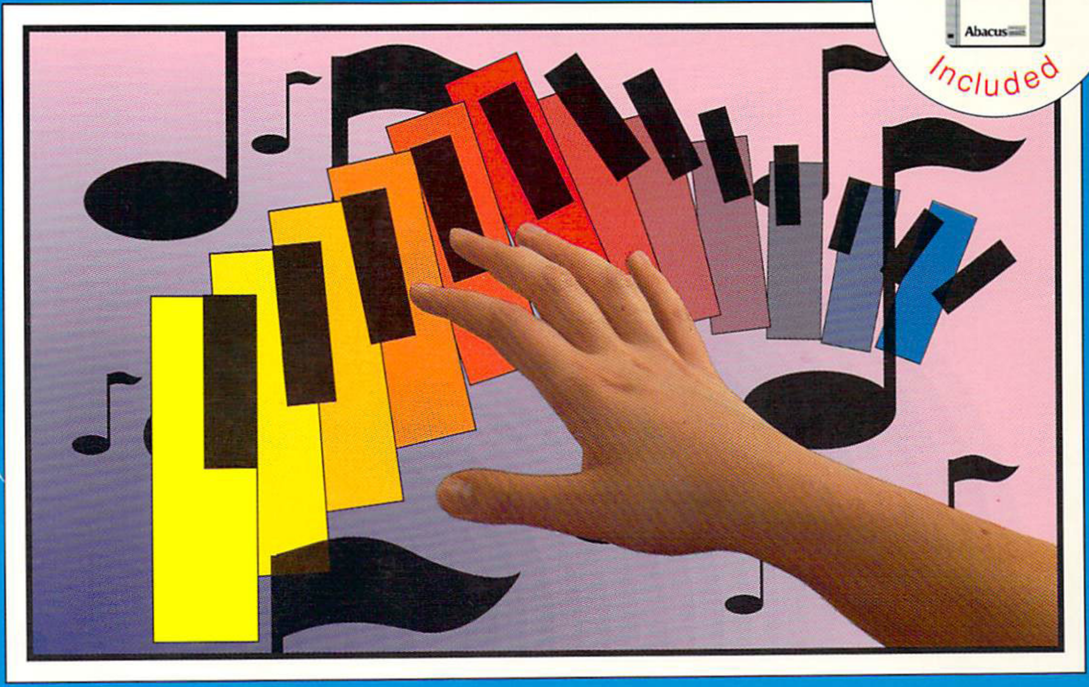
Making Music on the Amiga®

Comprehensive guide to understanding music on the Amiga

by Christian Spanik, Thomas Tai, & Holger Hahn

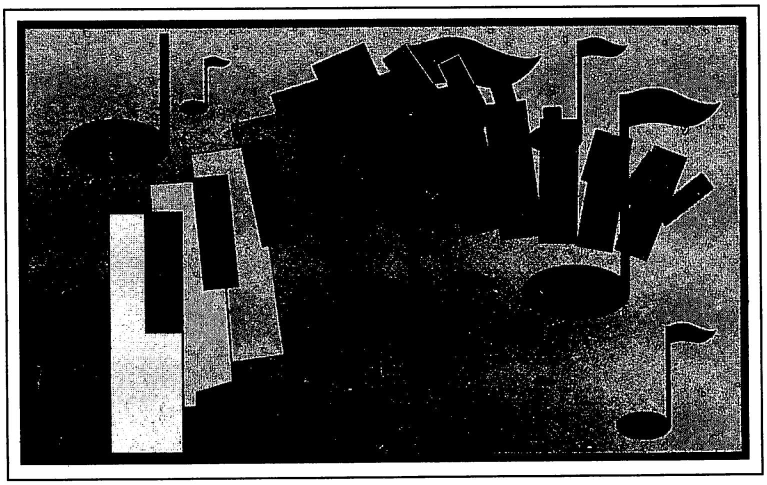
Companion Disk Included



Abacus
A Data Becker Book

Making Music on the Amiga

Christian Spanik
Thomas Tai, Holger Hahn



A Data Becker Book
published by
Abacus

First Printing
Printed in U.S.A.

Copyright © 1990	Abacus 5370 52nd Street SE Grand Rapids, MI 49512
Copyright © 1989	Data Becker, GmbH Merowingerstrasse 30 4000 Duesseldorf, West Germany
Editors	Louise Benzer, Jim Olsson, Gene Traas, Robbin Markley

Library of Congress Cataloging-in-Publication Data

```
Spanik, Christian, 1963-
   Making music on the Amiga / Christian Spanik, Thomas Tai, Holger
Hahn.
     p.    cm.
  "A Data Becker book."
  Includes index.
  ISBN  1-55755-094-8
  1. Amiga (Computer)--Programming.  2. Computer composition.
I. Tai, Thomas, 1966-    . II. Hahn, Holger, 1960-    . III. Title.
MT723.S6  1990
781.3'44165--dc20
                                                          90-14431
                                                               CIP
                                                                MN
```

Foreword

Many Amiga users bought their Amigas to create great graphics, fine video and good sound. Abacus has published books about the first two subjects, and now proudly releases this book on music and the Amiga.

We've covered much ground in Amiga music production in this book. You'll find information here about basic sound generation, wave shapes, analog and digital sound, and much more.

You'll also find overviews of some of the finest music software ever developed for the Amiga, from public domain, shareware and commercial sources. If we felt that some documentation was lacking on a software product, we did our best to give you some tutorial information to help you get started in some of these software tools.

In addition to existing software products, you'll also learn a little about writing your own Amiga music code.

For those of you who are handy with a soldering iron, you'll also find a number of schematics for Amiga music projects, such as MIDI interfaces and sound digitizers.

Enjoy!

Christian Spanik
Thomas Tai
Holger Hahn

Table of Contents

Chapter 1

Sound Generation

Chapter 1 Sound Generation

Instead of explaining the theory of sound diffusion or the advantages and disadvantages of various microphones, we'll discuss the minimum sound elements that every musician needs. We'll also discuss producing sound with your Amiga.

1.1 Generating a sound

Sounds are generated by a *transmitter*, sent through a *channel* and arrive at the *receiver*. You may hear these three elements called by other names, but we'll use the terms transmitter, channel and receiver in this book.

The simplest example would be the singing voice. When you sing, air passing through your vocal chords make the vocal chords vibrate, producing a sound. Our body amplifies the sound through different cavities (e.g., your throat and sinuses), and the sound of your singing voice emerges from your mouth. This sound is carried through the air and eventually reaches a listener's ears. Musical instruments operate more or less on the same principle.

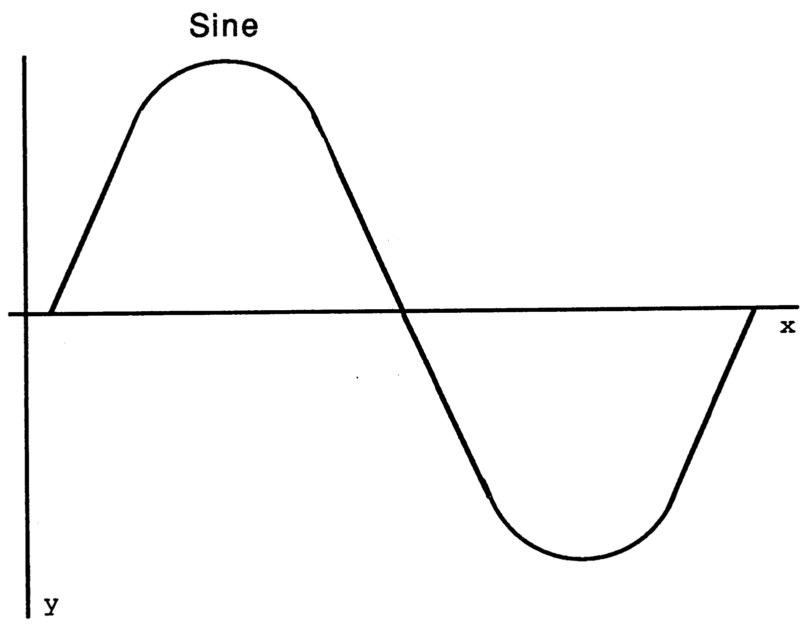
The Amiga generates sound through a series of electrical circuits. The electrical impulses representing this sound are transmitted through sound channels (more on this later) and sent to the Amiga's monitor speaker, where the sound is carried through the air to the listener's ears. At this point it isn't vital to know that the speaker causes the air molecules to vibrate. Just remember the basics: transmitter, channel, receiver.

1.2 What makes up a sound?

A sound can be described with four values: *intensity*, *duration*, *frequency*, and *timbre*.

Intensity	The intensity corresponds to the volume of a sound. It determines whether you are playing music "loudly" or "softly."
Duration	The duration represents the interval of time between the beginning and the end of a sound, as well as the interval between the beginning and end of a silence.
Frequency	The frequency (or pitch) determines how fast sound vibrates. A tuba plays at a lower pitch than a piccolo.
Timbre	The timbre (rhymes with "Amber") is also called *tone quality*. Different instruments can play at the same pitch, but will sound different because they have different timbres (i.e., an oboe and flute have different timbres because of they way sound is produced, their respective methods of construction, etc.).

In order to understand what makes up a sound, you must first know that a pure sound is represented by a *sine wave*. The following illustration shows a sine wave:



A pure sine wave sound doesn't really exist in nature, although the sound of the modern flute is fairly close. Most tone generators used in testing audio equipment can generate a true, pure sine wave.

A scientist named *Fourier* demonstrated that all natural sounds can be broken down into an infinite number of sine waves. Each of these sine waves has its own frequency and intensity. The sum of these sine waves represents the timbre.

Although you can create and add a certain number of these sine waves to the Amiga's onboard sound hardware, you will never have as many possibilities as with natural sound.

1.3 Analog sounds

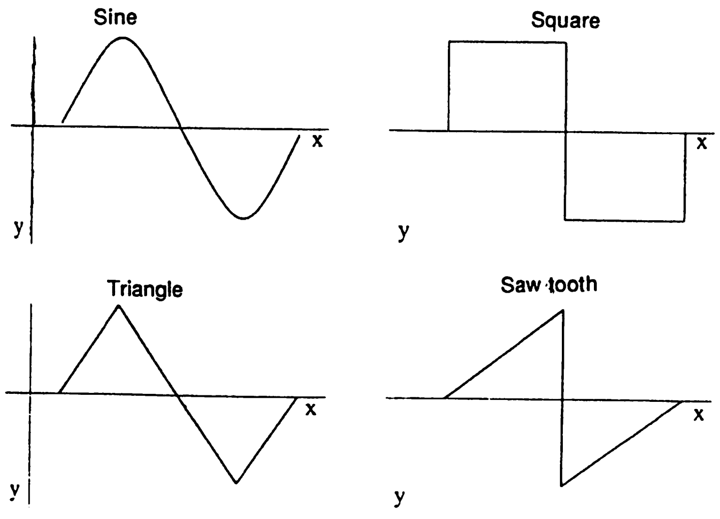
1.3.1 Wave shapes

When a computer or a synthesizer produces a sound, it is either *digital* or *analog*. Analog sounds are produced by one or more oscillators which produce a specific *wave shape*. Usually these wave shapes are square, sine, triangle and sawtooth waves.

Analog music synthesizers (e.g., the synthesizers manufactured by Robert A. Moog in the 1960s and 1970s) usually have square, sine and triangle oscillators. These oscillators can often be used in combination.

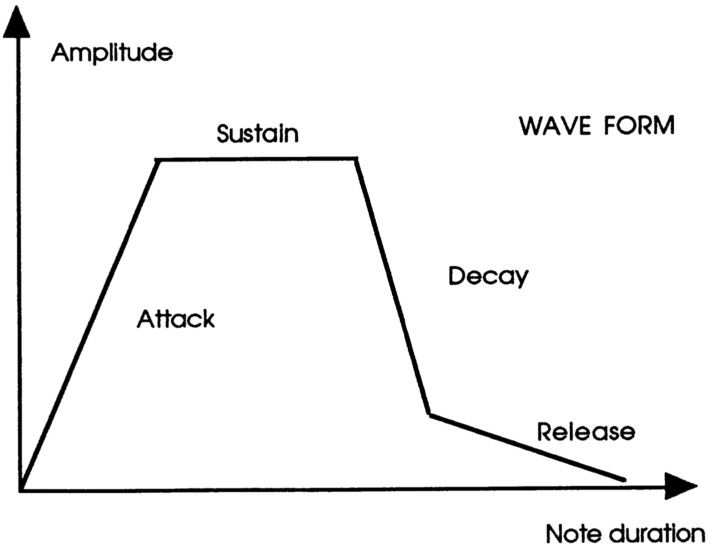
A fifth wave shape called *noise* exists, which is generated by a random sound. Noise is useful for sound effects, or for simply giving more consistency to a particular sound.

Other waveforms are based on these primary shapes. However, all the shapes can be broken down into sine waves. The illustration below shows how the four primary shapes compare:



1.3.2 ADSR

Four parameters with the initials *ADSR* (Attack, Decay, Sustain and Release) also affect sound. The ADSR factors determine the time required for the sound to reach its highest intensity (Attack), the time needed for the sound to return to its normal level (Decay), the duration of the sound at this level (Sustain) and its duration of release (Release). These four parameters, along with the wave shape, have the most influence on the timbre. The following illustration shows an example of ADSR at work and its effect on a sound:



1.3.3 Effects

A sophisticated synthesizer can offer special effects, such as *vibrato* (modulation in the sound producing a vibrating illusion) and *retardations*. When these retardations are very short they are called *chorus*; when they are longer, they're called delays or echoes, and are modified according to the time, modulation or phasing.

1.4 Digital sounds

Even though *analog sounds* can create an infinite number of different sounds, they cannot easily re-create all the available sounds in nature, such as the human voice.

These types of sounds can be created with *digital sounds*. The natural sounds mentioned above are created by reproducing a sound that was memorized beforehand, instead of by using oscillators. We won't discuss digitizing techniques in detail here, because digitizing can become extremely complicated. However, we will provide you with a brief definition of digitizing:

> The musician samples (captures) the natural sound from a source. This sample can be taken live using a microphone, or taken from a tape recording or compact disk.
>
> Special hardware (called a digitizer) and matching hardware samples the sound several thousand times a second (64,000 is a minimum). Each value obtained is then placed in the computer's memory as a series of zeros and ones. The digitized sound can then be played back through the computer's sound system or saved to disk for later recall. Also, this digitized sound (the series of zeros and ones) can be edited by the user through the software, changing the sound's qualities.

The Amiga can generate digital sound through programming, or through the digitizing methods described above. There is, however, a catch to all this. When you digitize a sound, you always lose a certain amount of information in relation to the original wave shape. This happens because you are moving the information from analog ("real world") sound to digital sound, and some data gets lost in the translation.

Electrical voltages can be positive or negative. During the digitizing process, the user specifies the two highest positive and negative values. These intervals are then cut into slices. We call this *resolution* or quantization.

On the Amiga, digitizing is performed 8 bits at a time, so there are 256 possible values. With natural sound the wave's shape can vary above and below two horizontal lines, so gaps can result in the digital sound.

Compact discs reproduce digitized music 16 bits at a time, which gives them a reproduction quality twice as good as that provided by the Amiga. However, Amiga owners should be happy with the high quality of sound from their machine. Other computers, with poor analog oscillators, offer some possibilities, but are much less musically exciting than the Amiga.

Chapter 2

Music Theory

Chapter 2 Music Theory

Since you bought this book to develop music on your Amiga, we assume that you already know some musical basics. If you played clarinet in high school orchestra, or took piano lessons during childhood, or played guitar in your own rock group once, you should have most of the musical elements you'll need for this book.

However, depending on your knowledge of music, you may need to review some basic terms. We'll cover some of the essentials in this chapter to help you in this review. If you need more intensive study in musical basics, check with your local bookstore for any texts on basic music studies.

2.1 Scales

A scale is a series of musical notes.

You may already know the basic note names used in the musical alphabet: A, B, C, D, E, F and G. You may also know the frequency of the concert pitch A-440, used for tuning by most professional orchestras: 440 Hz (cycles per second), which corresponds approximately to the dial tone of a telephone.

Perhaps you have also read about the frequencies of the other notes of the scale in the *Amiga System Programmer's Guide* or elsewhere. Let's go deeper into musical notation and scales.

2.1.1 The notes of the scale

We mentioned earlier that music uses seven basic note names. Below we have listed the notes used in the C Major scale (no sharps or flats in the key signature, and the base note or *tonic* of the scale is C). The notes listed here are notated in modern English notation, as well as the old *sol-fa* or *Latin* notation:

```
Sol-fa (Latin) notation: DO  RE  MI  FA  SOL  LA  TI
English notation       : C   D   E   F    G   A   B
```

2.1.2 Intervals

As the name implies, intervals represent the distance between two specific notes. Music is a complex form of applied mathematics, and intervals show just part of this application by dividing spacing between notes into numbers.

The largest named interval is called the *octave*. The top note of an octave is eight steps above the bottom note. For example, we listed the seven basic elements of the C Major scale in Section 2.1.1:

```
C   D   E   F    G   A   B
```

If we listed this scale with the octave of the bottom note of the scale (C), we'd get the following:

```
C   D   E   F    G   A   B   C
                             ^
```

The next note up in the C Major scale (the octave of C) is C.

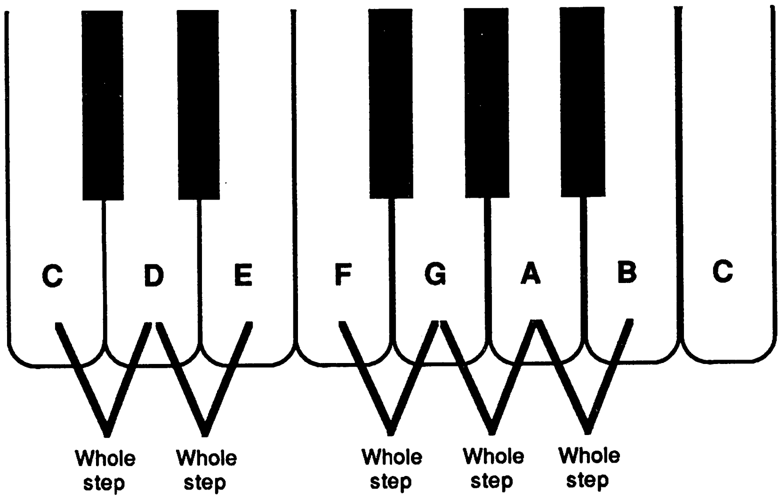
Let's look at this from a more mathematical angle. To determine the octave above a specific frequency, multiply the frequency of the note by two. To determine the octave below, divide the frequency by two.

We stated that A-440 is concert pitch, from which most orchestras take their tuning note before performing. The pitch A-440 vibrates at 440 cycles per second. If we multiply 440 Hz by two, we get the frequency of the octave above A: 880 Hz, which also represents a pitch named A. If we divide 880 Hz by 2, we get 440 Hz—the A at which we began. Now, if we divide 440 Hz by 2, we get 220 Hz—the A one octave below A-440.

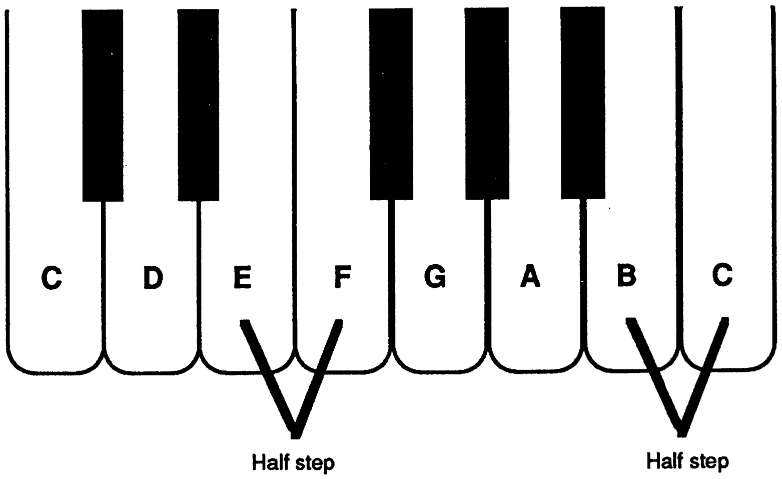
Some music development programs assign octave numbers to note names to avoid confusion (i.e., C1 is the bottom octave C, C2 is the C one octave above C1, C3 is the octave above C2, etc.).

Octaves are divided into *steps*. In the case of our C Major scale, these are the usual notes from C to C. There are two types of steps used in normal musical notation:

Whole step Also called *whole tones.* The following intervals in the C Major scale are whole steps (if you have an electronic keyboard or piano, you may want to try these out):



Half steps Also called *half tones.* The following intervals in the C Major scale are half steps (again, try these out on a keyboard instrument):



Add the five whole steps and the two half steps listed on the preceding page, and you'll get a total of six whole steps per octave:

```
1 + 1 + 1 + 1 + 1 + .5 + .5 = 6
```

Each whole step can be divided into nine intervals called *commas*. The *sharp* is five commas "above" the note and the *flat* is four commas "below" the note. Some instruments, such as the guitar and piano, only recognize half tones because of the way they are designed or tuned. Pianos are tuned in this way so that they can be played in any key and still sound equally in tune in any key. This is called *even temperament*. You can hear the differences between sharps and flats on instruments such as the trumpet or the violin. These instruments and their owners will play a sharp "brighter" than a flat. Conversely, a flat played on the same instrument will have a "duller" sound than a sharp. If you have ever wondered why blues are written with flats rather than sharps, now you know.

If you want to develop your own music program, the following ground rules will help you manage the intervals. They involve using the smallest interval, which is called the *savart*: The savart is defined as the smallest interval that can be detected by the human ear.

For two notes with respective frequencies F1 and F2, you can determine the number of savarts separating them using the following formula:

```
savarts = LOG10(F1/F2)*1000
```

If you don't want to calculate this by hand, any language used for programming the Amiga can perform this calculation. We used GFABasic.

To understand the octaves, you just need to calculate the number of savarts between A-440 and A-880.

```
LOG10(880/440) = 301.029995664
```

To simplify this, round off the result to 300. Unless your audience has a fine-tuned ear, they won't notice the difference between 300 and 301.029995664.

Since an octave is made up of six whole tones, there are 50 savarts between each tone and 50/9, or 5.55 savarts between each comma. (Let's call it an even 5.)

Remember the following two formulas, which will help you when you're calculating certain frequencies:

```
F1 = F2*10^(savarts/1000)
F2 = F1/10^(savarts/1000)
```

We've provided formulas instead of tables so that you can quickly compute exactly what pitches you want. However, remember that the formulas become invalid in extreme high and low musical ranges.

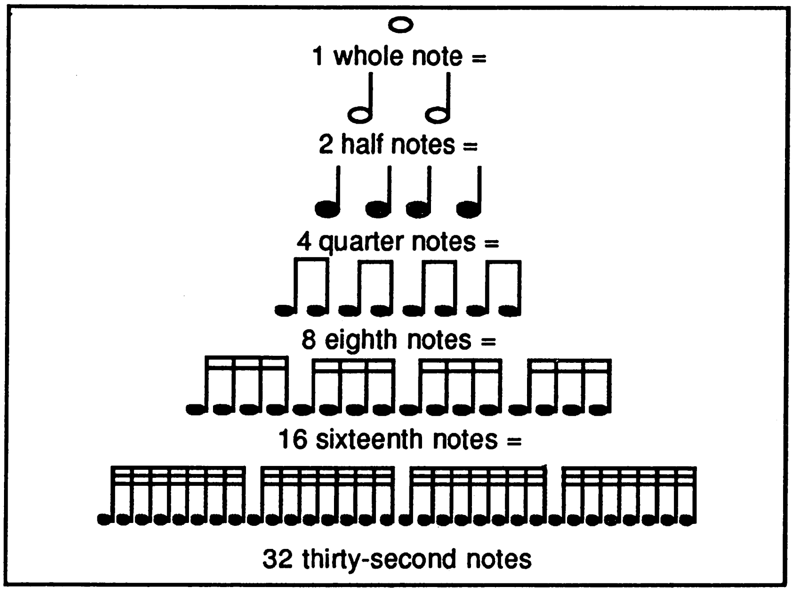
2.2 Rhythm

There are three elements of music: melody, harmony and rhythm. You can compose music without harmony, and even without melody, but not without some form of rhythm or timing. When you use samplers, sequencers, drum machines and, of course, an Amiga, rhythm becomes very important. So, we'll begin this section by discussing time.

2.2.1 Time

The largest unit in musical time notation is the *whole note*. This can be divided into two *half notes*. Each half note can be divided into two *quarter notes*, each of which can be further divided into two *eighth notes*. Then there are *sixteenth notes*, *thirty-second notes* and occasionally *sixty-fourth notes*. An external timing source or special software may be necessary in order to implement note values smaller than sixty-fourth notes.

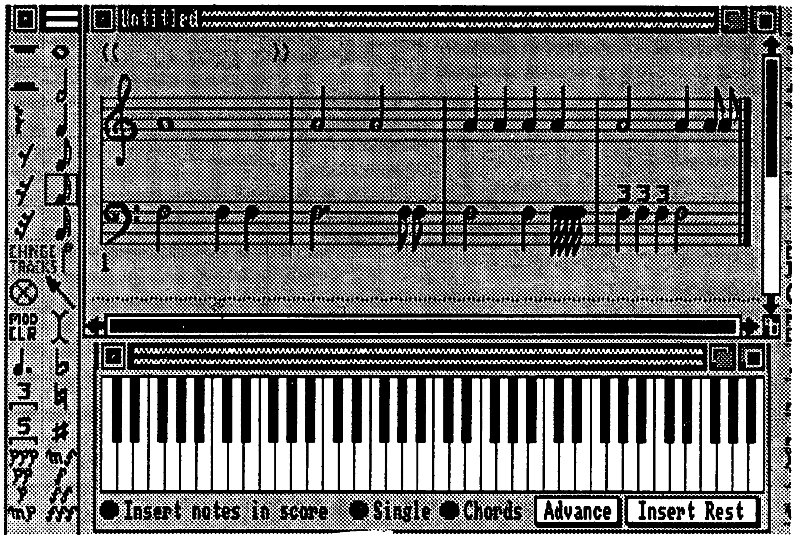
We mentioned in Section 2.1 that music is a complex form of applied math. We can see this even more clearly by viewing the division of equal amounts of note values. The following illustration should give you a clearer picture of this division, from whole note to thirty-second notes:



Adding a dot to the right of a note lengthens that note by one half its value. For example, if you add a dot to the right of a quarter note, the dot adds one half the quarter note's value (i.e., an eighth note) to the quarter note.

Triplets shift normal note values from duple (grouped in two) time to triple (grouped in three) time. Triplets accomplish in three notes what their normal counterparts perform in two notes.

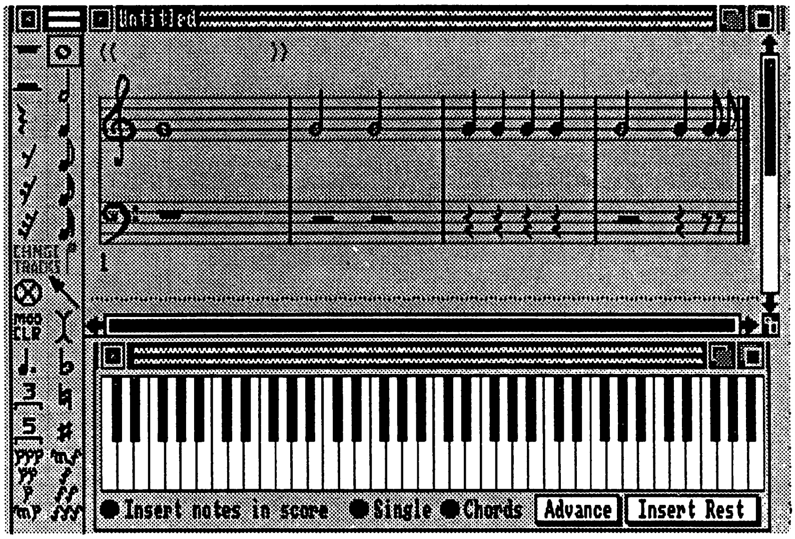
Here are some examples of dotted and triplet notes, as entered in the Deluxe Music Construction Set:



2.2.2 Rests

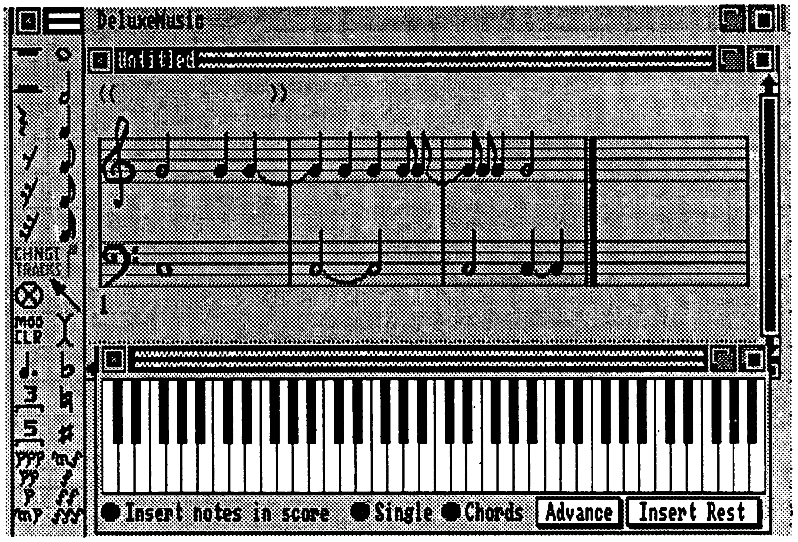
Rests represent silences in music, and can often be as important as the notes themselves. The *whole rest* corresponds to the whole note, the *half rest* to the half note and the *quarter rest* to the quarter note. The *eighth rest* has the same length as the eighth note. There are also *sixteenth rests*, *thirty-second rests*, etc.

Here are some rests as entered using the Deluxe Music Construction Set:



2.2.3 Ties

When two separate notes are joined by a *tie*, you play them as only one note (for example, two tied half notes equal a whole note).



2.2.4 Tempo

Most people call *tempo* the "beat", which has a much broader meaning than "tempo". Depending on the time signature, you select a reference value (e.g., the quarter note for binary time). The tempo is represented by the number of these values in a minute. The tempo is measured in beats per minute, which corresponds to the beating of a *metronome*.

Sheet music frequently lists a tempo in generic terms such as *Allegro*, *Moderato*, *Andante* or *Largo* (you can find these terms in a music dictionary). If the composer or editor of the sheet music had a specific tempo in mind, there may be a metronome listing printed to the top of the page, such as:

```
MM=100
```

This means that the basic unit of notation (e.g., the quarter note) should be counted at 100 beats per minute.

2.2.5 Binary meter

According to tradition, *binary meter* or *duple meter* is used in popular music (such as rock, rap or country). Binary meters are 2/4, 3/4, and the popular 4/4.

Let's look at the numbers we just listed. The first number (e.g., the 2 in 2/4) specifies the number of beats per measure, while the second number (the 4 in 2/4) indicates that the quarter note receives one beat. These numbers are called the *time signature*.

2.2.6 Ternary meters

Ternary meter or *triple meter* is often used in jazz. Common ternary meters are 6/8, 9/8 and 12/8. The unit is the dotted quarter note (equivalent to three eighth notes). So there are two in 6/8 time, three in 9/8 time and 4 in 12/8 time.

You may be wondering why we used the dotted quarter note as the unit of time. According to music theory, the unit of time is the eighth note. But for musicians, it's more practical to use the dotted quarter when the tempo is fast. For slow tempos, it is preferable to use the eighth note. In a musical score, the tempo is usually indicated by using the note that serves as a unit of time, followed by an equal sign and then the value.

2.2.7 Other meters

There are many other meters, such as 2/2 (two beats per measure, with the half note receiving one beat), 9/16, 12/4 and 33/16. Remember that these time signatures contain two important numbers: The number of beats per measure, and the type of note which receives one beat.

Some non-numerical time signatures also exist. The letter C indicates "common time" (4/4), and the letter ¢ indicates "cut time" (2/2).

2.2.8 Resolution

If you have ever programmed a drum machine or a sequencer, you may already be familiar with this term. The meter is divided into a certain number of intervals (32, for example). If you enter the instruments according to their real meter, they will "stick" to the nearest interval.

This is almost the same as the principle that we discussed with sound digitizing. The higher the resolution, the less mechanical the drum machine will sound (a slight irregularity makes the drum machine sound more "human"). However, you will have shorter pieces of music because of memory problems. Here again, resolution is present on the main musical software of the Amiga, especially those functioning as sequencers with MIDI interfaces.

2.3 Musical notation

Notes are placed on a *staff*, which consists of five horizontal bars. Staves contain various symbols which guide the musician through the playing of a piece of music.

However, these symbols can be different, depending on the instrument for which the music is intended. For example, a grand staff (two staves linked together vertically) is generally used for piano music. For the guitar, musical scores may indicate the chords with an oblique bar, which occasionally contains a Roman numeral that indicates the chord to be used. Roman numerals can also indicate artificial harmonics, glissandos, etc.

Of all the software currently available for the Amiga, none is able to utilize musical notation completely. Most of them only offer the simplest musical notation. This is understandable since these software products are only intended to guide synthesizers or manage the sound output of the Amiga. So, this software should be suitable for your needs.

Chapter 3

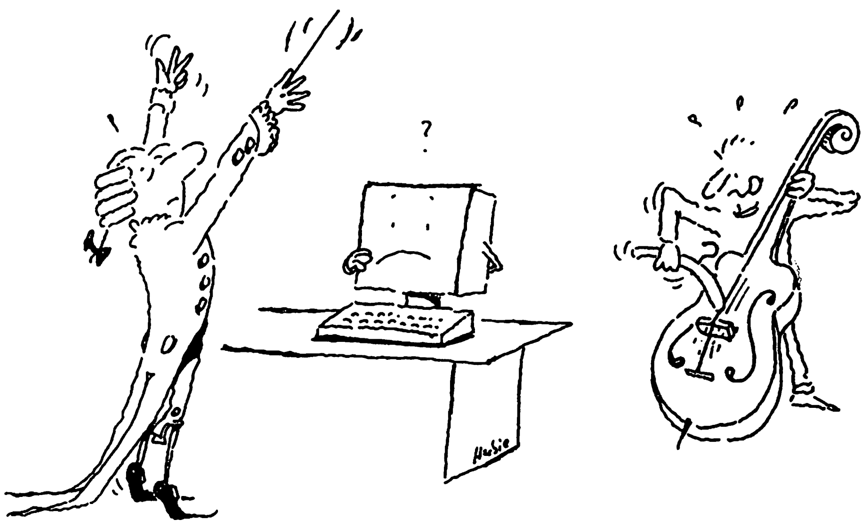
Programming Music

Chapter 3 Programming Music

The Amiga has great sound capabilities on its own. Like all computers, however, the Amiga must be told what to play and how to play it. That's your task—instructing the Amiga in playing music. This chapter describes how you can develop musical programs for playback on your Amiga.

3.1 Languages

Since there are so many programming languages available for the Amiga, we had a difficult time selecting the ones to use for this book. We decided to use only four languages: AmigaBASIC, GFABasic, the C programming language (available in different implementations) and assembly language. However, if you use other languages, such as MODULA-2 and Multi-Forth, and you have enough knowledge of the original language (BASIC, C or whatever) and your own language of choice, you should be able to adjust the examples accordingly.



3.2 AmigaBASIC

AmigaBASIC provides a lot of flexibility for sound management. It allows you to program tunes quickly and easily. Once you develop the musical code in AmigaBASIC, you can include music in most AmigaBASIC programs.

3.2.1 The SOUND command

Syntax :

```
SOUND frequency,duration,[volume][,channel]]

SOUND WAIT

SOUND RESUME
```

Let's look at the parameters used in the SOUND command one at a time:

Frequency

The *frequency* parameter specifies the pitch of a note. Its value can be between 20 Hz (the lowest limit of human hearing) and 20,000 Hz (the highest limit of human hearing). If you exceed these limits, AmigaBASIC automatically rounds off the frequency to the nearest appropriate value. Since these values represent the approximate range of human hearing, 20 Hz - 20,000 Hz should be enough for you, unless you want to communicate with whales or annoy your neighborhood dogs. Frequency is a mandatory parameter of the SOUND command.

Duration

The sound will continue for the time specified in *duration*. Even though the duration isn't expressed in seconds or BPM (Beats Per Minute), it's easy to determine. If you enter a value of 18.2 for duration, this sounds the note for exactly one second, or sets a tempo of 60 beats per minute. Values for duration can range from 0 to 77. This information may not be valid for other implementations of BASIC, or for some BASIC compilers. The duration parameter is also mandatory in the SOUND command.

Volume

The level of sound output can be regulated with the *volume* parameter. Values for volume can range from 0 (no sound) to 255 (LOUD). If you don't specify a value for volume, the AmigaBASIC interpreter defaults to a median value of 127. You won't have to change the value in the volume parameter unless you want to create effects like *crescendo* (gradual increase in volume) or *decrescendo* (gradual decrease in volume).

Channel

The last parameter, *channel*, specifies the voice of the Amiga that you have chosen to produce the sound. Since the Amiga has four voices, you can choose a number from 0 to 3. Channels 0 and 3 make up the voices on the left stereo channel, and channels 1 and 2 make up the voices on the right stereo channel. With AmigaBASIC's SOUND command, you cannot use one of the sound channels to modulate another channel. You may think that this limitation decreases AmigaBASIC's musical abilities. True, but one channel modulating another isn't used very often today.

The following example of the SOUND command plays the note A-440 for a duration of just over 1/2 second:

```
SOUND 440,10
```

In the following syntax, the channel is specified but the volume isn't:

```
SOUND 440,10,,1
```

Remember that the frequency and duration parameters are mandatory.

