this note as a reference to identify the relative octaves of the notes in the musical terminology)

After the sampling process, the sampled data are quantized and stored in memory. In order to obtain quantized data an analog to digital (ADC) converter is used. In chapter 5 a more detailed explanation is given.

3.3 Hartley Transform

It has been stated in chapter 1 that, the most common method of extracting the frequencies from a composite signal, is the use of Fourier transforms. After Cooley-Tukey found an algorithm to take the transform quickly, Fast Fourier Transform (FFT) has been the principal feature of digital signal processing [Cooley, Tukey 1965].

The Fourier transform maps the signal in the time domain to the complex frequency domain [Ref. chapter

1] The Hartley transform also maps the signal in the time domain to the frequency domain, but the frequency domain coefficients are not complex valued. This distinction halves the number of required arithmetic operations since complex multiplication requires four operations, and a complex addition requires two operations, while a real multiplication requires two operations, and a real addition requires one operation.

Furthermore, to keep the transformed data in memory, complex data array requires twice the memory of real array storage. This means that, with Hartley transform, more data samples can be kept and processed in the memory.

In order to obtain the frequency domain characteristic from time domain, or reversely, to map the frequency domain signal into time domain, Hartley transform functions are as follows: [Bracewell 1984]

Given a real sequence x[n] for n=0,1,...N-1,

$$H[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] \, \mathrm{cas}(2\pi nk/N) \qquad (3.2)$$

for $k = 0,1,2,...,N-1$

$$x[n] = \sum_{m=0}^{N-1} H[k]\ cas(2\pi nk/N) \qquad (3.3)$$

for $n = 0,1,2,\ldots,N-1$

where $cas(2\pi nk/N) = \cos(2\pi nk/N) + \sin(2\pi nk/N)$

The squared magnitude of the corresponding Hartley Transform is proportional to $[H^2(k) + H^2(-k)]$. It gives us the weight factors of corresponding frequencies in frequency domain.

The symbol n can be regarded as a time parameter and k/N can be regarded as a frequency measured in cycles per unit of time.

The fast Hartley transform (FHT) algorithm has been designed and proposed by Bracewell, and like the fast Fourier transform (FFT), it uses the same computational principles and butterfly type algorithms. The number of arithmetic operations is of the order of $N\log_2 N$ like FFT [Bracewell 1984].

3.4 Frequency Determination

The size of data *window*, i.e. the number of data points (N) which enter and leave the FHT algorithm, should be chosen as a power of two. The sampling rate

divided by window size, determines the frequency resolution of the system.

In this work N (window size) is chosen as 512; since a frequency resolution of 4096 / 512 = 8 Hz. is considered to be sufficient. Although doubling the window size would yield a frequency resolution of 4 Hz., it would also increase the computational effort.

Among the 512 frequency coefficients obtained using the FHT algorithm; only first 256 corresponding to the frequency range of our interest (0-2048 Hz.) are used.

CHAPTER 4

RECOGNITION AND UNDERSTANDING OF A MELODIC FORM

When the physical aspects of music are considered, it is observed that considerable differences exist between the sound recognized by ear, and the actual physical nature of the sound. The composite and complex characteristics of the tonal pattern constitute the reasons of this problem.

In order to define a melodic structure, mainly the frequencies that are the pitches of the notes have to be drawn out. This has been discussed above in section 1.1. The relative timing of the notes also have to be determined since a melodic pattern consist of notes essentially. This is considered below.

4.1 Determination of Duration of a Note

The system has been designed so that one window corresponds to a sixteenth note from the duration point of view. In the musical terminology, the duration of the addition of 4 sixteenth notes corresponds to one beat of metronome sound. In other words, the time that passes from one beat sound to another is divided into four, and

each division corresponds to a sixteenth note. Hence, each half period is represented by an eighth note. The duration of two sixteenth note is the same as the duration of one eighth note, and two eighth note duration also is the same as the duration of one fourth note.

In order to process the melody entirely, the whole sampled data are divided into windows. After processing one window, the frequency, that is, the pitch number of the related sixteenth note, is determined. If the consecutive window following the processed window has the same pitch, the relative timing values are added, and the duration of the relevant pitch is recognized.

The duration of one window is 125 msec. By this sampling period rate, a maximum of 8 windows in one second can be sampled. This corresponds to 4 windows in half a second. With this sampling rate it is possible to grasp the sixteenth notes in Allegro tempo. (Allegro is a tempo term in music which corresponds to 120 beats per minute). Allegro is the maximum limit to determine the sixteenth notes in the melody, which in this work will be considered fast enough to define the melodic pattern.

It has been stated before that, the duration of the note is related to the tempo. The period of a sixteenth note is calculated as follows:

$$\text{Freq. of a quarter note} = \frac{\text{tempo (beat/min)}}{60\ \text{(sec/min)}}$$

Freq. of a sixteenth note = 4 * Freq.of a quarter note.

For example, with tempo rate 90, The duration of a sixteenth note will be: 1/((90/6)0*4) = 167 msec.

In Figure 4.1, the tempo of the melodic pattern is 120 bpm. In this tempo rate, one window corresponds to 120 msec. which is the sixteenth note rate of Allegro. In this case, the window is shifted through the whole data with 512 point steps.

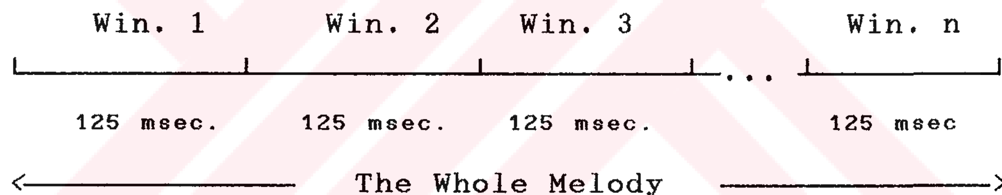


Figure 4.1 The exact division of the melodic pattern into the windows

When the tempo rate is less then 120 bpm (Allegro), tracing the whole sample is wasteful because the system has been designed to grasp the sixteenth and lower notes (e.g. eighth, quarter, and half notes). As long as the sampling rate of analog data is constant, the

duration of a window doesn't change. So, in order to obtain correct window positions, the window is sled through the whole sample by definite steps. It is possible to represent the melody with combinations of sixteenth notes, since the other notes with higher durations are composed of constant multiples of sixteenth notes. For example it is conceivable to divide a fourth note into four and represent it with 4 sixteenth notes.

The duration and the tempo of the melody determine the amount of sixteenth notes that is found in the melody. For example, in a melody with 100 tempo value and of 5 seconds duration, the number of sixteenth notes is :

$$5 * 4 * (100 / 60) = 33.$$

This is also the number of windows in the proposed system. To find the sixteenth note of the melody, the entire data sequence is divided into 33 steps. With the beginning of each step, the 512 points window is taken and processed.

In Figure 4.2 the tempo value of the melody is less then 120, and the window is sled through the data sequence by calculated steps.

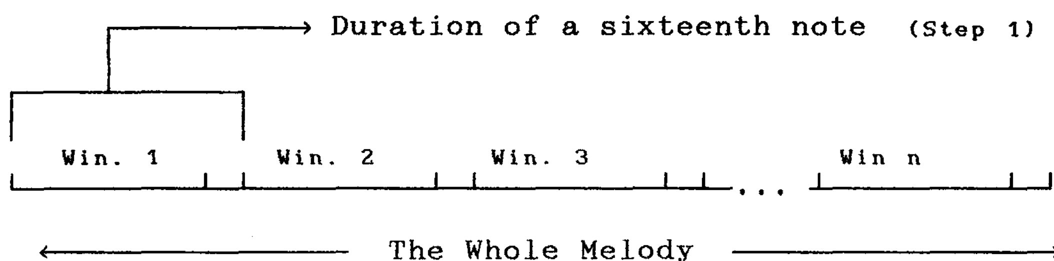


Figure 4.2 The stepwise division of the melodic pattern into the windows.

4.2 Discarding the Harmonics

In one part music transcription, melody has a one dimensional characteristic. The fundamental determines the pitch of the note. Extraction of the fundamental is the principal step of determination of the pitch. However, there are certain difficulties to grasp the fundamental.

The timbre of the instrument, determines the amplitudes of the constituent frequencies and their relative positions on the spectrum. This is due to the vibrating characteristics and the acoustics properties of sonorous bodies. The style of playing of an instrument also has effects on the energies of the partials. The missing fundamental is the result of this undeterministic nature of the musical instruments. Although the lowest partial determines the fundamental, energy of the upper partials may be greater then the fundamental.

For example, according to the timbrel characteristic of an instrument and style of playing, say, A_4(440.0 Hz) and E_5(659,26 Hz.) have been found in harmonic spectrum. E_5 is not an integer multiple of A_4. That is E_5 is not a harmonic of A_4, but they are both harmonics of A_3(220.0 Hz). In this situation, A_3 is missing from the spectrum, but it is heard, since the difference of E_5 and A_4 is A_3. As in this situation, it is necessary to decide whether the set of frequencies found in spectrum contains the fundamental or not.

In this implementation, in frequency domain, a threshold value has been defined, and the amplitudes that exceed this value, are regarded as excited frequencies. In order to grasp the missing fundamental the transformed data are processed according to the following heuristic algorithm:

<u>Algorithm 1:</u>

i) Denote the magnitude of processed samples as a_i
with (i=1,2,...,n); Where n = N/2 = 256

ii) fundamental <--- 1;

iii) partial1 <--- fundamental * 2 ;
partial2 <--- fundamental * 3; if partial2 > n then Goto (vi);

iv) if ($a_{partial1}$ > threshold and $a_{partial2}$ > threshold)
then $a_{fundamental} = a_{partial1} + a_{partial2}$;

v) fundamental <--- fundamental + 1; Goto (iii);

vi) End.

The existence of the first and second harmonics is enough to decide on the existence of the related fundamental. The algorithm above, looks to the first and second harmonics of a frequency component and if they exceed the threshold value, their amplitudes are added and assigned as the amplitude at the processed frequency.

The elimination of the upper harmonics is the next step on the pitch determination. In order to eliminate the harmonics, data samples are processed according to the following heuristic algorithm:

<u>Algorithm 2:</u>

i) Denote the magnitude of the processed samples as a_i
 with (i=1,2,...,n) Where n = N/2 = 256

ii) count <--- 1

iii) harmonic <--- 2*count ; if harmonic > n then Goto (ix)

iv) if a_{count} < threshold then Goto (viii)

v) if harmonic > n Goto (viii)

vi) $a_{harmonic}$ <--- 0

vii) harmonic <--- harmonic + count ; Goto (v)

viii) count <--- count + 1; Goto (iii)

ix) end.

The algorithm above, eliminates the upper harmonics which exceed the threshold level. At the end, the maximum lowest amplitude determines the exact pitch value.

4.3 Melody Transcription

Melody transcription is one of the most important aspects of this work. It aims to print the notes and the rests of the melody with their durations on the staff. The transcription is designed on the basis of G clef. (In musical notation F clef, C clef etc. can be used as a basis) All the notes are mapped to the staff by using the second line of the staff as the place of G note.

In order to find the notes of the melody the extracted frequencies are mapped to a note. Mapping is done according to the equal tempered scale (ref Sec.1). The entire frequency scale is divided into the half notes logarithmically. Around each half note, the ± quarter tone range is defined. The frequency that falls into this range is mapped to the related half note.

After the values of the notes are found, their durations are assigned to those notes. So the whole melodic form can be converted to the transcribed pattern.

The software that is designed in this work prints the transcribed melody on the graphic display. If it is desired to get the notes of the melody on the printed paper, a matrix printer can make a hard copy of the partition of the melody.

4.4 Melody Recognition

Melody recognition is another important aspect of this thesis. It aims to determine the similarity of two melodies that are input to the system, by using both note sequence and the durations of each note of both melodies. At the first glance it comes to mind that one to one comparison of the notes and the duration is enough to decide whether the two melodies have resemblance with each other or not. But if we think from the point of view of the melodic progression, an important problem comes into the existence: A musical piece with the same form can be written in different scales. For example; a C major scale melody can be played in A flat major or in G major or in another kind of scale. In such cases, the melodic structure does not change but the notes of the melody show variations. This is called "transposing" in musical terminology. If a melody is transposed to another scale, just the notes of the melody change, but the relative distances of successive notes remain the same. This is the crucial point to grasp the closeness of two

melodic structures.

In well-tempered musical notation, between each pair of consecutive notes there is a definite interval value which is an integer multiple of 1/2. In Figure 4.3 the intervals between the notes in one octave are shown.

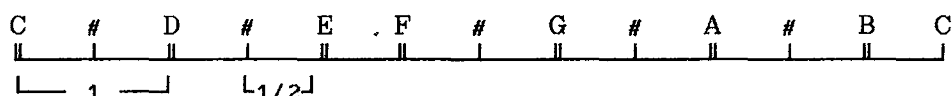


Figure 4.3 The intervals between the notes

In order to understand the melodic structure independent of the scale that is played, the whole pattern is translated to the intervals of the consecutive notes. In this translation 1/2 intervals, for example, the interval C-C# is taken as 1 unit, and 1 interval is taken as 2 units for convenience. After the translation of all notes to differential intervals, it is necessary to assign the note durations to the related interval.

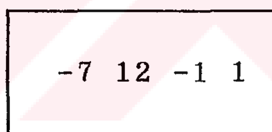
4.4.1 Coding of the Melodic Form with respect to Interval and Duration

In order to translate the melodic form to the interval and duration coded forms, the following method will be proposed:

i) Interval code: The interval between a note and the consecutive note is found and recorded, regardless of the initial note. The interval code corresponding to the note sequence shown in Figure 4.4 is given in Figure 4.5 (a). This code should be interpreted as follows: The second note is 7 half intervals below the initial note, the third note is 12 half intervals above the second note, etc. The code of the last note is discarded because of there is no successive note after than. If there is a rest in the melody, the interval number is chosen arbitrarily as 200, and the following note is coded with respect to the last note which appears before the rest.

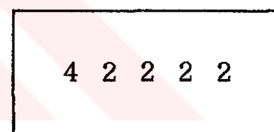
Figure 4.4 A note sequence

(a)



-7 12 -1 1

(b)



4 2 2 2 2

Figure 4.5 Coded form of the sequence

a) Interval code b) Duration code

ii) Duration code: The duration code of the same note sequence can be generated as in Figure 4.5 (b), where each number represents the duration of a note as a multiple of the duration of a sixteenth note, i.e., first note lasts for 4 sixteenth , second note lasts for 2 sixteenth, etc.

4.4.2 Comparison of Two Melodies

Comparison of two melodies is one of the basic aims of melody recognition. That is if it is possible to distinguish any two melodies, we can identify the musical forms.

In order to compare two melodies, first it is necessary to derive the interval and duration codes of the melodic forms. The procedure mentioned above to generate the interval code is used to find the interval progression of the melody. After this stage, if the forms of two melodies are the same in different scales, it is possible to say that the two melodies are identical in melodic structure.

The next step is to find the distance between the two melodies in terms of melody and tempo. This consists of two procedures. The first procedure just finds the the distance between the interval codes without regarding the duration of the notes, which gives a measure of the melodic difference. If melodies are correlated to some degree, then, the distance between the duration codes is considered. With this approach, it is possible to differentiate the similar melodies with different tempo rates.

a) Interval Difference:

In order to find the distance between the melodic structures, only the interval values are considered. Interval code of the first melody is stored in the first column of a matrix. Interval code of the second melody forms the first row of the same matrix. One to one correspondence of each element of the first row with the elements of the first column corresponds to the matching of the all notes in the melody.

In order to derive a measure for the similarity between the two melodies, the squared differences between all possible pairs chosen from the two interval codes are found as the matrix elements. Every element in the first row is matched with the first column that represents the second melody. The sum of the diagonal elements give the unsimilarity measure in terms of the melodic distance.

In Figure 4.6 and 4.7 two melodies are given. The interval codes of two melodies form the first row and the first column of the distance matrix.

