To the Graduate Council:

I am submitting herewith a thesis written by Frank S. King, III entitled "Music Command Interpreter: A Tool for Writing Computer Assisted Music Instruction Programs." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

David L. Matuszek

David L. Matuszek, Major Professor

We have read this thesis
and recommend its acceptance:

Donald Pederson

J. Michael Moshell

Accepted for the Council:

L. Evans Peck

Vice Chancellor
Graduate Studies and Research

MUSIC COMMAND INTERPRETER: A TOOL FOR WRITING COMPUTER ASSISTED MUSIC INSTRUCTION PROGRAMS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Frank S. King, III

June 1983

1180013

DEDICATION

In memory of my mother, Mary Lewis Byrd King, who unselfishly gave her time to teach me the necessary qualities needed to realize all of my present and future ambitions.

ACKNOWLEDGEMENTS

I would like to express appreciation to my father for his continued patience and support during my extended scholastic career.

ABSTRACT

The goal of the work presented in this thesis was to develop a programming tool which gives musical Input/Output capabilities to programs written in Applesoft BASIC. Computer Assisted Music Instruction (CAMI) would be an important application for programs having these capabilities.

The tool which was developed is called the Music Command Interpreter (MCI). MCI is designed to run on an Apple microcomputer having additional musical hardware peripherals. The required peripherals are the Mountain Computer Music System cards, the Alpha Syntauri music keyboard, and a keyboard interface card. MCI provides a programmer-friendly interface to Applesoft BASIC programs.

TABLE OF CONTENTS

CHAPTER PAGE

INTRODUCTION. 1

1. BACKGROUND TERMINOLOGY AND WORK IN DIGITAL MUSIC SYSTEMS DESIGNED FOR THE APPLE MICROCOMPUTER . . 3

Digital Music System Terminology 3
James Ross' Music Transcription System 7
The ALPHA III Music System 15
Monday Mercer's Program for Comparison of Note Files Generated by the ALPHA III Music System. 23

2. GUIDE TO USING THE MUSIC COMMAND INTERPRETER . . 29

How to Use the Music Command Language. 29
Setting Up an Applesoft Program Which Uses MCI 39

3. DESIGN OF THE MUSIC COMMAND INTERPRETER. 43

Overall System Design. 43
Design of the Musical I/O Routines 48

4. RESULTS. 53

BIBLIOGRAPHY. 58

APPENDICES. 60

APPENDIX A. MODIFYING ENVELOPE PRESET FILES. 61
APPENDIX B. MODIFYING CONVERSION TABLE FILES 64

VITA. 68

LIST OF FIGURES

FIGURE		PAGE
1.	TYPICAL WAVEFORM ENVELOPES	6
2.	ALGORITHM FOR THE ALPHA SYSTEM	18
3.	ALPHA SYSTEM'S METHOD OF DEFINING ENVELOPE SHAPES.	22
4.	EVENT DURATION DATA RECORDED BY THE ALPHA SYSTEM .	25
5.	OUTPUT FROM COMPARISON ROUTINES USING M6 AND NA6 .	26
6.	BASIC PROGRAM USING MCI.	30
7.	BASIC PROGRAM LOADING MCI AND ASSOCIATED TABLES. .	40
8.	ALGORITHM FOR THE INTERRUPT SERVICE ROUTINE. . . .	44
9.	ALGORITHM FOR THE MUSIC COMMAND INTERPRETER. . . .	46
10.	ALGORITHM FOR TRANSCRIPTION OF KEYS RECORDED IN SEQUENCE	50
11.	ALGORITHM FOR RECORDING THE KEYS OF A CHORD. . . .	52
12.	EXAMPLE EXECUTION OF MCI.PRESET.EDIT	62
13.	EXAMPLE EXECUTION OF MCI.C.TAB.EDIT.	65

INTRODUCTION

The goal of the work presented in this thesis was to develop a programming tool which gives musical Input/Output capabilities to programs written in Applesoft BASIC. Computer Assisted Music Instruction (CAMI) would be an important application for programs having these capabilities. Prior to this work, no programming tools existed for creating quality CAMI programs on the Apple Microcomputer. Such a program would use the music synthesis hardware available for the Apple such as the Mountain Computer Music System cards in combination with the Alpha Syntauri music keyboard and keyboard interface card. Quality CAMI programs are more easily produced if programming tools exist which relieve the CAMI programmer of the need to know specific memory locations in order to interface with the hardware or with the low level music generating software routines.

If the CAMI programs were written in a high level programming language such as PASCAL or FORTRAN, then the tools would most likely form a library of subroutines which could be called from one of these languages. However, in this work, the target language was Applesoft BASIC. Since this language had no provisions for parameter passing through callable subroutines, an approach other than the tool library concept had to be taken. Thus, an extension to the Applesoft BASIC language

was developed. This extension is called a "music command" and is interpreted by software known as the Music Command Interpreter (MCI). A music command is formed in a BASIC program by setting the string variable "MC$" equal to a string which is a member of the music command language. Control is then passed from the BASIC CAMI program to the MCI by making a call to the address where MCI has been loaded. MCI services the music command's request and returns to the BASIC program.

Chapter 1 introduces the terms and concepts used in digital music synthesis and reviews some of the background work involving the use of the Apple microcomputer in music synthesis. Then the music command language and how it is used is described in detail in Chapter 2. Chapter 3 discusses the design of the MCI and explains how the music commands were implemented in software. Chapter 4 discusses the results of this work.

CHAPTER 1

BACKGROUND TERMINOLOGY AND WORK IN DIGITAL MUSIC SYSTEMS DESIGNED FOR THE APPLE MICROCOMPUTER

1. DIGITAL MUSIC SYSTEM TERMINOLOGY

In order to understand and appreciate what digital music synthesis is, one must first understand the components of music itself. Music can be defined as the science or art of ordering tones in succession, in combination, and in time to produce a composition having unity and continuity. The length of time occupied by a tone is called its duration. A musical tone may be defined as consisting of three attributes: pitch, loudness, and timbre. If using the subjective definitions of these attributes, then one would say that pitch is the elevation of the tone, loudness is the intensity, and timbre is the quality or color. To design a digital music system however, one must develop definitions which lend themselves to quantitative measurement. Thus, pitch is defined in terms of wave frequency, which is the number of repetitions of a periodic wave form in a unit of time. Loudness is the maximum amplitude or height of the wave as it travels through its period. Timbre is a combination of the spectrum of waves which compose the waveform and of the envelope which defines the loudness of the waveform at a given point in time.

The concept of breaking the waveform heard in a particular tone into a spectrum of waves is much the same as the concept of breaking the visible light seen into a spectrum of component colors. The main component of a waveform is the fundamental wave whose period is equal in length to that of the tone being heard. The fundamental wave may be mixed with one or more other waves whose frequencies are some integer number higher than that of the fundamental wave. These higher frequency waves are called harmonics. The proportional amount each harmonic wave contributes to the waveform can be determined by analyzing the spectrum of the waveform. A major reason for one instrument sounding differently from another is that different instruments produce different waveforms and thus different spectra.

The other attribute of timbre is the envelope of the waveform produced by an instrument. This is another major reason for various instruments sounding differently. The envelope defines the loudness of the waveform at a given point in time during the time period in which the tone is heard. When a tone is heard, it takes some small amount of time for the tone to reach its maximum loudness. This small period of time is called the attack period of the envelope. Depending on the instrument being played, the initial loudness of the tone will decay to a level which the instrument will sustain until the player is through

playing the tone. The time during which the loudness decays from the maximum level to the sustain level is called the decay period of the envelope. The sustain period of the envelope lasts until the tone is allowed to rest. The time during which the loudness falls from the sustain to the resting level is called the release period of the envelope. The lengths of these four envelope periods vary from one kind of instrument to another. Three examples are displayed in Figure 1. In some instruments the envelopes may not even have decay or sustain periods. Also, some instruments cannot change their loudness as quickly as others; hence, the shape of the envelope created by one instrument may be drastically different from the shape created by another instrument.

The possible ranges of pitch and loudness are also important features that help to distinguish one instrument from another. When trying to duplicate the sound of a particular instrument using a digital synthesizer, one must keep in mind these ranges. It can be shown that if loudness and timbre are held constant while pitch is varied moderately, one perceives the loudness and timbre to vary slightly also. Similarly, if pitch and timbre are held constant while loudness is varied moderately, one perceives the pitch and timbre to vary slightly. However, if either pitch or loudness are greatly varied, the two tonal attributes being held constant are perceived to vary

