The Commodore 64 Music Book

SHIVA's friendly micro series

J Vogel and
N B Scrimshaw



2St50.00
13 DEC 29.85

The Commodore 64 Music Book

The Commodore 64 Music Book

James Vogel
Nevin B. Scrimshaw



Shiva Publishing Limited

This book was first published in America by Birkhauser Boston, Inc., entitled *The Commodore 64 Music Book: A Guide to Programming Music and Sound*. The text has been adapted to the UK market.

SHIVA PUBLISHING LIMITED
64 Welsh Row, Nantwich, Cheshire CW5 5ES, England

© Birkhauser Boston, Inc., 1983

ISBN 1 85014 019 7

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording and/or otherwise, without the prior written permission of the Publishers.

This book is sold subject to the Standard Conditions of Sale of Net Books and may not be resold in the UK below the net price given by the Publishers in their current price list.

Typeset and printed by Devon Print Group, Exeter

Contents

Foreword 1

Section I: Electronic Music Synthesis and the SID Chip

Introduction 4
1 Sound and Vibration 5
2 Frequency and Pitch 7
3 Waveforms 9
4 The Envelope Generator 12
5 Understanding Filters 14
6 Ring Modulation, Synchronization and Timbre Modulation 16

Section II: Programming Sound Controls

Introduction 20
7 Understanding the POKE Statement 22
8 Programming Pitch Controls 26
9 Advanced Pitch Control Techniques 28
10 Controlling Waveforms 30
11 Programming the ADSR Controls 33
12 Programming Volume Controls 37
13 Programming Note Duration 38

Section III: Programming Music Using READ/DATA

Introduction 42
14 READ/DATA Format 43
15 Setting Up the DATA Bank 46
16 Duration Control 48
17 The RESTORE Statement 50
18 Using Counters 52
19 Programming a Song for One Voice 54
20 DATA Manipulation 57
21 Short-cuts and Abbreviations 61
22 Multiple Voice Programming 63

Section IV: Special Sound Effects

	Introduction	70
23	Phaser and Siren Sound Effects	71
24	The White Noise Effect	75
25	Sound and Colour	77
26	Using the Random Number Function	81
27	Ring Modulator and Synchronization Effects	83
28	Programming Filters	87
29	Frequency Modulation	90

Section V: Programming Music using the Array Format

	Introduction	96
30	Simple Arrays	97
31	Using the Two-dimensional Array Format	100
32	Creating Two Characters with One Voice	104
33	Multiple Voices with Independent Duration	107
34	Real Time Music: An Organ Keyboard	113

Section VI: Finale

	Sounding a Final Note	120
	Bibliography	121
	Table of Note Values	122
	SID Chip Control Chart	124
	Keyboard ASCII Codes	125

Foreword

This book is designed to teach you about programming music and sound on your Commodore 64 home computer. You don't have to be an experienced programmer to use this book. We'll show you, step by step, how to program sound and musical ideas. If you're not a musician, we'll show you ways to create sound effects that you may find useful in some of your programming ventures. If you have an interest in music, we will offer ways to apply your musical knowledge and composition ideas to the realm of electronic computer music.

Your Commodore 64 is equipped with one of the most sophisticated sound chips on the home computer market today. It's called the SID chip—shorthand for Sound Interface Device. With this chip, the Commodore's 64K memory, its advanced colour graphics capabilities and easy-to-use BASIC language, the opportunities for creative experiments are almost as limitless as the human imagination.

This book treats the computer as a musical instrument. As with any instrument, a proper foundation in technique is always a good place to start. First notes are often tentative. But, as any musician will tell you, learning a new instrument can be as exciting as discovering a new world. In this case you will not be learning how to pluck strings, finger keyboards, or make a reed vibrate. You will be learning how to program electronic sound. You will learn about a process called Electronic Sound Synthesis that will enable you to take your computer/instrument, and not only make it sound as if it's plucking strings or fingering the notes of acoustical instruments, but actually create sounds and tones never before heard from these instruments.

With just a few BASIC and/or assembly language statements and commands, you can make your computer create all sorts of complex sound effects. These sound effects can be inserted into video game programs or incorporated into your computer graphic art projects. Imagine an instrument that can at one moment sound like rain falling on tin drums or a roaring brook in the springtime, and the next moment sound like a cow mooing or a rocket blasting off! Imagine an instrument that can play notes as fast as you can hear them, or one that can play your compositions—not only forwards, but backwards and even inside-out! How about an instrument that will assign colours to your notes and

create a graphic display of your music? Or maybe, an instrument to play an accompaniment while you play your flute or guitar? Finally, how about an instrument that will create its own compositions and original music?

All this and more is possible with your Commodore 64—and *you* can do it.

Acknowledgements

Special thanks to Anna Kirwan Vogel for manuscript preparation, and The Electronics Center of Northampton, USA, for technical assistance.

Section I:

Electronic Music Synthesis and the SID Chip

Introduction

These opening chapters are designed to give you an elementary understanding of electronic sound synthesis, as well as to introduce you to some of the 'ins and outs' of the SID chip. We suggest that you take the time now to familiarize yourself with this information. It will make your understanding of what follows that much greater.

Each chapter in this section will provide an explanation of a particular aspect of creating sound, followed by a program to demonstrate the 64's abilities in this context. At this stage, we want you to hear what some of the computer's sound controls can create. Later on, we'll show you how to use them.

If you are already familiar with these concepts, or if you are the type of person who likes to jump right in and experiment, by all means, skip ahead. After all, theory often comes after the fact of creation. But do take the time, somewhere along the line, to come back to this material. If you're going to build a house, you might as well start with a strong foundation; and if you're going to build a sound, it's a good idea to do the same. We've given you this formatted material first in the belief that it's better to understand what it is that you're trying to accomplish before you go ahead and program: that way, you're not just programming a bunch of meaningless numbers; you're learning how the computer works with the SID chip to produce the desired end.

1 Sound and Vibration

Sound is produced when a vibrating source sets air particles into motion. As the vibrating source pushes out, it compresses the air particles around it. As it pulls in, it pulls them apart. This is known as *compression* and *rarefaction*. As the air particles around the vibrating source undergo this compression and rarefaction, they bump into other air particles, setting them into motion. This is how the vibrations are carried through the air to the listener. A series of compressions and rarefactions is called a *sound wave*.

People have developed many different kinds of instruments which act as vibrating sources. Some of the first included sticks and rocks. Later on, drums were created by stretching skins over hollow logs. Still later came flutes, which used breath to set the air vibrating. And then vibrating strings were added and bellows were used to push air through organ pipes and reeds. All of these instruments produced sound by directly setting the air into motion.

Electronic instruments, such as our computer with its sound chip, produce sound in an indirect manner. They have oscillators which are signal-generating components. Electronic instruments produce variations in electrical current. This current is then processed through various filters and sound envelopes which we can program. The current then goes to an amplifier, which increases its power so that it can drive a speaker. The speaker, in turn, becomes the vibrating source which creates the sound waves.

Your Commodore 64 has three independently controlled oscillators, or voices, which you can program for pitch, waveform selection, and volume or amplitude. In addition, you can use the oscillators together to get complex effects such as *ring modulation* and *synchronization*.

If you would like to hear these oscillators in action, try typing in this program. It utilizes all three voices, each playing a different waveform and pitch.

```
 5 FOR Z = 54272 TO 54296 : POKE Z,Ø : NEXT
1Ø M = 54272 : POKE M+24,15
3Ø POKE M+5,168 : POKE M+6,169
4Ø POKE M+12,88 : POKE M+13,89
```

```
 5Ø  POKE M + 19,88 : POKE M + 2Ø,89
 6Ø  READ A,B,C,D,E,F,G
 7Ø  IF A < Ø THEN POKE M + 24,Ø : END
 8Ø  POKE M + 1,A : POKE M + Ø,B
 9Ø  POKE M + 8,C : POKE M + 7,D
1ØØ  POKE M + 15,E : POKE M + 14,F
11Ø  POKE M + 4,17 : POKE M + 11,33 : POKE M + 18,65
12Ø  POKE M + 16,128 : POKE M + 17,Ø
13Ø  FOR Z = 1 TO G : NEXT
14Ø  POKE M + 4,16 : POKE M + 11,32 : POKE M + 18,64
15Ø  GOTO 6Ø
2ØØ  DATA 15,21Ø,Ø,Ø,Ø,Ø,2ØØØ
21Ø  DATA 15,21Ø,21,31,Ø,Ø,2ØØØ
23Ø  DATA 15,21Ø,21,31,23,181,2ØØØ
24Ø  DATA 28,49,21,31,23,181,2ØØØ
25Ø  DATA 28,49,37,162,23,181,2ØØØ
26Ø  DATA 28,49,37,162,42,62,2ØØØ
27Ø  DATA 28,49,21,31,31,165,3ØØØ
8ØØ  DATA -1,-1,-1,-1,-1,-1,-1
```

On a new line, type RUN and you will hear the 64 at work.

If you are using your TV as a monitor, the chances are that you are also using it for sound. If you want to hear a richer, clearer sound, you can easily connect the audio output located on the back of your computer to your stereo or to an instrument amplifier.

2 Frequency and Pitch

We have said that a sound wave consists of a series of compressions and rarefactions that set the air in motion. The amount of time that it takes for one compression and rarefaction to occur is called a *cycle*. The *frequency* of a sound is determined by the number of cycles per second. When a sound wave is vibrating at a regular frequency, we say that it is a *pitched tone*. Frequency is a quantitative property of sound measured in cycles per second, or Hertz. Pitch is a qualitative or psychological property of sound. Frequency and pitch are not the same acoustically even though we use the terms interchangeably. The average person can hear a sound range with frequencies from about 20 cycles per second to about 20,000 cycles per second.

Your Commodore 64 can produce a full eight octaves' worth of pitches, ranging from a low of 16 cycles per second to a high of 3952 cycles per second. To give you an idea of what this means, compare this range to the human voice, which has a four octave range from about 125 to 1000 cycles per second, or to a grand piano with a range of 27.5 to 4186 cycles per second. Most orchestral music has a range of about 30 to 3951 cycles per second.

Here is a program that will demonstrate the pitch range of your computer.

```
  5  REM *** "PITCH RANGE" *** J.C. VOGEL
 10  FOR M = 54272 TO 54296 : POKE M,0 : NEXT
 20  POKE 54296,15
 30  POKE 54277,0 : POKE 54278,128
 40  POKE 54276,17
 50  FOR T = 1 TO 200 : NEXT
 60  READ A,B
 70  IF A =< 0 THEN GOTO 900
 80  POKE 54273,A : POKE 54272,B
 90  GOTO 20
100  DATA 1,12,1,102,1,145,1,221,2,24,2,204,3,35,3,187,4,48,5,152,6,71,7,119
110  DATA  8,97,11,48,12,143,14,239,16,195,22,96,25,30,29,223,33,135,44,
     193,50,60
```

```
120 DATA 59,190,67,15,89,131,100,121,134,30,179,6,200,243,238,248
130 DATA -1,-1
900 POKE 54296,0
910 END
```

Remember to type RUN on a new line to hear the program.

There are several ways to make the computer produce a desired pitch. We will discuss these in Chapter 8. For now, we just want to familiarize you with what your 64 can accomplish.

Here is another, longer program that demonstrates the chromatic scale for the entire range of the 64.

```
  1 REM *** NICK'S SCALER *** BY J.C. VOGEL & NICK
  5 FOR Z = 54272 TO 54296 : POKE Z,0 : NEXT
 10 FOR I = 0 TO 7
 20 POKE 54296,15
 30 POKE 54277,0 : POKE 54278,128
 40 POKE 54276,17
 50 READ A : IF A = -1 THEN GOTO 900
 60 A = A/2 ↑ I
 70 L% = A/256 : H% = A-(256*L%)
 80 IF TI < TM+10 THEN 80
 90 TM = TI
100 POKE 54273,L% : POKE 54272,H%
110 GOTO 40
200 DATA  64814,61176,57743,54502,51443,48556,45830,43258,40830
210 DATA  38539,36376,34334
800 DATA -1
900 RESTORE
910 NEXT I : FOR T = 1 TO 200 : NEXT : POKE 54296,0
920 END
```

3 Waveforms

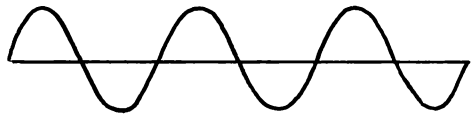
Generating a wide range of pitches is not the only job that the three oscillators perform. They also produce four different waveforms at each of the desired pitches. Each waveform has its own unique tone quality or colour. We call this tone colouring *timbre*, and it is one of the simplest and most powerful sound controls with which you can work. It is the difference in timbre or tone colouration that enables you to distinguish an instrument like the trumpet with its brassy sound from the warm tone of a violin. Timbre is largely the result of waveshape.

Before we go any further with our discussion of waveforms, perhaps you would like to hear them. Try typing in this program on your 64. It's a brief passage from 'The Flight of the Bumble Bee' by Rimsky Korsakoff. The program is designed to play the passage four times. The first time through it demonstrates the *sawtooth* waveform; the second time, the *triangle* waveform; the third time, the *pulse* waveform; and the final time through, it demonstrates the *white noise* waveform.

```
  5 REM *** FLIGHT OF THE BUMBLE BEE ***
 10 FOR M = 54272 TO 54296 : POKE M,0 : NEXT
 20 WF(0) = 33 : WF(1) = 17 : WF(2) = 65 : WF(3) = 129
 30 POKE 54296,15
 40 POKE 54274,9 : POKE 54275,255
 50 POKE 54277,137 : POKE 54278,128
 60 READ H,L
 70 IF H<0 THEN 900
 80 POKE 54273,H : POKE 54272,L
 90 POKE 54276,WF(I)
100 FOR T = 1 TO 50 : NEXT
110 GOTO 60
200 DATA 28,49,26,156,25,30,23,181,22,96
210 DATA 29,223,28,49,26,156,28,49
220 DATA 26,156,25,30,23,181,22,96
230 DATA 23,181,25,30,26,156,-1,-1
900 POKE 54276,0
910 RESTORE : I = I+1 : IF I = 4 THEN END
920 GOTO 60
```

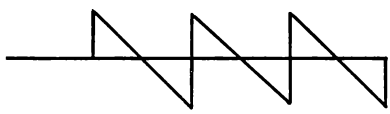
As you can hear, each waveform has its own distinct tone quality. This is because the shape of the wave governs the number and amplitude (volume) of the partials, or *harmonics*, that you hear. If you would like to explore this further, here is a more detailed explanation of waveforms, but don't get discouraged if you don't immediately grasp it. Remember, the SID is doing all this hard work for you. All you have to do is type in a few numbers.

Each pitched soundwave vibrates at a fundamental frequency. This fundamental harmonic, as it is known, is a sine wave. It looks like this on an oscilloscope:

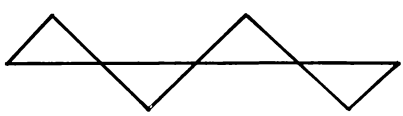


The fundamental is the actual pitch you hear. In addition to this fundamental, other sine waves are generated. The second harmonic vibrates at twice the frequency of the fundamental, while the third vibrates at three times the frequency. Theoretically, there are an infinite number of harmonics. The reason why you don't hear them is because each harmonic is progressively quieter.

In the sawtooth waveform, all the harmonics are present. Each harmonic, however, grows progressively quieter in a ratio that is equal to the reciprocal of the harmonic number. Thus, the second harmonic is one-half as loud as the fundamental, while the third harmonic is one-third as loud. The sawtooth waveform gets its name from its shape, which looks like this:

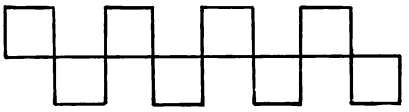


The triangle waveform contains only the odd harmonics in an amplitude ratio equal to the reciprocal of the square of its number. Thus, harmonic two would not be present, while harmonic three would be one-ninth as loud as the fundamental. Similarly, harmonic four would not be present, while harmonic five would be one-twenty-fifth as loud as the fundamental. The triangle waveform looks like this:



The pulse waveform is a rectangular shape. You can program the width of the pulse. Each rectangular wave that you program will exhibit a different harmonic content. That makes this your most variable

sounding waveform. The pulse waveform will look something like this, depending on your pulse width:



You can also create a square waveform with this setting. A square wave will contain only odd harmonics at an amplitude ratio equal to the reciprocal of the harmonic number. Thus, the third harmonic would be one-third as loud as the first, while the fifth would be one-fifth as loud.

The final waveform selection is white noise, which is a random generation of all frequencies at equal energy. This waveform is especially useful for percussive sounds, as well as for a wide variety of sound effects.

As you can see, each waveform generated a different range of harmonics. It is this difference that provides you with a wide variety of tone colours. Once again, the SID does all the work in generating these waveforms. You just tell it which one you'd like. We'll show you how this is done in Chapter 10.

4 The Envelope Generator

We have seen how the oscillators of the SID produce a pitched waveform. The next step in sound production is the control of the amplitude or volume of the wave. The SID accomplishes this with three controls. The first is a master volume setting which remains constant and governs all three voices simultaneously. The second control is an *amplitude modulator*, of which there are three, one for each voice. The amplitude modulator governs the volume output of each oscillator, and is in turn governed by its own *envelope generator*. The third control is the envelope generator which directs the amplitude modulator by defining how loud the sound will be during different phases of its production. These phases are called the *attack*, *decay*, *sustain* and *release* (or 'ADSR') of the sound. Together, they comprise an 'envelope' for the sound, hence the name, envelope generator.

The envelope generator is triggered when the waveform oscillator is *gated*, or 'turned on'. The first phase of the envelope is the attack. The attack controls how quickly the output of the oscillator will rise from zero to its greatest amplitude or volume. The next phase is the decay. The decay setting of the envelope governs how rapidly the output of the oscillator—the sound—will drop from its highest or loudest point to the next phase, called sustain. The sustain setting instructs the amplitude modulator to halt the decay of the sound at a given point and to sustain it at that level for as long as the waveform is gated. When the waveform is turned off, the envelope enters its final phase, which is the release. The release controls how quickly the volume will drop from the sustain level back down to zero.

The ADSR is a truly powerful control. It plays a major role in creating sounds that resemble acoustical instruments, as well as in creating wholly original sounds. This is because each instrument has its own unique envelope.

Consider the piano. It has a very explosive attack, rising rapidly to its peak volume. The sound then drops or decays over a relatively long period. When you remove your finger from the key, the note is released, and the sound quickly drops to zero.

A stringed instrument, such as a cello, can have several different kinds of envelopes, depending on how it is played. If it is bowed, the attack is rather gradual. According to the dynamics of the particular stroke, the

decay can be rather slight, followed by a long sustaining period. When the bow is removed, the note is released and gradually fades away. If the note is plucked, however, the attack will be quick and forceful. Then it will decay over a long period. This is a similar envelope to those for all plucked instruments.

Some instruments, such as the organ, have an instantaneous attack with absolutely no decay. Instead, the sustain holds the note at peak volume until release. The release, like the attack, is then instantaneous.

There are 16 possible settings for each phase of the envelope, so you can create just about any configuration you can imagine. We'll show you how in Chapter 11. If you're curious, try typing in this program. It will demonstrate four separate and unique envelopes using the same waveforms and pitches.

```
  5 REM *** FOUR ENVELOPES ***
 10 FOR M = 54272 TO 54296 : POKE M,0 : NEXT
 20 AD(0) = 144 : AD(1) = 8 : AD(2) = 65 : AD(3) = 0
 30 SR(0) = 243 : SR(1) = 9 : SR(2) = 0 : SR(3) = 128
 40 POKE 54296,15
 50 POKE 54277,AD(I) : POKE 54278,SR(I)
 60 READ H,L,DR
 70 IF H<0 THEN 900
 80 POKE 54273,H : POKE 54272,L
 90 POKE 54276,17
100 FOR T = 1 TO DR : NEXT
110 POKE 54276,16
120 GOTO 50
200 DATA  7,12,300,7,233,300,8,225,300,9,247,250,11,48,250,12,143,250,
    14,24,200
210 DATA 15,210,200,17,195,200,19,239,150,22,96,150,25,30,150,28,49,100
220 DATA 31,165,100,35,134,100,39,223,50,44,193,50,50,60,50,56,99,1750
230 DATA -1,-1,-1
900 RESTORE : I = I+1 : IF I = 4 THEN POKE 54296,0 : END
910 GOTO 50
```

5 Understanding Filters

So far, we have examined how electronic sound is created through the use of oscillators which can generate a wide range of pitches and four distinct waveforms. This pitched waveform is then processed through the amplitude modulator under the direction of the envelope generator, adding further definition to the overall output. At this point, the electronic signal encoding the sound is ready to be passed through the audio output to the amplifier.

In more advanced sound programming, however, we may want to refine our sound further. This can be done with a series of filters that are available in the SID. Again, this is not a necessary procedure when you are first starting out, but it is a highly desirable control to have in any synthesizer: it adds a great deal to the timbre or tone colouration possibilities of your sound. Here is an explanation of how the filters work. We'll show you how to control them in Chapter 28.