Figure 4.6 Melody 1 for melody comparison

Figure 4.7 Melody 2 for melody comparison

The interval codes of the two melodies are:

First melody :-2 2 2 -2 -2 -1 -2 3 -1 -2

Second melody :1 -2 2 2 -2 -2 -1 -2 3 -1 -2 2

The squared distance matrix is given in Figure 4.8

	1	-2	2	2	-2	-2	-1	-2	3	-1	-2	2
-2	9	0	16	16	0	0	1	0	25	1	0	16
2	1	16	0	0	16	16	9	16	1	9	16	0
2	1	16	0	0	16	16	9	16	1	9	16	0
-2	9	0	16	16	0	0	1	0	25	1	0	16
-2	9	0	16	16	0	0	1	0	25	1	0	16
-1	4	1	9	9	1	1	0	1	16	0	1	9
-2	9	0	16	16	0	0	1	0	25	1	0	16
3	4	25	1	1	25	25	16	25	0	16	25	1
-1	4	1	9	9	1	1	0	1	16	0	1	9
-2	9	0	16	16	0	0	1	0	25	1	0	16
Sum of squared differences ....:										85	0	92

Figure 4.8 The distance matrix of the two melodies.

In this proposed method, the similarity is inversely proportional with the sum of differences. The greater the sum, the less is the similarity between the melodies.

An advantage of this method is that, it is possible to find a melody which is covered by the other melody. We have to say that two melodies are the same, if one melody is a passage in another melody. In other words, a short melody which is a part of a long melody, can be identified by calculating the diagonal sums above the main diagonal (if the second melody covers the first melody) or below the main diagonal (if the first melody covers the second melody).

In the above example, if the main diagonal of the distance matrix is considered, the distance between the first 11 notes of the two melodies is found. The sum of the main diagonal terms is so large that one can conclude at the dissimilarity of the first melody to the first 11 notes of the second melody. However, if the diagonal above the main diagonal is considered, it is observed that the sum is 0, meaning that, first melody is similar to the passage in the second melody between its second and 12'th notes. In the proposed system, all possible diagonals for matching the melodies are investigated and according to the smallest sum of squared differences it is decided whether they have any resemblance or not.

b) Temporal Difference:

After the decision on resemblance of the notes, the temporal structures of the melodies are investigated. This raises another problem: It is necessary to decide that the two melodies have a big similarity in temporal form if the tempo rate of one of the melody is a multiple of the other. For example a melody can be played with 60 bpm. tempo rate and the same melody can be played with 100 bpm. tempo rate. This is called *'temporal warping'* So we have to decide that the two melodies are the same in this case. The melody given in Figure 4.9 is an example to this problem. In this case the same melody in Figure 4.6 has been played two times slower than before. So the every eighth notes change with fourth notes In this case the melodic progression is the same but slower than before.

Figure 4.9 The same melody in Figure 4.6 with temporal warped form.

In order to discriminate such cases among similar melodic structures, the temporal structure should also be investigated. The squared distance between the duration codes of the related melodies is calculated for that purpose. In order to find the temporal warping, the

duration of the melodies are multiplied with 0.5 1.5, 2, and squared differences are calculated for all. The smallest of the temporal distance gives the maximum similarity of the timing structures of the melodies. So the temporal warping problem disappears.

The duration codes of two melodies in Figures 4.6 and 4.9 are:

First Melody : 2 3 2 2 2 2 4 2 2 2 2

Second Melody : 4 4 4 4 4 4 8 4 3 4 1

Multiplication factors of Row:1, Column:1

	4	4	4	4	4	4	8	4	3	4	1
2	4	4	4	4	4	4	36	4	1	4	1
3	1	1	1	1	1	1	25	1	0	1	4
2	4	4	4	4	4	4	36	4	1	4	1
2	4	4	4	4	4	4	36	4	1	4	1
2	4	4	4	4	4	4	36	4	1	4	1
2	4	4	4	4	4	4	36	4	1	4	1
4	0	0	0	0	0	0	16	0	1	0	9
2	4	4	4	4	4	4	36	4	1	4	1
2	4	4	4	4	4	4	36	4	1	4	1
2	4	4	4	4	4	4	36	4	1	4	1
2	4	4	4	4	4	4	36	4	1	4	1

Sum of squared difference:...... 47

Figure 4.10 Difference between the temporal form of two melodies. (Mul Factors 1,1)

Multiplication factors of Row:0.5, Column:1

	2	2	2	2	2	2	4	2	2	2	1
2	0	0	0	0	0	0	4	0	0	0	1
3	1	1	1	1	1	1	1	1	1	1	4
2	0	0	0	0	0	0	9	0	0	0	1
2	0	0	0	0	0	0	4	0	0	0	1
2	0	0	0	0	0	0	4	0	0	0	1
2	0	0	0	0	0	0	4	0	0	0	1
4	4	4	4	4	4	4	0	4	4	4	9
2	0	0	0	0	0	0	4	0	0	0	1
2	0	0	0	0	0	0	4	0	0	0	1
2	0	0	0	0	0	0	4	0	0	0	1
2	0	0	0	0	0	0	4	0	0	0	1

Sum of squared difference:...... 2

Figure 4.11 Difference between the temporal form of two melodies. (Mul Factors 0.5,1)

Multiplication factors of Row:1, Column:0.5

	4	4	4	4	4	4	8	4	3	4	1
1	9	6	9	9	9	9	49	9	4	9	0
2	4	4	4	4	4	4	36	4	1	4	1
1	9	6	9	9	9	9	49	9	4	9	0
1	9	6	9	9	9	9	49	9	4	9	0
1	9	6	9	9	9	9	49	9	4	9	0
1	9	6	9	9	9	9	49	9	4	9	0
2	4	4	4	4	4	4	36	4	1	4	1
1	9	6	9	9	9	9	49	9	4	9	0
1	9	6	9	9	9	9	49	9	4	9	0
1	9	6	9	9	9	9	49	9	4	9	0
1	9	6	9	9	9	9	49	9	4	9	0

Sum of squared difference:...... 103

Figure 4.12 Difference between the temporal form of two melodies. (Mul Factors 1,0.5)

In Figures 4.10, 4.11 and 4.12 the duration codes of the two melodies form the first column and the first row of the difference matrices. For each figure, the duration codes are multiplied with different constants. Those constants determine the timing relation of the two melodic forms. It is found from the above matrices, that the tempo rate of the melody in Figure 4.9 is half of the melody in Figure 4.6. since the relative temporal distances of the two melodies is the minimum when the duration code of the second melody is multiplied by 0.5 value. Those are calculated by the developed software.

After finding the melodic distance between melodies in terms of squared differences between interval codes, and corresponding temporal distance in terms of squared differences between duration codes, it is decided whether two melodies are similar or not. If the sum of difference is 0, it is decided the two melodies are the same. The distance from 0 gives the unsimilarity degree of the two melodies.

CHAPTER 5

THE HARDWARE AND SOFTWARE IMPLEMENTATION

5.1 The Hardware Implementation

The hardware consists of a microphone, (1st stage), a circuit which includes a microphone amplifier and a low-pass filter, (2nd stage), an interface card to digitize the analog input, (3rd stage) and a 80286 based personal computer (4th stage). The interface card is inserted to the 62 pin slot of the computer. The Figure 4.1 shows the hardware modeling of the system. The schematic diagram of the circuit on the interface card is given in Appendix B.

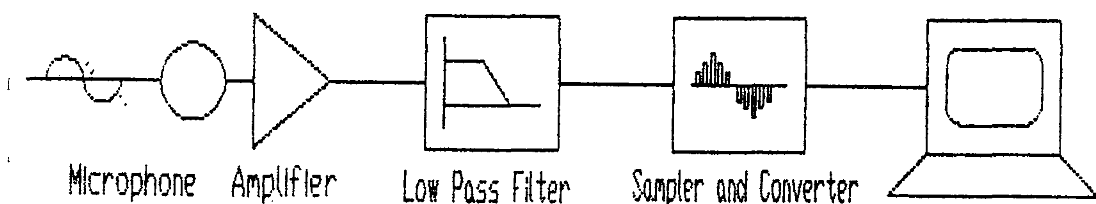


Figure 5.1 The Hardware model of the system

The analog input from the microphone is amplified and filtered at the second stage. The cutoff frequency of the low pass filter in this stage is 2048 Hz., which is half of the sampling frequency of the system. The amplified and the filtered signals swing around 0 volt level. This signal is the input of the third stage. There is an analog to digital converter on the interface card and the reference signals of the converter are 0 volt and 5 volts. In order to obtain maximum precision, the analog input has to swing around the 2.5 volts with peaks of 0 volt and 5 volts. The second stage output is biased with +2.5 volts with the bias circuitry on the interface card.

The analog-to-digital converter (ADC) of the system is a converter with 8 bits. Therefore signal is quantized into one of 256 ($=2^8$) different voltage levels. The quantization step size is 5/256 = 0.02 V. which is well enough for analyzing conventional musical signal. The data sheets of the ADC is given in Appendix C.

The interface card (see App. B) also has an address decoding logic and a three state buffer. The address of the output of the buffer is 0DFF (H). The output of the interface card is the digitized data and it is the input of the computer which is the fourth stage.

The IOW (Input-Output Write) signal of the computer is used to start the conversion, and the converted 8 bit data is the input of the three state buffer. With the active IOR (Input-Output Read) signal, the buffer is enabled and the data is read from address DFF $, and stored into the memory.

5.2 The Software Implementation

The program is written in Borland's Turbo C Ver.2. The listing of the program is given in Appendix A.

The procedure **sample_data** gets the digitized data from address 0DFF (H) memory location. The input parameters of the procedure are the amount of samples (**sampamount**) and the pointer(**sampptr**) of the memory location in which the data is stored. It is written in Inline Assembly Code and linked to the program. The delay constant **DELAY** provides 4096 Hz. sampling frequency, and it is assigned to 215 for this system. It is possible to increase or decrease the sampling frequency, by changing this constant. The amount of data to be sampled is determined by the procedure **Initialize**. The sampling time and the metronome number are the parameters of this procedure. The sampling time in seconds, is multiplied by a constant **WINDOW**, which is the size of a window, and eight times of the result determines the amount of the

sampled data.

The sampled data is stored in the memory and partitioned into windows. The 512 point data is taken as a window and transformed consecutively in the procedure **take_transform**. In this procedure Fast Hartley Transform routines are activated, and the transformed data is stored in a real data array **data_array**. The procedure **fht** takes the 512 point integer data array and transforms those data, and store them in a real data array. The sub procedures of the transform are **permute**, **trig_table**, **modify** and **butterfly**. The procedure **permute** organizes the ordering of the data at the first stage of the transform. The procedure **trig_table** determines the multiplying constants of the transform, and stores them into a real array. The procedure **butterfly** makes butterfly type array multiplication, which is used in fast transform algorithms. The procedure **modify** is used for array organization in the butterfly type data processing.

The routine **find_note** processes the transformed signal and extract the notes and the pitches of the related window. The note is converted to an integer value, in the range of 0 to 11, which corresponds to the notes from A to G#, and the pitch value is converted to a 10th power. By the addition of those two values a note can be represented by a long integer. For example 3rd

octave D is represented by the long integer 1005 as follows: D corresponds to 5, 3rd octave is 10^3=1000, 1000+5=1005. In order to extract the note name and the pitch value, the long integer which represents a note, is processed as follows: Take the integer logarithm of the data which represents the pitch value, and subtract the 10th power of the pitch from the data. For example, 1005 is given, (int) log10(1005) = 3 (pitch value), 1005-10^3=5 (note name). A faster scheme for note representation may be easily implemented for future real-time applications.

The routine **filter_transformed_signal** is used for finding the fundamental from its second and third harmonics. In this procedure, a real transformed data array is traced, and for each data point, its second and third multiples are examined. If those exceed the THRESHOLD (it is a global constant) then their amplitudes are added and assigned to the fundamental (See Algorithm 1, chapter 4). The THRESHOLD value is used to determine the excited frequencies.

The **find_freqs_without_harmonics** procedure discards the upper harmonics of a frequency in a window. The input of the procedure is an integer data array with a window size. This array is also used for the output of the procedure. The procedure, trace the whole data

array, and the point which exceeds the **THRESHOLD** level is regarded as an excited frequency. The integer multiples of the frequency is found and those are assigned to zero at the end (See Algorithm 2, chapter 4).

The output of the program is the score of the melody on the graphics display or on the printed output. The images for the music transcription are stored in the files "image.dat" and "note.dat" in binary format. In those files the 0's represent "pixel off", and the 1's represent "pixel on". The procedure **take_es_images** gets the images sixteenth, eighth, quarter and double rests from the file "image.dat" and stores them in the pointer variables s16, s8, s4, s2 respectively. Similarly the variables not16, not8, not4, not2 are assigned to the pointers for the sixteenth, eighth, quarter, and double notes respectively. The images for the notes are assigned in the routine **take_note_images**. The clef image is determined by a similar method.

The whole sampled data is processed with sliding windows and the excited notes are determined for each window. After obtaining the result of the processed data, the notes are stored in the file "notefile.dat". The procedure **musical_score** takes the notes from the file and processes them and transcripts the whole score on the monitor. The hard copy of the transcription can be get by

the routine print_graph. It is activated by pressing *Alt_P* keys.

In order to analyze the signal in time domain and also in the frequency domain, the procedures look_to_the_input_signal and spectrum_print are used. The WINDOW amount of data is shown on one frame. It is possible to analyze the consecutive windows. The routine look_to_the_input_signal shows the input sample data, and the routine spectrum_print shows the transformed and processed data.

Melody comparison is one of the features of this work. In order to recognize the correct melodic structure, we have to avoid the unwanted harmonics that appears because of abnormalities of the environment of the set-up, and the features of the musical instruments. The upper harmonics mislead the recognition environment. Because an harmonic instead of the correct note alters the form completely.

In order to avoid the misleading harmonics, some heuristics are proposed. The empirical results have shown that generally the harmonics appear as sixteenth notes, and they are the second, the third or the fourth harmonics of the preceding or succeeding notes. The procedure discard_harmonics finds those sixteenth notes

which are the harmonics of the preceding or the succeeding notes, and adds the values of those sixteenth notes to the corresponding fundamentals

The procedure **forming_structure** converts the note data in the form of the discrete sixteen notes to the three elements array which corresponds to the name of the note, duration and the pitch of the note. The procedure **transpose** uses those type of data and find the interval codes of the notes which are the elements of the musical form.

After the **transpose** procedure, the intervals and the duration of the related melodic structure are divided into two files. One of the file contains just the interval code of notes, and the other file contains the corresponding temporal code of the melodic form.

The procedure **get_note_matrix** gets the difference codes of two melodies and put those values to first row and first column of a matrix. After finding the absolute squared differences for each note in one melody with the other melody, it forms a complete matrix. The diagonals of the matrix give the distance of the two melody. The procedure **find_correlation** take this matrix and calculates the distance of the two melodic pattern by looking to the diagonals.

The same procedure **find_correlation** is used to find the distance of the melodic structure from the duration point of view. The procedure **get_rest_matrix** forms the matrix of the relevant durations of the intervals of the notes. The same structure as **get_note_matrix** is used in this procedure. However there are also inputs **mul_row** and **mul_col** for this procedure. **mul_row** is the multiplying factor of the durations of one melody and the **mul_col** is the multiplication factor for the other melody. Those multiplying factors are used to multiply the durations of the melodic structure in order to find the temporal warping of the melodies. Those variables are used in the procedure **mul_corr** by factors 0.5, 1, 1.5, 2. With those multiplying constants, the distance of the durations are calculated. The smallest value of sum of the square of distances is recorded and the multiplication factors for this distance is found as the constants for temporal warping.

CHAPTER 6

RESULTS AND DISCUSSION

The experiments were carried out on an IBM Compatible Personal Computer, with 80286 CPU and 80287 Math Coprocessor. The processing time of the melody changes with the duration and the tempo of the melodic pattern. The transformation of one window takes about 1.1 seconds. Since one window represents a sixteenth note, the amount of sixteenth notes in the melody determines the duration of the processing time. For example, the processing time of a melody with the tempo of 80 bpm. and 15 seconds duration, is about 85 seconds. Transcription takes fairly less time than transformation.

With 640 KB main memory, it is possible to sample and process about 2 minutes of music. This is considered long enough to determine a melodic form properly.

Most of the experiments have been done with the flute and the clarinet. With those instruments, music pieces have been played and sampled. After processing those data, the transcription of the pieces has been

obtained. The results of those experiments were quite satisfactory. At the end of the experiments, almost identical scores with the real partitions have been retrieved. In those experiments it is observed that, the tempo given to the computer and the tempo of the melody itself should match in order to get correct results. Sometimes, undesirable effects have been observed due to the unmatching of the tempos.