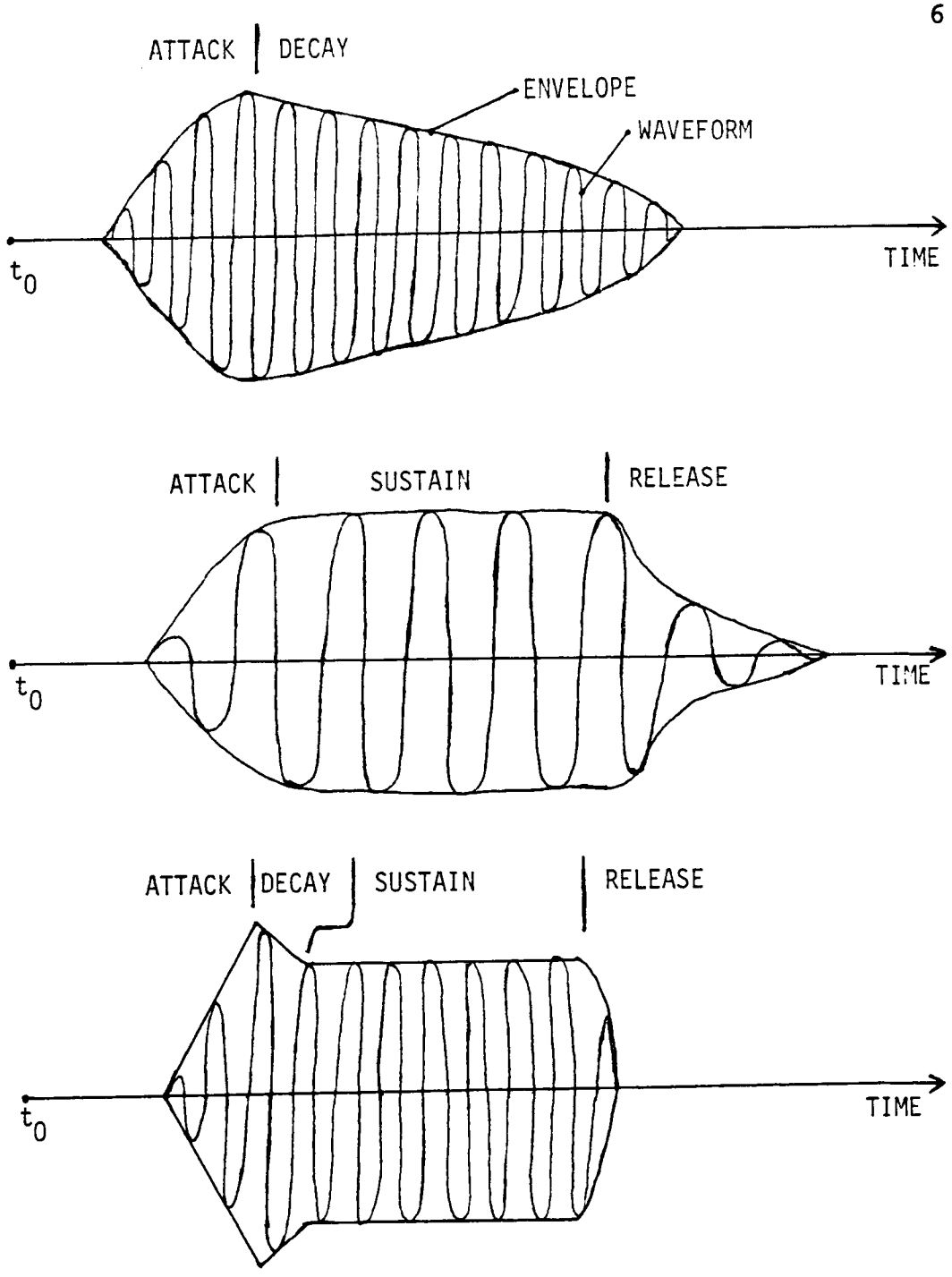


FIGURE 1

TYPICAL WAVEFORM ENVELOPES

significantly. Thus, when trying to replicate an instrument digitally, one cannot expect to produce tones of the same quality as the instrument when outside of the natural ranges of that instrument.

From the above definitions, it appears that a digital music system is a combination of software and hardware which is capable of simulating the three attributes of musical tone. A minimal digital music system must be able to produce at least one waveform and must have the ability to vary the pitch, loudness, and duration of that waveform. A good system could also modify the attributes of timbre and thus would be capable of replicating to some degree the tones produced by a particular instrument. The closeness of this replication depends on how much control the system has over the waveform's spectrum and its envelope as well as how much information on the instrument's timbre is available. The first system to be reviewed is the one developed for the Apple microcomputer by James Ross.

2. JAMES ROSS' MUSIC TRANSCRIPTION SYSTEM

The purpose of this system was to take real time input from a musical keyboard which was interfaced to an Apple microcomputer, record this input, and, at the same time, transmit the appropriate musical tones as real time output to an amplifier which was also interfaced to the

Apple. The system could, at some later point in time, use the recorded input to play back what had been played. This recorded input could also be used by the system to transcribe what had been played into musical notation which was displayed graphically on the Apple's screen.

Development of this system was completed in December, 1980. The system was designed with the best peripheral equipment available for the Apple at that time. Ross used a PAIA keyboard with thirty-six keys. This keyboard was interfaced to the Apple using an interface card developed by Doug Barnhardt at the University of Tennessee as an independent project in electrical engineering earlier that year. The Apple was interfaced to the amplifier using a commercially manufactured peripheral card called the ALF Music Synthesizer card.

When designing the software for an interactive real time digital music synthesizer, the designer is greatly restricted in the design by the hardware which is to be used. If the hardware is capable of controlling all of the attributes which compose a musical tone and is capable of keeping time, then the job of the software designer becomes more involved with the understanding of how to interface software with these hardware controls and less concerned with how to simulate the attributes of music in software. As the number of controls which could be set by the software is increased, the potential for developing a

system which can replicate the musical qualities of various real instruments increases accordingly. Also, such hardware would greatly reduce the real time programming effort required. Unfortunately, such sophisticated peripherals still do not exist for use with the Apple microcomputer.

Since timing is the central consideration in the design of any real time system, the timing method Ross used should be looked at first. The two most common timing methods used on microcomputers involve the use of hardware clocks which generate interrupts periodically and the use of software loops which take a known, constant amount of time to execute. Of course, the use of a hardware clock would be the preferred method. Since such a clock was not available to him, and since constant time software loops which also perform meaningful tasks are next to impossible to program and to debug, Ross used a less frequently seen timing method which, unfortunately, is fairly frequently used in many systems designed for the Apple. Most Apples are equipped with game paddles. A game paddle is simply a variable resistor which is interfaced to the Apple. Storing a value in a specific memory location associated with a paddle causes a capacitor to become charged. At some later point in time, the capacitor will become fully discharged. The time taken for the capacitor to discharge depends on the

resistance setting of the paddle. When the capacitor has become fully discharged, a specific bit located in another specific memory location associated with the paddle is set internally by the Apple's hardware. Since the status of this bit can be monitored by software, the system has a method of keeping time such that the actual length of a unit of time is a function of the paddle setting.

Ross' system was divided into three stages. Each stage began with the game paddle being fully charged. The time necessary for each stage to execute was less than the time required for a game paddle to become fully discharged, assuming that the paddle had been set within the limits of some predefined range of resistance. After completing its designated tasks, each stage waited for the bit to be set which indicated that the paddle was completely discharged. At this point, the paddle was recharged and control passed to the next stage. All three stages comprised one big interrupt service routine which masked out other interrupts during its entire execution. The timing unit used to record the duration of a note was the length of time required to execute the entire interrupt service routine, or in other words, the time required for a paddle to discharge three times. According to Ross, by the time the service routine was ready to return, another interrupt was certain to be pending.

The PAIA keyboard interface card was the source of the interrupts to be serviced by the system. The PAIA continuously scanned the keys and sent a value to the interface card for each keypress it found and also each time it completed a scan of the keyboard. Each of the keys and the end of scan condition had a specific value associated with them. The interface generated an interrupt each time a value was received from the PAIA. There was no indication from the PAIA as to when a key had been released. Therefore, the software was required to maintain a list of keys currently pressed. The keys considered to be pressed were kept in order by their associated values. By continuously reading a value from the PAIA interface card upon each interrupt and comparing this with the keys stored in the list, the system could detect when a key was no longer being pressed and thus removed it from the list.

The process of analyzing the data obtained from the PAIA keyboard occurred during the first stage of the interrupt service routine. The other functions of this stage were to count the number of time units occurring between changes in the state of the keyboard and to record these changes as coded events along with the duration of time which had elapsed since the last event. To hear what had been recorded, a routine called Playback was substituted for stage one of the service routine.

Playback simply used the values obtained from the event-duration recording as if they were values sent from the PAIA interface card. The second stage of the service routine was responsible for interpreting any key which might have been pressed on the Apple's keyboard and for making the appropriate response. The third stage was responsible for determining the status of the envelopes of the tones currently being played and for relaying this information to the ALF card so that the card produced the appropriate output signals to the amplification system.

The interface that the ALF card provided between the software and the amplifier was an important factor in determining the overall quality of musical tones that the system was capable of producing. Ross' system was designed so that several ALF card's could be attached to the Apple and used at once. Each ALF card had three output channels. Each output channel had two controls which could be set by software. The controls were for the frequency and the loudness output from the channel. Thus, the system could output a different frequency from each output channel available and have each of the channels set to different loudness levels. Of course, for a system containing only one ALF card, there is a limit of only three notes being sounded simultaneously, but this is enough to play any melody or series of triads.

Although the ALF card had the ability to control the tonal attributes of frequency and loudness, there was an unfortunate limitation in its ability to change the timbre of the tones that were played. By having the ability to change the loudness of a frequency being output over a period of time, the system was able to simulate the envelope aspect of timbre, but the ALF card did not provide a way in which to change the waveform of the frequency that was output. Each channel was fed a square wave at a frequency of 1.8MHz. The function of the frequency control of each channel was to reduce this initial frequency down to the level desired for output. Since the only waveform produced by the ALF card was the square wave, the system was unable to simulate the timbre of any of the classic musical instruments.

Finally, the ability of Ross' system to transcribe recorded event-duration data into musical notation will now be examined. The data was parsed from left to right. As each key-press event was encountered, the transcription routine would look ahead to find the corresponding key-release event. The sum of the durations after the key-press event through and including the key-release event was determined to be the duration that the key was pressed. If there were a very short period of rest following the key's release, less than or equal to the duration of a sixty-forth note, then this duration was

added back to the duration of the key under consideration. This was meant to allow for the time it would take a person playing the keyboard to lift a finger from one key and place it down on another key. With the duration finally calculated, the note was assigned the appropriate duration symbol(s) using the following algorithm:

```
ASSIGN Duration-Symbol of closest fit
       to Note-Duration leaving Remainder;
WHILE  Remainder > 0  do
  begin
    Note-Duration = Remainder;
    ASSIGN Tie-Symbol to link duration symbols;
    ASSIGN Duration-Symbol of closest fit
           to Note-Duration leaving Remainder;
  end;
```

In conclusion, Ross evaluated this system to be best suited as an input mechanism to a musical editor. He pointed out that due to the error introduced by his timing method and due to the rigidness of the transcription algorithm, there was little chance that a person recording music from the keyboard would end up with a transcription that looked exactly as that person intended it to look. However, Ross warned that when he tried adding heuristics to the transcription algorithm, the probability of generating transcription errors was increased. It is the opinion of the Author that Ross would have obtained better results by using an editable look-up table, which included the entire range of possible durations up to a whole note, instead of trying to make the symbolic assignments algorithmically. In fact, if transcription is a primary

goal of a digital music system, as it would be in a CAMI system, then the system should have some method of limiting the complexity of the event-duration data which is to be transcribed. However, doing this might also limit the variety of the types of music which could be played using the system as an instrument for playing enjoyment and recording. Therefore, the designer of a digital music system must keep in mind that a system built for the purpose of music enjoyment and exact recording would require a different design than a system built for the purpose of music transcription and comparison. Although a system may try to fill both purposes, the system will probably only excel at one of them. In Ross' system, the best met purpose was that of music enjoyment and recording ability. Finally, Ross suggested that a better designed digital music transcription system "... would involve a real-time hardware clock which generates interrupts at given intervals."