The SID chip is equipped with three kinds of filters, which you may use individually or together. You also have the option of using these filters on any or all of your voices at once. The filters are called the 'low-pass' filter, the 'high-pass' filter and the 'band-pass' filter. Their main function is to increase or decrease the amplitude or intensity of frequencies. When we increase the amplitude, we *resonate* the frequency. When we decrease the amplitude, we *attenuate* the frequency. This function is also called 'passing and rejecting'.

You will recall from our discussion of waveforms that a given pitch produces a fundamental tone and a series of harmonics, or partials. The number and intensity of these partials gives the sound its timbre. Through the use of filters, we can further control the harmonic content of our sound. This is called *subtractive synthesis*.

To use the low-pass filter, you first choose a cut-off frequency. The filter will allow all the harmonics below the cut-off to pass unaltered. All the harmonics above the cut-off will be attenuated (decreased) progressively. This is one of the most commonly used filters because it produces a brassy-sounding timbre for pitched notes.

The high-pass filter is the reverse of the low-pass filter. It allows all the harmonics above the cut-off frequency to pass unaltered, and attenuates all those that fall below. This filter is not especially good for pitched

tones because it filters out the fundamental and most of the lower harmonics. It is very useful when used with the white noise waveform, producing a wide variety of sounds.

The third filter is the band-pass filter. It allows only a narrow band of frequencies around the cut-off point. All those above and below the band will be attenuated.

You can obtain the opposite effect of a band-pass filter by combining a high and a low filter. This is known as a band reject filter. It will pass all frequencies above and below the cut-off, and reject those at the cut-off. This is a very specialized filter.

You can further control these filters with a *resonance* control. This control emphasizes the frequencies around the cut-off frequency selected for the filter.

One final word about the SID filter concerns the rate of attenuation. Each filter is permanently set to attenuate the frequencies at a given rate. This rate is measured in decibels (dB) per octave. A decibel is a relative measurement, a ratio between a fixed sound and another sound. It is not an absolute quantity. It has been determined that the smallest difference in loudness between two pitches that the human ear can detect is three dB. It has also been determined that a three dB increase corresponds to a doubling of the sound's intensity.

The SID's low-pass and high-pass filters are set to attentuate frequencies at 12 dB per octave. The band-pass filter attenuates frequencies at six dB.

To get a brief demonstration of the power of these filters, try out this program. It will produce the same tone filtered through a low-pass filter, a high-pass filter, and finally, a band-pass filter.

```
  5  REM *** FILTERS ***
 10  FOR M = 54272 TO 54296 : POKE M,0 : NEXT
 15  POKE 54295,V(I) : POKE 54296,F(I)
 16  POKE 54293,LC(I) : POKE 54294,HC(I)
 20  V(0) = 0 : V(1) = 1 : V(2) = 1 : V(3) = 1 : REM 54295
 30  F(0) = 15 : F(1) = 31 : F(2) = 47 : F(3) = 79
 40  LC(0) = 0 : LC(1) = 0 : LC(2) = 5 : LC(3) = 0
 50  HC(0) = 0 : HC(1) = 100 : HC(2) = 135 : HC(3) = 110
 60  POKE 54295,V(I) : POKE 54296,F(I)
 70  POKE 54293,LC(I) : POKE 54294,HC(I)
 80  POKE 54277,0 : POKE 54278,128
 90  POKE 54273,11 : POKE 54272,218
100  POKE 54276,33
110  FOR DR = 1 TO 2500 : NEXT
120  POKE 54276,32
130  I = I+1 : IF I = 4 THEN POKE 54296,0 : END
140  GOTO 60
```

6 Ring Modulation, Synchronization and Timbre Modulation

The SID offers you three additional ways to alter your sound, other than those discussed in previous chapters.

The first is the *ring modulator*. It's one of the easiest controls to use, and produces a wide array of crazy and funny sounds. When set, the ring modulator multiplies the output of two oscillators, producing the sum and difference of the harmonics. Here is what is sounds like.

```
  5 REM *** POTS AND PANS ***
 10 FOR M = 54272 TO 54286 : POKE M,0 : NEXT
 20 POKE 54296,15
 30 POKE 54277,7 : POKE 54278,0
 40 READ H,L,DR
 50 IF H<0 THEN 900
 60 POKE 54287,H : POKE 54273,L
 70 POKE 54276,21
 80 FOR T = 1 TO DR : NEXT
 90 POKE 54276,20
 95 FOR B = 1 TO 34 : NEXT
100 GOTO 30
200 DATA 18,209,75,21,31,75,23,181,150,28,49,150,28,49,225,31,165,75
210 DATA 28,49,150,23,181,150,18,209,225,21,31,75,23,181,150,23,181,150
220 DATA 21,31,150,18,209,150,21,31,450
250 DATA -1,-1,-1
900 RESTORE : I = I+1 : IF I = 2 THEN POKE 54296,0 : END
910 GOTO 30
```

The *sync* function of the SID synchronizes the fundamental frequencies of two oscillators. If you vary the pitches of the oscillators, you can produce a broad variety of complex harmonics. Type in this program to hear what synchronization can sound like. It adds the sync effect to the 'Nick's Scaler' program used in Chapter 2. We'll show you how to set these controls in Chapter 27.

```
  1  REM *** NICK'S SCALER *** BY J.C. VOGEL & NICK
  2  REM *** SYNC ***
  5  FOR Z = 54272 TO 54296 : POKE Z,Ø : NEXT
 1Ø  FOR I = Ø TO 7
 2Ø  POKE 54296,15
 3Ø  POKE 54277,Ø : POKE 54278,128
 4Ø  POKE 54276,19
 5Ø  READ A : IF A = -1 THEN GOTO 9ØØ
 6Ø  A = A/2 ↑ I
 7Ø  L% = A/256 : H% = A-(256*L%)
 8Ø  IF TI < TM + 1Ø THEN 8Ø
 9Ø  TM = TI
1ØØ  POKE 54287,L% : POKE 54273,H%
11Ø  GOTO 4Ø
2ØØ  DATA 64814,61176,57743,545Ø2,51443,48556,458Ø3Ø,43258,4Ø83Ø,38539,
21Ø  DATA 36376,34334
8ØØ  DATA -1
9ØØ  RESTORE
91Ø  NEXT I : FOR T = 1 TO 2ØØ : NEXT : POKE 54296,Ø
92Ø  END
```

Finally, some of the most dramatic sound colours can be produced by a special function that allows you to modulate a sound, or a series of sounds of a voice, using the output of the oscillator and envelope generator of Voice 3. This modulation can produce complex sound effects, like the Phaser or a Wah-wah pedal. We'll describe how to do this in Chapter 29.

Section II:

Programming Sound Controls

Introduction

Now that you have an overview of the actual process the SID goes through to produce a sound, you're ready to learn how to program its controls. Let's jump right in and construct a short sound program.

```
 5  FOR M = 54272 TO 54296 : POKE M,Ø : NEXT
1Ø  POKE 54296,15
2Ø  POKE 54277,Ø : POKE 54278,128
3Ø  POKE 54272,28 : POKE 54273,49
4Ø  POKE 54276,17
5Ø  FOR T = 1 TO 9ØØ : NEXT
6Ø  POKE 54276,16
```

Here is a line-by-line description of what this program is doing.

Line 5 is a standard command to clear the sound chip of any past programming information that might be in its memory.

Line 1Ø sets the master volume at 15, its maximum setting.

Line 2Ø sets the ADSR controls for Voice 1. The first POKE statement sets the attack/decay cycle for Voice 1, while the second POKE statement sets the sustain/release cycle for Voice 1.

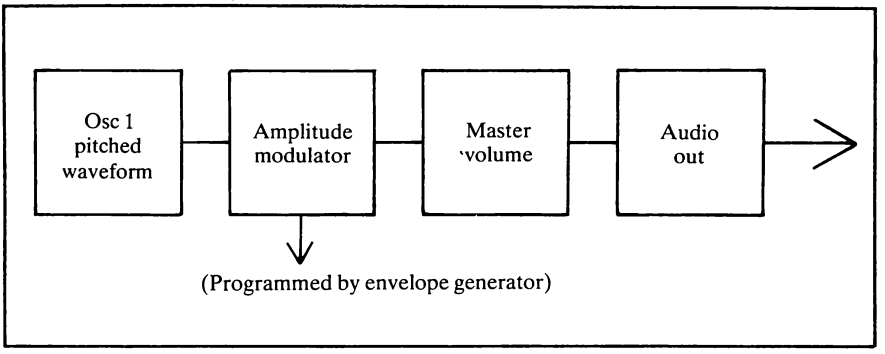
Line 3Ø encodes the pitch, using two POKE statements. The first number encodes the low frequency number for Voice 1. The second number encodes the high frequency. Taken together, they encode the pitch A 44Ø.

Line 4Ø turns on the triangle waveform by opening up a gate. This gate triggers the ADSR to begin its attack/decay and sustain cycles.

Line 5Ø is a time loop. It instructs the computer to wait for a time and then proceed to the next command. This controls the duration of the sound.

Line 6Ø closes the gate, which signals the release cycle of the ADSR and the end of the note.

With these commands, we have instructed the oscillators of Voice 1 to produce a pitched waveform that was passed through an ADSR envelope and volume control.



This is the basic format for programming a sound. Adding filters and ring modulators is like icing on the cake. We'll get to them later. For now, let's take a closer look at how to program these main controls, which are the primary ingredients in our sound recipe.

7 Understanding the POKE Statement

As you probably noticed from our sample programs in Section I, the POKE statement is the primary command used in music programming. The POKE statement instructs the computer to go to a specific control register or *byte* and enter a value. One way to understand this is to picture the SID chip as the control panel of a synthesizer containing a number of dials. These control dials are called bytes, and each dial, or byte, has eight master settings, or *bits*. Each bit represents a value on the dial. By turning the dial to a setting, you get the value of the setting. It's not unlike turning the volume dial on your stereo. Using this analogy, the POKE statement becomes your hand reaching out and turning the appropriate dial to the desired setting.

A POKE statement is usually expressed as follows:

```
10Ø  POKE 54277,75
```

The first number, 10Ø, is the line number of your program. This is followed by POKE 54277, which instructs the computer to go to control register number 54277. This number is known as an *address*. It is the location of a particular byte, in this case, the byte that controls the attack/decay setting for Voice 1.

The SID chip contains 29 of these control registers or bytes. If 29 controls sound like a lot to handle, take heart. The first seven controls operate Voice 1. The next two sets of seven are duplicates of the first. They are used to control Voices 2 and 3. So when you know the first seven, you've already got a handle on 21. The remaining eight bytes control various filter settings and Analog/Digital TA/D converters for interfacing the SID with external audio signals. There is a handy chart of 'SID Chip Control Registers' on page 24.

After POKEing the address of the desired control, enter a comma, followed by a number from Ø–255 inclusive. This number represents the value that you are instructing the computer to enter on to the byte. By entering this value, you are, in effect, directly programming each of the eight bits contained on the byte. But, "why a number between Ø and 255?" you may ask. The reason is, we are using a programming language called BASIC.

The purpose of this language is to act as an intermediary between human language and the computer's language. We use a number system built around 1Ø, the decimal system. The computer uses a system built around two, the binary system. We have 1Ø characters (Ø–9) with which to build our numbers, while the computer has just two (Ø, 1). The computer uses the Ø to turn off a bit and uses a 1 to turn on a bit. When we tell the computer to POKE 54277,75, it will convert our 75 into binary, and use this number as a code to turn on and off the bits located at the byte with the address 54277. It would look like this.

Atk 3	Atk 2	Atk 1	Atk 0	Dec 3	Dec 2	Dec 1	Dec 0
0	1	0	0	1	0	1	1
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Register number 54277

That's because the decimal number 75 is the same as the binary number ØlØØlØ11. Thus, our statement POKE 54277,75 turns on decay bits Ø, 1, 3 and attack bit 6, giving us a combination of three decay settings and one attack setting. So you see, our number 75 does a lot of work.

It's very easy to learn how to make this conversion, so don't be alarmed. You don't have to be a mathematical genius: all you do is add. Follow this simple procedure, and you'll gain control of your sound settings in no time.

128	64	32	16	8	4	2	1
0	1	0	0	1	0	1	1
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

The chart above represents our control register. If you want to turn on a bit, just use the decimal number above the bit when entering your value. If you want more than one bit, just add. The computer will do the conversion for you. We used the number 75. That is made up of 64 + 8 + 2 + 1. If we wanted 76, then our register would look like this:

128	64	32	16	8	4	2	1
0	1	0	0	1	1	0	0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

64 + 8 + 4 = 76

SID Chip Control Chart

		Bit	7	6	5	4	3	2	1	0
		Decimal	128	64	32	16	8	4	2	1
	Address	Function	**Voice 1**							
0	54272	Low FQ	FQ 7	FQ 6	FQ 5	FQ 4	FQ 3	FQ 2	FQ 1	FQ 0
1	54273	High FQ	FQ 15	FQ 14	FQ 13	FQ 12	FQ 11	FQ 10	FQ 9	FQ 8
2	54274	Low PW	PW 7	PW 6	PW 5	PW 4	PW 3	PW 2	PW 1	PW 0
3	54275	High PW	/////	/////	/////	/////	PW 11	PW 10	PW 9	PW 8
4	54276	Waveform	Noise				Test	R mod	Sync	Gate
5	54277	Atk/Dec	Atk 3	Atk 2	Atk 1	Atk 0	Dec 3	Dec 2	Dec 1	Dec 0
6	54278	Sus/Rel	Sus 3	Sus 2	Sus 1	Sus 0	Rel 3	Rel 2	Rel 1	Rel 0
			Voice 2							
7	54279	Low FQ	FQ 7	FQ 6	FQ 5	FQ 4	FQ 3	FQ 2	FQ 1	FQ 0
8	54280	High FQ	FQ 15	FQ 14	FQ 13	FQ 12	FQ 11	FQ 10	FQ 9	FQ 8
9	54281	Low PW	PW 7	PW 6	PW 5	PW 4	PW 3	PW 2	PW 1	PW 0
10	54282	High PW	/////	/////	/////	/////	PW 11	PW 10	PW 9	PW 8
11	54283	Waveform	Noise				Test	R mod	Sync	Gate
12	54284	Atk/Dec	Atk 3	Atk 2	Atk 1	Atk 0	Dec 3	Dec 2	Dec 1	Dec 0
13	54285	Sus/Rel	Sus 3	Sus 2	Sus 1	Sus 0	Rel 3	Rel 2	Rel 1	Rel 0
			Voice 3							
14	54286	Low FQ	FQ 7	FQ 6	FQ 5	FQ 4	FQ 3	FQ 2	FQ 1	FQ 0
15	54287	High FQ	FQ 15	FQ 14	FQ 13	FQ 12	FQ 11	FQ 10	FQ 9	FQ 8
16	54288	Low PW	PW 7	PW 6	PW 5	PW 4	PW 3	PW 2	PW 1	PW 0
17	54289	High PW	/////	/////	/////	/////	PW 11	PW 10	PW 9	PW 8
18	54290	Waveform	Noise				Test	R mod	Sync	Gate
19	54291	Atk/Dec	Atk 3	Atk 2	Atk 1	Atk 0	Dec 3	Dec 2	Dec 1	Dec 0
20	54292	Sus/Rel	Sus 3	Sus 2	Sus 1	Sus 0	Rel 3	Rel 2	Rel 1	Rel 0
			Volume/Filters/Misc.							
21	54293	Low FC	/////	/////	/////	/////	/////	FC 2	FC 1	FC 0
22	54294	High FC	FC 10	FC 9	FC 8	FC 7	FC 6	FC 5	FC 4	FC 3
23	54295	Res/Filt	Res 3	Res 2	Res 1	Res 0	Filtex	Filt 3	Filt 2	Filt 1
24	54296	Mode/Vol	3 off	HP	BP	LP	Vol 3	Vol 2	Vol 1	Vol 0
25	54297	Pot 1	7	6	5	4	3	2	1	0
26	54298	Pot 2	7	6	5	4	3	2	1	0
27	54299	Osc 3	7	6	5	4	3	2	1	0
28	54300	Env 3	7	6	5	4	3	2	1	0

So, why Ø–255 inclusive? The reason is, you can encode combinations of these base two place values to obtain any number between Ø and 255 inclusive. Zero is the smallest (Ø + Ø + Ø + Ø + Ø + Ø + Ø + Ø = Ø), and 255 is the largest because 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255. If you count the Ø, that gives you 256 possible ways to program the eight bits. This gives you a lot of control for your sound settings, and it's not that hard to manage.

Opposite is a chart to aid you in this process. It shows the address number, and function of each control, and the value located on each bit. The abbreviations are as follows:

Low FQ	low frequency number	Low FC	low frequency cut-off
High FQ	high frequency number	High FC	high frequency cut-off
Low PW	low pulse width	Res/Filt	resonance/filter voice
High PW	high pulse width	Mode/Vol	filter mode/volume
△	triangle waveform	HP	high pass filter
⩘	sawtooth waveform	BP	band pass filter
⊓⊔⊓	pulse waveform	LP	low pass filter
Noise	white noise waveform	Pot	potentiometer
Atk/Dec	attack/decay	Osc 3	oscillator 3
Sus/Rel	sustain/release	Env 3	envelope generator 3

8 Programming Pitch Controls

As you may remember, one of the main functions of the oscillators is to produce a pitched tone. There are a number of ways of programming the pitch controls of the SID. Each oscillator or voice has two control registers, or bytes, that encode the pitch. In the *Commodore 64 User's Guide*, these are referred to as the *low frequency* and *high frequency* numbers. This does not mean that a high frequency is encoded on the high frequency number and a low frequency encoded on the low frequency number. It simply means that the number encoding the frequency may often be larger than 255, which is the largest number a byte can handle. Therefore, the entire number is encoded as a 16-bit number, or two bytes. The low frequency number is the first half and the high frequency number is the second half.

By far the easiest method of controlling the pitch is to use the 'Table of Note Values' located in the *User's Guide*, Appendix M, or, for your convenience, Section VI of this book. The table contains the high and low frequency numbers for 96 notes, or eight octaves, based on A =440, the standard concert A. A = 440 is listed as A4, which means that it is the A of the fourth octave. The table lists this note as high 28, low 49.

To encode this note for Voice 1, simply:

```
POKE 54273,28
POKE 54272,49
```

As you can see by looking at the SID Chip Control Chart, these POKE numbers refer to the address of the pitch control registers for Voice 1. If you want to encode this same note for Voice 2, you would:

```
POKE 54280,28
POKE 54279,49
```

Similarly, Voice 3 would be:

```
POKE 54287,28
POKE 54286,49
```

As you can see, the pitch numbers remain the same for each voice. Using this method, you can accurately program the pitches for a song. We will show you how in Chapter 19.

For now, however, you may want to practise using the Table of Note Values. Here is a program that will make it easy. It is designed around an INPUT statement.

```
  5  REM *** HI LOW ***
 1Ø  FOR M = 54272 TO 54296 : POKE M,Ø : NEXT
 2Ø  PRINT CHR$(147)
 3Ø  PRINT "ENTER NUMBER THEN HIT RETURN"
 4Ø  PRINT "ENTER Ø TO END"
 5Ø  INPUT "HIGH FQ #";A
 6Ø  IF A = Ø THEN END
 7Ø  INPUT "LOW FQ #";B
 8Ø  POKE 54296,15
 9Ø  POKE 54278,24Ø
1ØØ  POKE 54273,A : POKE 54272,B
11Ø  POKE 54276,17
12Ø  FOR T = 1 TO 1ØØØ : NEXT
13Ø  POKE 54296,Ø
14Ø  GOTO 1Ø
```

Select a pitch from the table, and enter the high and low values when requested. The program automatically POKEs these values into the high and low frequency control registers for Voice 1. You will then hear your note played by the computer. Try a few.

9 Advanced Pitch Control Techniques

There may be times when you want to program a pitch that does not appear in the Table of Note Values. To do this, you must use a formula. (The formula was used to compile the table in the first place.)

The first step is to convert the actual frequency of the pitch (FA) to a frequency number (FQ). The formula for this is:

FQ = FA / .06097

As an example, let's say we want to program A at 444 cycles per second instead of 440. We have to convert our actual frequency, 444, to a frequency number. Using the Commodore as a calculator, divide the actual frequency 444 by .06097.

(FA) 444 / .06097 = (FQ) 7282.26997

Rounded off, our frequency number is therefore 7282.

The next step is to convert the frequency number 7282 into a high frequency number (HF), and a low frequency number (LF).

To obtain the high frequency number, use the formula:

HF = FQ / 256

and truncate the number. In our example this gives:

7282 / 256 = 28.4648438

Therefore HF = 28.

To obtain the low frequency number, use the formula:

LF = FQ − (256 * HF)

and round off the number. Thus:

LF = 7282 − (256 * 28) = 114

To encode your note for Voice 1:

```
POKE 54273,28
POKE 54272,114
```

This is an especially handy tool if you are interested in programming music with different tuning systems for notes—as opposed to the standard convention of 'equal temperament'. It also gives you the ability to program quarter tones.

This is an advanced technique that will be particularly useful to certain people, but no one should be afraid to experiment. The real value of our computer lies in its capacity to create sounds not ordinarily heard.

Another useful technique utilizes the fact that an octave vibrates at twice the speed of the octave below it. With this knowledge, you can construct a program that will generate the notes of all eight octaves, simply by encoding the frequency numbers for the highest octave and dividing by two the appropriate number of times. This is especially useful because of the computer's binary orientation.

