

1-1-1980

# Design and simulation of an experimental microcomputer-based instructional system for music.

Irwin Stuart Smith

*University of Massachusetts Amherst*

Follow this and additional works at: [https://scholarworks.umass.edu/dissertations\\_1](https://scholarworks.umass.edu/dissertations_1)

---

## Recommended Citation

Smith, Irwin Stuart, "Design and simulation of an experimental microcomputer-based instructional system for music." (1980).  
*Doctoral Dissertations 1896 - February 2014*. 3621.  
[https://scholarworks.umass.edu/dissertations\\_1/3621](https://scholarworks.umass.edu/dissertations_1/3621)

This Open Access Dissertation is brought to you for free and open access by ScholarWorks@UMass Amherst. It has been accepted for inclusion in Doctoral Dissertations 1896 - February 2014 by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact [scholarworks@library.umass.edu](mailto:scholarworks@library.umass.edu).



312066013586586

DESIGN AND SIMULATION  
OF AN  
EXPERIMENTAL MICROCOMPUTER-BASED  
INSTRUCTIONAL SYSTEM FOR MUSIC

A Dissertation Presented

By

IRWIN STUART SMITH

Submitted to the Graduate School of the  
University of Massachusetts in partial fulfillment  
of the requirements for the degree of

DOCTOR OF EDUCATION

February 1980

School of Education



Irwin Stuart Smith

1980

All Rights Reserved

DESIGN AND SIMULATION  
OF AN  
EXPERIMENTAL MICROCOMPUTER-BASED  
INSTRUCTIONAL SYSTEM FOR MUSIC

A Dissertation Presented

By

IRWIN STUART SMITH

Approved as to style and content by:

*Howard A. Peelle*

Howard A. Peelle, Chairperson

*Portia C. Elliott*

Portia C. Elliott, Member

*Robert W. Mallery*

Robert W. Mallery, Member

*Mario Fantini*

Mario Fantini, Dean  
School of Education

To Betty, Marge, and Irwin

## A C K N O W L E D G M E N T S

Somewhat to my surprise, the pursuit of an advanced degree has turned out to be not at all an unpleasant experience. I attribute this happy circumstance in no small measure to the many able and congenial people with whom I have had an opportunity to work. Several of these people made very significant contributions to this project, and I want to thank them here for their efforts on my behalf.

Howard Peelle, chairman of my dissertation committee, offered dozens of excellent ideas for all aspects of the music system. His timely and perceptive criticisms of the project as it developed spared me what otherwise would have been countless hours of fruitless labor. Committee members Portia Elliott and Robert Mallary, who brought two quite different and uniquely valuable points of view to this work, gave me sound practical advice throughout the project. Jake Epstein, who graciously agreed to serve as an unofficial consultant on this project, added a much-needed professional musician's perspective on my activities and ideas.

My friend David Wright, a software engineer with the Digital Equipment Corporation, suggested several important simplifications of the internal architecture of the original music system design. Conrad Wogrin, Director of the University Computing Center, and UCC staff members Jim Burrill, Pat Driscoll, Judy Smith, and Clark Wiedmann helped me solve the inevitable technical problems that arose during the project. Susan Lander typed the intermediate and final drafts of the dissertation and also served as expert advisor

on matters of manuscript form and layout. My mother, Marge Smith, read the next-to-final draft and did a thorough critique of the entire text. Her emendations of the latter make the present document considerably more readable than my original.

I want to give special thanks to those of my students who participated in the pilot studies; these kids took their roles as consultants and test subjects seriously and provided me with a great deal of useful information and encouragement. I also want to thank my boss, Tom Elliot, Dean of ULowell's College of Music, and John Ogasapian, chairman of the College's Academic Studies department, for helping to arrange my sabbatical leave during the 1977-78 academic year and for making the adjustments in my 1978-79 duty assignment that allowed me to complete this project on schedule.



ABSTRACT

Design and Simulation  
of an  
Experimental Microcomputer-Based  
Instructional System for Music

(February, 1980)

Irwin Stuart Smith, B.A., Rutgers University

M.F.A., Brandeis University, Ed.D., University of Massachusetts

Directed by: Dr. Howard A. Peelle

Designs for two computer-based music instruction systems are presented. One design is for a hypothetical hand-calculator-like device, while the other is for a prototype system that was actually built. The prototype simulates the projected hand-held device, and the former served as a vehicle for testing the basic design ideas of the latter.

The proposed "music calculator" can be programmed to provide conventional drill and practice in traditional musical skills, but the resources of the system are organized in such a way that it is not locked into any one instructional mode. Among the features offered by the proposed system are (1) an on-line library of musical pieces that can be accessed by their incipits, (2) an "undo" function-- or "panic button"--that exactly reverses the effects of the last operation performed, and (3) a two-leveled programming mechanism with a set of general purpose music functions which users can employ to create their own programs and with an additional set of specialized functions that courseware authors can use to create lessons and games.

The prototype is a computer-based music system designed to be operated in connection with a time-sharing computer. It consists of a simple four-voice tone generator, a special keyboard and interface, a standard CRT-type computer terminal, and some APL software. Although the prototype simulates most of the essential features of the "music calculator", its behavior differs in several key respects from that planned for the hand-held device. These departures are largely caused by the characteristics of the time-sharing environment in which the prototype was implemented.

Two informal pilot studies conducted with the prototype show that the concept of a powerful instructional computer music system has genuine appeal for the two groups of college students who participated in the studies. The pilot studies also show, however, that the prototype is not a good model of the projected hand-held device and that a better research tool is needed before the "music calculator" idea can be developed further.

## TABLE OF CONTENTS

ACKNOWLEDGMENTS . . . . .	v
ABSTRACT . . . . .	vii
LIST OF TABLES . . . . .	xi
ILLUSTRATIONS . . . . .	xii
 Chapter	
I. INTRODUCTION . . . . .	1
Focus of the Project . . . . .	1
Background . . . . .	2
Statement of the Problem . . . . .	5
Proposed Solution to the Problem . . . . .	7
Scope and Activities . . . . .	11
II. REVIEW OF THE LITERATURE . . . . .	15
Introduction . . . . .	15
Systems for Instruction in Traditional Music Subjects. . . . .	16
Composition-Oriented Systems . . . . .	19
"Responsive Environment" Systems . . . . .	24
Miscellaneous Music Devices. . . . .	29
III. DESIGN . . . . .	31
Introduction . . . . .	31
The Theoretical System . . . . .	35
The Prototype . . . . .	64
Unresolved Design Issues . . . . .	81
IV. TESTING: METHODS AND RESULTS . . . . .	87
Introduction . . . . .	87
Pilot Study I . . . . .	89
Pilot Study II . . . . .	103
Concluding Note . . . . .	109
V. SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS FOR FURTHER RESEARCH . . . . .	111
Summary . . . . .	111
Conclusions . . . . .	112
Recommendations for Further Research . . . . .	118

Table of Contents (continued)

BIBLIOGRAPHY . . . . .	121
Appendix	
A. MUSIC FUNCTION SUMMARY . . . . .	127
B. LISTINGS OF THE APL SIMULATION . . . . .	148
C. PILOT STUDY QUESTIONNAIRE . . . . .	177
D. SUGGESTED APPLICATIONS OF THE MUSIC SYSTEM . . . . .	181
E. LIBRARY AND PROGRAM CATALOGS . . . . .	184
F. PROGRAMMING EXAMPLES . . . . .	187

LIST OF TABLES

1. Operational Modes of the Music System . . . . .	52
2. Identifiers Used in the APL Simulation . . . . .	75

## ILLUSTRATIONS

### Figure

1. The Theoretical System . . . . .	37
2. Flowchart of the Executive Routine . . . . .	60
3. APL Version of the Executive Routine . . . . .	61
4. Prototype Keyboard Layout . . . . .	71

### Photograph

1. The GUIDO Intervals Lesson Display . . . . .	18
2. The Prototype. . . . .	68

# C H A P T E R I

## INTRODUCTION

### Focus of the Project

The principal focus of the project described here is the design of a completely self-contained, hand-calculator-sized music instruction system. Although the proposed system can be programmed to provide conventional drill and practice in traditional musical skills, the resources of the system are organized in such a way that it is not locked into any one instructional mode. People can use the system in whatever way they wish to learn and explore music on their own.

Since the calculator-sized music system cannot be realized with existing electronic components, this project also focused on the design, construction, and testing of a prototype music system to simulate the hand-held device. The prototype served as a test vehicle for the basic design ideas of the hand-held music system. It was also used in two pilot studies that were conducted in order to evaluate the music system design and to gather information that will be helpful in improving it.

The "music calculator" concept provided a readily grasped image that quickly conveyed the essence of this project. As such, it was both goal and point of reference for all those who participated in the project, and it was a useful guide in organizing and directing the diverse activities that had to be undertaken. Consequently, although the actual music calculator could not be brought into



being, the concept of such a device remains the central unifying idea of the work reported here.

### Background

Over the past twelve years there has been a growing effort to develop computer-based instructional systems for music. Several of the major research projects in this area have produced systems which are now in actual use as part of regular college-level programs in music (see, e.g., the summary in Hofstetter, 1979b). On the whole, however, the computer has so far had a negligible impact on musical instruction.

Some idea of the extent of computer use in musical instruction can be gained from three recent studies. Jones (1976) conducted a survey to determine the status of computer-assisted instruction within the 429 colleges and universities then accredited by the National Association of Schools of Music (NASM). Of the 389 schools that responded to the survey, only fourteen said that they employ computer-assisted instruction. Arenson (1978) conducted a survey to determine what equipment is available for computer-based musical instruction at the schools of each of the 139 members of the National Consortium for Computer-Based Musical Instruction (NCCBMI). Although twenty-eight schools indicated that they have an appropriate computer, only sixteen said that they have a computer-controlled audio device suitable for musical instruction. Finally, Taylor and Parrish (1978) conducted a nation-wide study of attitudes toward, and the uses of



computer-assisted instruction in public schools and in college music departments. They found that 79% of the respondents among both the public school districts and the college music departments do not use a computer for any purpose, and that only 43% of the school districts and 31% of the college music departments believed the computer to be a necessary instructional tool for music.

It is evident from studies like these that the computer does in fact play a very small role in musical instruction at present. A significant reason that the computer has not made more headway as an instructional tool is that many educators are simply unaware of its potential uses. Taylor and Parrish make the following observation based on their analysis of the data gathered during the study mentioned above:

An important result of this study was the very strong indication that a large number of respondents had little understanding of the computer and its applications in music education. This was not the case for programmed instruction. But it is not particularly surprising that music educators do not know how to deal with the computer--it has not been considered an essential tool in their profession, as it has been in the sciences and business (1978, p. 20).

Lack of "computer literacy" is by no means the only reason that computer-based instructional systems are not more widely used in music. Other factors responsible for slowing the diffusion of computer-based musical instruction are the following:

- 1) computer-based music systems of all kinds are expensive.

Except for devices that are properly considered toys (Parker Brothers' Merlin, for example), existing computer-based music systems are

priced out of the reach of ordinary individuals and, in many cases, out of the reach of educational institutions as well. Two of the best and most widely available computer-based music systems--the SYNCLAVIER and GUIDO--are a case in point. The SYNCLAVIER (Alonso, Appleton, and Jones, 1977) is oriented mainly toward the learning of techniques of musical composition. The purchase price of a minimum configuration that can be used by one person at a time is about \$15,000 as of this writing. The GUIDO system (Hofstetter, 1975) is a PLATO-based ear-training facility. At the University of Delaware, where GUIDO was originally developed, it costs \$3,850 per year to operate each of eight student terminals, for a total cost of \$30,800 per year (Hofstetter, 1979b).

2) computer-based music systems require special knowledge and skills. Many computer-based music systems require of their users knowledge and skills unrelated to the musical tasks to be accomplished. For example, some typing ability is needed in order to use the computer terminals associated with several of the systems discussed in Chapter II. Some of these systems also require their users to know something about the operating system of the host computer and about certain programming languages, the text editor, the file system, etc. Much of this is of course completely unfamiliar to musicians and music educators, many of whom regard computer-based music systems as being difficult to use. The twin concerns of high cost and difficulty of use are a familiar motif running through

studies of computer use in music from the earliest (Ihrke, 1972) to the most recent (Jones, 1976; Arenson, 1978; and Taylor and Parrish, 1978).

3) many computer-based music systems are not "portable".

Many computer-based music systems are inextricably embedded in the unique set of conditions at the installations where they were developed. That is, they are tied to a specific make and model of computer, or to a particular programming language, or to a special piece of hardware used for music, and so on. As a result it is difficult or impossible to duplicate such music facilities elsewhere. For example, the Stanford ear-training system (Herrold, 1974; Kuhn, 1974; Killam, Lorton, and Schubert, 1975) is written in the relatively rare SAIL language. It also uses an expensive electronic organ and custom interface. In addition this system ties up two time-sharing ports on the host computer, one for the student terminal and one for the organ. Although this system does work rather well, it would be very difficult to set up anywhere else. Several of the systems described in Chapter II present similar situations.

#### Statement of the Problem

While systems like SYNCLAVIER and GUIDO have essentially solved the portability problem and made a good start on the difficulty-of-use problem, no existing computer-based music system can be said to have overcome the cost problem. The purchase price and/or annual

operating cost of every system reported in the literature to date is measured in the thousands of dollars. Clearly if the potential of the computer as an educational tool in music is to be realized, a determined effort must be made to bring the cost down, but without sacrificing the capabilities that have attracted musicians and music educators to the computer in the first place.

The problem to be addressed here, then, is how to produce a computer-based musical instruction system that is simultaneously low in cost, easy to use, and portable. In order to be considered a "solution" to this problem, a system should meet the following requirements:

- 1) It must support conventional instructional applications, particularly ear-training, but it must also permit compositional activities and free experimentation with music. Formal "lesson" programs must not be hard-wired into the system but rather implemented through some more general programming mechanism that is also available to users for creating their own programs.

- 2) It must not require any special skills such as typing ability or facility on a musical keyboard. Furthermore the system must not require any special knowledge of computers or computer programming.

- 3) It must cost no more than, say, a basic home computer system or scientific calculator (\$500 or less).

### Proposed Solution to the Problem

The solution proposed to the problem outlined above is a completely self-contained, hand-calculator-sized computer music system. The proposed system has its own multi-voice music synthesizer, keyboard, and operating controls. It also provides a number of single-keystroke music functions and the means for running both pre-programmed and user-defined lessons, games, exercises, etc.

The model for the proposed solution. The model underlying the proposed solution is the ubiquitous mathematical hand-calculator. The calculator is of course inherently portable and, judged by the overwhelming number of examples around us, can be made and sold at a price that individuals can afford. Moreover the calculator is evidently a device that people find easy to use.

Instructional systems have already been successfully developed using this approach. Perhaps the best examples are the calculator-like "electronic learning aids" manufactured by Texas Instruments (Texas Instruments, 1978). Among these are Speak and Spell, Spelling B, Dataman, and Li'l Professor. Speak and Spell is especially significant for this project since it contains a complete speech synthesizer and a pre-programmed vocabulary of over 200 words, features resembling those that would be needed in a hand-held music system.

The key characteristic of both the mathematical calculator and the electronic learning aids is that they are special-purpose



devices. They are tailored to specific applications and groups of users. Also they economize by providing only those resources actually needed to accomplish their designed functions. Virtually all existing music systems, however, are based in general-purpose computers--often large mainframes. Consequently there is usually a good deal of expensive "overhead" in the form of excess computational capacity and unused features in these systems. Moreover, unless the creator of the system has been extraordinarily thorough, the user of a system based in a general-purpose computer must contend directly with some of the technical aspects of the computer system itself (e.g., the operating system, the file system, a text editor, language translators, etc.). For all of these reasons, then, the calculator provides a more attractive model to follow.

Features of the proposed system. The system proposed here has a short (1-octave) musical keyboard used to enter notes into the system's memory, and an internal music synthesizer for playing stored musical pieces. The system provides a number of single-keystroke compositional functions that can be used to modify existing musical pieces or to create entirely new ones. The system has access to libraries of musical compositions and application programs which are stored externally in interchangeable plug-in memory modules. In addition, the proposed system has three features not found in any computer music system as of this writing:

1) a library of musical compositions accessed through a "hum a few bars" retrieval scheme. In order to retrieve a musical piece from the system's library, the user need not look up a file name or file number (although the system does provide a conventional numbered file retrieval feature). Instead the user can simply key in the first few notes of the desired composition and then have the system find the corresponding piece automatically. Since the system attempts to find the best match between the user-supplied notes and the items listed in the library catalog, mistakes in the user's input will not necessarily prevent the system from finding the piece.

2) an "undo" operation. The "undo" operation permits users to exactly reverse the effects of the last operation executed. If, for example, while working on a complex musical piece, a user inadvertently invokes some procedure which damages or destroys the piece, the user can restore the piece to its original form simply by pressing the "undo" button.

3) two levels of programmability. The music system can be programmed to perform entire sequences of actions automatically. Users can construct their own programs from any valid combination of the functions and data accessible via the keys on the device's front panel. The system also has a second group of specialized, or "privileged" functions that professional programmers can employ to create complex games and lessons.

Feasibility of the proposed system. As noted at the outset, the hand-held system cannot be built at this time. The principal

reason is that the tiny music synthesizer required by the design does not yet exist. In fact, as of this writing, it takes upwards of twenty individual integrated circuits to implement a barely adequate four-voice tone generator. Of course this number of integrated circuits would by itself fill up most of the room inside the system's case.

A secondary problem is that at present the large memory capacity required to implement all of the desired functions of the music system cannot be provided within a calculator-sized case at reasonable cost. A potentially suitable memory component does exist, Texas Instruments' TMS-4164 (which is capable of storing 65,536 bits of information). However, these are currently priced at \$125 each, and eight of them are required to form the complete memory unit. Obviously this component alone would drive the price of the hand-held unit out of reach and thus defeat one of the main design goals: low cost.

Most of the remaining components--microprocessors, displays, pre-programmable memory modules, etc.--needed to realize at least a preliminary version of the hand-held music system are available as standard parts. Since both the sound synthesis and memory technologies needed to complete the device are being developed right now (see, e.g., Computer Music Journal, passim, and Hodges, 1977), it is reasonable to predict that it will be possible to build a completely self-contained, calculator-sized music system some time in the mid-1980's.



Simulation of the proposed system. For the purposes of this project, a prototype music system was designed and built to simulate the hand-held system. The prototype consists of a simple four-voice tone generator, a special keyboard and interface, an ordinary CRT computer terminal, and some APL software. The prototype permits entry of music via its 1-octave musical keyboard. It can play music consisting of up to four independent voices. The operation of the prototype's hardware is controlled by APL programs running on the Cyber 175 time-sharing computer system at the University of Massachusetts-Amherst. The APL routines provide all of the musical and data processing functions attributed to the final calculator-sized device, and APL files provide the necessary libraries of musical compositions and application programs (games, lessons, etc.). A full description of the prototype, its use, and the differences between it and the ultimate system envisioned in the design is given in Chapter III.

#### Scope and Activities

The overall process of producing the music system design can be resolved into three distinct, but interrelated tasks:

- 1) design of the abstract structure of the music system
- 2) physical realization of the prototype
- 3) testing of the prototype

Below are a summary of the methodologies used at each step and an overview of the actual activities undertaken.

Design of the abstract structure. The design of the abstract structure of the music system employed both "top down" and "bottom up" strategies. On the one hand, the author started with a very general idea of how the system should be structured and how it should work, and then gradually filled in the details at progressively lower levels. On the other hand, the author also initially developed a list of the desirable functions and then progressively integrated these into a complete system.

When this preliminary design work was completed, both the "top down" and "bottom up" structures were translated into APL functions that could be executed on the University of Massachusetts' time-sharing system (the "top down" structure became what in Chapter III is called the "executive routine", and the "bottom up" structures became the "regular" and "privileged" functions of the music system). Together these APL functions constitute a simulation of the music system.

Having all of the music system functions available in the form of executable APL routines made it possible to interact with the hypothetical music system and observe its behavior. Thus the APL simulation became the most important tool for the further development of the music system design. Moreover, since the APL simulation provided the most comprehensive definition of every aspect of the design, changes in the simulation were therefore equivalent to changes in the design itself.

Physical realization of the prototype. The three principal tasks involved in producing the prototype were:

- 1) designing a keyboard suitable for research purposes
- 2) building the actual keyboard and a simple synthesizer for audio output
- 3) substituting the prototype keyboard for the standard computer terminal as the principal device used to interact with the APL simulation

Except for the first task, the design of the keyboard, this work was all straightforward. The design of the keyboard, however, required the resolution of many issues. Among these were the most appropriate physical form for each control, the proper grouping and spacing of sets of controls, and the best way to indicate the function of each control. All of these issues and the solutions finally adopted are discussed in detail in Chapter III.

The prototype system is primarily a research tool, not the penultimate step in the development of a marketable product. The prototype has certain physical and functional characteristics that differ from those envisioned for the final hand-held music system. These differences must be taken into account in drawing conclusions about either system.

Testing of the prototype. Two pilot studies were conducted as part of the overall design process. Each of these studies employed a group of college students who were asked to perform a set of typical musical tasks with the prototype. These studies did not set out

to prove or disprove any specific hypotheses concerning the music system. Instead they were intended to elicit information that would be helpful in evaluating the design and in furthering its development. The results of the pilot studies are therefore not necessarily generalizable outside the confines of this project.

Three principal kinds of data were gathered during the pilot studies:

- 1) spontaneous comments by users. These were noted down as the subjects worked with the prototype.
- 2) answers to specific questions. Each subject filled out a questionnaire that asks about the ease or difficulty of using certain system features, the need for changes or improvements in the system, the possible uses of the system, etc.
- 3) the actual interactions with the system. The prototype has a record-keeping facility which automatically records the date and time of every session and every interaction with the system during each session. It also tags any erroneous operations (e.g., errors in syntax or attempts at "illegal" procedures).

The methods and results of both pilot studies are covered in detail in Chapter IV.

C H A P T E R   I I  
REVIEW OF THE LITERATURE

Introduction

Since the earliest projects in the late 1960's, work in instructional applications of computers in music has proceeded in three different directions:

- 1) enhancement of instruction in traditional music subject areas such as basic terminology, musical notation, and ear-training
- 2) development of systems to facilitate the learning of compositional techniques
- 3) creation of "responsive environments" in which people can learn and explore music on their own

While these three categories of activity are not mutually exclusive, most of the existing instructional applications of computers in music do fall quite clearly into one or another of them. Although there are indications of growing interaction and coalescence of interests among workers in these three areas, certainly the grand synthesis of the principal achievements of these areas into the "complete computer music system" envisioned by Peters (1977) has not yet been accomplished.

The music system design reported in this study belongs primarily to the third, "responsive environment" category. Work in each of the other areas has had an influence on the evolution of this



design, however. Accordingly this chapter summarizes the relevant work in all three areas.

Systems for Computer-Based Instruction  
in Traditional Music Subjects

Over the past decade, several schools have developed computer-based instructional systems for music and have introduced them as a regular part of their curricular programs. These systems provide instruction in one or more of the following areas of the usual college music curriculum:

- 1) music fundamentals (notation, terminology, scale and chord structures, etc.)
- 2) ear-training (aural recognition of intervals, chords, melodic and rhythmic patterns, etc.)
- 3) teacher training (methods, tests and measurement, etc.)

A comprehensive survey of these instructional systems can be found in Hofstetter (1979b). With two exceptions, these systems are implemented on large time-sharing computers. The two exceptions are the CLEF system (Hultberg, Hultberg, and Tenny, 1979), which is based in a dedicated minicomputer, and AVICOM (Peters, 1979), which is a microprocessor-based system. Most of the systems listed by Hofstetter employ drill and practice as the dominant instructional mode. Reports of the educational performance of the principal instructional systems can be found in Deihl (1971), Deihl and Ziegler (1973), Peters (1975), Placek (1974), Kuhn (1974), Hofstetter

(1975, 1976, 1977a, 1977b, 1978, 1979a), Herrold (1973), Killam, Lorton, and Schubert (1975), Vaughn (1977), and Arenson (1979).

Most of the workers in this field belong to the National Consortium for Computer-Based Musical Instruction (NCCBMI), a special interest group within the Association for the Development of Computer-based Instructional Systems (ADCIS). NCCBMI members regularly report their activities in the ADCIS Newsletter; many members also publish the results of their research in ADCIS' Journal of Computer-Based Instruction.

A selected bibliography of the entire field of computer applications in musical instruction can be found in Peters and Eddins (1978).

Since most of the work in this field has had very little direct influence on the music system project reported here, this work will not be described further. However, the subject of computer-based instruction in traditional music subject areas could not be left without mentioning the unique PLATO-based "GUIDO" system developed by Hofstetter (1975). GUIDO (Graded Units for Interactive Dictation Operations) is a set of musical games and exercises designed to teach basic aural skills. GUIDO takes full advantage of the resources provided by PLATO to achieve its objectives.

Photograph 1 shows the "GAME" display used with the GUIDO intervals lesson (i.e., this is the picture students see in the screen of the PLATO terminal). In this GUIDO lesson, the intervals to be identified by the students are played on a synthesizer connected

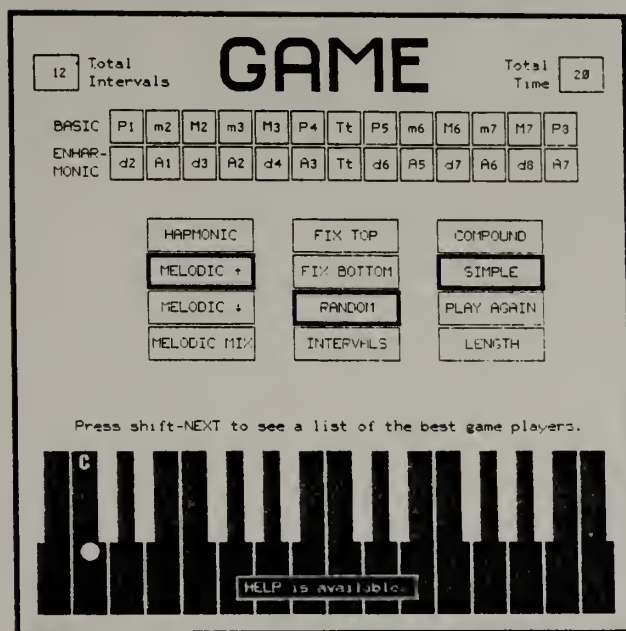


Photo. 1. The GUIDO intervals lesson display. © 1977 by the University of Delaware. Authored by Fred T. Hofstetter and William H. Lynch (used by permission).



to the PLATO terminal. Students enter their responses by touching the appropriate answer boxes on the GAME display, which employs the PLATO "touch" panel. The display brightens around each box touched to confirm that the answer has been received by the system. The box in the upper left corner maintains a running total of the number of items attempted, and the box in the upper right corner shows the total amount of time used to guess all items so far. Students can also touch certain "control" boxes to make the problems easier or to make them more difficult, to move to new material, or to repeat the same items.