3. THE ALPHA III MUSIC SYSTEM

The Alpha III music system was developed by Syntauri Limited of Palo Alto, California. In a paper written by members of Syntauri Limited and presented at the sixty-seventh convention of the Audio Engineering Society, this system was defined as being "... a modular, general purpose microcomputer based instrument intended for music

composition and real time performance." The system did not pretend to have any transcription ability. Thus, the Alpha system had a well defined limited purpose which it accomplished very well. Although this system was not intended to be used as a tool for CAMI programming, knowledge of its design was helpful to the author in designing a CAMI programming tool.

The interfacing hardware and keyboard used by this system far surpassed the qualities and ablities of the hardware used in Ross' system. The Alpha system used a sixty-one key keyboard and a keyboard interface card which were both sold by Syntauri. The synthesizer cards which interfaced the Apple to the amplifier were sold by Mountain Computer. The advantages of each of these pieces of hardware will be discussed as the overall system design is described.

As was the case with the previous system studied, the hardware used was important in the overall design of the Alpha system software. Due to this system's limited design goal, the timing method used was not as important to the design. The Alpha system's designer, Charlie Kellner, had a choice of two timing methods. Since the Alpha system was not concerned with transcription, there was no need to keep time in terms of absolute time units. Instead, the duration associated with an event could represent the number of relative time units which passed

between that event and the next event. The length of a relative time unit depended on the time required to execute one pass through the main loop of the software. This time varied according to the state of the musical keyboard. The playback of an event-duration recording could simply indicate to the system what change was to be made in its image of the state of the keyboard and how many times it should cycle through its main software loop in this state before looking for another change in its keyboard image. Thus, one possible timing method was to use software-influenced, relative timing units. The other possible timing method involved the use of a hardware clock which was located on one of the Mountain Computer Music System Cards and which generated an interrupt once every eight milliseconds. The method chosen for the Alpha III system involved the concept of relative timing units. However, a later version of this system, called Alpha Plus, used the hardware timing method. Either way, both systems provided excellent apparent real time response.

Although the Alpha Plus system had a few enhancements over the Alpha III system and the timing methods were different, both designs were basically the same. Therefore, the remaining design description will refer to the two Alpha systems in general. The algorithm shown in Figure 2 represents the general structure of the system software. The details given in the next few paragraphs

```
ENTER-HERE from BASIC;
INITIALIZE Tables and Hardware-Settings;
REPEAT
  IF ( Record-Mode-Is-On ) then READ Music-Keyboard-Bit-Map
  ELSE  READ PreRecorded-Event-Duration-Data;
  IF ( Event-Information  .indicates. Only-Upper-Contact-Hit ) then
      "use duration from now until lower contact hit for Velocity
                                                      calculation"
  ELSE IF ( Event-Information  .indicates. Lower-Contact-Hit ) then
    Begin
      ASSIGN Corresponding-Tone  to  Available-Pair-of-Oscillators;
      SET Available-Pair-of-Oscillators to Appropriate-Frequency;
      CALCULATE Velocity and SAVE for Envelope-Calculations
    End
  ELSE IF ( Event-Information  .indicates. Key-Released ) then
    DEASSIGN Corresponding-Pair-of-Oscillators
  ELSE CONTINUE;          (* No change in keyboard state *)
  FOR All Assigned-Oscillator-Pairs
    Begin
      UPDATE  Primary-and-Percussion-Envelope-Status;
      SET-LOUDNESS of Primary-Oscillator;
      SET-LOUDNESS of Percussion-Oscillator
    End
UNTIL  Key-Pressed-on-Apple-Keyboard;
RETURN to BASIC;
```

FIGURE 2

ALGORITHM FOR THE ALPHA SYSTEM

concerning the system hardware and software should add to the understanding of this algorithm.

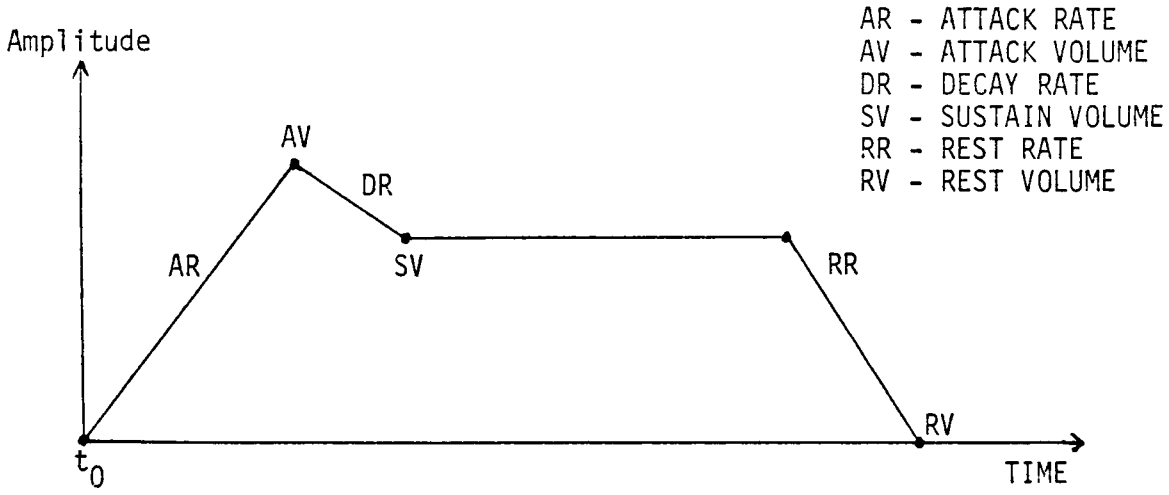
The Syntauri keyboard had an upper and lower contact associated with each key. Thus, each key could be in one of the following three possible states: up, partially down, or completely down. Each contact was mapped to a specific bit in memory located on the interface card. Since there were 122 contacts, the bit map was sixteen bytes long. There was a routine in the system software which read this bit map and returned an eight bit value which indicated the key number in the least six significant bits. The most significant two bits indicated which of the three possible states this key was in. Having two contacts for each key allowed the software to calculate the velocity of the key as it traveled downward. Since key velocity should be proportional to how hard a key was hit, the velocity sensing ability gave the user a control over the relative loudness of each individual tone sounded.

The Mountain Computer Music Synthesis Cards gave the software the ability to control all of the attributes of musical tone including the types of waveforms used. The cards were plugged in adjacent to one another in two of the Apple's peripheral I/O slots. The cards provided stereo output with each output channel being driven by eight oscillators. Each oscillator had the following

three software selectable controls: frequency, loudness, and waveform. The waveform setting of an oscillator pointed to the memory page number of a 256 byte table in which the pattern of the waveform was stored. The frequency setting indicated the size of the sampling steps to be taken through the waveform table. According to sampling theory, the maximum step size could be no more than half of the length of the waveform table. The oscillators were able to receive samples from the waveform tables without the aid of the system software because of the Direct Memory Access (DMA) capabilities of the synthesizer cards. The DMA fetched one byte of waveform information every two microseconds. At this rate it could update each of the sixteen oscillators once every thiry-two microseconds. With a sampling rate of 31,250 samples per second, each oscillator was capable of producing frequencies as high as 15,000 Hz. The loudness setting allowed each oscillator to generate its own envelope pattern. For each tone output by the system, the software assigned two oscillators, one for each of the output channels. The waveform produced by one oscillator was labeled as the primary waveform of the tone, and the waveform produced by the other oscillator was labeled as the percussion waveform.

At the end of each loop through the software, the system tested to see if any key on the Apple's keyboard had been pressed. If so, control was turned over to a portion of the Alpha system written in BASIC which provided the user with an interactive interface to the rest of the system which was written in 6502 machine language. This interface allowed the user to change various settings, referred to as presets, within the machine language side of the system. These presets influenced such things as the range of frequencies to be represented by the music keyboard, the shape of the envelopes formed, and the mode in which the system was to perform (ie. record or playback). An example of the method used by the Alpha system to define the shape of primary and percussion envelopes using presets is given in Figure 3. Due to the nature of the Applesoft Basic language and its lack of true subroutine programming capabilities, all of the preset changes were accomplished by way of PEEKs and POKEs to various memory locations accessible to both sides of the system. This made the interface code almost entirely unreadable unless a list of the common memory locations and the nature of their contents was made available. Any programmer would have trouble understanding what this code was meant to accomplish.