For an example of this technique, examine the program entitled 'Nick's Scaler' in Chapter 2.

10 Controlling Waveforms

Besides producing a pitched tone, the oscillators of the SID generate four waveforms. As discussed earlier, they are the triangle, sawtooth, pulse and white noise waveforms. If you look at the SID Chip Control Chart, you will see that each voice has its own waveform control register. The address for Voice 1 is 54276; for Voice 2, it is 54283; and for Voice 3, 5429Ø. Each of these three control registers or bytes can be thought of in this way:

Decimal	128	64	32	16	8	4	2	1
Function	White noise	Pulse	Saw-tooth	Tri-angle	Test	Ring mod	Sync	Gate
Bit	7	6	5	4	3	2	1	0

Each waveform works in coordination with the *gate*. To turn on a waveform, simply take the decimal number for the chosen waveform, and add it to the gate. Thus, a triangle waveform can be selected by adding 16 + 1, or 17. Similarly, the sawtooth is turned on by 33, pulse by 65, and white noise by 129. The gate is a very important element of this operation, because it acts as a trigger for the ADSR, or envelope generator. When the gate is opened, it signals the ADSR to begin its attack/decay and sustain cycles. When it is closed, it signals the ADSR to begin its release cycle. For this reason, it is very important to set the ADSR controls into your program ahead of the waveform.

Let's return to our original sound program.

```
 5 FOR M = 54272 TO 54296 : POKE M,Ø : NEXT
1Ø POKE 54296,15
2Ø POKE 54277,Ø : POKE 54278,128
3Ø POKE 54272,28 : POKE 54273,49
4Ø POKE 54276,17
5Ø FOR T = 1 TO 9ØØ : NEXT
6Ø POKE 54276,16
```

As you can see, we gated (turned on) the waveform in line 40, after we had programmed the information for the ADSR in line 20 and the pitch in line 30. Gating a waveform without this information is like opening your mouth and having no sound come out—you've got to tell your vocal chords to vibrate a message first. Line 50 sets the duration of the sound. Line 60 stops the sound (closes the gate) and thus begins the release of the envelope.

If you look at the SID Chip Control Chart for the waveform control register you will notice that it also contains bits for the ring modulation and synchronization functions. These functions work with the waveforms. We'll show you how they work in Chapter 27, which is devoted to ring modulation and synchronization.

The 'test' bit (bit 3) has a special function. When turned on, it resets the oscillator. You probably won't have to use this function if you follow one rule—*take care* when adding waveforms. It is possible to add two waveform settings together, however, if you add a waveform while the white noise waveform is on, the output of the white noise waveform may lock up. If this happens, the bit must be reset or it will remain silent.

The triangle waveform produces a mellow, round sound. When used with the proper envelope, it is useful for producing sounds similar to those of a flute, an organ, or a French horn.

The sawtooth waveform is much brighter and edgier, useful for brass and reeds.

The white noise waveform gives you a hissing sound, which can be refined to produce all kinds of sound effects such as explosions, hand claps, snare drums and rocket engines. The pulse waveform is the most varied sound, because you can alter its harmonic content by controlling the pulse length. This also makes it the most difficult to program.

PROGRAMMING THE PULSE WAVEFORM

As you may recall, the pulse waveform is a rectangular shape. You can define the width of the pulse by programming two controls. If you take a quick look at the SID Chip Control Chart, you will see that the low and high pulse addresses for Voice 1 are 54274 and 54275 respectively. For Voice 2, they are 54281 and 54282, and for Voice 3, 54288 and 54289. Together, these two controls form a 12-bit number.

	Decimal	128	64	32	16	8	4	2	1
54274	Low PW	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0
54275	High PW	X	X	X	X	PW11	PW10	PW9	PW8

High pulse numbers range from Ø–15, and low pulse numbers can be from Ø–255. The easiest way to understand these settings is through experimentation. For this reason, we have supplied you with the following program. It is similar to the program we used for exploring the high and low frequency numbers.

We'll get you started by telling you that a setting of Ø for the low pulse width number and 8 for the high pulse width number will create a square wave.

```
 5  REM *** HI LOW PULSE ***
1Ø  FOR M = 54272 TO 54296 : POKE M,Ø : NEXT
2Ø  PRINT CHR$(147)
3Ø  PRINT "ENTER NUMBER THEN HIT RETURN"
4Ø  PRINT "ENTER (-1) TO END"
5Ø  INPUT "HIGH PW # (Ø-15)";A
6Ø  IF A = -1 THEN END
7Ø  INPUT "LOW PW # (Ø-255)";B
8Ø  POKE 54274,B : POKE 54275,A
9Ø  POKE 54296,15
1ØØ  POKE 54273,28 : POKE 54272,49
11Ø  POKE 54278,24Ø
12Ø  POKE 54276,65
13Ø  FOR T = 1 TO 1ØØØ : NEXT
14Ø  POKE 54276,64
15Ø  POKE 54296,Ø
16Ø  GOTO 2Ø
```

If you would like to experiment with these waveforms you can use any of the programs we've provided. Just change the waveform control number in any of the program lines containing POKE statements with the addresses 54276, 54283 or 5429Ø. Remember to POKE a high and low pulse width command if you want to use the pulse waveform.

11 Programming the ADSR Controls

The envelope generator of the SID is one of the most powerful and unique controls. Each voice has its own envelope generator which is located on two bytes. The first byte contains the attack/decay function while the second contains the sustain/release. If you examine the SID Chip Control Chart, you see that the address for the Voice 1 attack/decay register is 54277. The address for sustain/release is 54278. For Voice 2, the register numbers are 54284 and 54285. For Voice 3, the register numbers are 54291 and 54292.

Decimal	128	64	32	16	8	4	2	1
Function	Atk 3	Atk 2	Atk 1	Atk 0	Dec 3	Dec 2	Dec 1	Dec 0
Function	Sus 3	Sus 2	Sus 1	Sus 0	Rel 3	Rel 2	Rel 1	Rel 0
Bit	7	6	5	4	3	2	1	0

Registers 54277 and 54278

ATTACK SETTINGS

There are 16 possible attack settings built around the numbers Ø, 16, 32, 64, 128. The other settings are derived by adding any combination of these numbers. The maximum attack setting is 24Ø, while the minimum setting is Ø. As you may remember, the attack controls how rapidly a note rises from zero to its peak volume. A setting of Ø results in the volume rising instantaneously to its peak volume. This is like an organ. As soon as you press the key, you get the maximum sound.

A setting of 24Ø gives you a very slow and gradual rise to peak volume, while a setting of 128 provides an intermediate rise.

The attack cycle begins when the waveform gate is opened.

DECAY SETTINGS

Like the attack function, the decay function also has 16 possible settings. These settings are based around the numbers Ø, 1, 2, 4, 8. By adding any or all of these numbers, you can set the controls from Ø to 15. The decay function has its parameters defined by the attack and sustain settings.

The decay controls how rapidly a note drops from peak volume to a point defined by the sustain setting. A setting of Ø results in almost no decay. A setting of 15 produces a long and gradual decay.

COMBINING ATTACK AND DECAY

When you have selected your attack and decay settings, you must then add them together and encode the sum as one number. For example, let's say you have selected a fast attack of 16 and a medium decay of 5. You then add them together (16 + 5 = 21), and insert them into the program with:

```
POKE 54277,21
```

SUSTAIN SETTINGS

The sustain function uses the same decimal control numbers as the attack: Ø, 16, 32, 64, 128. The sustain cycle of the ADSR follows the decay cycle and defines the lower end of the decay. It then sustains that volume level until the waveform gate is closed.

A setting of Ø provides no sustain. This is like a piano without the sustain pedal—the pitch simply fades away. A setting of 128 provides a sizeable sustain—producing a volume function similar to applying the sustain pedal right after a note is struck. A setting of 24Ø produces the maximum sustain.

RELEASE SETTINGS

The release cycle is triggered by the closing of the waveform gate. At this point, the volume of the note drops from the sustain level to zero at a rate defined by the release setting. These settings are built around Ø, 1, 2, 4, 8, again giving you a choice of Ø to 15. A setting of Ø produces no release effect. This is also like an organ. As soon as you remove your finger from the key, the note ends.

A setting of 15 gives the maximum release, producing the effect of the sound trailing away.

COMBINING SUSTAIN AND RELEASE

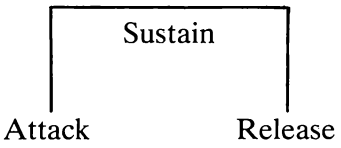
To encode the sustain/release settings, follow the procedure used for the attack/decay. Take the two settings and add them together. Thus a sustain of 128 and a release of 9 are encoded as 137 and placed into the program as follows:

```
POKE 54278,137
```

SOME FINAL HINTS

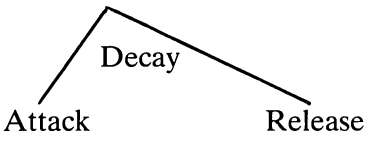
The real trick to using the ADSR is learning how the attack/decay/sustain/release all work together. Try to envisage the envelope you want to create. Think about how the sound goes through its stages. Then try to create these stages using the settings. Often, drawing a diagram is helpful when plotting out your sound. Here is one suitable for an organ-like envelope.

Attack Ø, Decay Ø, Sustain 128, Release Ø.



Here is one for a harp.

Attack 4, Decay 15, Sustain Ø, Release 15.



Combined with the proper waveform, pitch range and sometimes with the help of filters, the ADSR plays a big role in the total sound produced.

One final reminder about the ADSR. Always place the settings towards the beginning of your program before the waveform control settings. This way, when you open the gate and trigger the ADSR, the computer already has the envelope in its memory.

If you would like to practise some ADSR settings, you can use the 'Four Envelopes' program in Chapter 4. This allows you to program four groups of settings each time you run the program. Line 2Ø of that program controls the attack/decay and line 3Ø controls the sustain/release.

```
2Ø  AD(Ø) = 144: AD(1) = 8: AD(2) = 65: AD(3) = Ø
3Ø  SR(Ø) = 243: SR(1) = 9: SR(2) = Ø: SR(3) = 128
```

By changing the values to the right of the equals signs, you can enter your new settings into an array. Try it out.

12 Programming Volume Controls

Of all the SID's sound controls, the volume is perhaps the easiest to control. It is located on register 54296 which it shares with the filter mode.

Decimal	128	64	32	16	8	4	2	1
Function	Osc 3 off	HP	BP	LP	Vol 3	Vol 2	Vol 1	Vol 0
Bit	7	6	5	4	3	2	1	0

Register 54296

The volume controls are located on the first four bits. The settings range from Ø (softest) to 15 (loudest). Set the volume at the beginning of your program.

```
POKE 54296,15
```

In most cases, the maximum setting of 15 is advised. This one volume setting controls all three voices during the course of your program. It is a good idea to end your program by setting the volume to Ø. This prevents the annoying buzz that often remains after a sound program has run.

13 Programming Note Duration

There are a number of ways in which you can control the duration of a note. The simplest method is to use a time loop, as we did in our sample sound program at the beginning of this section. The time loop is created by a FOR/NEXT statement placed between the commands that open and close the waveform gate.

```
POKE 54276,17
FOR DR = 1 TO 5Ø0: NEXT
POKE 54276,16
```

The time loop instructs the computer to open the gate, wait for a time and then close the gate. You can lengthen the note by increasing the number within the loop. Thus, FOR DR = 1 TO 1ØØØ is twice as long as our original setting. Similarly, you can shorten the duration by decreasing the number within the loop. FOR DR = 1 TO 25Ø produces a note with half the duration.

By replacing the second number of the loop with a variable in a READ statement, and placing the value in a DATA statement, you can control the length of each individual note. We used this technique in several of our sample programs. We'll teach you how to use it in Chapter 19.

Another easy method of controlling the duration of a note is to utilize the computer's 'jiffy clock' which reads an interval timer. The jiffy clock is initialized to zero when the computer is turned on. It then increments by 1 every 1/6Øth of a second. TI is a reserved system variable that contains the current value of the jiffy clock. Start your program by setting the variable, TM in this case, equal to the timer (TI).

```
8Ø  TM = TI
```

You can place the following statements, which control the duration of your note, between the lines controlling the waveform gate.

```
70  POKE 54276,17
80  IF TI < TM + 10 THEN 80
90  TM = TI
100 POKE 54276,16
```

Line 80 instructs the computer to wait until the computer time (TI) is 10 time intervals from where it began before moving on to the next line.

Line 90 resets TM to the current clock time.

To increase the duration, increase the number added to TM. To decrease the duration, decrease the number. Thus TM + 20 is twice as long as the original setting, and TM + 5 is half as long. This method was employed in the program 'Nick's Scaler' which appears in Chapter 2. We will show you other ways to use the jiffy clock in later music programs.

Section III:

Programming Music Using READ/DATA

Introduction

When programming music, you always want to bear in mind that the computer is your instrument as well as your composing sheet. The SID chip is the actual sound generating instrument. The BASIC programming language and the assembly language statements provide you with a number of different methods for encoding your music, and thus, act as your method of notation. The approach you employ for encoding your music is largely dependent upon the structure of the selected piece. Is it short or long? Does it use single or multiple voices? Are there many repetitions of small phrases, or relatively few repetitions? Is the tempo slow or fast? Do you want to control the duration of each note for each voice, or is unison the aim? These are all questions which you should consider when tailoring a specific music program. The answers to these questions will often determine your programming format.

One function which your computer/instrument performs very well is organizing a collection of information, known as DATA, and presenting it in whatever order you have programmed. This makes it particularly well suited to developing music, provided that you can convert musical ideas such as pitch and duration into DATA, and then structure this DATA into a program that will present it in the proper order at the proper time.

The BASIC language offers many tools with which to accomplish this task. Among them are the READ/DATA statement; the use of arrays; time loops; counters; conditional statements such as IF/THEN; flags (known as delimiters); and directional statements, such as GOTO. These are some of the most powerful tools of programming, and you may already be familiar with them. If you're not, we'll show you how they work. If you've worked with them before, we'll show you how they are used in music programming in coordination with the SID's sound controls.

14 READ/DATA Format

One of the most useful methods of sound programming employs the READ/DATA format. This is an especially desirable format for use with short sound effects and simple songs. It is an easy procedure to modify our 'Sound Format Program' (Section II) into a READ/DATA format.

Here is the original program.

```
 1  REM *** SOUND FORMAT ***
 5  FOR M = 54272 TO 54296 : POKE M,Ø : NEXT
1Ø  POKE 54296,15
2Ø  POKE 54277,Ø : POKE 54278,128
3Ø  POKE 54273,28 : POKE 54272,49
4Ø  POKE 54276,17
5Ø  FOR T = 1 TO 9ØØ : NEXT
6Ø  POKE 54276,16
```

Here is a modified version using the READ/DATA format. This program plays a simple C scale.

```
 1  REM *** READ/DATA FORMAT I ***
 5  FOR M = 54272 TO 54296 : POKE M,Ø : NEXT
1Ø  POKE 54296,15
2Ø  POKE 54277,Ø : POKE 54278,128
25  READ H,L : IF H<Ø THEN POKE 54296,Ø : END
3Ø  POKE 54273,H : POKE 54272,L
4Ø  POKE 54276,17
5Ø  FOR T = 1 TO 9ØØ : NEXT
6Ø  POKE 54276,16
7Ø  GOTO 25
1ØØ  DATA 16,195,18,2Ø9,21,31,22,96,25,3Ø
11Ø  DATA 28,49,31,165,33,135,-1,-1
```

Here is a line-by-line description of how this program works.

Line 5 clears the sound chip.
Line 1Ø sets the master volume to 15.
Line 2Ø sets the ADSR controls for Voice 1.
Line 25 instructs the computer to READ the first two pieces of DATA in the DATA bank, and to call them H and L respectively. It then tells the computer to check the value of H and see IF it is less than (<) Ø. IF it is less than (<) Ø, THEN the computer is instructed to END the program. IF it is not less than (<) Ø, THEN the computer should proceed to the next line.
Line 3Ø POKEs the high frequency number register for Voice 1 with the current value of H, and POKEs the low frequency register for Voice 1 with the current value of L.
Line 4Ø turns on the triangle waveform and opens the gate for Voice 1, triggering the ADSR cycle.
Line 5Ø plays the note for the duration encoded in the time loop.
Line 6Ø closes the gate, triggering the release cycle and end of note.
Line 7Ø instructs the computer to GO back to line 25 and begin, which will take the next two pieces of DATA in the DATA bank as H and L.
Line 1ØØ DATA for the high and low frequency numbers for the notes C, D, E, F and G of the fourth octave.
Line 11Ø DATA for notes A, B and C, as well as the flag (delimiter) −1.

UNDERSTANDING THE LOOP

Let's look more carefully at the loop that is set up between lines 25–7Ø. As you can see, we have placed the READ statement in line 25. This begins a loop which cycles between line 25 and line 7Ø and then instructs the computer to return to line 25. When placed in a program loop such as the one used here, the first two pieces of information are READ by the computer in the order in which they appear in the DATA bank (which begins in line 1ØØ). The computer then assigns these two pieces of information the variable names chosen in the READ statement. In this case, H and L are assigned the values 16 and 195 respectively. The computer then checks to see if the value of H is less than Ø. If it is, the computer ends the program. In this case, the value is 16, so the computer continues to the next line.

In line 3Ø, the values of H and L are POKEd into the appropriate registers, thus defining the pitch C-4 as found in the Table of Note Values. The computer then passes on to lines 4Ø–6Ø, which instruct it to play the note for the proper duration, and then to close the gate. Having satisfied these commands, the computer passes on to line 7Ø, where it receives the instruction to go back to line 25 and READ the next two pieces of information. This time through the loop, the computer plucks

the next two pieces of information from line 1Ø0, thus giving H and L the respective values 18 and 2Ø9—which are the frequency numbers for D. This information is processed through the loop in the same way. Eventually, all the notes for a C scale are READ and played. Finally, on the ninth pass of the loop, the computer reads the values −1, −1 for H and L. H is thus less than (<) Ø, so the program ends.

FLAGS

By placing a −1 in the DATA bank, we have set up a 'flag' or marker, also called a 'delimiter'. You will find these flags very useful for manipulating the DATA in your programs. They are also helpful as signals for the computer to break out of a loop. If we had not included −1, −1 in our DATA as well as the conditional statement:

```
IF H < Ø THEN END
```

in line 25, the computer would have continued to READ DATA after the eighth note, and would have encountered nothing in the bank. It would then print:

```
OUT OF DATA ERROR IN 25
```

We'll show you other uses for delimiters in later programs.

15 Setting Up the DATA Bank

There are a few simple rules that should be followed when setting up a DATA bank. The DATA bank is usually placed at the end of your program.

The first step is to select a line number. In our case—the C scale program that appears in the previous chapter—the DATA bank begins with line 1Ø0. This is followed by the statement DATA. At this point, you can enter the values you wish to be READ. Each value must be followed by a comma. In this way, the computer can differentiate each piece of information.

A comma should *not* be placed after the final value entered on a DATA line. If you place a comma after the final value of a line, it throws the computer out of sync and produces and ERROR statement. To see and hear the effects of this mistake, go to line 1Ø0 of your program, and place a comma after the last value, which is 3Ø. Then, press RETURN to enter the change. Now, RUN the program. Aside from the strange tones following the fifth note, you will also notice:

? ILLEGAL QUANTITY ERROR IN 3Ø

printed at the bottom of your screen.

The Commodore 64 can handle up to two lines of values for any single DATA entry. It will automatically continue entering values on the second line if you overload the first. If you overload the second line, it will continue on to a third. However, if you do spill on to a third line, even if only with the cursor, all DATA items on the first two lines will be completely ignored! Also, any items on the third line are treated as a new line, with the first item on the line being treated as the line number! This can be *very* damaging to your program. For this reason, you should take care, when entering your DATA, not to enter more than two lines for any single line number. If you need more space, type the next line number, followed by DATA, and continue entering your values.

Another rule that should be followed when compiling the DATA bank is always to include the same number of DATA values as asked for in the READ statement. For instance, our statement asks the computer to READ H and L. We must therefore include two minus ones (−1, −1) at the end of our DATA. Otherwise, the computer will be looking for a value that is nonexistent, and will print an OUT OF DATA ERROR statement.

All the DATA values in the bank are treated as a continuous list, and are READ left to right, from the lowest line number to the highest.

16 Duration Control

We are not restricted in the number and kind of values to be READ in any one statement. We can, for example, expand our present READ/DATA format, from one that just READs a high and a low frequency number, to one that will READ a high and a low number as well as a duration number. In this manner, we may vary the duration of each pitch.

```
 1  REM *** READ/DATA FORMAT II ***
 5  FOR M = 54272 TO 54296 : POKE M,Ø : NEXT
1Ø  POKE 54296,15
2Ø  POKE 54277,Ø : POKE 54278,128
25  READ H,L,DR : IF H<Ø THEN POKE 54296,Ø : END
3Ø  POKE 54273,H : POKE 54272,L
4Ø  POKE 54276,17
5Ø  FOR T = 1 TO DR : NEXT
6Ø  POKE 54276,16
7Ø  GOTO 25
1ØØ DATA 16,195,4ØØ,18,2Ø9,4ØØ,21,31,6ØØ,22,96,2ØØ,25,3Ø,2ØØ
11Ø DATA 28,49,2ØØ,31,165,2ØØ,33,135,1ØØØ,-1,--1,-1
```

As you can see, this program is very similar to Format I. We have amended line 25 by adding an extra variable (DR) for duration.

```
25  READ H,L,DR: IF H < Ø THEN END
```

We have also amended line 5Ø and replaced the 9ØØ with the variable DR, so that it now reads:

```
5Ø  FOR T = 1 TO DR: NEXT
```

You will also notice that we have added another −1 to the flag. This is to compensate for the third value added to the READ statement.