A novel feature of the GAME display is the musical keyboard at the bottom: students can play it just like a real one. It is available at all times as a diagnostic aid, and it can be used in many other ways, too. Students are free to employ it in whatever manner they feel will help them with their musical problems.

As of this writing, GUIDO represents the state of the art in computer-based instruction in traditional music subjects. All of the lessons are well thought out. The GAME and other displays are exemplary for their economy and clarity.

#### Composition-Oriented Systems

Ever since the origins of "computer music" in the late 1950's, great effort has been exerted in the development of powerful computer-based sound-generating facilities for composers. As a result there is now a large inventory of technologies which offer

the composer an unprecedentedly rich palette of musical sonorities and effects. The digital computer has in fact become one of the most versatile and precise instruments composers have ever had for translating their musical ideas into sound.

In the last few years, work in the field of computer music has gradually broadened its scope to include the entire process of creating musical works. Consequently, in addition to the basic sound synthesis capability, many computer music systems now also provide other facilities to assist the composer in his task. Among these aids are interactive editing of musical scores, automatic score printing (of publishable quality), direct capture and display of input from musical keyboards, entry of music in traditional staff notation, automatic compositional manipulations, and analysis of acoustic waveforms.

The literature of computer music is so extensive that any attempt to summarize it would go well beyond the scope of this survey. A selected bibliography of the field can be found in Snell (1977). Since 1977 the major work in this field has been reported in Computer Music Journal (CMJ). In addition to publishing new work CMJ from time to time reprints relevant documents from other publications. The seminal work in this field is Mathews' The Technology of Computer Music (1969), which describes the fundamental techniques of computer sound synthesis.

Most of the work in computer music has not been oriented towards instructional applications but rather toward musical

composition. However, computer music systems are used increasingly as a regular part of college-level music composition, electronic music, and advanced music theory courses. Moreover the newer computer music systems contain features which were originally designed as aids to the composer but which make these systems attractive for a variety of educational uses as well. Two systems in particular, the SYNCLAVIER and MPL, lend themselves especially well to instructional uses in both musical composition and other areas.

The SYNCLAVIER (Alonso, Appleton, and Jones, 1977) is produced by the New England Digital Corporation, of Norwich, Vermont. It includes a complete minicomputer system, an 8-voice real-time digital sound synthesizer, a standard 5-octave organ keyboard, and a set of 90 controls used to set the operating parameters of the machine and to store and retrieve musical compositions. The SYNCLAVIER is an interactive facility that allows the user to create compositions by playing successive voices (or "tracks") on the organ keyboard. Notes are stored in the system's memory as they are played. Since each of the voices (tracks) can be manipulated independently of the others, it is relatively easy to make corrections on, additions to, or deletions from a work in progress. A handy feature of the SYNCLAVIER is that playback tempo can be changed without affecting the pitches of the notes. As a result a user with limited keyboard skill can play very slowly while entering the individual parts of a composition but then have the system play the final piece back at any desired faster tempo. Since all of the SYNCLAVIER's functions

are directly accessible via the 90 operating controls, there is no need for students to learn a programming language before they can use the system. The minicomputer system which is at the heart of the SYNCLAVIER is provided with a compiler for the XPL language and a set of subroutines to control the synthesizer, however. Thus many other kinds of musical applications can be implemented simply by writing the appropriate programs.

MPL (Musical Program Library) is a comprehensive computer music system implemented on the Xerox/Honeywell time-sharing system at Oberlin College by Gary Nelson (1977). MPL offers quadraphonic sound synthesis, interactive musical score editing, automatic score printing, and a number of musical composition and analysis functions. The key feature of MPL is that almost the entire system is written in APL. This gives the user access to both the full musical resources of MPL and all the resources of APL in a single, unified interactive environment. Further, because APL is the host language, MPL inherently has the desirable characteristic that users who know APL can write their own custom interfaces ("front ends") to the system. One drawback of MPL, however, is that it does not generate sound in real time. Instead there is a variable delay for computation between the time a musical score is given to the computer and the time the final output is played. The actual length of the delay depends on both the length and the complexity of the composition.

One other composition-oriented system should be mentioned here, Harold Alles' Portable Digital Sound Synthesis System (Alles, 1977a, 1977b, 1977c; Alles and diGiugno, 1977; Lawson and Mathews, 1977; Bayer, 1977). This extremely powerful system is quite possibly the first member of a whole new family of musical instruments: small, portable devices capable of producing in real time sound approaching the complexity of a modest orchestra. The Alles machine accomplishes this through compact and very high speed digital modules such as its synthesizer, which contains 64 complete Chowning-type FM "voices" on one 8-1/2" x 10" circuit board. Other essential sound generating and processing functions are packed in a similarly dense fashion into the system. A performer plays this machine in much the same manner as any other keyboard instrument, except that he or she can program its internal computer to carry out certain operations automatically on command (e.g., to execute a passage that is too difficult for the performer to play). The entire machine weighs about 300 pounds, including its integral CRT, ASCII keyboard, dual floppy disk drives, two musical keyboards, and complement of real-time operating controls. In size, weight, and portability it is comparable to many of the vastly less powerful electronic keyboard instruments that rock bands routinely travel with (e.g., the ubiquitous Hammond "B-3" organ).



### "Responsive Environment" Systems

The term "responsive environment" is due to David Ashton (1973), who defines a responsive environment as one which "permits the learner to explore freely, to freely manipulate objects, to receive immediate feedback, to make full use of his capacity for discovering relations of various kinds, and to progress at his own speed" (1973, p. 1). This is an apt description of the ideals which motivated the design of several learner-oriented music systems, including of course the one David Ashton himself helped to design.

Ashton's definition immediately distinguishes responsive environments from the traditional musical instruction systems described above. In all of the latter, the major emphasis has been placed on careful design of lessons and on analysis of student performance. All of these systems (including GUIDO) teach specific, isolated, discrete bits of musical knowledge and specific musical skills. The student has little control over the choice of subject matter, lesson sequence, or the directions taken within each lesson.

The definition less clearly distinguishes the responsive environment from the composition-oriented systems, however. The most important practical differences are ones of emphasis. In composition-oriented systems, for example, the ability to synthesize sound in real time has often been sacrificed for the sake of achieving the most powerful sound synthesis capability. In responsive environments, on the other hand, a lesser sound synthesis

capability has always been accepted so that music can be played immediately and modified interactively.

A responsive environment can be realized in many ways. The following features, however, appear to be the minimum any such system must possess:

- 1) interactive operating environment. In order to permit truly free exploration, the user must be able to "converse" easily with the system and must have rapid access to all pertinent system resources. An especially important requirement is that the user be able to interrupt any system operation and to re-direct the system's activities into other desired directions.
- 2) real-time sound generation. In order to achieve the educational benefits of prompt feedback, sound must be available immediately on demand. The sound need not be of the highest quality, however.
- 3) on-line "archive" of music and a set of functions for manipulating music. Together these two features allow learners to play and manipulate whole musical structures without having first to acquire facility on a musical instrument; learners can proceed directly to fairly high-level musical operations. Without this pair of features, any computer music system remains essentially a tool for specialists.

The three music systems described below have all three of these features in common.

The music system for which the term "responsive environment" was coined was developed by A.C. Ashton (1970), Knowlton (1971), and D. Ashton (1971) at the University of Utah. This system is now a continuing joint endeavor of the University of Utah and Brigham Young University. A concise discussion of the design and uses of this system can be found in Knowlton (1972).

The Utah/BYU system combines two minicomputers, a CRT terminal, a graphics stylus and tablet, and an electronic organ. Music can be entered into the system in any of three ways:

- 1) by playing on the organ keyboard
- 2) by pointing to staff positions on the graphics tablet
- 3) by typing an encoded form of the music on the terminal

The system provides a variety of interactive musical transformations such as transposition, tempo change, etc. Music output can be in any of three forms:

- 1) sound (the electronic organ plays the composition under computer control)
- 2) standard score (the computer generates the ordinary musical notation for the piece and displays it on the CRT or prints a hard copy)
- 3) graphic score (the computer makes an X-Y plot of the music, similar in appearance to a piano roll, and displays it on



the CRT or prints a hard copy. The CRT graphic score can be displayed in real time, i.e., while music is playing) The Utah/BYU system has an archive of more than 100 musical works which can be accessed for performance and manipulation.

Pietro Grossi and a group at the National University Computing Center (CNUCE) in Pisa, Italy, have developed three interactive music systems over the past decade. The two earlier systems, developed between 1969 and 1975, are discussed in Baruzzi, Grossi, and Milani (1975), and Grossi and Sommi (1974). The most recent system, called "TAUMUS", is described in Grossi (1976). TAUMUS employs a portable 12-voice "audio terminal". Thus, although TAUMUS is based in a large IBM 370 time-sharing system located in Pisa, Grossi has been able to travel all over Europe giving live demonstrations of the music system.

TAUMUS has a musical archive capable of storing up to three million notes. Since this archive is on-line, users can call up and immediately play either their own stored pieces or any of the hundreds of works from the standard literature stored there. A unique feature of the archive is that more than one file can be retrieved at a time. This means, for example, that two pieces can be called up simultaneously and played either in counterpoint or with notes interleaved (one note from piece "A", followed by one note from piece "B", followed by one from "A", etc.).

TAUMUS offers a set of musical data processing functions. This set of functions includes all of the standard manipulations

of musical material (transposition, inversion, retrograde, etc.) as well as a number of unusual operations such as symmetrical expansion or contraction of the intervals between tones. The system also has some high-level compositional routines which can, for example, systematically produce complex variations of given musical material or automatically generate entire new compositions.

Jeanne Bamberger (1972, 1974a, 1974b, 1975, 1976) has developed a responsive environment capability within LOGO (Papert, 1970, 1972). LOGO itself is a general-purpose computational system which includes the LOGO evaluator, a time-sharing computer system, and various special devices such as robot "turtles", CRT displays, plotters, and a "music box" (a simple 4-voice tone generator). LOGO was designed to introduce beginners to the fundamental ideas of computation. The music features are only a small part of its overall capabilities.

The LOGO language (Abelson, Goodman, and Rudolph, 1974) contains a number of primitive operations for controlling the functions of the music box. These music primitives and the arithmetic, logical, and sequence-control operations of LOGO can be formally combined into procedures to play and transform music. The LOGO file system makes it quite easy to set up libraries (archives) of musical pieces.

Bamberger's approach differs from that of the CNUCE and Utah/BYU groups in that the computer and computer-controlled devices are not always the center of attention. Instead Bamberger uses the computer

music system as only one part of the total learning environment, which also includes bells, drums, a piano, etc.

One additional system that should be mentioned under the "responsive environment" heading is Xerox's "Dynabook" (Xerox PARC/LRG, 1976; Kay, 1977) and its "Smalltalk" programming language (Goldberg and Kay, 1977). If the final goal of the designers is attained, Dynabook will contain all of the features of LOGO, and much more, in a package the size of an ordinary loose-leaf notebook. The present, interim versions of Dynabook are about the size of the typical small-business minicomputer system.

Music can be played, edited, and composed on Dynabook. The system provides for real-time capture and display of music performed on its musical keyboards. Users can also enter music into the system by drawing pitch vs. time "scores" (similar to those of the Utah/BYU system) with the system's graphic input device. Finally, music can also be entered in an encoded form on Dynabook's alphanumeric keyboard. Once music has been entered into the system, it can be displayed on the CRT screen and edited with the graphic input device. The "Smalltalk" language provides ample capabilities for writing procedures that will perform interesting manipulations of musical material.

#### Miscellaneous Music Devices

Two recently developed devices deserve mention here if for no other reason than that they are small and inexpensive. The first is

Parker Brothers' microprocessor-based game, Merlin, a hand-held device about the size of an ordinary telephone handset. Among the games offered by Merlin is one which requires the user to match a sequence of random pitches played on the unit's internal tone generator. After hearing each sequence, the user responds by touching the keys corresponding to the notes just played. Merlin then indicates if and when the user gets a note wrong. The user can replay a pitch sequence as many times as desired and can set sequences to any length up to a maximum of 48 notes. If its pitches were a bit more accurate, Merlin could be used as a rudimentary ear-training system.

The other device that should be mentioned here is Videobrain, a home microcomputer-based television game manufactured by Umtech, Inc. Though Videobrain can be programmed by the user in a hybrid language unique to the system, it is designed mainly to run a variety of professionally-written games and application programs stored on plug-in program modules. One of these program modules contains a set of four lessons in music fundamentals. These lessons, written by Wolfgang Kuhn of Stanford, play tones over the television's loudspeaker and also display notes in traditional staff notation on the TV screen.

## C H A P T E R   I I I

### DESIGN

#### Introduction

This chapter presents designs for two computer-based music systems. One of these systems is a first approximation to the self-contained, hand-held device that is the ultimate goal of the work reported in this study. This system will be called the "theoretical system" here because, for the reasons enumerated in the first chapter, it cannot be built at this time; it exists only in the form of the description given in this chapter. The other system, an interim facility used for research, was actually built and tested during the 1978-1979 academic year. This system will be called the "prototype".

Design principles. Two general principles have guided the design of both music systems to this point. They are: (1) keep the system simple, and (2) tailor every aspect of the system to the non-music-specialist. The first principle entered the design process as a persistent effort to limit the knowledge required to operate the system to a small number of uncomplicated rules and to stay as close as possible to familiar musical concepts and usages. The second principle manifested itself as an attempt both to provide practical, layman-oriented musical resources and to avoid esoteric or very specialized features.



Design Issues. The issues which had to be addressed in designing the music systems can be grouped into categories corresponding to the three major stages of the design process, i.e.:

- 1) specification of the set of operations to be performed by the system
- 2) organization of the set of operations into a unified abstract system
- 3) physical realization of the abstract system

In the case of the prototype, all three stages were carried through to completion, while, in the case of the theoretical system, the third stage was carried only as far as a detailed written description, a flowchart, and a drawing of one possible physical embodiment of the abstract structure.

Specification of the function set. In specifying the operations to be performed by the music system, the hard work began after the preliminary list of functions had been developed. With so many existing computer music systems available as models, it was no trouble at all to develop a long list. The central issue at this stage of the design process was the tradeoff between the number of functions it would be desirable for the device to perform and the number of functions that (1) can be mentally grasped as a meaningful whole and (2) can actually be accommodated on a calculator-sized device. The function set must of course be sufficient to perform all of the musical manipulations that could reasonably be expected to be required by the typical instructional and recreational

applications of the device. On the other hand, the set of functions must not be so large as to lead to an overcrowded, over-complex device.

Organization of the system. The second stage, organizing the selected functions into a unified abstract system, involved three distinct but interrelated tasks and their associated design issues, i.e.:

- 1) devising a uniform set of rules for function behavior (e.g., where functions get their arguments from, where they leave their results, what they do when an error condition is detected, etc.)
- 2) devising an overall process within which the functions can operate and communicate with one another (i.e., a "meta-function" or "executive routine" that runs the whole system of functions)
- 3) devising a simple, uniform set of rules for using the functions (syntax)

The abstract system developed at this stage was translated into a working computer-based simulation. The music system functions and the executive routine were written up as APL functions while the various necessary memory components were simulated by APL variables and files. A standard computer terminal acted as keyboard, controls, and display. Since this APL simulation made it possible to interact with a hypothetical music system and observe its behavior, it became a valuable design tool (and arbiter of design issues) for



the remainder of the project. The APL simulation also became the basis of the prototype system.

Physical realization of the system. The third stage, the actual physical realization of the abstract system, involved resolution of the following issues:

- 1) format of controls. Should the control for a given function be implemented in a discrete form (pushbutton, toggle switch, etc.), or in a continuous form (rotating knob, slider, etc.)? Are there particular controls that should be given a special size, shape, or color?
- 2) organization and layout. Which controls should be clustered together, and which should be set apart by themselves? What is the proper spacing and geometric arrangement of the controls?
- 3) identification of functions. How should the function of each control be indicated: should each one be identified by (a) a descriptive word or abbreviation, or (b) a graphic symbol, or (c) a shape-coded control, or (d) some combination of the above, or (e) some entirely different notation?

Since the prototype and the theoretical system differ somewhat in both size and shape, two sets of solutions had to be found for these issues.

Status of the present designs. The theoretical system and the prototype are shown in this chapter in the state of development at which they had arrived by the end of the 1978-1979 academic year. At

this point the prototype had been used by the author, his students and colleagues, and others on an almost daily basis for more than two months. All of the essential features attributed here to the theoretical system were tested in one form or another on the prototype during this period. Similarly the prototype's APL programs are shown with all of the modifications that were made on the basis of the experience gained during the same time. Certain design issues were not resolved in an entirely satisfactory manner during this period, however. The chapter concludes with a discussion of these issues.

### The Theoretical System

The theoretical system is a self-contained computer music system the size of an ordinary hand calculator. It has a short (1-octave) musical keyboard used to enter notes into the system's memory, and an internal multi-voice synthesizer for playing stored musical pieces. The system provides a number of compositional functions that can be used to modify existing musical pieces or to create entirely new ones. The system has access to libraries of musical compositions and application programs which are stored externally in interchangeable plug-in memory modules. In addition the theoretical system has three features not found in any computer music system as of this writing:

- 1) a "hum a few bars" library retrieval scheme. Musical pieces in the library can be accessed by their incipits.

- 2) an "undo" function. The system provides a "panic button" that exactly reverses the effects of the last function executed.
- 3) two levels of programmability. Users can construct their own programs from any valid combination of function and data keys. The system also has a second group of specialized or "privileged" functions that professional programmers can use to create games and lessons.

Physical components of the system. Fig. 1 shows the present conception of the overall physical configuration of the theoretical system. The external features illustrated in this drawing and the major internal hardware components other than the computer are discussed below.

Musical keyboard. The musical keyboard comprises the first three rows of keys along the bottom of the unit. The musical "white notes" (second row from the bottom) are labeled with the appropriate pitch names, while the identities of the "black notes" (third row from the bottom) may be inferred simply from their relative positions. As can be seen, this keyboard spans only one octave. The range of the keyboard can be extended upward one additional octave, however. Keys struck while the "shift" button on the right hand side of the case is depressed will sound one octave higher than unshifted notes. The musical keyboard is always "live" in the sense that tones are always sounded by the system's internal synthesizer when the keys are pressed.

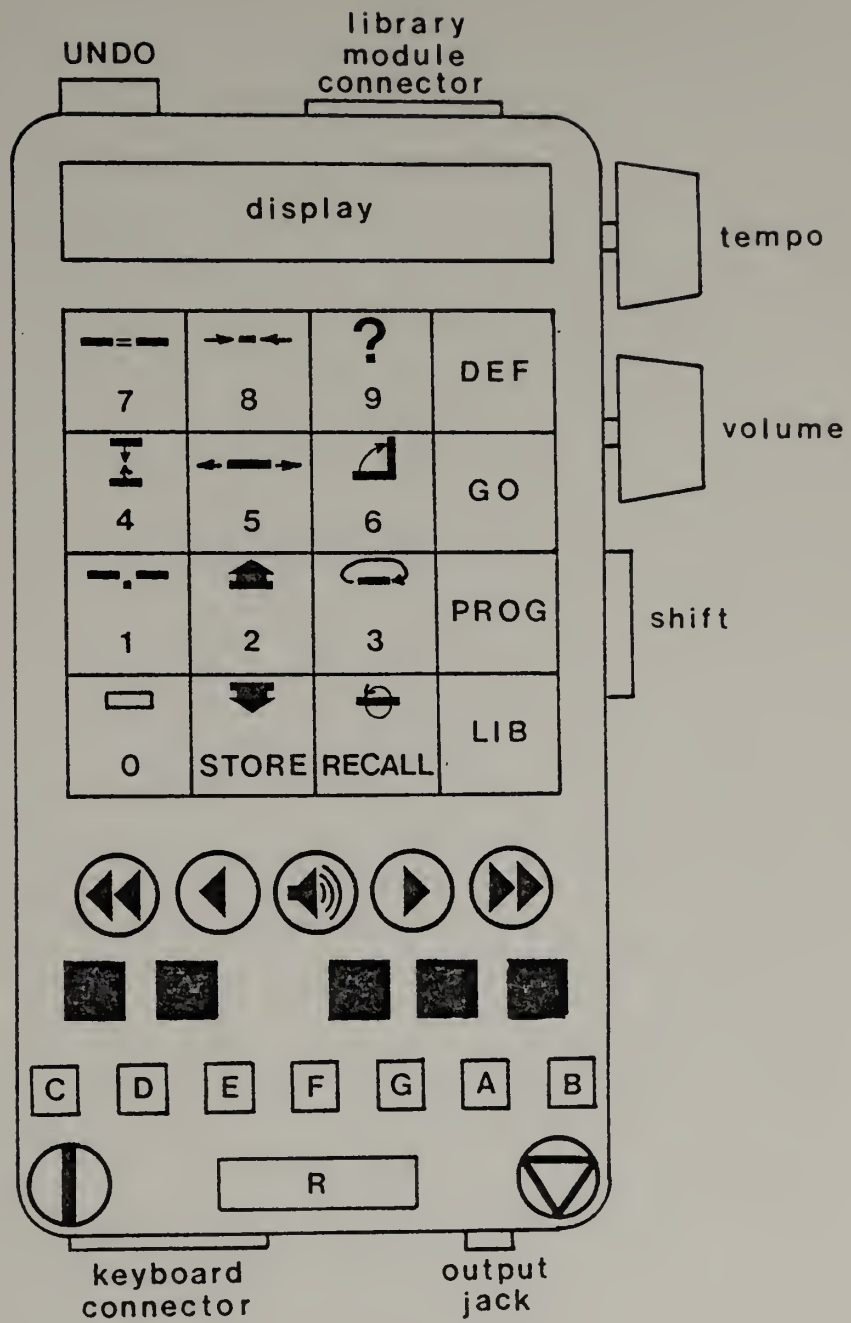


Figure 1. The Theoretical System

The bottom row contains three additional keys essential to the operation of the musical keyboard. The circular button on the left, which activates the DATA function (q.v.), is used to indicate to the system that it should record whatever is played on the keyboard. The circular button on the right, which activates the INSERT function (q.v.), is used to indicate that recording is complete and that no further notes are to be stored. A lamp under the left-hand button is turned on whenever the keyboard is recording.

The third key, marked "R", is for entering rests. The rest key is needed because of the peculiar manner in which the musical keyboard works. When the system is in "data entry mode" (q.v.), a standard 1-beat-long note is entered into the system's memory each time a note key is pressed. The actual length of time a key is held down and the actual length of time between keypresses are not recorded; the notes are simply strung together one after another in memory. The rest key is therefore necessary in order to provide a means for separating notes in time; it is the musical equivalent of the typewriter's space-bar. When the rest key is pressed while the system is in data entry mode, a standard 1-beat-long rest is entered into the system's memory.

The reason for having the musical keyboard work in this way is that it requires none of the manual skills needed to play a conventional musical keyboard. Since this tiny keyboard has very obvious limitations, however, tentative provision has been made for plugging a real musical keyboard into the system.



The PLAY and cursor controls. Immediately above the black notes of the musical keyboard is a set of five circular buttons (fourth row from the bottom). The center button, which bears the loudspeaker symbol, is the PLAY button. Pressing this button causes the system to perform whatever piece of music is currently stored in the "working area" (q.v.) of the system's memory. The other four buttons are used to move the "cursor" (q.v.), an imaginary place marker that can be moved among the notes in the working area.

Function keypad. Above the circular buttons is a 4 x 4 array of keys. These are used to call most of the other functions of the music system and to enter the numeric parameters required by some of the functions. Note that this is a two-level keypad: each of the keys in the three leftmost columns has two distinct functions, as indicated on the upper and lower portions of each key. The "normal" function of each key is the one shown on the lower half of the key (the functions associated with the graphic symbols used on the keys are all described in Appendix A). The function shown on the upper half is selected by holding down the shift button while striking the key. The functions associated with this set of keys are discussed below. (The upper portions of the keys in the rightmost column are spare positions that may be used for functions added in the future).

Display. Directly above the 4 x 4 keypad is a display unit used to communicate various kinds of information to the user. Among the items that may appear in the display are (1) prompts to enter commands or data, (2) pitch and time parameters of notes



played, (3) error messages, (4) elapsed time readouts, and (5) match/no-match indications (which usually will result from correct/incorrect responses to lesson and game problems). The display can be enabled and disabled under program control so that, for example, information about notes being played can be concealed from the user during games and lessons designed to be done entirely "by ear".

The UNDO button. The UNDO button (near the top left corner) is a sort of "panic button": it activates a function that exactly reverses the effects of the previous function executed. Since an inadvertent use of UNDO can be as devastating as the unintentional use of any other system function, this button is placed in a relatively out-of-the-way spot so that it is less likely to be hit accidentally.

Tempo and Volume controls. On the right-hand side of the case are two rotating controls. The Tempo control regulates the playback speed of music performed by the system. When the pointer on this control is straight up, the tempo will be 60 beats per minute. At this setting, the standard 1-beat-long notes and rests entered on the musical keyboard will each be 1 second long. Turning the Tempo control clockwise increases the tempo up to a maximum of about 600 beats per minute, while turning it counter-clockwise decreases the tempo down to a minimum of about 6 beats per minute. The actual playing durations of the standard notes and rests will vary accordingly. The Volume control (immediately below

the Tempo control) serves as both the overall loudness control and the on/off switch.

In the present conception of the theoretical system, the Tempo and Volume controls are the only ones that can affect music while it is being played by the system.