PRIMARY ENVELOPE PRESETS



PERCUSSION ENVELOPE PRESETS

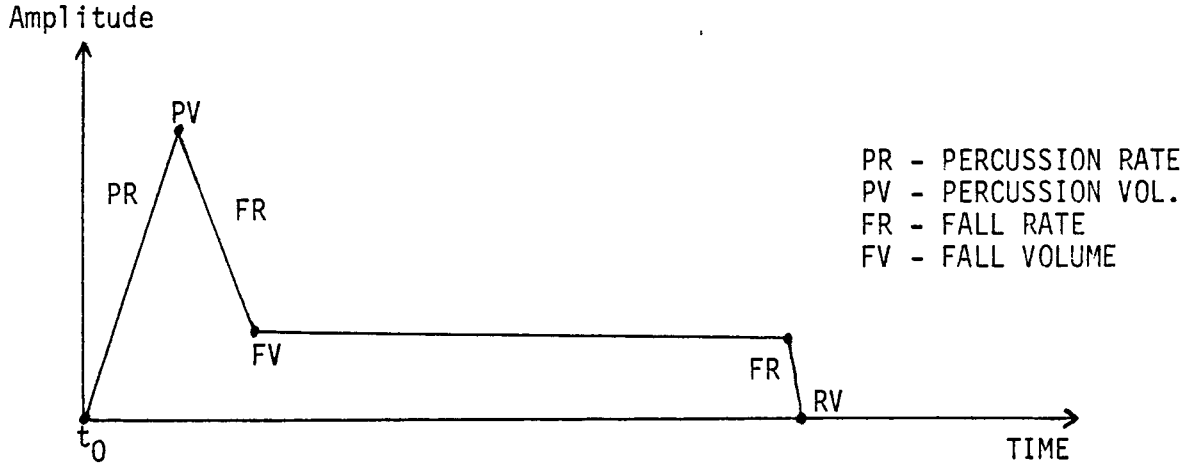


FIGURE 3

ALPHA SYSTEM'S METHOD OF DEFINING ENVELOPE SHAPES

In conclusion, the Alpha system is an excellent digital music system with a well focused purpose and the ability to create and record music of high tonal quality. However, due to its inability to transcribe recorded music, a CAMI language based on this system is not immediately possible. The next system to be discussed attempted to add the ability of transcription to the Alpha system without changing the basic design of the system.

4. MONDAY MERCER'S PROGRAM FOR COMPARISON OF NOTE FILES GENERATED BY THE ALPHA III MUSIC SYSTEM

This program provided a transcription system which normalized and compared two event-duration files recorded using the Alpha Plus music system. One file was considered to be the master file which represented a recording of music played as exactly as possible by a music instructor. The other file was considered to be a student file which represented a person's effort to duplicate the keyboard playing ability of the master. Due to the way the Alpha system records music played on the keyboard and due to the variable complexity of the Music that can be played on the keyboard, the major effort in this work was developing the normalization algorithms.

According to Ms. Mercer, "Many problems exist in duplicating note files in their original form, such as the order notes are pressed, the order they are released, the

extra presses and releases recorded that do not sound [upper contact being touched but not lower contact], and the time it takes keys to go from the upper to the lower contacts or from the lower to the upper contacts." A good example of this statement is shown in Figure 4, which was reproduced from Ms. Mercer's thesis. Since the student cannot be expected to duplicate events which are not even perceptible, these events must be removed and the remaining events ordered such that comparison can be made more directly and objectively.

An example of the output resulting from comparison of normalized files is shown in Figure 5, which is also reproduced from Ms. Mercer's thesis. In this example both the master and the student attempted to play two measures of Mozart's Sonata Number 1 in C Major. In this example, both the role of the master and the role of the student were played by Dr. Donald Pederson. The comparison routines split the sequence of keyboard configurations used to play the music into frames. The master's frames were lined up against the student's frames, were compared frame by frame, and any differences were noted to the right side of each frame pair. In this case, the student played key for key the same music as the master except in the last frame where the student forgot to play the C note of the fourth octave. This error was marked by a "X" in the rightmost column. In eleven of the

```
>RUN NOTEPEEKER
]

TYPE FILE NAME: E2B
E2B
29
0     0
88     0
92     0
95     0
223     0
216     0
220     0
93     2
29     254
22     24
0     28
2     31
0     88
2     24
108     255

]
```

* Zero time units after Key 28 is fully depressed,
Key 29 is touched and 2 time units later,
Key 29 is released having never been sounded.

```
Key# +   0 = Key# Released
Key# +  64 = Key# Touched
Key# + 192 = Key# Full Depressed
       254 = 253 Time Units + Flag Continuation
       255 = End of Recording
```

FIGURE 4

EVENT DURATION DATA RECORDED BY THE ALPHA SYSTEM

```
 RUN
INPUT MASTER FILE NAME: M6

INPUT STUDENT FILE NAME: NA6

 C--INDICATES NOTES PLAYED BY COMPUTER
 S--INDICATES NOTES PLAYED BY  STUDENT
 X--INDICATES NOTE(S) MISSED
 W--INDICATES DURATION AT LEAST 16 OFF
-1--INDICATES A REST
**************************************
NOTES PLAYED BY (MASTER/STUDENT)  HELD

M: C 3,C 4,                        40
S: C 3,C 4,                        56

M: G 3,C 4,                        40
S: G 3,C 4,                        56

M: E 3,C 4,                        36
S: E 3,C 4,                        76  W

M: G 3,C 4,                        36
S: G 3,C 4,                        84  W

M: C 3,E 4,                        32
S: C 3,E 4,                        84  W

M: G 3,E 4,                        32
S: G 3,E 4,                        100 W

M: E 3,G 4,                        40
S: E 3,G 4,                        80  W

M: G 3,G 4,                        32
S: G 3,G 4,                        68  W

M: D 3,B 3,                        40
S: D 3,B 3,                        100 W

M: G 3,B 3,                        36
S: G 3,B 3,                        112 W

M: F 3,B 3,                        36
S: F 3,B 3,                        120 W

M: G 3,C 4,                        20
S: G 3,C 4,                        52  W

M: G 3,D 4,                        16
S: G 3,D 4,                        132 W

M: C 3,C 4,                        36
S: C 3,C 4,                        36

M: G 3,C 4,                        32
S: G 3,C 4,                        48

M: E 3,C 4,                        40
S: E 3,C 4,                        56

M: G 3,C 4,                        40
S: G 3,                            60  X
```

FIGURE 5

OUTPUT FROM COMPARISON ROUTINES USING M6 AND NA6

frames however, the student's tempo was drastically off and this was marked by a "W" in the rightmost column. It is obvious that if the student introduced to the response an extra frame which did not exist in the master's example, the entire comparison becomes meaningless from that point onward unless the student completely fails to play another frame down the line. In the cases where the student's number of frames were unequal to the master's number of frames, the student's response was ruled not comparable.

As it stood, the comparison routine required the following scenario to occur in order to get output:

1. The instructor played and saved a music example on disk using the Alpha Plus system.
2. The instructor ran the normalizing routine on the file just saved. The event-duration data in the file was divided into frames. The frames were then saved on disk under a name related to the event-duration file to which it corresponds.
3. At some later point in time, the student ran the Alpha Plus system, attempted to play back the instructor's example on the keyboard, and saved the resulting recording on disk under some file name specified by the instructor.

4. The student then ran the comparison program and received the results on the screen in the format shown in Figure 5.

Of course, to produce a truely interactive CAMI course given all of the software described above, one would have to find a way to combine the BASIC driver of the Alpha Plus system with Ms. Mercer's normalizing and comparison routines. The programming required to accomplish this would impose the burden of having to understand the interface between the BASIC driver and the central 6502 machine language program. It is the opinion of the author that the need for such understanding should be removed from the instructor's programming efforts. An instructor trying to develop a computer assisted instruction course in any field should be able to program the course in a single programming language using the standard procedures of that language plus a few tools or extended procedures which complement the field of study being taught. The programming language used should never be as low level as machine or assembly language; and the programming language should never have to communicate with the procedures or tools at a level below that of a CALL statement.

CHAPTER 2

GUIDE TO USING THE MUSIC COMMAND INTERPRETER

1. HOW TO USE THE MUSIC COMMAND LANGUAGE

This chapter is intended to help a person with experience in Applesoft BASIC to program a CAMI course using MCI. Figure 6 shows a portion of a program which uses MCI to perform the following:

1. The user is asked to listen to a tune preceded and followed by four bass notes and to respond by playing back only the tune heard. The bass notes act as a metronome to set the tempo for the user.
2. The four bass notes are heard, followed by a short tune of thirteen notes, followed by four more bass notes.
3. The user has a fixed amount of time in which to begin the response and once the response begins, the user has the same fixed amount of time in which to finish the response.
4. If the response is completed before time is up, the user can either signal this fact to MCI by pressing the space bar or by waiting for the response time to expire.
5. The user is then shown the musical string notation returned from MCI's listening routines as the corresponding music is played back.

```
2600  REM  MAIN PROGRAM FOLLOWS
2700  REM
2900 MC$ = "TEMPO=03": CALL MCI
3000 MC$ = "RESPONSE=04": CALL MCI
3100 MC$ = "DELAY=18": CALL MCI
3200 MC$ = "WTIE.MIN=12": CALL MCI
3300 MC$ = "VOL=45": CALL MCI: REM  ACTUAL.VOLUME = VOL * 4
3400  REM
3600 B$ = "                    "
3700 B240$ = B$ + B$ + B$ + B$ + B$ + B$ + B$ + B$ + B$ + B$ + B$ + B$
3800  REM
4000 METRO$ = "( 17T 61E. T61T 17T 61E. T61T  17T 61E. T61T 17T 61E. T61T "
4100 TUNE$ = " 45Q 46E 45E 43E 41E 45Q. 46E 48Q 50E 46E 45Q 43Q 41H. )"
4200  REM
4400  PRINT "YOU ARE ABOUT TO HEAR A METRONOME FOLLOWED BY A 13 NOTE TUNE ";
4500  PRINT "FOLLOWED BY THE METRONOME AGAIN.  AFTER THE FOURTH BEAT OF THE ";
4600  PRINT "SECOND METRONOME, PLAY BACK THE 13 NOTE TUNE THAT YOU JUST HEARD."
4700  INPUT "PRESS RETURN WHEN YOU ARE READY TO BEGIN:";I$
4800  REM
5000 MC$ = "PLAY" + METRO$ + TUNE$: CALL MCI
5100 MC$ = "PLAY" + METRO$ + ")": CALL MCI
5200 LS$ = "LISTENS"
5300 MC$ = LS$ + B240$: CALL MCI
5400  PRINT : PRINT : PRINT
5500  PRINT "I HEARD YOU PLAY THIS: ";MC$
5600 MC$ = "PLAY" + MC$: CALL MCI
5700  REM
5900  REM
6000  PRINT "YOU ARE ABOUT TO HEAR A CHORD CONSISTING OF 3 NOTES."
6100  PRINT "AS SOON AS THE CHORD IS THROUGH PLAYING, PLAY IT BACK ON THE KEYBOARD."
6200  INPUT "PRESS RETURN WHEN YOU ARE READY TO BEGIN:";I$
6300  REM
6600 CS$ = "( 41,45,48W )"
6700 MC$ = "PLAY" + CS$: CALL MCI
6800  REM
6900 LC$ = "LISTENC=03"
7000 MC$ = LC$ + B240$: CALL MCI
7100  REM
7200  PRINT "I HEARD YOU PLAY THIS: ";MC$
7300 L =  LEN (MC$) -  LEN (CS$):XS$ = CS$ +  LEFT$ (B240$,L)
7400 I$ = "SORRY, THAT'S NOT WHAT I PLAYED."
7500  IF (MC$ = XS$) THEN I$ = "WOW MAN !!!  THAT'S WHAT I PLAYED."
7600  REM
7700 MC$ = "PLAY" + MC$: CALL MCI
7800  PRINT I$
7900  END
```