In this way, we have replaced a constant duration value with a variable. This variable is encoded as every third value in the DATA statement. Thus, each time through the loop, the computer places the

new value of DR into the time loop. Since time is a proportional concept, we can create quarter notes, eighth notes, sixteenth notes, etc. by adjusting the length of the duration number. If 200 is used for a sixteenth note, then 400 is an eighth note, 800 a quarter note, and so on. That's because a sixteenth note is half the duration of an eighth note while an eighth note is half the duration of a quarter note.

A similar program using the jiffy clock to control duration can be created using this format.

```
  1  REM *** READ/DATA FORMAT II *** JIFFY CLOCK ***
  5  FOR M = 54272 TO 54296 : POKE M,0 : NEXT
  7  TM = TI
 10  POKE 54296,15
 20  POKE 54277,0 : POKE 54278,128
 25  READ H,L,DR : IF H<0 THEN POKE 54296,0 : END
 30  POKE 54273,H : POKE 54272,L
 40  POKE 54276,17
 50  IF TI<TM+DR THEN 50
 55  TM = TI
 60  POKE 54276,16
 70  GOTO 25
100  DATA 16,195,25,18,209,25,21,31,45,22,96,15,25,30,15
110  DATA 28,49,15,31,165,15,33,135,75,-1,-1,-1
```

This program uses the jiffy clock instead of the time loop. The initial value is set in line 7, and in line 50 the conditional statement:

```
50  IF TI < TM + DR THEN 50
```

replaces our time loop. Now, the computer adds the value READ as DR to the current value of TM, and remains on line 50 until the interval timer, TI, catches up with the new value of TM. When TI is no longer less than TM, the computer moves on to:

```
55  TM = TI
```

which once again sets TM equal to the current value of the timer.

By placing our timer between line 40, which turns on the note, and line 60, which turns off the note, we can effectively control the duration of the pitch.

Once again, we use proportional values to create notes of different lengths. If 15 is a sixteenth note, then 30 is an eighth note and 60 is a quarter note.

17 The RESTORE Statement

As discussed earlier, when READing DATA, the computer always READs from left to right, from the lowest line number to the highest. It does this by maintaining an internal pointer to keep track of what to read next. Through the use of the RESTORE statement, we can reset this internal pointer back to the first piece of DATA.

This capacity is particularly useful when we want to repeat a section of music. To see how the RESTORE statement works, let's add it to our jiffy clock program, by amending line 25. The effect of this addition will be the creation of a continuous loop. To break out of the program, press the RUN/STOP key and the RESTORE key. It should be noted that the RESTORE *key* has nothing to do with the RESTORE *statement*. The key simply resets the computer to its original state. You can then re-LIST the program if you want to examine it.

```
  1 REM *** READ/DATA FORMAT I! *** JIFFY CLOCK *** RESTORE
  5 FOR M = 54272 TO 54296 : POKE M,Ø : NEXT
  7 TM = TI
 1Ø POKE 54296,15
 2Ø POKE 54277,Ø : POKE 54278,128
 25 READ H,L,DR : IF H<Ø THEN RESTORE : GOTO 25
 3Ø POKE 54273,H : POKE 54272,L
 4Ø POKE 54276,17
 5Ø IF TI<TM+DR THEN 5Ø
 55 TM = TI
 6Ø POKE 54276,16
 7Ø GOTO 25
1ØØ DATA 16,195,3Ø,18,2Ø9,3Ø,21,31,45,22,96,15,25,3Ø,15
11Ø DATA 28,49,15,31,165,15,33,135,75,-1,-1,-1
```

As you can see, we altered the conditional statement in line 25 to:

```
25 READ H,L,DR: IF H < Ø THEN RESTORE: GOTO 25
```

The computer ignores this statement until H is less than 0. When H is less than 0, it resets the internal pointer to the first element on the DATA list, goes back to the beginning of line 25, and READs the DATA once again. This creates our loop.

This is useful for certain sound effects, but for playing music, we need more control of the DATA than just a repetitious loop. We can gain some of this control by using the RESTORE statement in coordination with *counters*. We discuss the use of counters in the next chapter.

18 Using Counters

Counters are used to keep track of how many times the computer has performed a particular task. When used in coordination with conditional statements, they become useful tools for the manipulation of DATA. If we add a counter to our last program, we can then use it with a conditional statement to play the scale as many times as we wish, and then to break out of the loop. The following example will play the scale three times and then END.

```
 1  REM *** READ/DATA FORMAT II *** JIFFY CLOCK *** RESTORE
    *** COUNTERS
 5  FOR M = 54272 TO 54296 : POKE M,Ø : NEXT
 7  TM = TI
 1Ø  POKE 54296,15
 2Ø  POKE 54277,Ø : POKE 54278,128
 25  READ H,L,DR : IF H<Ø THEN 5ØØ
 3Ø  POKE 54273,H : POKE 54272,L
 4Ø  POKE 54276,17
 5Ø  IF TI<TM+DR THEN 5Ø
 55  TM = TI
 6Ø  POKE 54276,16
 7Ø  GOTO 25
 1ØØ  DATA 16,195,3Ø,18,2Ø9,3Ø,21,31,45,22,96,15,25,3Ø,15
 11Ø  DATA 28,49,15,31,165,15,33,135,75,-1,-1,-1
 5ØØ  C = C+1
 51Ø  IF C = 3 THEN POKE 54276,Ø : END
 52Ø  RESTORE : GOTO 25
```

In this program, we have again changed line 25:

```
25  READ H,L,DR: IF H < Ø THEN 5ØØ
```

Now when the computer encounters the flag −1, it goes to line 5ØØ, where it increments the counter.

```
5ØØ  C = C + 1
```

The first time through, C is automatically equal to Ø. Thus, the computer converts the counter as follows: Ø = Ø + 1, which means that the old C, which was equal to Ø, is now the new C, which is equal to 1.

After incrementing the counter in line 5ØØ, the computer moves on to line 51Ø, where it encounters a conditional statement:

```
51Ø  IF C = 3 THEN POKE 54296,Ø: END
```

At this point, C = 1; so the computer ignores everything to the right of THEN, and passes on to line 52Ø.

```
52Ø  RESTORE: GOTO 25
```

Now the computer resets the DATA pointer to the first element on the DATA list, and returns to line 25 to start the process again. This time through the loop, when it is directed to line 5ØØ, it again increments the counter by 1. Thus, C = C + 1 is converted to 1 = 1 + 1. The C which was 1 is now 2. Once again the computer passes the conditional statement in line 51Ø and RESTOREs the DATA for another pass through the loop.

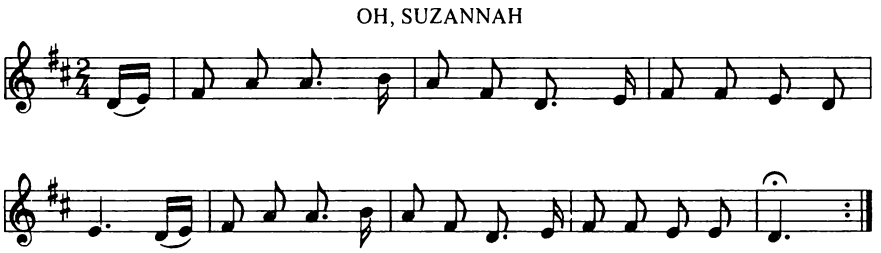
Finally, on the third pass, the counter is bumped up to 3. The conditional statement IF C = 3 is fulfilled, and the computer executes the commands to turn off the volume and end the program.

You will find counters very useful in a variety of situations. We'll show you some of these uses in later programs. For now, however, let's move on and program a piece of music.

19 Programming a Song for One Voice

Let's start our music programming with a song familiar to all. We've chosen 'Oh, Suzannah', played on a synthesized banjo.

Here is the song in standard musical notation.



Here is our program which plays this music.

```
  5  REM *** OH SUZANNAH BY STEPHEN FOSTER ***
 1Ø  FOR M = 54272 TO 54296 : POKE M,Ø : NEXT
 2Ø  POKE 54296,15
 3Ø  POKE 54277,7 : POKE 54278,5
 4Ø  READ H,L,DR
 5Ø  IF H < Ø THEN 9ØØ
 6Ø  POKE 54273,H : POKE 54272,L
 7Ø  POKE 54276,33
 8Ø  FOR T = 1 TO DR : NEXT
 9Ø  POKE54276,32
 95  FOR B = 1 TO 34 : NEXT
1ØØ  GOTO 4Ø
2ØØ  DATA 18,2Ø9,75,21,31,75,23,181,15Ø,28,49,15Ø,28,49,225,31,165,75
21Ø  DATA 28,49,15Ø,23,181,15Ø,18,2Ø9,225,21,31,75,23,181,15Ø,23,181,15Ø
22Ø  DATA 21,31,15Ø,18,2Ø9,15Ø,21,31,45Ø,18,2Ø9,75,21,31,75,23,181,15Ø
```

```
230  DATA 28,49,150,28,49,225,31,165,75,28,49,150,23,181,150,18,209,225
240  DATA 21,31,75,23,181,150,23,181,150,21,31,150,21,31,150,18,209,900
250  DATA -1,-1,-1
900  I = I+1 : IF I = 2 THEN POKE 54296,0 : END
910  RESTORE : GOTO 40
```

Here is a line-by-line description of how this program works.

Line 5	lists the title and composer.
Line 10	clears the sound chip.
Line 20	sets the volume to maximum.
Line 30	sets ADSR A = 0, D = 7, S = 0, R = 5; banjo effect for Voice 1.
Line 40	reads the high and low frequency numbers and the duration for each note.
Line 50	is a conditional statement used to control DATA.
Line 60	enters the current value for high and low frequency into the control registers for Voice 1.
Line 70	turns on the sawtooth waveform and opens the gate for Voice 1.
Line 80	sets up a time loop for the duration of each note.
Line 90	turns off the note and closes the gate.
Line 95	establishes a second time loop which puts a small silence between each note to enhance the banjo sound and articulation.
Line 100	creates a loop by returning to line 40 to READ the next note.
Lines 200–240	DATA bank containing information for each note: first number represents high frequency, second is the low frequency, and third the duration.
Line 250	is a flag to signal end of DATA.
Line 900	is the loop counter: it turns off the volume and ends the program on the second pass.
Line 910	RESTOREs the DATA and sends the computer back to line 40 to restart the song.

For this program, we used the READ/DATA format combined with a time loop for the duration. The timing values are as follows:

75 = sixteenth note
150 = eighth note
225 = dotted eighth note
300 = quarter note
450 = dotted quarter note

For the last note, we used a duration value of 900, simply because it sounded better. It's all right to break out of a strict numerical interpretation of the time value. This makes your music sound more 'human'

and less 'machine-like'. As you become more familiar with programming music, it's a good idea to experiment with all your variables. This gives your music a more personal touch.

Another little twist we added to this program was a second time loop in line 95. Not only does this have the effect of slowing the entire program down, it also puts a space between the notes. Line 95 adds only a thirty-second note to each duration which is not that noticeable. What it really does is separate the notes to make the articulation crisper, enhancing the overall banjo effect. When you have to describe a sound completely on the computer, many things that just happen when someone plays an instrument have to be worked into the computer description of each instrument. The numbers 1 to 34 were chosen by experimentation: 5Ø was too slow, 25 was too fast.

Finally, we used our flags, counter and RESTORE statement to manipulate the DATA. In this case, all we wanted was to repeat the entire DATA list and END. It is possible, however, to use these tools for more complex manipulation of DATA. To see how, take a look at the next chapter.

20 DATA Manipulation

It is probably dawning on you just about now that the main drawback to music programming is the quantity of numbers involved in setting up a DATA bank. There are, however, many tricks you can use to cut down on the number of notes you have to encode.

In the READ/DATA format, there is some flexibility for accomplishing this with the proper use of flags, counters, and the RESTORE statement. The chief disadvantage of this particular format is that you can only return to the beginning of the DATA list when you use the RESTORE statement. Still, it offers much in the way of a direct approach to programming while you are just starting out. Later on, we'll introduce you to the *array* format, which is more flexible in its ability to manipulate DATA, but a little harder to understand. The concepts that you will learn here, however, will also be applicable to arrays.

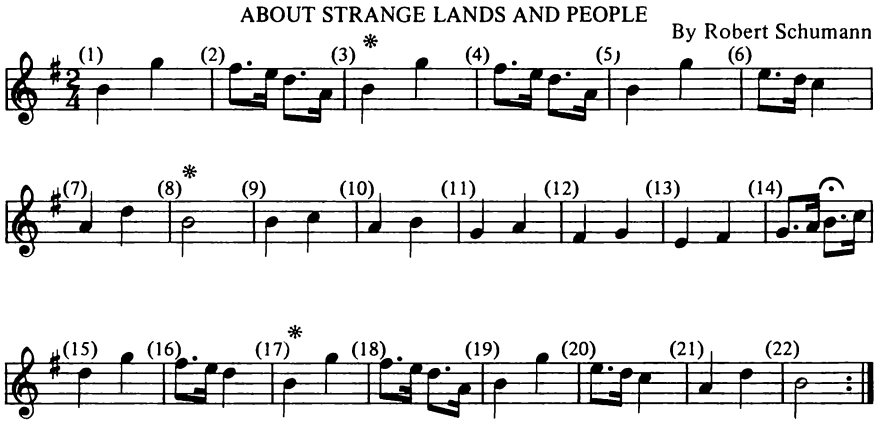
The following program illustrates some of the possible techniques you can use to manipulate DATA. It is a beautiful melody by Robert Schumann from Opus 15, 'Scenes From Childhood'. Type it in on your Commodore, sit back, and listen. This program runs for about two minutes.

```
 5 REM *** SCHUMANN ***
10 FOR M = 54272 TO 54296 : POKE M,0 : NEXT
20 POKE 54296,15
30 POKE 54277,47 : POKE 54278,136
35 J = J+1 : I = 0 : IF J = 3 THEN POKE 54296,0 : END
40 READ H,L,DR : IF H<0 THEN 1000
50 POKE 54273,H : POKE 54272,L
60 POKE 54276,17
70 FOR T = 1 TO DR : NEXT
80 POKE 54276,16
90 GOTO 40
```

```
100  DATA 31,165,1200,50,60,1200,47,107,950,42,62,250,37,162,950,28,49,
     250
110  DATA -1,-1,-1
120  DATA 31,165,1200,50,60,1200,42,60,950,37,162,250,33,135,1200
130  DATA 28,49,1200,37,162,1200,31,165,2400,-1,-1,-1
140  DATA 31,165,1200,33,135,1200,28,49,1200,31,165,1200,25,30,1200,28,
     49,1200
150  DATA 23,181,1200,25,30,1200,21,31,1200,23,181,1200,25,30,900,28,49,
     300
160  DATA 31,165,1000,33,135,300,37,162,1200,50,60,1200,47,107,900,42,
     62,300
170  DATA 37,162,1200,-1,-1,-1
1000 I = I+1 : IF I=1 THEN RESTORE : GOTO 40
1005 IF I = 2 THEN 40
1010 IF I = 3 THEN 40
1015 IF I = 4 THEN RESTORE : GOTO 40
1020 IF I = 5 THEN 40
1030 IF I = 6 THEN RESTORE : GOTO 35
```

As you can see, this program utilizes two counters: one in line 35 and one in line 1000. It also uses three flags: the first in line 110, the second in line 130, and the third in line 170. In addition, the RESTORE statement is used three times.

To understand what all this is doing, let's take a look at the musical notation for this piece.



The numbers represent the bar number. The *s represent the placement of a flag.

This song contains 54 notes. If we encode the high and low frequency numbers and the duration of each note, it requires 162 items of information. In addition, if we want to play it twice, that means 324 entries! Obviously, this is undesirable. Looking carefully at the piece we can see

that some of the music is used more than once. Using our counters and flags in coordination with the RESTORE statement, we can get this number of items down to about 100. If that still sounds like a lot of numbers—you are right. Turning the flow of electrons into real music is not the simplest of matters—lots of numbers are required, no matter what method you use. The reward comes from all the listening pleasure you and your friends will derive from the work. That doesn't mean, though, that you shouldn't try your hardest to cut down the number of entries you *have* to make.

We set up two counters in this program. The first in line 35.

```
35  J = J + 1: I = 0: IF J = 3 THEN POKE 54296,0: END
```

This counter keeps track of how many times the piece is played. The first time through J = 1; the second time, J = 2. After completing the second pass, we go back to the counter a third time, and thus trigger the end of the song.

We set up a second counter in line 1000. This counter (I) works with the flags as set up in line 40:

```
40  READ H,L,DR: IF H < 0 THEN 1000
```

This counter has been initialized to 0 in line 35. It is used to keep track of our flags.

The next step is to eliminate as many repetitious phrases as possible, always keeping in mind that once we use the RESTORE statement, we will be back at the beginning of our DATA list. Studying the musical notation, we see that bars 3 and 4 are identical to bars 1 and 2. Also, bars 17–22 are identical to bars 3–8. With this knowledge, we can eliminate 20 notes, or 60 DATA entries. DATA lines 100–110 contain the information for bars 1 and 2, followed by a flag −1, −1, −1. When the computer sees the flag, it goes to line 1000. The counter (I) is now 1, so the computer follows the command for 1. It RESTOREs the DATA, and goes back to line 40, where it re-READs the list. In this manner, we have eliminated the necessity of encoding bars 3 and 4.

The next time the computer arrives at the flag in line 110, it is again directed to line 1000. This time, the counter (I) is 2, so the computer follows the command for 2 in line 1005. As you see, the DATA is not RESTOREd this time through. This means that the computer will continue reading the DATA from its current position (line 120). In this way, we can place a flag in the DATA and use it several times, having it perform a different function each time.

The computer arrives at its next flag in line 130, which is the end of bar 8 in the musical notation. The counter is moved up to 3, and the computer is again directed back to line 40 and continues READing DATA for bar 9 (line 140).

In DATA line 170 the next flag is registered, bumping the counter up to 4. We are now at bar 17 of the music. Here is our chance to avoid encoding bars 17–22. We do this by RESTOREing the DATA and returning to the beginning of our list. Keep in mind that we wish to duplicate bars 3–8, and not bars 1 and 2.

This is accomplished when we arrive once again at our flag in line 11Ø. This time, the counter moves to 5, and the computer receives the instruction to keep READing. It does so until it re-encounters the flag in line 13Ø. At this point we are at the end of bar 22 in the music.

Our counter is now at 6. We have played the song through once, and it's time to tell the computer to play it again. We accomplish this by directing it back to line 35. At line 35, our counter (J), which keeps track of the number of times we play the song, is bumped up to 2. Our counter (I) is reset to Ø, and we begin the whole process again.

When we arrive at line 1Ø3Ø for the second time, the computer is directed to line 35. The counter (J) is moved up to 3. The volume is turned off, and the program ENDs.

If you are having any problems understanding these new concepts, take heart, and above all, don't give up. It often takes a while to absorb a new idea completely. If you need a break, take one, and let the ideas roll around for a while. If you like, skip ahead to some of the sound effects, and have a few laughs.

21 Short-cuts and Abbreviations

When you are typing in your programs, you will probably want to save as much time as possible. For this reason, we are going to introduce a few time-saving conventions.

The first convention is the use of abbreviations. The Commodore 64 allows you to abbreviate many keywords in the BASIC programming language. A complete list may be found in Appendix D of the *Commodore User's Guide*. An abbreviation is usually accomplished by typing the first letter of the command followed by a shifted second letter. Thus, POKE is abbreviated with a P and a shifted O. The shifted letter will appear on the screen as the graphics symbol located on the right of the front face of the letter key. A shifted O would therefore appear as an inverted L.

The advantage of the abbreviation is that it saves time and space. When the program is listed, the abbreviated command will appear in its full spelling. The abbreviation for DATA for example, D and shifted A, will appear as the full word DATA when the program is listed. You must take care to leave space at the end of your second DATA line if you use the abbreviated form. The space must be enough to accommodate the fully written commands. Otherwise, you might lose a few elements in your list.

Other abbreviations you may find useful are RE (shifted) S for RESTORE; and L (shifted) I for LIST.

The second time-saving convention you may find helpful is the use of initialization. Start your program by assigning the number 54272 to a variable such as M. The number 54272 is the number of the first register in the sound chip. Thus, if M = 54272, then M + 1 would equal 54273. Similarly, M + 24 would equal 54296. In this way, you can cut down on the length of your POKE addresses. If you look at your SID Chip Control Chart, you will see that we have provided you with a complete list of abbreviated numbers in the first column. Use the chart as follows: begin your program with an initialization:

```
M = 54272
```

Then, if you want to POKE a specific address, look for the number on the chart.