Shift button. As mentioned above, the shift button has two functions: (1) it selects the functions indicated on the upper halves of the keys in the 4 x 4 array, and (2) it transposes the pitches of the notes played on the musical keyboard up one octave.

Output jack. The quality of the sound generated by the tiny synthesizer contained within the system is expected to be quite good. Since no loudspeaker small enough to fit into a unit of the size shown in fig. 1 is capable of reproducing sound of the anticipated quality, no attempt was made to include a loudspeaker in the design. Instead the output of the synthesizer is simply brought out to a jack near the right-hand corner of the case. The user can plug an earphone into this jack or run a patchcord from the jack to an amplifier/loudspeaker system.

External keyboard connector. The external keyboard connector on the bottom left of the case is provided so that the system can be connected to a full-sized musical keyboard and used as a real musical instrument. This is presently considered an optional feature, however, and the details of its implementation have not been worked out.

The tempo clock. The tempo clock, which functions as the "metronome" of the music system, consists of two main components: a variable-frequency oscillator (VFO) and a counter register. The operating frequency of the VFO is set externally with the Tempo control. The output of the VFO is sent directly to the counter register, which is incremented by 1 every time the clock emits a pulse. The register can be cleared to zero at any time, and the current contents of the register can be read at any time.

During musical performances, the tempo clock interrupts the processor every hundredth of a beat. The processor then reads the value in the counter register and compares it with the starting time of the next event (beginning or ending of a note) in the working area. If the two numbers are the same, the processor takes the appropriate action (turning a note on or off). When the tempo clock is not being used to control the speed of a musical performance, it is available for other uses, such as measuring the user's response times during games and lessons.

The synthesizer. The synthesizer generates four independently-controllable musical "voices". These voices each have a range of eight octaves. They are, however, limited to the pitches of the equal-tempered chromatic scale. Each voice has its own envelope shaper, but the actual contour of the envelope is the same for all four voices. An essential feature of the synthesizer is that, once it receives the pitch, duration, and channel assignment data for a

note, it is able to produce the note entirely without further intervention from the computer.

Internal architecture. The final, hand-held version of the music system would undoubtedly be built around a standard general-purpose microprocessor and its associated support components. As a result the algorithms and data structures of the music system would not be directly represented in the physical hardware of the system's computer. Instead major features of the music system would have to be partly or wholly simulated in software. For example, while the data type "integer" could probably be represented directly in the form of the computer's own digital storage "word", it is extremely unlikely that there would be a direct hardware representation of the type "music". Similarly, while the ten branching functions provided by the music system would probably all have fairly close equivalents in the computer's own instruction set, the higher level music functions would all have to be simulated by procedures containing dozens of individual machine instructions.

Accordingly this section describes the conceptual internal structure of the music system, the way things appear to be laid out from the user's standpoint. The manner in which this structure is actually realized will depend on the resources provided by the processor, memory, and other devices ultimately chosen for use in implementing the music system.

Memory. From the user's point of view, the system's memory is divided into the following functionally distinct regions:

- 1) working area. The working area is the place in memory where music is created, edited, transformed, etc. All music data moving into, out of, or within the system must pass through the working area. The working area is an implicit operand of all the musical and data processing functions performed by the system. The function of the working area is therefore analogous to that of the accumulator in a single-address computer. The key difference is that the actual size of the working area varies, shrinking or expanding within the limits of available memory to accommodate whatever music data are placed in it.
- 2) user memory locations. The user memory locations are those areas of memory set aside for users to store their musical work. Each of these memory locations is identified by its own number. These locations function primarily as "scratch-pads" for tentative or incomplete work and as temporary storage for components of larger objects being formed in the working area. The user memory locations are analogous in function to the numbered memories found in some electronic hand calculators. However, like the working area, the user memory locations grow or contract within the limits of available memory to accommodate whatever music data are placed in them.
- 3) backup area. The backup area is a special place in memory set aside for use by the UNDO function (q.v.). With one exception, the backup area always contains a copy of the



contents of the working area as they stood just before the last function was executed. Thus the consequences of any operation on the contents of the working area can be reversed simply by copying the contents of the backup area into the working area. In the case of the single exception, the STORE function (q.v.), the backup area receives a copy of the contents of a specific user memory location. The reason for this difference is that the STORE function destroys the information previously contained in the user memory location while leaving the working area intact. Consequently it is the previous contents of the user memory location which must be saved if UNDO is to be able to reverse the effects of STORE.

- 4) keyboard buffer. When the system is in data entry mode, the character codes associated with keys played on the musical keyboard are temporarily stored in this special area of memory.
- 5) program buffer. When the system is in "program definition mode" (q.v.), the character codes associated with keys that are struck are stored in this special area of memory. Previously written programs can also be read into the program buffer from an external library module via the PROGRAM function (q.v.). The data stored in the program buffer remain there until they are either erased from the buffer or overwritten by new data.



- 6) number buffer. The number buffer is a temporary storage area for the digits of the numeric parameters required by several system functions. Whenever a digit key is struck while the system is in direct execution mode, the character code corresponding to that digit is catenated to the digit string in the number buffer.
- 7) music and program libraries. The system's libraries of musical pieces and application programs are stored externally in interchangeable read-only memory modules.

Special registers. The music system requires the following special registers:

- 1) loop counter. This special register is a down-only counter that can be (a) preset to any non-negative integer smaller than its modulus, and (b) decremented by 1. The loop counter can be tested for both zero and non-zero conditions.
- 2) program counter. This special register is an arithmetic unit capable of performing addition and subtraction. It is used as a pointer to the next program step to be executed when the system is running a program stored in the program buffer.
- 3) cursor. The cursor is a pointer used by several of the music system's functions. The cursor always points between notes in the working area. Notes can be inserted at the place indicated by the cursor, and the note immediately to

to the right of the cursor can be changed or deleted. The PLAY function always begins its performances from the note immediately to the right of the cursor.

Flags. The operation of the music system requires the following 1-bit registers, or logical "flags":

- 1) program flag. When this flag is set, the system is said to be in program definition mode.
- 2) data flag. When this flag is set, the system is said to be in data entry mode.
- 3) match flag. The state of this flag indicates whether the last matching operation performed by the MATCH function (q.v.) succeeded or failed (1 = success, 0 = failure).

Data types. The music system has only two elementary data types, the single character and the single note. Data structures are correspondingly restricted to strings of characters and matrices of notes. Internally the system also uses both single- and double-precision integers and character matrices, but these data types are not directly accessible to the user.

Character data. A unique character code is associated with each key on the system's front panel and with each of the "privileged" functions (q.v.). The type character is divided into the following sub-types:

- 1) music - 24 pitch characters and the rest character
- 2) digit - the digits 0 through 9

- 3) function - the characters corresponding to the remaining keys and to the privileged functions

The underlined sub-type names above are used throughout this chapter to designate both the characters belonging to each sub-type and their associated keys. Which meaning is intended should always be clear from the context.

Under the appropriate conditions, characters in one or more of the sub-types can be catenated into meaningful strings. In all cases the rule for forming a string is the same: each successive character entered is catenated to the right-hand end of the existing string.

Note data. A note is actually a composite entity consisting of at least four elementary parts:

- 1) starting time
- 2) channel assignment
- 3) pitch (which is itself a composite of octave register and pitch class)
- 4) duration

Each of these elementary parts, or parameters, is represented within the music system as an integer. The user cannot get at these numbers directly, however. The individual parameters of a note (or matrix of notes) are accessible only through music functions which take the entire note (or note-matrix) as an argument but then operate only on a specific parameter or parameters.

Notes can be catenated into matrices of any length within available storage. Each row of such a matrix contains the data for a specific individual note, and each column contains the values of a specific parameter for all the notes. The rows are ordered from top to bottom by starting time.

Data structures. Because of the calculator-like format of the music system, the ability to create data structures is very restricted. Note-matrices can be created only in the working area, and they can be stored only in the predefined user memory locations. Similarly character strings can be created only in the "buffer" areas specifically provided for them and only when the system is in the correct operational mode (q.v.). In order to mitigate the effects of these restrictions, the final implementation of the music system must have a memory management scheme that will allow data structures to grow and shrink arbitrarily within the limits of available memory. It is absolutely essential that this scheme include some kind of "garbage collection" facility to reclaim the unused areas of memory that are created as by-products of operations such as clearing the working area or storing a null program in the program buffer.

Operational modes. The music system has three distinct patterns of behavior, or "operational modes": direct execution mode, data entry mode, and program definition mode. Direct execution mode is the "normal" operating mode of the system. In direct execution mode, the system immediately executes the operations corresponding to

keys pressed. The system is automatically initialized in direct execution mode when it is turned on. Function keys are provided to move the system back and forth between direct execution mode and each of the other two modes. The details of all three modes are discussed below.

Direct execution mode. In direct execution mode, the operations corresponding to function keys are executed immediately when these keys are pressed. The characters corresponding to digit keys are automatically catenated to the right-hand end of the string in the number buffer. As successive digits are entered, this string will continue to grow until any function key is pressed. Once this happens the digit string is immediately evaluated as an integer and stored in a global variable. The number buffer is then cleared. Music keys pressed while the system is in direct execution mode will cause tones to sound, but they are otherwise ignored by the system. The user can interrupt an executing procedure simply by pressing any key; the latter action causes an immediate transfer of control to the "wait" step of the system's executive routine (q.v.). The DATA function (q.v.) key moves the system from direct execution mode to data entry mode, while the DEFINE function (q.v.) key moves it to program definition mode.

Data entry mode. In data entry mode, the characters corresponding to any music keys played are automatically catenated to the right-hand end of the string in the keyboard buffer. As successive notes or rests are entered, this string will continue to grow until the INSERT function key is pressed. INSERT translates the character



string in the keyboard buffer into the corresponding note-matrix and inserts this note-matrix into the working area at the place indicated by the cursor. INSERT then clears the keyboard buffer and returns the system to direct execution mode.

Program definition mode. In program definition mode, the system catenates the character associated with each key pressed to the right-hand end of the string in the program buffer. The system does not execute the procedure associated with any function key except DEFINE, nor does it put the characters corresponding to digit and music keys in their respective buffers. The character string stored in the program buffer constitutes a program which can be executed when the system is returned to direct execution mode. Once in program definition mode, the system stays in this mode until the DEFINE function key is pressed again. DEFINE closes the definition of the program and returns the system to direct execution mode. The program which has been defined can now be executed at any time and any number of times.

The operational modes of the music system are summarized in Table 1.

Function Repertoire. The music system has two broad categories of functions, "regular" and privileged". The regular functions are those directly accessible to the music system user, while the privileged functions are those accessible only to the programmers who create the games, lessons, utility programs, etc., for the system's plug-in memory modules. The regular functions are



Table 1. Operational Modes of the Music System

DATA ENTRY MODE	DIRECT EXECUTION MODE	PROGRAM DEFINITION MODE
entered via DATA left via INSERT	"normal" mode	entered via DEFINE left via DEFINE
1) all <u>functions</u> except INSERT are ignored  2) all <u>digits</u> are ignored  3) all <u>music</u> key-codes are catenated to the right-hand end of the string in the keyboard buffer	1) all <u>functions</u> are executed immediately  2) all <u>digits</u> are catenated to the right-hand end of the string in the number buffer  3) <u>music</u> keys are ignored	1) all <u>function</u> key-codes except DEFINE are catenated to right-hand end of the string in the program buffer  2) all <u>digits</u> are catenated to the right-hand end of the string in the program buffer  3) all <u>music</u> key-codes are catenated to the right-hand end of the program buffer
Note: the musical keyboard is "live" in all three modes; the tone corresponding to any <u>music</u> key pressed is always played by the system.		

predominantly high level musical and data processing operations like playing a piece of music, recording music, or accessing a music library file. The privileged functions, on the other hand, are low level machine-oriented operations such as setting a flag, decrementing a counter, branching, etc. Appendix F contains programming examples utilizing both types of functions.

Since the syntactical and operational characteristics of all functions in both categories are identical, the assignment of a function to one category or the other is not irrevocable. Any function in the privileged category can be moved into the regular category merely by providing a suitably encoded key or other device to represent it on the system's control panel. Similarly the removal of an existing key or control immediately puts the corresponding regular function into the privileged category.

Syntax. The syntax employed by the music system is very simple. Functions are executed in the order that they are called; there are no precedence rules or parenthesization. If a function requires a numeric parameter, the value of this parameter can be provided in either of two ways. The value can be entered immediately before the call to the function that will use it, or the value can be determined by the system itself. In the latter case, the system will use the normally assumed, or "default" value of the parameter for the function. Any numeric data entered immediately before a function that does not take a numeric parameter will simply be ignored. In the function summary below, the letter "n" before a function name indicates that the function takes a numeric parameter.

Regular functions. The following summary of the regular functions is intended only as an overview. A more formal and detailed description of each function can be found in Appendix A.

1) basic music functions. The functions in this group enable the user to change the pitches and durations of notes in the working area. Ordinarily these functions apply only to the note immediately to the right of the cursor. To apply any of them to the entire contents of the working area, precede the function call with END (q.v.). The functions in this group are:

- (a) nAUGMENT - multiplies note durations by n (default = 2)
- (b) nDIMINISH - divides note durations by n (default = 2)
- (c) nRAISE - transposes pitches up n semitones (default = 1)
- (d) nLOWER - transposes pitches down n semitones (default = 1)

2) music transformation functions. The functions in this group operate on the entire contents of the working area, and they typically produce musical results very different in character from the original object. The functions in group are:

- (a) INVERT - inverts all pitches about the first pitch
- (b) REVERSE - reverses the order of the notes
- (c) nVERTICALIZE - changes a linear sequence of notes into a series of n-note chords (default = 1)
- (d) nSHUFFLE - partitions a series of notes into groups consisting of n notes each, and then randomly rearranges the n-note groups (default = 1)

- 3) memory reference instructions. The functions in this group move, combine, and compare music data within the system. Each of these functions takes a numeric parameter, n, which specifies the user memory location to be employed. If no value is provided for the parameter, the system assumes location 0. The functions in this group are:
- (a) nSTORE - puts the contents of the working area into user memory location n
  - (b) nRECALL - brings the contents of user memory location n into the working area
  - (c) nCOMBINE - contrapuntally combines the notes stored in user memory location n with those already stored in the working area
  - (d) nMATCH - compares the notes in user memory location n with those in the working area and displays  
match!  
if corresponding notes are identical, but  
no match  
if they are not.
- 4) cursor functions. The cursor is an imaginary marker that indicates where in the sequence of notes in the working area insertions, deletions, changes, etc. are to be made. The cursor is always positioned to the left of the next note that can be changed or deleted. New notes can be added at the place the cursor points to. Using the functions in

this group, the cursor can be stepped forward or backward one or more notes at a time. Each note is played as the cursor traverses it. The cursor can also be moved by a single command to a point immediately before the first note in the working area or to a point immediately after the last note in the working area. The functions in this group are:

- (a) nSTEP - advances the cursor n notes (default = 1)
- (b) nBACK - backs the cursor up n notes (default = 1)
- (c) RESET - moves the cursor to a point immediately before the first note in the working area
- (d) END - moves the cursor to a point immediately after the last note in the working area

5) external library functions. The system has two external libraries: a library of musical compositions, or "archive", and a library of programs. Although both libraries would be physically contained in the same external read-only memory module, each may have its own directory, storage formats, access rules, etc. The following functions are used to access items in these libraries:

- (a) nLIBRARY - if preceded by a number, n, LIBRARY retrieves piece no. n from the library of musical compositions. If n is not specified, LIBRARY retrieves the piece whose eight-note incipit most closely matches the first eight notes of whatever music is in the working area.



- (b) nPROGRAM - retrieves program number n from the program library (n must be specified).
- 6) execute functions. The execute functions perform entire sequences of actions specified by the user. The functions in this group are:
- (a) PLAY - causes whatever music is presently in the working area to be performed on the system's synthesizer. The performance begins with the note immediately to the right of the cursor.
  - (b) GO - initiates execution of the program currently stored in the program buffer.
- 7) editing functions. The editing functions enable the user to drop specified notes or groups of notes from the working area. The functions in this group are:
- (a) CLEAR - clears the working area (i.e., deletes its entire contents)
  - (b) nDROP - deletes n notes from the piece in the working area, beginning with the note immediately to the right of the cursor (default = 1)
- 8) mode switches. When it is first powered up, the music system is in direct execution mode. The functions in this group are used to get the system into and out of its two other operational modes. These functions are:
- (a) DEFINE - switches the system back and forth between direct execution mode and program definition mode



- (b) DATA - switches the system from direct execution mode into data entry mode
  - (c) INSERT - switches the system from data entry mode back into direct execution mode. INSERT also translates the character string in the keyboard buffer into note-matrix form, and inserts this note-matrix into the working area at the place indicated by the cursor.
- 9) the UNDO function. UNDO is a function designed to help users extricate themselves from a troublesome situation into which they have strayed. UNDO exactly reverses the effects of the last function performed by the system. Thus, for example, it can be used to recover the previous contents of a user memory location that was accidentally overwritten or to restore a musical piece damaged by inadvertent use of one of the music functions.

Privileged functions. The privileged functions are a group of system operations available only to the creators of library programs. There are no keys on the system's front panel for the privileged functions; these functions are accessible only through whatever equipment is used to program the read-only memory modules which form the external libraries of the music system.

The privileged group contains functions which are absolutely essential for the creation of game and instructional programs but which are not necessarily of interest to the non-computer-specialists who are expected to be the primary users of the system. Included in the privileged group are:

- 1) conditional and unconditional branching functions
- 2) functions that provide access to the system's tempo clock
- 3) a software interrupt

There is no inherent reason why any of the privileged functions must be withheld from the user of the system. The decision to establish a privileged group was based primarily on a desire (1) to restrict the number of keys on the front panel to about the same number as a typical "scientific" calculator has, and (2) to avoid presenting the user with any functions requiring prior knowledge of computers and computer programming.

The privileged functions will not be discussed further here. See Appendix A for a full description of this group.

Executive routine. All of the operations of the music system, both the ones consciously initiated by the user and the "invisible" ones performed internally by the system itself, are coordinated by an overall "executive routine". Within the structure provided by the executive routine, the functions the user sees--the procedures represented by keys on the front panel--are actually subroutines called by the executive routine. Thus the executive routine is a level of conceptual structure lying between the regular and privileged functions and the actual computer hardware.

The operation of the executive routine is illustrated in two different ways here: a flowchart (fig. 2) and a set of APL functions (fig. 3). Both of these descriptions are intended only to show the essential structure and functioning of the executive routine.

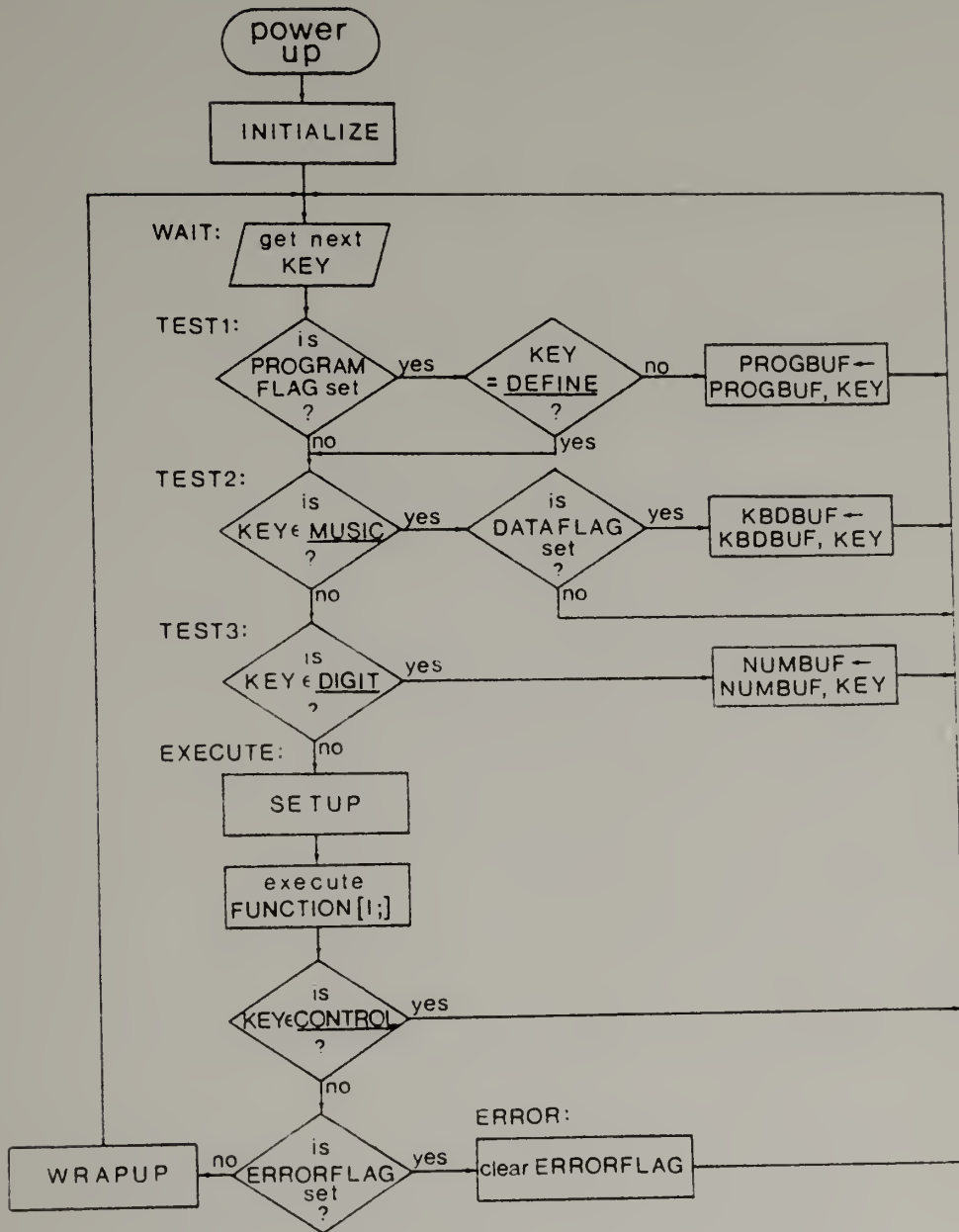


Figure 2. Flowchart of the Executive Routine.

```

▽EXECUTIVE[ ]▽
▽EXECUTIVE
[1] INITIALIZE
[2] WAIT: KEY←▽
[3] TEST1: →(¬PROGRAMFLAG)/TEST2
[4] →(KEY=DEFINE)/TEST2
[5] PROGBUF←PROGBUF,KEY
[6] →WAIT
[7] TEST2: →(¬KEY∈MUSIC)/TEST3
[8] →(¬DATAFLAG)/WAIT
[9] KBDBUF←KBDBUF,KEY
[10] →WAIT
[11] TEST3: →(¬KEY∈DIGIT)/EXECUTE
[12] NBUF←NBUF,KEY
[13] →WAIT
[14] EXECUTE: SETUP
[15]  FUNCTION[I;]
[16] →(KEY∈CONTROL)/WAIT
[17] →ERRORFLAG/ERROR
[18] WRAPUP
[19] →WAIT
[20] ERROR: ERRORFLAG←0
[21] →WAIT
▽

▽INITIALIZE[ ]▽
▽INITIALIZE
[1] PROGRAMFLAG←DATAFLAG←ERRORFLAG←0
[2] NUMBER←PREVNUM←PREVFUNC←I←0
[3] WORK←BACKUP←USERLOCS←10
[4] NBUF←PROGBUF←KBDBUF←' '
▽

▽SETUP[ ]▽
▽SETUP
[1] NUMBER←NBUF
[2] NBUF←' '
[3] I←FUNCTION\KEY
▽

▽WRAPUP[ ]▽
▽WRAPUP
[1] PREVNUM←NUMBER
[2] PREVFUNC←I
▽

```

Figure 3. APL Version of the Executive Routine.

Numerous details have been omitted so that the basic algorithm can be seen clearly.

The identifiers used in both the flowchart and the APL functions are:

- 1) CONTROL - a character vector that contains the character constants associated with the PLAY, STEP, BACK, RESET, and END functions
- 2) DATAFLAG - data flag (q.v.)
- 3) DEFINE - the character constant associated with the DEFINE function
- 4) DIGIT - a character vector containing all the members of the sub-type digit
- 5) ERRORFLAG - a logical flag that is set whenever an error condition is detected during the execution of any regular or privileged function
- 6) FUNCTION - a character matrix containing the names of all the regular and privileged functions
- 7) FUNCTION - a character vector that contains all the members of the sub-type function
- 8) I - the numerical index of KEY in the character vector FUNCTION
- 9) INITIALIZE - a routine that zeroes-out all working storage and clear all flags
- 10) KBDBUF - keyboard buffer (q.v.)
- 11) KEY - a character variable that always contains the code of the most recently pressed key of the code corresponding to the most recently executed privileged function

- 12) MUSIC - a character vector containing all the members of the sub-type music
- 13) NUMBER - an integer variable which contains the numeric parameter (if any) of the function about to be executed
- 14) NUMBUF - number buffer (q.v.)
- 15) PREVFUNC - an integer variable used to store the value of I after the execution of any regular or privileged function (except PLAY and the cursor functions)
- 16) PREVNUM - an integer variable used to store the value of NUMBER after the execution of any regular or privileged function (except PLAY and the cursor functions)
- 17) PROGBUF - program buffer (q.v.)
- 18) PROGRAMFLAG - program flag (q.v.)
- 19) SETUP - a routine that evaluates the digit-subtype character string in the number buffer as an integer, stores the integer in the global variable NUMBER, and then clears the number buffer
- 20) USERLOCS - user memory locations (q.v.)
- 21) WRAPUP - a routine that saves the values of KEY and NUMBER for possible use by the UNDO function
- 22) WORK - working area (q.v.)

Several important features of both the regular and privileged functions are implicit in the flowchart and APL routines. First of all it can be seen that each function is a self-contained subroutine called by the executive routine. Thus each of these functions is



a replaceable module which can be rewritten without affecting any other part of the system. Moreover a new function can be added to the set simply by concatenating its name to the FUNCTION matrix and concatenating its character code to the FUNCTION vector.