It's also possible to express the volume without specifying the channel:

```
SOUND 440,10,255
```

For the moment, we won't show the syntax with all four parameters (you'll find examples of them later).

The *SOUND WAIT* command lets you create a *sound wait list*. This means that, as the interpreter encounters a SOUND command, it will place the command in the wait list instead of executing it. When it receives the *SOUND RESUME* command, the interpreter executes all the sounds in the order in which they were entered in the wait list. You may have to keep a sound in memory when you want to combine graphics and sound.

3.2.2 The WAVE command

Syntax:

```
WAVE channel,table

WAVE channel,SIN
```

This command controls the waveform used in musical sound.

With the first syntax, WAVE channel,table, you can define the shape of your wave by setting up a table with at least 256 elements, which are signed (between -128 and +127) bytes (the sound is encoded on 8 bits). The most common ways of completing this table are by DATA lines, reading from a file or generating it with a function (generally a sine wave).

The second syntax, WAVE channel,SIN, generates a sine wave in a specific channel.

Since the WAVE command determines the timbre of your sound, and consequently, its quality, it is an important command. Perhaps you're wondering whether it's possible to generate a wave shape from an IFF 8SVX file (one with a sound coming from a digitizer) with the WAVE command. It can be done, but the result cannot be used. The WAVE command only accepts the first 256 bytes of information, whereas the IFF 8SVX file requires much more information. So, this method isn't useful since acceptable digital sound requires a minimum of several tens of thousands of bytes. We'll discuss other ways to load IFF sounds later.

3.2.3 Example programs

We wrote these examples to give you some general tools for music programming in AmigaBASIC. The following six programs demonstrate what you need to know in order to experiment with the sounds in AmigaBASIC.

Example 1: Pseudo-Classical Music

This example provides the frequency information in DATA statements. The main advantage to DATA lies in lower execution time. The notes themselves are selected by the program at random.

Since all the notes come from the natural Major scale, the "piece" that results won't always sound melodious, but then again, it won't sound too discordant. The sound intensity is also modulated at random, which adds a little more realism to the illusion of a classical piece.

Enter RUN in AmigaBASIC to run the program, and press <Ctrl><C> to stop the music.

```
RANDOMIZE TIMER
REM Initializing the variables
REM RNDTIM1.BAS
DIM notes(13)
FOR i=0 TO 12
READ notes(i)
NEXT i
Loop:
FOR i=0 TO 3
note = notes(INT(RND*12))
SOUND note,10,INT(RND*127),i
NEXT i
GOTO Loop
DATA 130.81,146.83,164.81,174.61
DATA 196.0,220.0,246.94,261.63,293.66
DATA 329.63,349.23,392.0,440.0
```

Example 2: Randomizing and multitasking

AmigaBASIC is capable of multitasking, except when executing SOUND commands. Here's our solution to the problem:

```
RANDOMIZE TIMER
REM initializing  the variables
REM RNDTIM2.BAS
DIM tone%(255)
FOR i=0 TO 255
```

```
tone%(i)=INT(RND*255)-128
NEXT i
FOR i=0 TO 3
WAVE i,tone%
NEXT i
SOUND WAIT
LINE (10,10)-(20,20)
FOR i=0 TO 3
FOR j=0 TO 5
LINE (50*j,20*i)-(60*i,40*j),,B
SOUND INT(RND*1000),INT(RND*10),INT(RND*127),i
NEXT j
NEXT i
SOUND RESUME
PRINT "Go!"
PRINT "Required - Include SOUND WAIT"
PRINT  "and SOUND RESUME"

FOR i =1 TO 10
PRINT
PRINT "This is an example of multi-tasking with sound"
PRINT
NEXT i
```

In this program, we demonstrated how to generate a random wave shape by using the WAVE command, and how to use the SOUND WAIT and SOUND RESUME commands. We stated earlier that sound data can be placed in the wait list for later execution, and this program shows how it's done.

The LINE commands are used so that you won't have to wait a long time. The PRINT commands demonstrate that it is possible to combine sound and graphics by using AmigaBASIC. However, you should use the SOUND WAIT command with caution, as it will not accept thousands of notes. If you exceed SOUND WAIT's limit, the only response the program may give you is an error message.

Example 3: Generating a wave shape

As we mentioned previously, you can generate a wave shape using calculations. This is the most effective method since IFF sounds cannot be loaded into BASIC, and DATA can only be used to load a precalculated waveform rapidly. So we have defined a function called FNTone which takes sine wave values. We have displayed this wave form by opening a window and a screen in BASIC.

```
RANDOMIZE TIMER
REM Initialization of the variables
REM
OPTION BASE 0
DIM Tone%(255)
PI=3.1415926536#
DEF FNTone(x)=INT(SIN(x*PI*4/255)*85)
SCREEN 1,640,220,1,2
WINDOW 1,"Tone",(1,1)-(600,185),0,1
Tone%(0)=FNTone(0)
FOR i=1 TO 255
Tone%(i)=FNTone(i)
LINE ((i-1),Tone%(i-1)+85)-(i,Tone%(i)+88)
NEXT i
FOR i=0 TO 3
WAVE i,Tone%
NEXT i
FOR i=0 TO 3
FOR j=0 TO 5
SOUND 55*j,10,127,i
NEXT j
NEXT i
PRINT "Press any key ..."
WHILE INKEY$=""
WEND
WINDOW CLOSE 1
SCREEN CLOSE 1
END
```

We suggest that you experiment with the FNTone function to see what variations can be created. Most analog synthesizer users do this to create various sounds. First try changing the PI multiplier (we had set it to 4 in our example). Although this method increases your calculation time, once you find an acceptable sound, you can enter the values through DATA statements.

To create other effects, apply, for example, a certain function to the elements from 0 to 127 and another one up to element 255. If you want to push this language to its maximum sound capabilities, enter analog functions for the frequency and volume. Some users find that this method is more practical than using a graphic wave shape editor.

Example 4: The scale

Now we'll demonstrate how you can use the formulas for calculating a succession of notes, which we presented earlier.

Beginning with a note as base, calculate 24 successive half steps corresponding to 2 octaves. Perform the calculation using real time.

The SOUND WAIT and SOUND RESUME commands don't always have to be used with AmigaBASIC. An interesting application for this example is transposing a piece of music. By changing the base note, you change the tonality of the scale and the calculation of all the notes between frequencies of 20 Hz and 20,000 Hz.

```
REM SCALE.BAS
DEF FNFlat(f)=f/10^(25/1000)
DEF FNSharp(f)=f*10^(25/1000)
WAVE 0,SIN
note#=220
FOR i=1 TO 24
SOUND note#,5
note#=FNSharp(note#)
NEXT i
FOR i=1 TO 24
SOUND note#,10
note#=FNFlat(note#)
NEXT i
```

Example 5: Harmony at last!

The following functions enable you to create chords easily. We mentioned some intervals earlier in this book (octaves, whole steps and half steps). Triads (three voice chords) are comprised of the tonic or root note, a third above the tonic and a fifth above the tonic.

Although you may have heard of thirds and fifths, you might not know that their frequencies are respectively 5/4 and 3/2 of the tonic. To get the fourth, multiply the tonic by 4/3 and to get an octave multiply by 2. So, instead of using a piano or calculating by hand to find these chords, you can use these functions.

```
REM HARMONY.BAS
DEF FNThird(f)=f*5/4
DEF FNFifth(f)=f*3/2
DEF FNSeventh(f)=f*10^(50/1000)
WAVE 0,SIN
WAVE 1,SIN
WAVE 2,SIN
note#=220
FOR i=1 TO 20
SOUND WAIT
SOUND note#,20,127,0
SOUND FNThird(note#),15,127,1
SOUND FNFifth(note#),10,127,2
SOUND RESUME
note#=FNSeventh(note#)
NEXT i
```

As you can see, you can easily transpose a piece of music by simply changing the tonic note.

Example 6: What about history?

Lulli (also spelled "Lully") was a court musician for France's King Louis XIV. His real fame came from a piece of popular music he composed: "Au clair de la lune". In this example, we'll use another piece of music composed by Lulli.

```
REM LULLI.BAS
PI=3.1415926536#
DEF FNSharp(f) = f*10^(25/1000)
DEF FNTone0%(x) = INT(SIN(x*PI/255)*127)
DEF FNTone1%(x) = INT(SIN(x*PI/255+PI/4)*127)
DEF FNTone2%(x) = INT(SIN(x*PI/255+PI/2)*127)
DEF FNTone3%(x) = INT(SIN(x*PI/255+3*PI/4)*127)
DIM Tone0%(255),Tone1%(255)
DIM Tone2%(255),Tone3%(255)
DIM Scale(36),Notes%(100),Duration%(100),Channel%(100)
PRINT "Initialization of the scale"
Scale(0)=220
FOR i=1 TO 35
Scale(i) = FNSharp(Scale(i-1))
NEXT i
PRINT "Initialization of the wave shapes"
FOR i=0 TO 255
Tone0%(i)=FNTone0%(i)
Tone1%(i)=FNTone1%(i)
Tone2%(i)=FNTone2%(i)
Tone3%(i)=FNTone3%(i)
NEXT i
WAVE 0,Tone0%
WAVE 1,Tone1%
WAVE 2,Tone2%
WAVE 3,Tone3%
PRINT "Reading the Music ..."
FOR i=0 TO 24
READ Notes%(i),Duration%(i),Channel%(i)
NEXT i
PRINT "Hit It, Maestro ... "
FOR i=0 TO 27
SOUND Scale (Notes%(i)),Duration%(i),255,Channel%(i)
NEXT i
DATA 0,20,0
DATA 12,20,1
DATA 4,20,2
DATA 7,20,3
DATA 4,10,0
DATA 16,10,1
DATA 4,10,0
DATA 16,10,1
DATA 2,10,0
DATA 14,10,1
DATA 4,10,0
DATA 16,10,1
DATA 2,10,0
DATA 14,10,1
DATA 0,20,0
DATA 12,20,1
DATA 4,20,2
DATA 7,20,3
DATA 4,10,0
DATA 16,10,1
DATA 4,10,0
DATA 16,10,1
DATA 2,10,0
DATA 14,10,1
DATA 4,20,0
DATA 16,20,1
DATA 6,20,0
DATA 18,20,1
END
```

This music program summarizes everything we've discussed so far. The notes of the scale, as well as the wave shapes, are automatically calculated. To create your own music, modify the timbre, the base note and the DATA statements.

This concludes our discussion of AmigaBASIC and its musical commands. Now let's move on to the sound capabilities of other languages.

3.3 Hardware and audio.device programming

For better control of the sound output, you must use the audio.device or hardware, and sometimes a combination of both the audio.device and hardware.

The audio.library lets you transmit any waveform through any Amiga sound channel, at any volume level and of any duration. Since there aren't many audio library source codes, the routines used for managing audio are very complicated. Between 1986 and 1988, it was very difficult to find information about this type of programming. There still isn't an actual audio library available, but some software tools have been introduced.

Programs that use hardware are easier to control, which gives them an advantage over audio.device tools. However, programs that use the audio.device are more reliable in a multitasking environment.

GFABasic is very effective in hardware programming because of its ABSOLUTE command. This command enables you to access a segment of the memory as if the memory were a variable. This is helpful when working with DMA registers.

We usually recommend that you avoid hardware programming because it's impossible to know whether all the programs in a multitasking environment require the same resources. Because of this, we used devices on the Amiga. Also, since programming methods are very similar among various devices, devices are more practical to use.

Audio hardware is an exception to this rule because, while programming the device, you won't have access to all the sound functions. You may notice this if you work with the demos packaged with other sound programs. However, you should avoid using these programs while you're experimenting. Programming the audio.device is a much safer method.

After reading this section, you can find more information about the audio.device in the *Advanced System Programmer's Guide for the Amiga*, available from Abacus.

3.3.1 The audio hardware of the Amiga

We mentioned earlier that the Amiga has four audio channels, which are controlled by the *Paula circuit.* Channels 0 and 3 are connected to the left sound out on the Amiga (white) and channels 1 and 2 are connected to the right sound out (red for right). The audio channels are tied into Paula by a *DMA circuit* (Direct Memory Access), which guarantees speed.

Of course it's possible to control these channels directly with the 68000 processor but this is very slow and not as efficient as using the Paula circuit. However, we don't recommend trying this method because no documentation exists at this writing. Also, if you know something about the Copper, you may have realized that it's possible to generate Copper lists for programming sound circuits. Although this isn't a bad idea, there's not much information on this either.

Now let's return to the DMA accesses. Sounds are stored in the memory, then sent through channels to the digital/analog transformer. The role of the digital/analog transformer is simply to generate an audio signal from bytes, which defines the wave shape. The signal is then sent towards the sound output. If you've already read Abacus' *Amiga System Programmer's Guide*, you should be familiar with this process.

When talking about DMA access, you must also discuss *CHIP memory* because this is where the process occurs. There must be an even number of bytes because the DMA transfers data in groups of two bytes.

You must give an address to the data containing the wave shapes. However, this address must be located in the CHIP memory (DMA memory addressable). Each audio channel has a DMA register into which you will place this address. The following are the registers we're talking about: *AUDxLCH* and *AUDxLCL* (with x having values from 0 to 3). The registers *AUDxLEN* determine the size of your wave shape. Since the DMA only deals with word transfer (two bytes), you must initialize these registers with the number of bytes divided by two.

Remember that the hardware will only allow you to specify wave shape lengths of less than 131,070 bytes (65,535 words). You'll have to re-program any length that goes beyond this limit.

The sound volume is regulated by the *AUDxVOL* registers, which accept values from 1 to 64. The most complicated registers to program are the *AUDxPER registers*, which indicate the sound frequency. We developed a function in GFABasic that will simplify this task. The following is a review of these registers, as well as their addresses. The base address of DMA circuits is $DFF000:

AUD0LCH	$A0	AUD1LCH	$B0
AUD0LCL	$A2	AUD1LCL	$B2
AUD0LEN	$A4	AUD1LEN	$B4
AUD0PER	$A6	AUD1PER	$B6
AUD0VOL	$A8	AUD1VOL	$B8
AUD2LCH	$C0	AUD3LCH	$D0
AUD2LCL	$C2	AUD3LCL	$D2
AUD2LEN	$C4	AUD3LEN	$D4
AUD2PER	$C6	AUD3PER	$D6
AUD2VOL	$C8	AUD3VOL	$D8

There are two ways to calculate the values in the *AUDxPER* registers. You may have defined the wave shape yourself (for example, using a sine wave function) or you may want to reproduce a sample sound. In the first instance, you must find the sampling frequency by using the following formula:

```
sampling_frequency = frequency * number_of_samples
```

In the second instance, you either must know the sampling frequency that you have used (all samplers give this frequency) or give it an arbitrary value (i.e., Perfect Sound uses 10,000 Hz). However, you may have to change this later if it isn't suitable.

Once you know this frequency, you have to use a second formula to determine the value to insert in AUDxPER. This is necessary because the DMA controller uses *bus cycles* instead of Hertz. Bus cycles determine the timing of the CHIP RAM, and depend on the *Amiga clock* for information.

On an *NTSC system*, the duration of a bus cycle is 1/3579545 second, and on a *PAL system* it is 1/3546895 second. You can determine which system you are on by consulting the DisplayFlags field of GfxBase.

This field will contain 1 for NTSC systems and 4 for PAL systems. In GFABasic, use the following:

```
pal! = INT{_GfxBase+206} AND 4

IF pal!
     clock%=3546895
ELSE
     clock%=3579545
ENDIF
```

The value to enter in AUDxPER is relative to the clock frequency. To determine this value, use the following formula:

```
period& = INT(clock%/frequency%)
```

This formula can easily be placed in a function.

The *ADKCON* and *DMACON* registers are also important when you're programming hardware. The DMACON register controls all the DMA accesses. For audio DMA accesses, only the first 4 bits of the DMACON register are important. These activate or deactivate each sound channel. Bit 0 corresponds to channel 0, etc. with bit 3 controlling channel 3. You'll also need bit 9 to activate or deactivate the series of DMA channels and bit 15, which activates or deactivates specific bits (0 erases and 1 activates).

For example, let's activate channel 0. Start by denying access to DNA and by deactivating the audio channels by entering the value $F (0000 0000 0000 1111) in DMACON. Since bit 9 is deactivated, it momentarily denies all access to DMA. Bit 15 at 0 indicates that you must erase the four sound channels without being concerned with the other DMA channels. Next, initialize ADKCON and the audio registers that we have discussed. Finally, reactivate the DMA by using the value $8201 (1000 0010 0000 0001), which will re-establish the DMA authorizations and activate audio channel 0.

The ADKCON register indicates whether one or several audio channels should act as *modulators*. It also acts as the disk controller. Bit 15 has the same use here as the one in DMACON because it indicates whether specified bits should be activated or erased. Bits 0 to 7 specify the use of each audio channel:

```
bit 0 :Modulation of channel 1 volume by audio channel 0
bit 1 :Modulation of channel 2 volume by audio channel 1
bit 2 :Modulation of channel 3 volume by audio channel 2
bit 3 :Audio channel 3 is cut off
bit 4 :Modulation of channel 1 period by audio channel 0
bit 5 :Modulation of channel 2 period by audio channel 1
bit 6 :Modulation of channel 3 period by audio channel 2
bit 7 :Audio channel 3 is cut off
```

To ensure that no channel will modulate another one, place the value $FF in ADKCON (0000 0000 1111 1111). This will erase bits 0 to 7.

Example of programming in GFABasic:

```
 Address of the DMA base SOUNDMA.GFA
dmabase%=&HDFF000
' Special registers
ABSOLUTE dmacon&,dmabase%+&H96
ABSOLUTE intena&,dmabase%+&H9A
ABSOLUTE adkcon&,dmabase+&H9E
' Registers of audio channel  0
ABSOLUTE aud0lc%,dmabase%+&HA0
ABSOLUTE aud0len&,dmabase%+&HA4
ABSOLUTE aud0per&,dmabase%+&HA6
ABSOLUTE aud0vol&,dmabase%+&HA8
'
clock%=@getclock                   ! PAL or NTSC ?
v_length%=64          ! One value larger or result may be
'                     ! invalid for aud0per < 124
audioliste%=MALLOC(v_length%,2)    ! Allocation of
                                   ' CHIP memory
IF audioliste%=0
  ALERT 0,"Impossible to play note| ",1,"Why ?|Quit",a%
  IF a%=1
    ERROR 103                 ! Error - memory insufficient
  ELSE
    END                                ! End
  ENDIF
```

```
ENDIF
FOR i=0 TO v_length%-1
  POKE audioliste%+i,@timbre(i)  ! Generate wave form
NEXT i
dmacon&=&HF                      ! control access to DMA
aud0lc%=audioliste%              ! much better than  a
POKE command
aud0len&=v_length% DIV 2
aud0vol&=64                      ! PLAY LOUD
aud0per&=@freq(440,v_length%)    ! store length to
'                                ! utilize the sample
adkcon&=&HFF                     ! modulation possible
dmacon&=WORD(&H8201)             ! Authorize DMA access
PRINT "Press mouse button to end"
REPEAT                           ! Attention
UNTIL MOUSEK
dmacon&=1                        ! Deactivate channel 0
~MFREE(audioliste%,v_length%)    ! Free memory
SYSTEM
'
FUNCTION timbre(i)               ! Sine wave form
  RETURN BYTE(SIN(i*PI/v_length%)*127)
ENDFUNC
FUNCTION freq(f,l)               ! Formula to calculate
AUDxPER
  RETURN INT(clock%/(f*l))
ENDFUNC
FUNCTION getclock                ! Determine frequency
  IF WORD{_GfxBase+206} AND 4
    RETURN 3546895
  ELSE
    RETURN 3579545
  ENDIF
ENDFUNC
```

With the Amiga it's possible to use one sound channel to modulate another by changing the values of the ADKCON register. You may choose to modulate either the frequency or the intensity or both simultaneously. This will allow you to create various sound effects.

In the following example, modulation occurs according to a choice that is given to you at the beginning of the program (Intensity, Frequency, both or neither). Channel 0 modulates channel 1. We also could have chosen other channels but the method is the same since only the registers change. Instead of containing information about the wave shape, the AUD0LC register contains data for modulation. This information will vary depending on the selected modulation.

Let's begin with *intensity modulation.* With each time interval, defined by AUD0PER, a value read in the data base and marked off by AUD0LEN will modify the intensity of channel 1. Of course, these values must be between 0 and 64.

The easiest sound effect to create is vibrato. You could also define an ADSR shape by specifying suitable values but this method really hasn't been thoroughly examined yet.

By making a diagram, you can see that the values of AUD0LEN control AUD1VOL. These first 4 bits of ADKCON allow one channel to modulate another. The AUD0LEN, AUD0LC and AUD0PER registers have to be initialized correctly. This means that, respectively, they will contain the length of the data for intensity modulation, the address from the data list that must be in CHIP memory, and the frequency of the modulation, which is actually the moment at which data coming from the list will modify the volume of the modulated channel.

With regard to frequency modulation, you could say that the values of AUD0LEN control AUD1PER. However, this is only correct if AUD1PER and AUD0PER have the same values (meaning that the two waves are synchronized). You'll create the most interesting, and, at the same time, the most complex effects when AUD1PER and AUD0PER have different values.

For frequency modulation, activate bits 4 to 7 on the ADKCON register. The AUD0LEN, AUD0LC and AUD0PER registers have to be initialized according to the same principles that are used for volume modulation.

These two types of modulation already offer an infinite number of combinations for producing sounds. But since this still might not be enough, you could use both simultaneously, which will provide an infinite number of possible combinations.

We prefer to give you functions that help you calculate the values of modulation (recall what we did for note intervals). However, these values must be simple enough so that you can modify the principal parameters. To initialize the list of values, alternate the information for each of the two modulations, first the intensity and then the frequency, and position the two corresponding ADKCON bits (according to our example these are bits 0 to 5). Once again you'll have to initialize AUD0LEN, AUD0LC and AUD0PER.

Now let's practice. The following is a program that allows you to decide on the kind of modulation you want:

```
' Address of the DMA base MODULATION.GFA
dmabase%=&HDFF000
' Special registers
ABSOLUTE dmaconr&,dmabase%+&H2
ABSOLUTE adconr&,dmabase%+&H10
ABSOLUTE dmacon&,dmabase%+&H96
ABSOLUTE intena&,dmabase%+&H9A
ABSOLUTE adkcon&,dmabase%+&H9E
' Registers of audio channel 0
ABSOLUTE aud0lc%,dmabase%+&HA0
ABSOLUTE aud0len&,dmabase%+&HA4
ABSOLUTE aud0per&,dmabase%+&HA6
ABSOLUTE aud0vol&,dmabase%+&HA8
' Registers of audio channel 1
ABSOLUTE aud1lc%,dmabase%+&HB0
ABSOLUTE aud1len&,dmabase%+&HB4
ABSOLUTE aud1per&,dmabase%+&HB6
ABSOLUTE aud1vol&,dmabase%+&HB8
'
clock%=@getclock                  ! PAL or NTSC ?
v_length%=64          ! One value larger or result may be
'                              ! invalid for aud0per < 124
longadsr%=80                    ! length of the ADSR data
audioliste%=MALLOC(v_length%,2)
' Allocation of CHIP memory
```

```
IF audioliste%<>0
  adsrliste%=MALLOC(longadsr%,2)
ENDIF
IF adsrliste%=0
  VOID MFREE(audioliste%,v_length%)
ENDIF
' The length of the wave form is stored in  oldlen%
'
oldlen%=longadsr%             ! adsrliste% will be free !
IF (audioliste%=0 OR adsrliste%=0)
  ALERT 0,"Impossible to play note|",1,"Why ?|Quit",a%
  IF a%=1
    ERROR 103                 ! Error insufficient memory
  ELSE
    END                            ! End
  ENDIF
ENDIF
FOR i=0 TO v_length%-1
  POKE audioliste%+i,@timbre(i)  ! Generate wave form
NEXT i
ALERT 0,"Change the modulation ?",1,"Yes|No",a%
IF a%=1
  ALERT 0,"What type ?",1,"Intensity|Frequency|Both",b%
  channel&=WORD(&H8203) ! the channel 0
  SELECT b%
  CASE 1
    choice&=WORD(&H8001)
    j=0
    FOR i=0 TO longadsr%/2-1
      READ mot&
      IF EVEN(i)
        DPOKE adsrliste%+j,mot&
        ADD j,2
      ENDIF
    NEXT i
    longadsr%=j
  CASE 2
    choice&=WORD(&H8010)
    j=0
    FOR i=0 TO longadsr%/2-1
```

```
        READ mot&
        IF ODD(i)
          DPOKE adsrliste%+j,mot&
          ADD j,2
        ENDIF
      NEXT i
      longadsr%=j
    CASE 3
      choice&=WORD(&H8011)
      FOR i=0 TO longadsr%/2-1
        READ mot&
        DPOKE adsrliste%+2*i,mot&
      NEXT i
    ENDSELECT
ELSE
  choice&=WORD(&H8000)
  channel&=WORD(&H8202) ! Channel 1 is authorized
ENDIF
dmacon&=&HF                         ! limit DMA access
aud0lc%=adsrliste%        ! Channel 0 waits for modulation
aud0len&=longadsr% DIV 2
aud0per&=8000
'
aud1lc%=audioliste%
aud1len&=v_length% DIV 2
aud1vol&=64                         ! PLAY LOUD
aud1per&=@freq(440,v_length%)       ! Store length to use
'                                   ! utilize the sample
adkcon&=&HFF                        ! modulation possible
adkcon&=choice&                      ! Type of modulation
and loudness
dmacon&=channel&! Authorize  access to DMA chans 0 and 1
' ATTENTION : Channel one is authorized
PRINT "Press a mouse button to end"
REPEAT                              ! Attention
UNTIL MOUSEK
dmacon&=3                      ! Deactivate channel 0 and 1
~MFREE(audioliste%,v_length%)       ! Free memory
~MFREE(adsrliste%,oldlen%)          ! adsrliste% is free !
SYSTEM
```

```
'
FUNCTION timbre(i)                ! Sine wave form
  RETURN BYTE(SIN(i*PI/v_length%)*127)
ENDFUNC
FUNCTION freq(f,l)          ! Formula to calculate AUDxPER
  RETURN INT(clock%/(f*l))
ENDFUNC
FUNCTION getclock                 ! Get frequency
  IF WORD{_GfxBase+206} AND 4
    RETURN 3546895
  ELSE
    RETURN 3579545
  ENDIF
ENDFUNC
' Data for the modulation : intensity,frequency
'
DATA 0,508,2,513,4,518,16,522,18,524,22,525,24,524,34,
522,48,518,56,513
DATA 64,508,56,503,48,498,34,494,24,492,22,491,18,492,
16,494,4,498,2,503
```

3.3.2 The audio.device: the device that makes itself heard

Directly accessing the hardware, even if it's interesting, isn't very practical in a multitasking environment. Because of this, there is an *audio.device*. Devices comprise the operating level immediately above the hardware level. These devices enforce the rules common to multiple tasks.

When you want to produce a sound, it's possible that another application has also decided to produce a sound at the same time. The sound may be generated by a word processor trying to indicate that it couldn't print the entire document, a hard disk autosave program that has finished its work, or an alarm that you programmed to tell you when it's time to shut down the Amiga and go watch a favorite television show.

Since the four audio channels are available, proper audio.device programming could enable you to divide these channels according to the needs of each application.

There may also be other rules. For example, one of your programs might need two stereo channels. Or suppose this program wants to keep channel 2 entirely separate from the other channels, for management with hardware by DMA or in direct access. These are channel allocations that may differ from one program to another. A word processor, for example, won't stop functioning if the audio.device doesn't allow it to use the channels that it specified. However, a music program can choose between stopping immediately or waiting for a free channel.

Since two programs can easily share the same sound channel, the computer doesn't simply stop. A system of priorities, managed by the audio.device, allows you to play a note of top priority before the other functions, which must wait because they have lower priorities.

IOAudio: input/output block

Before continuing, we should discuss this block's structure. The *input/output block* is made up of an *IORequest* structure, then is spread out with the necessary fields to the specific programming of the audio:

```
IOAudio + 0 : ioa_Request
                ioa_Request + 0 : io_Message
                          io_Message +  0 : mn_Node
                               0 : ln_Succ
                               4 : ln_Pred
                               8 : ln_Type
                               9 : ln_Pri
                               10 : ln_Name
                          io_Message + 14 : Reply_Port
                          io_Message + 18 : mn_Length
                ioa_Request +20 : io_Device
                          Address of the device
                ioa_Request +24 : io_Unit
                ioa_Request +28 : io_Command
                ioa_Request +30 : io_Flags
                ioa_Request +31 : io_Error
IOAudio +32 :   ioa_AllocKey
IOAudio +34 :   ioa_Data
IOAudio +38 :   ioa_Length
IOAudio +42 :   ioa_Period
IOAudio +44 :    ioa_Volume
IOAudio +46 :    ioa_Cycles
IOAudio +48 :    ioa_WriteMsg
                 ioa_WriteMsg + 0 : mn_Node
                                0 : ln_Succ
IOAudio +52                     4 : ln_Pred
IOAudio +56                     8 : ln_Type
IOAudio +57                     9 : ln_Pri
IOAudio +58                     10 : ln_Name
IOAudio +62      ioa_WriteMsg +14 : mn_ReplyPort
IOAudio +66      ioa_WriteMsg +18 : mn_Length
```

The total length of this structure is 68 bytes.

Before using the audio.device, you must reserve the necessary space for that structure. This means that you must choose between memory allocation and static structure.

With dynamic allocation, you can have as many of these structures as you need at a given time. However, this process requires much programming effort.

The advantage of static allocations is speed. You should use these when you don't want to allocate and free such structures repeatedly.

Since programming the audio.device is rather complicated when compared to other devices, you should familiarize yourself with the others before using this one. For more information, refer to the *Advanced System Programmer's Guide for the Amiga*, available from Abacus.

Audio.device commands

CMD_RESET	:	1	ADCMD_FREE	:	9
CMD_READ	:	2	ADCMD_SETPREC	:	10
CMD_WRITE	:	3	ADCMD_FINISH	:	11
CMD_STOP	:	6	ADCMD_PERVOL	:	12
CMD_START	:	7	ADCMD_LOCK	:	13
CMD_FLUSH	:	8	ADCMD_WAITCYCLE	:	14
			ADCMD_ALLOCATE	:	32

Note: We'll give you these values so you can use the audio.device with the GFABasic. If you program in the C language, use the file "device/audio.h". If you program in assembly language, use "device/audio.i.". This will give you a greater sense of security in case future versions of the system contain new standard commands.

In order to use one of these commands, you must place the command in the io_Command field of the IOAudio structure.

Flags of the audio.device

```
ADIOF_PERVOL       :  16    ADIOF_NOWAIT        :  64
ADIOF_SYNCCYCLE     :  32   ADIOF_WRITEMESSAGE : 128
```

These flags must be placed in the io_Flags field. If you want to include several of them, you must add them together, preferably by using an OR.

Error messages encountered while using the audio.device

```
ADIOERR_NOALLOCATION  : -10
ADIOERR_ALLOCFAILED   : -11
ADIOERR_CHANNELSTOLEN : -12
```

The audio.device will send you these messages in the io_Error field if there is a problem.

Before using the audio.device

1. *Saving memory for the IOAudio structure*

In GFABasic you would do this as follows:

```
ioaudioptr% = MALLOC(68,$ 10000) !MEMF_CLEAR
```

Be absolutely sure that you have properly performed the allocation (ioaudioptr% 0). If you want to use several blocks you may want to allocate all of them simultaneously:

```
nb_blocks|=3
ioaudioptr%=MALLOC(68*nb_blocks|,$ 10000)
```

2. *Audio.device opening*

This part is necessary so that the audio.device will know that you are going to use ioaudioptr%. The address of the device will be written in the io_Device field of ioaudioptr% (ioaudioptr%+20).

```
audioname$="audio.device"+CHR$(0)
error%=OpenDevice(V:audioname$,0,ioaudioptr%,0)
```

If the error% variable contains 0, then everything should work. If it doesn't work, you'll have to release ioaudioptr% and try this again.

3. *Creating a port*

Ports transmit messages between the audio.device and your program. This port should have a name because it is helpful when you use a debugger. Good debuggers display the name of each port, along with their address, so that you know what each port is doing. Also, if your program stops without closing this port, which happens sometimes, another task would be able to handle this.

```
portname$="BigBangAudioProject"
portname%=MALLOC (LEN(portname$)+1,0)
CHAR{portname%}=portname$
audioport%=CreatePort(portname%,0)
```

If everything has worked properly, the audioport% contains a value which isn't zero. The priority established here is zero, a value that should work well.

Remember that the IOAudio structure contains two places for *input/output ports.* The first indicates when the sound that you have sent is finished and the other indicates when the sound began.

You may be wondering why you need to know when the sound began. You'll need this information if you want to synchronize graphics with sound. In this case you'll have to create a second port.

4. *Choosing sound channels*

It is very important to know which channels you're using. The following are the values for each channel:

```
Channel number  3    2    1    0
Value           8    4    2    1
Bit number      3    2    1    0
```

The 0 and 3 channels are the left channels and the 1 and 2 channels are the right ones. To specify a combination of channels, add the values of each channel that is needed. For example, "9" indicates channels 0 and 3, while "6" would be for channels 1 and 2. Each element of the table contains the list of the channels you want to obtain in order of preference.

If you only need one channel, your table will contain the values 1, 2, 4 and 8. This gives priority to channel 0. If you had preferred number 2 first, you would have entered 4,1,2,8:

```
DIM channelmsk|(3)
channelmsk|(0)=4
channelmsk|(1)=1
channelmsk|(2)=2
channelmsk|(3)=8
```

If you want modulation, channel 3 is useless because it doesn't modulate anything. So the possible values are "3" (channel 0 will modulate channel 1), "6" (channel 1 will modulate channel 2) and "12" (so channel 2 can modulate channel 3). For a double modulation, the only possible value is "15" (channel 0 will modulate channel 1 and channel 2 will modulate channel 3).

```
DIM channelmsk|(2)
channelmsk|(0)=3
channelmsk|(1)=6
channelmsk|(2)=12
```

For stereo you'll have to specify the following values: 3 (channels 0 and 1), 5 (channels 0 and 2), 10 (channels 1 and 3) and 12 (channels 2 and 3).

```
DIM channelmsk|(3)
channelmsk|(0)=3
channelmsk|(1)=5
```

```
channelmsk|(2)=10
channelmsk|(3)=12
```

5. *Initializing the IOAudio structure*

The *ADCMD_ALLOCATE* command is used for channel allocation.

```
INT{ioaudioptr%+28} = 32
```

In the first ReplyPort field you'll specify the address of your messages port. If you have created a second port, indicate its address as well.

```
LONG{ioaudioptr%+14} = audioport%
```

Now we'll discuss the limits of your multitasking environment. The audio.device is known for being able to retrieve a channel that it had previously allocated. It's possible that your program will start to play a song and then another channel will steal its channels. However, it's possible that you were stealing the channels from another task to play your song. Fortunately this doesn't have to happen. In the ioa_Data field, you can specify the address of the table containing the channels that you want to obtain and in the ioa_Length field, the number of elements from this table in bytes.

```
LONG{ioaudioptr%+34}=V:channelmsk|(0)
LONG{ioaudioptr%+38}=4
```

The audio.device will automatically place your program in wait if none of the channels you've specified are free. However, you can ask it to give immediate control to your task by placing the *ADIOF_NOWAIT* flag in the io_Flags field.

```
BYTE{ioaudioptr%+30}=64
```

The priority you intend to use is important. You can specify a priority level that is between -128 and +127. When the audio.device accepts your requests for sound channels, it will first see if all of the channels are free by following the order that is specified in the channelmskl() table. If none of the choices mentioned in this table are suitable, the priority you selected will be used. Then the audio.device may have to "steal" channels that were allocated to a lower priority.

```
BYTE{ioaudioptr%+9}=-30
```

We still have to ask the audio.device to give us an allocation key. To do this, place the *AllocKey* field at 0.

```
INT{ioaudioptr%+32}=0
```

6. *Sound channel allocation*

Now that we have correctly initialized all of our variables, we must send the allocation command to the audio.device.

```
BeginIO(ioaudioptr%)
IF WaitIO(ioaudioptr%)
     @closeaudio(1)
ENDIF
```

Now we encounter a problem because GFABasic 3.03 doesn't recognize the *BeginIO* function. So we have to rewrite it by using the RCALL command.

```
DIM reg%(15)
PROCEDURE beginio(ioreq%)
dev%=LPEEK{ioreq%+20}
reg%(9)=ioreq%
reg%(14)=dev%
RCALL dev%-30,reg%()
RETURN
```

If everything worked properly, you'll recover, in the io_Unit field, the numbers of the allocated sound channels. If there is an allocation problem, the audio.device will indicate, in the io_Error field, the reason for the failure.

You can also initialize the ioaudioptr% structure before the OpenDevice call. In this case, the allocation of the channels immediately occurs without a call to BeginIO. Then you'll have to differentiate the channel allocation errors (ADIOERR_ ALLOCFAILED) from opening errors of the audio.device (IOERR_ OPENFAILED which has a value of -1). Performing an allocation twice is okay if you want to retain the modularity and clarity of your program.

We can now send our first notes towards the sound channels. There are still a lot of commands to discuss but *CMD_WRITE* is sufficient for now. We'll only briefly discuss the CMD_WRITE command since it is very similar to sound programming for the DMA channels.

Sound generation

1. *Creating a wave shape*

This wave shape can either be generated by using programming or by reading it from a file (IFF8SVX files are particularly suitable). The height of this wave shape is only limited by the available CHIP memory. The maximum length that each CMD_WRITE can handle is 130,170 bytes.