For example, POKE M + 11, 17 POKEs the waveform control register for Voice 2 with the triangle waveform. Similarly POKE M + 24, 15 sets the volume to its maximum setting.

We will often be using this convention in the following chapters. After a while, you won't have to look at the chart, as you will remember the code.

One last time-saving tip: make a master program for each music format and keep it on file. Then, when you want to program a piece, just load in the master and fill in whatever variables for waveform, ADSR, etc. you wish. Then, it's a simple matter of entering your new DATA and whatever commands are necessary for its manipulation.

Why not try out these techniques on the next program? It's for a simple bass line.

```
10  REM *** JAZZ BASS LINE ***
20  FOR Z = 54272 TO 54296 : POKE Z,0 : NEXT
30  M = 54272
40  POKE M+24,15
50  POKE M+5,9 : POKE M+6,15
55  J = J+1 : IF J = 3 THEN FOR T = 1 TO 1000 : NEXT : POKE M+24,0
    : END
60  READ H,L,DR : IF H<0 THEN 500
70  POKE M+1,H : POKE M,L
80  POKE M+4,33
90  FOR Q = 1 TO DR : NEXT
110 POKE M+4,32
115 FOR T = 1 TO 300 : NEXT
120 GOTO60
200 DATA 3,244,25,4,180,25,5,71,150,4,180,200
210 DATA 3,244,25,4,180,25,5,71,150,4,180,100,4,180,100,-1,-1,-1
220 DATA 5,71,25,6,71,25,7,12,150,6,71,200
230 DATA 5,71,25,6,71,25,7,12,150,6,71,100,6,71,100,-1,-1,-1
500 C = C+1 : IF C = 1 THEN RESTORE : GOTO 60
510 IF C = 2 THEN 60
520 IF C = 3 THEN RESTORE : GOTO 60
530 IF C = 4 THEN RESTORE : C = 0 : GOTO 55
```

22 Multiple Voice Programming

Now that you are familiar with some of the music programming techniques for one voice, it's a simple matter to get all three working for you. Here is a program that demonstrates the 64's ability to produce music with three part harmony. For this program, we use the READ/DATA format combined with the jiffy clock. You will notice that we have provided a separate DATA line for each of the song's thirty-six notes. Although this is certainly not necessary, it does provide for a certain amount of clarity. Another advantage of this method is that, should you enter a wrong value in your DATA, it is a simple matter to locate the error.

```
  1 PRINT "■" : REM *** CLEAR SCREEN ***
  5 PRINT "SWING LOW SWEET CHARIOT (TRAD)"
 10 PRINT "MUSIC ARRANGED BY JIM VOGEL"
100 M = 54272
110 POKE M+24,15
120 POKE M+5,9 : POKE M+6,0
130 POKE M+12,26 : POKE M+13,36
140 POKE M+19,24 : POKE M+20,202
150 T = TI
160 POKE M+4,32 : POKE M+11,32 : POKE M+18,16
170 READ X : IF X<0 THEN 690
180 READ H1,L1,H2,L2,H3,L3
190 POKE M+1,H1 : POKE M,L1 : POKE M+4,33
200 POKE M+8,H2 : POKE M+7,L2 : POKE M+11,33
210 POKE M+15,H3 : POKE M+14,L3 : POKE M+18,17
220 T = T+X
230 IF T>TI THEN GOTO 230
240 GOTO 160
```

```
300  DATA 30,28,49,33,135,5,152
310  DATA 60,22,96,14,24,0,0
320  DATA 30,28,49,33,135,5,152
330  DATA 45,22,96,29,223,9,104
340  DATA 15,22,96,0,0,0,0
350  DATA 15,18,209,29,223,0,0
360  DATA 45,16,195,10,143,6,71
370  DATA 15,22,96,16,195,7,12
380  DATA 15,22,96,11,48,0,0
390  DATA 15,22,96,14,24,9,104
400  DATA 15,22,96,11,48,0,0
410  DATA 15,28,49,21,31,8,97
420  DATA 15,33,135,0,0,0,0
430  DATA 30,33,135,21,31,7,12
440  DATA 30,33,135,22,96,8,97
450  DATA 30,0,0,0,0,18,209
460  DATA 30,0,0,21,31,16,195
490  DATA 30,0,0,0,0,14,239
500  DATA 15,37,162,22,96,14,24
510  DATA 15,33,135,0,0,14,24
520  DATA 60,28,49,0,0,16,195
530  DATA 30,33,135,22,96,9,247
540  DATA 45,22,96,14,239,9,104
550  DATA 15,22,96,14,239,9,104
560  DATA 15,18,209,0,0,0,0
570  DATA 45,16,195,14,239,10,143
580  DATA 15,22,96,16,195,11,48
590  DATA 15,22,96,0,0,0,0
600  DATA 15,22,96,18,209,9,104
610  DATA 15,22,96,0,0,0,0
620  DATA 15,28,49,18,209,7,119
630  DATA 15,28,49,0,0,0,0
640  DATA 30,25,30,14,239,8,97
650  DATA 30,22,96,14,24,11,48
660  DATA 30,0,0,0,0,8,97
670  DATA 30,0,0,0,0,5,152
680  DATA -1
690  C = C+1 : IF C = 2 THEN 1000
700  RESTORE
710  FOR J = 1 TO 200: NEXT
720  GOTO 150
1000 FOR Z = M TO 54296 : POKE Z,0 : NEXT Z
```

Here is a line-by-line description of how this program works.

Lines 1–1Ø	set up the screen display.
Line 1ØØ	initializes M to 54272.
Line 11Ø	turns on the volume.
Lines 12Ø–14Ø	sets the ADSR of voices 1–3 respectively.
Line 15Ø	sets T equal to the value of the jiffy clock.
Line 16Ø	sets the waveforms for voices 1–3 respectively.
Line 17Ø	READs the duration value X and checks if it is less than Ø. If it is, the computer is instructed to go to line 69Ø.
Line 18Ø	READs the high and low frequency numbers for voices 1–3 respectively.
Lines 19Ø–21Ø	POKE the high and low frequency numbers into the appropriate registers for voices 1–3. They also turn on the waveform by opening the gate.
Line 22Ø	adds the duration value X to the variable T.
Line 23Ø	holds the note on until the jiffy clock catches up with the new value of T.
Line 24Ø	creates the loop by going back for the next note.
Lines 3ØØ–68Ø	contain the DATA. The first piece of information is the duration value. The next six are the high and low frequency numbers for voices 1–3.
Line 69Ø	sets up the counter for the number of times the piece will be played.
Line 7ØØ	restores the DATA.
Line 71Ø	creates a silence before going on to the next line by using a loop.
Line 72Ø	creates a loop to play the song again.
Line 1ØØØ	turns off the chip.

With the ability to create harmony through the use of multiple voices, your music will take on a new richness and colour that was not present with just one voice.

You will find the method used in this program useful for many types of musical statements. Its one major drawback, however, is that it does not provide for independent control of the duration for each voice. You must, therefore, construct your programs with this in mind. All these voices must play for the same duration simultaneously. In other words, Voice 1 cannot play a quarter note at the same point in the program that Voices 2 and 3 are producing eighth notes. You can, however, create the illusion of truly independent voices by turning different voices on and off at different times during the program. This is done quite simply, by entering zeros for the high and low frequency numbers of the voices you wish to remain silent. The effect will be that of independent movement.

We have provided you with a more sophisticated program (in Section V) that will enable you to achieve true independent control of multiple voices. If you are curious, skip ahead and take a look.

Here is another program which is similar to the one you just entered.

```
  1 PRINT "■" : REM *** CLEAR SCREEN ***
  5 PRINT "> > JOSHUA FOUGHT THE BATTLE OF JERICHO < <"
 10 PRINT "> > > > MUSIC ARRANGED BY JIM VOGEL < < < <"
100 M = 54272
110 POKE M+24,15
120 POKE M+5,9 : POKE M+6,0
130 POKE M+12,26 : POKE M+13,36
140 POKE M+19,16 : POKE M+20,240
150 T = TI
160 POKE M+4,32 : POKE M+11,32 : POKE M+18,16
170 READ X : IF X<0 THEN 900
180 READ H1,L1,H2,L2,H3,L3
190 IF H1 THEN POKE M+1,H1 : POKE M,L1 : POKE M+4,33
200 IF H2 THEN POKE M+8,H2 : POKE M+7,L2 : POKE M+11,33
210 IF H3 THEN POKE M+15,H3 : POKE M+14,L3 : POKE M+18,17
220 T = T+X
230 IF T>TI THEN GOTO 230
240 GOTO 160
300 DATA 10,18,209,0,0,0,0
310 DATA 10,17,195,0,0,8,225
320 DATA 10,18,209,0,0,9,104
330 DATA 10,21,31,0,0,10,143
340 DATA 10,22,96,0,0,11,48
350 DATA 10,22,96,0,0,11,48
360 DATA 20,25,30,0,0,12,143
370 DATA 10,28,49,18,209,11,48
380 DATA 20,22,96,0,0,0,0
390 DATA 40,28,49,18,209,11,48
400 DATA 20,0,0,0,0,7,12
410 DATA 10,25,30,17,195,10,143
420 DATA 20,25,30,17,195,10,143
430 DATA 40,25,30,17,195,10,143
440 DATA 20,0,0,0,0,7,112
450 DATA 10,28,49,18,209,11,48
460 DATA 20,28,49,18,209,11,48
490 DATA 35,28,49,18,209,11,48
500 DATA 5,0,0,0,0,7,12
510 DATA 5,0,0,0,0,7,233
520 DATA 5,0,0,0,0,8,225
530 DATA 10,18,209,0,0,9,104
540 DATA 10,17,195,0,0,8,225
550 DATA 10,18,209,0,0,9,104
560 DATA 10,21,31,0,0,10,143
```

```
570  DATA 10,22,96,0,0,11,48
580  DATA 10,22,96,0,0,11,48
590  DATA 20,25,30,0,0,12,143
600  DATA 10,28,49,18,209,11,48
610  DATA 20,28,49,18,209,11,48
620  DATA 30,28,49,18,209,11,48
630  DATA 10,22,96,0,0,11,48
640  DATA 10,25,30,0,0,10,143
650  DATA 20,28,49,22,96,9,104
660  DATA 20,25,30,21,31,8,97
670  DATA 10,22,96,18,209,7,119
680  DATA 10,22,96,0,0,0,0
690  DATA 20,21,31,17,195,7,12
700  DATA 30,18,209,14,24,9,104
710  DATA 20,0,0,0,0,7,12
720  DATA 30,0,0,0,0,4,180
730  DATA -1
900  C = C+1 : IF C = 2 THEN 1000
910  RESTORE
920  FOR J = 1 TO 200 : NEXT
930  GOTO 150
1000 FOR Z = M TO 54296 : POKE Z,0 : NEXT Z
```

The major difference between this and the previous program lies in lines 190–210.

```
190  IF H1 THEN POKE M + 1,H1: POKE M,L1:
     POKE M + 4,33
200  IF H2 THEN POKE M + 8,H2: POKE M + 7,L2:
     POKE M + 11,33
210  IF H3 THEN POKE M + 15,H3: POKE M + 14,L3:
     POKE M + 18,17
```

As you can see, we have now made these lines into a conditional command. If the computer reads a zero value in H1, it immediately goes on to line 200. This is because the computer treats the zero in the statement as *no value*. If H1 does have a value, however, the computer will execute the remainder of the line before passing on to line 200.

You may wish to save the first section of this program—up to line 230—and use it as a template for your own future programs.

Section IV:

Special Sound Effects

Introduction

One of the truly entertaining aspects of the SID is its ability to create sound effects. This is one area of sound generation which can really call into play many of the 64's unique capabilities. You can, for instance, incorporate the 64's ability to generate random numbers into a sound effects program, as we've done in the program 'Insect Speech'. You can also utilize the 64's STEP function to create a program that will 'scoop' pitch, producing *glissando* and siren-like sounds. All sorts of interesting effects may be created through the use of the SID's ring modulation and synchronization bits. In addition, the sometimes unruly-sounding waveforms can be moulded into a more sophisticated sound through the use of the three filters and resonator bits of the SID. Finally, you can utilize these effects as subroutines, or incorporate them into the main body of video programs to add another dimension to your computer art or games.

Let's move ahead and learn how all this can be done.

23 Phaser and Siren Sound Effects

The STEP statement is a very useful BASIC command in sound effects programming. The STEP command is used in conjunction with a FOR/NEXT loop. Normally, a FOR/NEXT loop, such as:

```
FOR T = 1 TO 100: NEXT
```

will increment by one each time through the loop. With the addition of the STEP statement, we can increment the counter by whatever value we choose. For example:

```
FOR T = 5 TO 50 STEP 5: NEXT
```

will loop ten times—progressing from 5 to 50 by 'steps' of 5. The increment occurs each time the computer encounters the NEXT.

With this in mind, take a look at our first sound effect.

```
  5  REM *** GLISS EFFECT ***
 10  FOR M = 0 TO 24 : POKE 54272+M,0 : NEXT
 20  M = 54272
 30  POKE M+24,15
 40  POKE M+5,136 : POKE M+6,240
 50  FOR X = 255 TO 4 STEP -2
 60  POKE M+1,X
 70  POKE M+4,17
 80  FOR K = 1 TO 4 : NEXT K : NEXT X
200  C = C+1 : IF C = 3 THEN 500
210  GOTO 50
500  POKE M+24,0 : END
```

This program creates a scooping of pitch effect, also known as a *glissando*. In this program, the pitch scoops upwards.

Here is a line-by-line description.

Line 1Ø clears the chip.
Line 2Ø initializes M to equal 54272.
Line 3Ø turns on the volume.
Line 4Ø sets the ADSR.
Line 5Ø assigns the variable X to every other value between 4 and 255.
Line 6Ø takes the current value of X and POKEs it into the high frequency number register of Voice 1.
Line 7Ø turns on the waveform and opens the gate.
Line 8Ø sets up a time loop of 1 TO 4—then loops back for the next X.
Line 2ØØ establishes the counter—IF 3 THEN END the program.
Line 21Ø instructs the computer to go back to 5Ø and do it again.
Line 5ØØ turns off the volume. END.

You will notice that in creating this effect, we have changed a few rules. We did not, as has been our custom, utilize both the high and low frequency numbers. That is because we were not interested in creating a specific pitch, such as A-44Ø, which requires two numbers. You can use the high frequency register alone to produce pitched notes if you are not concerned with exact interval relationships. Using the low frequency register alone, however, is not advisable because it produces very little sound.

You will also notice that, while we turned on the waveform and opened the gate, we never closed it with a 16. This choice produces a constant *on* sound of the program. This is also why our sustain/release setting is 24Ø, maximum sustain and no release. Since the release cycle of the ADSR is triggered by the closing of the gate, no setting we might have used would ever have had a chance to function. So, even though there are many notes produced, the effect is one of continuous sound getting higher and higher. This is repeated twice.

The most important point to grasp, however, is our use of the STEP command. Each value of X that was obtained from the loop was POKEd into the high frequency register: the first time a value of 4, then 6, then 8, etc. This is the element of the program responsible for the gradual rise in pitch.

We can also invert the STEP command by using negative numbers. For example:

```
FOR T = 5Ø TO 5 STEP −5: NEXT
```

will loop ten times, creating the values 5Ø, 45, 4Ø, 35, etc. for the variable T. We use this technique in the following program.

```
  5  REM *** GLISS EFFECT ***
 10  FOR M = 0 TO 24 : POKE 54272+M,0 : NEXT
 20  M = 54272
 30  POKE M+24,15
 40  POKE M+5,136 : POKE M+6,240
 50  FOR X = 4 TO 255 STEP 2
 60  POKE M+1,X
 70  POKE M+4,17
 80  FOR K = 1 TO 4 : NEXT K : NEXT X
200  C = C+1 : IF C = 3 THEN 500
210  GOTO 50
500  POKE M+24,0 : END
```

This program is identical to the one for our first effect except for:

```
50  FOR X = 255 TO 4 STEP -2
```

This line is responsible for the pitch descending instead of ascending as in the previous program.

You can combine these two programs to produce a 'siren-like' effect.

```
  5  REM *** SIREN EFFECT ***
 10  FOR M = 0 TO 24 : POKE 54272+M,0 : NEXT
 20  M = 54272
 30  POKE M+24,6
 40  POKE M+5,136 : POKE M+6,240
 50  FOR X = 4 TO 255 STEP 1
 60  POKE M+1,X
 70  POKE M+4,33
 80  FOR K = 1 TO 6 : NEXT K : NEXT X
 90  FOR Z = 255 TO 6 STEP -1
100  POKE M+1,Z
110  FOR J = 1 TO 4 : NEXT J : NEXT Z
200  C = C+1 : IF C = 3 THEN 500
210  GOTO 50
500  POKE M+24,0 : END
```

Still another effect that may be obtained through the use of the STEP statement is the 'Phaser' effect.

```
  5  REM *** PHASER EFFECT ***
 10  FOR M = 0 TO 24 : POKE 54272+M,0 : NEXT
 20  M = 54272
 30  POKE M+24,15
```

```
40  POKE M+5,136 : POKE M+6,255
50  POKE M+4,17
60  FOR K = 1 TO 15
70  FOR Z = 250 TO 0 STEP -20
80  POKE M+1,Z
90  NEXT Z : NEXT K
100 POKE M+4,16
500 POKE M+24,0 : END
```

This program produces the sound of a Sci-Fi space weapon. The sputtering effect is caused by the −20 STEP in line 70 *and* the opening and closing of the waveform gate.

24 The White Noise Effect

The easiest effect to produce on the 64 is the white noise effect. Simply POKE 129 into the waveform register. You can use the white noise waveform for explosions, bombs, gunfire and all sorts of havoc.

BLAST OFF

Here is just one example of its many uses—it dramatizes well the power of the white noise. It's called 'Blast Off'. Once again, we utilize Nick's Scaler Program; only this time we make two simple changes. In line 4Ø, we change the triangle waveform of 17 to a white noise waveform of 129, and in line 8Ø, we cut down the timer from TM + 1Ø to TM + 2. The result is quite startling. Our eight-octave chromatic scale turns into a rocket blasting off and finally trailing away into space! (Note: when used in a program, % means integer.)

```
  1 REM *** BLAST OFF BY J.C. VOGEL & NICK
  5 FOR Z = 54272 TO 54296 : POKE Z,Ø : NEXT
 1Ø FOR I = Ø TO 7
 2Ø POKE 54296,15
 3Ø POKE 54277,Ø : POKE 54278,24Ø
 4Ø POKE 54276,129
 5Ø READ A : IF A = -1 THEN GOTO 9ØØ
 6Ø A = A/2 ↑ I
 7Ø L% = A/256 : H% = A-(256*L%)
 8Ø IF TI<TM+2 THEN 8Ø
 9Ø TM = TI
1ØØ POKE 54273,L% : POKE 54272,H%
11Ø GOTO 4Ø
2ØØ DATA 64814,61176,57743,54502,51443,48556,45830,43258,40830,38539,
     36376,34334
```

```
800 DATA -1
900 RESTORE
910 NEXT I : FOR T = 1 TO 3000 : NEXT : POKE 54296,0
920 END
```

LOCOMOTIVE

Here is a white noise effect that creates the sound of a locomotive.

```
 50 REM *** LOCOMOTIVE ***
100 M = 54272
110 FOR Z = 54272 TO 54296 : POKE Z,0 : NEXT
120 FOR J = 1 TO 40
130 POKE M+24,15
140 POKE M+5,8
150 POKE M+1,20
160 POKE M+4,129
170 FOR Q = 1 TO 50 : NEXT Q
180 POKE M+4,128
190 NEXT J
200 POKE M+24,0
```

For this effect we used a nested loop. Line 120 creates a loop that makes the entire effect play 40 times while line 170 controls the duration of each chugging sound. This chugging sound is created by using a low attack number (line 140) with no sustain, combined with the white noise waveform.

White noise effects can vary greatly when filtered through a high-pass filter. We'll show you how to set these filters in Chapter 28.

25 Sound and Colour

One of the most common sounds wafting out of the halls of video arcades is something like this:

```
  5 REM *** VIDEO GAME EFFECT ***
 10 FOR M = 0 TO 24 : POKE 54272+M,0 : NEXT
 20 M = 54272
 30 POKE M+24,15
 40 POKE M+5,136 : POKE M+6,240
 50 FOR X = 4 TO 255 STEP 25
 60 POKE M+1,X
 70 POKE M+4,33
 80 FOR K = 1 TO 10 : NEXT K : NEXT X
 90 FOR Z = 255 TO 6 STEP -25
100 POKE M+1,Z
110 FOR J = 1 TO 2 : NEXT J : NEXT Z
200 C = C+1 : IF C = 3 THEN 500
210 GOTO 50
500 POKE M+24,0 : END
```

Similar in form to our 'Siren' program, this one creates a completely different effect. You can get a lot of mileage from a simple STEP command by varying the length of your time loop.

It is important to remember, though, that using time loops ties up the computer on the lines executing the loop. When incorporated into video programs, this may not always be desirable. (You may have noticed that, on many video games, the action freezes when the sound comes on. Now you know why.) When the time loop causes such a problem, we suggest you use the jiffy clock. Using the clock allows you to go off and check conditional statements and then return to check the clock.