FIGURE 6

BASIC PROGRAM USING MCI

6. The user is asked to listen to a chord consisting of three notes and to respond by playing back the chord heard.
7. The chord is heard for a duration of one whole note.
8. The user has a fixed amount of time to complete the formation of a three note chord.
9. As soon as three different keys have been detected to be pressed and released, the chord is recorded and the response is over.
10. The user is the shown the musical notation returned from MCI's listening routines as the corresponding music is played back.
11. Finally, the user is informed whether or not the correct chord response was given.

This program illustrates how each of the commands in the music command language may be used. The remainder of section 2.1 gives an explanation of each of these commands along with their syntax.

All commands are invoked using the following general form:

$$\text{MC\$} = \left(\begin{matrix}\text{string-variable}\\ \text{string-constant}\end{matrix}\right)\left[+\left(\begin{matrix}\text{string-variable}\\ \text{string-constant}\end{matrix}\right)\right]^{*} \quad : \text{CALL MCI}$$

The string variable MC$ must be reserved for the assignment of music commands. The music command is always a string constant or stored in a string variable or is formed by the concatenation of substrings. The variable

MCI is equal to the address where the interpreter begins. Issuing a call to this address gives the control of the system to MCI which then performs the command stored in the string variable MC$. The two types of music commands recognized by MCI are commands which alter certain parameters used by MCI such as volume and tempo and commands which cause musical Input/Output to be performed. The musical I/O commands will be discussed first.

"PLAY(k_0 k_1 ... k_n)"

The PLAY command causes MCI to play zero or more key sets. The symbol k above indicates a key set. A key set describes the keys which should be playing at a particular time, the duration of this playing, and whether these keys are already down or are to be attacked. A key set has the following form:

[T]nn[,nn]* d[d]b

where nn represents a two digit number between 00 and 61, d represents a character used to form one of the duration symbols, b represents a blank, and T, if present, indicates that the key set is musically tied to the previous key set. Music represented by one or more key sets contained within parenthesis will be considered to be in the form of "musical string notation".

The keys on the Syntauri board are numbered from left to right as 00 to 60. The key number 61 is used to represent a rest note. Here are the duration symbols, their meanings, and their relative durations:

T	thirty-second	3
S	sixteenth note	6
ET	eighth-triplet	8
S.	dotted-sixteenth	9
E	eighth note	12
QT	quarter-triplet	16
E.	dotted-eighth	18
Q	quarter note	24
Q.	dotted-quarter	36
H	half note	48
H.	dotted-half	72
W	whole note	96

Shown in lines 5000, 5100, 5600, 6700, and 7700 of Figure 6 are examples of PLAY command strings which are formed by the concatenation of various literal and/or variable strings. Although strings are limited to a length of 255 in Applesoft, a tune requiring a large number of key sets can be played by dividing the tune into two or more PLAY commands which are invoked one after the other. The entry and exit from MCI is so quick that no unintentional rest is heard between the PLAY commands.

"LISTENS"

The LISTEN-Sequence command causes MCI to record the keys pressed on the Syntauri keyboard as a sequence of notes played and to return a transcription of this recording to BASIC in musical string notation. If more than one key is pressed at a time, only the most recent

key is heard to play and the others are considered to be released. In effect this gives the keyboard only one key rollover. This extra control over the keyboard input allows for a simpler transcription technique than would otherwise be required. The LISTENS command is to be used for teaching tempo and/or learning pitches produced by the individual keys on the keyboard. The transcription results are returned to BASIC as the contents of MC$. Since the maximum length of a string is 255, MCI limits the resulting string to a length of approximately 245 and places a + symbol immediately following the closing parenthesis if it had to cut the result short to prevent an overflow.

To invoke a LISTENS command, MC$ must be set equal to the string "LISTENS" concatenated with another string. Otherwise, MCI would not only alter the contents of MC$ but would also alter the contents of other string variables. This is due to the method Applesoft uses to perform string assignment statements. It is suggested that at some early point in the program, a string variable of 240 blanks be constructed as in lines 3600 and 3700 of Figure 6 and that this be the string variable concatenated with the LISTENS command.

From the time LISTENS is invoked, the user has a specific amount of time to begin playing the keyboard. Once playing has begun, the time left is reset to the full

amount so that the user has enough time to complete the required tune. It is recommended that the length of time allowed for a sequence response be set approximately one whole note longer than the length of the tune to be played. At any time previous to, during, or after the response, the user may press the space bar which will terminate recording and begin the transcription process followed by return to BASIC.

"LISTENC=nn"

The LISTEN-Chord command is designed for teaching the pitch combinations needed to form various musical chords. The command must be invoked, as with the LISTENS command, by setting MC$ equal to the string "LISTENC=nn" concatenated with another string before calling MCI. The nn represents a two digit number which indicates the maximum number of keys which will be recorded as pressed. Above this number, the user can press other keys but no new notes will be heard. Following the completion of the user's response, the results will be transcribed to form a key set and returned to BASIC in the variable MC$ in musical notation form. The keys placed in the key set are numerically ordered by MCI. From the time LISTENC is invoked, the user has only the time alloted to complete the response. Unlike LISTENS, this time is not reset once the user's response begins.

This command might also be used to record responses requiring only single key presses such that a return to BASIC is initiated as soon as the key is released. This would be achieved by the command "LISTENC=01". On the other hand, free play of the keyboard with eight key rollover could be achieved by the command "LISTENC=99" and by allowing a long response time. This may prove useful in situations where the instructor wants to allow the student to warm up on the keyboard without being monitored. Now that the musical I/O commands have been covered, the parameter changing commands will be discussed.

"TEMPO=nn"

The TEMPO command sets the length of the time units used to measure duration. As the time units are set smaller, the tempo gets faster. MCI has this parameter set to a default value of 02 which is the fastest tempo that should be used. The suggested range is from 02 to 05.

"RESPONSE=nn"

The RESPONSE command sets the amount of time in which the user has to respond during either of the listening commands. The time set is measured in whole notes. The actual time varies depending upon the tempo setting. The

time setting should allow slightly more time than is necessary to complete the desired response. For example, an instructor who wishes the student to play a sequence with a total duration of three whole notes would probably want to set the time to 04. MCI has this parameter set at a default value of 06. The possible range is from 01 to 99.

"DELAY=nn"

The DELAY command sets the relative maximum delay allowed between lifting a finger off of a key and putting it down on another key before a rest note is considered to have occured. This parameter is used by the routines which transcribe sequential input into musical string notation. If it is below the maximum duration, the delay is added back to the key from which the finger was lifted. If it is greater than or equal to the maximum, the delay is considered to be an intended rest and is transcribed as such. By controlling the delay parameter, one has a control over the exactness or the leniency of the transcription process. The actual delay time depends on the tempo setting. The default setting for DELAY is 24 which corresponds to a quarter note. The possible range is from 01 to 96.

"WTIE.MIN=nn"

The WTIE.MIN command sets the minimum duration of time greater than a whole note that a key must be pressed before the total duration should be considered as a whole note tied to another note. The actual time depends on the tempo setting. Depending on the WTIE.MIN setting and assuming that a user intended to play a whole note, MCI will ignore a trivial mistake, such as holding a key for a duration of ninety-nine time units instead of ninety-six. If MCI accounted for the extra three time units, then the total duration would be transcribed as a whole note tied to a thirty-second note. Instead, the transcription routine throws away the extra sliver of time. MCI has this parameter set at the default value of 24. The possible range is from 01 to 96.

"VOL=nn"

The VOL command is used to set the maximum value for the Mountain Computer card's overall volume control. MCI takes this relative value and multiplies it by four to get the actual maximum. MCI increases the volume to this level when it is called to perform musical I/O and turns the volume off as soon as it completes the I/O. MCI has this parameter set at a default value of 50. Since the actual volume range is from zero to 255, the possible range for the relative value is from 00 to 63. If a

relative value above 63 is used, then the actual volume is set equal to four times the difference of the relative value and 64.

"KBSLOT=nn" and "MHSLOT=nn"

These two commands are used to change the default settings for the slot locations of the keyboard interface and the leftmost Mountain Computer card. The default values are 05 and 02 respectively. These commands should only be used in programs run on systems which are configured differently. They are not shown in the example program since the cards were already in their default slots.