2. *Initializing the input/output block*

We've already discussed all the important aspects of initializing. The CMD_WRITE command is in the io_Command field and the message port that is used is indicated.

```
LONG{ioaudioptr%+14}=audioport%
INT{ioaudioptr%+28}=3
```

One particular aspect is initializing the io_Flags field. If you position the *ADIOF_PERVOL* flag, the audio.device accepts your new values for the period and the volume. If you don't position this flag, the audio.device will retain its former values. You can use the *ADIOF_WRITEMESSAGE* flag if you have initialized the additional messages port and if you want to be informed when the sound will begin, in which case you will have to wait for the signal with the GetMsg(writeport%) function.

```
BYTE{ioaudioptr%+30}=16
```

The other parameters are initialized in the same way as for the hardware except the duration, which is expressed in bytes instead of words. The audio.device divides this number by two. However, don't specify odd-number lengths.

```
LONG{ioaudioptr%+34}=forme%
LONG{ioaudioptr%+38}=v_length%
```

```
INT{ioaudioptr%+42}=periode%
INT{ioaudioptr%+44}=volume%
INT{ioaudioptr%+46}=duration%
```

Send the command with BeginIO:

```
beginio(ioaudioptr%)
IF WaitIO(ioaudioptr%)
     @closeaudio(0)
ENDIF
```

Example

This example plays exactly the same wave form as our example of hardware programming. We have re-grouped audio.device initializing in the initaudio procedure and the sounds are played with the playsound procedure.

```
' AUDIODEVICE.GFA
' This program is a simple example of utilizing the
' audio.device. It was inspired by an article by Dan
' Baker of CATS in the AmigaMAIL of  March/April 1988.
'
initaudio
' Initialization of the wave form
'
v_length%=64
audioliste%=MALLOC(v_length%,2)
IF audioliste%=0
@closeaudio(1)
ENDIF
FOR i|=0 TO 63
  POKE audioliste%+i|,@timbre(i|)
NEXT i|
periode%=@freq(440,64)
'
'
nbpasses%=v_length% DIV maxlen%
reste%=v_length% MOD maxlen%
i|=0
WHILE i|<nbpasses%
```

```
playsound(port%,audioliste%+maxlen%*i|,maxlen%,1000,perio
de%,64,iobptr%)
  INC i|
WEND
playsound(port%,audioliste%+maxlen%*i|,reste%,1000,period
e%,64,iobptr%)
@closeaudio(0)
'
PROCEDURE
playsound(port%,forme%,v_length%,duration%,periode%,volum
e%,iobptr%)
  ' Play one note thru the audio.device
  ' port% indicates the Port utilized
  ' forme%=wave form
  ' V_length%=length of the wave form
  ' Duration%=number of  cycles
  ' periode%=period of  reproduction
  ' volume%=sound intensity (0-65)
  ' iobptr%=pointer to the audio block
  '
  ' Initialization of blockc
  LONG{iobptr%+14}=port%                    ! ReplyPort
  INT{iobptr%+28}=3                         ! CMD_WRITE
  BYTE{iobptr%+30}=16                       ! ADIOF_PERVOL
  LONG{iobptr%+34}=forme%
  LONG{iobptr%+38}=v_length%
  INT{iobptr%+42}=periode%
  INT{iobptr%+44}=volume%
  INT{iobptr%+46}=duration%
  '
  ' Execute the commands
  beginio(iobptr%)
  IF (WaitIO(iobptr%))
  @closeaudio(0)
  ENDIF
RETURN
PROCEDURE initaudio
  clock%=@getclock   ! Get frequency
  maxlen%=130170     ! Maximum accepted by the
```

```
 'audio.device
 '
 DIM r%(15),channelmsk|(3)
 channelmsk|(0)=1
 channelmsk|(1)=2
 channelmsk|(2)=4
 channelmsk|(3)=8
 '
 ' Pointer to the sorted entries in the audio block
 iobptr%=MALLOC(68,&H10003) ! MEMF_CHIP | MEMF_CLEAR |
MEMF_PUBLIC
 IF iobptr%=0 THEN
 @closeaudio(4)
 ENDIF
 '
 ' Creation of the Audio port -
 ' Retrieve with the FindPort function
 a$="GFABasicAudioDevPort"+CHR$(0)
 port%=CreatePort(V:a$,0)
 IF port%=0 THEN
 @closeaudio(3)
 ENDIF
 '
 ' Initialization of the iobptr% and opening the device
 LONG{iobptr%+14}=port%                  ! ReplyPort
 BYTE{iobptr%+9}=-40                     ! Priority
 INT{iobptr%+28}=32                      ! ADCMD_ALLOCATE
 BYTE{iobptr%+30}=64                     ! ADIOF_NOWAIT
 INT{iobptr%+32}=0                       ! AllocKey
 LONG{iobptr%+34}=V:channelmsk|(0)
 LONG{iobptr%+38}=4
 '
 ' Open the Audio.device
 a$="audio.device"+CHR$(0)
 device%=OpenDevice(V:a$,0,iobptr%,0)
 IF device%<>0 THEN
 @closeaudio(2)
 ENDIF
 '
 ' Allocation of the channels--
```

```
  ' The function BeginIO is the only one
  ' usable with the audio.device.
  ' Do not use  SendIO or DoIO.
  beginio(iobptr%)
  IF (WaitIO(iobptr%))
  @closeaudio(1)
  ENDIF
RETURN
PROCEDURE beginio(req%)
  ' BeginIO is not supported in  V3.03. However this
  ' procedure is equivalent.
  fdev%=LPEEK(req%+20)  ! Address of the device
  r%(9)=req%            ! Pass pointer of IORequest in A1
  r%(14)=fdev%          ! Pointer to the device in A6
  RCALL fdev%-30,r%()   ! BEGIN_IO is found at offset -30
RETURN
PROCEDURE closeaudio(ind)
  ' Free memory
  ' ind : return index
  SELECT ind
  CASE 0
    ~MFREE(audioliste%,v_length%)
    CONT
  CASE 1
    ~DeletePort(port%)
    CONT
  CASE 2
    ~CloseDevice(iobptr%)
    CONT
  CASE 3
    ~MFREE(iobptr%,68)
    CONT
  CASE 4
    '
  ENDSELECT
  '  PRINT "End of program. Return code: ";ind
  SYSTEM
RETURN
FUNCTION timbre(i)                ! Sine wave form
  RETURN BYTE(SIN(i*PI/v_length%)*127)
```

```
ENDFUNC
FUNCTION freq(f%,l%)
  ' Calculate a value of the period that affects a
  'channel frequency
  ' and the number of sample per second
  ' f% : frequency of the note of an instrument
  ' and samplesPerSec
  ' l% : samplesPerHiCycle then the  instrument and l
  RETURN INT(clock%/(f%*l%))
ENDFUNC
FUNCTION getclock                ! Get frequency
  ' Test  DisplayFlags of the structure GfxBase ; PAL = 4
  IF WORD{_GfxBase+206} AND 4
    RETURN 3546895 ! Mode PAL
  ELSE
    RETURN 3579545 ! Mode NTSC
  ENDIF
ENDFUNC
```

We can now improve this program by adding routines for reading files in IFF-8SVX format.

Chapter 4

IFF 8SVX and SMUS Formats

[illegible]

Chapter 4 IFF 8SVX and SMUS Formats

In this chapter we'll discuss sound programming techniques that enable you to read IFF files.

4.1 IFF standard

The IFF standard (Interchange File Format) was developed in 1985 by the Electronic Arts company. The concept behind this standard was to enable users to create, with software, a graphic or a piece of music which could be read by other programs.

The IFF standard enables you to change software while retaining your earlier creations. So you can use one program to draw the contours of an image, fill in the details of the image by using another program, and then create special effects with a third program.

Not many people know that the IFF format is used with the clipboard.device. During copy functions in text programs, IFF FTXT (Formatted Text) format enables your applications to transfer text in bold, italics, or other character styles. Even fonts can be transferred. So, theoretically, this will also work for everything that is in the IFF format, whether it is graphics, music, etc.

The word "FORM" indicates the start of an IFF file. The long word that follows specifies the length of the form. The type of the file is specified in the four letter word that follows (i.e., SMUS for a musical score). Immediately after the type is the code for the first *chunk* (or data block), followed by its length in bytes. Data about the chunk is listed next.

The main IFF formats are ILBM (pictures), FTXT (text), SMUS (musical scores), and 8SVX (sounds). The newest formats include ANIM (animation) and PGTB (program traces).

The IFF formats that are devoted to music on the Amiga are the *8SVX* (8 bit Sampled Voice) format, which is used for reproducing sampled sounds, and *SMUS* (Simple Musical Score), which is used for recording and reproducing musical scores. These formats are used to simplify the creation of music with Amiga music software, and then to reproduce these pieces of music in commercial programs, such as in games, or in piloting MIDI synthesizers.

It is very important that the software you use, or create, is able to accept these formats. Otherwise, it may be impossible to exchange your musical scores with other software.

The main problem with IFF SMUS and 8SVX files is that it is almost impossible to find program source codes that can be listed. In spite of hundreds of diskettes in the public domain, the number of musical programs is very limited. We have provided source codes that we've created in GFABasic. Although they may not be algorithmically perfect, at least you can access them.

4.2 IFF 8SVX format

The IFF 8SVX format stores and reproduces sampled sounds on 8 bits. The 8 bit sounds have either been sampled or created by a program. The value of each sample is between -128 and +127. You can reduce the size of the file by almost 50% by using a Fibonnacci Delta compression algorithm.

If the file is an instrument file, it will be set up so that you'll be able to get the most sounds from the instrument. The file will contain a sampling of the instrument based on several octaves with the highest octave first, followed by the others.

Theoretically, the limit for the octave number is 255 but seven octaves is an acceptable maximum. Later we'll show you how to determine the highest note of an instrument file.

The 8SVX format is set up according to a standard form. At the beginning of these files, is a header, called VHDR (Voice HeaDeR), that helps you determine the kind of sound you're working with. An IFF8SVX file begins as follows:

```
FORM####8SVXVHDR####
```

The keyword FORM signifies that you're working with an IFF file. The four subsequent bytes indicate the total length of the information contained in this FORM. Usually the length of the file will be -8, but in order to observe IFF specifications, you must follow the "PROP," "LIST," and "CAT" headers.

You may only have one FORM 8SVX in a file (the simplest case) or the file may be one of many IFF specifications, which describe musical pieces, graphics, animations, other sounds, etc. If there are other types of IFF files, the file will begin with either the "LIST" or "CAT" identifier.

If you're only interested in graphics, look for the FORM ILBM keyword. The keyword CAT indicates that more than one of these types of files may be available.

If you encounter a FORM 8SVX, you will advance according to the number of characters specified between the identifiers FORM and

8SVX. This number must be even; otherwise it won't function properly.

Then you'll encounter a new keyword and continue to search until you find the FORM.ILBM you want.

The LIST identifier works the same way as CAT. However, LIST can also contain PROP identifiers, which define a default characteristic for the subsequent FORMs. A FORM identifier can also be included in the data of another FORM. However, this wouldn't be valid for SMUS and 8SVX FORMs.

Now let's return to the 8SVX file. The bytes following the VHDR identifier (the chunk) indicate the size of the file. The most dangerous trap in the IFF format is the indicated size of a chunk.

This doesn't necessarily have to be an even number. However, the effective size must be even. If a chunk indicates a length of 9 bytes, you should use the first 9 bytes for processing; the tenth byte should contain a zero. This is called the *fillbyte* because it "pads" the length of the chunk; it isn't involved with the length information of the chunk.

It's very important to use the fillbyte when you're creating your own files in the IFF format. Only a zero can be used in the fillbyte.

Now that you have carefully read all these warnings, let's look at the VHDR chunk. The following is its definition in the C programming language:

```
typedef LONG Fixed;

typedef struct {
ULONG    oneShotHiSamples,        ( 0)
         repeat HiSamples,        ( 4)
         samplesperHiCycle;       (.8)
UWORD    samplesperSec;           (12)
UBYTE    ctOctave,                (14)
         sCompression;            (15)
Fixed volume                      (16)
}  Voice8Header
```

The length of the chunk is given by the 4 bytes following the VHDR identifier (currently there are 20). During the reading you have to differentiate the instruments from the simple sampled sounds. You also have to consider the Fibonacci Delta compression, which we'll discuss shortly.

The number of bytes representing the size of a sample is *oneShotHiSamples + repeatHiSamples* for the highest octave. Double this to determine the next highest octave and then continue to do this in order to determine the following octaves.

You may be wondering why we've separated the number of samples into two variables. We did this because, in order to produce the sound, we first have to use the part contained in OneShotHiSamples, then repeat, as many times as necessary, the part contained in repeatHiSamples (which contains the sustain). This algorithm will allow you to determine the total size of the sound:

```
length%=oneShotHiSamples%+repeatHiSamples%
i|+1
WHILE i|<ctOctave|
      ADD length%,length%*2
      INC i
WEND
```

If you want, you can replace the line, that contains ADD, with:

```
ADD length%,SHL(length%,1)
```

The value of samplesperHiCycle is determined during the sampling. It gives you the number of samples per cycle. For example, if you want to define a sound of 440 Hz, you'll need 440 * samplesperHiCycle.

With instrument files, there are several available octaves. Their number is located in ctOctave.

To determine how many samples will be needed while using the second octave, simply use the following formula: 440 * samplesperHiCycle * 2.

```
frequency = samplesperSec / samplesperHiCycle
```

By using the following formula, you can find the frequency of each of the octaves included in an instrument file:

```
DIM frequency(ctOctave|-1)
FOR i|=0 To ctOctave|-1
  frequency(i|)=samplespersec&/samplesperHiCycle*2^(i|-1)
NEXT i|
```

You may think that the biggest problem is determining which note you're using. However, if you know the frequency of A-440 (440 Hz), you'll be able to determine the note. Remember the formula we discussed at the beginning of the book:

```
Savarts=LOG10(f1/440)*1000
```

The value returned here will be positive if the note is higher than the A an octave above 440 Hz, and negative if its frequency is lower than this. The number of half tones between A an octave above 440 Hz and the note will be:

```
Halftones%=12*savarts/301
```

If you're using a notes chart, use A-440 with an index of 69, since this is the index taken from the MIDI interface. The index of the note you're looking for will be:

```
Index%=69+halftones%
```

We'll discuss this in more detail when we present SMUS files.

We still need to discuss compression and volume. Currently only one type of compression is used in IFF-8SVX: the Fibonacci Delta compression. When using this compression, the sCompression variable contains the value 1. We'll present the decompression algorithm in GFABasic in the next program.

The volume corresponds to the sound level of the restitution. It's helpful if some information about wave shape (ATAK and RLSE chunks) is available.

The NAME, (c), AUTH, ANNO, ATAK, and RLSE chunks are optional. The first four contain parts of a text: the name of the sound, its copyrights, its author, and particular notes for the sound. The NAME chunk might be useful if you use SMUS and want to identify an instrument.

The ATAK and RLSE chunks contain the envelope of the sound (ADSR) in the form of points. Since each point is 6 bytes long, you can calculate the number of points by dividing the size of the chunk by 6.

The following is its definition in the C programming language:

```
Typedef struct {
     UWORD duration;       (0)
     Fixed dest;    (2)
}  EGPoint;
```

Each point indicates a volume modification for a duration that is expressed in milliseconds. By placing the initial volume at zero, you can calculate the output volume by multiplying the volume obtained in the VHDR chunk by "dest".

The following program reads and executes an 8SVX file. The sound routine is the same as that of the preceding program. Subsequently, it won't be able to read SMUS files, so we'll replace it with a group of routines created by Steven A. Bennett and modified by Rob Peck.

```
' Player 8SVX -- 30/06/1989
' Dominique Lorre --  Micro Application / Abacus
'
' This program plays 8SVX songs utilizing the
audio.device
' The program was inspired by an article by Dan Baker of
' CATS published in AmigaMAIL of  March/April 1988. The
' reading of *SVX file used the Fibonacci Delta method of
' file compression.
'
' Select the sample file/ Your disk name may differ
FILESELECT "IFF-8SVX
Files","Load","Amiga_Music:Songs/",nom$
'
```

```
initaudio
lit8svx(nom$)
' Set the size of the file
' to 130170 since this is limited by the hardware
nbpasses%=v_length% DIV maxlen%
reste%=v_length% MOD maxlen%
i|=0
WHILE i|<nbpasses%

playsound(port%,audioliste%+maxlen%*i|,maxlen%,1,periode%
,64,iobptr%)
  INC i|
WEND
playsound(port%,audioliste%+maxlen%*i|,reste%,1,periode%,
64,iobptr%)
@closeaudio(0)
'
PROCEDURE
playsound(port%,forme%,v_length%,duration%,periode%,volum
e%,iobptr%)
  ' Plays a note using the audio.device
  ' port% indicated the port utilized
  ' forme%=wave form
  ' v_length%=length of the wave form
  ' duration%=number of cycles
  ' periode%=period of reproduction
  ' volume%=asound intensity (0-65)
  ' iobptr%=pointer to the sorted entries in the
  ' audio block
  '
  ' Initialization of the block
  LONG{iobptr%+14}=port%                    ! ReplyPort
  INT{iobptr%+28}=3                         ! CMD_WRITE
  BYTE{iobptr%+30}=16                       ! ADIOF_PERVOL
  LONG{iobptr%+34}=forme%
  LONG{iobptr%+38}=v_length%
  INT{iobptr%+42}=periode%
  INT{iobptr%+44}=volume%
  INT{iobptr%+46}=duration%
  '
  ' Start the commands
  beginio(iobptr%)
  IF (WaitIO(iobptr%))
  @closeaudio(0)
  ENDIF
RETURN
PROCEDURE initaudio
  clock%=@getclock  ! Frequency of the clock
  maxlen%=130170    ! maximum values acceptable to the
audio.device
  '
  DIM r%(15),channelmsk|(3),codetodelta&(15)
  channelmsk|(0)=1
  channelmsk|(1)=2
  channelmsk|(2)=4
  channelmsk|(3)=8
  '
  FOR i|=0 TO 15           ! Data for the decompression
    READ codetodelta&(i|)
  NEXT i|
  ' Pointer to the sorted entries in the audio block
  iobptr%=MALLOC(68,&H10003) ! MEMF_CHIP | MEMF_CLEAR |
MEMF_PUBLIC
  IF iobptr%=0 THEN
  @closeaudio(4)
  ENDIF
  '
  ' Creation of the audio port
  '
  a$="GFABasicAudioDevPort"+CHR$(0)
  port%=CreatePort(V:a$,0)
  IF port%=0 THEN
  @closeaudio(3)
  ENDIF
  '
  ' Initialization of iobptr% and opening device
  LONG{iobptr%+14}=port%              ! ReplyPort
  BYTE{iobptr%+9}=-40                 ! Prioriti
  INT{iobptr%+28}=32                  ! ADCMD_ALLOCATE
  BYTE{iobptr%+30}=64                 ! ADIOF_NOWAIT
  INT{iobptr%+32}=0                      ! AllocKey
  LONG{iobptr%+34}=V:channelmsk|(0)
  LONG{iobptr%+38}=4
  '
  ' Opening the audio device
  a$="audio.device"+CHR$(0)
  device%=OpenDevice(V:a$,0,iobptr%,0)
  IF device%<>0 THEN
  @closeaudio(2)
  ENDIF
  '
  ' Allocation of the channels -- The function BeginIO is
  ' the only function used by the audio.device. Don't
  ' use SendIO or DoIO.
  beginio(iobptr%)
  IF (WaitIO(iobptr%))
  @closeaudio(1)
  ENDIF
RETURN
PROCEDURE beginio(req%)
  ' BeginIO is not supported in  V3.03. This procedure is
  ' equivalent.
  fdev%=LPEEK(req%+20)  ! Address to device
  r%(9)=req%            ! Pass the  pointer IORequest ins
A1
  r%(14)=fdev%          ! Pointer to the Device in A6
  RCALL fdev%-30,r%()   ! BEGIN_IO found at offsett -30
RETURN
PROCEDURE closeaudio(ind)
  ' Free memory
  ' ind : return index
  SELECT ind
  CASE 0
    ~MFREE(audioliste%,v_length%)
    CONT
  CASE 1
    ~CloseDevice(iobptr%)
    CONT
  CASE 2
    ~DeletePort(port%)
    CONT
  CASE 3
    ~MFREE(iobptr%,68)
    CONT
  CASE 4
    '
  ENDSELECT
  '  PRINT "End of program. Return code : ";ind
  SYSTEM
RETURN
PROCEDURE lit8svx(nom$)
  OPEN "I",#1,nom$
  get4chars
  IF a$<>"FORM"
    ALERT 0,"Not IFF format file!",0,"OK",a%
    END
  ENDIF
  get4chars
  get4chars
  IF a$<>"8SVX"
    ALERT 0,"Not IFF 8SVX file !",0,"OK",a%
    END
  ENDIF
  get4chars
  IF a$="VHDR" ! recover the sample information
    get4chars  ! pass the table of the chunk (20?)
    ' The actual table is 20 values?
    '
    oneshothisamples%=CVL(INPUT$(4,#1))
    repeathisamples%=CVL(INPUT$(4,#1))
    samplesperhicycle%=CVL(INPUT$(4,#1))
    samplespersec&=CVI(INPUT$(2,#1))
    ctoctave|=INP(1)
    scompression|=INP(1)
    volume%=CVL(INPUT$(4,#1))
  ENDIF
  get4chars
  WHILE a$<>"BODY"
    skipchunk
    get4chars
    WEND
    longchunk%=CVL(INPUT$(4,#1))
    ' Does not allow for future forms of compression
    '   possibles
    IF scompression|
      templiste%=MALLOC(longchunk%,&H5)
      IF templiste%=0
      @closeaudio(1)
      ENDIF
    ENDIF
    i|=0
    v_length%=0
    WHILE i|<ctoctave|
      ADD
v_length%,(oneshothisamples%+repeathisamples%)*2^i|
      INC i|
    WEND
    audioliste%=MALLOC(v_length%,2)
    IF audioliste%=0
      IF templiste%
        ~MFREE(templiste%,longchunk%)
      ENDIF
    @closeaudio(1)
    ENDIF
    nbpasses%=longchunk% DIV 512
    reste%=longchunk% MOD 512
    FOR i&=0 TO nbpasses%-1
      BGET #1,audioliste%+512*i&,512
    NEXT i&
    IF reste%
      BGET #1,audioliste%+512*nbpasses%,reste%
    ENDIF
    CLOSE #1
    IF scompression|
      BMOVE audioliste%,templiste%,longchunk%
      dunpack(templiste%,longchunk%,audioliste%)
      ~MFREE(templiste%,longchunk%)
    ENDIF
    periode%=@freq(samplespersec&,1)
RETURN
```

```
PROCEDURE dunpack(source%,n%,dest%)
  ' Decompress compressed files using the Fibonacci Delta
method
  ~@dlunpack(source%+2,n%-2,dest%,BYTE{source%+1})
RETURN
PROCEDURE get4chars
  ' Read the 4 characters
  a$=INPUT$(4,#1)
RETURN
PROCEDURE skipchunk
  LOCAL l,i
  ' Ignore a chunk with out regard for the consequences
  l=CVL(INPUT$(4,#1))
  IF ODD(l)
    INC l
  ENDIF
  FOR i=1 TO l
    VOID INP(#1)
  NEXT i
RETURN
FUNCTION freq(f%,l%)
  ' Calculate the value of the period affecting a channel
  ' and the frequency and the number of sample per second
  ' f% : frequency of the instrument and samplesPerSec
  ' l% : samplesPerHiCycle of the instrument and 1
  RETURN INT(clock%/(f%*l%))
ENDFUNC
' The function getclock is taken from the specifications
' of Carolyn Scheppner of CATS published in AmigaMail
FUNCTION getclock                ! Determine frequency
  ' Test the DisplayFlags of structure GfxBase ; PAL = 4
  IF WORD{_GfxBase+206} AND 4
    RETURN 3546895 ! Mode PAL
  ELSE
    RETURN 3579545 ! Mode NTSC
  ENDIF
ENDFUNC
FUNCTION dlunpack(source%,n%,dest%,x&)
  ' Function to decompress  Fibonacci Delta
  ' source% : Buffer entry
  ' n% : table of buffer entry
  ' dest% : Buffer sorted
  ' x& : Value of the preceding char
  ' (the algorithm calculates the new value
  ' of the preceding value)
  '
  lim%=MUL(n%,2)
  FOR i%=0 TO lim%-1
    d|=BYTE{source%+SHR(i%,1)}
    IF AND(i%,1)
      d|=AND(d|,&HF)
    ELSE
      d|=SHR|(d|,4)
    ENDIF
    ADD x&,codetodelta&(d|)
    BYTE{dest%+i%}=BYTE(x&)
  NEXT i%
  RETURN x&   ! we return the calculated value of x&
  '
ENDFUNC
' Data for the decompression Fibonacci Delta
DATA -34,-21,-13,-8,-5,-3,-2,-1,0,1,2,3,5,8,13,21
```

4.3 IFF SMUS format

Although the SMUS format wasn't popular with developers, it's a format that can be easily understood by the average user and programmer.

In this section we'll present a SMUS file listing. If you work in the C language, it will take some extra effort in order to use this example in C. You can use the routines developed by the two programmers we mentioned earlier. These routines are included on the companion diskette for this book.

Before presenting the program, let's take a close look at the SMUS format. The two important characteristics of the SMUS format are the tracks and the instruments.

A *track* is like a melody line. It's a succession of notes, chords (notes of the same duration played together), and ties. Events, such as a change of volume (e.g., crescendo or decrescendo) or a change of tempo or instrument, can appear on a track.

The instruments are those used during the music program. Several tracks may use the same instrument and one track may change instruments. The instruments must correspond either to an IFF 8SVX instrument file, which we discussed in the preceding section, or to a MIDI channel (although this isn't the case for all software).

However, you don't always have to follow these requirements. You could very easily manage the instruments as you want. Sonix, for example, uses specific instruments. However, the SMUS format only allows 256 possible instruments.

An IFF SMUS file begins as follows:

```
FORM####SMUSSHDR####
```

The SHDR chunk indicates a SScoreHeader. The following is its structure in the C language:

```
typedef struct {
UWORD tempo;        (0)
UBYTE volume;       (2)
UBYTE ctTrack;      (3)
}SScoreHeader;
```

The two most important pieces of information are the tempo and ctTrack (number of tracks). We express tempo in 128ths of a quarter note per minute. Volume is a value between 0 and 127. Since we are only concerned with volumes between 0 and 64 on the Amiga, we'll divide this field by 4. The number of tracks is indicated by ctTrack.

After SHDR there are INS1 chunks. The following is their definition in the C programming language:

```
typedef struct {
UBYTE register;     (0)
UBYTE type;  (1)
UBYTE data1,data2;  (2),(3)
char name[]  (4)
}  RefInstrument;
```

In the file the size of the chunk appears immediately after the four letters "INS1":

```
INS1####|register|type|data1|data2|name.....
```

Assign a number between 0 and 255 to each instrument declared in the file. This number is also located in "register".

"Type" indicates whether it's a MIDI instrument (type = 1), an IFF 8SVX (type = 0), or some other kind of instrument. Sonix uses different instruments than IFF 8SVX and MIDI, but apparently a type hasn't been assigned to them.

The fields for data1 and data2 are used for instruments other than IFF instruments; our program doesn't consider these. Name is the name of the instrument, which is a succession of characters whose length you calculate with the following formula:

```
v_length = ckSize - 4;
```

ckSize is the length of the chunk.

Now we'll discuss the format of the tracks. A track begins with the TRAK identifier, followed by its length on a long word (4 bytes). Then there is the number of events of the track written in words, and finally the events, which are coded on two bytes. The first byte is the identifier and the second contains the data. The notes are coded from 0 to 127, which correspond to MIDI events and rests have an indentifier of 128. The following is their definition:

```
typedef struct {
     BYTE sID;
     UBYTE data;
} SEvent;
```

sID = 129 This indicates a change of instrument. The track takes the data number for the instrument. We manage an instrument table in our program.

sID = 132 This regulates the volume of the current track. The value is between 0 and 127. To obtain a usable value, divide "data" by 4.

sID = 136 This indicates a new tempo which will be calculated according to the former tempo and to the value contained in "data".

```
tempo = tempo*(data + 1)/128
```

The other sID is either used for MIDI or for a music editor. Let's return to the sID from 0 to 127, which identifies notes and 128, which is for rests.

```
typedef struct {
     UBYTE tone;
     unsigned chord          :1,
     tieOut                  :1,
     nTuplet                 :2,
     dot                     :1,
     division                :3;
} SNote;
```

If you aren't familiar with the C language, you should know that this structure contains grouped bits:

```
     chord   dot
xxxxxxxx|7|6|5 4|3|2 1 0|
tone         nTuplet
     tie Out division
```

In this program, tone is like sID. The division (bits 0 to 2) gives the duration of a note in comparison to a whole note. This part contains 0 for a whole note, 1 for a half note, 2 for a quarter note, 3 for an eighth note, etc.

Bit number 7 of the data part indicates whether the note is a part of a chord. The number 6 indicates a tie. Bits 4 and 5 indicate the note type. These bits specify whether it's a full note, a triplet, a quintuplet or a septuplet:

```
00 : full note      (0) : multiply the duration by 1
01 : triplet        (1)   multiply the duration by 2/3
10 : quintuplet     (2)   multiply the duration by 4/5
11 : septuplet      (3)   multiply the duration by 6/7
```

Bit number 3 indicates that the note is dotted: you have to multiply its value by 3/2. The duration table() is initialized in our example program according to this principle.

The solution we use in our program to resolve ties is recursive:

```
Action read_chord
     Repeat<R>
       read_note
     As long as the note is a part of a chord and notes
     remain
     If the note is tied
        read following chord
        treating ties
     End If
End Action
```

This algorithm works because, when a note is tied, all the notes of its chord are marked as being ties. Now let's look at treating_ties:

```
Action treating_ties
     For index := 1 up to length(chordl)
            For index2 := 1 up to length (chord2)
               If sid(chordl) = sid(chord2)
                   sid(chord2) := alreadytreated
            duration(chordl) :=
            duration(chordl)+duration(chord2)
               EndIf
            End For
     End For
End Action
```

We have already seen how sID number 255 is reserved for these kinds of operations.

Example program:

The following is the final result, based on the routines we've been discussing in this chapter. The sound routines, placed in the public domain by Rob Peck, can be found in the original sources in the C programming language on the companion diskette for this book.

The demo listed on the following pages is split into two programs. The setup program SMUSPLAYER.GFA performs memory configuration routines, then calls the main program named SSMUS.GFA using the CHAIN command. SSMUS displays a file requester. Click on EXAMPLE.SMUS. SSMUS will display another file requester. Click on the BEEP name, and listen to the music.

Setup program

```
' SMUSPLAYER.GFA
CLEAR
RESERVE 100*1024
'
CHAIN "Amiga_Music:CH.4/SSMUS.GFA"
' your chain command may be different, depending
' on where you stored the SSMUS.GFA program
```