It can also be inferred from the flowchart and the APL listings that all of the functions share common duties above and beyond their individually assigned tasks. Specifically, every function is responsible for detecting its own error conditions and for setting the error flag if necessary. In addition each function is responsible for copying the contents of the working area (or user memory location) into the backup area if this information is necessary to undo the function's effects. If the function does not need the backup area, it is responsible for clearing it so that the system's "garbage collector" can reclaim the unused storage space.

Finally it can be seen that the information used by UNDO (PREVFUNC, PREVNUM) is not updated after the execution of PLAY or any of the cursor functions (STEP, BACK, RESET, and END). The primary reason for this feature is that it allows the user to step back and forth through a piece and to play it any number of times before deciding whether or not to UNDO the function that put the piece into its present form.

### The Prototype

The prototype is a computer-based music system designed to be operated in connection with a time-sharing computer. The prototype

was actually run on the CDC Cyber 175 computer system at the University of Massachusetts, but the characteristics of both its hardware and software components are such that it could employ any Cyber system which offers APLUM (the University of Massachusetts version of APL, which is also a standard product of CDC).

Although the prototype is not hand-held, it is portable. It breaks down into six lightweight parts, which are easily reassembled in about fifteen minutes. Moreover the prototype can be operated virtually anywhere since it incorporates an acoustic coupler which gives it access to the computer over the regular telephone network. The prototype was in fact set up and run at the author's home, at the University of Lowell's College of Music, and at the School of Education at the University of Massachusetts/Amherst.

The main purpose of the prototype was to act as a test vehicle for the basic design ideas of the theoretical system. Accordingly the prototype was designed to simulate the intended functional characteristics of the theoretical system as nearly as possible. There are, however, significant differences between the theoretical system and the prototype, as will be explained in detail below. Most of these differences are an inevitable consequence of the fact that while the theoretical system is designed around a dedicated micro-computer, the prototype is embedded in a large time-sharing computer system. Other important differences arose from the fact that the design of the prototype had to be frozen early in 1979 so that the prototype could actually be built and tested while the theoretical

system's design was (and is) free to continue its evolution.

Moreover, once it was built and running, the prototype itself became one of the prime motivating factors behind changes and improvements in the design of the theoretical system.

The actual construction of the prototype began with the design of the keyboard layout. No attempt was made to condense the keyboard to the projected size of the hand-held unit. Instead the configuration was deliberately made larger and more open because this arrangement would make the keyboard easier to change and maintain. It was also felt that a large, single-level (one function per key) keyboard would be simpler to use, and therefore, better for research purposes than a device which economizes on space by using a multiple-level keyboard and selector button(s). Once the keyboard layout was established, it was a relatively straightforward matter to construct the device from standard electrical hardware components.

The completed keyboard was substituted for the computer terminal as the device used to interact with the APL simulation. This substitution was readily accomplished since the correspondence between specific character codes and specific functions performed by the APL simulation had already been established within the simulation itself. It was only necessary to arrange for the keyboard to transmit the character associated with the function designation of each of its keys. This was easily accomplished with an interface built from a few standard electronic parts.

The only remaining component that had to be provided in order to complete the prototype was some kind of sound output device. This had to be a multi-voice, or "polyphonic" device in order to mirror the corresponding feature of the hand-held system. On the other hand, this device did not have to be a professional studio-quality music synthesizer in order to fulfill the prototype's function as a research tool. Accordingly a four-voice adaptation of a simple single-voice circuit designed by Lancaster (1974) was constructed.

Physical components of the system. Photograph 2 shows the physical components of the prototype as they were arranged during the pilot studies reported in the next chapter. From left to right, the devices shown in this photograph are:

- 1) the synthesizer (far left, in what looks like a loudspeaker cabinet)
- 2) the prototype keyboard and interface (the box with the buttons on it, and the box directly behind it)
- 3) the CRT terminal
- 4) the acoustic coupler

Each of the devices is described below.

The synthesizer. The synthesizer is a simple four-voice music device. It has four square-wave generators, each of which can be turned on and off independently of the others. Each tone generator can play any note of the equal-tempered chromatic scale over a range of eight octaves, with middle-C being the center of each tone generator's range. The synthesizer is controlled directly by the



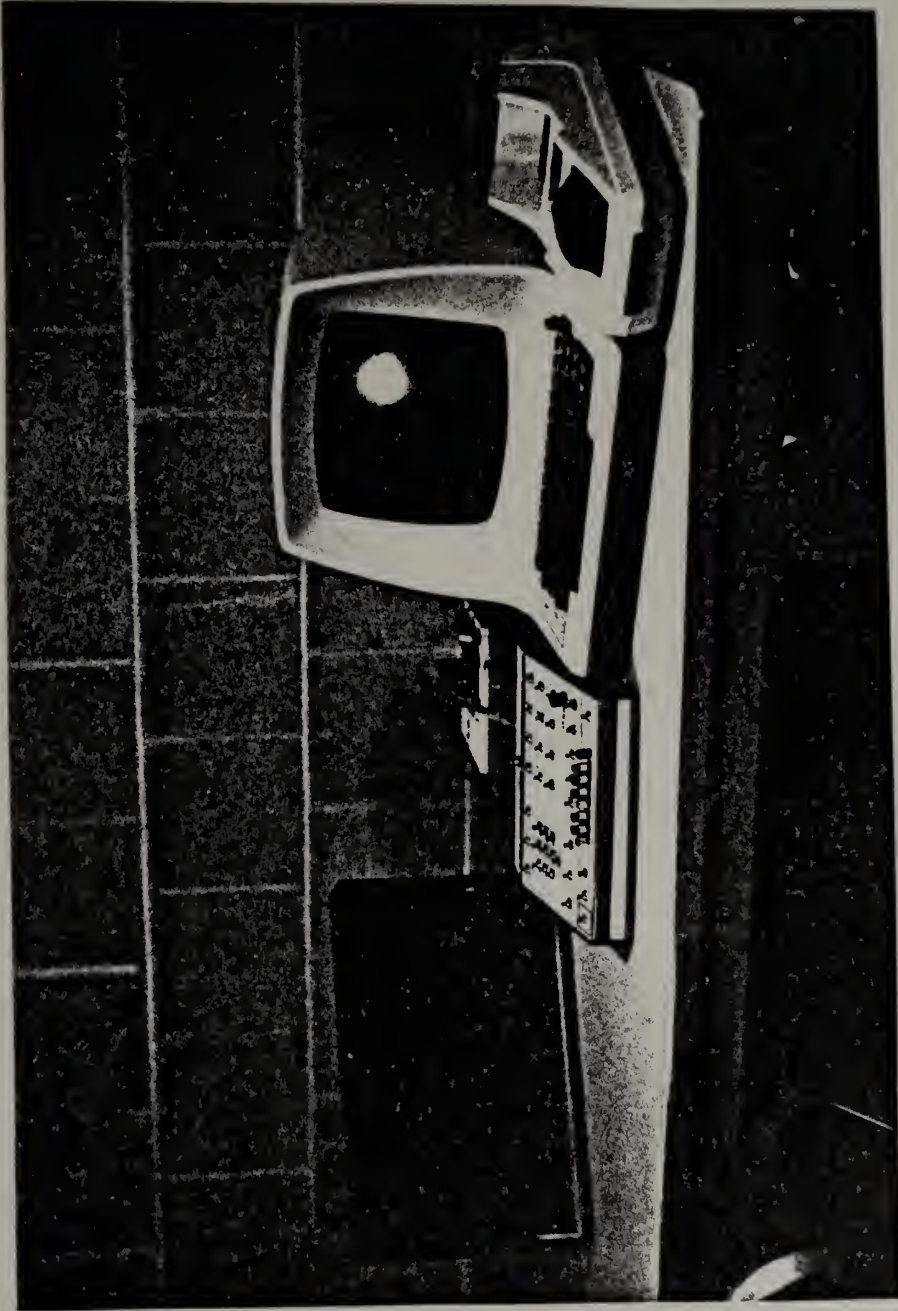


Photo. 2. The Prototype  
(Photo: Jay Forrest)

same ASCII character codes sent to the CRT terminal. Certain codes turn voices on or off, other codes select the pitches of notes, and still other codes select the octave registers of notes. The synthesizer is capable of operating at data rates from 150 to 4800 baud (a jumper inside the case is used to select the appropriate rate). The synthesizer has no provision for regulation of loudness; consequently no volume control is provided on either the synthesizer itself or the prototype's keyboard.

The synthesizer received frequent criticism during the pilot studies reported in the next chapter. The principal objections were that all of the pitches were slightly out of tune, that note durations were not accurate, and that notes which should have been simultaneous were always separated by a perceptible delay.

Although these problems could not be corrected during the pilot studies, they are actually moderately easy to fix. The synthesizer's pitches could be brought up to professional musical standards simply by providing the synthesizer with more accurate frequency dividers. Nothing more would be required than the addition of four integrated circuits and some consequent juggling of the existing circuitry inside the synthesizer's case.

The two other problems, inaccurate durations and lack of simultaneity, could both be solved merely by connecting the synthesizer to a high-speed computer port. Because it was connected to the computer over ordinary telephone lines during the pilot studies, the synthesizer had to be run at the standard 300 baud (30 characters/sec)



data rate. At this relatively slow speed, the time resolution of notes is very crude. In fact, note durations can be approximated only to the nearest tenth of a second, and "simultaneous" notes cannot be made to start less than a tenth of a second apart.

However the synthesizer is capable of operating at up to 4800 baud (480 characters/sec) without any modification. At this higher speed, note durations would be accurate to better than 0.01 second, and the delay between notes intended to be simultaneous would be less than 0.01 second. Since human hearing cannot resolve such small time differences, both of the time-related problems would disappear.

The prototype keyboard and interface. Fig. 4 is a drawing of the prototype's keyboard (shown about three-fourths its actual size). As can be seen, this keyboard is laid out differently from the theoretical system's: it is larger and less dense, and it contains some buttons not present on the theoretical system and is missing some others that are. The prototype keyboard unit contains only switches; all of its electronic components are contained on the "breadboard" directly behind it. The breadboard is an interface that translates keyboard switch closures into the ASCII codes associated with the specific keys pressed and then transmits these codes to the CRT terminal. For example, when any of the musical "white note" keys along the bottom of the keyboard is pressed, the interface transmits the ASCII code for "C", "D", "E", "F", "G", "A", or "B"--whichever is appropriate. Similarly, when any of the ten numeric keys is pressed, the interface transmits the ASCII code for

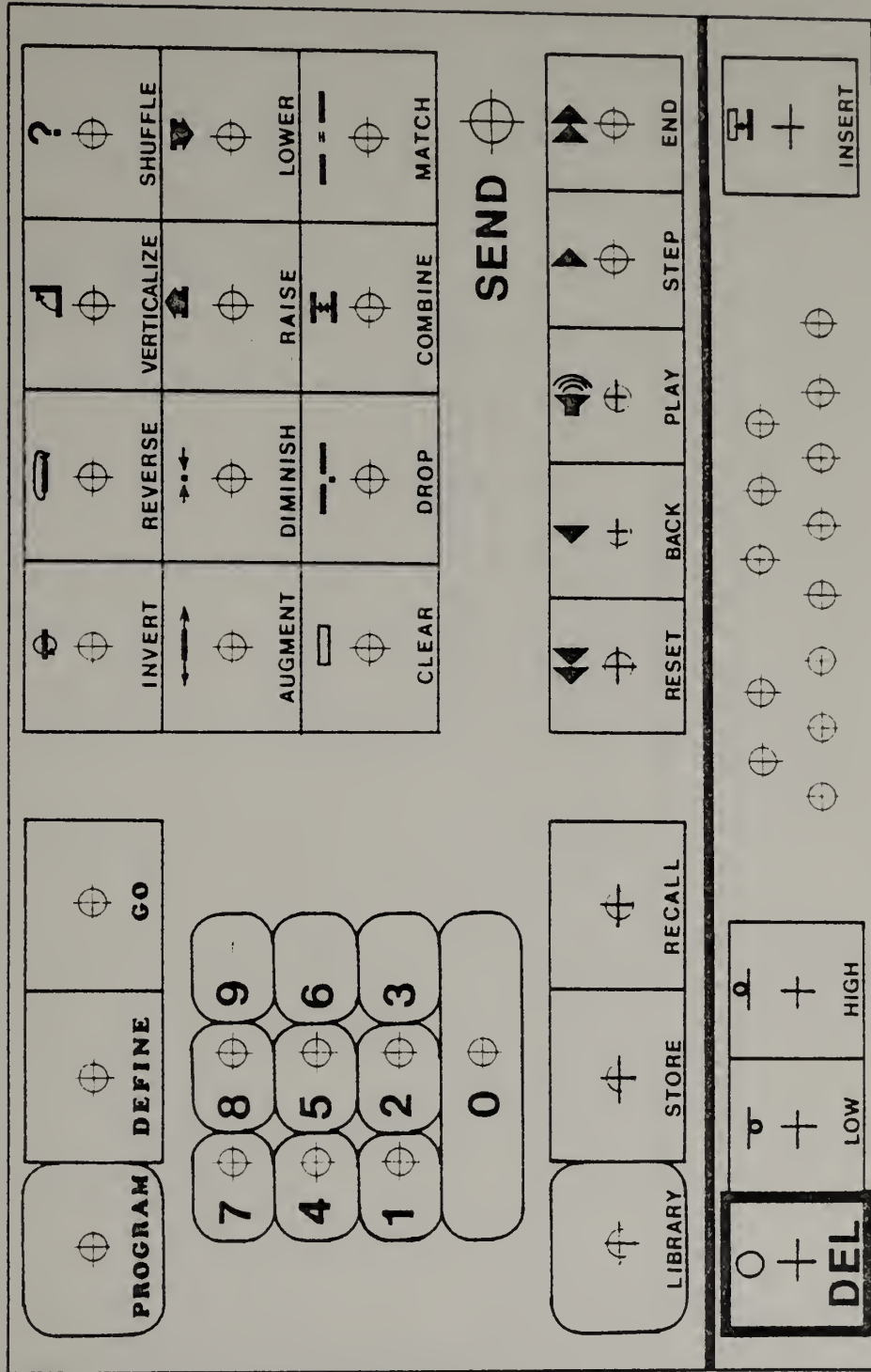


Figure 4. Prototype Keyboard Layout.

the corresponding digit, 0-9. Each of the other keys also has associated with it a unique ASCII code that is transmitted whenever it is pressed. The entire set of codes and the associations of specific codes with specific keys on the prototype's keyboard are of course known to the APL simulation which actually runs the prototype.

The prototype keyboard is a single-level device; each key has only one function. Accordingly there is no shift button. The lack of a shift button necessitates the addition of two special purpose keys, one to place the keyboard in its low octave and another one to place it in its high octave. These two keys are paired together between the DEL key and the "C" key of the musical keyboard. When one of these octave keys is pressed, all subsequent notes sound in the corresponding register until the opposite key is pressed. A single key could have been used to toggle the keyboard back and forth between octaves, but two keys--each of which forces the keyboard into a specific register--seemed like a more reliable way of handling this problem.

The CRT terminal. The CRT terminal has two principal functions:

- 1) It serves as the primary means for communicating with the remote time-sharing computer. It is used, for example, to log on and off the system, to access files, to edit programs, etc.
- 2) It serves as the prototype's display unit (the prototype's keyboard does not have a display of its own).

The specific terminal used with the prototype, a Lear-Siegler model ADM-3A, offers one especially handy feature: an extra RS-232C interface. This extra interface allowed the prototype's electronics to be plugged directly into the terminal itself. Without the extra interface an additional device would have to have been built to permit both the CRT terminal and the prototype's keyboard interface to be connected to the same data transmission line.

Acoustic coupler. The acoustic coupler gives the CRT terminal (and therefore the prototype's entire array of electronic components) access to the computer via telephone.

Software-simulated components of the system. All of the remaining parts of the prototype are either embodied in APL functions and APL files created by the author or else derived from features provided by the APL system itself. Taken together these components constitute a software simulation of the theoretical system. The purposes of this section are (1) to describe each of the software components, (2) to show the relationship of each software component to the corresponding aspect of the "real" theoretical system as described earlier in this chapter, and (3) to explain any differences between the two. All of the software-simulated components of the prototype are found in the APL listings of Appendix B.

Tempo clock. Because of the fixed and relatively slow rate of data transmission at which the prototype was forced to operate, it was not possible to implement the tempo clock feature in the prototype's hardware (note that the prototype's keyboard does not have

a Tempo control). The principal features of the tempo clock were therefore simulated in software.

The effect of the Tempo control is simulated by the assignment of a value to the global variable TEMPO. This is accomplished by first entering the code for the USR function (q.v.) on the CRT terminal and then typing in the appropriate APL assignment statement. This procedure is of course very cumbersome and, unlike the manipulation of the Tempo control of the theoretical system, is incapable of performance while music is playing.

The effect of the tempo clock's counter register (which continuously accumulates clock pulses) is simulated by the global variable, TIMER, and APL's  $\square$ TS system variable. Whenever the value of TIMER is needed, the DST function (q.v.) derives the updated value of TIMER from  $\square$ TS as shown in the listing.

Memory, special registers and flags. With one exception, the APL simulation has a counterpart for each of the memory components, special registers, and flags described above in the section on the internal architecture of the theoretical system. The various regions of random access memory are simulated as global variables in the workspace, as are the special registers and the flags. The two libraries are simulated as APL files (see Appendix E for a listing of the items in each library). The exception is the "program counter", which is not implemented in the simulation. Table 2 summarizes the correspondences between the theoretical system and the APL simulation.



Table 2. Identifiers Used in the APL Simulation

THEORETICAL SYSTEM COMPONENT	APL IDENTIFIER	TYPE
working area	X	n x 4 numeric matrix
user memory locations	V0 through V9	n x 4 numeric matrix
backup area	U	n x 4 numeric matrix
keyboard buffer	KBDBUF	n x 4 numeric matrix
program buffer	PROGBUF	character vector
number buffer	NBUF*	character vector
music library	MUSLIB	APL file
program library	PROGLIB	APL file
loop counter	COUNTER	numeric scalar
program counter	(not implemented)	N/A
cursor	CURSOR	numeric scalar
program flag	MODEFLAG	logical/numeric
data flag	DATAFLAG AND KBDFLAG**	logical/numeric
match flag	MATCHFLAG	logical/numeric

\*In the APL simulation, NUMFLAG is used to indicate the presence of at least one character in the number buffer.

\*\*In the APL simulation, KBDFLAG indicates that data entry mode has been entered via the musical keyboard rather than through a call to the DATA function.



The reason that the "program counter" was not implemented is that the simulation's executive routine, MACHINE (q.v.), already provides a string-interpretation mechanism that does not require a counter. So, instead of providing a second mechanism to execute programs, the author used the tools that were already available. The simulation's XCT function (q.v.) simply puts the contents of the program buffer (PROGBUF) into the global variable INPUT at which point MACHINE interprets the program character string just as if it had been entered at the terminal.

One other difference between the theoretical system and the simulation that can be seen here is that the simulation's keyboard buffer (KBDBUF) is a numeric matrix instead of a character vector. This is merely an anomaly left over from an earlier version of the simulation.

Data types. In the simulation, the treatment of data types corresponds quite closely to that of the theoretical system. The only elementary data types directly accessible to the user are the single character and the single note, and the only data structures are the character string and the note matrix.

A "note" in the simulation is a numeric vector of length 4. The elements of the vector are: (1) starting time, (2) channel assignment, (3) pitch, and (4) duration. Both starting time and duration are expressed in terms of the reciprocal of the current data rate. If, for example, the prototype is running at 30 characters per second (300 baud), the starting time of a given note is the

number of thirtieths of a second from the beginning of the piece until it is this note's turn to be played; the duration of the note is how long it lasts in terms of thirtieths of a second. "Channel assignment" is an integer in the range 0-3 that indicates which of the synthesizer's four voices is to play the note. "Pitch" is an integer in the range 0-95 that indicates which tone within the eight-octave (96 note) range of the voice is to be played. Individual note vectors can be catenated to form  $n \times 4$  note matrices.

Operational modes. The theoretical system and the prototype differ most widely in the respective patterns of behavior each exhibits in each of the three operational modes. Most of these differences were not intentionally created; they are rather the unavoidable result of the decision to base the prototype in a time-sharing computer system.

The fundamental cause of all the differences in behavior is the fact that the time-sharing computer does not respond immediately to each character transmitted to it. Instead it waits for a specific character, the "carriage return", which indicates the end of a message from the user. The immediate consequences of this fact for the prototype are:

- 1) the musical keyboard cannot cause notes to sound when it is played (the prototype's musical keyboard is never "live")
- 2) the STEP and BACK function keys cannot cause notes to sound when they are pressed
- 3) functions cannot be executed when their keys are pressed

In each of these three cases, the reason that the particular action cannot take place is that a carriage return must always be transmitted before the computer will respond. Naturally the prototype needs its own carriage return button; it is marked SEND in fig. 4.

The pattern of behavior imposed on the prototype by the time-sharing environment affects all three of the prototype's operational modes, but none more than direct execution mode. Indeed, direct execution mode typically appears to be anything but direct. Not only is a carriage return (SEND) interposed between command and execution, but so also is a delay whose length depends on the number and types of users simultaneously on the time-sharing system.

In an attempt to compensate for these drawbacks, some features differing from those of the theoretical system were implemented in the prototype's hardware and in the APL simulation. The first such feature is a provision to allow any number of keys to be entered before pressing SEND. This feature spares the user the bother and delay of having to press SEND after each and every keystroke, as would otherwise be the case. This "string interpreter" is a real convenience, but it has two less desirable side effects:

- 1) it blurs the distinction between direct execution mode on the one hand and program definition and data entry modes on the other because all three modes now permit entry of multiple keystrokes
- 2) it opens up the possibility of erroneous keystrokes being buried several characters back in a string, out of the reach of the UNDO function

The first problem still stands. The second was alleviated somewhat by providing the DEL ("delete") button on the prototype's keyboard. DEL issues the ASCII "escape" code, which causes the computer to ignore the entire input string.

Another, less drastic difference in behavior between the theoretical system and the prototype lies in the mechanism for getting from direct execution mode to data entry mode. The difference is that the simulation immediately moves from direct execution mode to data entry mode when any one of the music keys is pressed. The reason for this is that, since the prototype has a non-playing musical keyboard (notes do not sound when the keys are pressed), its music keys can have no plausible purpose other than to enter the corresponding notes into memory. Therefore the striking of any of the prototype's music keys can be used as a signal to the simulation to go immediately into data entry mode. Accordingly the key corresponding to the DATA function is omitted from the prototype's keyboard (the DATA key is the one in the lower left-hand corner in the drawing of the theoretical system, fig. 1). The INSERT key is retained in the prototype, however, since there must still be some way to get back to direct execution mode.

A final difference in behavior between the theoretical system and the prototype is that in the prototype neither an executing program nor a piece being played can be stopped by striking a key on the keyboard. In the theoretical system, on the other hand, pressing any key during program execution or musical performance causes an

immediate jump to the "wait" step of the executive routine. This feature was omitted in the prototype since there did not appear to be any reliable way to interrupt the computer at a specific desired point in a program or musical piece.

Function repertoire. The function repertoire of the APL simulation is almost identical to that of the theoretical system. The simulation has one additional function, USR (USeR). The USR function, invoked at the terminal, permits any valid single-line APL expression to be entered and executed within the simulation. USR was designed mainly as a debugging tool and diagnostic aid. It can be used, for example, to examine the contents of specific variables and to set variables to particular values of interest. The simulation does not have the privileged NOTE function (q.v.). This function is not needed because the simulation assumes that all music characters are to be stored in the keyboard buffer unless the system is in program definition mode.

Executive routine. The simulation's executive routine, MACHINE, is not quite the same as the one shown for the theoretical system in figs. 2 and 3. The major difference is that the APL simulation's executive routine has a built-in record-keeping facility. The record-keeping facility stores (in the variable RECORD) the numeric index of the character code of every key pressed by a user during an entire session with the prototype, and it tags any entries that caused error conditions by storing the negative of the index.



The record-keeping facility was put in to keep a history of user interactions with the system and to serve as a diagnostic aid in case any unusual bugs were discovered in the simulation.

### Unresolved Design Issues

As of this writing, a number of design issues remain unresolved. Unlike the various flaws, inconsistencies, and bugs discovered during the testing phase, these problems have no "right" solutions. Instead each of these issues hinges on one or more of the basic design points that determine the fundamental character of the music system as a whole. Resolution of each of these issues therefore involves a decision as to what the music system is, how it can be used, who can use it, etc. The pros and cons of each issue are stated briefly below.

The musical keyboard. When the music system is in data entry mode, it records a standard one-beat-long note or rest for each music key pressed. Some people feel that this arrangement is unnatural and that the system should be changed so that it would record the actual durations of the notes and rests played on the keyboard. The argument against this change is that it would require users to develop some manual skill in order to be able to use the keyboard effectively. This change would therefore move the music system in the "specialist" direction.

The keyboard was also criticized for the means it employs to change register: the HIGH and LOW keys on the prototype and the



shift key on the theoretical system. Both techniques are awkward and they gain only one additional octave. Clearly there is room for improvement here.

INSERT. The INSERT function takes the sequence of notes in the keyboard buffer and inserts it into the sequence of notes already in the working area at the place indicated by the cursor. It then clears the keyboard buffer and returns the system to direct execution mode.

A frequent complaint was that this procedure is too complicated and that INSERT should simply overwrite or replace the contents of the working area. This suggestion was rejected, however, because it would make the process of adding or changing individual notes in the working area awkward and roundabout. For example, under the suggested version of INSERT, the following steps would be needed to insert a single note into a piece of music in the working area:

- 1) STORE the piece in a user memory location
- 2) switch to data entry mode (via the DATA function)
- 3) hit the desired note
- 4) switch back into direct execution mode (via the INSERT function)
- 5) STORE the note in the working area in a different user memory location
- 6) CLEAR the working area
- 7) RECALL the piece from the user memory location
- 8) STEP the cursor to the desired place
- 9) RECALL the note from the user memory location

With the present version of INSERT, however, the steps required are:

- 1) STEP the cursor to the desired place
- 2) switch into data entry mode
- 3) hit the note
- 4) switch back into direct execution mode.

The END - (function) - RESET sequence. The AUGMENT, DIMINISH, RAISE, and LOWER functions have as their default argument the note immediately to the right of the cursor. To cause any of these four functions to operate on the entire contents of the working area simultaneously, it is necessary to precede the desired function with END, and then follow it with RESET. Many people feel that this three-keystroke sequence is inconvenient and that it should be eliminated by making the entire contents of the working area the default argument of these four functions.