Figure 5.1 shows the score of the Second Part of J.S.Bach's Flute Sonata No 4. Figure 5.2 shows the transcription of the same sonata with the system. In this example, 15 seconds of data have been taken and processed. The tempo of the melody is 100 bpm. It is observed that quite successful transcription of the melody has been achieved.

Some experiments have also been done with the violin and the guitar. It is observed that the timbre of the instrument has effects on the output. The results of those experiments are not as successful as the results obtained from the clarinet and the flute. The reason for this is that the energies of the guitar and the violin are scattered through the whole frequency spectrum.

Sometimes in those instruments, only the second or the third harmonics have been excited while no energy

Figure 6.1 The score of the J.S.Bach's No 4 Flute Sonata Part 2 Allegro.

Figure 6.2 The Transcription of J.S.Bach's 4th Flute Sonata. The melody has been played by the flute.

has been found in the fundamental. For those cases it has not been possible to grasp the fundamental. But even in those experiments, the results that were obtained can give an indication about the basic nature of the melodic form.

Another experiment has been carried out with an unsustained type instrument (in which the energy of the instrument decreases after stimulation e.g. piano, guitar, etc.). To simulate the piano characteristic, a synthesizer in piano mode has been used, and the melody which is stored in the synthesizer's memory has been played. After the transcription of the melody has been obtained, the score has been played with another instrument, or the synthesizer itself. It has been observed that the result was the correct score of the melody. In Figure 6.3 the result of the process is shown. The tempo of the melody is 90 bpm.

Some experiments have been carried out to determine the effects of the tempo of the melodic forms. With different tempo values, different melodies have been processed and it is found that the correct results have been obtained whether the given melody is Adagio or Allegro. But if the given rate is changed while playing, undesirable results are obtained.

Figure 6.3 The Transcription of the melody of the synthesizer's memory. The melody has been played by the synthesizer.

In Figure 6.4 the score of a Telemann's piece is given.The tempo of the piece is Adagio. In order to see the effects of the tempo rates, the melody has been played again on the flute and the transcription has been done by the system. In Figure 6.5 the result of the transcribed melody is given. It is again observed that, the same melodic form has been obtained.

The Figure 6.6 and Figure 6.7, show an original Haendel's piece and its transcription with the system, respectively. The melody has been played with the clarinet. The tempo of the melody is 60 bpm. When the

6. AN A

für Querflöte ohne Baß, d-moll

Figure 6.4 The score of Telemann's Fantasie No 6

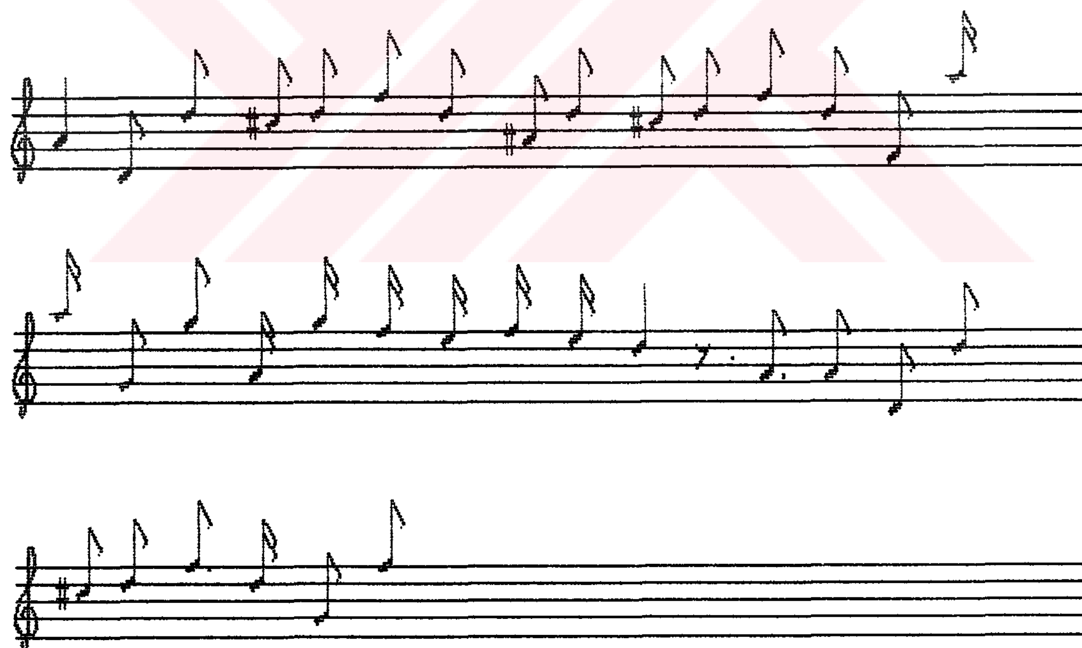


Figure 6.5 The transcription of Telemann's Fantasie by the system. The instrument is flute.

Figure 6.6 The score of Haendel's piece.

Figure 6.7 The transcription of the Haendel's piece. The melody has been played by clarinet

result is matched, it is drawn that the correct result have been obtained.

Melody recognition has also been achieved by the comparison method. As it is explained in chapter 4, two melodies are compared and the distance measure between their note and duration values are found. The results of the experiments have shown that it is possible to distinguish the two melodic forms. In Figure 6.8 two similar melodies are given and the distance between them is found. Also the distance between two unsimilar melodies in Figure 6.9 have been compared and their distance matrix is formed and their distance measure is obtained. It is observed that the sum of the squared distance values of two similar melodies is much less than the unsimilar melodies.

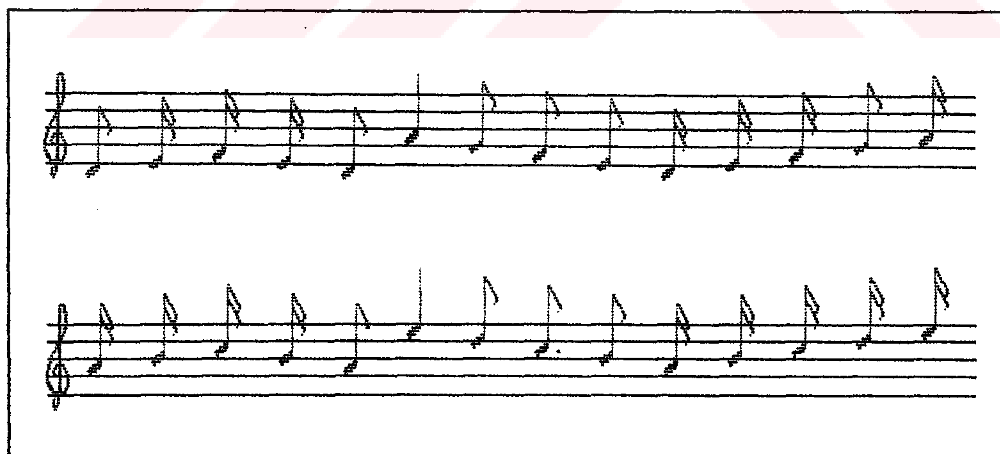


Figure 6.8 Two similar melodies for melody comparison

The interval distance of the melodies : 0

The duration distance of the melodies : 3

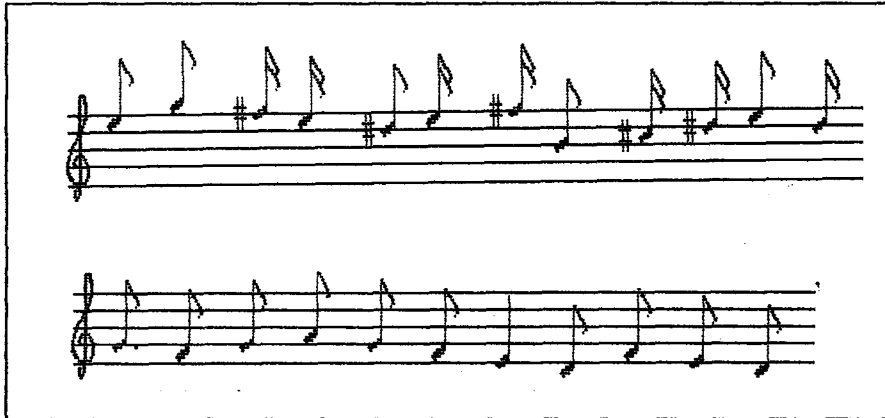


Figure 6.9 Two unsimilar melodies for melody comparison

The interval distance of the melodies : 206,93,88

This work is one of the steps on the way of four part music transcription. It is also planned in future that a melody will be harmonized in predetermined form. The melody recognition step in this work is an important future to define a melodic form.

The prototype designed in this work can be developed and used in the music education. The children can easily make contact between sound and the note with this developed computer aided musical environment, and acquire a musical background by themselves. It can be also an opportunity to grasp the musical intelligence of a child in the early ages, with the improved form of this

designed work.

In summary, although some undeterministic problems occur due to the timberic characteristics of the instruments, the results of the work seem to be quite satisfactory for one part music transcription and melody recognition. The solution of the transposition problem and the proposed heuristics can be used in the planned four part music transcription and harmonization.

REFERENCES

Ahmed, N., Rao, K.R., 1975. Orthogonal Transforms for Digital Signal Processing, Springer-Verlag, Berlin, Heidelberg, New York.

Bracewell, Ronald, N., 1984. "The Fast Hartley Transform", Proceedings of IEEE, Vol. 72, No 8, August, pp. 1010-1018.

Cooley, J., W., Tukey, J., W., 1965. "An Algorithm for the Machine Computation of Complex Fourier Series", Math. Computation, Vol. 19, April, pp 297-301

Donington, Robert, 1982. Music and its Instruments, Metheuen & Co. Ltd., New York.

Ebcioğlu, Kemal, 1979. "Strict Counterpoint: A Case Study in Musical Composition by Computers", M. S., Thesis, Department of Computer Engineering Ankara: M.E.T.U.

Ebcioğlu, Kemal, 1988. " An Expert System for Harmonizing Four-part Chorales", Computer Music Journal, Vol. 12, No 3, pp. 43-51.

Gualtieri, D.M, 1986. "MIDI Output Interface to a Parallel Printer Port", Computer Music Journal, Vol. 10, No 3, Fall, pp 79-82.

Helmholtz, Hermann, L.F., 1954. On the Sensation of Tone, Dover Publications, Inc., New York.

İzmirli, Özgür, 1988, "Computer Analysis of Musical Sound" M.S. Thesis, Department of E. Electronics Engineering Ankara: M.E.T.U.

Larousse, 1984. Encyclopedia of Music, Hamyln.

Mackay, Andy, 1981. Electronic Music, Phaidon, Oxford.

Moorer, J., Anderson, 1977. "Signal Processing Aspects of Computer Music: A Survey", Proceedings of IEEE, August, pp 1108-1137.

Roads, Curtis, 1985. "Research in Music and Artificial Intelligence", ACM Computing Surveys, Vol. 17, No 2, pp 163-190.

Yavelow, Christopher, 1987. "A Report on the WorkShop for Music Notation by Computer", Computer Music Journal, Vol. 11, No 2, Summer, pp 65-70.