Main program

```
'SSMUS.GFA
' Reading SMUS files -- Only sounds are loaded
' The routines of the program are, in a large part
' inspired from articles written by Dan Baker,
' Steven A. Bennett, Rob Peck and Carolyn Scheppner
'
' This program is only an example.
' Synchronization, in case you use stereo, is average.
' Problems of memory allocation remain unsolved.
' A possible solution would be to treat notes
' measure by measure with the risk of having
' clicks between each measure. In spite of the
' programming faults we hope that the
' routines furnished will ease the task of
' programmers who use IFF-SMUS format in their programs.
' 6/07/89 -- Version mnumber: 0.9
' Dominique Lorre --Micro Application / Abacus
'
' Main Program
' -------------------
'
initaudio
'
ON ERROR GOSUB catcherr
'    -------- error treatment necessary for  FILESELECT
FILESELECT "SMUS
Files","Load","Amiga_Music:Partitions/",name$
'                           ---- selection of the SMUS files
initsmus
readsmus(name$)
'    ----  Read the file contents and before playing tune
ON BREAK GOSUB finit
'               ----- Quit in case of CTRL-SHIFT-ALT
'
PRINT "SMUSPlayer -- Version 0.9 -- Press CTRL-SHIFT-
left ALT"
PRINT "to interrupt the program"
'
```

```
playsmus
'       Play music
'
PRINT "Press both mouse buttons to exit."
REPEAT
UNTIL MOUSEK=3
finishaudio
'          Stop all sounds
freesmus(0)
@closeaudio(1)
'
PROCEDURE initaudio
  '      ---------
  ' Initialization of the audio.device -- the procedure
  ' is based on the routines published by Rob Peck and
  ' Steven A. Bennett.
  '
  ' Number of the audio buffers audio
  audbuffers%=20
  '
  ' Declaration of the global and dynamic ports
  '
  globalname$="SMUSGlobalPort"
  dynamicname$="SMUSDynamicPort"
  lg%=LEN(globalname$)+1
  globalname%=MALLOC(lg%,&H10000)
  IF globalname%=0
  @closeaudio(9)
  ENDIF
  ld%=LEN(dynamicname$)+1
  dynamicname%=MALLOC(ld%,&H10000)
  IF dynamicname%=0
  @closeaudio(8)
  ENDIF
  ' You are probably wondering why we have not used
  ' a$="MonPort"+CHR$(0)
  ' monport%=CreatePort(V:a$,0)
  '
  ' The reason is that a$ is a GFABasic variable
  ' whose contents may eventually change.
  ' The ports name could change at the same time.
  ' causing random errors.
  CHAR{globalname%}=globalname$
  CHAR{dynamicname%}=dynamicname$
  '
  clock%=@getclock     ! Get clock frequency
  '
  ' Global variables for IFF instruments  --
  ' 256 instruments possible
  '
  DIM oneshothisamples%(255),repeathisamples%(255),
samplesperhicycle%(255)
  DIM samplespersec&(255),ctoctave|(255),volume%(255)
  DIM v_length%(255),audioliste%(255)
  '
  DIM codetodelta&(15) ! Table for Fibonacci delta
                       ' decompression
  '
  DIM r%(15)           ! Table of registers for calling
                       ' the BeginIO function
  '
  DIM inuse!(audbuffers%-1),replyport%(3),unit%(3)
  DIM audbuffer%(audbuffers%-1),dynamix%(3)
  '
  ' Initialization of the tables
  '
  ARRAYFILL inuse!(),FALSE
  ARRAYFILL dynamix%(),0
  '
  FOR i|=0 TO 3
    unit%(i|)=SHL(1,i|)
  NEXT i|
  '
  ' Here channelmsk is no table since only one entry is
  ' necessary
  '
  channelmsk|=15          ! All channels or nothing
  '
  FOR i|=0 TO 15       ! Data for Fibonacci Delta
                       ' decompression
  READ codetodelta&(i|)
NEXT i|
'
' Memory allocation for static memory buffers
'
ioa%=MALLOC(68*(audbuffers%+1),&H10000)
'                   MEMF_CLEAR : essential for Blocs IO
IF ioa%
  FOR i|=0 TO audbuffers%-1
    audbuffer%(i|)=ioa%+68*(i|+1)
  NEXT i|
ELSE
@closeaudio(7)
ENDIF
'
' Opening the audio.device
'
' The audiostruct routine allows you to use the
' IOAudio structure system easily from GFABasic.
' It does slow down execution.
audiostruct(ioa%)
ioa_length%=0
a$="audio.device"+CHR$(0)
aerror%=OpenDevice(V:a$,0,ioa%,0)
IF aerror%<>0 THEN
@closeaudio(6)
ENDIF
device%=io_device%
'
p$="SMUSControlPort"
lc%=LEN(p$)+1
no_controlport%=MALLOC(lc%,&H10000)
IF no_controlport%=0
@closeaudio(5)
ENDIF
CHAR{no_controlport%}=p$
controlport%=CreatePort(no_controlport%,0)
IF controlport%=0
@closeaudio(4)
ENDIF
```

```
'
' Initialization of ioa% and opening the device
'
io_ln_pri|=BYTE(-40) ! Priorities
io_mn_replyport%=controlport%
io_command&=32          ! io_Command = ADCMD_ALLOCATE
io_flags|=64 OR 1       !  ADIOF_NOWAIT and IOF_QUICK
ioa_allockey&=0         ! allocation
ioa_data%=V:channelmsk| ! choose channel
ioa_length%=1
'
' Allocation  channel  -- The BeginIO function is the
' only function usable with the audio.device. Do not
' use SendIO or DoIO.
beginio(ioa%)
'
'
IF (WaitIO(ioa%))
  ' Response from device. !
@closeaudio(3)
ENDIF
' Response from device
'
'
IF NOT (io_flags| AND 1)
  ~GetMsg(io_mn_replyport%)
ENDIF
' allocation
allockey&=ioa_allockey&
'
' Create the audio port and choose the channel
' SMUSPlayerPort0 and SMUSPlayerPort3
'
p$="SMUSPlayerPort"
'
lp%=LEN(p$)+2
nomport%=MALLOC(lp%*4,0)
IF nomport%=0
@closeaudio(2)
ENDIF
```

```
    FOR i|=0 TO 3
      px$=p$+CHR$(48+i|)
      nomportx%=nomport%+lp%*i|
      CHAR{nomportx%}=px$
      replyport%(i|)=CreatePort(nomportx%,0)
      IF replyport%(i|)=0
      @closeaudio(1)
      ENDIF
    NEXT i|
  RETURN
  PROCEDURE beginio(req%)
    ' req% : Address of the request block sorted entries
    ' (IORequest)
    '
    ' BeginIO is not supported in  V3.03. This routine is
    ' equivalent
    '
    fdev%=LPEEK(req%+20)  ! Address of the device
    ARRAYFILL r%(),0
    r%(9)=req%          ! Pass the pointer IORequest in A1
    r%(14)=fdev%          ! Pointer to the device in A6
    RCALL fdev%-30,r%()   ! BEGIN_IO found at offset -30
  RETURN
  PROCEDURE finishaudio
    ' Stop playing - free pointers
    FOR i|=0 TO 3
      flushchannel(i|)
    NEXT i|
    DELAY 1
    reemployiob
  RETURN
  PROCEDURE flushchannel(i|)
    '
    ' end the song
    ' i| : audio channel
    '
    audiostruct(ioa%)
    io_device%=device%
    io_mn_replyport%=controlport%
    initblock(ioa%,i|)
    io_command&=8                  ! Command : CMD_FLUSH
    io_flags|=1                    ! Flags   : IOF_QUICK
    ' IOF_QUICK permits passing commands
    beginio(ioa%)
    ~WaitIO(ioa%)
  RETURN
  PROCEDURE closeaudio(ind|)
    '
    ' Free memory
    ' ind| : return index
    '
    '
    LOCAL i|
    SELECT ind|
    CASE 1
      FOR i|=0 TO 3
        IF replyport%(i|)
          ~DeletePort(replyport%(i|))
        ENDIF
      NEXT i|
      CONT
    CASE 2
      ~DeletePort(controlport%)
      CONT
    CASE 3
      ~MFREE(nomport%,lp%*4)
      CONT
    CASE 4
      ~MFREE(no_controlport%,lc%)
      CONT
    CASE 5
      ~CloseDevice(ioa%)
      CONT
    CASE 6
      ~MFREE(ioa%,68*(audbuffers%+1))
      CONT
    CASE 7
      ~MFREE(dynamicname%,ld%)
      CONT
    CASE 8
    ~MFREE(globalname%,lg%)
    CONT
  CASE 9
    '
  ENDSELECT
  PRINT "End of program. Return code : ";ind|
  PRINT "Free memory : ";FRE(0)
  DELAY 2
  SYSTEM
RETURN
PROCEDURE lit8svx(name$,register|)
  '
  ' Instrument from SMUS file.
  ' name$      : name of the instrument.
  ' register| : index of the instrument
  '
  '
  WHILE NOT EXIST(name$)
    ALERT 0,"Instrument not
found|"+name$,2,"Quit|Select",a%
    IF a%=1
      freesmus(0)
    @closeaudio(1)
    ELSE
      FILESELECT "Selection an IFF
instrument","Select","Amiga_Music:Songs/",name$
    ENDIF
  WEND
  ' Choose instrument ...
  OPEN "I",#2,name$
  get4chars(2)
  '
  ' Is file inIFF-8SVX format
  ' If not end program
  '
  IF a$<>"FORM"
    ALERT 0,name$+"Not IFF file!",0,"OK",a%
  @freesmus(0)
  @closeaudio(1)
    END
```

```
ENDIF
get4chars(2)
get4chars(2)
'
IF a$<>"8SVX"
  ALERT 0,name$+"Not IFF 8SVX format file!",0,"OK",a%
  freesmus(0)
@closeaudio(1)
  END
ENDIF
get4chars(2)
IF a$="VHDR" ! get sample information
  get4chars(2)   ! pass chchunk
  '
  oneshothisamples%(register|)=CVL(INPUT$(4,#2))
  '
  repeathisamples%(register|)=CVL(INPUT$(4,#2))
  '
  ' Number of sample per cycle
  samplesperhicycle%(register|)=CVL(INPUT$(4,#2))
  '
  ' length of sample (sample per second)
  samplespersec&(register|)=CVI(INPUT$(2,#2))
  '
  ' Number of octaves for the instrument
  ctoctave|(register|)=INP(2)
  '
  ' scompression| is a local procedure
  scompression|=INP(2)
  '
  ' Intensity of the instrument
  volume%(register|)=CVL(INPUT$(4,#2))
ENDIF
'
get4chars(2)
'
' Ignore the  BODY
WHILE a$<>"BODY"
  skipchunk(2)
  get4chars(2)
  WEND
  '
  ' Length of CHYNK
  longchunk%=CVL(INPUT$(4,#2))
  '
  ' No future form of compression possible
  IF scompression|
    templiste%=MALLOC(longchunk%,&H10000)
    IF templiste%=0
    @freesmus(0)
    @closeaudio(1)
    ENDIF
  ENDIF
  '
  v_length%(register|)=0
  hisamples%=oneshothisamples%(register|)+
repeathisamples%(register|)
  i|=0
  '
  '
  WHILE i|<ctoctave|(register|)
    ADD v_length%(register|),SHL(hisamples%,i|)
    INC i|
  WEND
  '
  ' Allocation of memory for the instrument
  '
  audioliste%(register|)=MALLOC(v_length%(register|),2)
  '                                          - MEMF_CHIP
!
  IF audioliste%(register|)=0
    IF templiste%
      ~MFREE(templiste%,longchunk%)
    ENDIF
  @freesmus(1)
  @closeaudio(1)
  ENDIF
  '
  ' Routine "optimized " for changing sample
  '
  nbpasses%=longchunk% DIV 512
  reste%=longchunk% MOD 512
  FOR i&=0 TO nbpasses%-1
    BGET #2,audioliste%(register|)+512*i&,512
  NEXT i&
  IF reste%
    BGET #2,audioliste%(register|)+512*nbpasses%,reste%
  ENDIF
  CLOSE #2
  '
  '
  ' Fibonacci Delta decompression is necessary (rare for
  ' instruments)
  IF scompression|
    BMOVE audioliste%(register|),templiste%,longchunk%
    dunpack(templiste%,longchunk%,audioliste%(register|))
    ~MFREE(templiste%,longchunk%)
  ENDIF
RETURN
PROCEDURE initsmus
  ' Initializations necessary for handling IFF-SMUS files
  '
  ' regins|(track|) reserve the number for the instrument
  '  utilized
  DIM regins|(255)
  ' duration(i) reserve the duration in seconds of a note
  DIM duration(63)
  ARRAYFILL duration(),240*128
  FOR i|=0 TO 7
    FOR j|=0 TO 1
      FOR k|=0 TO 3
        ind|=i|+j|*8+k|*16
        ' round duration
        duration(ind|)=duration(ind|)*2^(-i|)
        IF j|
          ' Note
          duration(ind|)=duration(ind|)*1.5
        ENDIF
        SELECT k|
        CASE 1
```

```
          ' Third
          duration(ind|)=duration(ind|)*2/3
        CASE 2
          ' Fifth
          duration(ind|)=duration(ind|)*4/5
        CASE 3
          ' Seventh
          duration(ind|)=duration(ind|)*6/7
        ENDSELECT
      NEXT k|
    NEXT j|
  NEXT i|
  '
  ' Frequency of note
  ' the notes  0 ` 127 correspond to
  ' MIDI systems.
  DIM note(127)
  note(33)=55 ! la0 = 55 Hz
  '
  ' Calculate the octave change plus the precision
  FOR i|=45 TO 127 STEP 12
    note(i|)=note(i|-12)*2
  NEXT i|
  '
  note(21)=55/2
  note(9)=note(21)/2
  '
  ' Notes 0 to 8
  '
  FOR i|=8 DOWNTO 0
    note(i|)=@flat(note(i|+1))
  NEXT i|
  '
  ' Calculate octave per octave
  FOR i|=10 TO 127
    IF (i|-9) MOD 12
      note(i|)=@sharp(note(i|-1))
    ENDIF
  NEXT i|
RETURN
```

```
PROCEDURE readsmus(name$)
  '
  ' name$ : name of the IFF-SMUS- file
  '
  LOCAL i|
  '
  ' If file does not exist branch to catcherr routine
  '
  OPEN "i",#1,name$
  '
  get4chars(1) ! Read the  Header IFF
  '
  IF a$<>"FORM"
    ALERT 0,"Not IFF file IFF !",0,"OK?",a%
  @closeaudio(1)
  ENDIF
  get4chars(1)
  get4chars(1)
  IF a$<>"SMUS"
    ALERT 0,"Not IFF SMUS file!",0,"OK",a%
  @closeaudio(1)
  ENDIF
  '
  get4chars(1)
  IF a$="SHDR"
    '
    ' Read the Score HeaDeR (entire partition)
    '
    get4chars(1)
    tempo%=INP(1)*256+INP(1)  ! Assurance that tempo>32767
    volume|=INP(1)
    ' nbtracks| : number of tracks from file
    nbtracks|=INP(1)
    ' tracks%() : length of track
    ' events%() : address of track
    DIM tracks%(nbtracks|-1),events%(nbtracks|-1)
  ENDIF
  get4chars(1)
  '
  ' Read the instruments
' It is possible to create a IFF-SMUS file without
' instruments
'
WHILE a$<>"TRAK"
  SELECT a$
  CASE "INS1"
    INC nbins|
    tsize%=CVL(INPUT$(4,#1))-4
    register|=INP(1)
    type|=INP(1)
    IF type|=0
      regins|(nbins|-1)=register|
    ENDIF
    dat1|=INP(1)
    dat2|=INP(1)
    '
    ' Read the instrument
    ' use default instrument
    name$="Amiga_Music:Songs/"+INPUT$(tsize%,#1)
    lit8svx(name$,register|)
    '
    ' Set instrument volume at 64 (PLAY LOUD)
    volume%(register|)=64
    '
    IF ODD(tsize%)
      ~INP(1)
    ENDIF
  DEFAULT
    ' no TRACK no INS1 : ignore the chunk
    skipchunk(1)
  ENDSELECT
  ' Read chunk
  get4chars(1)
WEND
'
'
FOR i|=0 TO nbtracks|-1
  '
  '
  nbevents%=CVL(INPUT$(4,#1))
```

```
    tracks%(i|)=nbevents%
    events%(i|)=MALLOC(nbevents%,&H10000)
    IF events%(i|)
      ' Change the track in memory
      BGET #1,events%(i|),nbevents%
    ELSE
      CLOSE #1
      freesmus(0)
    @closeaudio(1)
    ENDIF
    IF i|<nbtracks|-1
      get4chars(1)
    ENDIF
  NEXT i|
  CLOSE #1
  ' Initialize the tables  waves%() (dynamic) and
numoctave|() (static)
  initoctaves
RETURN
PROCEDURE getsid(track|,VAR ind%)
  ' Read the chain Sound event IDentifier
  '
  sid|=BYTE{events%(track|)+ind%}
  se_data|=BYTE{events%(track|)+ind%+1}
  ADD ind%,2
RETURN
PROCEDURE getnote(track|,VAR ind%)
  DO WHILE ind%<tracks%(track|)
  @getsid(track|,ind%)
    SELECT sid|
    CASE 129
      ' Change instrument : Change volume also?
      regins|(track|)=se_data|
    CASE 132
      ' Change the volume
      volume%(regins|(track|))=se_data| DIV 4
    CASE 136
      ' Change the  tempo of the partition
      tempo%=tempo%+(se_data|+1)/128
    ENDSELECT
```

```
      ' note
    LOOP UNTIL sid|<=128 OR sid|=255
  RETURN
  PROCEDURE allocchord(VAR pointer%)
    pointer%=MALLOC(98,&H10000)
    IF pointer%=0
    @freesmus(0)
    @closeaudio(1)
    ENDIF
    chordstruct(pointer%)
  RETURN
  PROCEDURE freechord(pointer%)
    ~FreeMem(pointer%,98)
  RETURN
  PROCEDURE getchord(track|,pointer%)
    LOCAL pointer2%
    chordstruct(pointer%)
    DO
    @getnote(track|,ind%)
      BYTE{sid%+lenchord%}=sid|
      BYTE{se_data%+lenchord%}=se_data|
      '
      ' duration of indices 6 bits
      duration|=SHR|(SHL|(se_data|,2),2)
      '
      ' The duration in seconds depends on the tempo
      FLOAT{duration%+8*lenchord%}=duration(duration|)/
  tempo%
      INC lenchord%
    LOOP WHILE BTST(se_data|,7) AND ind%<tracks%(track|)
    '
    IF BTST(se_data|,6) AND sid|<>255
      allocchord(pointer2%)
      ind%=LONG{pointer%+94}
    @getchord(track|,pointer2%)
      link_mus(track|,pointer%,pointer2%)
      freechord(pointer2%)
    ENDIF
  RETURN
  PROCEDURE link_mus(track|,ptr1%,ptr2%)
```

```
    chordstruct(ptr1%)
    duration2%=ptr2%
    sid2%=ptr2%+80
    ABSOLUTE lenchord2%,ptr2%+90
    ABSOLUTE ind2%,ptr2%+94
    '
    FOR n|=0 TO lenchord%-1
      FOR p|=0 TO lenchord2%-1
        IF BYTE{sid%+n|}=BYTE{sid2%+p|}
    .     FLOAT{duration%+n|*8}=FLOAT{duration%+n|*8}+
FLOAT{duration2%+p|*8}
          BYTE{events%(track|)+ind2%-(lenchord2%-p|)*2}=255
        ENDIF
      NEXT p|
    NEXT n|
RETURN
PROCEDURE playchord(nbchan|)
  nbchan|=MIN(4,nbchan|)
  FOR m|=0 TO nbchan|-1
    IF vol&(m|)
      oneshot%=SHL(oneshothisamples%(ins|(m|)),
oct|(m|)-1)
      long%=MUL(cyc&(m|),len%(m|))
      IF long%<=oneshot%
        playnote(m|,wf%(m|),long%,1,per&(m|),vol&(m|))
      ELSE
        '
        ' play oneshot in entirety
        playnote(m|,wf%(m|),oneshot%,1,per&(m|),vol&(m|))
        ' Calculate the number of cycles
        SUB cyc&(m|),DIV(oneshot%,len%(m|))
        '
        ADD wf%(m|),oneshot%
        '
        ' play repetitive parts
        playnote(m|,wf%(m|),len%(m|),cyc&(m|),per&(m|),
vol&(m|))
      ENDIF
    ELSE
      '
      playnote(m|,wf%(m|),len%(m|),1,per&(m|),0)
    ENDIF
  NEXT m|
RETURN
PROCEDURE playsmus
  LOCAL i|,j|,pointer%
  '
  ' These variables serve to pass parameters
  DIM wf%(10),len%(10),cyc&(10),per&(10),vol&(10),
ins|(10),oct|(10)
  DIM ind%(nbtracks|-1),encore!(nbtracks|-1)
  ARRAYFILL atteint!(),FALSE
  DO
    fini!=TRUE
    channel|=0
    FOR i|=0 TO nbtracks|-1
      instr|=regins|(i|)
      encore!(i|)=(ind%(i|)<tracks%(i|))
      IF encore!(i|)
        '
        allocchord(pointer%)
        ind%=ind%(i|)
      @getchord(i|,pointer%)
        chordstruct(pointer%)
        ind%(i|)=ind%
        '
        lenchord%=MIN(lenchord%,4)
        IF BYTE{sid%}=128
          '
          samplespercycle%=samplesperhicycle%(instr|)
          periode%=447
          cycle%=clock%*(FLOAT{duration%})/
(samplespercycle%*periode%)
          wf%(channel|)=LONG{waves%+instr|*32}
          len%(channel|)=samplespercycle%*cycle%
          cyc&(channel|)=1
          per&(channel|)=periode%
          vol&(channel|)=0
          INC channel|
        ELSE
```

```
          FOR j|=0 TO lenchord%-1
            IF BYTE{sid%+j|}<255
              '
              octave|=numoctave|(instr|,BYTE{sid%+j|})
              '
samplespercycle%=SHL(samplesperhicycle%(instr|),
(octave|-1))
              '
              periode%=@freq(note(BYTE{sid%+j|}),
samplespercycle%)
              '
              cycle%=FLOAT{duration%+j|*8}*
note(BYTE{sid%+j|})
              '
              wf%(channel|)=LONG{waves%+instr|*32+
(octave|-1)*4}
              '
              ' number of sample for one vibration
              len%(channel|)=samplespercycle%
              '
              cyc&(channel|)=cycle%
              per&(channel|)=periode%
              '
              ' Modulation for a specific instrument
              ' The value 0 is placed for the  volume of
              '  the instrument
              vol&(channel|)=volume%(instr|)
              '
              oct|(channel|)=octave|
              ins|(channel|)=instr|
              INC channel|
            ENDIF
          NEXT j|
        ENDIF
        freechord(pointer%)
        fini!=FALSE
      ENDIF
    NEXT i|
    IF channel|
      playchord(channel|)
    ENDIF
  LOOP UNTIL fini!
RETURN
PROCEDURE freesmus(ind|)
  SELECT ind|
  CASE 0
    i|=0
    DO WHILE events%(i|)
      ~MFREE(events%(i|),tracks%(i|))
      INC i|
    LOOP WHILE i|<nbtracks|
    CONT
  CASE 1
    FOR i|=0 TO nbins|-1
      IF audioliste%(i|)
        ~MFREE(audioliste%(i|),v_length%(i|))
      ENDIF
    NEXT i|
    CONT
  DEFAULT
    IF waves%
      ~MFREE(waves%,8192)
    ENDIF
  ENDSELECT
RETURN
PROCEDURE audiostruct(iob%)
  ' utilize the structure IOAudio in GFABasic
  ' The name corresponds with a definition
  ' in C or in assembler
  '
  ABSOLUTE io_ln_pri|,iob%+9
  ABSOLUTE io_ln_name%,iob%+10
  ABSOLUTE io_mn_replyport%,iob%+14
  ABSOLUTE io_mn_length&,iob%+18
  ABSOLUTE io_device%,iob%+20
  ABSOLUTE io_unit%,iob%+24
  ABSOLUTE io_command&,iob%+28
  ABSOLUTE io_flags|,iob%+30
  ABSOLUTE io_error|,iob%+31
```

```
    ABSOLUTE ioa_allockey&,iob%+32
    ABSOLUTE ioa_data%,iob%+34
    ABSOLUTE ioa_length%,iob%+38
    ABSOLUTE ioa_period&,iob%+42
    ABSOLUTE ioa_volume&,iob%+44
    ABSOLUTE ioa_cycles&,iob%+46
  RETURN
  PROCEDURE chordstruct(chord%)
    ' Structure of a chord
    ' in C is :
    ' struct Chord {
    '                   float   ch_Duree[10];
    '                   UBYTE   ch_ Sid[10];
    '                   LONG    ch_Len;
    '                   LONG    ch_Indice;
    ' };
    duration%=chord%
    sid%=chord%+80
    ABSOLUTE lenchord%,chord%+90
    ABSOLUTE ind%,chord%+94
  RETURN
  PROCEDURE dunpack(source%,n%,dest%)
    ' Decompress the file using Fibonacci Delta
    ' decompression method
    ~@d1unpack(source%+2,n%-2,dest%,BYTE{source%+1})
  RETURN
  PROCEDURE get4chars(ind|)
    ' Read 4 characters
    a$=INPUT$(4,#ind|)
  RETURN
  PROCEDURE skipchunk(ind|)
    LOCAL l%,i%
    ' Ignore chunk and  consequences
    l%=CVL(INPUT$(4,#ind|))
    '
    ' Attention to pad-byte
    IF ODD(l%)
      INC l%
    ENDIF
    FOR i%=1 TO l%
    VOID INP(#ind|)
  NEXT i%
RETURN
PROCEDURE playnote(channel|,wf%,len%,cyc&,per&,vol&)
  ' Procedure by Rob Peck adapted to GFABasic
  IF channel|<0 OR channel|>3
    PRINT "False channel"
  ELSE
    iob%=@getiob(channel|)
    IF iob%
      initblock(iob%,channel|)
      ioa_data%=wf%
      ioa_length%=len%
      ioa_cycles&=cyc&
      ioa_period&=per&
      ioa_volume&=vol&
      beginio(iob%)
    ENDIF
  ENDIF
RETURN
PROCEDURE initblock(iob%,channel|)
  ' Procedure by Rob Peck adapted to GFABAsic
  audiostruct(iob%)
  io_unit%=unit%(channel|)
  ioa_allockey&=allockey&
  io_command&=3
  io_flags|=&H10
RETURN
PROCEDURE reemployiob
  ' Procedure by Rob Peck adapted to GFABasic
  ' The WriteMsg Port is ignored by the program
  FOR i|=0 TO 3
    mp%=replyport%(i|)
    iob%=GetMsg(mp%)
    WHILE iob%
      ~@freeiob(iob%,i|)
      iob%=GetMsg(mp%)
    WEND
  NEXT i|
RETURN
```

```
PROCEDURE initoctaves
  LOCAL i|,j|,instr|,frequence,savart,indice|,longwave%
  DIM numoctave|(255,127)
  '
  ' Corresponds to table waves%(255,7)
  waves%=MALLOC(8192,&H10000)
  IF waves%=0
    freesmus(0)
  @closeaudio(1)
  ENDIF
  FOR i|=0 TO nbins|-1
    instr|=regins|(i|)
    ' Frequency of the not plus loudness
    frequence=samplespersec&(instr|)/
samplesperhicycle%(instr|)
    '
    ' (index 69, 440Hz)
    savart=LOG10(frequence/440)*1000
    '
    ' index of the note(301 savarts/octave at 12
    ' demi-tones per octave)
    indice|=INT(12*savart/301)+69
    '
    FOR j|=indice| TO 127
      numoctave|(instr|,j|)=1
    NEXT j|
    '
    FOR j|=indice|-1 DOWNTO 0

numoctave|(instr|,j|)=MIN(ctoctave|(instr|),(indice|-j|)
DIV 12+2)
    NEXT j|
    '
    ' waves%(instr|,0)=audioliste%(instr|)
    LONG{waves%+instr|*32}=audioliste%(instr|)
    ' Length of the sample per octave 1
    longwave%=oneshothisamples%(instr|)+
repeathisamples%(instr|)
    FOR j|=1 TO ctoctave|(instr|)-1
      '
      ' Address of the wave form
LONG{waves%+instr|*32+j|*4}=LONG{waves%+instr|*32+(j|-
1)*4}+longwave%
      longwave%=SHL(longwave%,1)
    NEXT j|
  NEXT i|
RETURN
PROCEDURE catcherr
  PRINT "SMUSPlayer: ";ERR$(ERR);" ... Error !"
  PRINT "Closing the program ..."
@closeaudio(1)
RETURN
PROCEDURE finit
  PRINT "SMUSPlayer: Interrupted ..."
  finishaudio
@freesmus(0)
@closeaudio(1)
RETURN
PROCEDURE traceur
  PRINT TRACE$
  REPEAT
  UNTIL MOUSEK
RETURN
FUNCTION getiob(channel|)
  reemployiob
  FOR k|=0 TO audbuffers%-1
    IF NOT inuse!(k|)
      inuse!(k|)=TRUE
      audiostruct(audbuffer%(k|))
      io_device%=device%
      io_mn_replyport%=replyport%(channel|)
      io_mn_length&=k|
      io_ln_name%=globalname%
      RETURN audbuffer%(k|)
    ENDIF
  NEXT k|
  iob%=MALLOC(68,&H10000)
  IF iob%=0
    RETURN 0
    ELSE
      audiostruct(iob%)
      io_device%=device%
      io_mn_replyport%=replyport%(channel|)
      io_ln_name%=dynamicname%
      io_mn_length&=dynamix%(channel|)
      INC dynamix%(channel|)
      RETURN iob%
    ENDIF
  ENDFUNC
  FUNCTION freeiob(iob%,channel|)
    audiostruct(iob%)
    IF io_ln_name%=dynamicname%
      ~MFREE(iob%,68)
      IF dynamix%(channel|)
        DEC dynamix%(channel|)
      ENDIF
      RETURN 0
    ENDIF
    IF io_ln_name%=globalname%
      i&=io_mn_length&
      IF i&<audbuffers%
        inuse!(i&)=FALSE
      ENDIF
      RETURN 0
    ELSE
      PRINT "FreeIOB : No corresponding name... Error
unknown"
      RETURN -1
    ENDIF
ENDFUNC
FUNCTION freq(f%,l%)
  ' Calculate the value of the period affecting the
  ' channel and frequency
  ' and the number of samples per second
  ' f% : frequency of the note for the instrument and
  ' samplesPerSec
  ' l% : samplesPerHiCycle of the instrument
  RETURN INT(clock%/(f%*l%))
ENDFUNC
```

```
FUNCTION getclock                ! Determine frequency
  'Test the DisplayFlags of the structure GfxBase;PAL = 4
  IF WORD{_GfxBase+206} AND 4
    RETURN 3546895 ! Mode PAL
  ELSE
    RETURN 3579545 ! Mode NTSC
  ENDIF
ENDFUNC
FUNCTION dlunpack(source%,n%,dest%,x&)
  ' Function decompressing using Fibonacci Delta
  ' source% : Buffer entries
  ' n% : table of the buffer entries
  ' dest% : Sorted buffer
  ' x& : Value of the preceding char
  lim%=MUL(n%,2)
  FOR i%=0 TO lim%-1
    d|=BYTE{source%+SHR(i%,1)}
    IF AND(i%,1)
      d|=AND(d|,&HF)
    ELSE
      d|=SHR|(d|,4)
    ENDIF
    ADD x&,codetodelta&(d|)
    BYTE{dest%+i%}=BYTE(x&)
  NEXT i%
  RETURN x&  ! return calculated value
ENDFUNC
FUNCTION sharp(f)
  RETURN f*10^(25/1000)
ENDFUNC
FUNCTION flat(f)
  RETURN f/10^(25/1000)
ENDFUNC
' Data for decompression Fibonacci Delta
DATA -34,-21,-13,-8,-5,-3,-2,-1,0,1,2,3,5,8,13,21
```

Chapter 5

Music Software

Chapter 5 Music Software

In this chapter we'll discuss some of the music software that is currently available. You'll find general information about the programs and an overview of their commands and functions. With this information you should be able to determine what kind of capabilities each program offers.

5.1 Sonix

The Sonix program from Aegis is a high performance music synthesis and composition program. With this program it's possible to combine instruments and digitize AudioMaster II files. Sonix also contains a built-in MIDI controller and supports 8 MIDI voices.

With Sonix, the notes are displayed on the screen as they are entered on the staff. This way editing music can be done quickly and easily. Basically Sonix's setup consists of three screens: Score, Keyboard and Instruments. All of these contain their own menus and controls. The mouse is used for almost all the functions in Sonix. The left mouse button is the Selection button and the right mouse button is the Menu button.

5.1.1 Before you begin

Before starting the program, make a backup copy of the master disk. Now, using the backup copy diskette, follow these steps:

1. Turn on your system. Load Kickstart (Amiga 1000 only) or wait for the icon to appear on the screen requesting the Workbench diskette (Amiga 500 or 2000).

2. When the icon appears requesting the Workbench diskette, insert the Sonix diskette in the drive.

3. The Sonix program should start automatically. If not (e.g., if you have an older version of Sonix), the Workbench screen will appear. Double-click the Sonix disk icon.

4. A requester containing information about Sonix may appear. If so, click on the Ok gadget.

5. The Sonix Instruments window will appear.

6. Access the Screens menu to access the other parts of the program.

Generating a data diskette

You should use a data diskette to store the scores and instruments you need. With Sonix you can work with one or several disk drives. Obviously it's easier to work with two drives.

There are two ways to create a data diskette. Either use a formatted blank diskette or recopy instruments from the original data diskette onto another diskette.

Normally you should use the second method when working with Sonix. The following steps show you how to create a data diskette that contains all the instruments of the original diskette.

1. Copy the original data diskette.

2. Keep the Instruments and Scores drawers and their contents in the backup data diskette, but delete everything else from this backup data diskette.

3. Erase Jays Songs from the Scores drawer.

Now you can save your music on this diskette. To avoid any problems, load the instruments from the data diskette before you start to compose.

5.1.2 Menu overview

Sonix's three screens are Score, Keyboard, and Instruments. The Instruments screen appears when you start Sonix. In this section we'll present a brief overview of the screens and their menus.

In order to display a screen's menu, press the Menu button (right mouse button) and then move the cursor across the menu bar. When a menu isn't displayed, the title of the current score or instrument is displayed in the title bar.

Score Screen

This screen contains four menus:

Screens: By using this menu you can move among the screens.

```
Screens        Project       Edit      Option


Score          ------------ Displays the staff
Keyboard       ------------ Displays Keyboard screen
Instruments    ------------ Displays Instruments screen
```

Screens menu and its items

Project: This menu contains commands for copying scores between memory, disks and screen, deleting from disk, returning to previous instrument settings, and exiting the program.

```
Screens        Project       Edit      Option


     New              ---- New score
     Load             ---- Load a score
     Save             ---- Save a score in progress
     Save As...       ---- Save score under a new name
     Revert           ---- Load previously saved data
     Delete           ---- Delete score from diskette
     Print            ---- Print score
     About SONIX      ---- Display copyright window
     Quit             ---- End session
```

Project menu and its items

Edit: This menu contains commands for editing your scores.

```
Screens      Project      Edit      Option


     Cut              ---- Cut out a part of the score
     Copy             ---- Copy a part of the score
     Paste            ---- Restore a part of the score
     Clear            ---- Erase a part of the score
     Play             ---- Listen to a part of the score
     Repeat           ---- Repeat a part of the score
     Octave Up        ---- Raise part of a score by one
                           octave
     Octave Down      ---- Lower part of a score by one
                           octave
     Half Step Up     ---- Raise part of a score by one
                           half tone
     Half Step Down   ---- Lower part of a score by one
                           half tone

             Edit menu and its items
```

Option: This menu provides settings for determining the key and time signatures of a score and adjusting the volume of instruments.

```
Screens      Project      Edit      Option


     Signatures   ---- Time and key signature definition
     Mix Down     ---- Voice mixing requester


          Option menu and its items
```

Keyboard Screen

This screen contains two menus:

Screens: This menu allows you to move among the screens.

```
Screens      Keyboard


Score        ------------ Displays the staff
Keyboard     ------------ Displays Keyboard screen
Instruments  ------------ Displays Instruments screen


             Screens menu and its items
```

Keyboard: This menu contains the Instrument command, which enables you to open different instruments for the keyboard.

```
Screens      Keyboard


     Instrument -- Defines instrument used
     Volume     -- Adjusts instrument volume


                  Keyboard menu and its items
```

Instruments Screen

This screen contains three menus:

Screens: This menu allows you to move among the screens.

```
Screens      Project      Waveforms


Score        ------------ Displays the staff
Keyboard     ------------ Displays Keyboard screen
Instruments  ------------ Displays Instruments screen


             Screens menu and its items
```

Project: This menu contains commands for copying instruments between memory and a disk.

```
Screens       Keyboard       Waveforms

     New             ---- New score
     Load            ---- Load instrument
     Save            ---- Save instrument in progress
     Save As...      ---- Save instrument under a new name
     Revert          ---- Load previously saved instrument
     Delete          ---- Delete instrument from diskette
     Quit            ---- End session

              Project menu and its items
```

Waveforms: This menu contains different waveforms that can be used to change the sound of an instrument on the synthesizer.

```
Screens       Keyboard       Waveforms

     Square          ---- Add square wave
     Sine            ---- Add sine wave
     Triangle        ---- Add triangle wave
     Ramp Up         ---- Add upward sweep to wave
     Ramp Down       ---- Add downward sweep to wave

              Waveforms menu and its items
```

5.1.3 Tips and tricks

As with many programs, you may have some difficulties performing certain tasks. The following are some tips for helping you with some of the problems you may encounter.

Changing instruments

After an instrument has played its part in a composition, you may want to change to another instrument. In order to do this you must load the data for the new instrument and place it in front of the first measure of the appropriate track.

The Copy command from this program not only focuses on notes but also on the data of the instrument that is playing them. Because of this, the same instrument is called upon several times during a score. So if you modify the first call, the next call will produce a change. For this reason you should only include the notes when copying sequences.

Selecting isolated instruments

If you want to load the data for a specific instrument in order to hear its timbre, remember that the time needed to do this will increase depending on how much data is located on the diskette. We recommend that you place the instrument that you're testing in any track you want. The Amiga will enter it as an instrument you're using. Once you've done this you can quickly reload the data from RAM at any time.

Echo

For an instrument with a long echo (rate 4), the note that follows the one being played will cut off the end of the echo. If you're using one or several tracks you can avoid this by using each track individually. In other words, the first track plays the first note of the sequence, the second track plays the second note, the third track plays the third one, and the last track plays the fourth. Then, during the fifth note, the first track will play again. Usually this allows enough time to hear the echo completely.

You may have to experiment with this until it works. However, you'll find that an instrument with a long echo gives depth to the score.

Using IFF instruments

If you use instruments that you've carefully sampled, you probably won't hear anything when you press the keys on the keyboard, even if you have a MIDI keyboard.

This happens because the Amiga places IFF instruments, which you've sampled, in the highest octave. So a normal keyboard cannot reproduce them. Since these sounds can only be heard in the highest octave, they should only be used there. It's especially difficult to hear percussion instruments.

However, for a melodious sound, you should sample a range covering three octaves with a frequency signal in the middle position. Sampling on an octave generates a recording frequency signal at the same time as a reproduction frequency at the lowest rate. While recording, be sure to use a sampling frequency that's high enough.

Try this experiment for percussion sounds: After using a note in the highest audible octave, place a note in another octave (inaudible). Generally you'll receive a very interesting echo effect when it is played back.

Frequency reduction

If you use many sampled sounds, you'll notice that you can't play some of the sounds included on the Sonix diskette in lower registers. For example, let's discuss the "Strings" sound. Unfortunately, this sound can only be used in the lower octaves. Using it in the upper register only produces a crackling sound at best. This happens because the sound was sampled so high that when it is used it is played in inaudible ranges.

A way to avoid this is to use a digitizer program, such as Perfect Sound, which enables you to reduce frequency. Try doing the following:

Load the sound digitizer program. Choose the Load Dump command for sounds in Sonix format, or Load IFF for IFF sounds (e.g., Electric Piano). After reading the directory, select the Instruments subdirectory (this may take awhile to do). Click on the STRINGS.SS sound and load it into memory.

Remember that sounds in the Sonix format must end in the "SS" suffix. IFF sounds, however, begin with "IFF" and end with the "INSTR" suffix. Since the other sounds are analog, they aren't sampled.

Now that the sound is in memory, select the FREQ.=FREQ./2 command from the menu. The program will ask you for the new name of this sound, which is "STRINGS.SS". The frequency of the sound drops an octave because the oscillation period doubles. The sound appears on the screen under its new name. Simply save it in "Instruments" by selecting the Save Dump command (for Sonix sounds) or the Save IFF command (for IFF sounds). You have to reload the Sonix program for the rest of the sounds.

Copying a sound

When creating a data diskette, you must remember the following guidelines:

You can place digitized sounds on the diskette under the IFF (Interchange Format File) format or under the Sonix format. If it's an IFF sound, the filename will be "IFF Name.INSTR". "Name" refers to the name of the reproduced instrument.

The Sonix format consists of two parts. Let's use the "STRINGS" sound again as an example. "STRINGS.SS" contains data that applies to the sampling. However, "STRINGS.INSTR" contains surrounding curves, modulations, etc. This is the file modified by the preceding program. The sampling data remains unchanged.

Remember that if you copy sound data under the Sonix format, you'll have to re-copy both files that are related to the instrument. For IFF sounds, one file is sufficient.

There is also a third kind of sound. These are sounds generated by Sonix but which aren't sampled (e.g., "SYNTH-BASS"). These sounds can be identified by the ".INSTR" suffix and by their size of 512 bytes. Sonix may even re-copy these sounds.

5.2 AudioMaster II

One of the most powerful and easy-to-use programs for sampling and editing stereo sound is AudioMaster II from Aegis. This program, which works with any Amiga sampler, extends the hardware's capability. The only thing that limits AudioMaster II's capabilities is the sampler and Amiga hardware that are used.

AudioMaster II contains many features that make it an excellent program. Because AudioMaster II allows you to set parameters and save them to a configuration file, you can customize the program to fit your own needs. AudioMaster II also enables you to select a range of the sound displayed, and increase its size so that it fills the entire screen. This make editing samples easy.

One of this program's most important features is its Seek Zero and Seek Loop functions. These functions simplify selecting loops, which are sections of a sample that are repeated when you want a sample sustained. These functions attempt to find the best looping points, which can be a very difficult task if you try setting your own loop points "by ear."

AudioMaster II also contains sound effects capabilities. By using various sound effects, you can add some creativity to your music. The program is capable of playing sounds backwards, adding echo, tuning the sample, changing the volume, and filtering out high frequencies.

Even though AudioMaster II has many impressive features, many of its capabilities are not completely compatible with the IFF standard. So AudioMaster II samples cannot be used in many music programs.

However, AudioMaster II will accept samples from other digitizing programs, such as Perfect Sound (more on this product later), and will allow you to save your samples in either Sonix or IFF format. Sonix format lets you load this information into Sonix, and IFF format lets you load this information into some other IFF compatible music programs.

5.2.1 Before you begin

Before starting the program, make a backup copy of the master diskette and data diskette. Now, using the backup master diskette, follow these steps:

1. Turn on your system. Load Kickstart (Amiga 1000 only) or wait for the icon to appear on the screen requesting the Workbench diskette (Amiga 500 or 2000).
2. When the icon appears requesting the Workbench diskette, insert the AudioMaster II diskette in the internal drive and the AudioMaster II data disk in the external drive.
3. The Workbench screen will appear. Double-click the AudioMasterII disk icon. When the AudioMasterII disk window opens, double-click the AudioMasterII icon.
4. A requester containing information about AudioMaster II may appear. If so, click on the OK gadget.
5. The Aegis AudioMaster II window will appear.
6. Press the right mouse button to view the menu titles in the menu bar.

5.2.2 Menu overview

The following is a list of some of the items found in AudioMaster II's menus. Refer to the manual for a complete listing.

Project Menu

About...
This item displays information about AudioMaster II.

Load...
This item is used for loading samples.

To Edit Window
This submenu item loads the selected waveform file into the main waveform area and displays the waveform in the Edit window.

To Copy Buffer

This submenu item loads the selected waveform directly into the copy buffer.