The case for keeping the single-note argument for AUGMENT and DIMINISH is that (1) the Tempo control already provides a means for making overall changes in the speed of performances, and (2) AUGMENT and DIMINISH are the only mechanisms available for making accurate changes in the values of individual notes and for setting up precise duration relationships between notes.

The case for keeping the single-note default for RAISE and LOWER is not quite so strong since the system provides no other mechanism for transposing a whole piece. Perhaps what is really needed, however, is a "transpose" control that can raise or lower pitches while they are playing.

SHUFFLE. The present version of the SHUFFLE function is sufficient for randomizing the presentation of musical items in simple lesson or game programs. To some extent SHUFFLE can also be used compositionally. For example, by having it work on carefully prepared musical source data, SHUFFLE can approximate some of the effects achieved by the more sophisticated probabilistic procedures devised by Hiller (1969), Xenakis (1971), and Grossi and Sommi (1974). The unresolved question here is whether such procedures are too esoteric to be implemented in a device like the present one, or whether the curiosity about such things shown by several of the people who tried the prototype is an indication that additional composition-generating functions should be added to the present design.

#### Subsidiary issues.

"Stop" function. In the prototype there is no mechanism for stopping the system while it is playing music or executing a program. It is clearly necessary, however, to have some kind of "stop" mechanism. Accordingly, in the theoretical system, both musical performance and program execution can be stopped simply by striking any key on the control panel (this causes control to be passed immediately to the "wait" step of the executive routine). While this takes care of the problem, it might be better to have a specific STOP button analogous to the ones found on most tape recorders and cassette players.

COMBINE. The present version of the COMBINE function is not symmetrical, in the sense that its two arguments are not in general

interchangeable. While the notes and rests in the working area may represent any number of independent musical voices, the notes and rests in the user memory location must represent a single musical voice. This restriction seems unnecessary in the present design though it might make sense in a system each of whose output channels could be given its own distinct timbre, envelope shape, volume level, etc.--in this case COMBINE would have to make sure that each new voice part added was assigned to the proper channel.

LIBRARY. The "hum a few bars" library accessing facility is based on a simple algorithm devised by Bridgman (1950) and subsequently employed by Bryden and Hughes (1969). This algorithm does not search the musical pieces themselves for the notes that match the user's input, but rather a directory of musical incipits provided by the person who created the library. Although this procedure is a reasonable first approximation to the desired facility, it is too limited. This algorithm will fail to find a piece in the library if, for example, the user provides it with any important thematic idea from the piece other than the one(s) in the directory. The final hand-held unit should therefore employ more sophisticated pattern-matching techniques which are capable of searching the musical pieces themselves.

UNDO. The present version of UNDO exactly reverses the effects of the last function executed. Thus users can easily recover from any single mistake they make if they catch it in time. Although UNDO in this form is already costly in terms of the storage space

it consumes, there is some educational justification for increasing its power. If UNDO could "reverse-execute" an entire session step-by-step all the way back to the point when the music system was first turned on, it would be a powerful tool for helping users to discover the causes of the mistakes they make. Such an UNDO would, however, make extravagant demands on the system's memory.



## C H A P T E R I V

### TESTING: METHODS AND RESULTS

#### Introduction

In order to test and evaluate the prototype music system, two pilot studies were conducted at the University of Lowell during April 1979. Although the basic format of both studies was the same, each study was intended to elicit a different kind of information about the prototype and each study drew upon a different population of students for its subjects.

The first pilot study was essentially a shakedown test of the prototype. The purposes of this study were (1) to give the entire system a rigorous workout in order to uncover any problems with its hardware or software components, and (2) to determine whether the prototype had any awkward, illogical, or otherwise poorly human-engineered features.

The originally intended purposes of the second pilot study were (1) to discover what kinds of approaches a group of college students would take in solving a set of typical musical problems, and (2) to determine whether or not the prototype lent itself readily to the particular approaches selected. The knowledge gained in observing the students' problem-solving behavior and reactions to the system was then to have been used as a basis for refining the design of the theoretical music system. Unfortunately, because of certain

operational characteristics of the prototype, this plan could not be carried out. These characteristics, all derived from the time-sharing environment in which the prototype functions, prevented the subjects from using the system at the level of efficiency necessary to achieve the original objectives. Consequently, the second pilot study was limited to elaborating and confirming information gained in the first pilot study.

Both pilot studies had the same three major components:

- 1) a 50-minute lecture/discussion on the music system project  
(given to the entire group of potential subjects)
- 2) a one-hour individual session, during which each subject was asked to perform a series of typical musical tasks using the system
- 3) a questionnaire which each subject filled out after his/her individual session with the system

The purpose of the lecture/discussion was to give the subjects all the information they would need in order to make an informed decision concerning whether or not to participate in this study. The purpose of the individual sessions was of course to provide an opportunity to observe the subjects and the prototype system in action. And, finally, the purposes of the questionnaire were (1) to gather essential background information on the subjects, and (2) to get the subjects' own reactions, observations, thoughts, etc. on key aspects of the prototype. The differences between the two studies lay mainly in the types of activities that were emphasized in the

individual sessions. In the first study, the subjects were encouraged to try a wide variety of operations with the prototype so that the whole system would be exercised. In the second study, activity during the individual sessions was confined mainly to the prescribed set of musical tasks.

Both pilot studies employed University of Lowell undergraduates as subjects. The majority of students who participated in the first pilot study were music majors, all enrolled in the author's Electronic Music course. All of the students who participated in the second pilot study were non-music majors, enrolled in the author's History of Jazz course.

Although a number of difficulties were encountered in both pilot studies, the overall response of the participants was positive and encouraging. Evidently most of the participants were intrigued by the idea of a powerful computer music system that you can hold in your hand, and consequently they were willing to overlook the flaws--some quite serious--that they found in the prototype system they actually worked with.

### Pilot Study I

The first pilot study had two main purposes. The first was to exercise every part of the prototype system thoroughly in order to bring to light any problems the system might have. Among the major questions to be answered were:

- 1) Do the APL functions, which are the real heart of the prototype, work properly in a variety of situations?
- 2) Do all of the prototype's hardware components function correctly, and can they withstand continuous usage?
- 3) Is the telecommunications link between Lowell and Amherst reliable?

The second major purpose of this study was to discover whether the prototype had any features that were poorly designed from the human engineering standpoint. Among the questions to be answered in this connection were:

- 1) Is the keyboard layout clear and logical?
- 2) Are all of the necessary system functions provided, and are they in an appropriate form?
- 3) Is all of the information necessary to operate the system provided, and is it readily available?

Subject profile. The subjects employed in the first pilot study were eighteen of the twenty-four students enrolled in the author's Electronic Music course during the Spring 1979 term. All twenty-four students signed up for the study. Unfortunately problems with the computer system and the telephone link prevented completion of six individual sessions. Since Electronic Music is offered by the College of Music as an upper-level music theory elective, the majority of the students (sixteen) were juniors and seniors in the College. Of the two non-music students, one was an electrical

engineering major and the other was an undeclared liberal arts major who has since declared music as his major. The students ranged in age from 20 to 23. Their average age was 21.3 years.

As would be expected, the musical background listed by the sixteen music majors was considerable. Every music student of course plays a musical instrument, with twelve of the students playing two or more. All of the music students indicated at least three years of formal training on their principal musical instrument, and three students claimed as much as thirteen years. The average for the whole group of music majors was about 8.4 years of formal lessons on the principal instrument. All but one of the music students had participated in their high-school's music program. The liberal arts student who subsequently changed his major to music said that he plays two musical instruments and that he has had formal lessons on the principal one for two years. Like the music majors, he had also participated in a high-school music program. The remaining student, the electrical engineering major, claimed to have no previous musical experience of any sort.

Four students indicated that they had had some experience with computers. The electrical engineering student said that he was an assistant computer operator at the University of Lowell Computer Center and that he had done a good deal of programming in FORTRAN and assembly language. Two of the music students said that they had done some computer programming: one student said that he had used BASIC and FORTRAN, the other that he had used XPL and FORTRAN.



A third music student said that he had done a little bit of programming in BASIC as part of a high-school mathematics course.

The most significant common denominator among all eighteen students was the fact that they had all been working with the College's SYNCLAVIER system for two months prior to the pilot study. Consequently these students were acquainted with many of the fundamental ideas involved in using a computer to store, transform, and play music.

Activities. The first group lecture was given on April 3, 1979, during the regular meeting of the author's Electronic Music class. The lecture covered (1) the basic ideas of both the theoretical and the prototype systems, (2) related work in instructional applications of computers in music and in other fields, and (3) the nature of the pilot study in which the group was being asked to participate.

Following the lecture the floor was opened for question and discussion. Since the students in this class were relatively sophisticated vis-a-vis the type of work undertaken in the music system project, the discussion focused on a few fairly technical points. Some students, for example, were interested in the integrated circuits used in the prototype's synthesizer. Other students wanted to know about the computer in Amherst and how it was hooked up to us in Lowell. Still others were concerned with how best to market the final hand-held system.

At the end of the meeting, the students were asked to sign up for one-hour individual sessions. These sessions began the next day, April 4, and ran through April 11. During this period, eighteen individual sessions were completed. The sessions were conducted in the author's office in Durgin Hall, which houses the College of Music on the University of Lowell's south campus. The prototype was set up in the office exactly as shown earlier in photograph 2. The author was present to answer questions and provide assistance throughout all of the sessions.

Because one of the main purposes of this study was to test the prototype fully, the activities undertaken during each session varied considerably. However, the following four activities were used as guidelines for all of the sessions:

- 1) Pick a tune in the system catalog and play it.
- 2) Play the pitch-matching game (i.e., match a set of notes picked at random by the system).
- 3) Retrieve the "tune with mistake(s)" from the system's library and then correct all of the wrong notes found in the tune.
- 4) Make a full three-part performance of one of the rounds listed in the catalog ("Are You Sleeping" or "Three Blind Mice").

These particular activities were chosen for two reasons. The first is that together they require the use of all of the basic system operations (performance of music, storage and retrieval of music

data, editing, entry of music data, and program execution). The second reason is that, since these tasks are representative of the kinds of things that might be done with the hand-held system, they constitute a realistic test of the prototype design.

All eighteen students performed all four tasks. Most of the students also tried other things, largely of the undirected, what-happens-if-I-push-this-button sort. Far from being pointless, the latter activities often forced malfunctions that led to the discovery of significant faults in both the hardware and software components of the system.

Upon completing the individual session, each student was asked to fill out a two-part questionnaire (Appendix C). Part I of the questionnaire requests information about the subject's academic and musical background and about his/her previous experience with computers. This background information is summarized in the "Subject Profile" section above. Part II of the questionnaire asks the subject to comment on various aspects of the prototype design and to suggest uses for the final hand-held system. The comments are summarized in the "Results of the Questionnaires" section below. The suggested uses are included in Appendix D, which is an informal compilation of ideas offered by students in both pilot studies, as well as by friends, colleagues, and other students of the author.

Results of the individual sessions. Although there was considerable experimentation with the prototype during the individual sessions,

most of the activity centered around the four prescribed tasks.

These were:

- 1) Pick a tune in the system catalog and play it.
- 2) Play the pitch-matching game.
- 3) Correct all the mistakes found in a familiar tune.
- 4) Make a full three-part performance of a well known round.

All of the students tried at least these four activities with the prototype.

All of the students found it trivially easy to retrieve a library tune and play it. Most also found it very easy to retrieve and play one or more variants of the pitch-matching game. Accordingly both of these activities were accomplished in the first few minutes of each individual session and with only minimal assistance from the author. However, most of the students expressed some irritation at the slow response of the system and at the necessity to press the SEND button before anything would happen.

The third task involved fixing the wrong notes in a familiar tune ("Yankee Doodle", but with two notes a semitone flat). During the first day of individual sessions (April 4), the students found this problem almost impossible to solve. The main reason was that there was no easy or reliable way to move the cursor to a given note. Because of the inherent characteristics of the time-sharing system, STEP and BACK cannot cause tones to sound when these keys are pressed as a carriage return must be transmitted first. Consequently it is almost impossible to find a wrong note "by ear", i.e., by stepping

the cursor one note at a time while listening for the mistake. Thus about all the students could do was play the entire melody several times in order to memorize it and then mentally count up the number of STEPs required to get the cursor over to the wrong note. Needless to say this made the process of fixing even a single note quite painful. All of the first day's students said that they needed some kind of visual display of the notes and some kind of feedback on cursor movement.

Two different displays were implemented after the April 4 sessions were completed. One is intended to assist with coarse movement of the cursor, the other with fine movement. The first display, generated by the PLAY function, gives just the letter pitch of each note played. This information is simply strung out across the CRT screen on as many lines as are needed. The resulting panoramic picture shows all of the notes in an entire composition (unless it is very long), which makes finding the approximate position of any given note fairly easy. The second display, generated by the STEP and BACK functions, shows the complete pitch, octave register, and duration data of each note traversed by the cursor. The information for each note is displayed on its own line. Each line begins with the ordinal number of the note within the entire sequence of notes currently in the working area. This second display makes it relatively easy to home in on a specific note once its approximate position has been determined.



Although the editing process still moved rather slowly after the displays were provided, the remaining students in the study had much less difficulty solving the fix-the-notes problem. Generally the only assistance they required was a brief explanation of the cursor functions and the DROP and/or RAISE functions. In solving this problem, most of the students at first ignored the music functions altogether. The typical first stab at correcting a wrong note involved (1) stepping the cursor over to the wrong note and then (2) pressing the key of the presumably correct note. Most of the students assumed that simply keying the correct note would automatically replace or overwrite the wrong one. Since the system does not work this way, the author had to explain to the students who tried this approach that they had merely inserted an additional new note and that the wrong note was still there. The author then directed the students' attention to the 3 x 4 array of music function keys on the upper right portion of the panel. At this point the majority of these students "discovered" the DROP button. While some first asked about the action performed by DROP, others went directly ahead and fixed the wrong notes. The remaining students had noticed the RAISE button and had correctly guessed that it could be used to fix the wrong notes since they were both a semi-tone flat. Again, some went directly ahead with the repairs while others first asked about the action performed by the function.

An unanticipated consequence of providing the note displays just described was that the next day, April 5, the students demanded the

capability to step precise multiples of notes back and forth through the working area via a single command as at this point STEP and BACK could move the cursor only one note at a time. Since they could see exactly where a given note was, these students felt that it was a needless inconvenience to have to press the STEP and BACK keys several times to move the cursor to that note. They felt that, instead, it should be possible simply to precede the STEP and BACK commands with the proper number of steps to accomplish the desired move. Since this was clearly a more efficient way of handling STEP and BACK, these functions were changed accordingly at the end of the second day's sessions.

Several students suggested that a repetition factor also be implemented for notes and rests entered on the musical keyboard. This suggestion was rejected. The reason is that the author wanted, and still wants, to save numeric information interspersed with input from the musical keyboard for purposes such as change of octave register, assignment of output channel, and choice of timbre.

The final task, creating the three-part round, turned out to be a real struggle for most of the students. In fact only two of them were able to solve this problem without assistance. Not surprisingly these two were the music majors who had had significant computer programming experience. The difficulties here were evidently conceptual. Before beginning this task in each session, the author asked each student to describe how he or she would go about creating the three-part round. Many of the procedures offered were vague or

insufficient--certainly incapable of being translated into a program that could be executed by a machine. A typical suggested "procedure" was: "Somebody starts singing, and then somebody else starts."

A complete and logically correct procedure was proposed, however, in more or less the same form by several students. Although this procedure is not possible with the prototype--or with the theoretical system as presently conceived--it suggests a fundamentally different and possibly better way of organizing the operations of the music system. The idea is to put some kind of "marker" into the working area at the point where each voice after the first is to begin. As each marker is encountered during a performance, the marker causes successive voices, previously stored elsewhere in memory, to be started at the right time. A more active variant of the same idea was also expressed by several students. The latter said that it should be possible to start the pre-stored voices manually while the first voice is already playing. A special "cue" button or buttons could be provided for this purpose.

The most frequent activity spontaneously undertaken by the students was the attempt to play a tune on the prototype's 12-note musical keyboard. Most of the students who tried this were dismayed by the fact that tones do not sound when the keys are struck. This peculiarity is derived from the same characteristic of the time-sharing environment that prevents STEP and BACK from sounding tones, i.e., the necessity for the user to transmit a carriage return before the system will act. Many of the students who tried to pick out

tunes were immediately discouraged from further playing by this delay. Other students persisted in their attempts, but, because of the lack of immediate acoustic feedback, they were unable to hear when they had hit a wrong key or when an insufficiently depressed key had failed to register. As a result most of these students were disappointed when they finally heard their melodies.

Before the end of the individual sessions, it was obvious that the problem of the non-playing keyboard and the problem of the inaudible cursor movements performed by STEP and BACK were seriously hindering the students in performing the prescribed tasks, especially numbers 3 and 4, and discouraging them from experimenting with the prototype. Taken together with the often sluggish response of the computer system, these problems were acting as a sort of "governor" that prevented the students from operating the prototype beyond a certain speed or level of performance. There was nothing that could be done about this situation, however, since the difficulties grow directly out of inherent features of the time-sharing system in which the prototype is based.

Results of the questionnaires. After completing the individual session, each student filled out a two-part questionnaire (Appendix C). The responses to the first part, which asks for the details of each subject's academic, musical, and computer background, are summarized above in the "Subject Profile" section. The responses to the second part are summarized here.

Part II. of the questionnaire asked the following questions:

- 1) Is the music system easy or difficult to use? Consider, for example, keyboard layout, the functions provided, the information shown in the CRT, the music "synthesizer", etc.
- 2) What additions and/or changes would improve the music system?
- 3) What could the music system be used for? How would you use it?
- 4) If the hand-calculator-sized music system described to you earlier were available, would you like to own one? If yes, how much would you be willing to pay for it?

Most of the students did not write at great length in answering these questions. This did not pose a problem for the study, however, since the students had already communicated a great deal of useful information verbally during the individual sessions.

Concerning the ease or difficulty of using the prototype (question 1), six of the eighteen students pointed to the various time-sharing problems as major obstacles. Five students said that the CRT display was confusing. Readers who did not see the prototype in action may wonder about the meaning of this particular criticism. The students were alluding to the great deal of meaningless information displayed on the CRT screen when the synthesizer is playing. The reason this information appears is that both the synthesizer and the CRT terminal are connected to the same telecommunications line and both respond to the same ASCII character codes. Unfortunately the useless information is interspersed with significant items such as prompts to enter commands or data, error messages, match/no match



indications, and elapsed time readouts. Several students complained that they could not easily distinguish the meaningful information from the garbage.

Six students commented favorably on the layout of the prototype's keyboard, while eleven others said that the keyboard was easy to use, but only after a little practice. Only one student felt that the keyboard needed major reorganization. Three students said that the procedures required to accomplish the four prescribed tasks appeared roundabout.

A variety of suggestions for improving the music system was offered in response to question 2. The four most frequent suggestions were:

- 1) provide a better synthesizer (eleven students)
- 2) provide an instruction manual and/or more instruction in the use of the system (six students)
- 3) simplify the system (three students)
- 4) provide more game and lesson programs (two students)

The remaining suggestions, each made by just a single student, were predominantly recommendations to add to the system what the students thought were new features. Actually all but two of the features proposed are already a part of the theoretical system design (e.g., ability to interface with a tape recorder, stereo system, or synthesizer; provision for a full-sized keyboard; graphic display of music; etc.). The two genuinely new proposals were (1) to make the keyboard velocity sensitive so that it can register the loudness of

notes played on it, and (2) to provide the ability to list and edit programs created in program definition mode.

The responses to question 3 heavily emphasized training in basic musicianship. Eleven of the eighteen students, for example, said that they would use the system for ear-training, and four students said they would use the device to improve their sight-singing skills. Many other suggestions were made by individual students. These are included in the general compilation of suggested applications in Appendix D.

The responses to the two parts of question 4 were as follows:

- 1) Thirteen students indicated that they would like to own one of the hand-held units while two students said they would have no use for it. Three students gave no response to this part of the question.
- 2) Fourteen students suggested price figures ranging from \$15 to \$200. The average was approximately \$85. Four students did not respond to this question.

### Pilot Study II

Many problems with the prototype were discovered during the first pilot study. Although most of these had been rectified well before the second pilot study was begun, the hard core of problems related to time-sharing remained. It had been observed in the earlier study that these difficulties materially impeded man-machine interaction, and the same pattern was found in this study too. The student

subjects, through no fault of their own, were simply unable to interact with the system at any higher a level than had their predecessors. The net result was that the amount of new information gained during the second pilot study was much less than had been hoped for.

Subject profile. The subjects employed in the second pilot study were thirteen of the twenty-five students enrolled in the author's History of Jazz course at the University of Lowell during the Spring 1979 term. A total of fifteen students volunteered, but once again computer and telephone problems prevented completion of all the individual sessions.

Since History of Jazz is a "service" course offered by the College of Music to the entire University community, the academic backgrounds listed by the participants in this study were quite varied. Of the thirteen students, five major in art, three in sociology, two in psychology, two in health professions, and one in economics. Since History of Jazz fulfills one of the University's "core" requirements, which are typically completed during the freshman and sophomore years, this second group was younger on the whole than the first group. The ages of the members of the second group ranged from 18 to 21. The average age was 19.8 years.

Of the thirteen students who participated in this study, only two indicated that they had had no previous musical background. All of the others said that they play, or have played, at least one musical instrument and that they have had formal lessons on the instrument. The lengths of formal training ranged from less than a

year to over eleven years. Eight of the students said they had participated in a high-school music program: four in performing ensembles, two in academic music courses, and two in both. Of these eight, three had also participated in one or more College of Music performing ensembles. Finally, one of the students, the economics major, had been a music major during his freshman year at Lowell.

It is not unusual to find this degree of musical experience among the non-music major students who take courses in the College of Music. Because of the strong professional orientation of the College--four of its five undergraduate programs prepare students for specific careers in music--even service courses tend mainly to attract students who have some prior musical background and are therefore reasonably confident of their ability to get good grades.

Ten of the students indicated that they had had no experience with computers, and two others said they had only run a few packaged programs on a time-sharing system. One student was taking a FORTRAN course during the pilot study and another had learned BASIC in high-school. The latter student had also worked with a process control computer during a summer job in a textile mill.

Activities. The second pilot study followed the same format used in the first, i.e., group lecture/discussion, individual sessions, final questionnaire. The lecture/discussion was given on April 18, 1979, during the regular meeting of the author's History of Jazz class. This lecture covered the same topics as the first one, but it was

less technical in nature since the students in the Jazz class were not music specialists and since they did not have the electronic and computer music technology background of the members of the first group. The lecture also covered the time-sharing problems encountered in the first pilot study. Most of the questions following the lecture revolved around what the students were going to be asked to do in the individual sessions. Accordingly the author gave a summary of the tasks that were to be performed with the prototype.

Following the lecture/discussion, the students were asked to sign up for individual one-hour sessions. These began the next day, April 19, and continued through April 25. Again the sessions were conducted in the author's office, with the prototype set up exactly as before. And, once again, the author was present throughout all of the sessions.

Reflecting both the experience gained during the first study and the fact that the students this time were not music specialists, the tasks prescribed for the second set of individual sessions were slightly different from those of the first pilot study. The tasks were as follows:

- 1) Pick a tune from the library and play it.
- 2) Pick a different tune from the library. Play it backwards and then play it upside down.
- 3) Play one or more versions of the pitch-matching game.
- 4) Get the "tune-with-mistake(s)" from the library and then correct the wrong notes in the tune.



- 5) Make a two-part performance of one of the rounds stored in the library ("Are You Sleeping" or "Three Blind Mice").

Task no. 2 was added in order to make sure that this group would try at least two of the music functions. Task no. 3, the pitch-matching game, was required in the first pilot study, with a simpler version of the game (program no. 9) added to the program library specifically for this group. Finally, the round problem, task no. 5 here, was reduced to making only a two-part performance. This simplification was effected because of the difficulty the first test group experienced in making a three-part round.

All thirteen students performed all five tasks. In most cases just about the entire length of the session was required to complete them all. Although the sessions were generally uneventful, there was one surprise. Evidently the author had described the five tasks to the class in very nearly the same words used in the list above. As a result the first three or four students showed up for their sessions carrying pieces of music that they had just taken out of the University's library (!).

Just as in the first study, each student filled out a questionnaire after finishing his/her individual session. The questionnaire form was exactly the same as that used in the first study.

Results of the individual sessions. All of the individual sessions proceeded smoothly and uneventfully. As was noted above, however, very little new information emerged from these sessions. Instead the sessions merely confirmed the earlier observation that the

time-sharing problems imposed a disappointingly low upper limit on the "energy level" of the overall man-machine system.

Nevertheless, one session proved to be moderately interesting. In this session it quickly became obvious that the student knew her way around the kinds of equipment used in the prototype. Though she had said that she had no previous experience with computers, she always seemed to know which button to press and when. To questioning she replied that she is a salesperson in a local department store and that her department contains electronic calculators, electronic games, and the complete line of Texas Instruments "electronic learning aids" (Speak and Spell, Dataman, Li'l Professor, etc.). Often when business is slow, she either takes out a new device and learns to use it or she experiments with one already familiar to her.

Results of the questionnaires. The questionnaires, like the individual sessions, elicited little new information. As with the first group, the majority (seven students) felt that the keyboard was easy to use, but only after a little practice. Two students found the CRT display confusing. There were no comments on the time-sharing problems, probably because the students had been warned about them in the introductory lecture.

The students' suggestions for improving the system essentially duplicated those offered by the first group. The most frequent suggestion, made by four students, was that more instruction and/or an instruction manual are needed. Since this comment was also made

by six of the eighteen students in the first study, it is clear that this is an area in need of immediate attention. The only entirely new proposal was a suggestion by two of the art majors that the keys should be color-coded.

The responses to question 3 emphasized basic musicianship, just as in the first study. Eight students mentioned ear-training as a use of the system. Five students said that the hand-held unit would be good for interesting children in music and introducing them to fundamental musical skills. Other uses suggested by individual members of the second group are included in the compilation in Appendix D.