2. SETTING UP AN APPLESOFT PROGRAM WHICH USES MCI

Before one can use the music commands, a few programming details need to be discussed. Figure 7 demonstrates how the beginning of an Applesoft program should look. The first concern is where MCI and the various tables it uses should be loaded into memory. The MCI source code is organized such that all of the data and tables used by the program are positioned before the starting address of the executable code. At loading time, MCI is loaded first and the tables are placed in their appropriate locations. In this example, MCI was assembled to be loaded beginning at hexadecimal page 80 which is

```
500  REM  SETTING UP TO USE MCI.
510  REM
520  REM
1000 PADDR = 32768: HIMEM: PADDR: REM ( 32768. = $8000 = PAGE $80 OF MEMORY )
1025  REM
1050 MC$ = "NULL"
1100 D$ =  CHR$ (4)
1150  REM
1200  PRINT D$;"BLOAD MCI.OBJ0"
1250  REM
1300  PRINT D$;"BLOAD MCI.PRESET,A";PADDR: REM  PRESETS = SCALE + INC + LIM + FC
1350  REM
1400 W1ADDR = PADDR + 256
1500  PRINT D$;"BLOAD MCI.WAVE1,A";W1ADDR
1550  REM
1600 W2ADDR = W1ADDR + 256
1700  PRINT D$;"BLOAD MCI.WAVE2,A";W2ADDR
1750  REM
1800 VLADDR = W2ADDR + 256
1900  PRINT D$;"BLOAD MCI.VLOG,A";VLADDR
1950  REM
2000 CTADDR = VLADDR + 256
2100  INPUT "USE PICKY/MOD/EASY CONVERSION TAB:";I$
2200  PRINT D$;"BLOAD MCI.C.TAB.";I$;",A";CTADDR
2250  REM
2300 MCI = CTADDR + 256
2400  REM
2500  REM
2600  REM  MAIN PROGRAM FOLLOWS
2700  REM
2800  REM
```

FIGURE 7

BASIC PROGRAM LOADING MCI AND ASSOCIATED TABLES

equal to decimal address 32768. HIMEM is set equal to this address so that Applesoft will not try to store the values of any variables above this address. If MCI is assembled to be loaded elsewhere, the only change in the source code is to the first line which normally reads "TOPPAGE EQU $80". The source must be assembled to load starting at a whole page address. The object code requires at least 6K bytes of load space.

The file MCI.PRESET contains the frequency scale to be used and the presets which describe the shapes of the primary and percussion envelopes to be used. This file can be modified using a program called MCI.PRESET.EDIT. The details needed to run this program are discussed in Appendix A. MCI.PRESET is loaded starting at the address pointed to by the variable PADDR. This file is 81 bytes long.

The files MCI.WAVE1 AND MCI.WAVE2 contain the waveform tables for the primary and percussion envelopes respectively. As described in section 1.3, the tables are each 256 bytes long. The file MCI.VLOG is also 256 bytes in length. It contains a logarithmic conversion table used by MCI's envelope generating routines to calculate the envelope volume settings. In order to create different waveform files, the program called WAVE III which is provided by Syntauri should be used. Likewise, Syntauri's program called LOG III should be used to create

a different MCI.VLOG file if so desired. In order to run these programs correctly, one should refer to the Alpha Syntauri manual.

The files MCI.C.TAB.EASY, MOD, and PICKY represent three levels of conversion exactness which can be used to transcribe the key duration tables recorded in the sequence routines into the musical string notation which is returned. Each conversion table is 256 bytes long. Although only one may be loaded at a time, they can be changed periodically throughout the program if the instructor wishes to vary the exactness with which the student must respond. The files can be edited to suit the instructor's taste by using a program called MCI.C.TAB.EDIT. The details needed to run this program are discussed in Appendix B.

In line 1050 of Figure 7, MC$ is set equal to the literal string "NULL". The actual contents of the string does not matter. In order for MCI to find it, the string variable MC$ should be the first string variable used in the program. If MCI fails to find MC$ then the interface between MCI and BASIC is lost. Also notice in line 2300 MCI is set equal to the address immediately following the conversion table. This is the location at which the executable code of MCI begins. So, to send MCI a command, set MC$ equal to the command and issue a call to the address stored in the variable MCI.

CHAPTER 3

DESIGN OF THE MUSIC COMMAND INTERPRETER

1. OVERALL SYSTEM DESIGN

As with the systems described in Chapter 1, the timing method used is a central part in the design of this real time system. Timing is only used for the musical I/O commands. Time is kept with an interrupt service routine which keeps track of several clocks and which sets a flag to communicate to the rest of the system when it is time to seek new key-event data. The interrupts are generated once every eight milliseconds by the clock on the Mountain Computer card. The algorithm used for the interrupt service routine is shown in Figure 8. The main clock of this routine keeps track of the interrupts as they are generated. This clock is initially set to the tempo value but is decremented upon each interrupt until it reaches zero. When this occurs, the main clock is reset and the whole-note clock is decremented along with the general timer clock. When the whole-note clock reaches zero, the response time clock is decremented. When the system is in listening mode, the response time clock keeps track of the time remaining for the user to complete the response. When the general timer clock expires, a timeout flag is set and the system becomes aware that it is time to seek new key-event data. The general timer clock is reset by

```
ENTRY via Round-About-Trip through Apple-Monitor
                          which TRASHES A-Register;
SALVAGE  A-Register from Memory-Location-$45;
PUSH  Registers;
DECREMENT  Main-Interrupt-Clock;
IF ( Main-Interrupt-Clock  =  0 ) then
  Begin
    Main-Interrupt-Clock := Tempo;
    DECREMENT  Whole-Note-Clock;
    IF ( Whole-Note-Clock  =  0 ) then
      Begin
        Whole-Note-Clock := Whole-Note-Duration;
        DECREMENT Response-Time
      End;
    DECREMENT  General-Timer;
    IF ( General-Timer  =  0 ) then  Timeout := TRUE
  End;
POP  Registers;
RETURN-from-INTERRUPT;
```

FIGURE 8

ALGORITHM FOR THE INTERRUPT SERVICE ROUTINE

one of the other system routines depending on the type of musical I/O that the system is performing. .

With the timing method in mind, the overall system design will now be examined. The algorithm for the Music Command Interpreter is shown in Figure 9. The basis of the programmer-friendly interface offered by MCI to Applesoft involves the ability of MCI to locate the string variable MC$ which contains the music command. After the string has been located, the type of music command which has been sent must be determined. If it is a parameter type command, the value sent is converted to a hexadecimal value and is stored in the appropriate data location. If the command is not recognized due to some syntactical problem, then control is returned to the BASIC program with no action having been taken. Therefore, if MCI does not seem to respond to a particular command, then it is probably due to the syntax of the command used.

If the command requires musical I/O, then the algorithm becomes more complex. First, all of the synthesizer hardware controls are initialized. Next, the system flags must be initialized to indicate which one of the musical I/O commands is being performed. Then, the main loop of the system is entered. If in play mode, the system continues to play the key sets until there are no more or until the user presses the space bar. If in sequence listening mode, the system continues to record

```
SAVE  Zero-Page-Locations used-by MCI;
LOCATE  Music-Command-String  SENT from BASIC;
DETERMINE  Command-Type;
IF ( Command-Type .requires. Music-I/O ) then
  Begin
    INITIALIZE  Synthesizer-Hardware-Controls;
    IF ( Command-Type  =  Play ) then  INITIALIZE System for Playing
    ELSE  INITIALIZE  System  for  Listening;
    CLEAR  Return-to-BASIC-Flag  and  Timeout-Flag;
    START  ANSWERING  INTERRUPTS;
    REPEAT
      IF ( Timeout ) then
        Begin
          CLEAR  Timeout-Flag;
          IF ( Command-Type  =  Play ) then
                GET  Tonelist from Play-Update
          ELSE  IF ( Command-Type  =  Listen-S ) then
                GET  Tonelist from Listen-S-Update
          ELSE  GET  Tonelist from Listen-C-Update;
          GET-NEXT  Tone from Tonelist;
          WHILE ( Tone  <>  End-List-Code ) do
            Begin
              ASSIGN  Tone to Available-Pair-of-Oscillators;
              SET-FREQUENCY of  Available-Pair-of-Oscillators;
              GET-NEXT  Tone from Tonelist
            End;
          FOR ALL  Assigned-Oscillator-Pairs
            Begin
              UPDATE  Primary-and-Percussion-Envelope-Status;
              SET-LOUDNESS  of  Primary-Oscillator;
              SET-LOUDNESS  of  Percussion-Oscillator
            End
        End
    UNTIL ( Return-to-BASIC-Flag  .is.  SET );
    TURN-OFF  all  Synthesizer-Hardware-Controls;
    IF ( Command-Type  =  Listen-S ) then
        TRANSCRIBE  Key-Duration-Table  into  Music-String;
    IF ( Command-Type  =  Listen-C ) then
        TRANSCRIBE  Chord-Event-Data  into  Music-String
  End
ELSE  IF ( Command-Type  .requires.  Parameter-Change ) then
       MAKE  Parameter-Change
ELSE  CONTINUE;                    (* undefined command *)
STOP ANSWERING INTERRUPTS;
RESTORE  Zero-Page-Locations  used-by  MCI;
RETURN to BASIC;
```

FIGURE 9

ALGORITHM FOR THE MUSIC COMMAND INTERPRETER

until response time expires or until the user presses the space bar. If in chord listening mode, the system continues to record until the response time has expired, until the user has pressed and released the required number of keys, or until the user presses the space bar. Pressing the space bar in any of these cases results in the return to Basic flag being set.

In all of the musical I/O modes there are routines used which are responsible for producing a periodic list of the key-events which are to be simulated. This list is refered to as the tonelist. The routines are asked to produce a tonelist when a timeout has been flagged by the interrupt service routine. In the listening modes the general timer is set back to one by the listening update routines. This allows a quick response time to key-events and gives the update routines a method of keeping up with duration by counting the number of timeout situations between key-events. In the play mode, the general timer is set to the duration of the next key set.