APPENDICES

APPENDIX A

SOFTWARE SOURCE LISTING

```
#include <alloc.h>
#include <math.h>
#include <dos.h>
#include <fcntl.h>
#include <stdio.h>
#include <graphics.h>
#include <time.h>
#include <alloc.h>
#include <stdlib.h>
#include <bios.h>
#include <conio.h>

#include "graph3.c"
#include "eminlib.c"
#include "dfnition.c"
#include "correlat.c"

typedef float data_array_type[WINDOW+1];

typedef struct tempstrc {
                              int amplitude;
                              int frequency;
                 };


typedef struct
   {
      int note_name; /* for rest note_name = 20 */
      int pitch;     /* for rest pitch = 0 */
      int duration;
   } formed_note;

data_array_type data_array;
float   n[MAXCOUNT+3];
char    far *sampptr;
char    dir,test_option;
int     i,j,size,iter,demo;
FILE    *f;
FILE    *o;
int     DELAY;
FILE    *notefile;
float   speclapoint;  /* The Point where the 440 A will be, on the spectrum */
void    *not2,*not4,*not8,*not16,*keysol,*keyfa;
void    *s2,*s4,*s8,*s16,*diez,*connection;
unsigned long sample_amount;
int t_a, f_a;
data_array_type sne, csn;
float accu[4][WINDOW+1];
int power_index,syze;
int metronom;
int THRESOLD;
unsigned long sample_processed;
float sec;
char far* scr_ptr;
char far* scr_type;


/*---------------FHT Procedures--------------*/

permute(int index)
{
   int i, j, s;

   j = 0;
   index--;
   for(i=1; i<=power_index; i++)
   {
      s = index / 2;
```

```
      j = 2 * j + index - 2 * s;
      index = s;
   }
   return(j + 1);
}

void trig_table(int nputs)
{
   int i;
   float angle, omega;

   angle = 0;
   omega = 2 * PI / nputs;
   for(i=1; i<=nputs; i++)
   {
      sne[i] = sin(angle);
      csn[i] = cos(angle);
      angle += omega;
   }
}


modify(int power, int s_start, int s_end, int index)
{
   if((s_start == index) || (power < 3))
   return(index);
   else
   return(s_start + s_end - index + 1);
}


void butterfly (int trig_ind, int i_1, int i_2, int i_3)
{
   accu[t_a][i_1] = accu[f_a][i_1] +
           accu[f_a][i_2] * csn[trig_ind] +
           accu[f_a][i_3] * sne[trig_ind];
   trig_ind += syze / 2;
   accu[t_a][i_2] = accu[f_a][i_1] +
           accu[f_a][i_2] * csn[trig_ind] +
           accu[f_a][i_3] * sne[trig_ind];
}


void fht (data_array_type data_array, int power_index,
       int syze, char transform)
{
    int i,j,k,trg_ind,trg_inc,power,
     i_temp,section,s_start,s_end;

    power = 1;
    f_a = 1;
    t_a = 2;
    trig_table(syze);
    for (i=1; i<=syze; i++)
    accu[f_a][permute(i)] = data_array[i];
    for (i=1; i<=power_index; i++)
    {
      j = 1;
      section = 1;
      trg_inc = syze / (2 * power);
      do
      {
       trg_ind = 1;
       s_start = j + power;
       s_end = (section + 1) * power;
       for (k=1; k<=power; k++)
       {
         butterfly(trg_ind, j, j+power, modify(power, s_start, s_end,
```

```
                                    j + power));
        trg_ind += trg_inc;
        j++;
      }
      j += power;
      section += 2;
   }
   while ( j <= syze);
   power *= 2;
   i_temp = t_a;
   t_a = f_a;
   f_a = i_temp;
  }
  if (transform == 0)
  {
     i = 1;
     do
     {
       data_array[i] = 2 * accu[f_a][i] / syze;
       i++;
     }
     while (i<=syze);
  }
  else
  {
      for(i=1; i<= syze; i++)
      data_array[i] = accu[f_a][i] / 20 ;
  }
}

/*------------------End Of FHT Procedures------------------------*/

int determine_thresold(int in_array[])
{
   int count;
   int thresold = 0, num_of_data = 0;


   for(count = 15; count < WINDOW / 2; count++)
      if(in_array [count] > 10)
      {
         thresold += in_array[count];
         num_of_data += 1;
      }
   if (num_of_data == 0)
      return(50);
   (int) thresold /= num_of_data;
   if (thresold < 20)
      thresold = 30;
   return (thresold);
}

/******************************************************************/

int look_char()
{
   char ch;

   ch = getch();
   if (ch == 0)
   {
      ch = getch();
      switch (ch)
      {
         case (25) : return(1);
         default : return(0);
      }
   }
```

```
   else
   if (ch == 0x1b)
      return (2);
   else return(0);
}

/****************************************************************/

void print_graph()
{
   int n, n1, n2, tt, i, mx, my;
   char prbyte;

   mx = getmaxx();
   my = getmaxy();
   n1 = (mx+121) % 256;
   n2 = (int)(mx+121) / 256;
   bdos(0x0005,0x001b,0);
   bdos(0x0005,0x0033,0);
   bdos(0x0005,0x0016,0);
   for (n = 0; n <= ((int)(my / 8) - 1); n++)
   {
      bdos(0x0005,0x001b,0);
      bdos(0x0005,0x004c,0);
      bdos(0x0005,(char)n1,0);
      bdos(0x0005,(char)n2,0);
      for (tt = 1; tt <= 120; tt++)
         bdos(0x0005,0x0000,0);
      for (tt = 0; tt <= mx; tt++)
      {
         prbyte = 0;
         for (i = 0; i <= 7; i++)
            if (getpixel(tt, n*8+i) != 0)
               prbyte += (1 << (7-i));
         bdos(0x0005,prbyte,0);
      }
      for (tt = 0; tt <= 40; tt++)
         bdos(0x0005,0x0000,0);
      bdos(0x0005,0x000a,0);
      bdos(0x0005,0x000d,0);
   }
   bdos(0x0005,0x001b,0);
   bdos(0x0005,0x004c,0);
   bdos(0x0005,(char)n1,0);
   bdos(0x0005,(char)n2,0);
   for (tt = 1; tt <= 120; tt++)
         bdos(0x0005,0x0000,0);
   for (tt = 0; tt <= mx; tt++)
   {
      prbyte = 0;
      for (i = 0; i <= (my % 8); i++)
         if (getpixel(tt,((int)(my / 8)*8+i) != 0))
            prbyte += (1 << (7-i));
      bdos(0x0005,prbyte,0);
   }
   for (tt = 1; tt <= 40; tt++)
      bdos(0x0005,0x0000,0);
   bdos(0x0005,0x000a,0);
   bdos(0x0005,0x000d,0);
}

/****************************************************************/

char *CreatMem (unsigned long MEMSIZE)
                         /* Creates memory for defined size
                              to use put the samples in        */
```

```
{
   char far *infunc;
   infunc = (char *) farcalloc(MEMSIZE, sizeof(char));
   if (infunc == NULL)
   {
      printf("\nfailed to creat a new allocated block! Now HALTED");
       exit(1);
   }
   return(infunc);
}

/*****************************************************************/

void getdelay()
{
   DELAY = 215;
}

/*****************************************************************/

void draw_input_signal(int gr_array[], int array_length)
{
   int x_point, y_point,dat_count;
   char * txtstr;

   txtstr = (char *) malloc(20);
   line(0, 0, 0, getmaxy());
   line(0, (getmaxy() / 2), getmaxx(), getmaxy() / 2);
     strcpy(txtstr," Amplitude");
   settextstyle(SMALL_FONT,VERT_DIR,6);
   outtextxy(0,60,txtstr);
   settextstyle(SMALL_FONT, 0, 7);
   strcpy(txtstr," Time");
   outtextxy(300,300,txtstr);
   settextstyle(SMALL_FONT, 0, 4);
   moveto(0,getmaxy() / 2);
   for(dat_count = 0; dat_count < array_length / 2; dat_count++)
   {
      y_point = (int) ((getmaxy() / 2) - 75 + gr_array[dat_count * 2]);
      x_point = (int) ((dat_count * 2) * (720.0/512.0));
      lineto(x_point, y_point);
      moveto(x_point, y_point);
   }
}

/*****************************************************************/

char * findname(long int innote)
{
   char * notename;
   int  notecount, pitch;
   char pitchname[2];

   notename = (char *) malloc(10);
   pitch = log10(innote);
   switch (innote - pow10(pitch))
   {
     case 0  : notename = "A  "   ; break;
     case 1  : notename = "A# "   ; break;
     case 2  : notename = "B  "   ; break;
     case 3  : notename = "C  "   ; break;
     case 4  : notename = "C# "   ; break;
     case 5  : notename = "D  "   ; break;
     case 6  : notename = "D# "   ; break;
     case 7  : notename = "E  "   ; break;
     case 8  : notename = "F  "   ; break;
     case 9  : notename = "F# "   ; break;
     case 10 : notename = "G  "   ; break;
```

```
      case 11 : notename = "G# "   ; break;
      case 12 : {
                notename = "A  ";
                pitch++;
                break;
                }
   }
   itoa(pitch,pitchname,10);
   notename = cat(notename,pitchname);
   return(notename);
}

/******************************************************************/

long findnote(long innote)
{
   int notecount, pitch;
   long outnote;

   n[12] = 110;
   n[0] = 55.0; n[1] = 58.3; n[2] = 61.7; n[3] = 65.4; n[4] = 69.3;
   n[5] = 73.4; n[6] = 77.8; n[7] = 82.4; n[8] = 87.3; n[9] = 92.5;
   n[10]= 98.0;
   n[11]= 103.8;


   for(pitch=0; pitch < 6; pitch++)
     if (innote < A1 * pow(2,pitch))
     break;
   pitch--;
   for(notecount = 0; notecount < MAXCOUNT+1; notecount++)
      if ((innote >= (n[notecount] * pow(2,pitch))) &&
          (innote < (n[notecount+1] * pow(2,pitch))))
          break;

      if (innote > (n[notecount] * pow(2,pitch) +
       n[notecount+1] * pow(2,pitch)) / 2)
          outnote = ++notecount + pow10(++pitch);
       else
          outnote = notecount + pow10(++pitch);
       if (notecount == 12)
          outnote = pow10(++pitch);
       return(outnote);
}


/******************************************************************/

sort(struct tempstrc inplate[], int voicenum)
{
   int count1, count2, temp1, temp2;

   if(voicenum < 2)
      return(0);
   for(count1 = 0; count1 < voicenum; count1++)
      for(count2 = 0; count2 < voicenum; count2++)
         if(inplate[count1].amplitude > inplate[count2].amplitude)
         {
            temp1 = inplate[count1].amplitude;
            temp2 = inplate[count1].frequency;
            inplate[count1].amplitude = inplate[count2].amplitude;
            inplate[count1].frequency = inplate[count2].frequency;
            inplate[count2].amplitude = temp1;
            inplate[count2].frequency = temp2;
         }
   return(0);
}
```

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

void findfreqs(int x_axis,int numofvoices,
               int notearray[], long result_array[])
/*    Finds the maximum frequencies given the multiple voices   */

{
   struct tempstrc temparray[4]; /* maximum 4 voices */
   int template[5];
   int count, notecount;

   for(count = 0; count < numofvoices; count++)
   {
      temparray[count].amplitude = 0;
      temparray[count].frequency = 0;
   }
   for(count = 6; count < x_axis / 2; count++)
   {
      for(notecount = 4; notecount >= 0; notecount--)
      {
         template[notecount] = abs(notearray[count - ( 4 - notecount)]);
      }
      if ((template[2] > template[0]) && (template[2] > template[4])
       && (template[2] > template[3]) && (template[2] > template[1]))
       {
         if (template[2] > temparray[numofvoices-1].amplitude)
         {
            temparray[numofvoices-1].amplitude = template[2];
            temparray[numofvoices-1].frequency = count - 2;
            sort(temparray, numofvoices);
         }
      }
   }
   for(count = 0; count < numofvoices; count++)
      result_array[count] =(long) temparray[count].frequency;
}

/*****************************************************************/

void findfreqs_without_harmonics(int x_axis,int numofvoices,
               int notearray[], long result_array[])
/*    Finds the maximum frequencies given the multiple voices   */

{
   struct tempstrc temparray[4]; /* maximum 4 voices */
   int template[5];
   int count, notecount, discard, clear_points;

   THRESOLD = determine_thresold(notearray);
   for(count = 0; count < numofvoices; count++)
   {
      temparray[count].amplitude = 0;
      temparray[count].frequency = 0;
   }
   for(count = 16; count < x_axis / 2; count++)
   {
      for(discard = 2 * count; discard < x_axis / 2; discard += count)
         for (clear_points = -3; clear_points <= 3; clear_points++)
         {
            if (notearray[count] >= notearray[discard + clear_points])
               notearray[discard + clear_points] = 0;
            if (notearray[count] >= THRESOLD)
               notearray[discard+clear_points] = 0;
         }
      for(notecount = 4; notecount >= 0; notecount--)
      {
         template[notecount] = abs(notearray[count - ( 4 - notecount)]);
      }
```

```
      if (template[2] >= THRESOLD)
      {
         if ((template[2] > template[0]) && (template[2] > template[4])
          && (template[2] >= template[3]) && (template[2] >=  template[1]))
          {
            if (template[2] > temparray[numofvoices-1].amplitude)
            {
               temparray[numofvoices-1].amplitude = template[2];
               temparray[numofvoices-1].frequency = count - 2;
               sort(temparray, numofvoices);
               if (temparray[1].amplitude > temparray[0].amplitude)
               {
               temparray[1].amplitude *= 2;
               sort(temparray, numofvoices);
               }
            }
         }
      }
   }
   for(count = 0; count < numofvoices; count++)
      result_array[count] =(long) temparray[count].frequency;
}

/******************************************************************/

void take_transform(int tr_type)
{
   FILE *trfile;
   unsigned long count, size, windsize, tr_amount;
   int dummy;
   int step,remain_time;

   step = (int) ((sample_amount*WINDOW / sample_processed) - WINDOW);
   if((trfile = fopen("trnsfrm.dat", "w+b")) == NULL)
   {
      printf("TRNSFRM.DAT file couldn't be created");
      exit(0);
   }
   f = fopen("sample.dat","rb");
   sampptr = CreatMem(sample_amount * WINDOW);
   fread(sampptr, sizeof(char), sample_amount * WINDOW, f);
   clrscr();
   cursor_off();
   draw_rect(22,9,58,16,2);
   textcolor(CYAN);
   for (remain_time = 0; remain_time++ < 40 ;)
   {
      gotoxy(20+remain_time,19);
      putch(0xDB);
   }
   textcolor(LIGHTGRAY);
   gotoxy(32,21);
   printf("Processing Time");
   for(tr_amount = 0; tr_amount < sample_processed - tr_type; tr_amount++)
   {
      for(windsize = 0; windsize < WINDOW; windsize++)
         data_array[windsize+1] = *(sampptr + tr_amount * WINDOW
         + windsize + (tr_type * WINDOW / 2) + tr_amount*(step) + 256);
      gotoxy(26,12);
      printf("Window %d is being Prosessed" ,tr_amount + 1);
      remain_time = (int) (tr_amount+1) * 40 / sample_processed;
      highvideo();
      textattr(BLUE+(CYAN<<4));
      for(dummy = 0; dummy < remain_time; dummy++)
      {
         gotoxy(20+remain_time - dummy,19);
         putch(0xC4);
      }
```

```
         textattr(LIGHTGRAY+(BLACK<<4));
      lowvideo();
      fht(data_array, power_index, syze, reverse);
      for (count = 0; count < WINDOW; count++)
      {
         dummy = (int) abs(data_array[count]);
         fwrite(&dummy, sizeof(dummy),1, trfile);
      }
   }
   cursor_on();
   farfree(sampptr);
   fclose(f);
   fclose(trfile);
}

/*****************************************************************/

void filter_transformed_signal(int x_axis, int notearray[])
{
   int fundamental;
   int discard, ok=1;
   int template[5], notecount, clear_points, third_point;

   THRESOLD = determine_thresold(notearray);
   for(fundamental = 16; fundamental * 3 + 3 <= x_axis / 2; fundamental++)
   {
      for(notecount = 4; notecount >= 0; notecount--)
         template[notecount] = abs(notearray[fundamental - ( 2 - notecount)]);
      if (template[2] >= THRESOLD)
      {
         if ((template[2] > template[0]) && (template[2] > template[4])
          && (template[2] >= template[3]) && (template[2] >=  template[1]))
         {
            ok = 1;
            for (clear_points = -3; ((clear_points <= 2) && ok); clear_points++)
               if (notearray[fundamental*2 + clear_points] >= THRESOLD)
               for (third_point = -3 ; ((third_point <=2) && ok); third_point++)
                     if(notearray[fundamental*3 + third_point] >= THRESOLD)
                     {
                           notearray[fundamental] =
notearray[fundamental*2+clear_points]+

    notearray[fundamental*3+third_point];
                           ok = 0;
                     }
         }
      }
   }
}

/*****************************************************************/

sort_note(int vnum, long notearray[])
{
   int count1, count2;
   long temp;

   if (vnum < 2)
      return(0);
   for (count1 = 0; count1 < vnum; count1++)
      for (count2 = count1; count2 < vnum; count2++)
         if (notearray[count1] <  notearray[count2])
         {
            temp = notearray[count1];
            notearray[count1] = notearray[count2];
            notearray[count2] = temp;
         }
   return(0);
```

```
}

/*****************************************************************/

void dump_note(int trtype, int vnum)
{
   int trarray[WINDOW], count, count2;
   long looknote[4];
   int control;

   o = fopen("notefile.dat","wb+");
   if (trtype == 1)
      f = fopen("trnsfrm.dat","rb");
   else
      f = fopen("trnsfrm2.dat","rb");
   for(count = 0; count < sample_processed; count++)
   {
      fread(trarray, sizeof(trarray), 1, f);
      findfreqs_without_harmonics(WINDOW, vnum, trarray, looknote);
      for(count2 = 0; count2 < vnum; count2++)
      {
         looknote[count2] = (int)(looknote[count2] * 440.0 / speclapoint);
         looknote[count2] = findnote(looknote[count2]);
      }
      sort_note(vnum,looknote);
      fwrite(looknote, sizeof(looknote), 1, o);
   }
   fclose(o);
   fclose(f);
 }

/*****************************************************************/

void discard_harmonics()
{
   long  prenote[4], middlenote[4], postnote[4];
   int   count;
   FILE  *out;
   int   prefreq, middlefreq, postfreq, prepitch, middlepitch, postpitch;

   f = fopen("notefile.dat", "rb");
   out = fopen("notalar.dat","wb+");
   fread(&prenote, sizeof(prenote), 1, f);
   fwrite(&prenote, sizeof(prenote), 1, out);
   fread(&middlenote, sizeof(prenote), 1, f);
   for(count = 2; count < sample_processed; count++)
   {
      fread(&postnote, sizeof(prenote), 1, f);
      if((middlenote[0] != postnote[0]) && middlenote[0] != prenote[0])
      {
         prepitch    = (int)log10(prenote[0]);
         middlepitch = (int)log10(middlenote[0]);
         postpitch   = (int)log10(postnote[0]);
         prefreq     = prenote[0] - pow10(prepitch);
         middlefreq  = middlenote[0] - pow10(middlepitch);
         postfreq    = postnote[0] - pow10(postpitch);
         if (((prefreq+7) % 12) == middlefreq)
            if(((prefreq >= 5) && (middlepitch == (prepitch+2))) ||
               ((prefreq <  5) && (middlepitch == (prepitch+1))))
               *middlenote = *prenote;
         if (((postfreq+7) % 12) == middlefreq)
            if(((postfreq >= 5) && (middlepitch == (postpitch+2))) ||
               ((postfreq <  5) && (middlepitch == (postpitch+1))))
               *middlenote = *postnote;
      }
      fwrite(&middlenote, sizeof(middlenote), 1, out);
      *prenote = *middlenote;
      *middlenote = *postnote;
```

```
    }
    fwrite(&postnote, sizeof(middlenote), 1, out);
    fclose(f);
    fclose(out);
}

/*****************************************************************/

void heuristic1()
{
    int trarray1[WINDOW], trarray2[WINDOW], count, count2;

    o  = fopen("trnsfrm.dat","rb");
    f = fopen("trnsfrm2.dat","wb");
    fread(trarray1, sizeof(trarray1), 1, o);
    for (count = 0; count < sample_processed; count++)
    {
        fread(trarray2, sizeof(trarray1), 1, o);
        for(count2 = 0; count2 < WINDOW; count2++)
            trarray1[count2] = (trarray1[count2] + trarray2[count2]) / 2;
        fwrite(trarray1, sizeof(trarray1), 1, f);
    }
    fwrite(trarray2, sizeof(trarray1), 1, f);
    fclose(o);
    fclose(f);
}

/*****************************************************************/

void spctr_print(int vnum, int heurnum, int sptr)
{
    int trarray[WINDOW], count = 0;
    int ch = 0;

    graphinit();
    if(heurnum  == 0)
        o = fopen("trnsfrm.dat", "rb");
    else
        o = fopen("trnsfrm2.dat", "rb");
    while((count++ < sample_processed) && (ch != 2))
    {
        fread(trarray, sizeof(trarray), 1, o);
        filter_transformed_signal(WINDOW,trarray);
        draw_spctr(WINDOW, trarray, vnum, sptr);
        ch = look_char();
        if (ch == 1)
            print_graph();
    }
    fclose(o);
    closegraph();
}