In response to the first part of question 4, seven students said unconditionally that they would buy the hand-held unit while three others said they would buy it if the price were low enough. One student said he would not buy it, and two gave no answer at all. All but one of the students responded to the second part of question 4, which concerned the suggested price of the hand-held unit. The prices proposed ranged from \$20 to \$150, with the average being a little under \$70.

#### Concluding Note

Although the record-keeping facility of the prototype was active during all of the individual sessions in both pilot studies, the data gathered were not used for any purpose in the work reported here. The reasons for this are the following:

- 1) The data were collected only to provide a history of each student's interactions with the prototype. It was thought that these histories could be useful in determining the causes of any anomalies in system behavior and any errors consistently made by the study subjects. As it turned out the histories were not needed to solve the problems that actually arose.
- 2) The author had proposed no hypotheses which could be tested by analyzing the data.
- 3) The data are not a complete record of the pilot studies. About a third of the histories were lost because the record-keeping facility was not designed to withstand either interruptions of telephone service or equipment failures at the central computer site, both of which occurred more than once during the studies.
- 4) The problems of the time-sharing environment are so severe that the data gathered with the prototype can have little value in predicting how people would use the final hand-held system.

All of the data have nevertheless been preserved in hard-copy form and they are available for inspection.

## CHAPTER V

### SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS FOR FURTHER RESEARCH

#### Summary

The two music systems described here evolved in parallel over the course of the 1978-1979 academic year. The original concept of the theoretical system of course provided the fundamental guidelines for the development of the prototype. Once the prototype was built and running, however, the growing experience gained in using it began to reflect back on the theoretical system. As a result both systems metamorphosed quite far from the forms envisioned when this project was first proposed.

Before the design of the prototype finally crystallized (January 1979), it went through four distinct keyboard layouts, two revisions of the function repertoire, and one major change in internal architecture. After the prototype hardware was built and the APL simulation was completed in February 1979, the design underwent further changes as a result of the testing done during the pilot studies. All of these changes were reflected in appropriate ways in the design of the theoretical system. By the middle of May 1979, both systems had reached the form in which they are presented here.

The two pilot studies conducted during April 1979 generated a good deal of information about the prototype. As expected the first



pilot study uncovered some problems in the prototype. Among these were program bugs, confusing or inadequate information in the CRT display, and inconsistencies in the treatment of certain music system functions. To the extent that it was possible to do so, each problem was remedied as soon as it was found. As a result the participants in the second pilot study had a slightly better system to work with than their predecessors did.

There were, however, persistent problems that could not be corrected because they are inherent in the time-sharing environment in which the prototype is embedded. These time-sharing-related problems absorbed so much of the participants' attention that the goals of the second pilot study had to be scaled down. It had been hoped that the second study would provide an opportunity to observe the students' problem-solving behavior with the system. Because the prototype had proved to be more difficult to use than had been anticipated, however, the second pilot study became mainly an occasion to confirm and refine the information gathered during the first study.

### Conclusions

Two principal conclusions may be drawn from the work reported here:

- 1) On the whole the students who participated in the two pilot studies found the concept of a powerful instructional computer music system appealing. The responses observed during

the individual sessions and the answers given in the questionnaires bear evidence that the basic idea of the theoretical system struck a responsive chord in the majority of the students.

- 2) The prototype is not a good model of the theoretical system and, consequently, the data gathered in the course of testing the prototype cannot be directly used in improving the design of the theoretical system or in predicting how people would use the ultimate hand-held device. Instead most of the information generated by the pilot studies is primarily useful in pointing out areas within the prototype itself that could be improved to make it a better research tool.

The author underestimated the difficulties of modeling a real-time system in a time-sharing environment. During the planning and building of the prototype, the normal operating characteristics of time-sharing were expected to be nothing more than a minor nuisance. In reality these characteristics turned out to be major stumbling blocks and they dominated the subjects' perception of the prototype.

The overall effect of the time-sharing-related problems is to obscure the structure and functioning of the prototype. Ironically, the "string interpreter" feature, added to the prototype specifically to help people work more efficiently, actually contributed to this obscurity. The reason is that, because strings of commands could be entered at the keyboard before a SEND was finally transmitted,

confusion often arose over which of several observed musical effects to attribute to which of the several keystrokes.

The most disappointing effect of the time-sharing-related problems was that students were forced to work "by eye". The feebleness of the means for generating prompt acoustic feedback--particularly the non-playing keyboard and the non-playing STEP and BACK functions--frustrated all efforts to work by ear and compelled the students to rely primarily on visual information to navigate their way through the musical tasks.

In retrospect it is obvious that the prototype should have been built around a suitable dedicated mini- or micro-computer. Had this been done, the problems associated with time-sharing would never have arisen, and the resulting version of the prototype would have been a much more effective research tool.

The prototype as an instructional tool. Although the prototype was intended primarily as a vehicle for testing the basic design ideas of the theoretical system, it can also be considered as an instructional system in its own right. As such, the following conclusions concerning the prototype can be drawn from its observed performance during the pilot studies:

- 1) The prototype is a good medium for games, lessons, exercises, etc. that require the user to enter only a few notes at a time. The two- and three-note versions of the pitch-matching game, for example, were quite successful; throughout the pilot studies, the students found it easy to play these games. On

the other hand, the prototype is not well suited to any activity that requires the entry of more than, say, a dozen notes at a time, or that requires extensive changing or editing of notes. The means for entering and editing notes are simply too weak and unreliable: there is no substitute for the feedback provided by a "live" keyboard and audible cursor.

- 2) Because of the limited facilities for entering and editing notes, the prototype does not lend itself to the traditional procedures of musical composition. The prototype does, however, support a certain kind of compositional experimentation. It is easy for example, to create "new" compositions by retrieving pieces from the system's library and then subjecting them to transformation by such functions as INVERT, REVERSE, or SHUFFLE. Several of the people who tried this procedure expressed some interest in having more composition-generating functions.

Biases in the music system designs. Despite the obvious differences between the two systems, the theoretical system and the prototype share some significant family resemblances. In fact it can be said that the theoretical system is in some ways just a compact version of the prototype, not a fundamentally different animal. The reason for these similarities is that the same biases or assumptions underlie both designs.



The "scientific instrument" model. Both systems share a rather "square", scientific-instrument-like appearance. There is no reason why the systems must be this way; this particular model was adopted mainly because it provided a familiar starting point. Actually, if the work reported here is carried all the way through to successful realization of a hand-held music system, it is unlikely that the final device will have this rectangular configuration. Instead it is more likely to have a shape reminiscent of some traditional musical instrument or, possibly, a shape with flowing, biomorphic contours. Moreover the controls would almost certainly not be switches laid out in a rectangular grid. Instead a variety of pressure- and motion-sensitive controls would be employed, and they would be deployed in an arrangement designed to conform to the human hand.

The "composer" model. Both music systems are essentially devices for creating and manipulating musical scores which, however, have the peculiar property that they can play themselves via the PLAY function. Although these "scores" may consist of only a few notes, as do, for example, the typical responses required by game and lesson programs, the fact remains that all the resources of both systems are organized around one central activity, forming some desired musical object in the working area. Accordingly the present systems put the user in the role of composer, i.e., someone who is active during the creation of a work but relatively passive during performance. The user's role during musical performance by the present systems is in fact restricted to adjusting the overall tempo and volume of the music.



Some of the comments elicited during the pilot studies suggest that the "composer" model may be the wrong one for the system to embody. For example, the "marker" and "cue" ideas proposed by some of the students during the studies lead to a conception of the music system as a set of robot performers who can be given parts and told when to play. In such a system, the user would be cast in the very different role of conductor, i.e., someone who directs activities while they are happening.

Pitch bias. Both of the music systems make it much easier to deal with the pitch dimension of music than with its time dimension. This bias in favor of pitch phenomena can be seen in both of the following aspects of the systems:

- 1) The pitches of notes can be specified directly merely by striking the correct note keys. Durations, however, can be specified only by stepping note-by-note through a previously entered string of notes and applying the AUGMENT or DIMINISH function to each note in turn.
- 2) The music system provides six functions (INVERT, REVERSE, VERTICALIZE, SHUFFLE, RAISE, and LOWER) that are primarily concerned with pitch, but only two (AUGMENT and DIMINISH) concerned with time.

The design could be brought into better balance vis-a-vis the time dimension by providing a single "rhythm button". In data entry mode, users could record rhythms by tapping them out on this button. The recorded rhythms could then be manipulated and played independently,

or they could be combined with some existing pitch structure in the working area.

### Recommendations for Further Research

The results of the work reported here suggest that there is need for further research directed toward both the short-term goal of refining the original music system design and the long-term goal of producing the ultimate system. Five key areas of work are outlined below.

New prototype system. Design, build, and test a new prototype music system based in a dedicated mini- or micro-computer. The following hardware improvements might be included in the new system:

- 1) a more powerful synthesizer
- 2) real-time controls for tempo, volume, and pitch
- 3) graphic display of music
- 4) a "rhythm key" for recording note values
- 5) a keyboard color-coded by categories of functions

In addition the new prototype might make the privileged functions available either on the front panel or in some other readily accessible place. If this were done, the testing of the new device could include an experiment to determine if people who have no background in computer programming can understand the privileged functions and use them effectively.

Symbolic notation. Taking as a starting point the symbols presently used on the panels of both systems, develop a complete set of symbols to represent the operations of the music system. Couple this symbol set with some appropriate symbolic means for representing music itself. Keep in mind that both the musical and the operation symbols will ultimately have to be displayed in a tiny screen such as the one depicted in fig. 1.

Physical configuration. Design and test alternate physical configurations for the final system. Some possibilities are shapes derived from those of traditional musical instruments and shapes evolved purely from human factors and human engineering considerations. In any case begin with a different physical model from that of the electronic hand calculator.

Artificial intelligence features. Explore the possibility of building certain artificial intelligence features into the music system. Some good candidates are:

- 1) a facility that automatically corrects obvious errors made by the user. This facility would correct both system-type errors (e.g., incorrect command syntax) and musical errors (such as completely unmotivated chromatic notes in an otherwise wholly diatonic context).
- 2) self-instructional features to help users learn how to operate the system. The internal "teacher" might, for example, attempt to diagnose the causes of a particular kind

of error made by the user and then display a short warning message just before the user is about to make that error again.

- 3) an acoustic input facility that can transcribe music directly from live performance. This would permit music to be entered into the system by such means as playing, singing, whistling, or others. This facility might include a "normalizing" function to translate the acoustic input into the nearest approximation of standard pitches and note values.

"Conductor"-model system. Design a revised system which focuses on real-time manipulation of music. Many of the functions available in the present system could be converted into procedures that can be applied to music during performances. In addition new functions could be added to allow musical voices to be entered from the keyboard or brought in from memory while other music is already playing. The SHUFFLE function might be extended to include generation of random-note patterns in real time. This extension would form the basis of an automatic composition facility that users could control and experiment with as notes are playing.

## BIBLIOGRAPHY

- Abelson, H., N. Goodman, and L. Rudolph. LOGO Manual. LOGO Memo No. 7. Cambridge: MIT Artificial Intelligence Laboratory, 1974.
- Alles, H.G. A modular approach to building large digital synthesis systems. Computer Music Journal, 1977a, 1(4), 10-13.
- \_\_\_\_\_. A portable digital synthesis system. Computer Music Journal, 1977b, 1(4), 5-6.
- \_\_\_\_\_. A 256-channel performer input device. Computer Music Journal, 1977c, 1(4), 14-15.
- \_\_\_\_\_ and P. diGiugno. A one-card 64-channel digital synthesizer. Computer Music Journal, 1977, 1(4), 7-9.
- Alonso, S., J.H. Appleton, and C. Jones. A computer music system for every university. Creative Computing, 1977, 3(2), 57-60.
- Arenson, M. Computer-based ear-training instruction for non-music majors. Proceedings of the 1979 Winter Conference of the Association for the Development of Computer-Based Instructional Systems. San Diego, CA, February, 1979, 949-958.
- \_\_\_\_\_. An examination of computer-based educational hardware at twenty-eight NCCBIMI member schools. Journal of Computer-Based Instruction, 1978, 5(1,2), 38-40.
- Ashton, A.C. Electronics, music, and computers. Unpublished doctoral dissertation, University of Utah, 1970.
- Ashton, D.M. Design of an Educational Environment with a Computer-Controlled Organ. Salt Lake City: Intermountain Regional Medical Program, 1973. (EDRS No. ED 079 173).
- \_\_\_\_\_. Teaching music fundamentals using a computer, controlled organ, and display scope. Unpublished masters thesis, University of Utah, 1971.
- Bamberger, Jeanne. Capturing intuitive knowledge in procedural description. Artificial Intelligence Memo No. 38, LOGO Memo No. 42. Cambridge: MIT Artificial Intelligence Laboratory, 1976.
- \_\_\_\_\_. Developing a musical ear: A new experiment. Artificial Intelligence Memo No. 264, LOGO Memo No. 6. Cambridge: MIT Artificial Intelligence Laboratory, 1972.



- Bamberger, Jeanne. The development of musical intelligence I: Strategies for representing simple rhythms. Artificial Intelligence Memo No. 342, LOGO Memo No. 19. Cambridge: MIT Artificial Intelligence Laboratory, 1975.
- \_\_\_\_\_. The luxury of necessity. LOGO Memo No. 12. Cambridge: MIT Artificial Intelligence Laboratory, 1974a.
- \_\_\_\_\_. What's in a tune? LOGO Memo No. 13. Cambridge: MIT Artificial Intelligence Laboratory, 1974b.
- Baruzzi, G., P. Grossi, and M. Milani. Musical Studies: Summary of the Activity from 1967-1975. Pisa: CNUCE/CNR, 1975.
- Bayer, D. Real-time software for a digital synthesizer. Computer Music Journal, 1977, 1(4), 22-23.
- Bridgman, N. L'Etablissement d'un catalogue par incipit musicaux. Musica Disciplina, 1950, 4, 65.
- Bryden, J.R. and D.G. Hughes. An Index of Gregorian Chant, Volume II: Thematic Index. Cambridge: Harvard University Press, 1969.
- Deihl, N.C. Computer-assisted instruction and instrumental music: Implications for teaching and research. Journal of Research in Music Education, 1971, 19, 299-306.
- \_\_\_\_\_ and R.H. Ziegler. Evaluation of a CAI program in articulation, phrasing, and rhythm for intermediate instrumentalists. Council for Research in Music Education Bulletin, 1973, 31, 1-11.
- Goldberg, A. and A.C. Kay. Teaching Smalltalk. Palo Alto: Xerox Palo Alto Research Center, 1977.
- Grossi, P. Studi Musicali: Modalita Operative del TAUMUS, Software di Gestione del Terminale Audio TAU2. Pisa: CNUCE/CNR, 1976.
- \_\_\_\_\_ and G. Sommi. Studi Musicali: DCMP--Versione per il Sistema 360/67. Pisa: CNUCE/CNR, 1974.
- Herrold, R.M. Computer-assisted instruction in music: A study of student performance in the Stanford ear-training program. Unpublished doctoral dissertation, Stanford University, 1974.
- Hiller, L. Some compositional techniques involving the use of computers. Music by Computer. Ed. H. von Forster. New York: John Wiley, 1969, 71-83.
- Hodges, D.A. Microelectronic Memories. Scientific American, 1977, 237(3), 130-145.

Hofstetter, F.T. Computer-based recognition of perceptual patterns in harmonic dictation exercises. Journal of Research in Music Education, 1978, 26, 111-119.

\_\_\_\_\_. Controlled evaluation of a competency-based approach to teaching aural interval identification. Proceedings of the 1979 Winter Conference of the Association for the Development of Computer-Based Instructional Systems, San Diego, CA, February, 1979a, 935-948.

\_\_\_\_\_. Foundation, organization, and purposes of the National Consortium for Computer-Based Musical Instruction. Journal of Computer-Based Musical Instruction, 1976, 3(1), 21-33.

\_\_\_\_\_. GUIDO: An interactive computer-based system for instruction and research in ear-training. Journal of Computer-Based Instruction, 1975, 1(4), 100-106.

\_\_\_\_\_. Interactive simulation/games as an aid to musical learning. Proceedings of the 1977 Winter Conference of the Association for the Development of Computer-Based Instructional Systems. Wilmington, DE, February, 1977a, 104-117.

\_\_\_\_\_. Microelectronics and music education. Music Educators Journal, 1979b, 65(8), 39-45.

\_\_\_\_\_. Music dream machines: New realities for computer-based musical instruction. Creative Computing, 1977b, 3(2), 50-54.

Hultberg, M.L., W.E. Hultberg and T. Tenny. Project CLEF: CAI in music theory--update 1979. Proceedings of the 1979 Winter Conference of the Association for the Development of Computer-Based Instructional Systems. San Diego, CA, February, 1979, 928-934.

Ihrke, W.R. A Study of the Present State of Electronic Music Training, Including Computer-Assisted Instruction: A Bibliography. Final Report. Washington, D.C.: National Center for Educational Research and Development, 1972. (EDRS No. ED 063 806).

Jones, M.J. Computer-assisted instruction in music: A survey with attendant recommendations (Doctoral dissertation, Northwestern University, 1976). Dissertation Abstracts International, 1976, 36, 7264A-7265A. (University Microfilms No. 76-11,899)

Kay, A.C. Microelectronics and the personal computer. Scientific American, 1977, 237(3), 231-244.

Killam, R., P. Lorton and E. Schubert. Interval recognition: A study of student accuracy of identification of harmonic and melodic internals. Journal of Music Theory, 1975, 19(2), 212-234.

- Knowlton, P.H. Capture and display of keyboard music. Datamation, 1972, 18(5), 56-60.
- \_\_\_\_\_. Interactive communication and display of keyboard music. Unpublished doctoral dissertation, University of Utah, 1971.
- Kuhn, W.E. Computer-assisted instruction in music: Drill and practice in ear-training. College Music Symposium, 1974, 14, 89-101.
- Lancaster, D.E. TTL Cookbook. Indianapolis: Howard W. Sams, 1974.
- Lawson, J. and M.V. Mathews. Computer program to control a digital real-time synthesizer. Computer Music Journal, 1977, 1(4), 16-21.
- Mathews, M.V. The Technology of Computer Music. Cambridge: MIT Press, 1969.
- Nelson, G. MPL: A program library for musical data processing. Creative Computing, 1977, 3(2), 76-81.
- Papert, S. Teaching children thinking. IFIP Conference on Computer Education. Amsterdam: North-Holland, 1970.
- \_\_\_\_\_. Teaching children to be mathematicians versus teaching about mathematics. International Journal of Mathematical Education in Science and Technology, 1972, 3, 249-262.
- Peters, G.D. Courseware development for micro-computer based education in music. Proceedings of the 1979 Winter Conference of the Association for the Development of Computer-Based Instructional Systems. San Diego, CA, February, 1979, 922-927.
- \_\_\_\_\_. The complete computer-based music system: A teaching system—a musician's tool. Proceedings of the 1977 Winter Conference of the Association for the Development of Computer-Based Instructional Systems. Wilmington, DE, February, 1977, 93-100.
- \_\_\_\_\_. The development of computer-assisted instructional materials for use in the teaching of instrumental music via PLATO IV. Final Report, Undergraduate Instructional Award. Urbana-Champaign: University of Illinois, 1975.
- \_\_\_\_\_ and J.M. Eddins. Applications of computers to music pedagogy, analysis, and research: A selected bibliography. Journal of Computer-Based Instruction, 1978, 5(1,2), 41-44.

- Placek, R.W. Design and trial of a computer-assisted lesson in rhythm. Journal of Research in Music Education, 1974, 22, 13-23.
- Snell, J. Computer music bibliography. Creative Computing, 1977, 3(2), 54-56.
- Taylor, J.A. and J.W. Parrish. A national survey on the uses of, and attitudes toward programmed instruction and computers in public school and college music education. Journal of Computer-Based Instruction, 1978, 5(1,2), 11-21.
- Texas Instruments. Electronic Calculators and Learning Aids from Texas Instruments. Houston: Texas Instruments Inquiry Answering Service, 1978.
- Vaughn, A.C. A study of the contrast between computer-assisted instruction and the traditional teacher/learner method of instruction in basic musicianship (Doctoral dissertation, Oregon State University, 1977). Dissertation Abstracts International, 1977, 38, 3357A. (University Microfilms No. 77-25,414)
- Xenakis, I. Formalized Music. Bloomington: Indiana University Press, 1971.
- Xerox PARC Learning Research Group. Personal Dynamic Media. Palo Alto: Xerox Palo Alto Research Center, 1976.

A P P E N D I C E S



## A P P E N D I X A

### MUSIC FUNCTION SUMMARY

This appendix outlines all of the music system functions. The information provided for each function is as follows:

- 1) function - the name of the procedure and/or a brief phrase that describes its action(s). The name is followed by the graphic symbol (if any) associated with the function in the theoretical and prototype systems.
- 2) mnemonic - the three-letter (upper case) identifier used to designate the function in the APL-based prototype system (if the identifier is preceded by a lower case "n", this indicates that the function takes a numeric argument)
- 3) description - an account of the principal action(s) performed by the function
- 4) error conditions - a list of the circumstances (if any) under which the system will abort the execution of the function

#### Regular Functions

---

function:      AUGMENT      ←————→

mnemonic:      nAUG

description: AUG multiplies the duration of the event (note or rest) immediately to the right of the cursor by a factor of n. If n is not specified, the traditional value of 2 is assumed. n=0 is legal but it will produce unpredictable musical results. AUG can be applied to all of

the events in the working area simultaneously by executing the END function prior to entering n and AUG.

error  
conditions: none

---

function: DIMINISH     

mnemonic: nDIM

description: DIM multiplies the duration of the event (note or rest) immediately to the right of the cursor by a factor of  $1/n$ . If n is not specified, the traditional value of 2 is assumed. DIM can be applied to all of the events in the working area simultaneously by executing the END function prior to entering n and DIM.

error  
conditions: DIM is aborted if  $n = 0$ .

---

function: RAISE (transpose up)     


mnemonic: nRAI

description: RAI transposes the note immediately to the right of the cursor up n semitones. If n is not specified, it is assumed to be 1. RAI can be applied to all of the notes in the working area simultaneously by executing the END function prior to entering n and RAI.

error

conditions: RAI is aborted if completion of the operation would produce one or more notes outside the range of the system's music synthesizer.

---

function: LOWER (transpose down) 


mnemonic: nLOW

description: LOW transposes the note immediately to the right of the cursor down n semitones. If n is not specified, it is assumed to be 1. LOW can be applied to all of the notes in the working area simultaneously by executing the END function prior to entering n and LOW.

error

conditions: LOW is aborted if completion of the operation would produce one or more notes outside the range of the system's music synthesizer.

---

function: INVERT (mirror inversion) 

mnemonic: INV


description: INV inverts the pitches of all the notes in the working area about the first pitch.

error

conditions: INV is aborted if completion of the operation would produce one or more notes outside the range of the music system's synthesizer.

---

---


function: REVERSE (retrograde) 

mnemonic: REV

description: REV reverses the order of the events (notes and rests) in the working area.

error conditions: none

---

function: VERTICALIZE 


mnemonic: nVER

description: VER reorganizes the events (notes and rests) in the working area into a series of groups--single notes, intervals, triads, or chords--consisting of n simultaneous events each. If n is not specified, it is assumed to be 1.

error conditions: VER is aborted under either of the following circumstances:

- 1)  $n = 0$ .
- 2) the working area cannot be partitioned into a whole number of n-event groups.

---

function: SHUFFLE 

mnemonic: nSHU

description: SHU randomly reorders the events (notes and rests) in the working area but leaves the order within each group of n events intact. SHU can therefore be used to scramble

sequences of single notes, intervals, triads, or chords. If  $n$  is not specified, it is assumed to be 1.

error

conditions: SHU is aborted under either of the following circumstances:

- 1)  $n = 0$ .
- 2) the working area cannot be partitioned into a whole number of  $n$ -event groups.

---

function: STORE

mnemonic: nSTO

description: STO stores the contents of the working area in user memory location  $n$ . If  $n$  is not specified, location 0 is assumed. The previous contents of location  $n$  are stored in the backup area.

error

conditions: STO is aborted if  $n$  is not in the range 0-9.

---

function: RECALL

mnemonic: nRCL

description: RCL inserts the contents of user memory location  $n$  into the working area at the current position of the cursor. If  $n$  is not specified, location 0 is assumed.

error

conditions: RCL is aborted if  $n$  is not in the range 0-9.

---



---

function: COMBINE **I**

mnemonic: nCOM

description: COM sort-merges the events (notes and rests) in user memory location n into the events in the working area. If n is not specified, location 0 is assumed. The key for the sort operation is the starting time of each event. Each time COM is invoked, the events brought in from the user memory location are assigned to an audio output channel which has not been allocated to any set of events already stored in the working area. COM assumes that the sequence of events in the user memory location represents a single musical voice. If this is not the case, the musical results of COM will be unpredictable.

error

conditions: COM is aborted under either of the following circumstances:

- 1) n is not in the range 0-9.
  - 2) the total number of musical voices represented by the combined contents of the working area and the user memory location exceeds the number of audio output channels available in the system's music synthesizer.
-

---

function: MATCH    **==**

mnemonic: nMAT

description: MAT performs a note-by-note, rest-by-rest comparison of the working area and user memory location n. If n is not specified, location 0 is assumed. If the pitches and durations of corresponding notes and the durations of corresponding rests are identical, the MATCH flag is set and "MATCH!" is displayed. If one or more notes or rests do not match, the MATCH flag is cleared and "NO MATCH" is displayed.

error

conditions: MAT is aborted if n is not in the range 0-9.