The tonelist produced by an update routine is actually a list of codes which are identical in meaning to those used in the Alpha system to indicate which keys to assign and which to release from the oscillator pairs. In fact, the only Alpha software which has been relatively untouched by the MCI design is the software responsible for the assignment of oscillator pairs, that responsible

for generating and maintaining the waveform envelopes, and that responsible for getting key-event values from the keyboard bit map. Approximately 20 percent of the MCI code is composed of slightly altered Alpha system code. About 5 percent of the MCI code is significantly altered Alpha system code and the remaining 75 percent is completely new code.

2. DESIGN OF THE MUSICAL I/O ROUTINES

Although the play routine will recognize a string for input which is more complex than that produced by either of the listening routines, the design of the play routine was the simplest of the three to develop. Each tonelist returned by play update contains a list of new key events. The list begins with a code which indicates that all currently playing keys are to be released. In the case of a key set being tied to the previous key set, the play update routine resets the general timer to the new duration and sends back a single code in the tonelist indicating that no change in the keyboard state has taken place. The play routine simply sets a return to BASIC flag when there are no more note sets to process.

The most difficult routines to design were those that involved the recording and transcription of sequential musical input into musical string notation. The design of the listen update routine is the easiest of these to

explain. If the keyboard bit map indicates that a new keypress has occurred, then the tonelist returns with the code to have all assigned oscillators turned off, followed by the code to assign the new key to an oscillator pair. Whether or not the previously played key had been released, only the most recently pressed key would be heard or recorded. This simple recording technique only keeps track of the current key being played and the amount of time which passes before a different key or a rest is played. Such a recording is already normalized and ready for transcription, unlike the recordings produced by the Alpha system which recorded every single event detected on the keyboard.

The design of the transcription routine which converts the key-duration recording to musical string notation is more complex. The success of this routine is critical in reaching the goal of producing comparable strings. The algorithm for this routine is given in Figure 10. Each key and duration is processed in the main loop of this routine. Looking ahead to the next key and duration pair occurs in order to determine such things as possible ties and possible delays which need to be added to the current duration being processed. This main loop is executed until the recording is completely transcribed or until the recording produces a string which is too large to be stored in an Applesoft string variable.

```
Music-String := "( ";
String-Length := 2;
IF ( Pointer  >  0 ) then
  Begin
    String-Overflow := FALSE;
    Duration-Tie := FALSE;
    Key-Duration-Table( Pointer ) := End-of-Recording-Marker;
    Pointer := 0;
    REPEAT
      GET  Key and Duration  from  Key-Duration-Table( Pointer );
      GET  Next-Key and Next-Duration  from
                        Key-Duration-Table( Pointer + 2 );
      IF ( Key  <>  Rest ) then
        Begin
          IF ( Nest-Key = Rest  .and.  Next-Duration < DELAY ) then
            Begin
              Duration := Duration + Next-Duration;
              Pointer := Pointer + 2;
              IF ( Duration > Whole-Note-Duration ) then
                            Duration := Whole-Note-Duration
            End
        End;
      IF ( Duration-Tie ) then  PUT a "T" into Music-String;
      PUT  Key-Number into Music-String;
      CONVERT  Duration to Duration-Symbol;
      PUT  Duration-Symbol into Music-String;
      IF ( Key = Next-Key  .and.  Next-Duration >= WTIE.MIN ) then
                                              Duration.Tie := TRUE
      ELSE  Duration-Tie := FALSE;
      UPDATE  String-Length;
      IF ( String-Length  >=  240 ) then  String-Overflow := TRUE;
      Pointer := Pointer + 2
    UNTIL ( Key-Duration-Table(Pointer) = End-of-Record-Marker  .or.
                                                  String-Overflow );
  End;
Music-String( String-Length ) := ")";
IF ( String-Overflow ) then  Music-String( String-length+1 ) := "+";
```

FIGURE 10

ALGORITHM FOR TRANSCRIPTION OF KEYS RECORDED IN SEQUENCE

The only aspect of the design of the chord listening routines that is particularly interesting is the routine which records the key input according to the heuristics defined for the LISTENC command. The algorithm for this updating routine is given in Figure 11. It can be seen from the algorithm that pressing and releasing the same key several times does not trick the routine into counting this as several key releases. The main loop is continued until the keyboard state is stable or until the specified number of keys have been released. If the latter is the case, the return to BASIC flag will be set so that the system will respond by performing the chord transcription and returning to BASIC. The design of the chord transcription routine is much simpler. It merely sorts the keys in the key set into numerical order, places the whole-note duration symbol at the end of the key set, and places the key set into the string variable MC$.

```
ENTER from MCI MAIN-LOOP;
Tonelist := NIL;
READ-KEYBOARD-MAP  for Key-Event;
WHILE ( Key-Event <> No-Change-Code
        .and.    .not. Return-to-BASIC ) do
  Begin
    IF ( Key-Event .indicates. Key-Release ) .or.
        ( CARDINALITY( Key-Set ) < Chord-Size ) then
      Begin
        PUT  Key-Event  at-end-of  Tonelist;
        IF ( Key-Event .indicates. Key-Press ) then
           IF ( Key .element-of. Key-Set ) then
                          DECREMENT Release-Count
           ELSE  Key-Set := Key-Set .union. Key
        ELSE
          Begin
            INCREMENT  Release-Count;
            IF ( Release-Count = Chord-Size ) then
                                Return-to-BASIC := TRUE
          End;
        READ-KEYBOARD-MAP  for Key-Event
      End
    ELSE
      Begin
        KEY-Event := No-Change-Code;
        PUT  Key-Event  at-end-of Tonelist
      End
  End;
RETURN to MCI MAIN-LOOP;
```

FIGURE 11

ALGORITHM FOR RECORDING THE KEYS OF A CHORD

CHAPTER 4

RESULTS

Previous to this work, digital music systems developed for the Apple microcomputer were designed with the initial goals of accurately recording musical keyboard events and accurately playing back the recorded events if requested. The recordings were stored in an event-duration form which would have made editing a recording of only a few measures very laborious. Transcription of the event-duration recordings to a musical notation which could be edited was an afterthought of the initial goals. Once the recordings were transcribed, the user was not provided with a way to return the musical notation to an event-duration form which could be played back. The time critical portions of these systems were written in assembly language. Generally, the interfacing of assembly routines with the BASIC language was awkward. This was especially true when information needed to be passed between the two levels.

The MCI system was designed with two initial goals which were significantly different from the initial goals of previous musical systems. The initial goals of the design required MCI to provide a programmer-friendly interface to the BASIC language and to use an easy to translate musical string notation for recording. The work

required to design this new system has proven to be worth the effort.

The MCI system does not offer any musical special effects and is not as useful a compositional instrument as the Alpha system. However, in comparison to Ms. Mercer's CAI system which was designed to operate along with the Alpha software, MCI is more effective in several important ways. First, the musical string notation which MCI uses to send keyboard input back to the CAMI program is already normalized and is easily translated to other forms of musical notation. Second, the string notation may be used by the MCI playing routines without modification. Therefore, it would be a trivial task to write an Applesoft BASIC program which requested input from the music keyboard and then allowed the used to edit the string returned if so desired. The program could then allow the string to be tested by sending it to the MCI playing routines. If the user wanted to keep the string for a future use, the program could easily save it on disk. Such a program could be used by an instructor to create examples to be used in CAMI programs.

Another advantage of using MCI is that it allows an instructor to specify the type of musical input, either a tune or a chord, which is expected from the student. With this ability, the instructor can know the form of the string notation which will be returned. The instructor

should have several comparison routines available. Each of these should be designed to receive a pair of strings of a particular type and to return specific comparative information about these strings.

Using MCI, a CAMI program can keep many tunes in memory at a time by storing them in string arrays. Using the Alpha system, however, a CAMI program would have to read two tune files from the disk before the tune comparison process could begin. Since time delays in CAI programs usually increase the rate at which a student becomes fatigued with a lesson, it is a significant advantage that MCI does not require time-consuming disk accesses to pass information to the CAMI program.

The set of music commands which are used to change various parameters within MCI enable the BASIC programmer to produce code which is more readable than if POKE commands were required to be used. Also, since the programmer is not required to know the memory addresses of these parameters, MCI's data section may be modified without requiring changes in the CAMI programs.

The MCI routines which record sequential keyboard input use a conversion table to produce the string which is returned to the BASIC program. Using a conversion table to translate from integer duration values to musical duration symbols gives an instructor added flexibility in determining how close the timing of a student's must be to

the intended response in order to be considered correct. This degree of flexibility would not be possible if the conversion were performed using algorithms entirely.

The ultimate measure of the worth of a system designed to be used for Computer Assisted Instruction is its performance in the teaching and learning environment. The major goals for a CAI system to achieve before testing can even begin are the acceptance of the system by the group of instructors for which it is designed and the willingness of these instructors to write CAI programs which use the system. The Music department of the University of Tennessee at Knoxville has a digital music laboratory which contains several Apple computers with each computer having the hardware necessary to use MCI. Dr. Donald Pederson of the Music department and a few of his graduate students have been testing MCI. They are not only satisfied with the system's ability to provide apparent real time response to the music keyboard, but also are pleased with the ease with which a CAMI BASIC program can now be written.

Other major goals for a CAI system to achieve are the acceptance of the resulting CAI programs by the students and the willingness of the students to use these programs as a method of learning. To this date, the Music department has not had an opportunity to test an MCI-based CAMI program on a class of students. However, the

department is in the process of developing several MCI-based CAMI programs designed to complement already existing music courses. These courses involve refining the student's ability to keep time, to distinguish various tones from one another, to recognize various tonal combinations, and to respond on a musical keyboard. An increase in the average student performance in these courses would indicate that the MCI-based CAMI programs are fulfilling their intended purpose.