/*****************************************************************/

void take_samples()
     /*  Asks the sample size and allocates memory,
         get the samples and put them in.           */
{
    unsigned long samplesize;
    char ch;            /*extern char far *sampptr;*/

    f = fopen("sample.dat","wb");
    clrscr();
    cursor_off();
    draw_rect(17,9,63,16,3);
    gotoxy(18,12);
    printf("      Press any key to sample data !");
    getch();
```

```
   sampptr = CreatMem(sample_amount * WINDOW);
   putch('\007');
   sample_data(sampptr,sample_amount * WINDOW);
   putch('\007');
   cursor_on();
   fwrite(sampptr,sizeof(char),sample_amount*WINDOW,f);
   farfree(sampptr);
   fclose(f);
}

/******************************************************************/

readinputs()
{
   int count;
   int dummy;
   if ((f = fopen("dataout","rb")) == NULL)
   {
      printf(" Cannot open input file.\n");
      return(1);
   }
   clrscr();
   for(count=1; count<= WINDOW; count++)
   {
      fread(&dummy,sizeof(dummy),1,f);
      data_array[count] = dummy;
      printf("%d ",dummy);
      if (feof(f))
      printf("EOF has reached");
   }
   return(0);
}

/******************************************************************/

int find_duration(int notedur)
{
   int duration;

   switch (notedur)
   {
      case 1 : duration = 16; break;
      case 2 :
      case 3 : duration = 8 ; break;
      case 4 :
      case 5 :
      case 6 :
      case 7 : duration = 4 ; break;
      case 8 :
      case 9 :
      case 10:
      case 11:
      case 12:
      case 13:
      case 14:
      case 15:
      case 16: duration = 2 ; break;

      default : duration = 2; break;
      }
   return (int) duration;
}

/******************************************************************/

void musical_score(int vnum)
{
   long notearray1[4], notearray2[4];
```

```
int lineno, column_pos, printnotecontrol, duration[4];
int count, count2;
int pitch, note;
int noteduration;
int bps;
int sec_duration;
int static rest_control;

graphinit();
rest_control = 0;
porte();

for (count = 0; count < vnum; count++)
   duration[count] = 1;
printnotecontrol = 0;
lineno = 1; /* porte no */
column_pos = 30;
f = fopen("notalar.dat", "rb");
fread(notearray1, sizeof(notearray1), vnum, f);
for (count = 1; count < sample_processed; count++)
{
   fread(notearray2, sizeof(notearray2), vnum, f);
   for(count2 = 0; count2 < vnum; count2++)
   {
      if (notearray2[count2] == notearray1[count2])
         duration[count2] ++;
      else
      {
         pitch = log10(notearray1[count2]);
         note = notearray1[count2] - pow10(pitch);
         if ((int)log10(notearray2[count2]) == 1)
            duration[count2] += 1;
         if (pitch == 1)
         {
            if (!rest_control)
            rest_control = 1;
            else
            if (duration[count2] > 1)
            {
            print_rest(duration[count2],column_pos,lineno);
            printnotecontrol = 1;
            }
         }
         else
         {
            if (duration[count2] ==10)
            {
            print_note(note, pitch, 8, column_pos, lineno);
            column_pos += 30;
            print_note(note, pitch, 2, column_pos, lineno);
            }
            else
            if (duration[count2] == 14)
            {
            print_note(note, pitch, 8, column_pos, lineno);
            column_pos += 30;
            print_note(note, pitch, 6, column_pos, lineno);
            }
            else
            if (duration[count2] > 17)
            {
            sec_duration = duration[count2] - 16;
            print_note(note, pitch, 16, column_pos, lineno);
            column_pos += 30;
            print_note(note, pitch, sec_duration, column_pos, lineno);
            }
            else
            print_note(note, pitch, duration[count2], column_pos, lineno);
```

```
                printnotecontrol = 1;
                rest_control = 1;
             }
             notearray1[count2] = notearray2[count2];
             duration[count2] = 1;
          }
       }
       if (printnotecontrol)
       {
          printnotecontrol = 0;
          column_pos += 40;
          if(column_pos  > 600)
          {
             lineno++;
             column_pos = 30;
          }
       }
    }
    if (look_char() == 1)
       print_graph();
    closegraph();
    fclose(f);
}

/*****************************************************************/

void look_to_the_input_signal()
{
    int wind_amount = 0, count;
    int draw_array[WINDOW];
    int ch = 0;

    f = fopen("sample.dat","rb");
    sampptr = CreatMem(sample_amount * WINDOW);
    fread(sampptr, sizeof(char), sample_amount * WINDOW, f);
    graphinit();

    while((wind_amount++ < sample_processed) && (ch != 2))
    {
       for(count = 0; count < WINDOW; count++)
          draw_array[count] = sampptr[WINDOW * wind_amount + count];
       draw_input_signal(draw_array, WINDOW);
       ch = look_char();
       if (ch == 1)
          print_graph();
       clearviewport();
    }
    farfree(sampptr);
    fclose(f);
    closegraph();
}


/*****************************************************************/

void forming_structure()
{
    formed_note individual_note;
    FILE        *formed_fp;
    long        notearray1[4], notearray2[4];
    int         count1,count2;
    int         static first_rest;
    int         note_dur;

    clrscr();
    f = fopen("notalar.dat","rb");
    formed_fp = fopen("melody.dat", "wba");
    note_dur = 1;
```

```
   first_rest = 0;
   fread(notearray1, sizeof(notearray1), 1, f);
   for(count1 = 1; count1 < sample_processed; count1++)
   {
      fread(notearray2, sizeof(notearray1), 1, f);
      if (notearray2[0] == notearray1[0])
         note_dur++;
      else
      {
         individual_note.pitch = log10(notearray1[0]);
         individual_note.note_name = notearray1[0] -
                                 pow10(individual_note.pitch);
         individual_note.duration = note_dur;
         if (individual_note.pitch == 1)
         {
            if(!first_rest)
            {
               first_rest = 1;
               note_dur = 1;
            }
            else
            {
               individual_note.pitch = 0;
               individual_note.note_name = 20;
               fwrite(&individual_note, sizeof(individual_note), 1,
                     formed_fp);
               note_dur = 1;
            }
         }
         else {
            fwrite(&individual_note, sizeof(individual_note), 1,
                     formed_fp);
            note_dur = 1;
            first_rest = 1;
         }
      }
      notearray1[0] = notearray2[0];
   }
   fclose (f);
   fclose(formed_fp);
   formed_fp = fopen("melody.dat", "rb");
   while (!feof(formed_fp)) {
      fread(&individual_note, sizeof(individual_note), 1, formed_fp);
      printf("\n%d %d %d", individual_note.note_name,individual_note.pitch,
                        individual_note.duration);
   }
   fclose(formed_fp);
}

/*****************************************************************/

void transpose(char *output_file, char *restfile)
{
   typedef struct {
           int gap;
           int duration;
   } trans_note;

   formed_note pre_note, post_note, es_note, dummy;
   FILE        *f_tr, *tr_out, *rest;
   trans_note  note_tr;
   int         es_found;
   int         count, *esvalue, *value0;
   int         file_end_control;

   esvalue = (int *) malloc(1);
   value0  = (int *) malloc(1);
```

```
   *esvalue = 200;
   *value0 = 0;
   clrscr();
   es_found = 0;
   rest = fopen(restfile,"w+b");
   f_tr = fopen("melody.dat", "rb");
   tr_out = fopen(output_file,"wb+");
   fread(&pre_note, sizeof(pre_note), 1, f_tr);
   do
   {
      fread(&post_note, sizeof(post_note), 1, f_tr);
      if (post_note.note_name == 20)
      {
         es_found = 1;
         file_end_control = fread(&es_note, sizeof(es_note), 1, f_tr);
         if (file_end_control)
         {
            dummy = es_note;
            es_note = post_note;
            post_note = dummy;
         }
         else
            es_note = post_note;
      }
      note_tr.gap = (post_note.note_name + post_note.pitch * 12) -
                    (pre_note.note_name + pre_note.pitch * 12);
      note_tr.duration = pre_note.duration;
      pre_note = post_note;
         fwrite(&note_tr.gap, sizeof(int), 1, tr_out);

         for (count = 0; count++ < (note_tr.duration-1);)
         {
            fwrite(value0, sizeof(int), 1, tr_out);
            printf("%d ",0);
         }
         fwrite(&note_tr.duration, sizeof(int), 1, rest);
      if (es_found)
      {
         note_tr.gap = 200;
         note_tr.duration = es_note.duration;
         fwrite(esvalue, sizeof(int), 1, tr_out);
         for (count = 0; count++< (note_tr.duration-1);)
            fwrite(esvalue, sizeof(int), 1, tr_out);
         es_found = 0;
      }
   }
   while (!feof(f_tr));
   fclose(rest);
   fclose(f_tr);
   fclose(tr_out);
}

/******************************************************************/

void get_met_no()
{
   printf("\n Enter the Metronom Number : ");
   scanf("%d",&metronom);
}

/******************************************************************/

void beat_sound()
{
   int beat_no, xpoint;
   char ch;

   clrscr();
```

```
   draw_rect(22,9,58,16,3);
   gotoxy(24,12);
   printf("Enter the Metronom Number : ");
   scanf("%d",&beat_no);
   cursor_off();
   beat_no = (int) ((1000.0 * 60.0 / beat_no) - 100);
   do
   {
      for (xpoint = 20; xpoint < 60; xpoint++)
      {
          gotoxy(xpoint - 1,18);
          putch(255);
          gotoxy(xpoint,18);
          putch(0xb2);
          delay((int) (beat_no / 40));
      }
      sound(200);
      delay(100);
      nosound();
      for (xpoint = 60; xpoint > 20; xpoint--)
      {
          gotoxy(xpoint + 1,18);
          putch(255);
          gotoxy(xpoint,18);
          putch(0xb2);
          delay((int) (beat_no / 40));
      }
      sound(200);
      delay(100);
      nosound();
   }
   while(!kbhit());
   getch();
   cursor_on();
}

/*******************************************************************/

void allocate_images()
{

   unsigned pixel;
   int g_drv = DETECT, g_mde;
   unsigned size;

   initgraph(&g_drv, &g_mde,"");
   size = imagesize(0, 0, 10, 26);
   not4 = malloc(size);
   not2 = malloc(size);
   size  = imagesize(0, 0, 19, 26);
   not16 = malloc(size);
   not8 = malloc(size);
   size = imagesize(0,0,9,3);
   s2 = malloc(size);
   size = imagesize(0,0,7,17);
   s4 = malloc(size);
   size = imagesize(0,0,9,9);
   s8 = malloc(size);
   size = imagesize(0,0,12,10);
   s16 = malloc(size);
   size = imagesize(0,0,5,9);
   diez = malloc(size);
   size = imagesize(0,0,15,37);
   keysol = malloc(size);
   size = imagesize(0,0,21,20);
   keyfa = malloc(size);
   size = imagesize(0,0,68,8);
   connection = malloc(size);
```

```
   take_note_images(not16,not8,not4,not2);
   take_es_images(s2,s4,s8,s16,diez,keysol,keyfa,connection);
   closegraph();
}

/****************************************************************/

void calc_samp_pro()
{
  sample_processed = (unsigned long) sec / (60.0 / metronom / 4);
}

/****************************************************************/

void initialize()
{
   clrscr();
   draw_rect(21,9,59,16,2);
   gotoxy(23,11);
   printf("Enter the Sampling Time in Sec. ");
   scanf("%f",&sec);
   gotoxy(23,13);
   printf("Enter the Metronom Number ");
   scanf("%d",&metronom);
   sample_amount = (int) sec * 8; /*With DELAY=215 freq of window = 8 Hz*/
   calc_samp_pro();
}

/****************************************************************/
void menu1();
/****************************************************************/

void menu1_1()
{
  int sel1;

  sel1 = menu_make(34,10,3,"Menu 1_1","Sample Data", "Take Transform",
                                       "Return", " ", " ", " ",14);
  switch (sel1)
  {
    case 1 : take_samples();
             menu1_1(); break;
    case 2 : take_transform(0);
             menu1_1(); break;
    case 3 : menu1(); break;
    default : menu1_1();
  }
}

/****************************************************************/

void menu2_1()
{
  int sel1;

  sel1 = menu_make(26,8,4,"Menu 2_1","Spectrum Print Without Harmonics",
                                     "Spectrum Print with Harmonics",
                                     "Time Domain Signal", "Return", " "," ",32);
  switch (sel1)
  {
    case 1 : spctr_print(3,0,0); menu2_1(); break;
    case 2 : spctr_print(3,0,1); menu2_1(); break;
    case 3 : look_to_the_input_signal(); menu2_1(); break;
    case 4 : menu1(); break;
  }
}

void menu3_1()
```

```
{
  int sel3;

  sel3 = menu_make(35,11,2,"Menu 3_1","Print Score",
                                    "Return",
                                    " ", " "," "," ",11);
  switch (sel3)
  {
    case 1 : dump_note(1,1);
             discard_harmonics();
             musical_score(1);
             menu3_1();
             break;
    case 2 : menu1(); break;
  }
}


/*****************************************************************/

void menu4_1()
{
  int sel4;

  sel4 = menu_make(34,10,3,"Menu 4_1","Initilization",
                                    "Metronom Beat",
                                    "Return", " "," "," ",13);
  switch (sel4)
  {
    case 1 : initialize(); menu4_1(); break;
    case 2 : beat_sound();
               menu4_1();
               break;
    case 3 : menu1(); break;
  }
}

/*****************************************************************/

void menu5_1()
{
   int sel5;
   char *filename;

   sel5 = menu_make(27,7,5,"Menu 5_1","First Melody Cognition",
                    "Second Melody Cognition","Correlation Between two Melodies",
                    "Heuristical Correlation","Return"," ",
                    32);
  switch (sel5)
  {
    case 1 : take_samples();
             take_transform(0);
             dump_note(1,1);
             discard_harmonics();
             forming_structure();
             transpose("trnsp1.dat", "rest1.dat");
             menu5_1();
             break;

    case 2 : take_samples();
             take_transform(0);
             dump_note(1,1);
             discard_harmonics();
             forming_structure();
             transpose("trnsp2.dat", "rest2.dat");
             menu5_1();
             break;
```

```
    case 3 : correlation(0);
             menu5_1();
             break;

    case 4 : correlation(1);
                  menu5_1();
                  break;

    case 5 : menu1(); break;

    default : menu5_1();
  }
}

/*******************************************************************/

void menu1()
{
   int select;

   select = menu_make(25,6,6,"Menu 1_0","Sampling and Transforming",
                "Frequency and Time Domain Analysing","Note Printing",
                "Initilization and Metronom", "Melody Compansation","Exit",40);
   switch (select)
   {
      case 1  : menu1_1(); break;
      case 2  : menu2_1(); break;
      case 3  : menu3_1(); break;
      case 4  : menu4_1(); break;
      case 5  : menu5_1(); break;
      case 6  : break;
      default : menu1();
   }
}

/*******************************************************************/

main()
{
   scr_type = (char *) farmalloc(sizeof(char));
   scr_ptr = (char *) farmalloc(sizeof(char)*160*20);
   scr_type = MK_FP(0x40,49);
   if (*scr_type == 7)
      scr_ptr = MK_FP(0xb000,000);
   else
      scr_ptr = MK_FP(0xb800,000);
   allocate_images();
   getdelay();
   initialize();
   speclapoint = DELAY * 56.0 / 215.0;
   power_index = 9;
   syze = WINDOW;
   menu1();
}→
```

```
/*Graph3.c*/
#include "dfnition.c"

extern void findfreqs_without_harmonics(int x_axis,int numofvoices,
                int notearray[], long result_array[]);
extern void findfreqs(int x_axis,int numofvoices,
                int notearray[], long result_array[]);
extern float speclapoint;
extern char * findname(long innote);
long findnote();
int gr_drv, gr_mde, gr_err;
int ErrorCode;
void *not2,*not4,*not8,*not16,*keysol,*keyfa,*connect;
void *s2,*s4,*s8,*s16,*diez;
extern int THRESOLD;

/***************************************************************/

graphinit()
{
   detectgraph(&gr_drv, &gr_mde);
   if (gr_drv < 0)
   {
      printf("No graphics hardware detected\n");
      exit(1);
   }
   if (gr_mde == EGAHI)
   gr_mde = EGALO;
   initgraph(&gr_drv, &gr_mde, "");
   gr_err = graphresult();

   if (gr_err < 0)
   {
      printf("initgraph error: %s.\n",
             grapherrormsg(gr_err));
      exit(1);
   }
   return(0);
}