RAM Scan

This submenu item shows what is currently in memory and what was in memory before you began to use AudioMaster II.

Save

This item saves data to diskette.

Waveform

This submenu item saves the entire waveform.

Ranged Data

This submenu item saves the range selected in the Edit window.

Save Configuration

This submenu item enables you to save to disk the various settings you've selected.

Edit1 Menu

Cut

The Cut item enables you to cut and remove a previously defined range. This range is temporarily stored in the copy buffer if the Enable Cut To Buffer function from the User Options menu is activated. If this isn't activated, the data will be lost. Also, if the removed data exceeds 100 kilobytes, it won't be saved in the copy buffer.

Note: You'll know when the removed data is saved because the screen will blink briefly.

Copy

The defined range is copied directly into the intermediate copy buffer. You can place the copied range in another part of the wave using the Paste command. You can also mix the copied range with other waves by using the Mix Waveforms... command found in the Special FX menu.

Paste
This item inserts the data from the copy buffer into the Edit window at the current cursor position.

Clear Buffer
This item allows you to erase the contents of the copy buffer. However all the data located in the buffer will be lost.

Replace
Replace allows you to insert the data, which is in the copy buffer, at the current cursor location.

Zero...
With the Zero item you can fill the data in the copy buffer or the data ranged on the screen with zeros. You should do this when you need portions of data to equal 0 (e.g., when you want to add space to a waveform).

Invert...
This item inverts the waveform contained in the range or copy buffer.

Swap Chan...
This item lets you swap channel information on a stereo sample.

Edit Freehand
With Edit Freehand you can draw or modify a defined waveform, which consists of a series of dots. By gently moving the cursor and pressing the left mouse button, you can modify the waveform. However., you should only concentrate on a small area of the display. The part you have created can be copied and inserted later.

Clear all Data
This item resets the program. So by using this function you can erase all the data relative to the waves in memory. If you accidentally activate this function you can retrieve your data by using RAM Scan. However, this only works with the first sampled wave because it is placed in the RAM chip. Unfortunately RAM Scan doesn't recognize memory extensions.

Edit2 Menu

Cursor to Start
This item moves the cursor to the beginning of the waveform.

Cursor to End
This item enables you to move the cursor to the end of the waveform.

Replicate Loop
This item enables you to duplicate a range selected for a loop.

Add Workspace
This item adds memory for waveform editing.

Swap Buffer & Main
This item swaps the contents of the Edit window for the contents of the copy buffer.

Special FX Menu

Echo...
To create an echo effect, you must first range a desired area. You'll hear the echo at the remaining part of the waveform. You can also use the Zero item (Edit1 menu) to add some empty space at the end of the waveform; this increases the echo effect.

Echo Rate
> The Echo Rate slider gadget lets you define the repetition speed of the echo in 60ths of a second. The echoes are superimposed on each other (echo on echo on echo). Once you have defined the pitch of the echo, you can no longer modify it.

Decay Rate
> The Decay Rate slider gadget allows you to determine the speed at which the echo will subside. If you use a value of 40, for example, the subsequent echo will be 40% quieter than the previous one. If you use the maximum value, AudioMaster II will use the repetition speed (Echo Rate) and the size of the wave to define the extent of the echo.

Number of Echoes

This gadget determines the number of successive echoes.

Backwards

This item allows you to reverse a range or the copy buffer.

Mix Waveforms...

This item lets you mix the main waveform with the one in the copy buffer. Two identical waves or two different waves may be mixed in this item.

Volume %

This defines the volume level of the wave, in the copy buffer, that will be added to the main waveform.

Flange

This modifies the pitch of the wave in the copy buffer by changing the speed. For example, if you increase the speed, the effect is a lower tone. In other words, the result corresponds to an echo placed a fraction of a second after the end of the wave.

Change Volume

This item allows you to modify the volume of a range. You may choose from a field from 0 to 200% (0 = no volume, 200 = volume twice as loud). Always begin by setting the volume at 100% (Start Volume %) and finish with 0% (End Volume %).

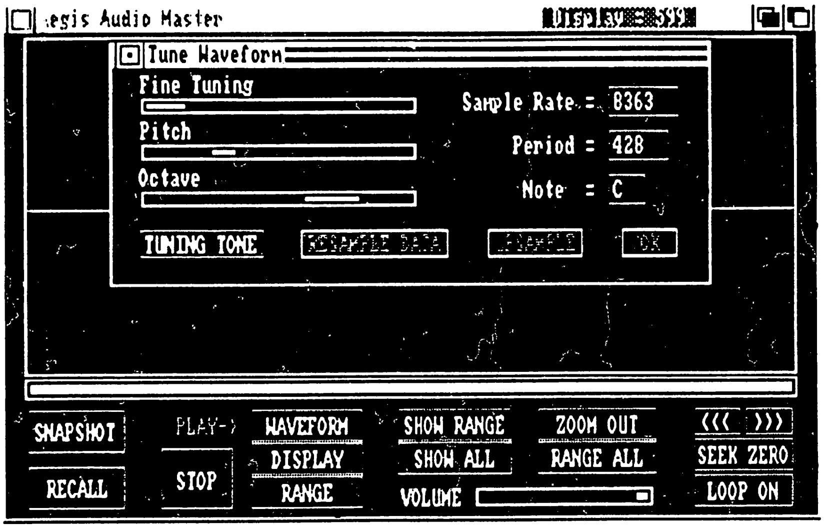
Tune Waveform

This is a special function that enables you to play back a waveform at greater than 29K sample speed. This is quite an achievement since it has never been possible to do this on the Amiga: The Amiga's hardware is restricted to 29K speed.

The tuning controls in the Tune Waveform menu enable you to adjust the pitch of the sound. If you choose the Play Waveform item before selecting Tune Waveform, you can modify the pitch of the sound while you're listening to it.

UPSAMPLE

Clicking this gadget reduces the wave by one half. The waveform will sound a full octave higher.



Sample Rate = 8363
Period = 428
Note = C

User Options Menu

In this menu, you'll find several options that help you simplify your work with AudioMaster II. Some of these options are already selected by the program. If you don't want to use an option, selecting it will switch it off.

Enable OK Prompts
This function helps minimize mistakes because it displays the prompt, "Are you sure?" when using functions that may permanently alter memory. However, if you're familiar with the program and this bothers you, switch off this function.

Enable Mix Pre-Scan
When you mix two different waves, you might get a distortion because the result could exceed the maximum values supported by the sound chip. So a compensation is required. With Pre-Scan you can determine how much compensation is needed.

Enable Cut to Buffer
When you cut data with the Cut function, AudioMaster II automatically places them in the copy buffer. However, this will only work if the cut part is under 100 kilobytes. If you want to keep the data that is already located in the copy buffer, disable the Enable Cut To Buffer function.

Enable Loop Limit
This item toggles the default loop limit of 128K.

Enable Ramp Pre-Scan
This item toggles clipping of waveforms that go beyond maximum limits.

Enable Audio Filter
This item toggles the audio filter found on Amiga 500s and Amiga 2000. This audio filter controls sound quality.

Co-ordinate Display...
This item specifies display coordinates.

Set Snapshot Default...
With this item you can choose to which device (external RAM, etc.) your snapshot will save. You can choose DF0:, VD0:, and RAM disk. Make your selection by clicking on the name of the appropriate device.

Set Screen Colors...
This item specifies screen colors.

Set Sample Config...
This item configures sampling information to your sampler.

Set Type of Waveform
When you choose this option, a window called "Set Type Of Waveform" appears on the screen. You can then choose between IFF and Sonix formats. Samples Per Cycle (SPC), such as 8, 16, 32, etc. are automatically set to play in the proper octave for both IFF and Sonix formats.

If the wave is reproduced in an incorrect octave, adjust the values (i.e., double the value for a higher octave). Do the opposite if you want a lower octave. To change this value, click in the edit field using the mouse pointer. The old values can be erased by using <Del> or <Backspace>.

IFF waveforms include information about the sampling rate at which they should be played. AudioMaster II recognizes and reproduces them. However, Sonix doesn't contain this information. Because of this, AudioMaster II selects a sampling degree of 8363 SPC. So before working with AudioMaster II, save a file as IFF unless it is intended for Sonix.

HiFi Menu

This menu allows saving and playing of a waveform in high-fidelity format.

5.2.3 Tips and tricks

The following are some guidelines you can use when working with AudioMaster II.

Sampling with AudioMaster II

Sound sampling has been very popular among Amiga users. Because of this popularity, there is now enough software and hardware available to make any Amiga owner satisfied.

For our examples, we used the Golem digitizer, which is equipped with a frequency generator that significantly improves the sound quality (beyond 20,000 Hz). We connected it to an Amiga 1000 at the parallel port, leaving the joystick port free. Our digitizer is also equipped with a potentiometer and an LED screen. This makes it much easier to perform input signals. Since the Golem digitizer has a metal case, it won't be affected by interference.

Since the digitizer comes with a long cable, it can be placed beside the Amiga. For other models, such as the Amiga 500 and Amiga 2000, you must use another cable, which you connect to the joystick port. The electrical current reaches all of the hardware through this cable.

Digitizing hardware often comes with a microphone and basic software for recording and sound playback. If your hardware is equipped with a parallel port and a joystick port, you'll be able to use it with AudioMaster II.

We chose this program because it is not only one of the best for sampling but it also has many editing capabilities. With AudioMaster II it's easy to generate digitized sound of professional quality, with a minimum of cash outlay.

Parallel Port

Sample Rate
With Sample Rate, there is a sample speed of 8363 sps, which is suitable for a large number of samplings. Many instruments can be used, which is suitable for sound demos.

With this kind of sampling, you'll have a longer recording time because the number of bytes used per second will be less. Approximately three minutes of quality music can be saved on a diskette.

Sometimes the sampling speed must be much greater: 19886 sps. Because of this it's possible to achieve a richly detailed wave, which produces a clear sound reproduction.

Waveforms that require extensive work should always be numbered at this speed. This enables you to modify little details. Sounds sampled using this mode have such a high quality timbre that they sound like a HiFi recording performed in a studio.

Joystick Port

You should avoid hardware that only uses this port because it doesn't produce very high quality sounds. A sampling of a spoken word might use a speed of 8363 sps but that's the limit of this kind of machine's capabilities. Occasionally it's possible to assign words to a speed of 14914 sps, which provides a better reproduction.

Sampling Options

Before sampling, you must define the sampling speed and the size of the piece in bytes. The speed, size, and reproduction time are closely related. Reproduction time is defined according to the following formula:

```
Size in bytes
 ------------------------= Reproduction time in seconds
Sampling speed (SpS)
```

If you want to give a certain reproduction duration to a sampling, multiply that duration by the sampling speed to get the number of bytes needed.

```
Reproduction time x Sampling speed  = Size in bytes
```

If you want to use a well-defined sampling speed and size (in bytes), you'll have to work within the following limits:

The highest frequency you can sample is equal to half of the sampling speed. This means that when you're working in a fast sample speed on the parallel port, you can sample frequencies of up to 10 KHz. If you detect a higher frequency during the sampling, you'll hear a kind of sound overload. It is, however, possible to get a higher sampling speed by using a memory extension capable of up to 8.5 megabytes. More significant waves, such as sampled pieces of music, require more time for continuous editing.

Sample Size

The Sample Size gadget defines the amount of memory that is used for the sampling. This is repeated on the screen. If you push the setting to the maximum, all the available memory will be used.

One of the advantages of AudioMaster II is that it can execute large samplings by using a memory extension. However, AudioMaster II only recognizes the largest part of monoblock memory. So the sampling may be smaller than you expect.

An auto-configured memory extension may present a problem because the sampled piece must be in only one memory block. So if you have two memory extensions of 2 megabytes each, 4 megabytes will be identified for sampling space. But you can really only use 2 megabytes. The reason for this is that there are several recognition bytes at the beginning of the second block. These bytes are needed for managing an operating system. So think of AudioMaster II as having one memory space that is divided into two.

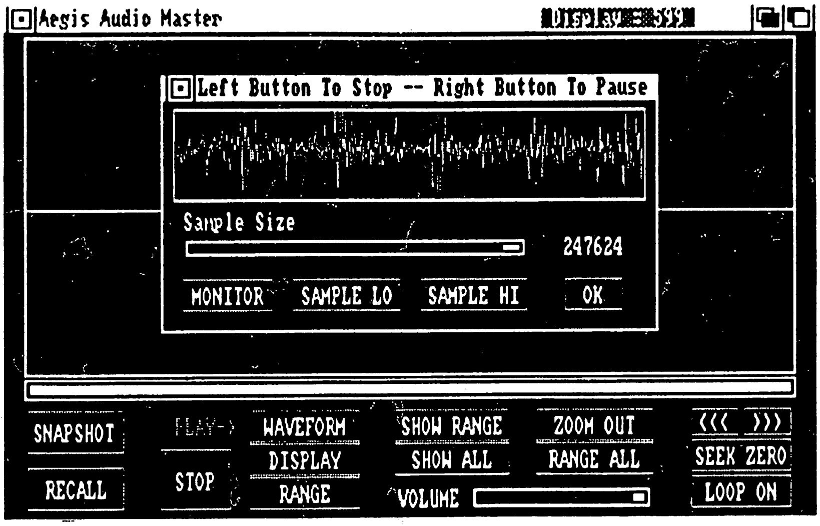
You could ask your computer dealer if there is any way to combine these two extensions. Even if you have memory extensions that aren't auto-configured, you still must ensure that the free memory isn't separate. To do this, use the Addmem command from the Shell. Don't forget that if you work with a large memory extension it will take more time to make changes.

The Oscilloscope

This function enables you to see the incoming sound data. The Monitor button allows you to see the sound levels of the incoming sound in the window of the requester.

To pause this display, click the right mouse button. By quickly clicking this button several times you can check for "clipping", which can distort the sound. You can detect this by watching several seconds of the wave on the screen.

The oscilloscope is very important for testing incoming signals because it allows you to detect a saturation point (clipping). If you discover clipping, adjust the volume on the sampling hardware.



Information on Sounds

Sampled sounds are divided in two categories:

1. Sounds that have an overall pitch, for example a violin, trumpet, guitar, etc.

2. Pieces of music made up of more than one pitch over the duration of the sound, for example "Pump up the Volume" by M.A.A.R.S. These are called "events".

In a program that plays music, the events are played back at the same frequency as they were recorded. However, before playing it back, you must place the events at the middle position on the staff.

When recording instruments this is very important because AudioMaster II allows you to regulate the pitch of the sound later. You should always try to tune a sound at a median value so that other programs will be able to reproduce the event.

Recording Sounds

There are two ways to record sounds. You can either use a microphone or a stereo connected to the digitizing hardware. Actually there is also a third way to record sounds - by using a MIDI interface - but we won't discuss this at the moment.

Using the Memory

Don't use a very large sampling size because a waveform takes up a large amount of memory. This is true even if you use a memory extension.

You can try to sample to the acceptable limit, but most editing functions won't work beyond 100 Kbytes. You would need a very large memory extension. Almost all the instrument waves are located 10 Kbytes below the main octave. Waveforms saved as 5-octave instruments will be eight times larger than the original since each octave doubles in size as the octave drops.

Starting the Sampling

Now let's try a sampling. We'll show you everything you need to do for a good sampling. First copy the diskette that contains the AudioMaster II program and format several blank diskettes.

Perhaps you want to sample directly from your synthesizer or another instrument, such as your stereo or television. If so, make sure that you have all the proper cables.

You should choose a quiet place for your sampling. While using a microphone, noises from a distant train or the street can ruin your recording of, for example, the sound of a fly. So select your location carefully. If your cables aren't protected, you may receive interference from electrical appliances, such as fans, etc.

Clean all the cable connections with cotton and alcohol. This significantly improves the quality of the data transmission.

To keep background noises at a low level, set the input volume on the digitizer at a low level and the output volume of the keyboard or sound source at a high level. Remember that you're using a stereo system.

Since keyboards, electric guitars, and other electronic instruments have a low level output signal, we recommend that you use an amplifier. On the other hand, the sampling has to be high enough to avoid all saturation. If you have hardware that is capable of special effects, you can link it directly to the digitalizer. You may get some strange sounds.

First switch on your Amiga. If you have an Amiga 1000, insert the Kickstart diskette. Then insert the AudioMaster II diskette. Start the program by clicking on its icon twice.

When AudioMaster II is loaded, the editing window and the command menu will appear on the screen. Choose the Sampler option in the Project menu. You can define the sampling type (either Sample LO or Sample HI) and its size.

Next click on the Monitor box. By adjusting the volume of the stereo installation, you can see how the wave loses some of its size. If you click on the right mouse button, the curve will stand still.

The pause function allows you to verify whether the settings are correct so you can avoid saturating the sampling process. If you press the left mouse button, you'll leave the Monitor mode.

Click once in the Sample HI box. The is the same as activating the Monitor mode, except that if you click in this box again the sampling process will begin. The screen becomes grey. Once the defined memory space is full of music or sounds that you want to sample, the screen returns to its original color. The captured wave is displayed on the screen.

By selecting the Play Waveform or Display option, you can listen to the sampled piece. If you want to see the reproduced sound on the waveform, select the Play Display option. A horizontal white line will move from left to right on the editing screen. It shows the place on the curve where the sound is currently located. If you click on the Stop box, this "cursor" stops at the place where the sound has stopped. If you select the Loop On option the wave will continually repeat itself at the lower right of the control menu.

If the sound quality isn't good enough, use the options found in the Special Effects menu to modify the volume. The Change Waveform option provides three settings with corresponding data. By slowly moving them towards the right, you can modify the sampling speed. Begin the entire sampling again. The sound will definitely improve.

Reproducing Excerpts

To reproduce a portion of your sampling you must define the excerpt you want to hear. To do this simply use the mouse. Move the cursor to the place where the excerpt should begin, press the left mouse button and move the mouse to the right or left. The defined range appears with a white background. Select the Play Range option located under Play Waveform. Only the defined range will be played and the cursor will indicate, on the curve, the sound you're hearing.

Editing Sounds

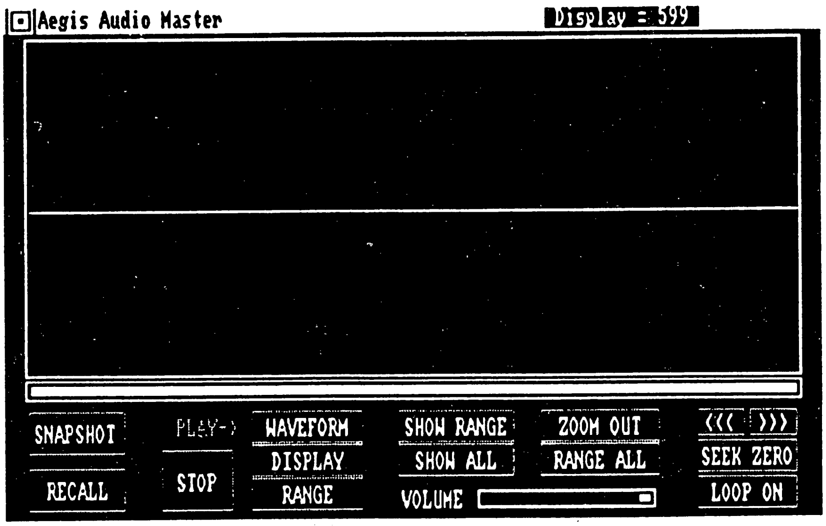
First load the sound you want to modify by using the Load option from the Project menu. Again, you have two choices: Load to Screen Display and Load to Copy Buffer. Select the first option.

You'll soon see the waveform appear in the editing window. Define the first quarter of it as an excerpt and place it in the intermediate Copy Buffer. To do this use the Copy function from the Edit menu.

Before you can edit a waveform you must define a section or range on which you can work. To do this, click on the mouse button and drag the cursor to the section of the wave that you want to edit and release the mouse button. The space that you defined appears in white. To clear this range simply click anywhere else in the editing window.

Use the Show Range function to magnify the range you selected; simply press the Show Range button. Continue to do this until you see 599 bytes indicated at the top of the edit window.

When you increase the size of the waveform, you'll see that it is made up of dots. To display the rest of the waveform, drag the scroll bar at the bottom of the screen. You'll see the following options:



Zoom Out
This function has the opposite effect of Show Range. The defined range's size is reduced.

Show All
This command returns the waveform to its normal display so that the entire waveform is shown in the Edit window.

Range All
If this function is selected, a menu command will affect the entire contents of the Edit window. So the entire window is included in the range. Anything outside the window isn't included.

Loop
When the Loop function is active you'll see two red vertical bars on both sides of the editing window.

By using the Play Waveform and Loop On options, you can continuously reproduce the defined piece. To stop editing, you must click in the Stop field.

Waveform
The Waveform option allows you to play the entire wave. As you're listening to it, you can adjust the Repeat Points, Volume, and Pitch.

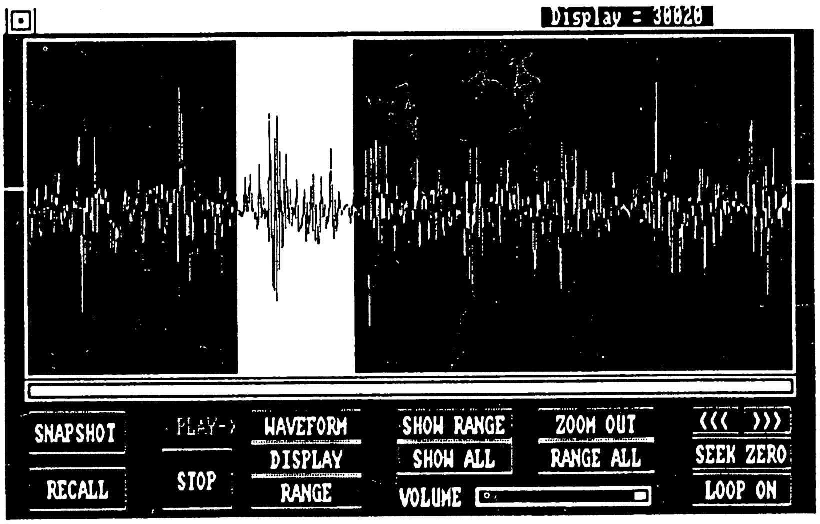
Display
This function plays only what is currently on the Edit window. A cursor, which moves across the window, indicates which part of the waveform is currently playing. If you interrupt the sound with Stop, the cursor will remain where the sound stopped.

Range
If a range is set and you select this command, only the ranged data is played.

Seek Zero
Seek Zero looks for the nearest zero crossover point (no volume) for the currently active repeat marker. This way, you can find the optimum break point during a Loop function. You can then get a clear repetition loop without any interference. If this doesn't work on the first try, keep trying.

Now let's add an echo to the excerpt. Select the corresponding option in the Special Effects menu. A window containing three settings, Echo Rate, Decay Rate, and Number of Echoes, will appear.

The first option allows you to define the speed at which the echo will be repeated. The values listed correspond to equal steps of a 60th of a second. The second option, Decay Rate, indicates the speed at which the echo will die. If, for example, you use the value 40, the following echo will be 40% less than the preceding one. The final option, Number of Echoes, defines the number of echoes.



We won't change the first two settings but we'll set the number of echoes to "1". Now with the mouse press the Echo button.

AudioMaster II asks you a security question such as: "Are you sure?" which you answer by pressing the OK button. After you've done this, AudioMaster II indicates, in a window, that it is working. When this window disappears, your excerpt will contain an echo.

Once again click in the Play Waveform option so that you can hear this echo. You can also reverse the defined excerpt. Choose the Backwards option from the Special Effects menu. Again, AudioMaster II asks you whether you're sure you want to do this. This function operates very quickly.

The process for reversing is the same as the previous procedure. A window appears, indicating that the work has been done; then it disappears. Next you'll hear the beginning of the piece played backwards but it will have a normal echo. As long as the excerpt is still on the screen, it can be returned to its normal direction by using Backwards.

Now we're going to mark the rest of our piece of music. This will give us a chance to experiment with the Flange function. First empty the Copy Buffer. Then place the marked part of the music into it.

To do this, begin by using the Clear Copy Buffer function from the Edit menu. Then choose the Flange option from the Project menu and move the settings completely to the right. You'll notice the Flange effect; it determines the pitch of the data. Once you've set everything, click in the Mix option from the Flange window at the bottom left. After asking you the same security question, to which you answer OK, AudioMaster II will take a few seconds to perform the work.

Actually AudioMaster II is performing a series of calculations. The bigger the curve, the longer these calculations will last, even up to a few minutes. When the calculation is completed, you can listen to the result on the entire piece.

AudioMaster II also lets you load a sound from a game diskette or a diskette with sound sources. But most people don't know where to get files that contain sound sources. Look at the directory of all the programs on a diskette. You may see an IFF file that contains sound sources. For certain games, for example, you could replace a simple whistle with a mighty roar. You'll also find a lot of sound sources on public domain diskettes. They are generally saved according to the IFF standard and can be used or even modified for your own scores.

For unfamiliar sounds, you might have to use the Tune Waveform function. You can set the register and the sampling speed. The Save Ranged Data function allows you to move the sound back in its place. In this way you'll have diskettes with your own personalized sounds.

5.3 Dynamic Drums

Dynamic Drums is an easy-to-use music program that is suitable for all Amiga owners who love to create music. This program enables you to create your own studio quality drum tracks or load any preset drum rhythms, such as rock, funk, jazz, country and Latin styles. Dynamic Drums includes one hundred drum samples that can be assigned to the Amiga keyboard. It's also possible to incorporate your own Amiga standard IFF sound samples.

This program enables users who don't have the ability or the dexterity of an experienced drummer to express their own rhythmic abilities. Users can create and edit drum rhythms one beat at a time. Dynamic Drums automatically fixes errors as you play.

The program includes a cassette with step-by-step instructions with examples of various techniques, such as laying down a basic drum track and advanced syncopation techniques.

With its studio features, such as fully adjustable volume and tuning levels and a randomizing option that varies these levels, you can create a more natural effect. So, by using Dynamic Drums you can easily create professional sounding drum tracks.

5.3.1 Before you begin

Before starting the program, make backup copies of Disk No. 1 and Disk No. 2. To start Dynamic Drums, follow these steps:

1. Turn on your system. Load Kickstart (Amiga 1000 only) or wait for the icon to appear on the screen requesting the Workbench diskette (Amiga 500 or 2000).

2. When the icon requesting the Workbench diskette appears, insert the diskette marked Disk No. 1 in the internal drive and the diskette marked Disk No. 2 in the external drive.

3. The Workbench screen will appear. Double-click the DD1 disk icon. When the DD1 disk window opens, double-click the Dynamic Drums icon.

4. The Dynamic Drums window will appear.

5. Press the right mouse button to view the menu titles in the menu bar.

5.3.2 Menu overview

The following is a list of some of the commands found in Dynamic Drums' menus. Refer to the manual for a complete listing.

Song Menu

A song is an arrangement of patterns. The Song menu allows access to song files.

Load Song
This item loads a song from diskette. It displays a file selector box from which you can select a song. Dynamic Drums may also ask you to load another drumkit if the song uses a different drumkit from the one currently in memory. You should click on the YES gadget because the song may not sound the same with the current drumkit.

Save Song
This item saves the song in memory to diskette.

Delete Song
This item deletes a song from diskette.

Clear Song
This item clears song data from memory. This option erases not only the text in the song window but also patterns A through J.

Filter Toggle
This item toggles the audio filter found on Amiga 500 and Amiga 2000 models.

Save Config.
This item saves the current Dynamic Drums configuration.

Once you have loaded a song, click on the PLAY gadget in the Song window or press the <F2> key. Click on PLAY again (or press the <F2> key) to stop playing the song.

Pattern Menu

Patterns are short rhythms that can be combined to make a song. This menu contains many of the same options as the Song menu.

Load Pattern
This item loads a pattern from diskette. It displays a file selector box from which you can select a pattern.

Save Pattern
This item saves the pattern in memory to diskette.

Delete Pattern
This item deletes a pattern from diskette.

Clear Pattern
This item clears pattern data from memory.

Copy Pattern
This item clears pattern data from memory. A requester appears, asking you to select the pattern to which you want the current pattern in memory copied.

The Graphic Pattern Display, which is a collection of blue and white dots on a black background, appears in the pattern window. This display is similar to a graph: Beats (time) are represented horizontally and the number of the drum that is currently playing is represented vertically. Click on the PLAY gadget, which is located above the Graphic Pattern Display. To stop the pattern, click on the STOP gadget.

The patterns are stored in pattern banks A-J, which are displayed at the top of the pattern window. In order to record your own pattern, you must clear the existing pattern in the currently selected bank. To do this, select the Clear Pattern option. Then click on the RECORD gadget and enter your pattern. The metronome keeps track of the number of beats per minute. You can switch this on or off. When it's on you'll hear a click that's beating time.

Drumkit Menu

A drumkit is a collection of coordinated drum samples that are used to create rhythms.

As soon as the program loads, you can select a drumkit. In order to do this, select any of the drumkits from the large requester box that appears on the screen. Then click on the OK gadget and the loading will begin. This might take one or two minutes depending on the drumkit.

When the drumkit is loaded, each of the drums are assigned a key on the numeric keypad. The names are displayed in the Drum Keypad window. The numeric keypad also contains options such as Repeat <.>, Accent <Enter>, and Alternating Accent <->.

This menu enables you to load and save drumkits.

Load Drumkit
This item lets you load a drumkit from diskette.

Load Drum
This item lets you load a single percussion sound or other IFF file from diskette for addition to a drumkit.

Save Drumkit
This item lets you save a drumkit to diskette.

Delete Drumkit
This item lets you delete an existing drumkit from diskette.

5.3.3 Tips and tricks

MIDI Interface

If you have a MIDI interface you'll be able to do many more things with this program. If you activate the Sync In item beforehand, you can write directly in a pattern using the instrument. This makes composing with Dynamic Drums much easier. If you don't have a MIDI interface, just ignore this menu.

What is a MIDI interface?
Hardware that is capable of digitization and MIDI communication are

frequently used with the Amiga. Music programing and digitization can provide more than just entertainment.

It doesn't matter whether you're a beginner or an expert. You can still find a program that will suit your needs. Today software and hardware are so advanced that almost anyone can program music on the Amiga.

Installing a MIDI interface or a digitizer is actually quite simple. In fact, these two pieces of equipment are placed in the serial interface connector and parallel interface connector respectively.

Some of these devices also have switches or potentiometers which allow you to choose and regulate the signals. A normal piece of equipment works with a resolution of 8 bits, which is sufficient for most people's needs. For professional support systems, this equipment works on 16 bits and allows for a frequency resolution of up to 100 KHz. This is usually too complicated for most users.

Adjustments aren't needed for the MIDI interface. You just have to ensure that the interface and instrument are linked together using one or two cables.

Through this kind of interface, you can connect up to 16 MIDI devices to the Amiga. The computer then becomes the master keyboard. But the Amiga can also be used as a special effects device for a MIDI keyboard.

Basically the MIDI interface allows you to link computers and musical instruments together. But it's also possible to link the instruments to each other. In these instances there is a master device and one or more devices under its command. In other words, any piece played by the master instrument can be immediately repeated on the instruments under it.

If the master instrument commands several other devices, each of these can be addressed through a particular channel. If you have a good distribution of the different voices, you can produce a new interpretation for any piece of music.

Loading and Generating a Drumkit

The first time you load Dynamic Drums, it asks for a drumkit. This provides a set of numbers for the instruments. If you don't necessarily want to keep this particular set of drums, you can load new ones.

First, in the Drumkit menu, select the Load Drum option. A dialog window appears, allowing you to load the existing instruments on the diskette.

Now we'll discuss how to organize the drums. You'll see that the basic structure of a group of drums is often identical. There is usually a bass drum and a snare drum. We suggest that you load the BassDrum in number "0".

Note: You should always define the bass drum as being in number 0 and the snare drum in number 1. This will enable you to access these instruments easily when you need them instead of having to look for them.

When you compose a group of instruments, always try to keep those with a low pitch together and those with a high pitch together. This is important for creating a group of instruments that creates sounds in all pitches and effectively utilizes stereo capabilities.

Remember to distribute the instruments over all four channels to avoid saturation. By using this same division or group of instruments you can define the sounds that are played loudly and those played in the background.

Loading Other Instruments

If you have different sound sources located on several diskettes, it's possible to load other instrument sounds.

To do this, place the diskette that contains your sources in the drive. Select the Load Drum item. Dynamic Drums will display the following:

```
WRONG DISK
```

Click OK and change the name of the diskette in the dialog window by entering DF0: or DF1:, whichever is appropriate. Then Dynamic Drums loads the contents of the main directory of the diskette in the drive. Simply select the instrument you want.

Generating a Beat

Although the instruments you want to use are re-grouped in a drumkit, you still must generate a true beat.

An easy way to generate beats is to use rhythms that already exist. You already have a kind of beat in the Pattern window but it isn't from your composition.

You'll have to add some instruments in order to create a piece of music. A tune that is composed of only two instruments is boring and lifeless.

When you want to generate your own beats, simply start the metronome and click on the RECORD gadget on the Pattern window. Then, with the help of the mouse, position the instruments you want. An instrument can also play directly from the numeric keypad.

Repetition and Mixing

When a pattern is finished, first recopy it in the Pattern window that is available. To move from one pattern to another, click on the corresponding letter.

Since the beat you chose earlier is still in each pattern, you should change it only slightly. For, example, you might have a guitar solo accompanied by a sax and then a drum solo accompanied by a horn. If you carefully combine instruments, you can create brilliant sounds. By creating patterns, you can compose a piece of music.

Since you don't have to adhere to a succession of patterns, you can really use your imagination. By combining patterns freely, you can sequentially reproduce each one several times. It's best to enter these combinations directly in the Song window so that you can save your work.

If you want to do more than simply generate beats and songs, you can also record in stereo, which will make your work sound even more professional.

Now you need to save the song and drumkit. The values defined for range, the channels used, and the volume are automatically saved. Each option can be saved separately. For Save Song, information about the drumkit and pattern is also saved. Later we'll discuss how to save each element separately.

Speed and Tempos

If you have predefined tempos or rhythms, you can change them by adjusting the performance speed. To do this, slightly move the Tempo setting in the Pattern window. Shifting it towards the right increases the performance speed.

The modified speed is shown in the lower right corner, under the letters BPM (Beat Per Minute). The snare drum, for example, can execute the beat. If you're using the metronome, a clicking sound will repeat the beat. This helps you compose music while keeping correct time.

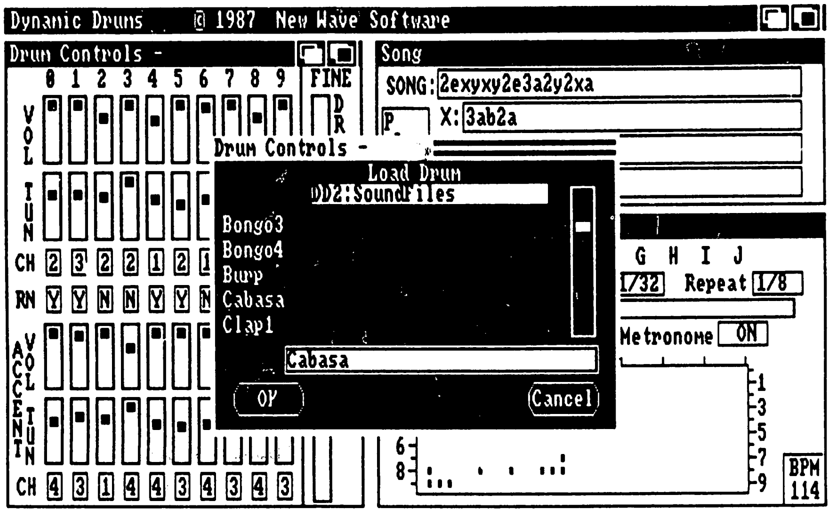
If you think you can compose music without using the metronome, make the beat visible by placing dots in the Pattern window. This will provide a random beat that doesn't respond to any defined rules.

Saving Drumkits

Once you've defined a piece of music you've composed, you must save it. Since each part of this program can be saved separately, each menu provides the elements that are needed to do this.

After you've correctly defined the drumkit, save it so that you can access it at a later time. To do this, select the Save Drumkit option.

The dialog window for this operation appears. Enter the name of the drumkit on the bottom line and press OK. All the data that corresponds to setting the mixing table will be saved. By doing this now, you won't have to redo this preliminary work when you use this drumkit again.



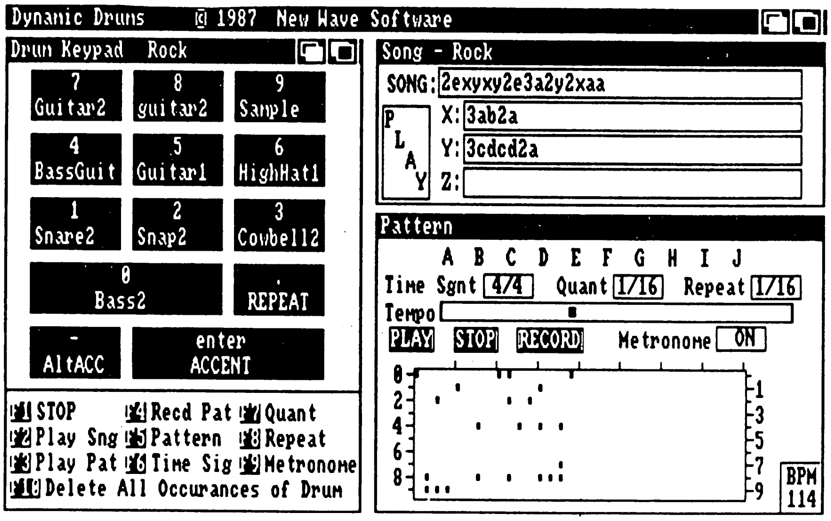
Saving Songs

The method for saving musical compositions is similar to the way drumkits are saved. However, use the Save Song option of the corresponding menu instead. Remember that when you save a piece of music using this option, the drumkit that is used is automatically saved.