Here is one way that you might insert this sound program into a game. Our intention here is merely to show you the technique of integrating sound with simple graphics commands.

```
 4  PRINT "■" : REM ** CLEAR SCREEN ***
 5  REM *** VIDEOGAME EFFECT ***
10  FOR M = 0 TO 24 : POKE 54272+M,0 : NEXT
20  M = 54272
30  POKE M+24,15
40  POKE M+5,136 : POKE M+6,240
50  FOR X = 5 TO 255 STEP 20
60  POKE M+1,X
70  POKE M+4,33
80  FOR K = 1 TO 15 : NEXT K : NEXT X
90  FOR Z = 255 TO 6 STEP -20
100 POKE M+1,Z
110 FOR J = 1 TO 12 : NEXT J : NEXT Z
200 C = C+1 : IF C = 5 THEN 500
205 POKE 53281,2 : FOR T = 1 TO 75 : NEXT
206 POKE 53281,11
210 GOTO 50
500 PRINT "*** 25 POINTS *** TRY AGAIN"
510 FOR T = 1 TO 100 : NEXT : POKE M+24,0 : END
```

Our program clears the screen, plays the effect, changes the background colour to red (line 205) and then to black (line 206), repeats the process four times, and then prints out a message before ending.

Here is a program that produces sixteen coloured bars, one for each of the 64's colours, across your display screen. After each bar, a tone is produced. Make sure to enter the end quote and semicolon in the second display line of line 60 or the graphics will not run correctly. The actual number of spaces included inside the quotes is not important as it just controls the width of the coloured bar—5 is plenty. (The reversed field R in line 60 is produced by pressing the control key and 9 simultaneously.)

```
10  POKE 53280,0 : POKE 53281,0
20  PRINT "■" : REM *** CLEAR SCREEN ***
30  FOR W = 1 TO 16
40  READ A
50  PRINT
60  PRINT CHR$(A); : PRINT "■R■     ";
65  REM *** SOUND ***
70  POKE 54296,15
```

```
 80  POKE 54277,0 : POKE 54278,128
 90  POKE 54273,28 : POKE 54272,49
100  POKE 54276,17
110  FOR T = 1 TO 100 : NEXT
120  POKE 54276,16
130  NEXT W
135  POKE 54296,0 : END
140  REM *** COLOUR DATA ***
150  DATA 5,28,30,31,129,144
160  DATA 149,150,151,152,153,154,155
170  DATA 156,158,159
```

As you can see from the REM statement, the sound commands begin on line 70 and continue to line 120. They should, by this time, be quite familiar.

You can diversify this program by changing your sound programming. This is accomplished by expanding your READ statement to include values to be POKEd into the high and low frequency number registers.

```
 10  POKE 53280,0 : POKE 53281,0
 20  PRINT "♥" : REM *** CLEAR SCREEN ***
 30  FOR W = 1 TO 16
 40  READ A,B,C
 50  PRINT
 60  PRINT CHR$(A); : PRINT "R     ";
 65  REM *** SOUND ***
 70  POKE 54296,15
 80  POKE 54277,0 : POKE 54278,128
 90  POKE 54273,B : POKE 54272,C
100  POKE 54276,17
110  FOR T = 1 TO 100 : NEXT
120  POKE 54276,16
130  NEXT
140  REM *** COLOUR & NOTE DATA ***
150  DATA 5,33,135,28,37,162,30,42,62,31,44,193,129,50,60,144,56,99
160  DATA 149,63,75,150,67,15
170  DATA 151,67,15,152,63,75,153,56,99,154,50,60,155,44,193,156,42,62
180  DATA 158,37,162,159,33,135
```

We changed line 40 from:

```
40  READ A
```

to:

```
40  READ A,B,C
```

We then changed line 9Ø to:

```
9Ø  POKE 54273,B: POKE 54272,C
```

Finally, we altered our DATA bank by placing the new values for B and C in between the old values for A.

Our program now displays the coloured bars, and after each bar appears a different note of the diatonic scale is sounded—ascending for the first eight colours, and descending for the second eight.

If you are perplexed by some of the graphics commands, we suggest you consult your *User's Guide* or the *Programmer's Reference Guide*. You may also want to read *Easy Programming for the Commodore 64* by Ian Stewart and Robin Jones, Shiva.

26 Using the Random Number Function

The Commodore 64 has a built-in feature for generating a series of random numbers. Actually, these are not truly random numbers, as any series may be reproduced by starting with the same seed number. For our purposes, however, the effect is random enough. RND is the BASIC command to call this function into play.

Here is a program that uses the random number function.

```
  5 REM *** INSECT SPEECH ***
 10 FOR M = 0 TO 24 : POKE 54272 + M,0 : NEXT
 20 M = 54272
 30 POKE M + 24,15
 40 POKE M + 5,16 : POKE M + 6,255
 50 FOR Z = 1 TO 5
 60 POKE M + 1,RND(Z)*50 + 54
 70 POKE M + 4,33
 80 FOR K = 1 TO 4 : NEXT K
 90 POKE M + 4,32
100 NEXT Z
200 C = C + 1 : IF C = 25 THEN 500
210 GOTO 50
500 POKE M + 24,0 : END
```

In this program, we introduce the random number function in lines 50 and 60.

```
50 FOR Z = 1 TO 5
60 POKE M + 1, RND(Z) * 50 + 54
```

M + 1 is POKEd with a random number between 54 and 94. When the computer comes across the RND, it produces a random floating point decimal number between Ø.Ø and 1.Ø. The value of Z is the seed. Actually, this is a 'dummy' number. The computer is more interested in knowing if the seed is positive, negative, or Ø. It then takes its decimal number and multiplies it by 5Ø and then adds 54 to that. Since the random number is always 1 or less, the largest number obtainable is 94. The smallest is 54. The computer then truncates the number, and presents the final value as an integer. This random integer is then POKEd into the high frequency register, producing our sound effect.

To vary the pitch of this effect, you can alter the numbers 5Ø and 54. A higher total will produce a higher pitch. For a greater variety of pitch, you can make the spread between the numbers greater.

Try a few changes to see what we mean. The only constraint which you must observe is that you cannot use a number larger than 255. If you do, you will get an "ILLEGAL QUANTITY" error message, since the high frequency register does not accept numbers larger than 255.

Here's another random number program that adds yet another twist. We call it 'Just Plain Nuts'—for obvious auditory reasons.

```
  5 REM *** JUST PLAIN NUTS ***
 1Ø FOR M = Ø TO 24 : POKE 54272+M,Ø : NEXT
 2Ø M = 54272
 3Ø POKE M+24,15
 4Ø POKE M+5,16 : POKE M+6,255
 5Ø FOR Z = 1 TO 15
 6Ø POKE M+1,RND(Z)*225+2Ø
 7Ø POKE M+4,17
 8Ø FOR K = 1 TO 4 : NEXT K
 9Ø POKE M+4,32
1ØØ NEXT Z
2ØØ C = C+1 : IF C = 25 THEN 5ØØ
21Ø GOTO 5Ø
5ØØ POKE M+24,Ø : END
```

We use essentially the same format for this program as for 'Insect Speech'—the twists come in lines 7Ø and 9Ø. What we've done in line 7Ø is to open the waveform gate and turn on the triangle waveform. In line 9Ø, we close the gate, but change the waveform to 32. This means that the release cycle, which was set at its maximum, now has a sawtooth waveform. Try this trick on some of your music programs if you get tired of the way they're sounding.

27 Ring Modulator and Synchronization Effects

With the SID's ring modulator you can produce astounding sound effects. When set, the ring modulator takes the output of two oscillators and produces tones that are the sum and difference of their output. To understand how to set the ring modulator bit, you have to take a look at the waveform registers for each voice.

Decimal	128	64	32	16	8	4	2	1
Function	White noise	Pulse	Saw-tooth	Tri-angle	Test	Ring mod	Sync	Gate
Bit	7	6	5	4	3	2	1	0

As you can see, the ring modulator is located on bit 2, with a decimal control number of 4. To utilize the function, you must add the ring modulator to the triangle waveform. Thus, the statement:

```
POKE 54276,21
```

instructs the computer to use the ring modulator function for Voice 1.

```
 16 = triangle waveform
  4 = ring modulator
+ 1 = opens gate
-----
 21
```

In addition, you must mix the output of Oscillator 3 with that of Oscillator 1. This can be accomplished as follows:

```
POKE 54273,28: POKE 54287,49
```

What we have done here is to POKE values into the high frequency number registers of both Voice 1 and Voice 3. No further settings are required for Voice 3. It is not necessary to turn on Voice 3 waveforms or ADSR.

To hear what the ring modulator can do, let's insert these commands into a program we used early on, called 'Nick's Scaler'.

```
 1  REM *** RING MOD EFFECT *** BY J.C. VOGEL & NICK
 5  FOR Z = 54272 TO 54296 : POKE Z,Ø : NEXT
1Ø  FOR I = Ø TO 7
2Ø  POKE 54296,15
3Ø  POKE 54277,Ø : POKE 54278,24Ø
4Ø  POKE 54276,21
5Ø  READ A : IF A = -1 THEN GOTO 9ØØ
6Ø  A = A/2 ↑ I
7Ø  L% = A/256 : H% = A-(256*L%)
8Ø  IF TI < TM+2 THEN 8Ø
9Ø  TM = TI
1ØØ  POKE 54273,L% : POKE 54287,H%
11Ø  GOTO 4Ø
2ØØ  DATA 64814,61176,57743,545Ø2,51443,48556,4583Ø,43258,4Ø83Ø,38539
21Ø  DATA 36376,34334
8ØØ  DATA -1
9ØØ  RESTORE
91Ø  NEXT I : FOR T = 1 TO 3ØØØ : NEXT : POKE 54296,Ø
92Ø  END
```

Originally, this program played a chromatic scale for the entire range of the 64. Now, with just a few changes, we get a remarkably different-sounding program. We changed line 4Ø to use the ring modulators. It now reads:

```
4Ø  POKE 54276,21
```

In addition, we changed line 1ØØ to mix the output of oscillators 3 and 1. It now reads:

```
1ØØ  POKE 54273,L%: POKE 54287,H%
```

(By the way, % stands for integer.)

To utilize the ring modulator function for Voice 2:

```
POKE 54283,21
```

and mix it with the output of Voice 1 as follows:

```
POKE 5428Ø,28: POKE 54273,49
```

Similarly, for Voice 3, mix the output with Voice 2:

```
POKE 5429Ø,21
POKE 54287,28: POKE 5428Ø,49
```

You will notice that no other settings are used for Voice 3 other than its high frequency number. This is all the 64 needs when it is using ring modulation. The ADSR or other setting for Voice 3 will have no effect. The same rules apply as for mixing the other voices.

SYNC FUNCTION

The sync function synchronizes the fundamental frequencies of the pitches. It is sometimes used when combining the two waveforms to produce a desired overall sound. For instance, a pulse wave and a sawtooth wave combined may produce an undesirable beating effect. The sync function can be used to eliminate this problem. This procedure, however, is only for very specific instances. A more general use of the sync is for sound effects. It can create very unusual types of electronic sounds.

You can produce the sync effect by using exactly the same procedures as we used for the ring modulator function. The only difference is that, instead of POKEing 21 into the waveform register, you POKE 19. This is because the sync function is located on bit 1, with a decimal control number of 2.

```
  16  =  triangle waveform
   2  =  sync function
+  1  =  gate
------
  19
```

To hear what the sync can do, let's alter another one of our programs. We've chosen 'Jazz Bass Line', which appears at the end of Chapter 21.

```
10  REM *** SYNC BASS LINE ***
20  FOR Z = 54272 TO 54296 : POKE Z,0 : NEXT
30  M = 54272
40  POKE M+24,15
50  POKE M+5,9 : POKE M+6,15
55  J = J+1 : IF J = 3 THEN FOR T = 1 TO 1000 : NEXT : POKE M+24,0
    : END
60  READ H,L,DR : IF H<0 THEN 500
70  POKE M+15,H : POKE M+1,L
80  POKE M+4,19
90  FOR Q = 1 TO DR : NEXT
110 POKE M+4,18
115 FOR T = 1 TO 30 : NEXT
120 GOTO 60
```

```
200  DATA 3,244,25,4,180,25,5,71,150,4,180,200
210  DATA 3,244,25,4,180,25,5,71,150,4,180,100,4,180,100,-1,-1,-1
220  DATA 5,71,25,6,71,25,7,12,150,6,71,200
230  DATA 5,71,25,6,71,25,7,12,150,6,71,100,6,71,100,-1,-1,-1
500  C = C+1 : IF C = 1 THEN RESTORE : GOTO 60
510  IF C = 2 THEN 60
520  IF C = 3 THEN RESTORE : GOTO 60
530  IF C = 4 THEN RESTORE : C = 0 : GOTO 55
```

The only changes occur in line 70, where we mix in the output of the Voice 3 high register; in line 80, where we activate the sync; and in line 110, where we turn it off.

It should be noted that the sync function works better when the lower pitch is in the mixed-in oscillator—as, in this case, Voice 3. That is why we have placed Voice 3 first. If you examine the DATA, you will see that the first variable is always lower. Other than this, the same rules that we used for mixing the voices for the ring modulator can be used for the sync function. To utilize the sync for Voice 2, POKE 54283,19 and mix it with the output of Voice 1 as follows:

```
POKE 54280,28: POKE 54273,49
```

Similarly for Voice 3, mix the output with Voice 2.

```
POKE 54290,21
POKE 54287,28: POKE 54280,49
```

28 Programming Filters

Using the SID's filters is not so much a sound 'effect' as a way of 'affecting' your sound. If you have not, as yet, read Chapter 5, we suggest you go back and do so now.

As stated earlier, the SID offers you three filters, called low-pass, high-pass and band-pass. A band-reject filter may be produced by adding the high-pass and low-pass filters together. You can also add resonance to each filter setting.

The filter controls are located on registers 54293–54296.

	Decimal	128	64	32	16	8	4	2	1
54293	Low FC						FC2	FC1	FC0
54294	High FC	FC10	FC9	FC8	FC7	FC6	FC5	FC4	FC3
54295	Res/Filt Voice	Res 3	Res 2	Res 1	Res 0	Filtex	Filt 3	Filt 2	Filt 1
54296	Filt/Vol	3 off	HP	BP	LP	Vol 3	Vol 2	Vol 1	Vol 0
	Bit	7	6	5	4	3	2	1	0

To utilize the filters, follow these steps:

1. Choose the voices you wish to filter. The control for voice selection is located at register 54295. For Voice 1, POKE 54295,1; for Voice 2, POKE 54295,2; for Voice 3, POKE 54295,4. For a combination, such as Voices 1 and 3, just add the decimal control numbers: POKE 54295,5. Enter this setting on to a program line near the beginning of the program, preferably before the ADSR and waveform settings. The *filtex* setting is for external audio input signals.
2. Choose your filter mode. The control is located on register 54296, the same as your volume control. The low-pass has a 16 control number, band-pass a 32, and high-pass a 64. Add the filter setting to the volume. Thus, a low-pass filter with a maximum volume setting

would be programmed with POKE 54296,31. Enter this into your program where you usually enter the volume.

```
  15  =  volume
+ 16  =  low-pass filter
------
  31  =  setting
```

3. Choose the cut-off frequency. The cut-off frequency is encoded as an 11-bit number, so two registers are necessary. They are 54293 and 54294. The low bit number on register 54293 must fall within Ø–7 inclusive. The high bit number on register 54294 can be within Ø–255 inclusive. The higher the number entered, the higher the frequency cut-off. Enter these settings into the program immediately after the filter voice setting in step 1.
4. If you want to add resonance to your filter settings, simply add the decimal control numbers to your voice selection on register 54295. Thus, a maximum resonance setting for a Voice 1 filter would be programmed as POKE 54295,241.

```
 128  =  resonance bit 3
  64  =  resonance bit 2
  32  =  resonance bit 1
  16  =  resonance bit 0
+  1  =  filter
------
 241  =  setting
```

Here is a sample program showing how these settings are entered into a program.

```
  1  REM *** FILTER SETTING ***
  5  FOR M = 54272 TO 54296 : POKE M,Ø : NEXT
 1Ø  POKE 54295,129 : POKE 54293,Ø : POKE 54294,1ØØ
 15  POKE 54296,31
 2Ø  POKE 54277,Ø : POKE 54278,128
 25  READ H,L : IF H<Ø THEN END
 3Ø  POKE 54273,H : POKE 54272,L
 4Ø  POKE 54276,33
 5Ø  FOR T = 1 TO 9ØØ : NEXT
 6Ø  POKE 54276,32
 7Ø  GOTO 25
1ØØ  DATA 16,195,18,2Ø9,21,31,22,96,25,3Ø
11Ø  DATA 28,49,31,165,33,135,-1,-1
```

All of our settings for the filters are encoded on lines 1Ø and 15.

Line 1Ø sets the filter for Voice 1 with moderate resonance: the low cut-off number is Ø; the high cut-off number is 1ØØ.
Line 15 sets the low-pass filter and the volume at 15.

To hear what this program would sound like without the filter, simply change the setting for register 54295 from 129 to Ø. This also demonstrates that, unless you specify which voice you wish to filter, all your other filter settings have no effect on the program.

Using the programmable filter is a fairly subtle procedure, as there are a number of settings which you must control. The cut-off frequency number is particularly significant in controlling the output of your sound. The approximate cut-off frequencies range from 3Ø Hz to 12 kHz. Hz stands for Hertz, which are the same as cycles per second. Again, the higher the number, the higher the fequency.

To understand this better, look at our sample filter program. We use a low-pass filter which passes the frequencies below the cut-off number, and attenuates those above. We use a setting of 54293,Ø: 54294,1ØØ. If we raise our cut-off number to 54293,Ø: 54294,24Ø, our cut-off sound is closer to the original unfiltered sound. That is because we are raising the cut-off point—thus allowing more of the original harmonics to pass unfiltered.

If we were using the high-pass filter, the output result would be the opposite, since the high-pass filter passes those harmonics above cut-off while attenuating those below.

The best way to 'get hold of' the filter controls is to put aside an hour and experiment with different settings and different filters.

29 Frequency Modulation

A multitude of sound effects can be produced through frequency modulation. Frequency modulation occurs when you mix the frequency output of two generators. This is accomplished on the 64 with the use of a new register number, 54299. This register is a 'read only' register, which means that you don't POKE it, you PEEK it. The PEEK function allows you to read what has been stored on a particular byte.

Register 54299 allows the 64 to read the upper eight bits of the Voice 3 oscillator. The numbers generated by that oscillator are regulated by the waveform selection. If the triangle waveform is chosen, the oscillator will output a series of numbers which increase from Ø to 255, and then decrease from 255 back down to Ø, at a rate defined by the frequency. If the sawtooth waveform is chosen, the output will be a series which increases from Ø to 255. If the pulse waveform is chosen, the output jumps between Ø and 255. The rate of change of the numbers is defined by the frequency of Voice 3. The white noise waveform will result in a random series of numbers.

If you mix the output of this register with another voice, you have frequency modulation. In order to accomplish this, you must turn off the audio output of Voice 3, by setting the 3 'off' bit located on register 54296. The waveform will now be used to modulate another voice for frequency modulation. POKE 54296,143 is the correct setting.

```
  128  =  3 off
+  15  =  volume
-----
  143
```

To see how this works, type in this program:

```
1ØØ  M = 54272
11Ø  FOR Z = 54272 TO 54299 : POKE Z,Ø : NEXT
12Ø  POKE M+14,2Ø : POKE M+18,16
```

```
130  POKE M+24,143
140  POKE M+6,240 : POKE M+4,33
150  N = 5728
160  FOR T = 1 TO 300
170  FQ = N+PEEK(M+27)*100
180  HF = INT(FQ/256) : LF = FQ-HF*256
190  POKE M,LF : POKE M+1,HF
200  NEXT
210  POKE M+24, 0
```

Here is a line-by-line description.

Line 100 initializes M to 54272.
Line 110 clears the chip.
Line 120 sets the low frequency register for Voice 3; sets the waveform for Voice 3.
Line 130 turns off Voice 3 audio output; sets volume to 15.
Line 140 sets ADSR for Voice 1; sets waveform for Voice 1.
Line 150 sets (N) equal to frequency number (5728) obtained from table. This is the carrier frequency.
Line 160 begins play loop.
Line 170 sets the frequency FQ equal to N + the value read from 54299 * 100. (54299 is the upper eight bits of Voice 3 oscillator.) This is the frequency modulation line.
Line 180 converts the frequency number (produced in line 170) to a high and a low frequency number. (See Chapter 9.)
Line 190 sets the high and low frequency number registers for Voice 1 with the numbers derived from line 180.
Line 200 instructs the computer to go back and do it again.
Line 210 turns off the volume.

If you are unclear about the formula used in line 180, there is an explanation in Chapter 9.

To experience the true range of this effect, try changing the variables. We suggest you start with line 120. Change the waveform selection to sawtooth, then pulse, and then white noise, to hear the various forms of modulation. Don't forget to add a pulse width number when you use the pulse waveform.

After you've experimented with the waveforms, try changing the low frequency number for Voice 3, also in line 120. Another number you can change is in line 150. To obtain a new number, consult the Table of Note Values, or make one up.