/***************************************************************/

void draw_ver(int x)
{
  int i;
  for (i=10; i<getmaxy() - 5; i++)
  putpixel(x,i,7);
}

/***************************************************************/

void draw_hor(int y)
{
   int count;
   char far* txtstr;
   int pos;

   txtstr = (char*)malloc(20);
   strcpy(txtstr," Frequency (Hz.)");
   line(0, y, getmaxx(), y);
   settextstyle(SANS_SERIF_FONT, 0, 1);
   ErrorCode = graphresult();        /* check result              */
   if( ErrorCode != grOk )                   /* if error occured            */
   {
     closegraph();
     printf(" Graphics System Error: %s\n", grapherrormsg( ErrorCode ) );
     exit( 1 );
   }
```

```
   outtextxy(290, y+30, txtstr);
   strcpy(txtstr," Amplitude");
   settextstyle(SMALL_FONT,VERT_DIR,5);
   outtextxy(15,100,txtstr);
   settextstyle(SMALL_FONT, 0, 5);
   for(count = (int)(440.0 / speclapoint);
       count < (int) (440.0 / speclapoint *256.0);
       count += (int) (440.0 / speclapoint * 256.0 / 10.0))
       {
         pos = (int) count * speclapoint / 440.0;
         line(2 * pos * HORAXIS_ALIGN, y, 2 * pos * HORAXIS_ALIGN, y+4);
         txtstr = itoa(count, txtstr, 10);
         outtextxy(2 * pos * HORAXIS_ALIGN, y+4, txtstr);
       }
}

/*********************************************************/

void draw_rec()
{
   rectangle(0,0,getmaxx(),getmaxy());
   line(0,4,getmaxx(),0);
   line(0,getmaxy()-4,getmaxx(),0);
}

/*********************************************************/

void draw_spctr (int x_axis, int result_array[], int voicenum, int spctr_typ)
/*Draws the Spectrum over the x Axis. Input : Window size*/
{
   int count, incount;
   int dummy;
   char freqvar, *freqptr;
   char look[10];
   long printfreq;
   char * notename;
   long looknote[3];

   clearviewport();
   notename = (char *)malloc(10);
   if (spctr_typ == 0)
      findfreqs_without_harmonics(x_axis, voicenum, result_array, looknote);
   else
      findfreqs(x_axis, voicenum, result_array, looknote);
   freqptr = &freqvar;
   draw_ver(0);
   draw_hor(YAxis);
   rectangle(getmaxx()-200,0,getmaxx(),75);
   outtextxy(getmaxx()-180,7,"Point");
   outtextxy(getmaxx()-110,7,"Freq.");
   outtextxy(getmaxx()-50,7,"Note");
   line(getmaxx()-200,20,getmaxx(),20);
    setlinestyle(DOTTED_LINE,0,NORM_WIDTH);
   line(0,YAxis - THRESOLD, getmaxx(), YAxis - THRESOLD);
   outtextxy(0,YAxis - THRESOLD - 10,"   Treshold Value");
   setlinestyle(SOLID_LINE,0,NORM_WIDTH);
   for(count = 2; count < x_axis / 2; count++)
      line(count * 2 * HORAXIS_ALIGN, YAxis,
      count * 2 * HORAXIS_ALIGN, YAxis - abs(result_array[count]));
   for (count = 0; count < voicenum; count++)
   {
      freqptr = itoa(looknote[count], look, 10);
      outtextxy(getmaxx()-180,25 + 10 * count,freqptr);
      looknote[count] = abs((looknote[count]) * 440.0 / speclapoint);
      printfreq = findnote(looknote[count]);
      if (printfreq < 0)
      {
         closegraph();
```

```
            printf("\n   No RESPONSE !...");
            abort();
            }
        notename = findname(printfreq);
        outtextxy(getmaxx()-110, 25 + 10 * count, itoa(looknote[count], look, 10));
        outtextxy(getmaxx()-50 ,25 + 10 * count, notename);
    }
}

/*****************************************************************/

void printtime()
{
    struct  time timenow;

    gettime(&timenow);
    printf("\n %d %d %d %d",timenow.ti_hour,timenow.ti_min,
                timenow.ti_sec,timenow.ti_hund);
}

/*****************************************************************/

void drawporte(int position)
{
    unsigned lineno;

    setcolor(LIGHTGRAY);
    for(lineno = 0; lineno++ < 5;)
        line(5,position + lineno * 6, 770, position + lineno * 6);
}

/*****************************************************************/

take_note_images(void *note16, void *note8, void *note4, void *note2)
{
    unsigned xpoint,ypoint;
    int pixel;
    unsigned pos;

    int g_drv = DETECT, g_mde;
    FILE *fp;

    initgraph(&g_drv, &g_mde,"");
    fp = fopen("note.dat","r");
    for(pos =0; pos <40; pos+=30)
    {
        xpoint = pos;
        ypoint = 20;
        line(pos+9,0,pos+9,20);
        while ((pixel = fgetc(fp)) != '*')
        {
            if (pixel == 10)
            {
                ypoint++;
                xpoint = pos;
            }
            if(pixel == '1')
                putpixel(xpoint, ypoint, 7);
            xpoint++;
        }
    }

    for(pos =0; pos <40; pos+=30)
    {
        xpoint = 60 + pos;
        ypoint = 20;
        line(pos+60+9,0,pos+60+9,20);
        while ((pixel = fgetc(fp)) != '*')
```

```
        {
           if (pixel == 10)
           {
              ypoint++;
              xpoint = pos+60;
           }
           if(pixel == '1')
              putpixel(xpoint, ypoint, 7);
           xpoint++;
        }
        xpoint = 60 + pos + 9;
        ypoint = 0;
        while ((pixel = fgetc(fp)) != '*')
        {
           if (pixel == 10)
           {
              ypoint++;
              xpoint = pos+60+9;
           }
           if(pixel == '1')
              putpixel(xpoint, ypoint, 7);
           xpoint++;
           }
        }
     getimage(1, 0, 10, 26, note4);
     getimage(31,1, 40, 26, note2);
     getimage(61,1, 77, 26, note8);
     getimage(91,1,107, 26, note16);
     closegraph();
  }

  /**********************************************************/

  void take_solkey(void *solkey)
  {
     int g_drv = DETECT, g_mde, pixel, xpoint, ypoint, pos;
     FILE *fp;

     xpoint = ypoint = 0;
     initgraph(&g_drv, &g_mde,"");
     fp = fopen("note.dat","r");
     while ((pixel = fgetc(fp)) != '*')
     {
        if (pixel == 10)
        {
           ypoint++;
           xpoint = 0;
        }
        if(pixel == '1')
           putpixel(xpoint, ypoint, 7);
        xpoint++;
     }
     getimage(0, 0, 15, 37, solkey);
     getch();
     closegraph();
     fclose(fp);
  }

  /**********************************************************/

  void take_es_images(void *es2, void *es4, void *es8, void *es16, void *diyez,
                          void *solkey, void *fakey, void *connect)
  {
     unsigned xpoint,ypoint;
     int pixel;
     unsigned pos;

     int g_drv = DETECT, g_mde;
```

```
     *fp;

   initgraph(&g_drv, &g_mde,"");
   fp = fopen("es.dat","r");
   for(pos =0; pos <130; pos+=30)
   {
      xpoint = pos;
      ypoint = 0;
      while ((pixel = fgetc(fp)) != '*')
      {
         if (pixel == 10)
         {
            ypoint++;
            xpoint = pos;
         }
         if(pixel == '1')
            putpixel(xpoint, ypoint, 7);
         xpoint++;
      }
   }
   getimage(1, 0, 9, 3, es2);
   getimage(31,0, 37, 17, es4);
   getimage(61,0, 69, 9, es8);
   getimage(91,0,102, 10, es16);
   getimage(121,0,128,11,diyez);
   clearviewport();
   xpoint = ypoint = 0;
   while ((pixel = fgetc(fp)) != '*')
   {
      if (pixel == 10)
      {
         ypoint++;
         xpoint = 0;
      }
      if(pixel == '1')
         putpixel(xpoint, ypoint, 7);
      xpoint++;
   }
   getimage(0, 0, 15, 37, solkey);
   clearviewport();
   xpoint = ypoint = 0;
   while ((pixel = fgetc(fp)) != '*')
   {
      if (pixel == 10)
      {
         ypoint++;
         xpoint = 0;
      }
      if(pixel == '1')
         putpixel(xpoint, ypoint, 7);
      xpoint++;
   }
   getimage(0, 0, 21, 20, fakey);
   clearviewport();
   xpoint = ypoint = 0;
   while ((pixel = fgetc(fp)) != '*')
   {
      if (pixel == 10)
      {
         ypoint++;
         xpoint = 0;
      }
      if(pixel == '1')
         putpixel(xpoint, ypoint, 7);
      xpoint++;
   }
   getimage(0, 0, 68, 8, connect);
   closegraph();
```

```
   fclose(fp);
}

/*******************************************************/

void dot_print(int x_point, int y_point)
{
   int line,column;

   for(line = 0; line <2; line++)
      for(column = 0; column < 2; column++)
         putpixel(x_point+column,y_point+line,GREEN);
}

/*******************************************************/

void porte()
{
   int port_no, line = UPPER_LINE;

   for(port_no = 0; port_no < 4; port_no++)
   {
      drawporte(line);
         putimage(3, line-2, keysol, OR_PUT);
      line += 80;
   }
}

/*******************************************************/

void random_put(void *n8, void *n2)
{
   int ranx, rany = 1, count;

  for(count = 0; count < 750; count += 20)
  {
     putimage(count, rany += 2 , n8, NOT_PUT);
     putimage(count, (rany + 100) , n2, AND_PUT);
  }
}

/*******************************************************/

void print_note(int note, int pitch, int duration, int col_no, int line_no)
{
   void *not;
   int row_no, found, count,ref_line;

   found = 0;
   ref_line = UPPER_LINE + (line_no-1) * 80;
   switch (duration)
   {
      case 1 : not = not16; break;
      case 2 :
      case 3 : not = not8 ; break;
      case 4 :
      case 5 :
      case 6 : not = not4; break;
      case 7 :
      case 8 :
      case 9 :
      case 10:
      case 11:
      case 12:
      case 13:
      case 14:
      case 15:
      case 16: not = not2; break;
```

```
      default : not = not2; break;
   }

   switch (note)
   {
      case  0 : row_no = 39; found = 1; break;
      case  1 : row_no = 39; break;
      case  2 : row_no = 36; found = 1; break;
      case  3 : row_no = 33; found = 1; break;
      case  4 : row_no = 33; break;
      case  5 : row_no = 30; found = 1; break;
      case  6 : row_no = 30; break;
      case  7 : row_no = 27; found = 1; break;
      case  8 : row_no = 24; found = 1; break;
      case  9 : row_no = 24; break;
      case 10 : row_no = 21; found = 1; break;
      case 11 : row_no = 21; break;
      case 12 : row_no = 39; ++pitch; found = 1; break;
   }
   row_no -= (pitch - 4) * 21;
   row_no += ((line_no-1) * 80);
   if (!found)
   {
      putimage(col_no, row_no + 16, diez, OR_PUT);
      col_no += 12;
   }
   putimage(col_no, row_no, not, OR_PUT);
   if (duration == 3 || duration == 6 || (duration >10 && duration <14))
      dot_print(col_no +14, row_no +22);
   if ((note == 3 || note == 4) && (pitch == 3))
      line(col_no - 3, row_no+22, col_no + 10, row_no+22);
   if ((note == 2 ) && (pitch == 3))
      line(col_no - 3, row_no+19, col_no + 10, row_no+19);
   if ((note == 0 || note == 1) && (pitch == 3))
   {
      line(col_no - 3, row_no+16, col_no + 10, row_no+16);
      line(col_no - 3, row_no+22, col_no + 10, row_no+22);
   }
   if(pitch >= 5)
      for (count = 0; count <= (18 - (row_no - 80*(line_no-1)) % (80 + ref_line + 6))
/ 6; count++)
         line(col_no - 3, ref_line - count*6 ,
              col_no + 10, ref_line - count*6);
   if(pitch < 3)
      for (count = 0; (int)count <((row_no + 25) - (ref_line + 30)) / 6; count++)
          line(col_no - 3, ref_line + 30 + count*6,
               col_no + 10, ref_line + 30 + count*6);
}

/*************************************************************/

void print_rest(int duration, int col_no, int pos_no)

{
   void *es;
   int row_no, found, count;

   switch (duration)
   {
      case 1 : es = s16; break;
      case 2 :
      case 3 : es = s8 ; break;
      case 4 :
      case 5 :
      case 6 :
      case 7 : es = s4; break;
      case 8 :
      case 9 :
```

```
      case 10:
      case 11:
      case 12:
      case 13:
      case 14:
      case 15:
      case 16: es = s2; break;

      default : es = s2; break;
   }
   row_no = UPPER_LINE + 80*(pos_no - 1) + 10;
   putimage(col_no, row_no, es, OR_PUT);
   col_no += 12;
   if (duration == 3 || duration == 6 || duration == 12)
   dot_print(col_no +10, row_no + 5);
}

→
```

```
/*Correlat.C*/
#define  ESFACTOR 5;

   int static col_no, row_no;
   int static note_mat[30][30];
   int static rest_mat[30][30];
   int i,j,dummy;

/*********************************************************/

int sqr(int number)
{
   return(number*number);
}


void heuristic()
{
   int inmatrix;
   int count;
   int f_note, m_note, p_note;
   int mean;
   int count2;

   mean = 0;
   for (count = 1; count < row_no; count++)
      mean+= abs(note_mat[0][count]);
   (int) mean/= row_no;
   for (count = 1; count < row_no-2; count++)
   {
      f_note = note_mat[0][count];
      m_note = note_mat[0][count+1];
      p_note = note_mat[0][count+2];
      if((f_note != 200) && (m_note != 200) && (p_note != 200))
      if(abs(f_note - m_note) > (mean + 15))
      {
         note_mat[0][count+1]-= 19;
         note_mat[0][count+2]+= 19;
      }
   }
   mean = 0;
   for (count = 1; count < row_no; count++)
      mean+= abs(note_mat[count][0]);
   (int) mean/= row_no;
   for (count = 1; count < row_no-2; count++)
   {
      f_note = note_mat[count][0];
      m_note = note_mat[count+1][0];
      p_note = note_mat[count+2][0];
      if(abs(f_note - m_note) > (mean + 10))/* && ((m_note + p_note) == 0))*/
      {
         note_mat[count+1][0]-= 19;
         note_mat[count+2][0]+= 19;
      }
   }
}

/*********************************************************/

void get_note_matrix(int hrstc)
{
  FILE   *intr1, *intr2;
  int    dummy, *dump;


   i = 1;
   clrscr();
   dump = &dummy;
```

```
   intr1 = fopen("trnsp1.dat","rb");
   intr2 = fopen("trnsp2.dat","rb");
   while(!feof(intr1))
   {
      fread(dump, sizeof(int), 1, intr1);
      if (*dump != 0)
         note_mat[i++][0] = *dump;
   }
   row_no = i-1;
   i = 1;
   while(!feof(intr2))
   {
      fread(dump, sizeof(int), 1, intr2);
      if (*dump != 0)
         note_mat[0][i++] = *dump;
   }
   col_no = i-1;
   if (hrstc)
      heuristic();
   for (i = 1; i <= row_no; i++)
      for (j = 1; j <= col_no; j++)
      {
         if ((note_mat[0][j] == 200) && (note_mat[i][0] != 200))
            note_mat[i][j] = ESFACTOR
         else
         if ((note_mat[0][j] != 200) && (note_mat[i][0] == 200))
            note_mat[i][j] = ESFACTOR
         else
         note_mat[i][j] = sqr(abs(note_mat[0][j] - note_mat[i][0]));
      }
   clrscr();
   for (i = 0; i <=row_no; i++)
   {
      printf("\n");
      for (j =0; j <= col_no; j++)
         printf("\ %2d", note_mat[i][j]);
   }
   fclose(intr1);
   fclose(intr2);
}