BIBLIOGRAPHY

BIBLIOGRAPHY

ALPHASYNTAURI User's Manual. (1980). Syntauri Ltd., Palo Alto, California.

Mercer, L. M. A System for the Comparison of Synthesized Music. Unpublished Master's Thesis, Department of Computer Science, University of Tennessee, Knoxville, August, 1982.

Music System Operation Manual. (1979). Mountain Computer Inc., Santa Cruz, California.

Ross, J. Music Transcription System. Unpublished Master's Thesis, Department of Computer Science, University of Tennessee, Knoxville, December, 1980.

APPENDICES

APPENDIX A

MODIFYING ENVELOPE PRESET FILES

The MCI system provides the ability to alter the timbre generated. The waveform and waveform envelopes may be changed by loading the appropriate tables into the corresponding data sections of MCI. Alpha Syntauri provides the software necessary to create new waveform tables. However, creating a new preset table requires an editing program provided along with the MCI system. This program is called MCI.PRESET.EDIT. By using this program, a specific timbre may be created which the instructor feels is most appropriate for presenting a particular musical example.

Figure 12 shows an example run of the MCI.PRESET.EDIT program. The user's responses are indicated by underlining. The program presents thirteen menu items from which to choose. An item is chosen by typing the key corresponding to the character to the left of that item. Pressing the carriage return is not necessary. The ten preset values used to describe the primary and percussion envelopes are listed first in the menu. In the example, a "2" was typed and the program responded by requesting a new value to be assigned to the corresponding preset. Next, the user selected to alter the frequency control (FC) setting. This setting is used in determining the

```
:RUN MCI.PRESET.EDIT
EDIT WHICH PRESET FILE?:MCI.PRESET

0:PR 200
1:FR 50
2:AR 200
3:DR 50
4:RR 100
5:PV 200
6:FV 150
7:AV 200
8:SV 175
9:RV 0
F:FC 30
T:TEST PRESETS ON KEYBOARD
E:END EDIT SESSION

CHOOSE PRESET TO BE CHANGED BY TYPING
ITS MENU CODE:2

ENTER NEW VALUE FOR PRESET #2:100

0:PR 200
1:FR 50
2:AR 100
3:DR 50
4:RR 100
5:PV 200
6:FV 150
7:AV 200
8:SV 175
9:RV 0
F:FC 30
T:TEST PRESETS ON KEYBOARD
E:END EDIT SESSION

CHOOSE PRESET TO BE CHANGED BY TYPING
ITS MENU CODE:F

ENTER FREQUENCY CONTROL VALUE:54

0:PR 200
1:FR 50
2:AR 100
3:DR 50
4:RR 100
5:PV 200
6:FV 150
7:AV 200
8:SV 175
9:RV 0
F:FC 54
T:TEST PRESETS ON KEYBOARD
E:END EDIT SESSION

CHOOSE PRESET TO BE CHANGED BY TYPING
ITS MENU CODE:T

PRESS RETURN TO EXIT TEST MODE: <CR>
0:PR 200
1:FR 50
2:AR 100
3:DR 50
4:RR 100
5:PV 200
6:FV 150
7:AV 200
8:SV 175
9:RV 0
F:FC 54
T:TEST PRESETS ON KEYBOARD
E:END EDIT SESSION

CHOOSE PRESET TO BE CHANGED BY TYPING
ITS MENU CODE:E

ENTER NAME OF FILE TO CONTAIN THESE
ENVELOPE SETTINGS:MCI.PRESET2
```

FIGURE 12

EXAMPLE EXECUTION OF MCI.PRESET.EDIT

frequency band which the keyboard will represent. The timbre produced by the current status of the presets can be tested by typing menu selection "T". Whenever "T" is chosen, MCI will be called using a "LISTEN=99" command. The response time will already have been set to "99". Control is returned to the editor by satisfying any of the following conditions: Apple return key is pressed, 99 musical key releases occur, or the duration of 99 whole notes elapses. Finally, the program may be exited by choosing "E". When this occurs, the user is asked to name the file to contain these new presets. The presets are then saved under that name and the program halts.

APPENDIX B

MODIFYING CONVERSION TABLE FILES

Following a recording of key-duration events, MCI uses a 256 byte conversion table to assist transcription of the recording into musical string notation. Considering a word as consisting of two bytes, then the first 96 words of the conversion table contain the information necessary to convert any duration value up to a whole note into the appropriate symbol code number. The remainder of the conversion table contains the information required to transcribe the assigned symbol code number into the appropriate musical string symbol. This particular information should never be altered. However, an editing program provided with the MCI system does allow the first 96 words of the table to be edited. Thus, a CAMI programmer has control over the exactness of the transcription process. This program is called MCI.C.TAB.EDIT.

Figure 13 shows an example run of the MCI.C.TAB.EDIT program. The user's responses are indicated by underlining. The menu provides four items from which to choose. In the example, the user selected to list the contents of the conversion table by typing "L". The listing first presents the non-editable information used to transcribe symbol code numbers into musical string

```
:RUN MCI.C.TAB.EDIT

EDIT WHICH CONVERSION TABLE?:MCI.C.TAB.PICKY


TYPE 'D' TO MODIFY A DURATION,
  OR 'R' TO MODIFY A DURATION RANGE,
  OR 'L' TO LIST THE CONVERSION TABLE,
  OR 'E' TO END EDITING . . . .L


CONVERSION TABLE CONTAINS THE FOLLOWING:
SYMBOL     SYMBOL #     STANDARD-DURATION
  T            1             3
  S            2             6
  ET           3             8
  S.           4             9
  E            5            12
  QT           6            16
  E.           7            18
  Q            8            24
  Q.           9            36
  H           10            48
  H.          11            72
  W           12            96

DURATION     SYMBOL #     EXCESS-DURATION
     1           0             0
     2           0             0
     3           1             0
     4           1             0
     5           2             0
     6           2             0
     7           2             0
     8           3             0
     9           4             0
    10           4             0
    11           5             0
    12           5             0
    13           5             0
    14           5             0
    15           5             3
    16           6             0
    17           6             0
    18           7             0
    19           7             0
    20           7             0
    21           7             3
    22           8             0
    23           8             0
    24           8             0
    25           8             0
    26           8             0
    27           8             3
    28           8             6
    29           8             6
    30           8             6
    31           8             8
    32           8             8
    33           8             9
    34           9             0
    35           9             0
    36           9             0
    37           9             0
    38           9             0
    39           9             3
    40           9             6
    41           9             6
    42           9             6
    43           9             6
    44           9             8
    45           9             9
    46          10             0
    47          10             0
    48          10             0
    49          10             0
    50          10             0
```

FIGURE 13

EXAMPLE EXECUTION OF MCI.C.TAB.EDIT

```
51          10          3
52          10          6
53          10          6
54          10          6
55          10          6
56          10          8
57          10          9
58          10          9
59          10          12
60          10          12
61          10          12
62          10          15
63          10          15
64          10          16
65          10          18
66          10          18
67          10          18
68          10          21
69          10          21
70          11          0
71          11          0
72          11          0
73          11          0
74          11          0
75          11          3
76          11          6
77          11          6
78          11          6
79          11          6
80          11          8
81          11          9
82          11          12
83          11          12
84          11          12
85          11          12
86          11          15
87          11          15
88          11          16
89          11          18
90          11          18
91          11          18
92          11          21
93          11          21
94          12          0
95          12          0
96          12          0

PRESS RETURN TO CONTINUE . . . .

TYPE 'D' TO MODIFY A DURATION,
  OR 'R' TO MODIFY A DURATION RANGE,
  OR 'L' TO LIST THE CONVERSION TABLE,
  OR 'E' TO END EDITING . . . .R

ENTER LOW END OF RANGE (1-95): 1

ENTER HIGH END OF RANGE (2-96): 2

ENTER CODE # OF SYMBOL ASSIGNED:1

ENTER EXCESS DURATION TO BE ASSIGNED:0

TYPE 'D' TO MODIFY A DURATION,
  OR 'R' TO MODIFY A DURATION RANGE,
  OR 'L' TO LIST THE CONVERSION TABLE,
  OR 'E' TO END EDITING . . . .D

ENTER DURATION (1-96) TO BE MODIFIED:93
ENTER CODE # OF SYMBOL ASSIGNED:11

ENTER EXCESS DURATION TO BE ASSIGNED:20

TYPE 'D' TO MODIFY A DURATION,
  OR 'R' TO MODIFY A DURATION RANGE,
  OR 'L' TO LIST THE CONVERSION TABLE,
  OR 'E' TO END EDITING . . . .E


WHAT IS NEW CONVERSION TABLE
TO BE CALLED?:MCI.C.TAB.PICKY2

]
```

FIGURE 13

(CONTINUED)

symbols. Next, the list presents the 96 possible duration values. Each duration value has an associated symbol code number and excess duration value. The symbol code number is used to directly assign the appropriate musical string symbol. The excess duration value indicates the amount of duration which remains to be assigned. This is used to handle cases in which one musical string symbol is not enough to describe the entire duration of a note. If the excess duration value is greater than zero, then the resulting musical string notation will contain the primary duration symbol musically tied to one or more secondary duration symbols.

After producing the listing in the example, the user selected to edit the range of durations from one to two. The user assigned these durations a symbol code number of one and an excess duration value of zero. Next, the single duration of 93 was edited. Finally, the user selected to end the editing session and named the file to contain the new conversion table settings.

VITA

Frank Stovall King, III was born in Nashville, Tennessee, on October 4, 1958. He graduated from Franklin High School in 1976. In August 1980, he received a Bachelor of Arts degree in Molecular Biology from Vanderbilt University. He received a Master of Science degree in Computer Science from the University of Tennessee in June 1983.

Mr. King is a member of the Institute of Electrical and Electronics Engineers. After graduation, he will be employed with the National Aeronautics and Space Administration at the Ames Research Center in Moffett Field, California.