Finally, you can change a number in line 170. This is the number on the far right. It is currently set at 100. A change here has a dramatic effect. Try replacing the 100 with a 2 to hear what we mean. Frequency modulation is one of those effects that is particularly well-suited to a computer.

Another form of modulation can be accomplished using a similar method, but with the output of the top eight bits of the Voice 3 envelope generator. For this effect, use register 54300. You can mix the output of this register with the filter frequency output of another oscillator to create a 'phasing' effect. Your one rule here is that the gate for the third voice must be open. A waveform setting is not necessary.

Try out this program to hear the phasing effect.

```
100  M = 54272
110  FOR Z = 54272 TO 54300 : POKE Z,0 : NEXT
120  POKE M+24,143
130  POKE M+19,255
140  POKE M+6,240 : POKE M+4,33
145  POKE M+18,1
150  FOR T = 1 TO 300
160  N = 5728
170  FQ = N+PEEK(M+28)*10
180  HF = INT(FQ/256) : LF = FQ-HF*256
190  POKE M,LF : POKE M+1,HF
200  NEXT
210  POKE M+24,0
```

Here is the line-by-line description.

Line 100 initializes M equal to 54272.
Line 110 clears the chip.
Line 120 sets the volume; turns off the audio output for Voice 3.
Line 130 sets ADSR for Voice 3.
Line 140 sets ADSR for Voice 1.
Line 145 opens the gate for Voice 3.
Line 150 begins the loop.
Line 160 establishes the frequency number.
Line 170 adds the frequency number to 'the value read in register 54300' multiplied by 10.
Line 180 converts the FQ number to high and low FQ numbers.
Line 190 sets the high and low FQ registers for Voice 1.
Line 200 instructs the computer to go back and do it again.
Line 210 turns off the volume.

The general approach here is to take the output information from one voice and use it as an input to another. This technique can be used in different ways to produce a variety of effects.

The ability of the computer to perform complex mathematical functions at a rapid speed makes effects such as this one, and others that you

have heard, possible. Take some time to play with these ideas and see what you can do. The results are often surprising and almost always entertaining.

Now, let's move on and explore a new area of programming—using arrays. With the help of this new programming procedure, you will be able to play complex songs with independent control of multiple voices and note duration. You will also be able to create your own 'real time' organ using the 64's keyboard.

Section V:

Programming Music using the Array Format

Introduction

In Section III, we showed you how to program music using the READ/DATA format. That method proved quite useful for a variety of situations, but had a few drawbacks. The first drawback was its limitation in the manipulation of DATA. Our only option was to RESTORE the entire list and begin again. That's useful for some pieces of music, but for more complicated pieces it is not so useful. The second drawback was it required using ever-increasing lists of DATA, which, after a while became quite cumbersome. Finally, it did not allow for a practical method of controlling the multiple voices with independent duration control. For these reasons, we must look at another method to solve these problems. The method we'll discuss here involves a slightly more complicated technique—the use of arrays.

An array is essentially nothing more than a list. What makes it different from the list in a DATA statement, however, is that in creating an array, you set aside a portion of the computer's memory to store the list. Once the array is stored in memory, you can access it as many times as you like without having to RESTORE the information. Another powerful aspect of arrays is that each element in the list has its own name. That means that you can go directly to any element or portion of the list you desire.

You can even make up lists with many columns of numbers, as we will show you shortly. These are called dimensions of the array. This capacity is particularly useful in music programming because we often have occasion to call up groups of numbers such as the high and low frequency numbers.

We'll show you how to set up these different kinds of arrays in the following chapters.

30 Simple Arrays

Here is a program that we used in Section I to demonstrate our four waveforms. It accomplishes this through the use of a simple array.

```
  5  REM *** FLIGHT OF THE BUMBLE BEE ***
 10  FOR M = 54272 TO 54296 : POKE M,0 : NEXT
 20  WF(0) = 33 : WF(1) = 17 : WF(2) = 65 : WF(3) = 129
 30  POKE 54296,15
 40  POKE 54274,9 : POKE 54275,255
 50  POKE 54277,137 : POKE54278,128
 60  READ H,L
 70  IF H<0 THEN 900
 80  POKE 54273,H : POKE 54272,L
 90  POKE 54276,WF(I)
100  FOR T = 1 TO 50 : NEXT
110  GOTO 60
200  DATA 28,49,26,156,25,30,23,181,22,96
210  DATA 29,223,28,49,26,156,28,49
220  DATA 26,156,25,30,23,181,22,96
230  DATA 23,181,25,30,26,156,-1,-1
900  POKE 54276,0
910  RESTORE : I = I+1 : IF I = 4 THEN END
920  GOTO 60
```

We create our array in line 20. It consists of four parts, or elements, called subscripted variables. They are WF(0), WF(1), WF(2) and WF(3). As you can see, part of the name, WF, remains the same, but the part in parenthesis is variable. This part is known as the subscript. When the computer encounters a subscripted variable such as WF(0), it

recognizes it as an array, and immediately sets aside, or 'dimensions' a portion of memory to hold eleven elements. This means that, even though we only used four elements in our array, enough memory was automatically reserved potentially to hold WF(4)–WF(1Ø). If we had wished to use more than eleven elements, we would have had to use another array-devising procedure, which we will describe shortly. (That procedure utilizes the BASIC DIM statement.)

When creating an array with subscripted variables, the first character in the name must be a letter. The second optional character may be a numeral or a letter. Thus, WF(Ø), W3(Ø) and W(Ø) are all possible choices, while 3F(Ø), 33(Ø), 3(Ø) are syntactically illegal.

The other important task accomplished in line 2Ø was the assignment of information to be held in the array. Here we used a direct method by simply stating WF(Ø) = 33: WF(1) = 17 etc. Now the computer places into its memory the information that it stores 33 as the first element on its WF list, 17 as the second element etc.

On line 9Ø of the program, we tell the computer to utilize this information each time it goes through the loop established between lines 6Ø–11Ø.

```
9Ø  POKE 54276,WF(I)
```

In this case, I represents the current value of the counter set up in line 91Ø. Line 9Ø POKEs the value assigned to the Ith member of the array WF into address 54276. For a more graphic explanation, add the following line to your program:

```
9Ø  PRINT WF(I)
```

and try running it again. You should now see the waveform number of each note appear on the screen right after the note begins.

To demonstrate that the array is stored in memory, or to verify that a particular element in the array is correct, try this little trick. Without using a program line, simply type in PRINT WF(2) and hit RETURN. The computer should respond with a 65 printed on the screen.

USING THE READ/DATA STATEMENT TO LOAD THE ARRAY

Another method of loading an array is to READ in a DATA list. The difference between the DATA/READ into an array and a regular DATA list is that the DATA is now stored in memory, and each element of the list has been given a name. To demonstrate this quickly, try this short program.

```
1Ø  FOR I = Ø TO 3
2Ø  READ WF(I)
3Ø  NEXT
4Ø  PRINT WF(Ø),WF(1),WF(2),WF(3)
5Ø  DATA 17,33,65,129
```

After you run the program, which has no sound, type PRINT WF(2) and hit RETURN. The computer should respond with a 65, showing that the number 65 has been stored in an array under the name WF(2). We'll use this technique to load most of the arrays we use in music programming.

THE DIM STATEMENT

As mentioned earlier, when a subscripted variable is introduced into a program, a portion of memory is set aside to hold eleven elements. In this case, we have called them WF(Ø)–WF(1Ø). There may be occasions, however, when you wish to hold more than eleven elements in an array. When this happens, you must use the DIM statement.

The DIM statement instructs the computer to set aside or DIMension enough memory to hold a list. Thus, DIM WF(25) would DIMension an array to hold 26 elements: WF(Ø)–WF(25).

If we do not DIMension enough memory to hold our list, an error will occur during the execution of the program.

31 Using the Two-dimensional Array Format

We are now going to add the two-dimensional array format to our repertoire of programming techniques. The following program will demonstrate the main advantage of this method. The program plays a scale four times. The first time through it plays the whole scale. On the second pass it begins at the third note of the scale. On the third pass it begins at the fifth note, while on the final pass it plays the whole scale again. This demonstrates the increased flexibility in DATA manipulation that this form of programming offers.

Type in this program. As you do so, you might want to look at the line-by-line description that follows the program.

```
  5  REM *** 2 DIM ARRAY ***
 10  DIM V(20,2)
 15  M = 54272 : Z = 0 : C = 0
 20  FOR S = M TO M + 24 : POKE S,0 : NEXT
 30  POKE M + 24,15
 40  POKE M + 5,0 : POKE M + 6,240
 45  REM *** LOAD THE ARRAY ***
 50  READ H,L,DR
 60  V(Z,0) = H : V(Z,1) = L : V(Z,2) = DR : Z = Z + 1
 70  IF DR = -1 THEN GOTO 90
 80  GOTO 50
 85  REM *** PLAY THE MUSIC ***
 90  Z = 0
100  TM = TI : IF V(Z,0) = -1 THEN 1000
110  POKE M + 1,V(Z,0) : POKE M,V(Z,1)
120  POKE M + 4,17
130  IF TI < TM + V(Z,2) THEN 130
```

```
140  POKE M+4,16 : Z = Z+1
150  GOTO 100
200  DATA  16,195,8,18,209,8,21,31,8,22,96,8,25,30,8,28,29,8,31,165,8,33,
     135,24
210  DATA  -1,-1,-1
999  REM *** DATACONTROL ***
1000 C = C+1
1010 IF C = 1 THEN Z = 2
1012 IF C = 2 THEN Z = 4
1014 IF C = 3 THEN Z = 0
1016 IF C = 4 THEN POKE M+24,0 : END
1020 GOTO 100
```

Before we proceed with our line-by-line description here is an overview of what the program is doing. The program can be divided into three parts. In the first part, the DATA for the high and low frequency numbers and the duration are being READ into an array. The array 'V' has been DIMmed (20, 2), and would look like this:

Z	0	1	2
0	16	195	8
1	18	209	8
2	21	31	8
3	↓	↓	↓
4	↓	↓	↓
5	↓	↓	↓
6	↓	↓	↓
↓	↓	↓	↓
20	↓	↓	↓

This is known as a two-dimensional array. The statement:

DIM V(20, 2)

instructs the computer to create a two-dimensional array, twenty-one rows by three columns. The 2 in DIM V(20, 2) represents the three columns of information loaded with the DATA. The first column, 0, holds the HF number, while the second column, 1, holds the LF number. The third column, 2, holds the duration number. The 20 in the statement represents the note number or row number that has been loaded into the array using the counter Z in line 60.

The second part of the program (lines 90–150) uses the information stored in the array to play the music.

The third part of the program controls what information is used in the play section.

Now, here is the line-by-line explanation.

Line 10 establishes space for a two-dimensional array which can hold a maximum of twenty-one notes with three pieces of information for each note.

Line 15 initializes M to 54272; initializes the counters Z and C to 0.

Line 20 clears the chip.

Line 30 turns on the volume.

Line 40 sets the ADSR.

Line 50 reads the first three elements from the DATA list representing the high and low frequency and duration numbers, and assigns the variable names H, L and DR.

Line 60 loads the values read in line 50 into an array. The counter Z is then incremented by 1.

Line 70 checks to see if the current value of DR is −1. If it is, the program passes to line 90.

Line 80 creates a loop; instructs the computer to go back to line 50 to READ the next piece of information in the DATA.

Line 90 reinitializes the counter Z back to 0 for the play portion of the program.

Line 100 initializes TM to the jiffy clock TI; sets up a conditional statement: IF the value stored in the first column of the array reads −1, THEN the computer is instructed to go to 1000.

Line 110 POKEs the high and low frequency registers with the current values of V(Z,0) and V(Z,1) respectively.

Line 120 turns on the waveform and opens the gate.

Line 130 instructs the computer, IF the current value of TI is less than TM + the current value of V(Z,2) THEN remain on line 130; keep playing the note.

Line 140 closes the gate (ends the note); increments the counter Z.

Line 150 loops back to line 100 and plays the next note.

Line 200 DATA for FQ numbers and duration.

Line 210 DATA flag.

Line 1000 increments the counter C.

Line 1010 instructs the computer, IF C = 1, THEN set the note counter Z to equal 2 (which is the third note stored in the array).

Line 1012 instructs, IF C = 2, THEN set the note counter Z to equal 4 (which is the fifth note stored in the array).

Line 1014 instructs, IF C = 3, THEN set the note counter Z to equal 0 (which is the first note stored in the array).

Line 1016 instructs, IF C = 4, THEN turn off the volume and END the program.

Line 1020 instructs the computer to go to 100 and play the next note.

If you find the concept of using a *zero* as the first element in the array and a *one* as the second element too much of a 'double think', then initialize the Z to be equal to 1 in line 15 and line 9Ø. If you do this, then also increase the Zs in lines 1Ø1Ø, 1Ø12 and 1Ø14.

We hope that you now understand the array concept. If not, don't worry about it. You'll catch on, just give it time. You may also consult one of the many texts that are available on BASIC programming for more examples. You can still use the programs we've provided as a guide. Later on you'll be able to create your own programs to fit your needs.

32 Creating Two Characters with One Voice

Here is another program that utilizes the array format. It uses both a simple array and a two-dimensional array to create the impression of two different instruments, or characters, using one voice. Different phrases are played using different waveforms and ADSR settings. The effect is something like a call and response. You can use this technique to give the impression that many different instruments are involved in the playing of a piece, even though only three are available at any one time. Notice that, while we used an array to change the waveform, it was necessary to type in the new ADSR setting each time the voice changed. That is because the ADSR settings are not part of the loop that actually POKEs in the values that play the music.

Also notice that we used a number of flags in our DATA. This has made it possible to repeat much of the material without having to write out extra DATA entries, as in the READ/DATA format. While the main body of this program requires more lines than our previous method, it reduces the size of the DATA list. We'll shrink it even further in the next chapter.

For now, however, spend some time with the program and work it through. A line-by-line description follows the program. You can refer to it if you want clarification.

```
 5  REM *** MUSETTE *** II *** J.S.BACH
10  DIM V(200,2)
15  WF(0) = 16 : WF(1) = 64
20  M = 54272 : C = 0
25  FOR S = 54272 TO 54296 : POKE S,0 : NEXT
30  POKE M+2,25 : POKE M+3,8
35  POKE M+24,15
40  POKE M+6,255
45  REM *** LOAD THE ARRAY ***
50  READ H,L,DR
55  V(Z,0) = H : V(Z,1) = L : V(Z,2) = DR : Z = Z+1
60  IF DR = 0 THEN GOTO 80 : REM *** END LOADING ***
70  GOTO 50
```

```
 75  J = J+1 : IF J = 2 THEN POKE 54296,Ø : END
 8Ø  Z = Ø
 85  IF V(Z,Ø) = -1 THEN 1ØØØ
 9Ø  TM = TI
 95  REM *** PLAY THE MUSIC ***
1ØØ  POKE M+1,V(Z,Ø) : POKE M,V(Z,1)
11Ø  POKE M+4,WF(I)+1
12Ø  IF TI<TM+V(Z,2) THEN 12Ø
13Ø  POKE M+4,WF(I) : Z = Z+1
15Ø  GOTO 85
2ØØ  DATA 67,15,28,59,19Ø,7,56,99,7,5Ø,6Ø,7,44,193,7,-1,-1,-1
21Ø  DATA 28,49,7,29,223,7,33,135,14,29,223,14,28,49,14,-1,-1,-1
22Ø  DATA 25,3Ø,14,33,135,14,28,49,14,22,96,14,-1,-1,-1
23Ø  DATA 25,3Ø,14,33,135,18,22,96,36,-1,-1,-1,Ø,Ø,Ø
1ØØØ  C = C+1 : Z = Ø : I = Ø
1Ø1Ø  IF C = 2 THEN Z = 6 : I = 1 : POKE M+5,8 : POKE M+6,15
1Ø11  IF C = 3 THEN Z = 12 : I = 1
1Ø12  IF C = 4 THEN Z = Ø : I = Ø : POKE M+6,255
1Ø13  IF C = 5 THEN Z = Ø
1Ø15  IF C = 6 THEN Z = 6 : I = 1 : POKE M+5,8:POKE M+6,15
1Ø16  IF C = 7 THEN Z = 17 : I = 1
1Ø17  IF C = 8 THEN Z = Ø : C = Ø : POKE M+6,255 : GOTO 75
1Ø4Ø  GOTO 85
```

Here is the line-by-line description.

Line 1Ø	DIMs the array V(2ØØ,2) to hold note information.
Line 15	creates a simple array holding waveform information.
Line 2Ø	initializes M = 54272 and the counter C = Ø.
Line 25	clears the chip.
Line 3Ø	sets the low and high pulse width numbers.
Line 35	sets the volume.
Line 4Ø	sets the ADSR for the first voice.
Line 5Ø	reads the first three elements from the DATA list.
Line 55	stores the information in the two-dimensional array and increments the note counter Z.
Line 6Ø	signals the end of the array loading process.
Line 7Ø	instructs the computer to loop back and read the next 3 elements of note information from the DATA list.
Line 75	sets up the counter J to play the song twice and then end.
Line 8Ø	resets Z = Ø for the play portion of the program.
Line 85	checks for flags.
Line 9Ø	sets TM = TI, the jiffy clock.
Line 1ØØ	POKEs the high and low frequency number registers.

Line 11Ø	turns on the waveform and opens the gate by adding one to the waveform number WF(I).
Line 12Ø	checks the clock and plays the note for the duration value.
Line 13Ø	closes the gate using only the waveform number WF(I).
Lines 2ØØ–23Ø	DATA and flags. *Note* Ø,Ø,Ø—flag to signal the end of the READ section (see line 6Ø).
Lines 1ØØØ–1Ø17	manipulates the DATA in the note array by re-setting the note counter Z; changes the waveform WF(I); changes ADSR when necessary. It is the changes in waveform WF(I) and envelope (ADSR) that change the character of the voice.
Line 1Ø4Ø	instructs the computer to go back to the play section.

Now let's put all of our programming knowledge together and create a program that will control multiple voices with independent duration for each voice.

33 Multiple Voices with Independent Duration

This program will allow you to translate sheet music or your own compositions into a format that can be executed by the computer. Your one limitation will be the number of voices available at any given time—which, of course, is three.

To cut down on the length of the DATA list, we will be using a technique of encoding our pitch outlined in Chapter 9. With this approach, you do not have to enter the high and low frequency numbers. Instead, you enter a single number known as the frequency number. This number may also be obtained from the Table of Note Values.

In this program, three arrays are dimensioned to hold the information of the song. One holds the high frequency numbers of all three voices. A second holds the low frequency numbers, and a third holds the duration information.

During the READ portion of the program, the frequency numbers are READ from the DATA list. Each number then undergoes a mathematical operation built into the program. This operation breaks down the number into a high frequency and a low frequency number. This information is then loaded into the proper array. An explanation of the formula may be found in Chapter 9.

The second number READ from the DATA statement is the duration number. This number has an entirely new meaning for you, and is the secret to controlling the voices independently.

Here is how it works. The playing portion of the program is built around a short time loop. All three voices are processed through the same loop simultaneously. The length of one pass through the loop represents a sixteenth note, two passes an eighth note, etc. The loop time may be as long or short as suits the music's tempo. The number that is encoded as the duration represents the number of passes through the loop.

Since the duration of a note now means the number of passes through the loop, when the duration is read from the DATA list, it must load the array with the gated waveform number and pitch controls for each pass through the loop. On the final pass, it must load a closed gate waveform to turn off the note.

What this means is that a quarter note A-44Ø is represented in the DATA as 7217,4. After the note has been READ, a series of operations are performed upon the numbers, resulting in the loading of three arrays with the following information:

High FQ array	Low FQ array	Duration array
28	49	17
28	49	17
28	49	17
28	49	16

The program is designed so that all the information for Voice 1 is read in first. Then Voice 2 is read in, and finally Voice 3. The operation is quite complex. As a result, there is often a 3Ø second silence before you hear the music.

The playing portion of the program is the same as in our previous program. The only real difference is that you must use three counters instead of one to keep track of the three arrays.

The BASIC programming language does not operate with the same speed as machine language programming. The more conditional statements it must check the slower the program will run. Often, the first note duration of a song played for the first time will be slightly out of sync. This is because the computer is encountering the counters and conditional statements for the first time. You can compensate for this by programming a Ø,1 as the first piece of DATA for each voice. Then the set-up time will not interfere with the music.

We'll talk more about this program later. For now, type it in and sit back. The program plays continuously. If you want to end it, simply hit the RUN/STOP and RESTORE keys.