So, besides musical data, Dynamic Drums also saves all the information needed to re-compose the instrumental group just as you've defined it. Later, when you load again, this is the characteristic that enables you to get all the data from the drumkit that is used.

Saving Patterns

You can save any pattern so that you can use it in another piece of music later. Each value is saved, including those that have a tonic accent. The only parameter that isn't saved is the one relative to the speed.



SONG: 2exyxy2e3a2y2xaa
X: 3ab2a
Y: 3cdcd2a
Z:

Time Sgnt 4/4 Quant 1/16 Repeat 1/16
Tempo
Metronome ON
BPM 114

More on the MIDI Interface

The MIDI interface is very practical because it enables you to use an instrument or a stereo system in order to record a pattern.

If you own this type of interface, your work with Dynamic Drums will be simplified. When a pattern is in Record mode, it's possible to make an instrument play according to the data of this pattern.

5.4 Deluxe Music Construction Set

The Deluxe Music Construction Set (DMCS) from Electronic Arts is a sound editor that can turn your computer into a desktop music publishing studio. This program includes complete input, notation and score printing capabilities. It also enables you to enter notes and then listen to them immediately.

With DMCS you'll be able to use more than 10 digitized instruments and multiple playing styles, with articulations such as staccato and legato. You can also change instruments within each staff. Other features include a dynamic range from *ppp* to *fff*, a playback speed from 1 to 240 beats per minute, and a 16 channel MIDI out to control MIDI instruments.

DMCS simplifies musical composition by allowing you to see and hear notes before placing them. You can edit music by using the Full Cut, Paste, and Copy options. DMCS also includes many musical symbols, such as guitar chords, etc.

This program's printing capabilities enable you to re-position notes and chords so that your sheet music will be easy to read. It's also possible to stretch the space within, above and below each staff.

The Deluxe Music Construction Set uses several pulldown menus and windows on the screen. These functions can be accessed with either a mouse or a keyboard. The Score window is the central part of this program. This is where you enter, play and edit notes.

There are three different ways to enter music in the Score window:

1. You can load existing pieces from a diskette and then modify them.

2. Select notes from the Note palette window using the mouse, and place the notes on the staff.

3. Select the Insert Notes in Score box and play notes on the keyboard using the mouse.

5.4.1 Before you begin

Before starting the program, make a backup copy of the master disk. Now, using the backup copy diskette, follow these steps:

1. Turn on your system. Load Kickstart (Amiga 1000 only) or wait for the icon, requesting the Workbench diskette, to appear on the screen (Amiga 500 or 2000).

2. When the icon requesting the Workbench diskette appears, insert the Deluxe Music Construction Set diskette in the internal drive.

3. The Workbench screen will appear. Double-click the DD1 disk icon. When the DMCS disk window opens, double-click the DeluxeMusic icon. DMCS will load into memory.

5. Press the right mouse button to view the menu titles in the menu bar.

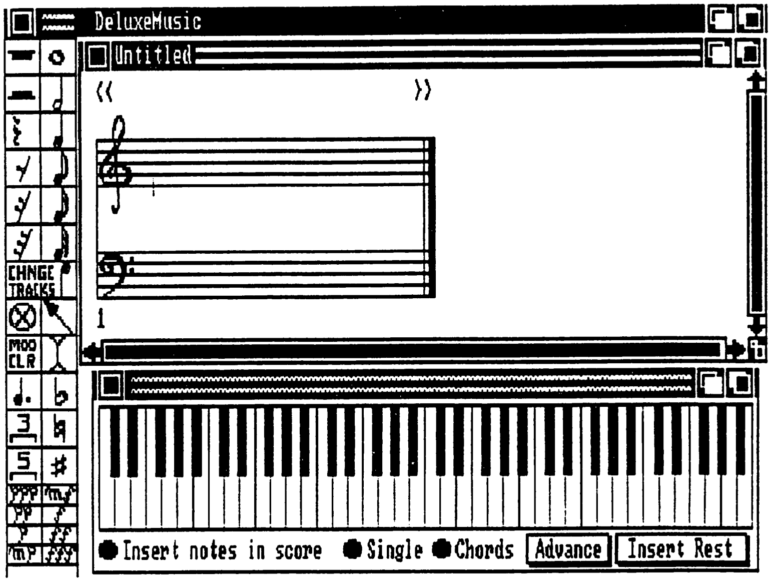
6. Double-click the DMCS icon.

5.4.2 Menu overview

The following is a list of some of the commands found in Deluxe Music Construction Set's menus. Refer to the manual for a complete listing.

File Menu

New Score
This item opens a blank, untitled score.



Open Score...
This item opens a requester that contains a directory of DMCS scores and Simple Music Interchange File Format (SMUS-IFF) files that are saved on the disk in the active drive. To open a file, click the score title you want to open and then click on Open.

Save
This item saves the current score onto a diskette. The changes are saved over the original version of the score.

Save As...
This item enables you to name a new score or rename a changed score and save it under the new name. This item is used the first time an untitled score is saved.

Revert
This item reloads the score from the disk. So the previously stored version is loaded.

Print Score...
This item prints a copy of the current score.

Print Pause
This item specifies the number of staves you want printed between pauses.

Show Memory
This item opens a window which displays the amount of memory free, clipboard size and other memory items pertaining to DMCS.

Quit
This item exits DMCS and returns you to the Workbench.

Save as SMUS file
This item saves the information in memory as an SMUS-IFF file.

Edit Menu

Undo
This item reverses your most recent action in the Edit, Notes, or Measures Menu.

Cut
This item removes all selected notes and rests from the score and puts them in the Clipboard.

Copy
This item copies all the selected notes and rests to the Clipboard. It doesn't erase anything from the score.

Paste
This item inserts the contents of the Clipboard into the score.

Clear
This item permanently removes the current section.

Select All
This item selects all the music in a score.

Window Menu

Score
This item allows you to open the Score window, which is the main composition window of DMCS.

Piano Keyboard
This item allows you to open the Piano Keyboard window, which contains a five octave keyboard. To play the notes of the piano, click on them with the pointer. This enables you to listen to a section of your composition before placing the notes in the score.

Score Setup
This item allows you to open the Score Setup window, in which you can adjust the playback volume and speed, add new staffs, modify existing staffs, and determine formatting.

Note Palette
This item allows you to open the Note Palette window, which contains notes and rests for varying durations and note modifiers (time modifiers, accidentals, and dynamic modifiers).

Play Menu

Play Song
This item plays the entire song.

Play Section
This item plays the section of the score that is marked with "Begin Section" and "End Section" marks.

Stop Play
This item stops the playback and returns the score to its normal display.

Resume Play
This item continues to play the score at the point where it was stopped.

Begin Section
This item specifies the insertion point of a section.

End Section
This item specifies the end of a section.

Flash Notes
This item instructs DMCS to flash notes (make the notes blink) as they are played.

Turn Page
This item keeps advancing the score even if the Flash Notes option isn't selected.

Player Piano
This item instructs DMCS to flash (blink) the piano keyboard keys as they are played.

Repeat Play
This item continuously plays the music until Stop Play is selected.

Notes Menu

Up Half Step
This item raises the selected notes one half step.

Down Half Step
This item lowers the selected notes one half step.

Up Level
This item raises selected notes to the next vertical position in the staff.

Down Level
This item lowers selected notes to the next vertical position in the staff.

Up Octave
This item raises the selected notes a full octave.

Down Octave
This item lowers the selected notes a full octave.

Invert Chord Up
This item raises the bottom tone of a chord by one octave.

Invert Chord Down
This item lowers the top tone of a chord by one octave.

Half Time
This item halves the duration of the selected notes and rests.

Double Time
This item doubles the duration of the selected notes and rests.

Flip Note Stem
This item toggles the directions of note stems up or down.

Set Play Style
This item opens a requester, which enables you to select a playing style for selected notes.

Groups Menu

Tie Notes (Up) & (Down)
These items connect selected notes that have the same pitch. The connected notes are played as one continuous tone.

Beam Notes
This item connects selected notes by using a black line.

Slur Notes & Slur Notes (Down)
These items slur selected notes, giving a smooth line for most instruments.

Crescendo
This item gradually increases the volume of a song.

Decrescendo
This item gradually decreases the volume of a song.

Octave Raise & Lower
These items mark the selected notes so that they are played an octave higher or lower than they were written.

Measures Menu

Set Time Signature...
This item sets a time signature for all the staffs in the score.

Set Key Signature...
This item sets a key signature that begins at the current cursor location.

Set Clef...
This item opens a requester box, which allows you to choose from four clefs.

Set Instrument
This item assigns the selected instrument in the Sounds Menu to the appropriate staff and measure.

Set Tempo
This item lets you set the tempo of the composition.

Erase Insts & Tempo
This item deletes instrument and tempo information from memory.

Insert Measure
This item inserts a blank measure before the measure that contains the cursor.

Split Measure
This item splits the current measure into two measures.

Join Measures
This item joins two separate measures into one.

Delete Measure
This item erases the measure that contains the cursor or the selected notes.

ReAlign Measure
This shifts notes proportionally in a measure.

Begin Repeat
This item inserts a repeat symbol in the current measure.

End Repeat
This item inserts an end repeat symbol in the current measure.

1st Ending
This item inserts a first ending marker in the current measure.

2nd Ending
This item inserts a second ending marker in the current measure.

Double Bar
This item inserts a double bar at the end of the measure that contains the cursor or the selected notes.

Sounds Menu

MIDI Channel...
This item opens a secondary menu, which enables you to select one of 16 MIDI channels.

MIDI Active and MIDI Input Enabled
This item switches the MIDI Port and MIDI input capability on or off.

MIDI Setup...
This item opens a requester box in which you can set the MIDI interface parameters.

Remove Instrument
This item removes the selected instrument from the Sounds menu and score.

Load Instrument...
This item opens a requester that contains a directory of DMCS instruments.

Keyboard Play Style
This item controls the playing style from the keyboard.

Fonts Menu

This menu lets you specify the font used in DCMS. The Fonts menu varies with the fonts available.

5.4.3 Tips and tricks

Loading from Diskette

With the File menu you can select the drive (DF0:, DF1:, or DH0:) from which you want to load a diskette. When you select the Open Score option from this menu, a window appears. This allows you to load existing music from a diskette. Simply click on the name of the score you want; this selection will be displayed in inverse video.

Now if you click on OK, the score you've chosen will be loaded in the memory and displayed on the staff. You can listen to it by using the Play menu.

DMCS Example

The best way to demonstrate DMCS's capabilities is with an example. Let's use the Cleopha music file found on your original DMCS diskette. Load Cleopha and you'll see the notes that appear in the window containing the staff.

Before playing this tune, be sure that both Flash Notes and Turn Pages are activated. These functions will be displayed in red letters when they are active.

While listening to the score, you should see the notes highlighted on the screen as they are played. The measures will also move along by screen page.

If you activate the Player Piano command, fingering will appear on the keyboard. However this might slow down the rhythm slightly because the fingering is displayed on the screen while you're listening to the music.

Repeat Play

The Repeat Play function allows you to define a specific section of a song and play it. This function can also be used to play a defined section repeatedly. This is useful for customizing your own compositions.

To repeat a section of a song, first you must define it by using the mouse. With the arrow from the Note Palette, click the insertion point on the first measure of the section you want to play. Then select Begin Section from the Play menu.

Next define the end of the section by clicking the insertion point on the appropriate measure. When you've done this select End Section from the Play menu.

Now select the Repeat Play option from the Play menu and choose Play Section. Your defined section will play continuously until you select Stop Play from the Play menu. The Repeat Play option only uses three staffs, leaving one free staff for a MIDI keyboard.

Speed

The speed or tempo of a song is measured in beats per minute. With DMCS you can change the tempo by using the Beats per Min control in the Score Setup window. With the mouse pointer slide the Beat per Min control to the right (speed up) or left (slow down). You can modify the tempo only for the section you've defined.

Let's use Cleopha as an example:

1. Ensure that the Score Setup window is open on your screen. If it isn't, select it from the Window menu. Currently Cleopha has a tempo of 133 beats per minute. Play the song several times to study the tempo.

2. Now move the cursor in the Beats per Min slider gadget to the 100 beats per minute setting. Play the song and listen to the difference.

3. Define a section of the song that you're going to edit. Click in the first measure and activate Set Tempo from the Measures window. Above the staff a little note appears along with a number ("100" in our example), which indicates that you set the number of beats per minute.

If, for example, you set the limits of the defined section to stop at the end of the third measure, the tempo will return to 133 beats per minute at the beginning of the fourth measure. So, DMCS will play the first three measures at 100 beats per minute and the rest of the song at 133 beats. You should experiment with this function so that you're aware of all the possibilities.

Sometimes it's difficult to distinguish any differences in the tempo if you only make slight changes. With DMCS you can switch off the sound for specific staffs. To do this, use the Staff Sound On gadget, which is located in the Score Setup window. When this gadget is deactivated, the red highlighting will switch off.

Changing the Time

DMCS allows you to change the time in any measure on any staff. Using the arrow from the Note Palette, click on the first measure of the first staff. Then select the Set Time Signature option from the Measures menu. The Set Time requester will appear on the screen.

Click on the "1" gadget of the Beats Per Bar row. Leave the "4" in the Single Beat Note row. This menu also allows you to modify the number of beats of each measure up to 99. To make these changes, you have to move the cursor from right to left or vice versa. When you have chosen the correct values, click on OK.

Changing the Key Signature

By using the Set Key Signature window in the Measures window you can change the octave being used for different measures. For example, you could change the key from E minor to C major.

Note: DMCS uses the letters A through G with their corresponding flats and sharps. We won't discuss this any further but if you need more information, refer to a music theory book.

In the Set Key Signature requester, you'll see that the default setting is C major to A minor. There are three ways you can change the key of music that has already been entered: "Don't Transpose", which leaves the notes unchanged and uses sharps, flats or naturals to keep the melody the same; "Transpose Up" and "Transpose Down", which moves the music up or down so it's written in the scale of the new key.

The key signature is written into the score as a group of sharps or flats. With the "no key change" option you can change the transposition style without changing the key signature. Once you have made your changes, confirm them by clicking on OK.

Generating Voices and Defining the Clef

Although you can use a maximum of eight staffs (voices) for a piece of music, only four of them can be played simultaneously. However, if you have a MIDI keyboard, you can play the other four instruments on it. The Add Staff gadget of the Score Setup option allows you to add new tracks.

You can also define the clef. There are four available clefs: Treble (Soprano), Tenor, Bass, and Alto. These clefs enable you to define different octaves.

It's also possible to delete voices in the same window by using the Delete Staff gadget from the Score Setup window.

Changing the Work Window

While working on a piece of music, you may need to display more white space above and below the staffs or to display more measures in the Score window.

DMCS allows you to change your working space. The Score Setup window contains two settings for this: "Space above staff" and "Space below staff".

Also, the Bars Per Line control, found in the Score Setup window, enables you to determine how many measures are visible on the screen at one time.

Simple Note Entry and Editing

To place notes on the staff, use the mouse. Simply select, from the Note Palette, the note, rest, modifier, or tool you want to use and then place it on the staff. The cursor then appears as the symbol you just selected.

When you click on the mouse button, DMCS plays the note you selected. By holding down the mouse button and moving the note on the staff, you can hear all the notes the cursor passes through. Once you release the button, the note will be dropped on the staff. If the Piano Keyboard window is opened, you'll be able to see the corresponding keys flash as you slide the note up and down.

5.4.4 More tips and tricks:

1. With the Up Half Step and the Down Half Step commands from the Notes menu, you can move a note a half step up or down.

2. Erase each note by selecting them and pressing Delete. You can also use the Note Palette eraser (this is a circle with an X in the center, similar to "⊗").

3. Place this symbol on the note you want to erase and press the left mouse button. If you've erased the wrong note, you can restore it by using the Undo function from the Edit menu.

4. You can remove a note from a chord by placing it on top of another note in the chord.

5. The Half Time and Double Time options from the Notes menu allow you to change the length of the note after it has been entered.

Merging Two Tracks

Some scores, especially ones for the piano, have different beat values (melody and accompaniment) for the same instrument. DMCS allows you to combine these tracks into one by using the Two Tracks Per Staff gadget in the Score Setup window.

In this mode, both the upper and lower tracks are used. With the Change Tracks control in the middle of the Note Palette you can quickly switch from one mode to another.

The notes entered in track one have note stems that point up and the notes entered in track two have note stems that point down.

Cut, Copy, and Paste Commands

Sometimes you may want to use repeated patterns throughout your score. The Copy and Paste commands allow you to copy such a passage and insert it in another location. The Cut command erases a passage, which it memorizes so that it can place it in another location.

To use these functions:

1. Click on the first note. Then, while continuing to press the left mouse button, move to the last note of the measure that you want to repeat.

2. To copy this area, use the Copy command from the Edit menu. The selected notes appear on the computer's clipboard.

3. Activate the Score Setup window and click the Add Staff gadget. When the third staff appears, select the arrow from the Note Palette and click the insertion point into the last staff.

4. Now select Paste from the Edit menu. A copy of the notes you selected are dropped onto the staff.

If you want to erase and copy the selected notes (i.e., completely remove a section of music and place it somewhere else), select the Cut command instead of Copy.

To copy this cut piece to one or several locations in the score, click the insertion point at the place where the selected notes should begin and then select the Paste command.

Accidentals

Accidentals are pitch modifiers called flats, naturals and sharps. In DMCS, they are located on the right side of the Note Palette, below the text insertion tool.

Let's use the G note for our example:

	FLAT	NATURAL	SHARP	
G Note	Gb	G	G#	A

1. Place any note on the staff.

2. If you take either a sharp or a flat from the Note Palette window, and click it on that note, the half tone accidental will appear on the screen.

3. To restore the note to its original tone, just take the natural symbol and click it on the note that was changed. It will resume its initial value on the staff.

If you want to modify or change all the notes, use the MOD CLR option from the Note Palette window. If you place accidentals on the clef using the key signature, they will affect all the notes written in the key signature.

Octaves, Crescendo, and Decrescendo

The Octave Raise and Octave Lower commands from the Groups menu indicate whether DMCS should play the notes an octave higher or lower than written.

To use these functions:

1. Select the first five measures of a piece of music.

2. Select the Octave Raise option from the Groups menu. If you look at the staff, you'll see that the notes have been raised an octave.

If you play this, you'll hear the difference in the first five measures. If you used the Octave Lower option instead, the first five measures would be played an octave lower.

Crescendo means that there is a progressive increase in sound. When you select this option from the Groups menu, the corresponding symbol appears below the selected notes.

When you listen to a score that contains notes that are played in crescendo, you'll notice that the sound level of the notes increases gradually. To remove a crescendo, select the Crescendo command again.

The Decrescendo command is the opposite of crescendo. The selected notes will be played with a volume that gradually decreases. Selecting this command a second time will deactivate it.

Dynamics

In addition to crescendo and decrescendo commands, you can also use the fff (the most fortissimo possible) through ppp (the most pianissimo possible) modifiers in the Note Palette. These modifiers help you shape the volume levels between and within the staff of your songs.

Repetitions

As you've already seen, you can use the Copy and Paste commands to repeat identical measures within a score. With DMCS you can also specify a passage to be repeated without actually copying the music. By using the Begin Repeat and the End Repeat commands from the Measures menu, you can repeat a selected section of music twice.

First and Second Endings

It's also possible to specify measures within a repeated passage that should be played on the first or second repeat only. To do this, use the 1st and 2nd Ending commands found in the Measures menu.

Inserting Text

With the insertion tool from the Note Palette, you can enter text at any location in the score. However, before you enter any text, click the Printer Width gadget in the Score Setup window to enlarge the onscreen score.

Chapter 6

Hardware Extensions

Chapter 6 Hardware Extensions

The circuits presented in this book are based on a MIDI interface and a digitizer. The MIDI interface enables you to connect a synthesizer, which is MIDI compatible, to the Amiga. This lets you compose pieces of music and then save them in the computer. Then you can play the pieces back on the MIDI synthesizer.

Another electrical circuit described in this book allows you to sample any sound or noise. The Amiga is capable of reproducing sounds that are generated in this way. The circuits are designed so that even beginners can handle them. However, you must know how to use a soldering iron and know something about electronics. (Editor's note: Abacus has not tested these circuits boards, so proceed with caution.)

Since there are many kinds of Amigas, we'll present the connection that is used for each type. These circuits can be adapted later for a different Amiga by changing the connection cable.

6.1 Tips for building circuits

Here are some helpful hints for assembling these and any other electronics projects at home.

When you have all the necessary components and the board for the printed circuit, transfer the layout in the correct direction on the copper side. Then place the circuit in a concentrated chloride solution to dissolve the unused copper. Once you've finished the printed circuit you must make the holes for connecting the components.

Take your time when assembling electronic equipment. Haste may lead to frustration, soldering iron burns, and a non-working project. Keep calm, and take your time.

Classify the components in the order they have to be assembled, starting with the ground and continuing with connections of wires, resistors, capacitors, diodes, and transistors. After you've done all of this, add the integrated circuits. To prevent the components from overheating, you should mount them with a heat sink or other device to draw heat away from the circuitry (ask your electronics store about heat sinks).

Also, to avoid the deterioration of the board when electricity passes through it, you should check the assembly of the different components. To do this use the following steps:

1.) Unplug your computer before connecting any new hardware. These precautions should protect both your computer and hardware extension and prevent any problems.

2.) Look for the places where the solder might make a bad electrical connection on the circuit.

3.) Verify the polarity of the diodes and electrolytic capacitors.

Note: The Abacus editors have not tested these circuit boards. So proceed with extreme caution if you attempt to build your own circuits.

6.2 MIDI interface

The following are the main characteristics of this interface:

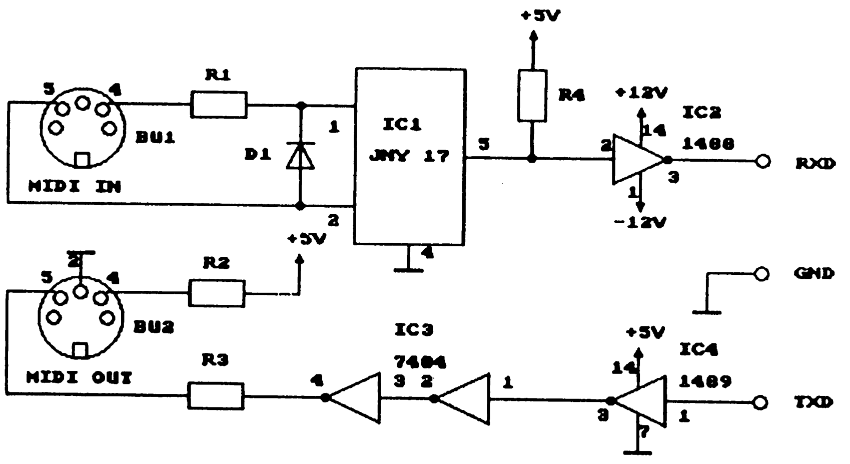
- Asynchronous serial transmission
- Transmission speed 31250 Baud
- 1 start bit
- 8 data bits
- 1 stop bit
- Power supply circuit 5 mA (Current present = 0 logic)
- One output handles an input
- One opto-coupler per input (rise and fall time less than 2ms)

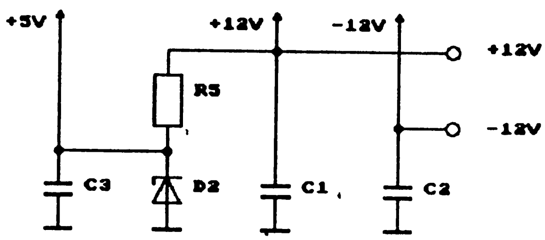
The first five characteristics are handled by programming because they are related to the highest transmission speed of the RS 232 standard. The last three conditions must be handled by the hardware.

At pins 4 and 5 of the DIN connector, the electric current has a force of 5 mA. This is executed by the IC1 opto-coupler and the R1, R2, and R3 resistors. Output is driven by two IC3 inverters. Theoretically it's possible to manage up to five MIDI OUT channels. But since most MIDI devices have a MIDI THRU connector, one MIDI OUT is all you need. The opto-coupler handles the insulation of each MIDI device.

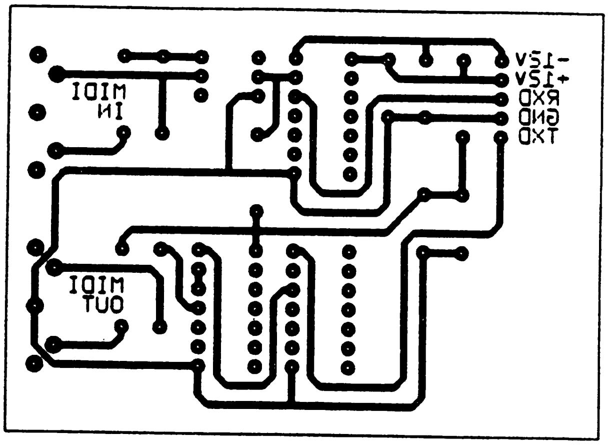
An RS 232 interface normally works with tension levels going from -12V to +12V. The levels used by the standard MIDI are type TTL 0V/+5V. So you have to adapt this current. The IC2 and IC4 convertors are responsible for this task.

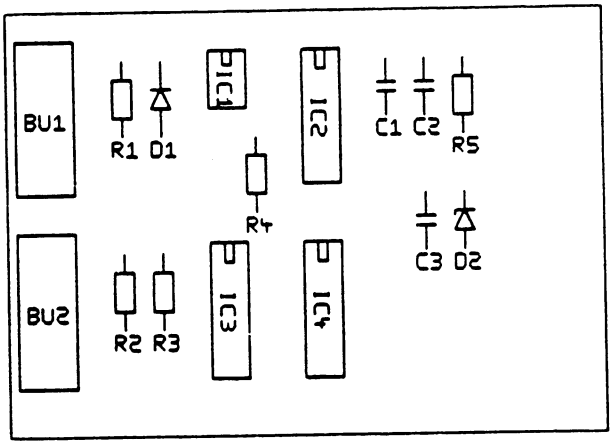
The following is the circuit diagram of the assembly so that you can see how it works:

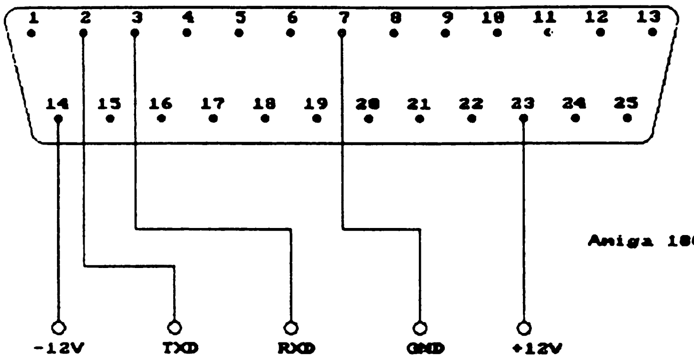


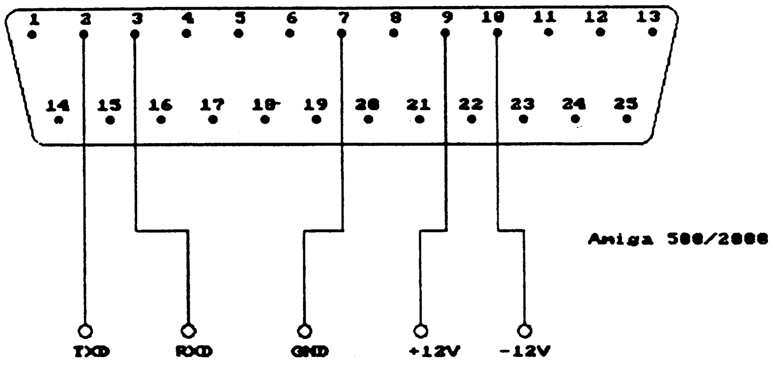


The layout of the circuit board and the diagram for setting it up are shown below. These diagrams, along with the list of components, should help you make this MIDI interface. Also shown are the connections to the Amiga.




BU1
BU2
R1 D1
R2 R3
IC1
IC2
IC3
IC4
R4
C1 C2
R5
C3 D2


Amiga 1000
-12V
TXD
RXD
GND
+12V


Amiga 500/2000
TXD
RXD
GND
+12V
-12V

List of components for the MIDI interface:

R1, R2, R3:	220 Ohm
R4:	4.7K Ohm
R5:	220 Ohm, 0.5 Watt
C1, C2,C3:	100nF
D1:	1N4148
D2:	Zener Diode 5.1V
BU1,BU2:	Female DIN 90 connector for printed circuit
IC1:	CNY 17
IC2:	1488 or 75188
IC3:	7404
IC4:	1489 or 75189

Misc:

1 foot of five-lead electrical cable
DB25 connector male pins (Amiga 1000) or female pins (Amiga 500/2000), according to the type of computer and its case

IC sockets:	1x6 pins
	3x14 pins

6.3 Sound digitizer

The digitizer converts analog information (e.g., from a stereo) into 8 bit digital data. In order to do this, we use an expansion card, in which the sound enters through an RCA jack (analog input) and, after being processed, is sent via a cable of wires and a submini connector (digital output).

We'll briefly discuss this system. The assembly can be divided into 5 parts: the filter, Sample/Hold module, frequency generator, negative voltage generator, and analog/digital convertor.

Because of the R2 resistor and the C5 capacitor, a low frequency limit of 4000 Hz is generated. This filter should allow frequencies, which are equal to only half of the digitizing frequencies, to be passed. This helps you avoid the aliasing frequencies.

Basically, the Sample/Hold module keeps the input signal constant during the conversion process. The T1 transistor acts as a switch for the input signal. The C7 is an intermediary buffer. The base of T1 is linked, by the R5 resistor, to the Busy signal of the IC1 convertor chip. If this chip is in a wait state, the Busy signal is at a high level, enabling T1.

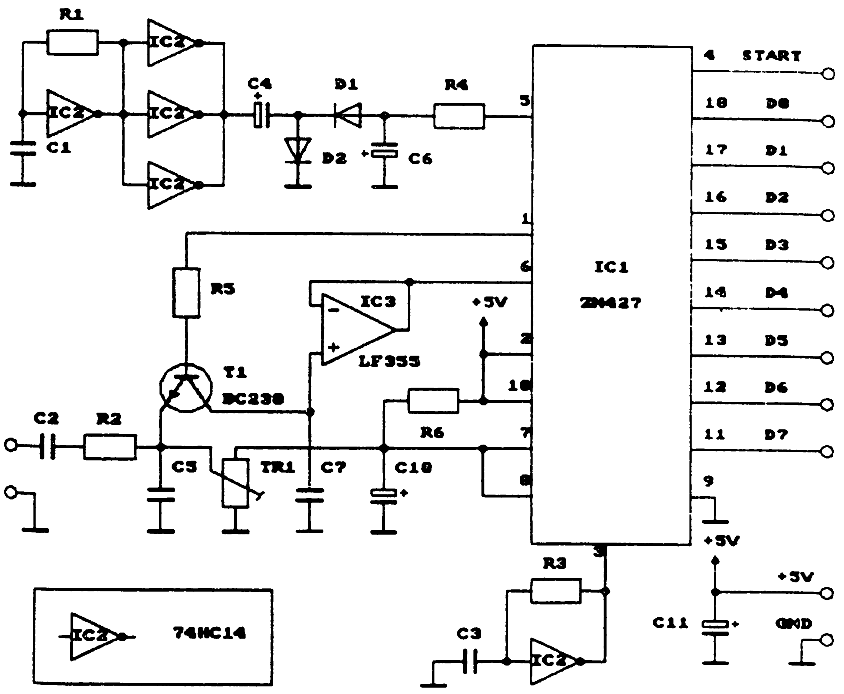
To move to a positive state (from 0 to +5V) the conversion process takes 10 cycles. At the end of this delay, T1 locks the input signal. This transmits the current voltage of C7 across the high impedance input of the IC3 amplifier to the analog input of IC1. This value doesn't change during the 10 frequency cycles. You obtain a frequency through an inverter (IC2) from R3 and C3, which together form an oscillator of 800 KHz.

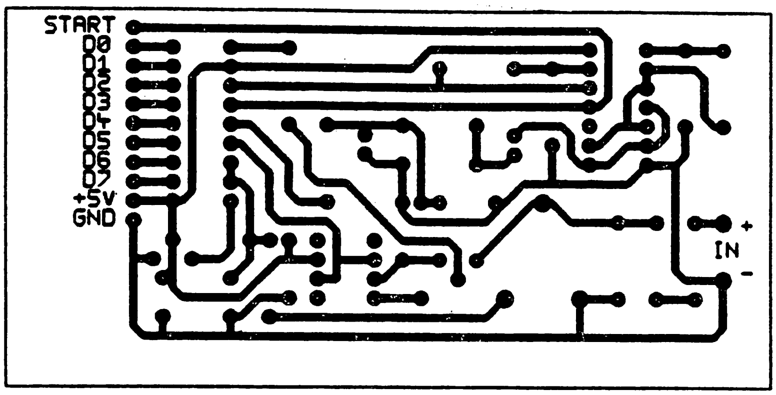
The convertor should have a negative voltage which isn't available at the parallel port. You must create this. With the inverter (IC2) at R1 and C1, make a new oscillator. At the output, there is a square wave signal, which oscillates between 0 and +5 volts. Using chain commutations, you can create a negative tension of -3.5 volts.

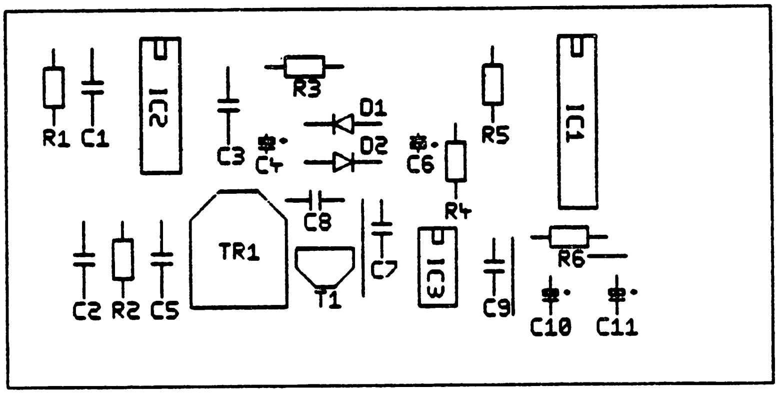
During this process, the C4 and C6 capacitors are simultaneously charged and discharged. You obtain the analog/digital conversion from

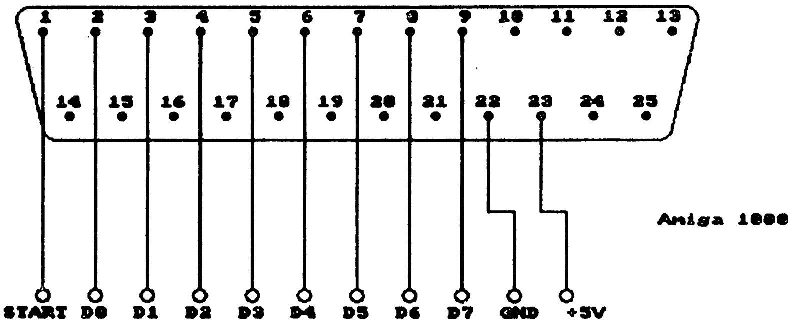
C1 (ZN247). We've chosen this component because it's easy to integrate into a computer and can easily be purchased.

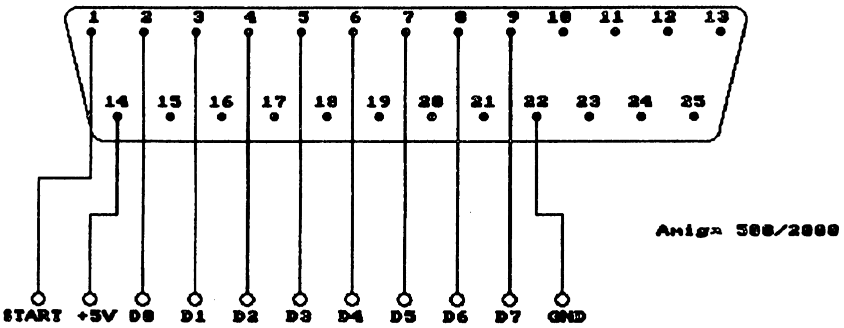
The summary circuit and board layout below should help you understand how this assembly works:






R1
C1
IC2
C3
R3
D1
D2
C4
C6
R5
IC1
R4
C8
TR1
T1
C7
IC3
C9
R6
C10
C11
C2
R2
C5


Amiga 1000
START
D0
D1
D2
D3
D4
D5
D6
D7
GND
+5V


Amiga 500/2000
START
+5V
D0
D1
D2
D3
D4
D5
D6
D7
GND

List of components for the sound digitizer:

R1, R3:	10K Ohm
R2:	1K Ohm
R4:	47K Ohm
R5:	1 M Ohm
R6:	390 Ohms
C1:	3.3 nF
C2, C8, C9:	100 nF
C3, C7:	150 pF
C4, C6:	100
C5:	35 nF
C10:	4.7 mF/10 Volts
C11:	1 mF/10 Volts
D1, D2:	1N4148
T1:	BS170
IC1:	ZN427
IC2:	74HC14
IC3:	LF355
TR1:	Trigger 5K linear

Miscellaneous:

2 feet of 11-lead electrical cable
One DB25 plug male (Amiga 1000) or female (Amiga 500/2000)
One RCA audio jack with a two-wire connecting cable
Three jumper wires

IC socket:	1x 18 pins
	1x 14 pins
	1x 8 pins

6.4 Sampling routine for digitizing

The digitizer whose setup we've just described can be used with the AudioMaster II program from Aegis. With this program you can immediately digitize any sound. If you don't have this program or if you want to write your own personal digitizing program, we'll present some routines that can be used as a framework.