---

function: STEP    **▶**

mnemonic: nSTE

description: STE advances the cursor n events (notes and rests) through the working area. If n is not specified, it is assumed to be 1. Each note is played as it is traversed by the cursor (rests are "played" silently). When the cursor is positioned after the last event in the working area, calls to STE have no effect.

error

conditions: none

---

---

function: BACK (backspace) ◀

mnemonic: nBKS

description: BKS moves the cursor back n events (notes and rests) through the working area. If n is not specified, it is assumed to be 1. Each note is played as it is traversed by the cursor (rests are "played" silently). When the cursor is positioned before the first event in the working area, calls to BKS have no effect.

error  
conditions: none

---

function: RESET (home the cursor) ◀◀

mnemonic: RST

description: RESET moves the cursor to a point immediately before the first event (note or rest) in the working area.

error  
conditions: none

---

function: END ▶▶

mnemonic: END

description: END moves the cursor to a point immediately after the last event (note or rest) in the working area. END is also used just prior to calls to RAI, LOW, AUG, and DIM to indicate to the system that these functions are to operate on the entire contents of the working area.

error  
conditions: none

---

function: LIBRARY (retrieve music file)

mnemonic: nLIB

description: LIB retrieves a specified musical piece from an external read-only memory connected to the system and loads the piece into the working area. If a valid n is specified, LIB accesses the n-th item in the library directory. If n is not specified, the system assumes that the first eight notes in the working area are the incipit of the desired piece. LIB extracts the melodic intervals between these notes and then searches the library directory for a piece beginning with these same intervals. If the working area contains fewer than eight notes, or if fewer than seven intervals match any incipit in the library directory, LIB will retrieve the piece whose incipit most nearly matches the notes in the working area.

error  
conditions: LIB is aborted under either of the following

circumstances:

- 1) neither n nor an incipit is specified.
  - 2) n is greater than the number of entries in the library directory.
-

---


function: PROGRAM (retrieve program file)  
mnemonic: nPRG  
description: PRG retrieves the n-th program stored in an external read-only memory connected to the system and loads it into the system's internal program buffer. The previous contents of the program buffer are lost.

## error

conditions: PRG is aborted under either of the following circumstances:

- 1) n is not specified.
- 2) n is greater than the number of entries in the program library directory.

---

function: PLAY   
mnemonic: PLA  
description: Starting from the event (note or rest) immediately to the right of the cursor, PLA decodes and then plays each of the notes in the working area on the system's music synthesizer. The pitches of the notes will be played exactly as specified, but starting times and durations will vary according to the current setting of the TEMPO control.

## error

conditions: none


---




---

function: GO (execute)  
mnemonic: XCT  
description: XCT initiates execution of the program presently stored in the program buffer. If the program buffer is empty, XCT has no effect.  
error conditions: none

---

function: CLEAR   
mnemonic: CLR  
description: CLR deletes the entire contents of the working area and repositions the cursor to a point immediately before the beginning of the working area.  
error conditions: none

---

function: DROP   
mnemonic: nDRO  
description: DRO deletes n events (notes and rests) from the working area, beginning with the event immediately to the right of the cursor. If n is not specified, it is assumed to be 1. Any calls to DRO when the working area is empty have no effect.  
error conditions: none

---

---


function: DEFINE

mnemonic: DEF

description: DEF switches the system back and forth between direct execution mode and program definition mode. Each time DEF is executed, it complements the logical state of the program flag.

error conditions: none

---


function: DATA 

mnemonic: DAT

description: DAT sets the system's internal flag, which switches the system into data entry mode.

error conditions: none

---

function: INSERT 

mnemonic: INS

description: INS translates the character data in the keyboard buffer into note-matrix form and inserts the note-matrix into the working area at the point indicated by the cursor. It then clears the keyboard buffer and clears the data flag, which switches the system from data entry mode back into direct execution mode.

error conditions: none

---

---

function: UNDO

mnemonic: UND

description: UND exactly reverses the effects of the last function executed. UND is therefore a sort of universal inverse function. UND is not defined for PLA, STE, BKS, RST, or END, or for the privileged functions.

error conditions: none

---

#### Privileged Functions

---

function: preset the counter

mnemonic: nSEC

description: SEC loads the number n into the counter. If n is not specified, it is assumed to be 0.

error conditions: none

---

---

function: decrement the counter

mnemonic: DEC

description: DEC subtracts 1 from the number in the counter register. If the counter register contains zero, the DEC function causes the counter to "wrap around" to its maximum value.

error conditions: none

---

---

function: enable the display

mnemonic: DON

description: DON enables the system's display. In the enabled state, the display will show any information sent to it (e.g., the basic pitch and time data of each note as it is played on the system's music synthesizer).

error  
conditions: none

---

function: disable the display

mnemonic: DOF

description: DOF disables the system's display. In the disabled state, the display will not show any of the information sent to it.

error  
conditions: none

---

function: start the timer

mnemonic: STT

description: STT clears the timer register to zero. Since the timer is directly connected to the system's tempo clock, the timer will begin counting again immediately following the execution of the STT function.

error  
conditions: none

---

---

function: display the timer

mnemonic: DST

description: DST displays the current contents of the timer register on the system's display (provided that the display has been enabled). Since the tempo clock rate is controlled externally by the user, the number displayed by DST must be interpreted in relation to the current tempo setting. When the TEMPO knob is set to 60 beats per minute (its "normal" setting), the display reads directly in hundredths of a second. At any other tempo setting, the displayed number must be multiplied by  $60/\text{TEMPO}$  to obtain a value in hundredths of a second.

error

conditions: none

---

function: jump to executive

mnemonic: JPX

description: JPX causes an immediate jump out of the piece being performed or the currently executing program and returns control to the "wait" step of the executive routine.

error

conditions: none

---



---

function: NOTE (load music constant)

mnemonic: NOT

description: NOT permits music-type data that are incorporated within a program to be read directly into the keyboard buffer while the program is executing. NOT sets the data flag, but it does not engage the mechanism that causes the system to hang up at the "wait" step of the executive routine. Instead, after the note function is executed, successive music characters are simply copied from the program buffer into the keyboard buffer (all non-music characters except the one representing INSERT are ignored). This process continues until the character associated with INSERT is encountered.

error  
conditions: none

---

Branch functions. The branch functions form a distinct subgroup within the privileged group. Under the appropriate conditions, each of the branch functions causes a transfer of control to a program step located a specified number of keystrokes before or after the step containing the branch function. The "target" of a branch function is therefore found simply by counting keystrokes forward or backward from the branch step. Consider, for example, the following program segment:

DAT

1 MAT

4 BBN

If the music entered following execution of the DAT step fails to match the contents of user memory location 1, the BBN function (Branch Back on Not-match) will cause a transfer of control back four keystrokes to the DAT step. Note that the branch step itself is not counted in determining the target of the branch.

All of the branch functions have identical error conditions. Execution of any branch function is aborted under any of the following circumstances:

- 1) the number of keystrokes to be skipped, n, is not specified.
- 2) the specified number of keystrokes to be skipped would cause a branch to a point before the beginning or after the end of the current program.
- 3)  $n = 0$  (i.e., the function causes a branch back to itself).

---

function: unconditional branch backward

mnemonic: nBRB

description: BRB causes a transfer of control back to the program step which is -n keystrokes from the one containing this function.

error  
conditions: (see discussion above)

---

---

function: unconditional branch forward

mnemonic: nBRF

description: BRF causes a transfer of control forward to the program step which is n keystrokes from the one containing this function.

error

conditions: (see discussion above)

---

function: branch forward on counter > 0

mnemonic: nBFP

description: If the number in the counter register is greater than zero, BFP causes a transfer of control forward to the program step which is n keystrokes from the one containing this function. If the number in the counter register is not greater than zero, the next program step in sequence is executed.

error

conditions: (see discussion above)

---

function: branch back on counter > 0

mnemonic: nBBP

description: If the number in the counter register is greater than zero, BBP causes a transfer of control back to the program step which is -n keystrokes from the one containing this function. If the number in the counter

register is not greater than zero, the next program step in sequence is executed.

error

conditions: (see discussion above)

---

function: branch forward on counter = 0

mnemonic: nBFZ

description: If the counter register contains zero, BFZ causes a transfer of control forward to the program step which is n keystrokes from the one containing this function. If the counter register contains anything other than zero, the next program step in sequence is executed.

error

conditions: (see discussion above)

---

function: branch back on counter = 0

mnemonic: nBBZ

description: If the counter register contains zero, BBZ causes a transfer of control back to the program step which is -n keystrokes from the one containing this function. If the counter register contains anything other than zero, the next program step in sequence is executed.

error

conditions: (see discussion above)

---

---

function: branch forward on match

mnemonic: nBFM

description: If the match flag is set, BFM causes a transfer of control forward to the program step which is n key-strokes from the one containing this function. If the match flag is clear, the next program step in sequence is executed.

error

conditions: (see discussion above)

---

function: branch back on match

mnemonic: nBBM

description: If the match flag is set, BBM causes a transfer of control back to the program step which is -n keystrokes from the one containing this function. If the match flag is clear, the next program step in sequence is executed.

error

conditions: (see discussion above)

---

function: branch forward on not-match

mnemonic: nBFN

description: If the match flag is clear, BFN causes a transfer of control forward to the program step which is n key-strokes from the one containing this function. If



the match flag is set, the next program step in sequence is executed.

error  
conditions: (see discussion above)

---

function: branch back on not-match

mnemonic: nBBN

description: If the match flag is clear, BBN causes a transfer of control back to the program step which is -n key-strokes from the one containing this function. If the match flag is set, the next program step in sequence is executed.

error  
conditions: (see discussion above)

---

A P P E N D I X B

LISTINGS OF THE APL SIMULATION

Global Constants and Variables

ASCII	a vector containing the ASCII characters that select the pitches of the tones produced by the synthesizer. The characters are arranged in ascending chromatic order starting from C (the octaves of tones are selected by the appropriate ASCII digit codes, 0-7).
BADKEY	a logical flag that is set whenever the simulation receives a character which is not an element of KEYS (q.v.)
CHANNEL	a vector containing the ASCII characters that select the audio output channels of the synthesizer
CLEAR	logical (boolean) zero
CONTROL	a numeric vector containing the indices of the characters associated with PLAY and the cursor functions in the character vector KEYS (q.v.)
COUNTER	the loop counter
CURSOR	the cursor
DATAFLAG	the data flag
ERRFLAG	the error flag

FILL a do-nothing "padding" character used to fill out the durations of notes played on the synthesizer

FMATRIX a character matrix containing the 3-letter mnemonic names of all the regular and privileged functions

FUNCTION a numeric scalar variable which is used to hold the index of the row in FMATRIX that contains the name of the next function to be executed

INPUT the character vector used both to receive keyboard input from the user and to hold the currently executing program

KBDBUF the keyboard buffer

KEY the numeric index in KEYS (q.v.) of the character most recently entered at the keyboard or read from a program string

KEYS a character vector containing all the characters recognized by the simulation. The associations of the characters are:

- 1-13: ascending chromatic scale, C-c
- 14-15: low- and high-octave keyboard registers
- 16-25: ASCII 0-9
- 26-69: regular and privileged functions

LIGHTFLAG a logical flag that is set to enable the displays generated by the system and cleared to disable them

MATCHFLAG the match flag

MODEFLAG the program flag

NBUF the number buffer

NOTES a matrix of the pitch characters used in the note displays generated by PLAY, STEP, and BACK

NUMBER a numeric scalar variable that contains the numeric parameter (if any) of the function about to be executed

NUMFLAG a logical flag which, when set, indicates that at least one digit character is in the number buffer

OCTAVE a two-element numeric vector containing the offsets used to place pitches entered from the keyboard into either the lower or upper register

OFF a character vector containing a sequence of ASCII characters used to turn all of the synthesizer's audio output channels off

OFFSET a numeric value chosen from OCTAVE (q.v.) which is added to the value of KEY when it represents a music-type character to obtain a patch number in the correct octave for notes entered at the keyboard

ON a character vector containing a sequence of ASCII characters used to turn all of the synthesizer's audio output channels on

PCOUNT the number of music characters presently in the keyboard buffer

PREVFUNC the numeric index of the row in FMATRIX that contains the name of the most recently executed function

PREVNUM a numeric scalar that contains the numeric parameter (if any) of the most recently executed function

PROGBUF the program buffer

RECNO the number of the next record to be saved by the simulation's record-keeping facility

RECORD a numeric vector that stores all the values of KEY during a session with the simulation (the record-keeping facility stores -KEY for any operation that generates an error condition)

RHOX the number of events (notes and rests) currently in the working area

SET logical (boolean) one

SIGN a two-element numeric vector used to tag the numeric KEY codes as they are stored in RECORD



TEMP a temporary storage area used to hold the unexecuted portion of a program string while the program pauses to accept input from the user (DAT copies the string into TEMP from INPUT, and INS copies the string back into INPUT)

TEMPO a numeric scalar variable that contains the current tempo setting (in beats per minute)

U the backup area

UMATRIX a character matrix containing the 3-letter mnemonic names of the functions used to reverse the effects of each of the functions named in FMATRIX

V0 - V9 the user memory locations

X the working area

XEMPTY a logical flag that is set whenever the working area is empty

```

▽ASK[ ]▽
▽Z←ASK B
[1]   ▢←B
[2]   Z←(√\B≠' ')/B+▢
▽

```

```

▽AUG[ ]▽
▽AUG;A
[1]   PREVNUM←1↑NUMBER,2
[2]   →XEMPTY/0
[3]   GETARG
[4]   A TMULT PREVNUM
[5]   U←10
▽

```

```

▽BBCHECK[ ]▽
▽BBCHECK S
[1]   →(0=×/ρNUMBER)/ERR1
[2]   →(NUMBER=0)/ERR2
[3]   RHOPB←ρPROGBUF
[4]   RHOI←ρINPUT
[5]   →((NUMBER+1)>RHOPB-RHOI)/ERR3
[6]   →0
[7]   ERR1: ' ### ',S,' NUMBER MISSING'
[8]   ERRFLAG←SET
[9]   →0
[10]  ERR2: ' ### ',S,' BY ZERO NOT ALLOWED'
[11]  ERRFLAG←SET
[12]  →0
[13]  ERR3: ' ### ',S,' OUT OF RANGE'
[14]  ERRFLAG←SET
▽

```

```

▽BBM_ ]▽
▽BBM;RHOPB;RHOI
[1]   BBCHECK 'BBM'
[2]   →ERRFLAG/0
[3]   PREVNUM←NUMBER
[4]   U←10
[5]   →(~MATCHFLAG)/0
[6]   BRANCHBACK
▽

```

```

▽BBN[□]▽
▽BBN;RHOPB;RHOI
[1]  BBCHECK- 'BBN'
[2]  →ERRFLAG/0
[3]  PREVNUM←NUMBER
[4]  U←10
[5]  →MATCHFLAG/0
[6]  BRANCHBACK
▽

```

```

▽BBP[□]▽
▽BBP;RHOPB;RHOI
[1]  BBCHECK'BBP'
[2]  →ERRFLAG/0
[3]  PREVNUM←NUMBER
[4]  U←10
[5]  →(COUNTER≤0)/0
[6]  BRANCHBACK
▽

```

```

▽BBZ[_]▽
▽BBZ;RHOPB;RHOI
[1]  BBCHECK 'BBZ'
[2]  →ERRFLAG/0
[3]  PREVNUM←NUMBER
[4]  U←10
[5]  →(COUNTER≠0)/0
[6]  BRANCHBACK
▽

```

```

▽BFCHECK[□]▽
▽BFCHECK S
[1]  →(0=×/ρNUMBER)/ERR1
[2]  →(NUMBER>ρINPUT)/ERR2
[3]  →(NUMBER<1)/ERR3
[4]  →0
[5]  ERR1: ' xxx ',S,' NUMBER MISSING'
[6]  ERRFLAG←SET
[7]  →0
[8]  ERR2: ' xxx ',S,' OUT OF RANGE'
[9]  ERRFLAG←SET
[10] →0
[11] ERR3: ' xxx ',S,' BY ZERO NOT ALLOWED'
[12] ERRFLAG←SET
▽

```

```

▽BFM[□]▽
▽BFM
[1]   BFCHECK 'BFM'
[2]   →ERRFLAG/0
[3]   PREVNUM←NUMBER
[4]   U←10
[5]   →(∼MATCHFLAG)/0
[6]   BRANCHFWD
▽

```

```

▽BFN[□]▽
▽BFN
[1]   BFCHECK 'BFN'
[2]   →ERRFLAG/0
[3]   PREVNUM←NUMBER
[4]   U←10
[5]   →MATCHFLAG/0
[6]   BRANCHFWD
▽

```

```

▽BFP[□]▽
▽BFP
[1]   BFCHECK 'BFP'
[2]   →ERRFLAG/0
[3]   PREVNUM←NUMBER
[4]   U←10
[5]   →(COUNTER≤0)/0
[6]   BRANCHFWD
▽

```

```

▽BFZ[□]▽
▽BFZ
[1]   BFCHECK 'BFZ'
[2]   →ERRFLAG/0
[3]   PREVNUM←NUMBER
[4]   U←10
[5]   →(COUNTER≠0)/0
[6]   BRANCHFWD
▽

```

```

      .KS[ ]▽
      ▽BKS;NUM;COUNT
[1]   →XEMPTY/0
[2]   NUM←1↑NUMBER,1
[3]   COUNT←0
[4]   ON
[5]   L:→(CURSOR≤1)/0
[6]   COUNT←COUNT+1
[7]   →(COUNT>NUM)/0
[8]   CURSOR←CURSOR-1
[9]   SSPLAY
[10]  →L
      ▽

```

```

      .BRB[ ]▽
      ▽BRB;RHOPB;RHOI
[1]   BBCHECK 'BRB'
[2]   →ERRFLAG/0
[3]   PREVNUM←NUMBER
[4]   U←10
[5]   BRANCHBACK
      ▽

```

```

      ▽BRF[ ]▽
      ▽BRF
[1]   BFCHECK 'BRF'
[2]   →ERRFLAG/0
[3]   PREVNUM←NUMBER
[4]   U←10
[5]   BRANCHFWD
      ▽

```

```

      .BRANCHBA K[ ]▽
      ▽BRANCHBACK
[1]   INPUT←((-NUMBER+1)↑(-RHOI)↑PROGBUF),INPUT
      ▽

```

```

      .BRANCHFWD[ ]▽
      ▽BRANCHFWD
[1]   INPUT←(NUMBER-1)↑INPUT
      ▽

```



```

∇CLR[ ]∇
∇CLR
[1]   PREVNUM←10
[2]   U←X
[3]   X←10
[4]   CURSOR←1
∇

∇COM[ ]∇
∇COM;T;IO4;NCHX;RONCHX;FREECH
[1]   ERRORCHECK
[2]   →ERRFLAG/0
[3]   PREVNUM←NUMBER
[4]   U←X
[5]   ⚡'T←V',∇NUMBER
[6]   →(0=×/ρT)/0
[7]   IO4←14
[8]   NCHX←(IO4∈X[;2])/IO4
[9]   RONCHX←ρNCHX
[10]  →(4<RONCHX+1)/ERR
[11]  FREECH←(~(IO4∈NCHX))/IO4
[12]  T[;2]←1↑FREECH
[13]  X←T;X
[14]  X←X[ΔX[;1];]
[15]  →0
[16]  ERR: ' ≠≠≠ TOO MANY VOICES'
[17]  ERRFLAG←SET
∇

```

```

. DAT[ ]∇
∇DAT
[1]   TEMP←INPUT
[2]   INPUT←''
[3]   ' << ENTER DATA'
[4]   DATAFLAG←SET
∇

```

```

∇DEC[ ]∇
∇DEC
[1]   U←PREVNUM←10
[2]   COUNTER←COUNTER-1
∇

```

```

        .DECODE[□]▽
        ▽DECODE;PITCH
[1]      →(KEY>69)/ERROR
[2]      →(KEY>13)/NEXT1
[3]      →KBDFLAG/A
[4]      KBDBUF←10
[5]      PCOUNT←0
[6]      KBDFLAG←SET
[7]      A:PITCH←(KEY≠13)×OFFSET+KEY
[8]      KBDBUF←KBDBUF,(30×PCOUNT),1,PITCH,30
[9]      PCOUNT←PCOUNT+1
[10]     RECORD←RECORD,KEY×SIGN[ERRFLAG+1]
[11]     →0
[12]     NEXT1:→(KEY>15)/NEXT2
[13]     OFFSET←OCTAVE[KEY-13]
[14]     →KBDFLAG/0
[15]     KBDFLAG←SET
[16]     RECORD←RECORD,KEY×SIGN[ERRFLAG+1]
[17]     →0
[18]     NEXT2:→(KEY>25)/NEXT3
[19]     NBUF←NBUF,KEY-16
[20]     →NUMFLAG/0
[21]     NUMFLAG←SET
[22]     RECORD←RECORD,KEY×SIGN[ERRFLAG+1]
[23]     →0
[24]     NEXT3:FUNCTION←KEY-25
[25]     →(~NUMFLAG)/NEXT4
[26]     NUMBER←NBUF
[27]     NBUF←''
[28]     NUMFLAG←CLEAR
[29]     NEXT4:→(FUNCTION≠11)/0
[30]     KBDFLAG←CLEAR
[31]     →0
[32]     ERROR:BADKEY←SET
        ▽

```

```

        .DEF[□]▽
        ▽DEF
[1]      PREVNUM←10
[2]      U←10
[3]      MODEFLAG←~MODEFLAG
[4]      →(~MODEFLAG)/0
[5]      PROGBUF←''
        ▽

```

```

∇DIM[ ]∇
∇DIM;A
[1]   PREVNUM←1↑NUMBER,2
[2]   U←10
[3]   →XEMPTY/0
[4]   →(PREVNUM=0)/ERR
[5]   GETARG
[6]   A TMULT÷PREVNUM
[7]   →0
[8]   ERR: ' ≠≠ DIMINUTION BY ZERO NOT ALLOWED'
[9]   ERFLAG←SET
∇

```

```

.DOF[ ]∇
∇DOF
[1]   LIGHTFLAG ← CLEAR
∇

```

```

.DON[ ]∇
.DON
[1]   LIGHTFLAG ← SET
∇

```

```

∇DRO[ ]∇
∇DRO;DUR;CM1;N
[1]   PREVNUM ← 1 ↑ NUMBER,1
[2]   U←X
[3]   →(PREVNUM=0)/0
[4]   →(XEMPTY∨CURSOR>RHOX)/0
[5]   N ← 0
[6]   L1: N ← N+1
[7]   →(N > PREVNUM)/0
[8]   →(RHOX=1)/L2
[9]   DUR←X[CURSOR;4]
[10]  CM1←CURSOR-1
[11]  X←((CM1,4)↑X)∓(CURSOR,0)↑X
[12]  X[;1]←(CM1↑X[;1]),(CM1↑X[;1])-DUR
[13]  X[;1]←X[;1]-X[1;1]
[14]  RHOX ← RHOX-1
[15]  → L1
[16]  L2: X ← 1 0
∇

```

```

∇DST[□]∇
∇DST;SECONDS;MINUTES
[1] SECONDS←□TS[6]
[2] MINUTES←□TS[5]
[3] →(SECONDS≥TIMER[2])/DISP
[4] SECONDS←SECONDS+60
[5] MINUTES←MINUTES-1
[6] DISP:''
[7] ' TIME × ',(∇(SECONDS-TIMER[2])+60×MINUTES-
TIMER[1]),' SECONDS'
[8] ''
∇

```

```

∇DUM[□]∇
∇DUM
[1] →0
∇

```

```

∇END[□]∇
∇END
[1] CURSOR←RHOX+1
∇

```

```

∇ERRORCHECK[□]∇
∇ERRORCHECK
[1] →(0≠×/ρNUMBER)/L
[2] NUMBER ← 0
[3] L: →(NUMBER>9)/ERR
[4] →0
[5] ERR: ' ≠≠ ILLEGAL MEMORY NUMBER'
[6] ERRFLAG←SET
∇

```

```

∇GETARG[□]∇
∇GETARG
[1] A←CURSOR
[2] →(CURSOR≤RHOX)/0
[3] A←\RHOX
∇

```

```

      .INS[ ]▽
      ▽INS;T
[1]   PREVNUM←10
[2]   U←X
[3]   →(0=×/ρKBDBUF)/C
[4]   T←(PCOUNT,4)ρKBDBUF
[5]   OFFSET ← 47
[6]   STUFF
[7]   KBDBUF ← 10
[8]   C: →(~DATAFLAG)/0
[9]   INPUT←TEMP
[10]  TEMP←' '
[11]  DATAFLAG←CLEAR
      ▽

      .INV[ ]
      .INV;TEMP;IRHOX;AXIS;NR
[1]   →XEMPTY/0
[2]   TEMP←,X[;3]
[3]   IRHOX←1RHOX
[4]   NR←(TEMP≠0)/IRHOX
[5]   AXIS+1↑TEMP[NR]
[6]   TEMP[NR]←AXIS-TEMP[NR]-AXIS
[7]   →(√/(TEMP[NR]>96),TEMP[NR]<1)/ERR
[8]   X[;3]←TEMP
[9]   →0
[10]  ERR: ' *** NOTES OUT OF RANGE '
[11]  ERRFLAG←SET
      ▽

      .INITIALIZE[ ]▽
      ▽INITIALIZE
[1]   V0←V1←V2←V3←V4←V5←V6←V7←V8←V9←10
[2]   FUNCTION←TIMER←PREVFUNC←PCOUNT←0
[3]   RECORD←KBDBUF←PREVNUM←NUMBER←X←U←10
[4]   PROGBUF←INPUT←NBUF←' '
[5]   LIGHTFLAG←ERRFLAG←DATAFLAG←NUMFLAG←KBDFLAG←MATCHFLAG←
MODEFLAG←CLEAR
[6]   BADKEY←CLEAR
[7]   OFFSET←47
[8]   CURSOR←1
[9]   TEMPO←60
[10]  □PW←80
[11]  □RL←+/□TS
      ▽

```



```

      .JPX[ ]▽
      ▽JPX
[1]   INPUT←' '
      ▽

      .LIB[ ]▽
      ▽LIB;INCIPIT;COMP;NUM;CATALOG;LEN
[1]   →(0≠×/ρNUMBER)/N
[2]   →XEMPTY/ERR1
[3]   'MUSLIB'FTIE 4
[4]   LEN←8\RH0X
[5]   CATALOG←FREAD 4,0
[6]   CATALOG←(((ρCATALOG)÷7),7)ρCATALOG
[7]   CATALOG←(0,-8-LEN)+CATALOG
[8]   INCIPIT←(LEN-1)↑,(1+X[;3])-(¯1)+X[;3]
[9]   COMP←CATALOG+.=INCIPIT
[10]  NUM←1↑(COMP= /COMP)/1↑ρCATALOG
[11]  ' '
[12]  ' PIECE NO. ',▽NUM
[13]  ' '
[14]  U←X
[15]  X←FREAD 4,NUM
[16]  X←(((ρX)÷4),4)ρX
[17]  FERASE 4
[18]  CURSOR←1
[19]  →0
[20]  N:'MUSLIB'FTIE 4
[21]  →(NUMBER≥FFREE 4)/ERR2
[22]  U←X
[23]  X←FREAD 4,NUMBER
[24]  X←(((ρX)÷4),4)ρX
[25]  FERASE 4
[26]  CURSOR←1
[27]  →0
[28]  ERR1: ' ≠≠ NO NOTES OR CATALOG NO. GIVEN'
[29]  ERRFLAG←SET
[30]  →0
[31]  ERR2: ' ≠≠ PIECE NO. ',(▽NUMBER),' NOT IN
CATALOG'
[32]  ERRFLAG←SET
[33]  FERASE 4
      ▽