/*********************************************************/

void find_correlation(int matrix[30][30])
{
   int cross_link;
   int i,j,count;
   int correlation;
   char ch;

   cross_link = abs(col_no - row_no) + 1;
   if (col_no > row_no)
   {
      for(count = 0; count < cross_link; count++)
      {
         correlation = 0;
         for(i = 1; i <= row_no; i++)
            correlation+= matrix[i][i+count];
         printf("\ncorrelation = %d",correlation);
      }
   }
   else
   {
      for(count = 0; count < cross_link; count++)
      {
         correlation = 0;
         for(i = 1; i <= col_no; i++)
            correlation+= matrix[i+count][i];
```

```
      }
   }
   while (!kbhit()) ;
   ch = getch();
}

/**************************************************************/

void get_rest_matrix(float mul_row, float mul_col)
{
   FILE  *r1, *r2;
   int *dump, dummy;

   i = 1;
   dump = &dummy;
   r1 = fopen("rest1.dat","rb");
   r2 = fopen("rest2.dat","rb");
   while(!feof(r1))
   {
      fread(dump, sizeof(int), 1, r1);
      if (*dump != 0)
        (int) rest_mat[i++][0] = ceil(*dump * mul_row);
   }
   row_no = i-1;
   i = 1;
   while(!feof(r2))
   {
      fread(dump, sizeof(int), 1, r2);
      if (*dump != 0)
         (int) rest_mat[0][i++] = ceil(*dump * mul_col);
   }
   col_no = i-1;

   for (i = 1; i <= row_no; i++)
      for (j = 1; j <= col_no; j++)
         rest_mat[i][j] = sqr(abs(rest_mat[0][j] - rest_mat[i][0]));
   for (i = 0; i <= row_no; i++)
   {
      printf("\n");
      for (j =0; j <= col_no; j++)
         printf("\ %2d", rest_mat[i][j]);
   }
   fclose(r1);
   fclose(r2);
}

/**************************************************************/

void mul_corr()
{
   int row,col;

   for (row = 1; row < 4; row++)
   for (col = 1; col < 4; col++)
   {
     printf("\n\nMultiplication factors of Row:%2.1f Column:%2.1f",
             row/2.0,col/2.0);
     get_rest_matrix(row/2.0,col/2.0);
     find_correlation(rest_mat);
   }
}

/**************************************************************/

void correlation(int heur)
{
   get_note_matrix(heur);
```

```
    find_correlation(note_mat);
    mul_corr();
}
→
```

```
/*Eminlib.C*/
char far* scr_ptr;

char * getline()
{
   char * inline;
   char c;
   int  count;

   inline = (char *) malloc(100);
   count = -1;
   while ((c = getchar()) != '\n')
   {
      count++;
      inline[count] = c;
   }
   return (inline);
}

char * cat(char * s1, char * s2)
{
   char * line;
   char c;
   int  count, ncoun;

   line = (char *) malloc(20);
   for(count = 0; c = s1[count], c != '\0'; count++)
      line[count] = c;
   ncoun = 0;
   for(ncoun = 0;c = s2[ncoun], c != '\0'; ncoun++)
      line[count++] = s2[ncoun];
      line[count] = '\0';
   return(line);
}

writelnstr(char instr[])
{
   int count;
   int cmd, abyte, port;

   for (count = 0; instr[count] != '\x0'; count++) {
       abyte = (int) instr[count];
       biosprint(0, abyte, 0);
   }
   biosprint(0, 0x000d, 0);
   biosprint(0, 0x000a, 0);
}

writestr(char instr[])
{
   int count;
   int cmd, abyte, port;

   for (count = 0; instr[count] != '\x0'; count++) {
       abyte = (int) instr[count];
       biosprint(0, abyte, 0);
   }
}


void draw_rect(int x1, int y1, int x2, int y2, int rect_type)
{
   int height;
   int count;
   int hor_line,ver_line,
       up_left_corner, up_right_corner, down_left_cor,
       down_right_cor;
   switch (rect_type) {
```

```
        case 1 : hor_line =  196;
                 ver_line =  179;
                 up_left_corner  = 218;
                 up_right_corner = 191;
                 down_left_cor   = 192;
                 down_right_cor  = 217;
                 break;
        case 2 : hor_line = 205;
                 ver_line = 179;
                 up_left_corner  = 213;
                 up_right_corner = 184;
                 down_left_cor   = 212;
                 down_right_cor  = 190;
                 break;
        case 3 : hor_line = 205;
                 ver_line = 186;
                 up_left_corner  = 201;
                 up_right_corner = 187;
                 down_left_cor   = 200;
                 down_right_cor  = 188;
                 break;
    }
    height = y2 - y1 - 1;
    gotoxy(x1, y1);
    putch(up_left_corner);
    for (count = x1+1; count < x2-1; count++)
       putch(hor_line);
    putch(up_right_corner);
    for (count = y1+1; count < y2; count++) {
       gotoxy(x1,count);
       putch(ver_line);
       gotoxy(x2-1,count);
       putch(ver_line);
    }
    gotoxy(x1, y1+height);
    putch(down_left_cor);
    for (count = x1+1; count < x2-1; count++)
       putch(hor_line);
    putchar(down_right_cor);
}

void cursor_on()
{
    struct REGPACK regs;
    regs.r_ax = 0x0100;
    regs.r_cx = 0x0C0D;
    intr(0x10, &regs);
}


void cursor_off()
{
    struct REGPACK regs;


    regs.r_ax = 0x0100;
    regs.r_cx = 0xffff;
    intr(0x10, &regs);
}

void rev_video(int y, int x, int length)
{
    int count;

    for(count = x-1; count < x+length-1; count++)
      *(scr_ptr + 2*count + (80*(y-1)*2) + 1) = 0x70;
}
```

```
void norm_video(int y, int x, int length)
{
   int count;

   *(scr_ptr + 2*(x-1) +(80*(y-1)*2)+1) = 0x0f;
   for(count = x; count < x+length-1; count++)
     *(scr_ptr + 2*count + (80*(y-1)*2) + 1) = 0x7;
}


void white_initial(char in_text[])
{
   int count, ch;

   textcolor(WHITE);
   cprintf("%c",in_text[0]);
   count = 1;
   textcolor(LIGHTGRAY);
   while ( (ch = in_text[count++]) != '\0')
      putchar(ch);
}


void menu_wind()
{
   int count;

   textcolor(LIGHTGRAY);
   clrscr();
   for (count = 2; count < 80; count++) {
   gotoxy(count,1);
   putchar(220);
   }
   for (count = 1; count < 24; count++) {
      gotoxy(1,count);
      putchar(179);
      gotoxy(80,count);
      putchar(179);
   }
   gotoxy(0,24);
   putchar(192);
   for (count = 2; count < 80; count++) {
      gotoxy(count,0);
      putchar(196);
   }
   gotoxy(80,24);
   putchar(217);
}

int menu_make(int x, int y, int choice_no,
               char menu_name[], char str1[], char str2[],
               char str3[], char str4[], char str5[],
               char str6[],int str_lngth)
{
   char in_choices[6][50];
   char ch;
   int length, sel_no, count;
   int select_char;
   char initial[6], small[6];
   int y_pos, y_up_pos, y_down_pos, end_pos;
   int menu_out;

   textcolor(LIGHTGRAY);
   textbackground(BLACK);
   ch = 0x20;
   menu_wind();
   cursor_off();
   strcpy(in_choices[0],str1);
```


T. C.
Yükseköğretim Kurulu
Dokümantasyon Merkezi

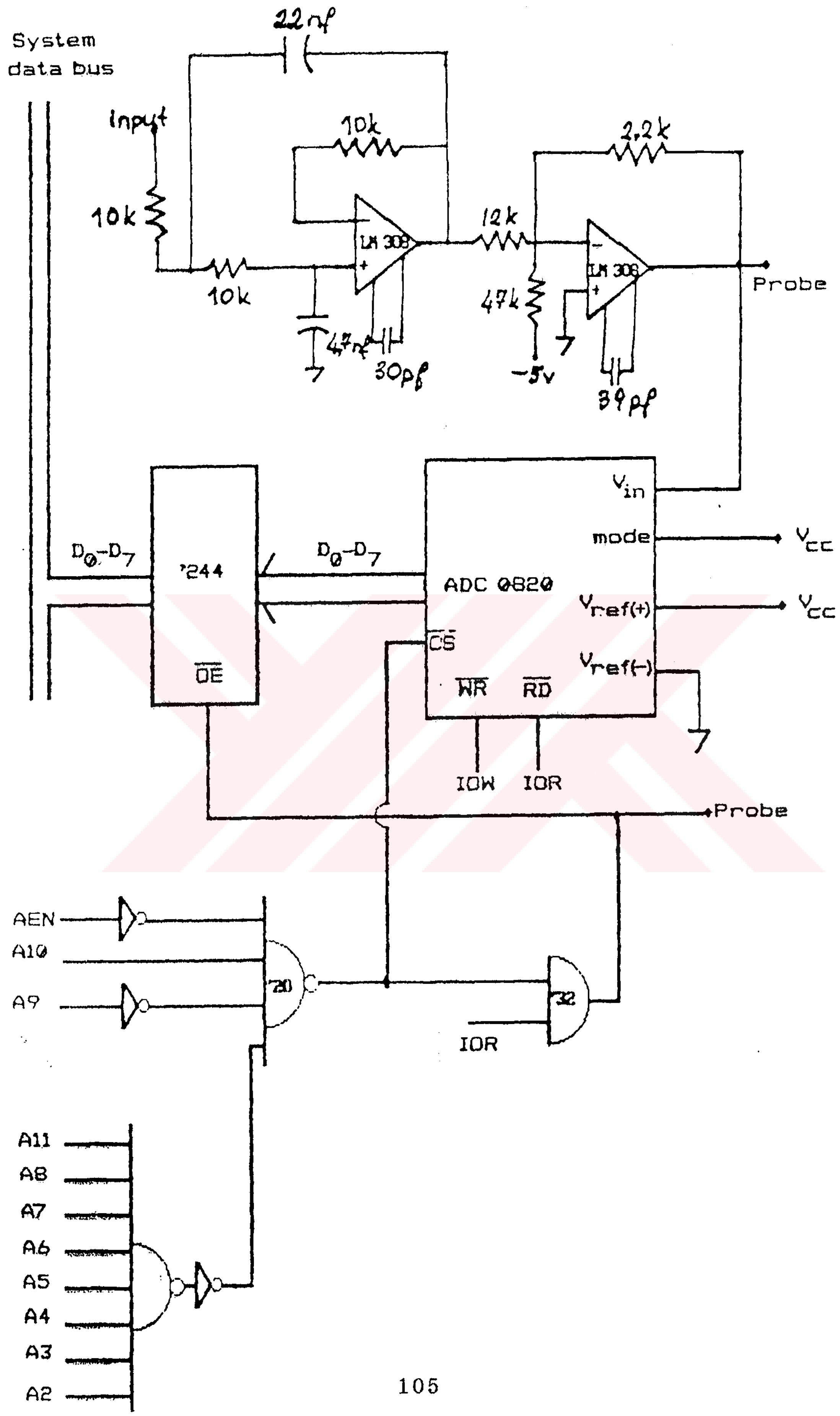
```
   strcpy(in_choices[1],str2);
   strcpy(in_choices[2],str3);
   strcpy(in_choices[3],str4);
   strcpy(in_choices[4],str5);
   strcpy(in_choices[5],str6);
   end_pos = y + 3 * (choice_no - 1);
   select_char = 0;
   for(count = 0; count < choice_no; count++) {
      initial[count] = in_choices[count][0];
      small[count] = initial[count] + 32;
   }
   sel_no = 1;
   gotoxy(3,2);
   printf("%s",menu_name);
   for(count = 0; count < choice_no; count++) {
      gotoxy(x,(y+count*3));
      white_initial(in_choices[count]);
   }

   do {
      y_pos = (sel_no - 1) * 3 + y;
      y_up_pos = y_pos + 3;
      if (y_up_pos > end_pos)
          y_up_pos = y;
      y_down_pos = y_pos - 3;
      if (y_down_pos < y)
          y_down_pos = end_pos;
      norm_video(y_up_pos, x, str_lngth);
      norm_video(y_down_pos, x, str_lngth);
      rev_video(y_pos, x, str_lngth);
      if(kbhit()) {
          ch = getch();
          if (ch == 0) {
           ch = getch();
           switch (ch) {
              case 80 : if(++sel_no == choice_no+1)
                                          sel_no = 1;
                                    break;
              case 72 :  if(--sel_no == 0)
                                          sel_no = choice_no;
                                    break;
           }
          }
          else {
           count = -1;
           while(++count <= choice_no) {
              if((ch == initial[count]) || (ch == small[count]))
                     menu_out = count+1;
                     select_char = 1;
           }
          }
        if (ch == 13)
          menu_out = sel_no;
      }
   }
   while( (ch != 27) && (ch != 13) && !select_char);
   if(ch == 27)
      menu_out = choice_no;
   cursor_on();
   return(menu_out);
}
```

```
/*Sample.C*/
#include <alloc.h>
#include <stdio.h>
#pragma inline
extern int DELAY;

void sample_data(inptr,sampamount)
char far *inptr;
unsigned long sampamount;


{
   unsigned long x;
   unsigned char dummy;

     asm  cli
     asm  push  ax
     asm  push  cx
     asm  push  dx

for (x=0; x < sampamount; x++)

{
     asm  mov dx,0dffh
     asm  out dx,al
     asm  mov cx,DELAY
     label:
     asm  dec cx
     asm  jnz label
     asm  mov dx,0dffh
     asm  in  al,dx
     asm  mov dummy,al
     *(inptr+x) = dummy;
}
     asm  pop   dx
     asm  pop   cx
     asm  pop   ax
     asm  sti