The INIT routine lets you change the printing port to read. You can record sound with Record and you can program the Soundlength and Samplefreq variables any way you want. For this last variable, don't use a value that exceeds 28,000 Hz. The Play subroutine enables you to hear the sound through DMA. You can interrupt the Record and Play routines by pressing the left mouse button.

The following is a list of these routines:

```
*
* Program for sampling from the Abacus Making Music on
* the Amiga book. The original source was assembled with
* AssemPro, so slight modifications may be necessary to
* assemble the program withDevPac by  HisSoft, asm by
* Lattice and asm by Aztec.
* SoundLength is reduced from 40000 to 10000 to
*  fit on diskette.
*
AudLoc0 EQU     $DFF0A0          ;Audio address
AudLen0 EQU     $DFF0A4
AudPer0 EQU     $DFF0A6
AudVol0 EQU     $DFF0A8
SoundLength EQU   10000          ;Modify this
SampleFreq EQU    12000          ;Modify this

IntEnaW EQU     $DFF09A
DmaConW EQU     $DFF096
drbl EQU        $BFE101          ; Address of port
ddrbl EQU       $BFE301
touche EQU      $BFE001

;               SECTION          "digit",CODE
```

```
initport:
     MOVE.B #0,ddrb1                  ;Place port in read e
     MOVE.B drb1,D1                   ; read not valid data
     RTS
record:
     LEA buffer,A0                    ;Transmit
     LEA loop1,A1                     ;
     LEA wait1,A2                     ;Register address
     LEA drb1,A4
     MOVE #$4000,IntEnaW              ; INTERRUPT
     BSR convertir                    ;Branch to Period/10
     SUBQ #4,D0

     CLR D4
     MOVE.B D0,D4
     MOVE #SoundLength,D2             ;to D2
     BRA.S branch1                    ;Branch
loop1:   MOVE.B (A4),D1               ;Read song data
     EORI #128,D1                     ;in Amiga format
     MOVE.B D1,(A0)+                  ;and save
     NOP
     MOVE D4,D0
wait1:       SUBQ #1,D0               ;attention
     BEQ.S waitbutton ;
     JMP (A2)                         ; to wait1
waitbutton:BTST #6,touche             ; Flush on press
     BEQ.S end1                       ; end
branch1:     SUBQ #1,D2               ;Branch
     BEQ.S end1                       ;if d2>0
     JMP (A1)                         ;or loop1
end1:MOVE #$C000,IntEnaW
     RTS
play:
     MOVE.L #buffer,AudLoc0           ;start address
     MOVE.L #buffer,AudLoc0+16        ;song data
     MOVE #SoundLength,D0             ;length
     LSR #1,D0                        ;diviser
     MOVE d0,AudLen0                  ;
     MOVE d0,AudLen0+16               ;AudioReg
     MOVE #64,AudVol0                 ;Define volume
     MOVE #64,AudVol0+16
     BSR convertir                  ;convert  period
     MULU #10,D0                    ;with  frequency
     MOVE d0,AudPer0
     MOVE d0,AudPer0+16
     MOVE #$8203,DmaConW            ;DMA service
wait:    BTST #6,touche             ;attention
     BNE.S wait                     ;
     MOVE #3,DmaConW                ;DMA service
     RTS

convertir:
     MOVE.L #SampleFreq,D0          ;freq. sample
     MOVE.L #715909,D3              ;the routine
     DIVU D0,D3                     ;convert
     LSR #1,D3                      ;sample frequency
     CLR.L D0                       ;and one value
     ADDX D3,D0                     ;AudPer/10
     RTS

;    SECTION "data",DATA
buffer: DS.B SoundLength
     END
```

6.5 Other MIDI applications

The term MIDI stands for Musical Instrument Digital Interface. This is a standard interface that enables several musical instruments to communicate. You can identify a MIDI compatible device by the MIDI IN and MIDI OUT connectors, which look like those used several years ago by high-fidelity. An advantage of these cables is that they are relatively inexpensive.

The MIDI device receives data through the MIDI IN connectors and sends data out through the MIDI OUT connectors. Many of these devices also have a third connector, called MIDI THRU, which lets you link different synthesizers and computers with each other. MIDI THRU transmits the emitted signals to each device.

To avoid losing data because the tolerance limits of the components were exceeded, you should use a star network each time you will be joining more than three devices. Synthesizer 1, 2, and 3, as well as the rhythm device and computer receive signals directly from the Master, which could be an Amiga or a master keyboard. In either case, you'll need an interface that contains several MIDI OUT connectors. This can easily be created by soldering other MIDI OUT connectors in parallel with the one already on the interface.

There are a lot of uses for the MIDI interface. For example, you could connect several synthesizers to a master keyboard and then play from the master.

However, you could use a computer as the master if it has a MIDI interface. Then you would be able to play a rhythm device and two synthesizers simultaneously. You could connect these devices in a chain or network as we previously described.

MIDI information is transmitted on 16 channels. But you probably won't use all of them unless you have 16 devices, each containing its own information.

If you're an experienced musician you may have tried to synchronize several devices, such as a rhythm device and a sequencer. You wanted both of them to play the same rhythm. However, if you've tried this you know that it can be very difficult.

The MIDI interface can easily solve this problem for you. After a specific command signal, the two devices will begin to play the program you've set up. However, there is a limit to this. In order to separate different MIDI devices electronically, you need an opto-coupler.

Having an opto-coupler eliminates electronic interference signals. However, when you connect several devices together, you could receive erroneous information because of the tolerance limits of this type of component. This is why you should use a star network each time you've connected more than three MIDI devices.

6.6 MIDI interface options

With the MIDI interface it's possible to play an entire group of instruments from one keyboard. The only limit to the amount of instruments used is the musician's finances. Many of the current synthesizers include Dynamic Attack, After Touch, Pitch Bending, and Modulation functions.

Dynamic attack enables you to recognize, for example, the speed with which the key has been pressed. This parameter modifies the timbre.

After Touch lets you define the force with which the key is going to be pressed. This parameter also affects the musical timbre.

You may have already heard of Pitch Bending from HIFI. This function enables you to modify the pitch of sounds and to harmonize them with other instruments. To adjust this, use a potentiometer or a joystick.

Modulation enables you to modify, at definite intervals, either the pitch of the sound or its volume in order to create sound effects like a police siren or "wahwah".

Saving sounds

Many of today's synthesizers are capable of saving the numerical data of a sound. This can be done in the internal memory of the device (which often isn't large enough), in external memories (such as RAM cartridge types, composed of EEPROMS, which can be expensive), or through a MIDI channel to a computer.

Obviously a computer is designed to save data. This ability provides an easy but effective way to save sounds. The computer will send the necessary commands to the synthesizer through the MIDI channel so that it can transmit the sound data to be saved. We use the term MIDI DUMP. Now you can save your music on a diskette. You can also re-load this data in the synthesizer by performing the reverse procedure.

Suppose that you wanted to keep scores that you've composed. You could use the traditional system of writing the music on paper. Another method is to memorize your music, although this is almost impossible.

Instead of using these methods, you can use the modern approach - saving the data for a piece of music on a diskette. There are many programs that can do this. Simply connect the keyboard (synthesizer) to the MIDI interface of your computer, start the program, reload the data, and then start playing. Even though this method won't eliminate errors, it will enable you to make changes easily.

Sequences

A sequence is a repeated succession of notes. It could correspond to a bass accompaniment or to a refrain. When musicians accompany themselves, they usually use their left hands on the keyboard.

However, you may find it easier to have your computer do this for you. Simply compose the accompaniment on the computer, indicate the frequency at which it should be played, and then concentrate on the melody.

This is just a brief summary of the capabilities that the MIDI interface provides. You'll learn much more by experimenting with this interface yourself. If you're not a musician, you should know that the MIDI interface permits linking in series between computers at a speed of 32,500 Baud.

6.7 MIDI and Deluxe Music Construction Set

After loading the DMCS in memory, select the MIDI Channel option in the Sound menu to specify the channel number of the MIDI keyboard that's connected. Select MIDI Active so that the MIDI options can be accessed.

If you also activate the MIDI Input Enabled option, you can capture the notes of the keyboard by using the MIDI interface. However, you cannot play instruments saved on the diskette from the keyboard.

The first thing you should do is confirm the MIDI Setup option. The system has to be designed for its own uses. So it's possible to give a certain value to the Input Delay parameter. This has the following meaning:

When a key, which is on the keyboard that is connected to MIDI, is pressed, the Amiga records it, including the length of time it was pressed. When you press a key, the corresponding note appears on the staff on the screen. The longer you press this note, the longer the note will be.

The Input Delay parameter enables you to define an amount of time, at the end of which the length of the note should be modified (i.e., moving from an eighth to a quarter note).

Send MIDI Clock is another important function. When it is activated, the Amiga sends a synchronization signal, along with the data of the note, in the MIDI channel. This is important to rhythm devices.

Now that you've set up the system for the MIDI link, you can begin to work. You can compose music directly from the keyboard and see the score on the screen.

However, you should be careful because your score isn't in real time but in a step by step mode. This is helpful for capturing precise sounds because you can avoid composition errors. Unfortunately, your own rhythm isn't included. The only way to restore this rhythm is by programming notes.

DMCS's biggest disadvantage is its relative rigidity for chords. Actually, if a measure ends with a chord, DMCS uses the same foundation tone and place for the subsequent measure.

The only way to avoid this is to indicate the chord using half of its duration and then double it through programming. Remember that, prior to this, the reference value for the length of the note was indicated in the Note Palette window.

Since the DMCS program is written in C, there are problems with the timing of the MIDI interface. For tempos over 200 beats per minute (BpM), synchronization is no longer correct. As a result, the Amiga no longer follows the rhythm.

Unfortunately, there isn't a solution to this problem. The only thing that you can do is to keep things slow and don't compose scores that exceed this value.

The creators of DMCS set up the program for the American electrical standard of 60 Hz. As a result, the tempos and the clock values aren't correct if your electrical system uses 50 Hz, as in Europe. In order to obtain the correct values, multiply the value you want by 5/6.

Note: In the Score Setup menu of DMCS the musician can modify a piece of music by an entire octave. Although this function is offered for internal note management, it doesn't affect the MIDI device.

Before playing notes on the keyboard, be sure that you click on the place in the score where they should appear. If you forget to do this, these notes will appear at the previous cursor location.

6.7.1 Some basic tips

You should remember the following tips when working with DMCS:

- It's only possible to change an instrument at the beginning of a measure.
- You can't erase the dot following a note or eliminate ties or triplets.

- If you're using two different diskettes, be sure that one of them is inserted in the drive before the first access.
- DMCS can edit scores that are difficult to edit with Sonix.

Important:

In order to edit a piece of music, you must work on two files. If an error (e.g., Disk Full or Guru Meditation) appears during the process, the file that you're using is automatically erased. So you should always make a copy of the piece on which you're working and frequently save your scores.

If you need more room on the diskette, you can replace the Splash file in the C directory. This corresponds to the display. Replace this file with an IFF image, which uses very little memory. This will provide several hundred extra bytes.

6.8 MIDI and Sonix

You've probably noticed that, besides the four usual instruments, Sonix contains four additional instruments. These can only be used with the MIDI channel. Simply click on Mode II to access them.

The synthesizer only plays a passive role in the Sonix program. Although it reproduces a tune from the computer, it can't be used to compose a tune on the keyboard while reproducing the notes on the screen. So you must take the tune you want to compose note by note.

Sonix also has its advantages. Initializing the Sonix interface is very easy. When the interface is connected, you must load the MIDI Patch file as an instrument. A table, in which you can make any adjustments, appears:

Channel	This parameter indicates the number of the MIDI channel on which the Amiga should send data. In other words, it indicates the channel on which the MIDI instrument is set.
Octave	Lets you modify the pitch of the sound to be played by the MIDI keyboard that's connected.
Factor	This controls the velocity sensitivity of the MIDI keyboard (i.e., how hard the keyboard keys are pressed).
Volume	Lets you regulate the volume of the keyboard that's connected.
Bender	Each synthesizer contains an adjustment for pitch, which slightly modifies the pitch of the sound. This adjustment corresponds to Pitch Bender, found on MIDI keyboards.
Wheel	This adjustment allows you to add vibrato to any note or chord.

6.9 Using MIDI Patch

Suppose they you've composed a piece of music that is ten measures long and you have two synthesizers. You want synthesizer A to play measures 1 to 5 and synthesizer B to play measures 6 to 10. To do this, synthesizer A obtains MIDI channel 1 and B receives MIDI channel 2.

First define MIDI Patch 1 as follows:

Channel: 1

Patch: 1

For the second synthesizer (B):

Channel: 2

Patch: 2

The other parameters will only be modified according to the desired effect. Place this last Patch starting with the sixth measure.

As a result of these settings, the Amiga will first detect Patch 1 and move the data toward MIDI channel 1 (synthesizer A). The first five measures will then play. At the beginning of the sixth measure, it will detect Patch 2 and move the data toward MIDI channel 2 (synthesizer B).

6.10 Tips and tricks for sampling

Sampling is usually quite simple. Insert the sampler in the corresponding port, switch on the computer, load the sampling program and record the sound.

Unfortunately, you don't always receive the desired results. The following are several tips to help you get the best results:

6.10.1 Recording

To record, you'll need an Amiga with at least 512K, a sampler and its corresponding software (e.g., the software mentioned in this book and the AudioMaster II program). You'll also need a microphone or equipment for sound recording.

You should send the sound source through a high-fidelity amplifier. There are usually two ways to obtain this kind of link. You could use the headphone jack but this only produces an average sound quality.

It's better to use the speaker outlet. Although this requires an intervention of the connections, the results are much better. If your installation has extra connectors for a second set of speakers, use these. Then you can still hear the sound that you're going to record.

6.10.2 Optimum volume adjustment

Most amplifiers contain adjustments for bass and treble. When all the connections are properly made (with the volume set at minimum) and the sampling program is loaded, you can begin optimum volume adjustment.

Activate the Monitor window and send any sound signal to the Amiga (e.g., a sound taped on a cassette). Slowly turn up the volume until you hear the sound signal clearly in the speaker.

To get the optimal signal, slowly turn the potentiometer until the signal is at the saturation limit. Now turn back the potentiometer until the signal is no longer saturated. You'll now have the best signal. Turn down the bass a little and add a bit of treble to get a more polished sound.

Note: The Amiga partially limits sounds above 400 Hz. However, the color of a sound is a matter of personal taste.

6.10.3 Digital recording

Now we'll present an example for digital recording. Suppose that you want to sample a sound at a frequency of 500 Hz in order to vary the pitch of the sound so that you can use it as an instrument for a piece of music. You must determine what frequency you should use to sample this sound.

Theoretically, doubling the signal frequency should be sufficient for creating a good reproduction. However, if you apply this rule (with a digitizing frequency of 1000 Hz), and then use this sound an octave lower than the one used in recording (250 Hz), you'll encounter problems. Actually, at this frequency level, the sound will no longer be clear.

This happens because, during a digital recording at 1000 Hz, the resolution (value per second) is 1000. To play this sound an octave lower, the reproduction frequency is 500 Hz. Obviously this will have an effect on the resolution (the sampling) that moves to 500.

So, for this last sound, the computer no longer has only 500 pieces of information, which produces a less defined sound. If you use a resolution of 2000, you can obtain a resolution of 1000 Hz by dropping an octave, which will produce a better quality sound reproduction.

Generally when you want to lower the sound a maximum of one octave, you must quadruple the value of the sound frequency you're recording. This is an excellent method for obtaining quality sounds. However, don't forget that this will require more memory.

Sometimes this rule won't exactly apply to the situation. In these instances you'll have to perform several experiments before accurately defining the frequency.

However, if the reproduction frequency should be identical to the recording frequency, the following method will help you avoid frustration.

When you want to sample a piece of long music, you must find a compromise between its sampling frequency and its size. When you listen to the sampled piece, the sound may seem slightly metallic. This distortion, called Aliasing, indicates that the frequency exceeds the chosen resolution. For a speed of 2000 Hz, it isn't possible to reproduce a sound of 3000 Hz.

To avoid this distortion, it's usually adequate to lower the trebles to the lowest setting. A slightly duller reproduction has practically no influence on the monitor's speakers.

6.10.4 Changing the sound after recording

This includes any modifications of the sound after its recorded. These modifications include changing the pitch of the sound, the volume and anything that might enhance the sound reproduction. The following is a detailed description and some tips you can use.

Changing the Pitch of a Sound

When you digitize, you change the sound pitch by changing the reproduction speed. Although this is the easiest method, it's not necessarily the best. There are other ways to change the pitch but they are complicated and usually can't be treated in real time.

The method for changing speed is fairly difficult for programs such as Sonix, DMCS, etc. Remember that a sound higher than the original one is shorter than a lower sound.

However, you don't need to follow this rule if you're using the Looping technique, which we'll explain later. However, you should know that Looping plays a sound over after it has been played the first time.

Now let's return to reproduction speed. Changing the speed is used for sampled sounds of musical instruments. With this technique you can also produce sound effects, such as a "chipmunk" voice.

Scaling, Change Volume

This allows you to change the sound volume of a sampling. When you want to play several musical instruments simultaneously, usually the volume of each one isn't properly adjusted.

For example, instrument A might overpower the other instruments, instrument B can only be heard occasionally, and instrument C can't be heard at all. This happens because, when sampling, you don't know how much one sound will hide another. You're concerned with the quality of the sound instead of the quantity.

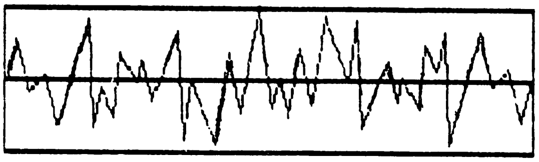
Some sampling programs let you change the volume of the final sound of the digitization. This process is called scaling. However, scaling doesn't have any effect on the volume of the sound channels; it only limits each value of the sampling.

While entering the data for scaling, the maximum level for the values is considered. So, only the highest volume level reached up to that point is retained from the defined level. The other levels are changed according to the scaling factor.

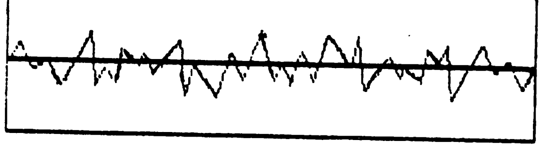
The following illustration should help you understand this:

Scaling

Maximum value (128)



Scaling value: 64



Maximum value: 64

Looping

Looping is one of the most important functions for sound data. Basically it is used for immediately repeating the sampling. You may already be familiar with looping since this technique is often used for sounds in computer games.

Some software companies sample entire pieces of music so that they don't have to compose their own music. However, if you don't have an Amiga with a large memory, you'll can only use basic examples, which are usually very short.

Looping can be used in various ways. For example, it can be used in commercial software for introducing an advertising tune, generating musical instruments or background noises, etc.

Simply looping a melody probably won't work because you can often hear interference noises. So, you must separate the good stuff from the junk.

Anyone can use looping. Simply define the beginning and the end of the piece you want to repeat. Sometimes that's all you need to do. However, producing a high quality sound requires extra work. Usually a bad loop will be produced if you haven't correctly considered the timing. Often you'll hear a click at the point where the looping begins. Most of the time you can eliminate these undesirable effects.

You must carefully listen to the piece from which you want to take an excerpt. Then select the part you want to loop. If you choose a refrain that doesn't contain words, you'll barely notice the repetition.

During the sampling process, ensure that you've left enough space before and after the piece to which you're adding the excerpt. Even though you can cut out parts, you can't add any. If the piece is too short, you have to record a new sampling.

For very rhythmical pieces, the percussions (such as drums, cymbals, etc.) represent the highest values in the sound volume. Mark them graphically by using a peak value that can quickly be determined. These points are generally the best points for the looping because they correspond to the rhythm and allow a loop that follows this rhythm.

Since a refrain or chorus usually begins with a percussion instrument, it's possible to place the beginning of the loop as close as possible to the peak signal. Place the end as close as possible to the last release of a percussion peak.

The first and last point must correspond. Often the curves of the two rhythmic beats are similar. This allows you to insert two points by hand. Remember that, ideally, the volume of the beginning and ending points must be identical or almost identical. If you're not careful, you'll hear a click, which is caused by the sound volume difference between these two points.

Now let's discuss the method for samplings that do not contain an accompaniment rhythm.

Eventually, all composers want to integrate a sound, that he/she has sampled, into a music program, such as Sonix, DMCS, etc. As an example, let's use the human voice, which is very difficult to loop.

We'll use an "AAAAAAAAHHHHHH" sound, which will be incorporated into a piece of music. The beginning point should be in the middle of a sampled piece in order to give it some body. The human voice (and many instruments) is difficult to digitize because the pitch of the sound isn't consistent. However, this quality gives life to the sound.

The pitch of the sound at the end point of the sampling should be, theoretically, the same as at the beginning. If you don't follow this rule, you'll again hear a click when you move from one pitch to another.

To avoid this, you should take a closer look at the wave shape of the sampling. Since the pitch of a sound is directly related to the number of impulses per second, you can represent it graphically. Remember that for a high sound, the lines are close together, but for a low sound the lines are relatively far apart.

If the beginning point of the loop sequence falls on a sound that's relatively high (lines of the wave are close together), the end point also should be located at a place where the sound is more or less at the same pitch. You can adjust differences in volume by using the expansion function.

Sampling using AudioMaster II

If you're using the sampler we've described in this book, in this section you'll find some helpful tips for better samplings with AudioMaster II.

Once you've linked the computer, sampler, and stereo system together, you can begin. Select the Sampler option in AudioMaster II and click the Monitor box. Speak slowly and clearly into the microphone; your "speech" will appear graphically.

Now try singing. Click Sample Hi and sing a note using "AAAAAAAHHHHHHH" for at least three or four seconds. Try to keep the same pitch and the same volume, otherwise you'll complicate a loop. Now that the digital sound of your voice is "in the box", leave the Sample option and return to the main menu.

Check the quality of your voice recording by using Play. If you don't like it, try again. It's very difficult to loop a poor quality recording.

Next select Loop On. Two red lines appear on either side of the screen. Place one on the left just behind the starting point of the sampling, but ignore the very first section of the sampling since it is outside the section to loop. You can control this positioning with Waveform. Then select Stop.

First mark the space on the left and enlarge this section. Place the left red line at a place where there is almost no sound volume (part of the curve near the median line in the sampling window). Now click Show All and repeat this process with the red line on the right. When you have found the ideal point, click Waveform. The sound should play without interference. Then you can save it.

However, this doesn't always work on the first try. If this happens, move one or both of the red lines slightly towards the right or left. Repeat this until you are completely satisfied with the result.

AudioMaster II contains a very helpful feature: Seek Zero. First position the two red lines in place. Start the reproduction using Waveform, then click on Seek Zero. The program then moves the last red line used onto a nil position of the curve. Next click on the second line and select Seek Zero again.

Theoretically, the interference should have disappeared. If you don't like the result, repeat both operations as often as necessary. All interference must disappear in order to have an acceptable result.

Chapter 7

Public Domain Music Programs

Chapter 7 Public Domain Music Programs

Perhaps after reading this book and viewing the descriptions of a few of these programs you have come to realize that your composing skills are not quite at the level of Beethoven or Mozart, or even Bon Jovi. Or, maybe you think you're a good composer, but you have no idea which of the programs in this book represent "entry-level" Amiga music software. No user wants to invest lots of money in a piece of software, then discover that he or she doesn't like the purpose or user interface of the software.

Fortunately, there are a number of public domain and shareware music programs available for the Amiga. Many of these programs generally contain easy routines to help you get started on music composition. Others may be more difficult to learn to use, but perform powerful tasks that the other music programs don't offer.

Public domain programs are copyright free, which means that you can legally obtain a copy from a friend, use it, and change it any way you want without paying anything for it. Shareware programs, however, are sold on a "try it, buy it" basis. If you use the program regularly, you should register it with the developer by paying a small fee. This will also ensure that you receive the latest updates.

As with many public domain or shareware programs, the documentation is often either minimal or nonexistent. We hope that this chapter will give you an idea of what these individual programs can do. However, if you find little or no documentation, you'll usually have to experiment to see what you can accomplish with each function.

Usually these programs include samples of music and sound effects that you can modify or add to your own compositions. Even if you never compose another note in your life, it's worth getting some of these programs just to listen to the samples they contain. Think of how impressed your friends will be when they hear the scream of a certain cartoon duck coming from your Amiga (more on this later).

7.1 MED, the Music Editor

MED is a public domain music program that lets you use 38 different instruments on all four tracks to compose your own songs. The MED program also includes six sample songs to demonstrate how the program works.

Each song done with the MED program is made up of up to 50 individual blocks. Each block is roughly one to eight measures in length, depending upon how much detail you put into each one and the tempo setting.

You can arrange or repeat these blocks in any way you want. For example, let's say you compose a simple song with only eight blocks numbered from #0 through #7. The MED program automatically calls the first block it creates block #0, the second one block #1, and so forth.

You can instruct MED to re-use blocks as needed. Suppose there are more elements to this song, but these elements use the same material. You could play blocks #0 through #6 in order, have MED go back and repeat block #5, play block #4 three times in a row, and finally replay blocks #7, #3, and #2 in a loop played five times. In this case, even though you only have eight different blocks, you can use them a total of 26 times to make a longer song.

MED comes with a large assortment of instrument samples. As with so many instrument sample groupings, some are excellent, while others are adequate.

Here is a list of the 38 instrument samples included in the version of MED that we tested. The MED composing program will "only" let you load 30 of these samples at one time. But then, how many of these will you actually use in a single 50 block song?

AnalogString	HiHat2	SineCZ
BassDrum1	JahrMarkt1	Slapbongo1
BassDrum2	MonoBass	SnareDrum2
BassDrum3	NightMare	Strings7

BassDrum4	OpenHi	Strings8
BassDrum5	Perc.8	StringsC
BigBow	Perco	StringsCM
Blast	Piano	SynBrass
Claps1	PopSnare2	SyntheBrass
Claps2	RoomBrass	TomDrum1
Crash	Sdrum1	TomDrum2
ElecBass	Sdrum3	TomDrum3
HiHat1	Shaker	

The MED program also features two unusual functions that have little to do with actual music composition, but are quite fascinating:

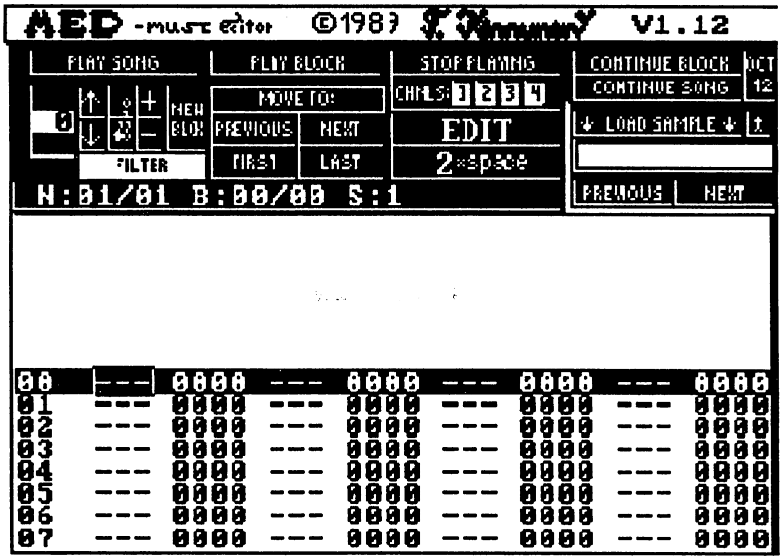
- Screen color changes.
- Controlling the mouse pointer.

First, MED lets you easily adjust the screen colors without returning to the Workbench. These colors can even be changed for each song.

Second, MED gives you some control over the jumping figure named "Topi" by the programmer. Topi serves as a mouse pointer, but will also jump up and down to the rhythm or instrument that you set. He will also "fall asleep" in wait pointer procedures (e.g., while you are doing such operations as loading songs). If all this seems a little silly to you, you can also set Topi so that he doesn't jump at all.

7.1.1 Starting MED

When you first call up the main MED program, a screen like this should appear:



```
00  ---  0000  ---  0000  ---  0000  ---  0000
01  ---  0000  ---  0000  ---  0000  ---  0000
02  ---  0000  ---  0000  ---  0000  ---  0000
03  ---  0000  ---  0000  ---  0000  ---  0000
04  ---  0000  ---  0000  ---  0000  ---  0000
05  ---  0000  ---  0000  ---  0000  ---  0000
06  ---  0000  ---  0000  ---  0000  ---  0000
07  ---  0000  ---  0000  ---  0000  ---  0000
```

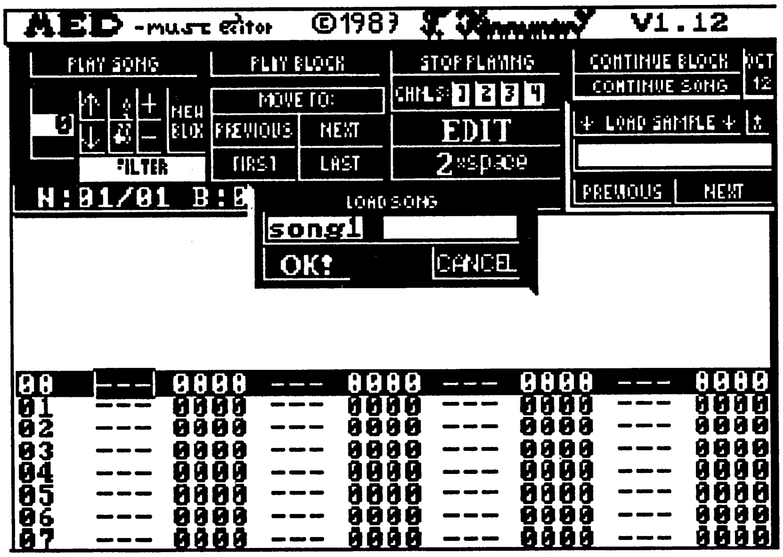
MED main screen

If you click with the right mouse button on the top title bar, you should get a series of pull-down menus. These menus will have the following headings:

```
Project  Block  Samples  Misc.
```

Let's start slowly. Go into the Project menu and select the Load song item. Type "song1" into the string gadget that appears.

You should now have a screen like this:



```
N:01/01 B:0
song1
00 ---  0000 ---  0000 ---  0000 ---  0000
01 ---  0000 ---  0000 ---  0000 ---  0000
02 ---  0000 ---  0000 ---  0000 ---  0000
03 ---  0000 ---  0000 ---  0000 ---  0000
04 ---  0000 ---  0000 ---  0000 ---  0000
05 ---  0000 ---  0000 ---  0000 ---  0000
06 ---  0000 ---  0000 ---  0000 ---  0000
07 ---  0000 ---  0000 ---  0000 ---  0000
```

Load "song1" into MED

Press the <Return> key or click on the OK! gadget to load the song.

You will notice that an information line like this will appear near the middle of your screen:

```
N:01/24 B:00/12 S:1
```

The first "01/24" indicates that this is the first block played out of a total of 24 times that we'll use the blocks. The next "00/12" shows that we are using block #00 out of 13 blocks that were actually composed (the last block is #12). The "S:1" refers to the sample number.

If this sounds a bit confusing, think of the example earlier where we only composed eight different blocks, but we used them a total of 26 times. In that case when we loaded up that imaginary song the information line might look like this:

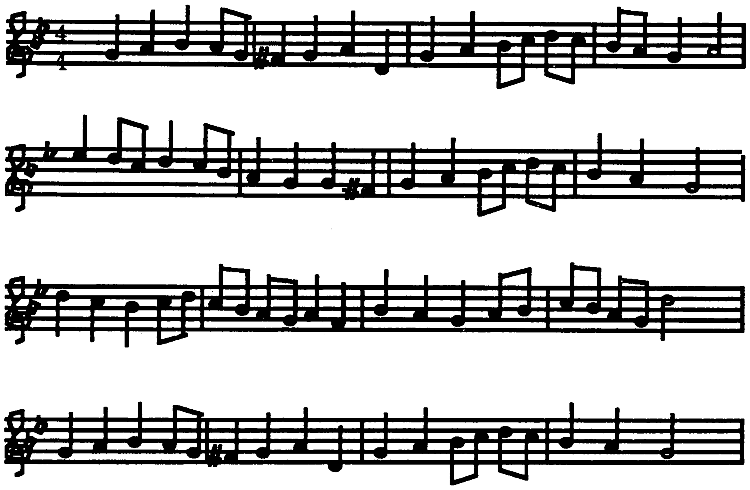
```
N:01/26 B:00/07 S:1
```

Try playing the song you loaded by clicking on the PLAY SONG gadget. If the volume is too high or low, adjust the volume on your monitor, or the stereo through which the Amiga is playing. If the song sounds "empty", remember that MED plays through both audio channels of the Amiga: If you're only playing a single channel through your monitor, hook your Amiga's audio outputs through a stereo system.

Song1 will keep playing in a continuous loop until you stop it by clicking on the STOP PLAYING gadget. The playback of Song1 will give you an idea of what you can accomplish with MED.

7.1.2 Composing Songs

Next, let's try entering a song into MED. This tune appears here with the permission of its composer, Gene Traas. First, let's look at what the actual piece of music looks like:



Use the NEW SONG option in the PROJECT menu. You should see a small requester containing these options:

```
CLEAR ALL

CLEAR SONG ONLY

CANCEL
```

Choose CLEAR SONG ONLY so that you can keep the sample instruments loaded with SONG1. (If you chose CLEAR ALL, you would have had to load each instrument one at a time.)

Next, let's pick an instrument from the samples we have loaded. You could go through these with a combination of <Shift> and either <Cursor Left> or <Cursor Right>. Another option is to use the PREVIOUS and NEXT button in the LOAD SAMPLE area. Just so you have some idea of what to expect, pick "Piano".

The little button at upper right that says "OCT 12" doesn't refer to the author's birthday. Instead, the MED program will let you compose in three octaves, but you can only use two at a time. Choose between OCT 12 for the lower two octaves or OCT 23 for the upper octaves.

When you actually want to place notes in your new song, click on EDIT. You can now add notes in any of the four channels. Move between the channels and to different notes with the cursor keys.

You now have to think of your computer keyboard as a piano keyboard. When you press certain keys on your keyboard, you will see a musical note instead of a letter. These notes are arranged the same way as a piano keyboard.

The following table shows you how to access two full octaves by pressing either the bottom key listed below each note for the lower octave or the top key for the upper octave:

C	C#	D	D#	E	F	F#	G	G#	A	A#	B
Q Z	2 S	W X	3 D	E C	R V	5 G	T B	6 H	Y N	7 J	U M

Now let's try actually entering a song. Place the cursor on the second group of dashes that look like "- - -". This puts our music in the second channel where you will hear each note as it's entered even if a stereo system isn't connected.

We're only going to be using two octaves, so you can leave the octave selector at either OCT 12 or OCT 23. This is a little like "painting by numbers" since we're more interested in the results we achieve than an explanation of music theory.

First press the "B" key. This should give you a G note in the lower octave, or "G-1".

The line will automatically advance from line #00 to line #01. We want our note to last a bit longer, so use the cursor to move to line #02 and type a "N" to get a "A-1" note. For our song, that will give us the equivalent of a quarter note. Skipping three lines will give us a half note, while not skipping any lines will give us an eighth note.

Complete this block with the following entries (leave the lines not listed blank):

LINE #	LETTER	NOTE	LINE #	LETTER	NOTE
00	B	G-1	32	3	D#2
02	N	A-1	34	W	D-2
04	J	A#1	35	Q	C-2
06	N	A-1	36	W	D-2
07	B	G-1	38	Q	C-2
08	G	F#1	39	J	A#1
10	B	G-1	40	N	A-1
12	N	A-1	42	B	G-1
14	X	D-1	44	B	G-1
16	B	G-1	46	G	F#1
18	N	A-1	48	B	G-1
20	J	A#1	50	N	A-1
21	Q	C-2	52	J	A#1
22	W	D-2	53	Q	C-2
23	Q	C-2	54	W	D-2
24	J	A#1	55	Q	C-2
25	N	A-1	56	J	A#1
26	B	G-1	58	N	A-1
28	N	A-1	60	B	G-1

If you want to hear what you've done to this point, click on PLAY SONG or PLAY BLOCK. Remember to use STOP PLAYING when you're finished.

Let's make this song a little more elaborate. Click on NEW BLCK to create a new block. (This will be called block #1 since our first block is named block #0.) You then click on NEXT in the MOVE TO: area.

Add these notes to your new block:

LINE #	LETTER	NOTE
00	W	D-2
02	Q	C-2
04	J	A#1
06	Q	C-2
07	W	D-2
08	Q	C-2
09	J	A#1
10	N	A-1
11	B	G-1
12	N	A-1
14	V	F-1
16	J	A#1
18	N	A-1
20	B	G-1
22	N	A-1
23	J	A#1
24	Q	C-2
25	J	A#1
26	N	A-1
27	B	G-1

LINE #	LETTER	NOTE
28	W	D-2
32	B	G-1
34	N	A-1
36	J	A#1
38	N	A-1
39	B	G-1
40	G	F#1
42	B	G-1
44	N	A-1
46	X	D-1
48	B	G-1
50	N	A-1
52	J	A#1
53	Q	C-2
54	W	D-2
55	Q	C-2
56	J	A#1
58	N	A-1
60	B	G-1

Add the two blocks together by first clicking on the down arrow on the left side of the screen and then the plus sign (+). Now click the + again and a "1" should appear below the first "0".

This means that our song will first play block #0 when it sees the "0" at the top of this list and then play block #1 when it sees the "1" next on the list.