```

▽LOGOFF[□]▽

▽LOGOFF

```
[1] 'SYSLOG' FTIE 2
[2] RECNO←FREAD 2,0
[3] RECORD FWRITE 2,RECNO
[4] RECNO←RECNO+1
[5] □TS FWRITE 2,RECNO
[6] RECNO←RECNO+1
[7] RECNO FWRITE 2,0
[8] FUNTIE 2
[9] ''
[10] ''
[11] ' BYE.....'
[12] ''
[13] ''
```

▽

▽LOGON[□]▽

▽LOGON

```
[1] ''
[2] ' WELCOME TO THE MUSIC SYSTEM'
[3] ''
[4] 'SYSLOG'FTIE 2
[5] RECNO←FREAD 2,0
[6] □TS FWRITE 2,RECNO
[7] RECNO←RECNO+1
[8] (ASK' NAME: ')FWRITE 2,RECNO
[9] RECNO←RECNO+1
[10] RECNO FWRITE 2,0
[11] FUNTIE 2
[12] ''
[13] ''
```

▽

▽LOW[□]▽

▽LOW;A

```
[1] PREVNUM←1↑NUMBER,1
[2] U←10
[3] →XEMPTY/0
[4] GETARG
[5] A TRANSPOSE-PREVNUM
```

▽

```

∇MACHINE[□]∇
∇MACHINE;CHAR
[1]   INITIALIZE
[2]   LOOP:OFF
[3]   INPUT←ASK')'
[4]   MORE:CHAR←1↑INPUT
[5]   INPUT←1↑INPUT
[6]   →(CHAR='\'')/0
[7]   →((CHAR='→')∨~MODEFLAG)/EXEC
[8]   PROGBUF←PROGBUF,CHAR
[9]   →CONTN
[10]  EXEC:KEY←KEYS\CHAR
[11]  DECODE
[12]  →BADKEY/ERROR
[13]  →(NUMFLAG∨KBDFLAG)/CONTN
[14]  SETUP
[15]  ⑆,FMATRIX[FUNCTION;]
[16]  →(∨/KEY=CONTROL)/WRAP
[17]  PREVFUNC←FUNCTION
[18]  WRAP:NUMBER←10
[19]  RECORD←RECORD,KEY×SIGN[ERRFLAG+1]
[20]  ERRFLAG←CLEAR
[21]  →CONTN
[22]  ERROR:' ≠≠ NON-EXISTENT KEY CODE'
[23]  BADKEY←CLEAR
[24]  CONTN:→(0≠×/ρINPUT)/MORE
[25]  →LOOP
∇

```

```

▽MAT[ ]▽
▽MAT;TX;T;TT
[1]  ERRORCHECK
[2]  →ERRFLAG/0
[3]  PREVNUM←NUMBER
[4]  →XEMPTY/0
[5]  ⚡'T+V',▽NUMBER
[6]  →(RHOX=1↑ρT)/C
[7]  MATCHFLAG←CLEAR
[8]  ''
[9]  ''
[10] ' NO MATCH'
[11] ''
[12] ''
[13] →0
[14] C:TX←X[;1 3 4]
[15] TT←T[;1 3 4]
[16] TX[;1]←TX[;1]-TX[1;1]
[17] TT[;1]←TT[;1]-TT[1;1]
[18] MATCHFLAG←^(,TX)=,TT
[19] →MATCHFLAG/M
[20] ''
[21] ''
[22] ' NO MATCH'
[23] ''
[24] ''
[25] →0
[26] M:''
[27] ''
[28] ' MATCH''
[29] ''
[30] ''
▽

```

```

VPLA[ ]V
VPLA;LEN;TX;RHONR;TRHONR;NR;RHOTX;DISPLAY;TEMP;
NREC;BUF;NOW;EVCOUNT;NCOUNT;STIME;CH;FUNC;SELECT;EVENT;
RCOUNT
[1]   LEN←RHOX-CURSOR-1
[2]   →(LEN=0)/0
[3]   TX←((-LEN),4)†X
[4]   TX[;1 4]←TX[;1 4]×60÷TEMPO
[5]   NR←(0≠TX[;3])/1LEN
[6]   →(~LIGHTFLAG)/NEXT
[7]   RHOTX←1†ρTX
[8]   DISPLAY←(RHOTX,4)ρ(4×RHOTX)ρ'R   '
[9]   DISPLAY[NR;]←NOTES[(1+12|TX[NR;3]);]
[10]  NEXT:RHONR←ρNR
[11]  TRHONR←2×RHONR
[12]  TEMP←(TRHONR,3)ρ0
[13]  TEMP[;1]←TX[NR;1],TX[NR;1]+TX[NR;4]-1
[14]  TEMP[;2]←TX[NR;2],TX[NR;2]
[15]  TEMP[;3]←TX[NR;3],RHONRρ-1
[16]  TEMP←TEMP[ΔTEMP[;1];]
[17]  TX←10
[18]  NREC←0
[19]  BUF←' '
[20]  'TOOTS'FCREATE 1
[21]  □TRAP 52
[22]  NOW←[TEMP[1;1]+0.5
[23]  EVCOUNT←0
[24]  NCOUNT←0
[25]  XFORM:NCOUNT←NCOUNT+1
[26]  →(NCOUNT>TRHONR)/WRAP
[27]  STIME←[TEMP[NCOUNT;1]+0.5
[28]  CH←CHANNEL[TEMP[NCOUNT;2]]
[29]  FUNC←TEMP[NCOUNT;3]
[30]  SELECT←(FUNC<0),2ρFUNC≥0
[31]  EVENT←CH,SELECT/'0',(▽|FUNC÷12),ASCII[1+12||
FUNC]
[32]  →(NOW≥STIME)/MORE
[33]  BUF←BUF,(0[STIME-NOW+EVCOUNT)ρFILL
[34]  EVCOUNT←0
[35]  NOW←STIME
[36]  MORE:BUF←BUF,EVENT
[37]  EVCOUNT←EVCOUNT+ρEVENT
[38]  CHECK:→(915>ρBUF)/XFORM
[39]  NREC←NREC+1
[40]  (915†BUF)FWRITE 1,NREC
[41]  BUF←915†BUF
[42]  →CHECK

```



```
[43] WRAP:NREC←NREC+1
[44]   (BUF,OFF)FWRITE 1,NREC
[45]   □PW←131071
[46]   ON
[47]   RCOUNT←0
[48] LOOP:RCOUNT←RCOUNT+1
[49]   →(RCOUNT>NREC)/EXIT
[50]   FREAD 1,RCOUNT
[51]   →LOOP
[52] EXIT:□TRAP10
[53]   □PW←80
[54]   FERASE 1
[55]   →(~LIGHTFLAG)/0
[56]   ''
[57]   ,DISPLAY
[58]   ''
```

▽

▽PRG[□]▽

▽PRG;N

```
[1]   →(0=×/ρNUMBER)/ERR1
[2]   'PROGLIB'FTIE 3
[3]   N←FREAD 3,0
[4]   →(NUMBER≥N)/ERR2
[5]   PROGBUF←FREAD 3,NUMBER
[6]   FERASE 3
[7]   →0
[8]   ERR1: '   ≠≠ NO PROGRAM NUMBER GIVEN'
[9]   ERRFLAG←SET
[10]  →0
[11]  ERR2: '   ≠≠ PROGRAM NO. ',(▽NUMBER),' NOT IN CATALOG'
[12]  ERRFLAG←SET
[13]  FERASE 3
```

▽

.RAI[□]▽

▽RAI;A

```
[1]   PREVNUM←1↑NUMBER,1
[2]   U←10
[3]   →XEMPTY/0
[4]   GETARG
[5]   A TRANSPOSE PREVNUM
```

▽

▽RCL\_□]▽

▽RCL;T

```
[1]   ERRORCHECK
[2]   →ERRFLAG/0
[3]   PREVNUM←NUMBER
[4]   U←X
[5]   ⚡'T←V',▽NUMBER
[6]   →(0=×/ρT)/0
[7]   STUFF
```

▽

▽RCM[□]▽

▽RCM

```
[1]   ⚡'V',(▽PREVNUM),'←U'
```

▽

```

∇RCX[ ]∇
∇RCX
[1]   X←U
[2]   U←10
∇

∇REV[ ]∇
∇REV;START;LAST
[1]   PREVNUM←10
[2]   U←10
[3]   →XEMPTY/0
[4]   START←X[1;1]
[5]   LAST←X[RHOX;1]+X[RHOX;4]
[6]   X[;1]←START+LAST-X[;1]+X[;4]
[7]   X←X[ΔX[;1];]
∇

∇RST[ ]∇
∇RST
[1]   CURSOR←1
∇

∇SEC[ ]∇
∇SEC
[1]   PREVNUM←1↑NUMBER,0
[2]   U←10
[3]   COUNTER←PREVNUM
∇

∇SETUP[ ]∇
.SETUP;RX
[1]   RX←ρX
[2]   RHOX←1↑RX
[3]   XEMPTY←0=×/RX
∇

```

```

,SHU[ ]▽
▽SHU;TRHOX;ORDER;HINGES;ALL
[1]  PREVNUM←1↑NUMBER,1
[2]  U←X
[3]  →XEMPTY/0
[4]  →((0≠PREVNUM|RHOX) ∨ PREVNUM=0)/ERR
[5]  TRHOX←RHOX÷PREVNUM
[6]  X[;1]←X[;1]-X[1;1]
[7]  HINGES←1+PREVNUM×(-1+1TRHOX)
[8]  ALL←,Q(PREVNUM,TRHOX)ρRHOXρHINGES
[9]  X←(TRHOX,4×PREVNUM)ρ,X
[10] X←X[(TRHOX?TRHOX);]
[11] X←(RHOX,4)ρ,X
[12] X[HINGES;1]←0,+ \-1+[/(TRHOX,PREVNUM)ρ,X[;4]
[13] X[;1] ← X[ALL;1]
[14] →0
[15] ERR: ' ≠≠ CANNOT GROUP BY ',▽PREVNUM
[16] ERRFLAG←SET
▽

```

```

▽SSPLAY[ ]▽
▽SSPLAY;CH;FUNC;OCT;PC;I;NOTE;DISPLAY
[1]  FUNC←X[CURSOR;3]
[2]  →(FUNC=0)/REST
[3]  I←1+12|FUNC
[4]  CH←CHANNEL[X[CURSOR;2]]
[5]  OCT←▽[FUNC÷12]
[6]  PC←ASCII[I]
[7]  NOTE←,NOTES[I;]
[8]  DISPLAY←' ',(▽CURSOR),'. ',NOTE,OCT,'/',▽X[
CURSOR;4]
[9]  DISPLAY,((24-ρDISPLAY)ρ' '),CH,OCT,PC,24ρFILL
[10] →0
[11] REST: ' ',(▽ CURSOR),'. R/',▽ X[CURSOR;4]
▽

```

```

▽STE[ ]▽
▽STE;NUM;COUNT
[1]   → XEMPTY/0
[2]   NUM ← 1 ↑ NUMBER,1
[3]   COUNT ← 0
[4]   ON
[5]   L: →(CURSOR > RHOX)/0
[6]   COUNT ← COUNT+1
[7]   →(COUNT > NUM)/0
[8]   SSPLAY
[9]   CURSOR ← CURSOR+1
[10]  → L
▽

```

```

▽STO[ ]▽
▽STO
[1]   ERRORCHECK
[2]   →ERRFLAG/0
[3]   PREVNUM←NUMBER
[4]   ⚡'U←V',▽NUMBER
[5]   ⚡'V',(▽NUMBER),'←X'
▽

```

```

▽STT[ ]▽
▽STT
[1]   U←PREVNUM←10
[2]   TIMER←(□TS)[5 6]
▽

```

```

▽STUFF[ ]▽
▽STUFF;CM1;RHOT;SEG1;SEG2;SEG3;FIRST;LAST
[1]   →(~XEMPTY)/S
[2]   X←T
[3]   →0
[4]   S:CM1←CURSOR-1
[5]   RHOT←1↑ρT
[6]   X←((CM1,4)↑X)↑T;(CM1,0)↑X
[7]   SEG1←CM1↑X[;1]
[8]   FIRST←1↑SEG1
[9]   LAST←(¯1↑CM1↑X[;4])↑¯1↑CM1↑X[;1]
[10]  SEG2←FIRST+LAST+RHOT↑CM1↑X[;1]
[11]  LAST←(¯1↑RHOT↑CM1↑X[;4])↑¯1↑RHOT↑CM1↑X[;1]
[12]  SEG3←FIRST+LAST+(RHOT+CM1)↑X[;1]
[13]  X[;1]←SEG1,SEG2,SEG3
▽

```

```

      ULT[ ]▽
▽A TMULT FACTOR;PREV
[1]   →(CURSOR≤RHOX)/N
[2]   X[A;1 4]←X[A;1 4]×FACTOR
[3]   →0
[4]   N:PREV←X[A;4]
[5]   X[A;4]←X[A;4]×FACTOR
[6]   X[;1]←X[;1]+(X[A;4]-PREV)×X[;1]>X[A;1]
      ▽

```

```

▽TRANSPOSE[ ]▽
▽A TRANSPOSE SEMITONES;TEMP;NR
[1]   TEMP←,X[A;3]
[2]   NR←(TEMP≠0)/,TEMP
[3]   TEMP[NR]←TEMP[NR]+SEMITONES
[4]   →(√/(TEMP[NR]>96),TEMP[NR]<1)/ERR
[5]   X[A;3]←TEMP
[6]   →0
[7]   ERR: ' ≠≠ NOTES OUT OF RANGE'
[8]   ERRFLAG←SET
      ▽

```

```

▽UDF[ ]▽
▽UDF
[1]   PROGBUF←' '
      ▽

```

```

▽UND[ ]▽
.UND
[1]   →(PREVFUNC=0)/0
[2]   NUMBER←PREVNUM
[3]   2,UMATRIX[PREVFUNC;]
[4]   PREVNUM←,0
      ▽

```

```

.USR[ ]▽
▽USR
[1]   PREVNUM←,0
[2]   U←,0
[3]   2ASK' \ \ '
      ▽

```



```

▽VER[ ]▽
▽VER;TRHOX;HINGES;ALL
[1]   PREVNUM←1↑NUMBER,1
[2]   U←X
[3]   →(XEMPTY▽PREVNUM=1)/0
[4]   →(0≠PREVNUM|RHOX)/ERR
[5]   TRHOX←RHOX÷PREVNUM
[6]   HINGES←1+PREVNUM×(-1)+1TRHOX
[7]   ALL←,Q(PREVNUM,TRHOX)ρRHOXρHINGES
[8]   X[HINGES;1]←0,+ \(-1)÷[(TRHOX,PREVNUM)ρ,X[;4]
[9]   X[;1]←X[ALL;1]
[10]  X[;2]←RHOXρ1PREVNUM
[11]  →0
[12]  ERR: ' ≠≠ CANNOT GROUP BY ',▽PREVNUM
[13]  ERRFLAG←SET
▽

```

```

▽XCT[ ]▽
▽XCT
[1]   PREVNUM←10
[2]   U←10
[3]   INPUT←PROGBUF
▽

```

*ASCII*  
 nΔL iε\_°∇'α□⊥|

*CHANNEL*  
 ◇ABC

*CLEAR*  
 0

*CONTROL*  
 41 42 43 44 45

*FILL*  
 Z

*KEYS*  
 nΔL iε\_°∇'α□⊥|T00123456789 -, +. / ( ; : - \* ? ρ [ ~ + u ω > † c † † ≥ - = P Q R S  
 T U V W X Y Z { - } D E F G

*NOTES*  
 n  
 n<  
 L  
 L<  
 ε  
 -  
 -<  
 ∇  
 ∇<  
 α  
 α<  
 ⊥

*OCTAVE*  
 35 47

*OFF*  
 ◇0A0B0C0

## FMATRIX

INV  
REV  
VER  
SHU  
AUG  
DIM  
RAI  
LOW  
COM  
DRO  
INS  
CLR  
LIB  
RCL  
MAT  
BKS  
STE  
PLA  
END  
RST  
PRG  
STO  
DEF  
XCT  
UND  
USR  
DAT  
BFM  
BBM  
SEC  
DEC  
BRF  
BRB  
STT  
DST  
DOF  
JPX  
BFZ  
BBZ  
DON  
BBN  
BFN  
BBP  
BFP



A P P E N D I X C  
PILOT STUDY QUESTIONNAIRE

PILOT STUDY FOR AN EXPERIMENTAL MUSIC SYSTEM

PART I. Please provide the information requested below. Leave blank any item that does not apply to you.

1.) Name: \_\_\_\_\_

2.) Age: \_\_\_\_\_

3.) Your major or concentration at ULowell OR your occupation:

\_\_\_\_\_

4.) Musical Instrument(s) you play: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

5.) Musical training:

a) private lessons (list instruments played and years studied):

b) high school (list any vocal or instrumental ensembles in which you participated and any music courses you took):

c) college (list any college-level music courses you have taken. Note: College of Music students need only give the name of their major program):

6.) Computer Background. Describe any experience you have in the use of digital computers (e.g., programming, business data processing, engineering or statistical computation, computer operation, etc.):



PART II. Please respond to the following questions:

- 1.) Is the music system easy or difficult to use. Consider, for example,

keyboard layout  
the functions provided  
the information shown in the CRT display  
the music "synthesizer"  
.....etc.

- 2.) What additions and/or changes would improve the music system?

(PART II., continued)

3.) What could the music system be used for? How would you use it?

4.) If the hand-calculator-sized music system described to you earlier were available, would you like to own one? If yes, how much would you be willing to pay for it?

A P P E N D I X D

SUGGESTED APPLICATIONS  
OF THE MUSIC SYSTEM

- 1) electronic pocket metronome. The system can be made to beep or click at precisely controlled rates.
- 2) pocket tuner. The system provides a very accurate source of equal-tempered pitches.
- 3) "composer's notebook". If the system is provided with a non-volatile memory, it can be used as a means for temporarily storing musical ideas until they can be written down or recorded in some more permanent form.
- 4) music student's "assistant". The system can be programmed to play the solutions to typical harmony and counterpoint exercises. This would be particularly helpful for students who have not developed enough keyboard skill to play their homework problems on the piano.
- 5) "accompanist". The system can be programmed to play accompaniments for vocalists and instrumentalists.
- 6) audible pocket reference manual. The library could contain entries for all common scales, chords, cadence formulas, ornaments, etc.
- 7) rhythmic problem solver. The system can be used to play rhythms that the performer cannot figure out. Typical cases would be unusual divisions of the beat, counter-rhythms, "metric modulation", etc.

- 8) pitch problem solver. The system can be used to play pitch patterns that the performer cannot figure out.
- 9) dictation exerciser. Standard music dictation exercises could be implemented as stored programs.
- 10) electronic music sequencer. If the output of the system is made electrically compatible with standard electronic music equipment, the system can be used as a powerful "sequencer". It can store and play back long sequences of notes with accurately specified pitches and durations, and it can generate new sequences of notes through the various musical transformation functions it provides.
- 11) portable library of music. The library can be filled with the pieces most commonly used for study and analysis.
- 12) automatic "composer". The SHUFFLE function and the other musical transformation functions permit the user to explore both automatic variation of existing musical pieces and automatic generation of new compositions.
- 13) musical "subject" analyzer. The ability to combine and transform variants of a given sequence of notes facilitates quick analysis of the possibilities of a fugue subject or twelve-tone row.
- 14) logic game. In addition to its more or less obvious uses as a musical game, the system could also provide games in logical thinking. For example: given a sequence of notes with some specific characteristic (e.g., all the notes are separated by whole steps), transform the sequence into one that has some

completely different characteristic (e.g., all the notes are separated by major thirds), and do this without resort to certain operations (e.g., entering notes at the keyboard or bringing them in from the library).

- 15) automated thematic index. The "hum a few bars" library indexing scheme can be used to discover the identity of a composition which can then be obtained from an ordinary library of musical scores and/or recordings. This would be much more convenient than present printed thematic indexes, which require the user to translate the theme into letters and/or numbers according to a formula before entering the index.

A P P E N D I X E  
LIBRARY AND PROGRAM CATALOGS



LIBRARY CATALOG

<u>Library Number</u>	<u>Name of Composition</u>
1.	C-major scale
2.	c natural minor scale
3.	c harmonic minor scale
4.	c melodic minor scale
5.	"Are You Sleeping"
6.	Beethoven, "Ode to Joy" theme from Ninth Symphony
7.	"Au Clair de Lune"
8.	"Yankee Doodle"
9.	"Three Blind Mice"
10.	"On Top of Old Smoky"
11.	Haydn, theme from 2nd movement of Symphony No. 94
12.	Tune-with-mistake(s)
13.	Purcell, Prelude in G
14.	Monk, "I Mean You"

PROGRAM CATALOG

<u>Program Number</u>	<u>Program Name/Description</u>
1.	Easy pitch-matching game. Repeat a 2-note sequence played by the system. The first note of the sequence is always C. The system displays the letter names of the notes on the CRT <u>before</u> asking you to guess the notes.
2.	3-note pitch-matching game, no display.
3.	4-note pitch-matching game, no display.
4.	5-note pitch-matching game, no display.
5.	6-note pitch-matching game, no display.
6.	7-note pitch-matching game, no display.
7.	8-note pitch-matching game, no display.
8.	"Composer" (generates a 4-voice composition consisting of random notes).
9.	Beginner's pitch-matching game. Repeat a three-note sequence which always consists of some combination of C's, D, and E. The first note is always C.
10.	Another "composer" program (generates a 4-voice composition consisting of notes drawn from the pentatonic scale).

# A P P E N D I X F

## PROGRAMMING EXAMPLES

### Program to Create 2-Part Rounds

```

nLIB      ; Get piece no. n from the library.
STO       ; Store it in user location 0.
DAT       ; Unlock the keyboard (i.e., go into data entry mode).
/rests/   ; Key in the proper number of rests to delay the second part,
INS       ;   and insert them into the working area.
COM       ; Combine 1st part (in loc. 0) with 2nd part (in working area).
PLA       ; Play the completed round.
    
```

### Two- to Eight-Note Pitch-Matching Game

```

1 LIB     ; Get C-major scale from the library.
REV       ; Reverse it and
DRO       ;   drop off C above middle-C.
SHU       ; Shuffle the remaining notes.
nSEC      ; Set loop counter to no. of notes to drop from shuffled scale.
DRO       ; Drop one note,
DEC       ;   decrement the counter, and
3 BBP     ;   if counter is still positive, go back for more.
NOT       ; Otherwise, set the data flag,
'C'       ;   copy middle C into the keyboard buffer,
INS       ;   and then insert it into the working area.
DOF       ; Make sure the display is off.
PLA       ; Play the sequence of notes.
STO       ; Save the sequence in location 0.
STT       ; Start the timer (i.e., begin measuring response time).
3 SEC     ; Set the loop counter for 3 guesses.
CLR       ; Clear the working area.
DAT       ; Get answer from user, and
MAT       ;   match against the notes in location 0.
-7 BFM    ; Skip ahead if note sequences match.
DEC       ; Otherwise, decrement the loop counter, and
-7 BBP    ;   if the counter is still > 0, go back for another try.
RCL       ; Otherwise, recall the original note sequence,
DON       ;   turn on the display, and
PLA       ;   play the sequence.
DON       ; Make sure the display is on, then
DST       ;   show elapsed time (user's response time).
    
```

Four-Voice Random Pentatonic "Composer"

```

CLR      ; Clear the working area.
NOT      ; Set the data flag, and
'CDEGA' ;   copy the notes of the pentatonic scale (and
/rests/  ;   some rests, if desired) into the keyboard buffer
INS      ;   and insert them into the working area.
STO      ; Store the scale in location 0.
12 RAI   ; Transpose original scale up one octave, and
RCL      ;   join it to the un-transposed version.
SHU      ; Shuffle the notes (and rests), and
STO      ;   store in location 0 as voice part no. 1.
SHU      ; Shuffle everything again, and
1 STO    ;   store in location 1 as voice part no. 2.
SHU      ; Shuffle everything again, and
2 STO    ;   store in location 2 as voice part no. 3.
SHU      ; Shuffle everything again to obtain last voice part.
COM      ; Merge in voice part no. 1, and
1 COM    ;   no. 2, and
2 COM    ;   no. 3.
PLA      ; Play the entire "composition".

```

Program to Generate and Play All 48 Forms  
of a 12-Tone Row

```

CLR      ; Clear the working area.
DAT      ; Go into data entry mode and
/row/    ;   get prime set from user.
12 SEC   ; Set the loop counter for 12 iterations.
STO      ; Store a copy of the prime set, and also
PLA      ;   play it.
INV      ; Invert the prime set,
1 STO    ;   store a copy of the inversion, and also
PLA      ;   play it.
CLR      ; Clear the working area.
RCL      ; Recall the prime set,
REV      ;   reverse (retrograde) it, and then
PLA      ;   play the retrograde.
CLR      ; Clear the working area again.
1 RCL    ; Recall the inversion,
REV      ;   reverse (retrograde) it, and then
PLA      ;   play the retrograde inversion.

```

(continued)

```
DEC      ; Decrement the loop counter, and
2 BFP    ;   if it is still > 0, skip the next step.
JPX      ; Otherwise, jump out of the program (program complete).
CLR      ; Clear the working area.
RCL      ; Recall the prime set, and
END      ;
RAI      ;   transpose it up a semitone.
RST      ;
26 BRB   ; Go back and repeat the procedure.
```