}
```

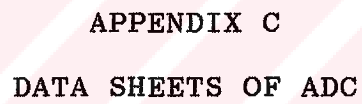
```
/*Dfnition.C*/
#define HORAXIS_ALIGN 640.0 / 512.0
#define UPPER_LINE 40
#define YAxis getmaxy() / 2 + 100
#define PI 3.14159265
#define WINDOW 512
#define forw    0
#define reverse 1
#define MAXCOUNT 11
#define A1 55
#define WINDOWTIME  64.0/440.0 /* LA4 = 64 then maxfreq = 256 * 440 / 64
                                      sampling frequency = 2 * (256 * 440 / 64
                                      1 / windowtime = sampling frequency / 512*/
```

APPENDIX B

THE SCHEMATIC DIAGRAM OF INTERFACE CARD



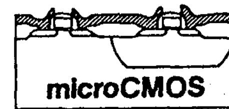
System
data bus
22nf
Input
10k
10k
10k
LM 308
47nf
30pf
2,2k
12k
47k
-5v
LM 308
39pf
Probe
D_0-D_7
'244
OE
D_0-D_7
V_{in}
mode
V_{cc}
ADC 0820
$V_{ref(+)}$
V_{cc}
CS
$V_{ref(-)}$
WR
RD
IOW
IOR
Probe
AEN
A10
A9
'20
'32
IOR
A11
A8
A7
A6
A5
A4
A3
A2



APPENDIX C

DATA SHEETS OF ADC

National Semiconductor



ADC0820 8-Bit High Speed μP Compatible A/D Converter with Track/Hold Function

General Description

By using a *half-flash conversion technique*, the 8-bit ADC0820 CMOS A/D offers a 1.5 μs conversion time and dissipates only 75 mW of power. The half-flash technique consists of 32 comparators, a most significant 4-bit ADC and a least significant 4-bit ADC.

The input to the ADC0820 is tracked and held by the input sampling circuitry eliminating the need for an external sample-and-hold *for signals moving at less than 100 mV/μs.*

For ease of interface to microprocessors, the ADC0820 has been designed to appear as a memory location or I/O port without the need for external interfacing logic.

Key Specifications

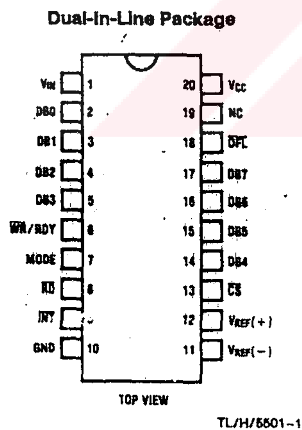
- Resolution: 8 Bits
- Conversion Time: 2.5 μs Max (RD Mode); 1.5 μs Max (WR-RD Mode)
- *Input signals with slew rate of 100 mV/μs converted without external sample-and-hold to 8 bits*
- Low Power: 75 mW Max
- Total Unadjusted Error: ± ½ LSB and ± 1 LSB

Features

- Built-in track-and-hold function
- No missing codes
- No external clocking
- Single supply—5 V_{DC}
- Easy interface to all microprocessors, or operates stand-alone
- Latched TRI-STATE® output
- Logic inputs and outputs meet both MOS and T^2L voltage level specifications
- *Operates ratiometrically or with any reference value equal to or less than* V_{CC}
- 0V to 5V analog input voltage range with single 5V supply
- No zero or full-scale adjust required
- Overflow output available for cascading
- 0.3" standard width 20-pin DIP

Connection and Functional Diagrams

Dual-In-Line Package



Order Number ADC0820D or ADC0820N
See NS Package D20A or N20A

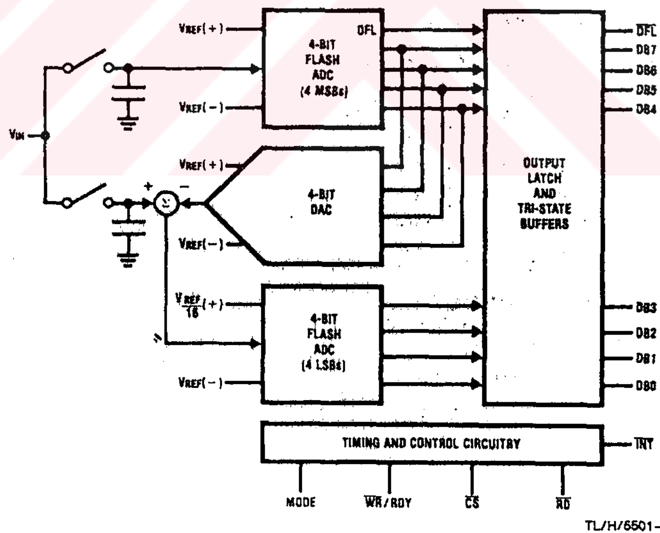


FIGURE 1

Absolute Maximum Ratings (Notes 1 & 2)

Supply Voltage (V_{CC})	10V
Logic Control Inputs	$-0.2V$ to $V_{CC}+0.2V$
Voltage at Other Inputs and Output	$-0.2V$ to $V_{CC}+0.2V$
Storage Temperature Range	$-65°C$ to $+150°C$
Package Dissipation at $T_A = 25°C$	875 mW
Lead Temp. (Soldering, 10 seconds)	300°C

Operating Conditions (Notes 1 & 2)

Temperature Range	$T_{MIN} \le T_A \le T_{MAX}$
ADC0820BD, ADC0820CD	$-55°C \le T_A \le +125°C$
ADC0820BCD, ADC0820CCD	$-40°C \le T_A \le +85°C$
ADC0820BCN, ADC0820CCN	$0°C \le T_A \le 70°C$
V_{CC} Range	4.5V to 8V

Converter Characteristics

The following specifications apply for RD mode (pin 7 = 0), $V_{CC}=5V$, $V_{REF}(+)=5V$, and $V_{REF}(-)=GND$ unless otherwise specified. **Boldface limits apply from T_{MIN} to T_{MAX};** all other limits $T_A=T_j=25°C$.

Parameter	Conditions	ADC0820BD, ADC0820CD ADC0820BCD, ADC0820CCD			ADC0820BCN, ADC0820CCN			Limit Units
		Typ (Note 6)	Tested Limit (Note 7)	Design Limit (Note 8)	Typ (Note 6)	Tested Limit (Note 7)	Design Limit (Note 8)	
Resolution			**8**			8	**8**	Bits
Total Unadjusted Error (Note 3)	ADC0820BD, BCD		**±½**					LSB
	ADC0820BCN					±½	**±½**	LSB
	ADC0820CD, CCD		**±1**					LSB
	ADC0820CCN					±1	**±1**	LSB
Minimum Reference Resistance		2.3	**1.25**		2.3	1.4	**1.25**	kΩ
Maximum Reference Resistance		2.3	**6**		2.3	5.3	**6**	kΩ
Maximum $V_{REF}(+)$ Input Voltage			**V_{CC}**			V_{CC}	**V_{CC}**	V
Minimum $V_{REF}(-)$ Input Voltage			**GND**			GND	**GND**	V
Minimum $V_{REF}(+)$ Input Voltage			**$V_{REF}(-)$**			$V_{REF}(-)$	**$V_{REF}(-)$**	V
Maximum $V_{REF}(-)$ Input Voltage			**$V_{REF}(+)$**			$V_{REF}(+)$	**$V_{REF}(+)$**	V
Maximum V_{IN} Input Voltage			**$V_{CC}+0.1$**			$V_{CC}+0.1$	**$V_{CC}+0.1$**	V
Minimum V_{IN} Input Voltage			**GND−0.1**			GND−0.1	**GND−0.1**	V
Maximum Analog Input Leakage Current	$\overline{CS}=V_{CC}$							
	$V_{IN}=V_{CC}$		**3**			0.3	**3**	μA
	$V_{IN}=GND$		**−3**			−0.3	**−3**	μA
Power Supply Sensitivity	$V_{CC}=5V \pm 5\%$	±1/16	**±¼**		±1/16	±¼	**±¼**	LSB

DC Electrical Characteristics

The following specifications apply for $V_{CC}=5V$, unless otherwise specified. **Boldface limits apply from T_{MIN} to T_{MAX}**; all other limits $T_A=T_J=25°C$.

Parameter	Conditions		ADC0820BD, ADC0820CD ADC0820BCD, ADC0820CCD Typ (Note 6)	ADC0820BD, ADC0820CD ADC0820BCD, ADC0820CCD Tested Limit (Note 7)	ADC0820BD, ADC0820CD ADC0820BCD, ADC0820CCD Design Limit (Note 8)	ADC0820BCN, ADC0820CCN Typ (Note 6)	ADC0820BCN, ADC0820CCN Tested Limit (Note 7)	ADC0820BCN, ADC0820CCN Design Limit (Note 8)	Limit Units
$V_{IN(1)}$, Logical "1" Input Voltage	$V_{CC}=5.25V$	$\overline{CS}$, $\overline{WR}$, $\overline{RD}$		**2.0**			2.0	**2.0**	V
		Mode		**3.5**			3.5	**3.5**	V
$V_{IN(0)}$, Logical "0" Input Voltage	$V_{CC}=4.75V$	$\overline{CS}$, $\overline{WR}$, $\overline{RD}$		**0.8**			0.8	**0.8**	V
		Mode		**1.5**			1.5	**1.5**	V
$I_{IN(1)}$, Logical "1" Input Current	$V_{IN(1)}=5V$; $\overline{CS}$, $\overline{RD}$ $V_{IN(1)}=5V$; $\overline{WR}$ $V_{IN(1)}=5V$; Mode		0.005 0.1 50	**1** **3** **200**		0.005 0.1 50	 0.3 170	**1** **3** **200**	µA µA µA
$I_{IN(0)}$, Logical "0" Input Current	$V_{IN(0)}=0V$; $\overline{CS}$, $\overline{RD}$, $\overline{WR}$, Mode		−0.005	**−1**		−0.005		**−1**	µA
$V_{OUT(1)}$, Logical "1" Output Voltage	$V_{CC}=4.75V$, $I_{OUT}=-360$ µA; DB0–DB7, $\overline{OFL}$, $\overline{INT}$ $V_{CC}=4.75V$, $I_{OUT}=-10$ µA; DB0–DB7, $\overline{OFL}$, $\overline{INT}$			**2.4** **4.5**			2.8 4.6	**2.4** **4.5**	V V
$V_{OUT(0)}$, Logical "0" Output Voltage	$V_{CC}=4.75V$, $I_{OUT}=1.6$ mA; DB0–DB7, $\overline{OFL}$, $\overline{INT}$, RDY			**0.4**			0.34	**0.4**	V
I_{OUT}, TRI-STATE Output Current	$V_{OUT}=5V$; DB0–DB7, RDY $V_{OUT}=0V$; DB0–DB7, RDY		0.1 −0.1	**3** **−3**		0.1 −0.1	0.3 −0.3	**3** **−3**	µA µA
I_{SOURCE}, Output Source Current	$V_{OUT}=0V$; DB0–DB7, $\overline{OFL}$ $\overline{INT}$		−12 −9	**−6** **−4.5**		−12 −9	−7.2 −5.3	**−6** **−4.5**	mA mA
I_{SINK}, Output Sink Current	$V_{OUT}=5V$; DB0–DB7, $\overline{OFL}$, $\overline{INT}$, RDY		14	**7**		14	8.4	**7**	mA
I_{CC}, Supply Current	$\overline{CS}=\overline{WR}=\overline{RD}=0$		7.5	**15**		7.5	13	**15**	mA

AC Electrical Characteristics

The following specifications apply for $V_{CC}=5V$, $t_r=t_f=20$ ns, $V_{REF}(+)=5V$, $V_{REF}(-)=0V$ and $T_A=25°C$ unless otherwise specified.

Parameter		Conditions	Typ (Note 6)	Tested Limit (Note 7)	Design Limit (Note 8)	Units
t_{CRD}, Conversion Time for RD Mode		Pin 7 = 0, *(Figure 2)*	1.6		2.5	µs
t_{ACC0}, Access Time (Delay from Falling Edge of $\overline{RD}$ to Output Valid)		Pin 7 = 0, *(Figure 2)*	$t_{CRD}+20$		$t_{CRD}+50$	ns
t_{CWR-RD}, Conversion Time for WR-RD Mode		Pin 7 = V_{CC}; t_{WR} = 600 ns, $t_{RD}=600$ ns; *(Figures 3a and 3b)*			1.52	µs
t_{WR}, Write Time	Min	Pin 7 = V_{CC}; *(Figures 3a and 3b)*		600		ns
	Max	(Note 4) See Graph	50			µs
t_{RD}, Read Time	Min	Pin 7 = V_{CC}; *(Figures 3a and 3b)* (Note 4) See Graph		600		ns
t_{ACC1}, Access Time (Delay from Falling Edge of $\overline{RD}$ to Output Valid)		Pin 7 = V_{CC}, $t_{RD}<t_I$; *(Figure 3a)* $C_L=15$ pF	190		280	ns
		$C_L=100$ pF	210		320	ns
t_{ACC2}, Access Time (Delay from Falling Edge of $\overline{RD}$ to Output Valid)		Pin 7 = V_{CC}, $t_{RD}>t_I$; *(Figure 3b)* $C_L=15$ pF	70		120	ns
		$C_L=100$ pF	90		150	ns

Timing Diagrams

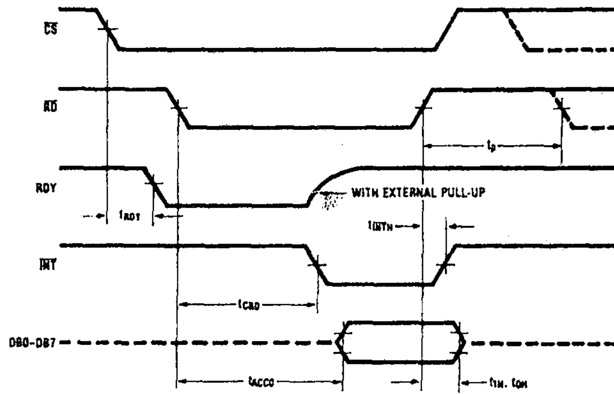


FIGURE 2. RD Mode (Pin 7 is Low)

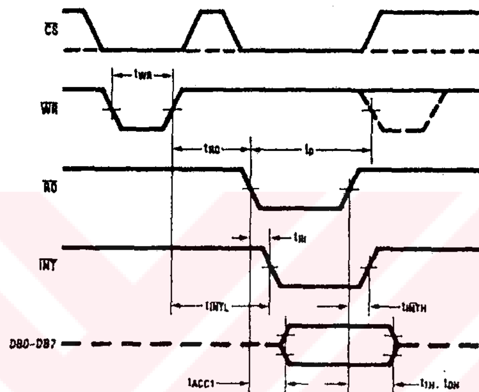


FIGURE 3a. WR-RD Mode (Pin 7 is High and $t_{RD} < t_I$)

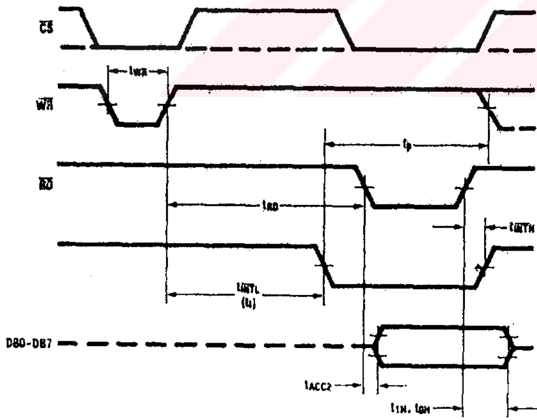


FIGURE 3b. WR-RD Mode (Pin 7 is High and $t_{RD} > t_I$)

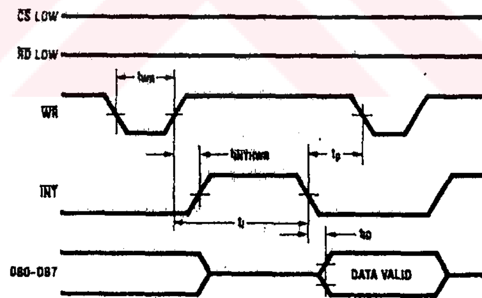


FIGURE 4. WR-RD Mode (Pin 7 is High) Stand-Alone Operation

APPENDIX D

EIGHT OCTAVES OF MUSICAL NOTE FREQUENCIES

Note	*Frequency*	*Note*	*Frequency*	*Note*	*Frequency*	*Note*	*Frequency*
C_0	16.35	C_2	65.41	C_4	261.63	C_6	1046.50
$C_{\#0}$	17.32	$C_{\#2}$	69.30	$C_{\#4}$	277.18	$C_{\#6}$	1108.73
D_0	18.35	D_2	73.42	D_4	293.66	D_6	1174.66
$D_{\#0}$	19.45	$D_{\#2}$	77.78	$D_{\#4}$	311.13	$D_{\#6}$	1244.51
E_0	20.60	E_2	82.41	E_4	329.63	E_6	1328,51
F_0	21.83	F_2	87.31	F_4	349.23	F_6	1396.91
$F_{\#0}$	23.12	$F_{\#2}$	92.50	$F_{\#4}$	369.99	$F_{\#6}$	1479.98
G_0	24.50	G_2	98.00	G_4	392.00	G_6	1567.98
$G_{\#0}$	25.96	$G_{\#2}$	103.83	$G_{\#4}$	415.30	$G_{\#6}$	1661.22
A_0	27.50	A_2	110.00	A_4	440.00	A_6	1760.00
$A_{\#0}$	29.14	$A_{\#2}$	116,54	$A_{\#4}$	466.16	$A_{\#6}$	1864.66
B_0	30.87	B_2	123.47	B_4	493.88	B_6	1975.53
C_1	32.70	C_3	130.81	C_5	523.25	C_7	2093.00
$C_{\#1}$	34.65	$C_{\#3}$	138.59	$C_{\#5}$	554.37	$C_{\#7}$	2217.46
D_1	36.71	D_3	146.83	D_5	587.33	D_7	2349.32
$D_{\#1}$	38.89	$D_{\#3}$	155.56	$D_{\#5}$	622.25	$D_{\#7}$	2489.02
E_1	41.20	E_3	164.81	E_5	659.26	E_7	2637.02
F_1	43.65	F_3	174.61	F_5	698.46	F_7	2793.83
$F_{\#1}$	46.25	$F_{\#3}$	185.00	$F_{\#5}$	739.99	$F_{\#7}$	2959.96
G_1	49.00	G_3	196.00	G_5	783.99	G_7	3135.96
$G_{\#1}$	51.91	$G_{\#3}$	207.65	$G_{\#5}$	830.61	$G_{\#7}$	3322.44
A_1	55.00	A_3	220.00	A_5	880.00	A_7	3520.00
$A_{\#1}$	58.27	$A_{\#3}$	233.08	$A_{\#5}$	932.33	$A_{\#7}$	3729.31
B_1	61.74	B_3	246.94	B_5	987.77	B_7	3951.07
						C_8	4186.01

T. C.
Yükseköğretim Kurulu
Dokümantasyon Merkezi