If you wanted to delete a block, use the arrow keys to place the number of the block you want removed in the shaded portion. Next, click the down arrow and the minus sign (-).

The two buttons between the arrows and the + and - keys allow you to move to the first and last blocks of the song.

Now let's use more than one track. We're going to add a different instrument on another track to create two-part harmony.

Go back to the first block (use the FIRST button) and make a copy with the COPY BLOCK function in the BLOCK menu. Use NEW

BLCK to create a third block (block #2). Move the cursor to this block with the LAST button and then use PASTE BLOCK. At this point your first block (#0) and last block (#2) should be identical.

Set the cursor on the third track (the third set of dashes) and reset the Octave button to the opposite of what you had before (in other words, if it was set at OCT 12, change it to OCT 23).

Next, use the NEXT sample button until you find a blank space and then click on the LOAD SAMPLE button. Type in "synBrass" where the blinking cursor is and then click on LOAD SAMPLE again. (If you get a sample error message, make sure that you typed in the name of the instrument exactly, paying close attention to the upper- and lower-case letters.)

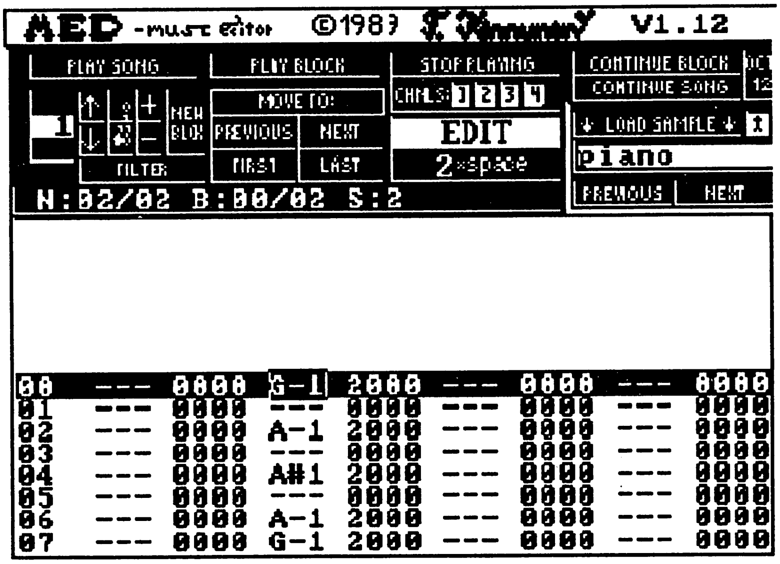
Now go through the first table and type in the letters on the given lines again. Notice that this time, however, instead of getting a "G-1" for B, you get a "G-2".

When you finish, add block #2 to the end of the song the same way you added the last block. This time on the last step you'll have to press the + twice to go from "0" to "2".

Repeat these procedures for the second block and a new fourth block (blocks #1 and #3).

Finally, choose the SET TEMPO option of the PROJECT menu. We just want to slow down the tempo a bit, so set the slider bar at 19 and click on OK.

Your finished screen should look something like this:



```
N:02/02 B:00/02 S:2

00  --- 0000 G-1 2000 --- 0000 --- 0000
01  --- 0000 --- 0000 --- 0000 --- 0000
02  --- 0000 A-1 2000 --- 0000 --- 0000
03  --- 0000 --- 0000 --- 0000 --- 0000
04  --- 0000 A#1 2000 --- 0000 --- 0000
05  --- 0000 --- 0000 --- 0000 --- 0000
06  --- 0000 A-1 2000 --- 0000 --- 0000
07  --- 0000 G-1 2000 --- 0000 --- 0000
```

MED screen with sample song

Now use PLAY SONG to hear the entire song. You can also save this song with the SAVE SONG function of the PROJECT menu.

7.1.3 Other MED Functions

Quit

This item, which is located under the Project menu, lets you exit MED. You can also exit MED by pressing the right <Amiga> key and the <Q> key.

Set Colors

The Set colors item under the Misc. menu lets you change the screen colors. When you call up this function, a slide control will let you adjust the RGB levels. Any current background screen color will be saved and reloaded with each song.

Reset colors will return the colors to their previous values.

Guy

This item from the Misc. menu lets you control how Topi jumps around. If you select Every 8. note, Topi will make a leap on every eighth note.

If you select Sample ctrl, Topi jumps whenever a note from a given sample is played. After you choose this option, pick a sample (such as drums), in which Topi will leap and then click on the small icon next to LOAD SAMPLE. If you pick more than one sample with this option, Topi will jump whenever any of these samples are played.

If you select Don't jump, this will prevent Topi from jumping. Unfortunately, if you still find Topi somewhat distracting, there isn't a way to change him to a normal pointer.

Filter

This gadget on the left side of your screen turns off the low-pass filter in the A500/2000/2500 models. This can improve the quality of the sound but might result in some distortion. The best way to see how this works is to play a song both with and without the filter.

When you save a song, the filter will be saved in its current state.

CONTINUE BLOCK
CONTINUE SONG

These two gadgets, near the upper right of your screen, let you restart your tune from the current cursor location.

LOAD SAMPLE

Clicking on the LOAD SAMPLE gadget will place a blinking cursor in the entry box below it. You can then type in the name of your instrument from the list at the beginning of this section.

To receive another clear entry box, click on NEXT or PREVIOUS. Then re-click on LOAD SAMPLE in order to get the blinking cursor to make entries.

Of course, this is a somewhat tedious process. One way to save some time is to first load a song that already has several instrument samples and then use the CLEAR SONG ONLY option when you activate the NEW SONG option. (This is what we did with our example song.)

Another possible option is to enter the names of the 30 instruments you are most likely to use and then save them as a song file named "INSTRUFILE". Then when you want to create a new song you can open this file and the most often used samples will be immediately available.

CHNLS 1 2 3 4

These gadgets on the main screen let you turn off or on each one of the four tracks.

2x space

This option, located under the EDIT gadget, automatically skips a line when you write music with EDIT. In our sample song, for example, you could have used this option to place notes on every other line for quarter notes and then gone back and filled in those lines in which you wanted eighth notes.

Block Menu

The items in this menu let you cut, copy, or paste an individual track or block.

Cut track	Lets you cut the selected track from memory.
Copy track	Lets you copy a track from memory for pasting in elsewhere.
Paste track	Lets you insert the track most recently cut or copied.
Cut block	Lets you cut the selected block from memory.
Copy block	Lets you copy a block from memory for pasting in elsewhere.
Paste block	Lets you insert the block most recently cut or copied.
Remove last block	Removes last block from memory.

Setting Volume

When you were adding notes in the sample song, you probably noticed the four digits that followed each note. The first digit identifies which instrument sample is being used with each individual note.

The next three digits let you make changes on each of the notes. If you leave these digits at "000", the MED program will just use the default settings or other settings that you might have specified with functions like the Set tempo item.

Several options are available here. However, we will just talk about how to set the volume and tempo of each individual note.

Set the volume by placing a "C" as the second digit after the note. The last two digits let you set a value between 00 and 63. Here, "00" is the lowest volume setting while "63" gives you the maximum volume.

For example, if you wanted a G note, in the lowest octave played by sample #1, to have a volume of 40, you would enter:

```
G-1  1C40
```

If you want a note to be "cut off" immediately after it is played, type in the second line:

```
A#2  5000
---  0C00
```

If you want a note to start out quietly but then go to full volume, try something like these two lines:

```
C-3  3C10
---  0C63
```

Setting Tempo

You can set the tempo of each individual note about the same way you set the volume. Here you use an "F" as your second digit. The last two digits will be the actual tempo in hex numbers.

For example, if you want the new tempo at 64, use a hex 40 like this:

```
B-2  7F40
```

The tempo will now stay at 64 for the rest of the song unless it is changed again.

You can also use this command to move to the next block by using a "00" as the last two digits:

```
D#1  2F00
```

This helps you if you're doing something like writing a song in 3/4 time or you just want a short block section.

The MED program is included on CUCUG Amiga disk #38. To get a copy of this disk, contact:

CUCUG
P.O. Box 716
Champaign, IL 61824-0716

At this writing, CUCUG charges $7.00 for each one of their disks (including shipping and handling).

7.2 Perfect Sound Sound Editor

The Perfect Sound Sound Editor is a shareware program designed primarily for use with the Perfect Sound stereo audio digitizer. This program will help you create, store, recall and edit various digitized sounds on your Amiga.

Using this program without getting the Perfect Sound digitizer is probably a little bit like buying a new car just to use the headlights. In fact, the program and digitizer are no longer sold separately. The entire package including hardware and software is very moderately priced and is available from several vendors.

If you run across an older copy of the Perfect Sound Sound Editor by itself, it comes with a variety of very realistic and often quite funny sounds, such as the cartoon duck's scream that we referred to earlier.

After you open this program, you can access each sound by using the OPEN function of the PROJECT menu (click on the title bar with the right mouse button to get this menu). When the LOAD A SAMPLE dialog box appears, click on the scroll bar at right to get the entire list of available sounds. Our cartoon duck scream is titled, appropriately enough, "AAAAHHHH!"

You can load up to 14 sound samples with this program. To hear each sample, highlight the sample name with the mouse and then click on the speaker icon located at lower right. You can also see a graph of each sample by clicking on the graph icon.

Without the digitizer, there are severe limits to what you can do with this program. However, there are a few interesting options that you can perform with each sample.

For example, with FLIP THIS SAMPLE, from the SAMPLE menu, you can invert a given sound sample. Changing the START and END scrollbars at the bottom of the screen will give you a beginning and end point of any part of the sample with which you want to work. When you use CHANGE PLAYBACK SPEED, from the EDIT menu, you can speed up or slow down the sample.

You can also add two samples together by using the INSERT MARKED RANGE function of the EDIT menu. When you call up this function, the program asks you to click on the name of the sample that you wish to add. This sample will then be pasted in where the POS'N cursor is located.

If you want to get a little more elaborate and lengthen a given sound sample, use the START and END scrollbars to mark the ends of a section you wish to copy. Use the graph icon to try and make the values of our endpoints match as closely as possible.

Place the POS'N scrollbar at the exact same point as either the START or END. Use the arrows at the end of POS'N to put the pointer at the exact value you want. It might also be a good idea to write this number down.

Now use the COPY RANGE TO NEW SLOT function in the EDIT menu to make a copy of this range to a new file. After you enter a name for this new file, it will take its place on the screen. Next, recall the original sample and make sure that the POS'N cursor is where you left it.

You can now use the INSERT MARKED RANGE function to paste the new sample in where the POS'N cursor is now located. If the sample is fairly short you might want to repeat this last step several times.

For example, a sample that is about 2,000 units long recorded at a speed of 10,000 will only last 1/5 of a second. However, if you keep the POS'N cursor at the same place and use the INSERT MARKED RANGE function ten times in a row, you then increase the length of your sample by two seconds. That might not sound like much until you realize that most of the samples are only about one or two seconds long.

The Perfect Sound Sound Editor is an older shareware program that is no longer sold separately from the digitizer. For more information about Perfect Sound, contact:

SunRise Industries
270 E. Main Street
Suite C
Los Gatos, CA 95032
(408) 354-3488

To receive a copy of not only this program but the Perfect Sound sound digitizer and a library of recorded sounds, send $69 to:

ComputAbility
5139 W. Clinton Ave.
Milwaukee, WI 53223
(800) 558-0003

7.3 Other programs

This last group of programs goes into the category of miscellaneous software—tools that loosely apply to the subject of Amiga music, without being music entry or sampling tools.

7.3.1 MakeSounds

MakeSounds is a public domain program that can be used to graph some (but not all) of the instrument samples that are included in the MED program. These samples have to be in the IFF format in order for the MakeSounds program to recognize them.

MakeSounds will also let you edit sound samples to create your own instruments. For example, you could take the ElecBass sound sample from the MED program and expand or compress it slightly (in other words, either shorten or lengthen it). You can also change the frequency of the sample and save it as a new instrument for your MED samples.

The MakeSounds program also comes with several of its own instrument files. These can also be copied and loaded into the MED program to expand the number of instrument samples available there.

To get a copy of MakeSounds, contact:

Mike Posehn
Granite Bay Software, Inc.
8280 Christian Lane
Loomis, CA 95650

7.3.2 CUCUG JukeBox Disks

The Champaign Urbana Commodore Users Group (CUCUG) offers two JukeBox disks from their collection of public domain and shareware. Both disks include a collection of digitized songs written with the Sonix music program.

These disks are mainly used to play prerecorded music. In fact, you could think of them the same way you would an audio cassette or CD. However, the sound quality is extremely good, particularly if you have your Amiga hooked up to a stereo system.

If you want to get a true idea of what an Amiga can accomplish with audio, these disks give you a very low cost demonstration. When most people think of computer music, they only think of the rather primitive sounds produced by video games and novelty Christmas cards. These disks will show you (and any friends you want to impress) that your Amiga can generate music that is almost impossible to tell apart from the background music of many popular songs on the radio.

JukeBox Disk #1 contains 11 different songs that were quite popular when the disk was first released in 1987. Although the songs seem a bit dated now, they still offer a good demonstration of Amiga audio. Other JukeBox disks may be available from CUCUG at this writing—write them for information.

You can play each of these songs individually, or play all of them in a 25 minute loop. The songs include:

- AxelF
- BabyEL
- CoolTunes
- GhostBust
- Maniac
- MiamiVice
- StandByMe
- Thriller
- ViewKill
- Yes
- YouBelong

JukeBox Disk #2 features 29 different holiday songs:

- Angels on High

- What Child is This
- Come All Ye Faithful
- Deck the Halls
- Drummer Boy
- Faith
- Faries
- Frosty the Snowman
- Grandma Got Run Over by a Reindeer
- Halleuja Chorus
- Hark the Herald Angels Sings
- House
- It Came one Night
- Jesu
- Jolly Holly Christmas
- Kings (we three)
- LChristmas
- Little Town of Bethlehem
- Manger (away in)
- O Holy Night
- Rudolph the Red-Nosed Reindeer
- Santa Claus is Coming
- Silver Bells
- Snow

- Through
- Tree
- We Wish You a Merry Christmas
- Xmas2
- Xmas4

You can buy any of the CUCUG disks by sending $7.00 each to the address listed at the end of the section on the MED program.

7.3.3 CUCUG Sound Effects Disks

At this writing CUCUG also offers two disks with different sound effects on them. The sound effects are written on a public domain program called "Sound" which can be accessed by following the documentation included on each disk.

Compared to the sound effects included with the Perfect Sound Sound Editor, however, there are very few editing functions that you can perform on these sounds. These disks are more like the JukeBox programs in that they are mostly made for just playing back the recorded sounds.

Most of these sound effects are just a few seconds long and include a famous line or two from popular movies and television shows. We haven't quite figured out how to include these files in other Amiga programs, but they are quite entertaining on their own.

Although the titles are somewhat descriptive, you'll just have to use your imagination to figure out what's included on these disks. Sounds #1 includes 17 different effects:

- .44 Magnum
- BrainOut
- DeadJim
- Doggedly
- Friend
- Rodney
- Sorry, Dave
- Stooges1
- Stooges2
- Stooges3

- I Feel Fine.
- MaeWest
- MakeMyDay
- Operational
- Toast
- Trek Theme
- Who's Bad?

Sounds #2 includes what are considered "traditional" sound effects, as well as famous lines. The 30 sound effects on this disk include:

- Boink1
- Boink2
- Boink3
- Bowman
- Buzzer
- CarHorn
- Don't_Worry
- Drip
- FirstLesson
- GoodMorning
- HeliIdle
- Hey-You
- HolyHole
- Horn
- Horn3
- Laugh
- LowBuzzer
- MachineGun
- Magilla
- ManScream
- Ow!
- Rococo
- Spanish
- ThatsIt
- Thud
- Twist-N-Shout
- Whistle
- WomanScream
- Woody
- YabbaDabba

You can buy any of the CUCUG disks by sending $7.00 each to the address listed at the end of the section on the MED program.

Appendices

Appendix A Glossary

Accidental	A sign that is used to raise or lower the pitch of a note or to cancel this type of change (see Sharp and Flat).
ADSR (Attack, Decay, Sustain and Release)	This determines the time required for the sound to reach its highest intensity, the time needed for the sound to return to its normal level, the duration of the sound at this level and its duration of release.
Analog sounds	A sound made by either a computer or a synthesizer. They are produced by one or several oscillators. Usually these are square, sine and triangle waves.
Audio.device	Any computer device that accepts and/or produces sound. Devices comprise the operating level immediately above the hardware level.
Bar line	This is a vertical line separating one measure from the next.
Binary rhythm	Rhythm used in popular music; usually in 4:4 time. Also called duple rhythm or duple meter.
BPM (Beats Per Minute)	Value which corresponds to the beating of a metronome. The number of beats per minute represents the tempo.
Clef	A symbol indicating the range of a staff. Popular clefs include the G (treble) clef, F (bass) clef and the C (alto and tenor) clef.
Comma	Nine tonal intervals within a half step.
Crescendo	A progressive increase of sound.

Decrescendo	A progressive decrease of sound.
Digital sounds	These sounds are created by reproducing a sound that was memorized beforehand instead of using oscillators.
Digitizer	Converts analog information into 8-bit digital data.
DMA (Direct Memory Access)	The audio channels are tied into Paula through DMA, which executes audio information at the fastest possible speed.
Double bar	This is a vertical line (see Bar line) usually signifying the end of a composition or the end of a section.
Duple meter	Meters grouped in twos or duplets. Sample meters are 2/4, 3/4 and 4/4.
Duration	This represents the interval of time between the beginning and the end of a sound.
Eighth note	This is a note that is equal in time to one-eighth of a whole note.
Eighth rest	This corresponds to a silence that lasts the same amount of time as an eighth note.
Even temperament	Perfected during the 1700s. Even temperament, used with keyboard instruments and fretted string instruments, tunes the instrument so that the distance between each pitch is equal over the course of an octave. This allows the musician to play the instrument in any key, with equal intonation.
Flat	This is an accidental that lowers the pitch of a note by one half note.

Fourier	A scientist who demonstrated that all natural sounds can be broken down into an infinite number of sine waves; each sound has its own frequency and intensity.
Frequency	This determines whether a sound is low or high pitched.
GHalf note	This is a note that is equal in time to one half of a whole note.
Half rest	This corresponds to a silence that lasts the same length of time as a half tone.
Half steps	This is the smallest interval on a keyboard.
Intensity	This corresponds to the volume of a sound. It determines whether you're playing loudly.
Key signature	Musical shorthand which indicate the key of a musical composition.
Measure	This is a musical unit on a staff, separated from other measures by bar lines.
Metronome	A device that is used to sound a steady beat.
MIDI	Acronym for Musical Instrument Digital Interface. Electronic equipment that allows a computer and MIDI-compatible electronic musical instruments to communicate.
MIDI interface	A device that enables you to connect a MIDI-compatible device to a computer.
Musical alphabet	These are the seven basic note names: A, B, C, D, E, F and G. Some European cultures use the letter name H to represent B natural, and the letter B to represent B flat.
Noise	A wave shape generated by a random sound.
Octave	This is the largest named interval.

Paula circuit	A chip, inside the Amiga, which is responsible for sound synthesis and converts digital values into analog signals.
Pitch	The sounding frequency of a note. Pitches are usually named from the musical alphabet (A, B, C, etc.).
Ports	These transmit messages between the audio.device and your program.
Quantization	This is a slight rhythmic irregularity which is used in some drum machines, making the electronic drum line sound more "human."
Quarter note	This a note that is equal in time to one fourth of a whole note.
Quarter rest	This corresponds to a silence that lasts the same amount of time as a quarter note.
Retardations	This is when a note is suspended.
Sampler	This is an electronic keyboard instrument capable of reproducing real life sounds electronically. See also Digitizer.
Savart	This is the smallest interval that the human ear can detect. Also the name of a French physicist who experimented with acoustics.
Scale	These are the notes: C, D, E, F, G, A, B.
Sharp	This is an accidental that raises the pitch of a note by one half note.
Sine wave	This is the purest waveform. It cannot be completely reproduced in natural sound.
Sixteenth note	A note that lasts as long as one sixteenth of a whole note.
Sixteenth rest	This corresponds to the sixteenth note.

Sixty-fourth note	A note that is equal in time to one sixty-fourth of a whole note.
Sixty-fourth rest	This corresponds to silence that lasts as long as a sixty-fourth note.
Slur	This is used to indicate a smooth musical phrase.
Sound wait list	When a SOUND command is encountered, it will be placed into the wait list instead of being executed.
Staff	Notes are placed on this, which consists of five horizontal bars. Staffs contain various symbols which guide the musician when playing a piece of music.
Steps	This is the distance between two adjacent notes on a scale.
Synthesizer	Electronic keyboard instruments.
Tempo	This is how fast a piece of music is performed.
Temperament	Tuning. Temperament can change from key to key on flexible-tuning instruments such as violins or flutes. Fretted or keyboard instruments (e.g., guitars or organs) are tempered in such a way that they can be played in any key with equal tuning (see Even temperament). Some synthesizers allow temperament changes as well as standard pitch adjustment for tuning to other instruments.
Ternary form	This is a musical form that consists of three sections. The third section is identical to the first. It is often used in instrumental music.
Thirty-second note	This is a note that lasts as long as one thirty-second of a whole note.

Thirty-second rest	This corresponds to a silence that lasts the same amount of time as a thirty-second note.
Tie	This is used to join two separate notes.
Timbre	This represents the different tone qualities of two sounds that have the same intensity and pitch.
Time signature	Musical shorthand of two numbers, indicating the number of beats per measure and which note unit receives one beat.
Triple meter	Meters grouped in threes or triplets. Sample meters are 6/8, 9/8 and 6/4.
Vibrato	This is a slight wavering in pitch, which occurs so quickly that it resembles a vibration and sounds like a single pitch.
Waveform	This represents a sound wave. Some scientists believe that any sound's waveform can be broken down into multiple sine waves.
Wave shape	Waves formed by analog sounds.
Whole note	The largest unit of time in musical notation.
Whole rest	This corresponds to a silence that lasts the same length of time as a whole note.

Appendix B List of MIDI codes

The following is a list of all available MIDI codes. We have included the name of the MIDI message, its meaning, and the codes to send. The codes are indicated in binary and decimal form. Certain instruments do not interpret some codes.

MESSAGE	MEANING	CODES		
Note off	Indicates the end of a note's transmission (sent after the note on code)	1001nnnn	0kkkkkkkk	00000000
		144-159	36-96	0
		Channel	Note	Note off
Note on	Indicates a note's transmission	1001nnnn	0kkkkkkkk	0vvvvvvvv
		144-159	36-96	1-127
		Channel	Note	Dynamic
After touch	Indicates additional pressure on a key	1101nnnn	0vvvvvvvv	
		208-222	0-127	
		Channel	Value	
Program change	Changing the memory	1100nnnn	0pppppppp	
		192-208	0-127	
		Channel	Program	
Modulation	Vibrato effect	1011nnnn	00000001	0vvvvvvvv
		176-191	1	0-127
		Channel	Code	Module
Pitch bend	Varies the note's pitch up or down	1110nnnn	00000000	0vvvvvvvv
		224-239	0	0-127
		Channel	Code	Speed
Hold on	Sustains the note	1011nnnn	01000000	0xxxxxxxx
		176-191	64	1-127
		Channel	Code	Duration
Hold off	Releases the note (sent after the Hold on code)	1011nnnn	01000000	00000000
		176-191	64	0
		Channel	Code	End
Portamento on	Activates portamento (carry over between one note to the next without a break)	1011nnnn	01000001	0xxxxxxxx
		176-191	65	1-127
		Channel	Code	Strength
Portamento off	Deactivates portamento	1011nnnn	01000001	00000000
		176-191	65	0
		Channel	Code	End
Portamento time	Duration of portamento	1011nnnn	00000101	0vvvvvvvv
		176-191	5	0-127
		Channel	Time	Duration

MESSAGE	MEANING	CODES		
Local on	Deactivates the external instrument connected to the MIDI interface (e.g., the keyboard)	1011nnnn 176-191 Channel	01111010 122 Local	01111111 127 On
Local off	Reactivates the external instrument	1011nnnn 176-191 Channel	01111010 122 Local	00000000 0 Off
All notes off	Deactivates all the notes	1011nnnn 176-191 Channel	01111011 123 Code	00000000 0 Off
Omni on	Instrument receives note information on all its available channels	1011nnnn 176-191 Channel	01111101 125 Code	00000000 0 On
Omni off	The instrument reverts to its initial mode (usually one channel)	1011nnnn 176-191 Channel	01111100 124 Code	00000000 0 Off
Mono on	Permits the selection of individual instrument channels	1011nnnn 176-191 Channel	01111110 126 Code	0000nnnn 0-15 Channel ad
Poly on	Permits polyphonic playback of all instrument voices over one channel	1011nnn 176-191 Channel	01111111 127 Code	00000000 0 On
Song pos pointer	Indicates the pointer's position in the music	11110010 242 Code	0lllllll 0-127 LSB	0hhhhhhh 0-127 MSB
Song select	Selects a piece of music	11110011 243 Code	0sssssss 0-127 Score	
Tune request	Tunes all the instruments according to their initial key	11110110 246 Code		
Timing clock	Impulse clock	11111000 248 Code		
Start	Starts a sequence	11111010 250 Code		
Continue	Reactivates an interrupted sequence	11111011 251 Code		

MESSAGE	MEANING	CODES
Stop	Interrupts a sequence	11111100 252 Code
Active sensing	Tests whether the connections are active	11111110 254 Code
System reset	Returns all the instruments to their initial mode	11111111 255 Code
System exclusive	Starts the exclusive information	11110000 240 Code
EOX	End of System Exclusive - Ends the exclusive information	11110111 247 Code

NOTE	CODE	NOTE	CODE
C	36	C#	37
D	38	D#	39
E	40	F	41
F#	42	G	43
G#	44	A	45
A#	46	B	47
C	48	C#	49
D	50	D#	51
E	52	F	53
F#	54	G	55
G#	56	A	57
A#	58	B	59
C	60	C#	61
D	62	D#	63
E	64	F	65
F#	66	G	67
G#	68	A	69
A#	70	B	71
C	72	C#	73
D	74	D#	75
E	76	F	77

NOTE	CODE	NOTE	CODE
F#	78	G	79
G#	80	A	81
A#	82	B	83
C	84	C#	85
D	86	D#	87
E	88	F	89
F#	90	G	91
G#	92	A	93
A#	94	B	95
C	96		

Appendix C MIDI Sequencers

MIDI sequencing software is unlike any of the software products discussed earlier in this book. A MIDI sequencer doesn't control the Amiga's sound devices. Instead, an Amiga sequencer package can only be used to control MIDI-compatible instruments from the Amiga through a MIDI interface.

A major goal of MIDI was to allow communication between electronic keyboard instruments. For example, prominent rock keyboardists of the 1970s maintained huge racks of multiple keyboard instruments. These racks gave the keyboardist a battery of sound options without changing voices. Sound changes on the old analog synthesizers of the time involved changing patch cords between sound modules, twiddling knobs and retuning the instrument.

MIDI allowed the keyboardist to switch quickly between keyboards and sounds at the press of a few buttons, and control all the instruments from a single (master) keyboard. Thus, MIDI's early application was to let keyboard (and other) instruments "talk" to each other through the performer.

Another goal of MIDI created a new form of sound control. The patch librarian was a software package designed for MIDI equipped computers. Earlier we mentioned patch cords, which allowed the musician to change sounds on analog synthesizers. The term patch librarian is a carryover from the days of patch cords. Different patch librarians are available for different synthesizers. The musician can define, reshape and refine a sound from the patch librarian software, send the new sound (or patch) to the keyboard, and try the sound out. If the sound is to the musician's liking, the patch can be retained in the synthesizer, or saved to diskette for later recall.

Musical instrument manufacturers eventually saw the need for equipment designed exclusively for MIDI recording—equipment that would allow direct recording of a musical instrument, editing, and direct playback through the instrument. From this concept grew the *sequencer*.

The early MIDI sequencers were dedicated (single-purpose) devices. These instruments included a disk drive and multiple MIDI ports (MIDI IN, MIDI OUT, MIDI THRU).

Even though these dedicated MIDI sequencers were useful, most home computers are capable of performing the same tasks. So, many hardware and software packages for all kinds of computers (including the Amiga) appeared on the market.

It's easy to see why the first software programs (especially DMCS and Sonix) were designed to utilize the Amiga's audio resources. Adding MIDI capabilities to these tools further enhanced them.

It was a natural transition to shift sequencers to computers. To fulfill the need for Amiga-based sequencing software, products such as Studio 24, Track 24, Master Tracks Pro and Big Band were developed. These MIDI sequencers exploit many of the Amiga's advantages, such as multitasking, a large memory, and quick access to hard disks.

The basic principle of a sequencer is simple. The MIDI system recognizes 16 channels that can be assigned to one or more synthesizers, or one or more channels on a single synthesizer. A sequencer accesses a certain number of tracks (usually 24 tracks are enough) that can be assigned to a MIDI channel.

Since the Amiga is linked to the synthesizers through a MIDI interface, the software programs inform the synthesizers either to read the values of the notes that were played (in the recording mode) or to tell the synthesizer which note to play.

Conversely, many sequencers can be instructed by the user to sense the impulses received from the synthesizer keyboard (i.e., the keystrokes made by the musician at the synthesizer keyboard), and store these impulses in memory. Once in memory, the musician can edit, play back and save these notes using the computer.

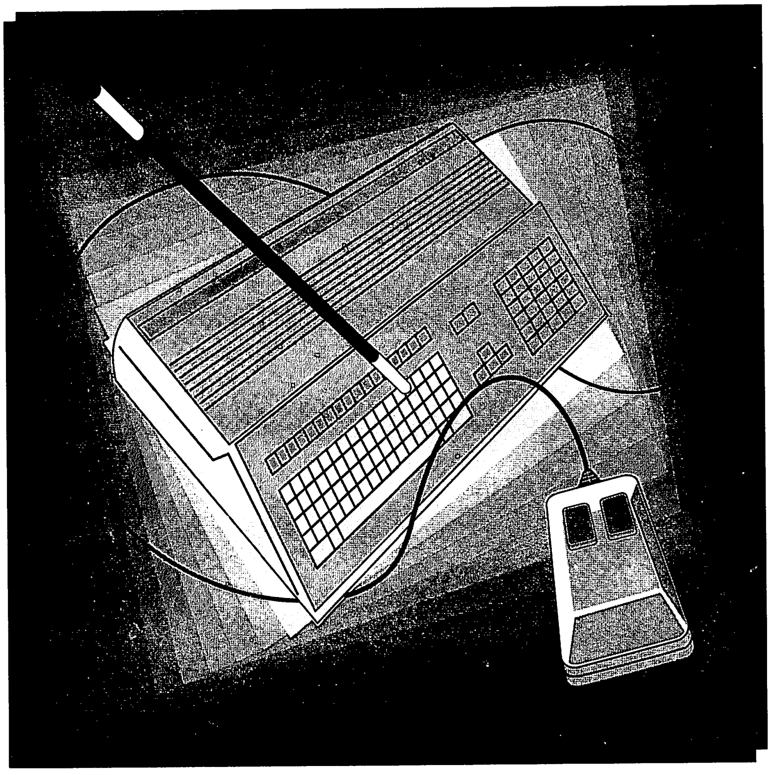
The number of tracks that are assigned to a synthesizer depend on how many lines it has (e.g., an 8 channel synthesizer can be assigned eight different tracks). If you've recorded in multitrack mode on tape recorders, you should be familiar with this.

Some elaborate synthesizers have multitimbral capability (i.e., you can assign a different voice type to each channel of the synthesizer). Multitimbral features let the musician score more complicated works on one or two synthesizers, such as orchestral or jazz band compositions.

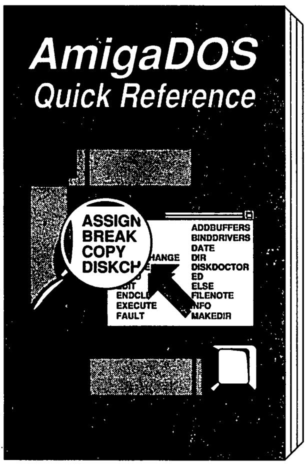
Index

Index

Abacus Amiga Catalog



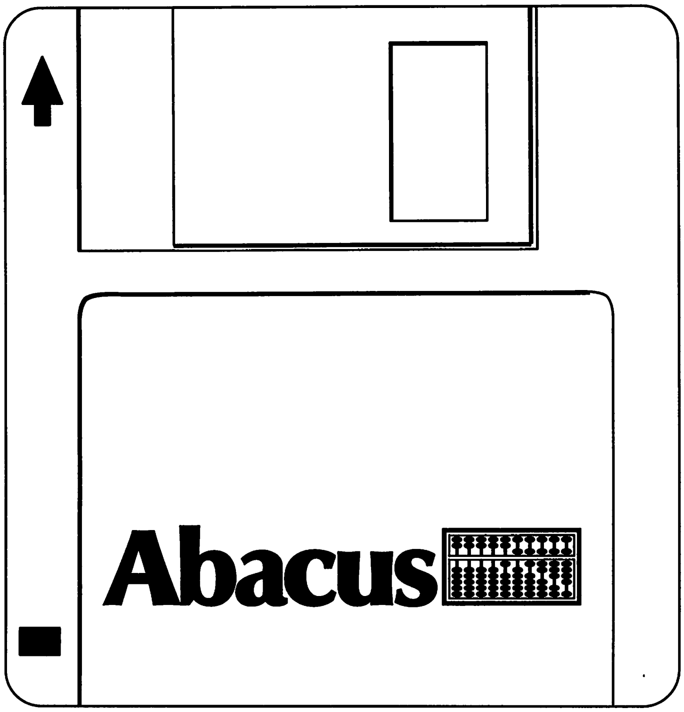
Order Toll Free 1-800-451-4319



Abacus Amiga Book Summary			
Vol.1	Amiga for Beginners	1-55755-021-2	$16.95
Vol.2	AmigaBASIC: Inside and Out	0-916439-87-9	$24.95
Vol.3	Amiga 3D Graphic Programming in BASIC	1-55755-044-1	$19.95
Vol.4	Amiga Machine Language	1-55755-025-5	$19.95
Vol.6	Amiga System Programmers Guide	1-55755-034-4	$34.95
Vol.7	Advanced System Programmers Guide	1-55755-047-6	$34.95
Vol.8	AmigaDOS: Inside and Out	1-55755-041-7	$19.95
Vol.9	Amiga Disk Drives: Inside and Out	1-55755-042-5	$29.95
Vol.10	'C' for Beginners	1-55755-045-X	$19.95
Vol.11	'C' for Advanced Programmers	1-55755-046-8	$24.95
Vol.13	Amiga Graphics: Inside & Out	1-55755-052-2	$34.95
Vol.14	Amiga Desktop Video Guide	1-55755-057-3	$19.95
Vol.15	Amiga Printers: Inside & Out w/ disk	1-55755-087-5	$34.95
Vol.16	Making Music on the Amiga w/disk	1-55755-094-8	$34.95
Vol.17	Best Amiga Tricks & Tips w/ disk	1-55755-107-3	$29.95
	AmigaDOS Quick Reference	1-55755-049-2	$9.95

Cut carefully along this line

Companion Diskette Enclosed



```
Amiga Music disk contents
     Ch.3 (dir)
          AUDIODEVICE.GFA          AUDIODEVICE.GFA.info
          harmony.bas              harmony.bas.info
          lulli.bas                lulli.bas.info
          MODULATION.GFA           MODULATION.GFA.info
          rndtim1.bas              rndtim1.bas.info
          rndtim2.bas              rndtim2.bas.info
          rndtim3.bas              rndtim3.bas.info
          scale.bas                scale.bas.info
          SONDMA.GFA               SONDMA.GFA.info
     Ch.4 (dir)
          8SVXPLAYER.GFA           8SVXPLAYER.GFA.info
          SMUSPLAYER.GFA           SMUSPlayer.GFA.info
          SSMUS.GFA
     songs (dir)
          beep                     MakingPlans
          PanFlute                 shook
     Partitions (dir)
          Example.SMUS
     ASSEMBLER (dir)
          sample.asm               sample.asm.info
          Sampler                  Sampler.info


          read.me                  Read.Me.info
          gfabasro                 Trashcan.info
```

Making Music on the Amiga®

The Amiga has an orchestra deep within it, just waiting for you to give the down beat.

Making Music on the Amiga takes you through all the aspects of music development on this great computer. Whether you're learning the fundamentals of music notation, the elements of sound synthesis or special circuitry to interface your Amiga to external musical instruments, you'll find it in this book.

Making Music on the Amiga explores many ways to reproduce sound. Don't have a MIDI interface? Build one! Want to sample sounds for later playback? Build a digitizer - you'll find plans for both printed here.

Making Music on the Amiga contains both basic information and detailed ways to make different music applications happen.The companion diskette contains public domain music in AmigaBASIC, C, GFA BASIC and assembly language.

US $34.95

ISBN 1-55755-094-8

9 781557 550941

Comprehensive guide to understanding music on the Amiga

Topics include:

- Basics of sound generation: Oscillations, digital sound, effects, delay, sustain, etc.
- Music programming in AmigaBASIC: Waveforms, timbre (tone quality) definition, frequency computation, etc.
- Hardware programming in GFA BASIC: Addressing specific registers, sound modulation (frequency and intensity), accessing audio.device, etc.
- IFF formats (8SVX and SMUS): Structure, sound tutorial programs, IFF access
- MIDI fundamentals: Concept, function, parameters, schematics and applications
- Digitization: Capturing and editing sound, schematics and applications
- Applications: Using *Perfect Sound*, *Aegis Sonix*, , *Deluxe Sound Digitizer*, *Deluxe Music Construction Set*, *Audio Master*, *Dynamic Drums*

Abacus

5370 52nd Street SE • Grand Rapids, MI 49512