This program demonstrates three independent voices. The first two play separate patterns that cycle over and over, while the third plays the melody.

```
1  PRINT "■" : REM *** CLEAR SCREEN ***
2  PRINT "*** MUSIC BY JIM VOGEL ***"
3  PRINT "*** PROGRAM DESIGN ***"
4  PRINT "*** JIM VOGEL AND NEVIN SCRIMSHAW ***"
5  PRINT "********************************"
6  PRINT "PLEASE WAIT: SETTING UP ARRAYS"
7  PRINT "********************************"
1Ø FOR S = 54272 TO 54296 : POKE S,Ø : NEXT
15 M = 54272
```

```
 20  DIM H(500,2),L(500,2),D(500,2)
 30  V(0) = 17 : V(1) = 17 : V(2) = 17
 40  REM *** LOAD ARRAY ***
100  FOR K = 0 TO 2
110  I = 0
120  READ N,DR
130  IF DR = 0 THEN 200
140  WF = V(K) : WX = WF - 1
150  HF% = N/256 : LF% = N - 256*HF%
160  IF DR = 1 THEN H(I,K) = HF% : L(I,K) = LF% : D(I,K) = WF :
     I = I+1 : GOTO 120
170  FOR J = 1 TO DR-1 : H(I,K) = HF% : L(I,K) = LF% : D(I,K) =
     WF : I = I+1 : NEXT
180  H(I,K) = HF% : L(I,K) = LF% : D(I,K) = WX
190  I = I+1 : GOTO 120
200  NEXT K
250  REM *** SOUND SETTINGS ***
300  POKE M+5,0 : POKE M+6,255
310  POKE M+12,0 : POKE M+13,255
320  POKE M+19,0 : POKE M+20,255
330  POKE M+24,15
399  REM *** FACING THE MUSIC ***
400  P1 = 0 : P2 = 0 : P3 = 0
410  POKE M,L(P1,0) : POKE M+1,H(P1,0)
420  POKE M+7,L(P2,1) : POKE M+8,H(P2,1)
430  POKE M+14,L(P3,2) : POKE M+15,H(P3,2)
440  POKE M+4,D(P1,0) : POKE M+11,D(P2,1) : POKE M+18,D(P3,2)
445  FOR T = 1 TO 3 : NEXT T
450  P1 = P1+1 : P2 = P2+1 : P3 = P3+1
455  IF D(P1,0) = 0 THEN 2000
460  IF D(P2,1) = 0 THEN 2500
465  IF D(P3,2) = 0 THEN 3000
470  GOTO 410
600  REM *** LESS DATA! ***
610  DATA 0,1,1432,2,3406,2,3215,2,2864,2,3215,2,3406,2
620  DATA 2145,12
630  DATA 1432,2,3406,2,3215,2,2864,2,3215,2,3406,2
640  DATA 4291,6,2145,6,0,0
700  DATA 0,1,0,48,5728,2,6812,2,5103,2,6430,2,5728,2,6812,2
710  DATA 8583,2,7647,2,6812,2,6430,2,5728,2,5103,2,0,0
800  DATA 0,1,0,96,17167,40,11457,4,17167,4,15294,44,11457,4,15294,40,
     1457,4
```

```
810  DATA 15294,4,13625,24,11457,24,12860,40,11457,4,9634,4,8583,44,
     11457,4
820  DATA 12860,40,11457,4,8583,4,7647,48,0,0
2000 P1 = 1 : GOTO 460
2500 P2 = 49 : GOTO 465
3000 P3 = 1 : P2 = 1 : GOTO 410
```

Here is the line-by-line description.

Lines 1–7	set up graphics display.
Line 10	clears the chip.
Line 15	initializes M to 54272.
Line 20	DIMs three arrays to hold High FQ, Low FQ and duration numbers.
Line 30	establishes an array to hold waveform numbers for three voices.
Line 40	REM: Array loading procedure begins.
Line 100	instructs the computer to go through the process three times—once for each voice—when K = 0, Voice 1; when K = 1, Voice 2; when K = 2, Voice 3.
Line 110	initializes the note counter I.
Line 120	READs the first two numbers from the DATA and calls them N (for note) and DR (for duration).
Line 130	instructs the computer, IF the duration number is 0 (the last entry for each voice's DATA), THEN it is go to 200 (the end reading for each voice).
Line 140	sets the variable WF equal to V(K). (The gated—on—waveform = WF; the closed gate waveform—off—is called WX(WF − 1).
Line 150	sets the integer portion of HF (high frequency) equal to the frequency number (N) divided by 256; and the integer portion of LF (low frequency) equal to the frequency number (N) minus the HF number times 256.
Line 160	instructs the computer, IF the duration number is 1, THEN store the high and low FQ number in the proper array and store the gated waveform number in the duration array; increment the note counter I; go to line 120 for the next note.
Line 170	establishes a FOR/NEXT loop that stores notes with durations greater than 1; fills the arrays with the high and low FQ numbers and a gated waveform duration for all but the last pass through the time loop of the note. On each pass, the note counter I is incremented by 1.
Line 180	stores the last beat in the array with the same high and low FQ numbers, but with a closed gate waveform. This line turns off the note and begins the release cycle.

Line 19Ø — increments the note counter I for the last beat stored in line 18Ø, and instructs the computer to go back to line 12Ø to read the next note.

Note This cycle will repeat until a Ø is read in the DR part of the READ statement. (See line 13Ø.)

Line 2ØØ — instructs the computer to go back and READ in the DATA for the next voice.
Line 25Ø — REM: Sound settings.
Lines 3ØØ–32Ø — ADSR for three voices.
Line 33Ø — sets the volume. (Optional filter settings should be placed here if used.)
Line 399 — REM: Play the music portion of the program.
Line 4ØØ — initializes the note counters for Voice 1, 2 and 3 to Ø.
Lines 41Ø–43Ø — POKEs the high and low frequency registers for all three voices.
Line 44Ø — POKEs the waveform control registers of all three voices with the numbers stored in the duration array.
Line 445 — establishes time loop for 16th note; plays the note.
Line 45Ø — increments the three counters.
Line 455 — checks the next duration for Voice 1; instructs the computer, IF the value stored in the array is Ø, to THEN go to line 2ØØ.
Line 46Ø — checks the next duration for Voice 2; instructs the computer, IF the value is Ø, to THEN go to line 25ØØ.
Line 465 — checks the next duration for Voice 3; instructs the computer, IF the value is Ø, to THEN go to line 3ØØØ.
Line 47Ø — instructs the computer to go back to line 41Ø and play the next $^{1}/_{16}$th bar (the next beat).

Note The Ø in the DATA is the last duration value for each voice. In the READ portion of the program, this acts as a flag to signal the end of the READ cycle for each voice. In the play portion, it acts as a flag to signal an exit from the program. This enables us to manipulate the order of notes for each voice.

Line 6ØØ — REM: DATA
Lines 61Ø–64Ø — Voice 1 DATA.
Lines 7ØØ–71Ø — Voice 2 DATA.
Lines 8ØØ–82Ø — Voice 3 DATA.

Note The first two pieces of DATA for all three voices are Ø,1. This is so that distortion of the first note, as described earlier, does not occur.

Line 2ØØØ — manipulates the array for Voice 1; instructs the computer to return to line 46Ø to check the next voice.

Line 25ØØ manipulates the array for Voice 2; instructs the computer to return to line 465 to check the next voice.

Line 3ØØØ manipulates the array for Voice 3; instructs the computer to return to line 41Ø to play the next note.

DEBUGGING THE PROGRAM

If you were extremely careful in typing in this program, then it ran. Great! Chances are, though, that you made some mistakes. Usually, if you make an error in this type of program, you will receive a message such as, BAD SUBSCRIPT IN LINE 17Ø.

To debug, try this procedure.

Locate what went wrong. List the line where the error occurred. For example:

```
LIST 17Ø

17Ø  FOR J = 1 TO DR - 1: H(I,K) = HF%: L(I,K) = LF%:
     D(I,K) = WF: I = I + 1: NEXT
```

Then, use the computer to print out the current value of each variable.

```
PRINT I
```

The computer responds:

```
I = 5Ø1
```

This means that you have more elements than can be accommodated by your array DIM H(5ØØ,2), and probably, somewhere along the line, the computer did not reset I to Ø when it was supposed to do so. Did you place the zeros at the end of the DATA? Were the values properly entered in the DATA? Did you leave one out? Did you put a 'full stop' between elements instead of a comma? etc.

Do this with each variable. In this way, you can track down the bug.

Be sure to save the program, even if it does not work properly. Then you can update it. There is always the possibility of a 'crash'. This occurs when the computer is frozen and you can't break out. The only option is to turn off the computer. If you've saved your work, you're safe. If not—well, there goes an hour.

Programming multiple voices with independent duration is no easy trick. It takes time and patience. Use our program as a model, and make up your own music. Or try and translate some sheet music, if you prefer. When you've become proficient, try to tailor your own program.

34 Real Time Music: An Organ Keyboard

Up until now, we have been playing 'computer time' music. That is, we write a program and the computer plays it for us. Your Commodore 64 has the ability, however, to play 'real time' music. You press a key, and the computer responds with a note. At the time of writing, Commodore has announced that it will be releasing a 'piano keyboard' which can be hooked up to your 64. This event, however, is still months away. The next best thing is to write a program to turn the keyboard of the computer into an instrument. There have been a few excellent examples of this type of program in recent computer magazines. There is also a wonderful program for a keyboard in the appendix of your *Commodore User's Guide*. Though a little difficult to understand, it's well worth taking the time to type it into your 64 and to save it on disc or tape.

We'd like to add our keyboard program to the growing list, and we'll show you how you can write one of your own. The beauty of this process is that you can assign all kinds of functions to different keys.

Our program is for a 'Chromatic Chord Organ'. With it, you have control of all three voices, a row of keys for each; a two-octave range for each voice; and a choice of two waveforms. The program allows you to hear permutations of a three-note chord over the two-octave range. In addition, you can use two voices as a drone and a third voice as a melody. This is because the voices remain ON for as long as you wish. The function keys are used to turn off the sound for each voice independently or all together. To end the program, you press the space bar.

Here is the program. The graphic display at the beginning is purely optional. We'll show you how the program works after you've played some 'real time' music.

```
100  PRINT "■" : REM *** CLEAR SCREEN ***
110  PRINT "***DONALD'S CHROMATIC CHORD ORGAN***"
120  PRINT "***BY DONALD KAHN & JIM VOGEL***"
125  PRINT "**********************************"
135  PRINT "***PRESS TOP ROW KEYS FOR VOICE I***"
145  PRINT "***PRESS 2ND ROW KEYS FOR VOICE II***"
155  PRINT "***PRESS 3RD ROW KEYS FOR VOICE III***"
160  PRINT "**********************************"
165  PRINT "PRESS F1 TO TURN OFF VOICE I"
170  PRINT
175  PRINT "PRESS F3 TO TURN OFF VOICE II"
180  PRINT
185  PRINT "PRESS F5 TO TURN OFF VOICE III"
190  PRINT
195  PRINT "PRESS F7 TO TURN OFF ALL VOICES"
200  PRINT
205  PRINT "PRESS F2 FOR TRIANGLE WAVEFORM"
210  PRINT
220  PRINT "PRESS F4 FOR SAWTOOTH WAVEFORM"
230  PRINT
240  PRINT "PRESS F6 FOR LOWER OCTAVE"
250  PRINT
260  PRINT "PRESS F8 FOR LOWER OCTAVE"
270  PRINT
280  PRINT "PRESS SPACE BAR TO END"
400  FOR W = 54272 TO 54296 : POKE W,0 : NEXT
410  Z = 54272
420  P = 0
430  DIM A(12),B(12),C(12),D(12,3)
440  FOR N = 0 TO 12 : READ D(N,0) : READ D(N,1) : NEXT
450  FOR N = 0 TO 12 : READ D(N,2) : READ D(N,3) : NEXT
460  FOR N = 0 TO 12 : READ A(N) : NEXT
470  FOR N = 0 TO 12 : READ B(N) : NEXT
480  FOR N = 0 TO 12 : READ C(N) : NEXT
490  POKE Z+24,15
500  POKE Z+5,0 : POKE Z+6,240
510  POKE Z+12,0 : POKE Z+13,240
520  POKE Z+19,0 : POKE Z+20,240
530  POKE Z+4,17 : POKE Z+11,17 : POKE Z+18,17
```

```
540  GET M$ : IF M$ = "" THEN 540
550  M = ASC(M$)
560  IF M = 32 THEN 900
570  IF M = 136 THEN POKE Z + 1,0 : POKE Z,0 : POKE Z + 8,0 : POKE
     Z + 7,0 : POKE Z + 15,0
580  IF M = 136 THEN POKE Z + 14,0 : GOTO 540
600  IF M = 139 THEN P = 0 : GOTO 540
610  IF M = 140 THEN P = 2 : GOTO 540
620  IF M = 133 THEN POKE Z + 1,0 : POKE Z,0 : GOTO 540
630  IF M = 134 THEN POKE Z + 8,0 : POKE Z + 7,0 : GOTO 540
640  IF M = 135 THEN POKE Z + 15,0 : POKE Z + 14,0 : GOTO 540
650  IF M = 138 THEN POKE Z + 4,33 : POKE Z + 11,33 : POKE Z + 18,33 :
     GOTO 540
660  IF M = 137 THEN POKE Z + 4,17 : POKE Z + 11,17 : POKE Z + 18,17 :
     GOTO 540
670  I = 0
680  IF M = A(I) THEN POKE Z + 1,D(I,P) : POKE Z,D(I,P + 1) : GOTO 540
690  IF M = B(I) THEN POKE Z + 8,D(I,P) : POKE Z + 7,D(I,P + 1) : GOTO 540
700  IF M = C(I) THEN POKE Z + 15,D(I,P) : POKE Z + 14,D(I,P + 1) : GOTO
     540
710  I = I + 1
720  IF I < = 12 THEN 680
730  GOTO 540
900  FOR L = 54272 TO 54296 : POKE L,0 : NEXT
1000 DATA 16,195,17,195,18,209,19,239,21,31,22,96,23,181,25,30,26,156,
     28,49
1010 DATA 29,223,31,165,33,135
1020 DATA 33,135,35,134,37,162,39,223,42,62,44,193,47,107,50,60,53,57,56,
     99
1030 DATA 59,190,63,75,67,15
1040 DATA 95,49,50,51,52,53,54,55,56,57,48,43,45
1050 DATA 81,87,69,82,84,89,85,73,79,80,64,42,94
1060 DATA 65,83,68,70,71,72,74,75,76,58,59,61,13
```

HOW TO BUILD AN ORGAN KEYBOARD

To build our organ keyboard, we made use of our knowledge of arrays, combined with the BASIC GET command and the Commodore ASC codes.

The GET command

The GET command instructs the computer to read each key that you type. The character is then stored in the keyboard buffer. The GET statement usually takes this form:

```
10  GET A$: IF A$ = " " THEN 10
```

When this line is encountered, the computer stays on the line cycling back and forth, waiting for you to press a key. When the key is pressed, the computer executes whatever commands are linked to that specific key. Usually it is necessary to have a command to return to the GET A$ line, so that new information can be processed.

The ASC code

The 64 uses the ASC code to evaluate the character received in the GET statement. A table of ASC codes for the 64 may be found in the appendices of both the *User's Guide* and the *Programmer's Reference Guide*. Each key on your computer has an ASC code, which is a number that specifically relates to that key. An ASC code number is programmed as ASC(X) where X equals the code number of the desired key. If you check your table, you will see that ASC(32) means the space bar, while ASC(65) means the letter A, and so on.

With this knowledge, you can understand the concept that lies behind this program.

PROGRAM OVERVIEW

This organ program is put together in modules. The first module (lines 100–280) contains the graphics, which are optional. The next section (lines 430–480) creates arrays to hold the frequency numbers for two full octaves. In addition, it establishes three more arrays to hold the ASC codes for the three rows of keys we will be using. All this information is stored in the DATA bank.

The next module is for permanent sound setting (lines 490–530).

The following section contains the GET command and a number of conditional statements to evaluate the character received (lines 540–730).

The final module is the 'play' portion of the program. It POKEs the appropriate sound registers with values stored in the arrays (lines 560–700).

For a closer look, follow this line-by-line program description.

Line 400	clears the chip.
Line 420	initializes the note counter P to 0.

Line 43Ø	DIMs four arrays. A, B and C will hold the ASC codes for the first three rows of keys. D is a two-dimensional array, and will hold the high and low FQ numbers for two octaves.
Lines 44Ø–48Ø	READs the DATA list and stores the information in the arrays.
Line 49Ø	sets the volume.
Lines 5ØØ–52Ø	sets the ADSR for three voices.
Line 53Ø	sets the waveform for three voices.
Line 54Ø	is the GET command: stay on line 54Ø until a key is pressed. The character received will be named M$.
Line 55Ø	sets M equal to the ASC code of the character received in the GET command, M$.
Line 56Ø	instructs the computer, IF space bar, to THEN go to 9ØØ (end program).
Lines 57Ø–58Ø	IF F7 (function key), turn off the sound for all three voices; go back to line 54Ø and GET next key.
Line 6ØØ	IF F6, THEN use the lower octave pitches go back to line 54Ø and GET next key pressed.
Line 61Ø	IF F8, THEN use the upper octave pitches; go back to line 54Ø and GET next key pressed.
Line 62Ø	IF F1, THEN turn off Voice1; go back to line 54Ø and GET next key pressed.
Line 63Ø	IF F3, THEN turn off Voice 2; go back to line 54Ø and GET next key.
Line 64Ø	IF F5, THEN turn off Voice 3; go back to line 54Ø and GET next key pressed.
Line 65Ø	IF F4, THEN turn on sawtooth waveform; go back to line 54Ø and GET next key pressed.
Line 66Ø	IF F2, THEN turn on triangle waveform, go to line 54Ø for next key.
Line 67Ø	sets the counter I to Ø. (This counter is used for lines 68Ø–72Ø.)
Lines 68Ø–72Ø	establish a loop to evaluate the top three rows of keys. When the key is found, the appropriate high and low frequency registers are set.

Note all three rows are set to play the same chromatic scale. If the second key of the second row is pressed, the second note of the second voice is played.

Line 73Ø	instructs the computer, after checking all the conditionals, go back the GET the next note.
Line 9ØØ	ends the program.
Lines 1ØØØ–1Ø1Ø	DATA for low octave high and low FQ numbers.
Lines 1Ø2Ø–1Ø3Ø	DATA for upper octave high and low FQ numbers.
Line 1Ø4Ø	DATA ASC codes for top row.
Line 1Ø5Ø	DATA ASC codes for second row.
Line 1Ø6Ø	DATA ASC codes for third row.

The organ takes a bit of time to get used to. Since most people are used to the major scale and the organ produces the chromatic scale, the notes do not work the way many people would expect. But figuring out the notes can be an interesting investigation. This organ offers much opportunity for unique chordal structures.

Also try the drone effect. Set up a two-note drone in the lower octave, using the top two rows. Then, switch to the upper octave, using the Function 8 key. Use the third row for your melody lines. It will remain in the upper octave. As long as you do not press a key in the top two rows, they will continue droning in the lower octave.

You can play some beautiful, slow meditative music using this set-up.

If you want to change the pitches of the keys, you can do so by altering the DATA in lines 1000–1030. Different octaves may be selected, or, if you prefer, a diatonic, minor or modal scale.

If you alter the DATA in lines 1000–1030 to the list below, your chromatic chord organ will be converted into an F Lydian modal organ. If you use an F and C drone in the top two rows, you have the tuning to play a North Indian Classical Raga called Yaman Kalyan.

The drone is obtained by pressing (←) on the top row and (T) on the second row.

```
1000 DATA 11,48,12,143,14,24,15,210,16,195,18,209,21,31,22,96,25,30,28,49
1010 DATA 31,165,33,135,37,162
1020 DATA 22,96,25,30,28,49,31,165,33,135,37,162,42,62,44,193,50,60,56,99
1030 DATA 63,75,67,15,75,69
```

Try making up your own organ. When you do, make sure to program 26 notes, no more, no less. That's what it takes for the program to run properly.

Section VI:

Finale

Sounding a Final Note

Computers have come to play a major role in the industrialized, and now, even in the developing nations of the world. Their uses are so far-reaching and varied that we cannot even begin to list them. With the advent of the new and inexpensive microprocessors, the so-called 'revolution' has taken the next step: into our home.

Like so many of the great technological advances of our time, computers have the capacity, if used wisely, to enrich our lives and culture. It is up to us to develop the 'human' side of computers. What better area for this effort, then, than music—a truly international language, a form of communication that crosses time and cultural barriers?

There are some who feel that the computer has no place in music. "How can a machine," they ask, "express the subtlety of the human spirit?" To a certain extent, these people are correct. We can never 'replace' the musician by a computer. But it should be noted that a computer program is an extension of the person who created it as a vehicle for musical expression, just as the computer itself is an extension of the human mind reaching out for knowledge. It is a tool for scientists, artists, mathematicians, business people, teachers, and, yes, even musicians.

It is in this spirit that we have written this book. We hope that it has been of some help in your music projects, and we are very interested in hearing from you about these projects. We invite you to write with questions, comments and programs to Jim Vogel or Nevin Scrimshaw, c/o Ashfield Books, Main Street, Ashfield, MA 01330, USA.

Bibliography

Brook, Barry S. Editor. *Musicology and the Computer* and *Musicology 1966–2000: A Practical Problem*, American Musicological Society/ G.N.Y.C., City University Press: 1965.

Commodore 64 Programmer's Reference Guide, Commodore Business Machines; Howard W. Sams and Co., Inc.; first edition, third printing: 1983.

Commodore 64 User's Guide, Commodore Business Machines; Howard W. Sams and Co., Inc.; second edition: 1982.

Howe, Hubert S., Jr., *Electronic Music Synthesis*, W.W. Norton, New York and London: 1975.

Lincoln, Harry B., Editor, *The Computer and Music*, Cornell University Press, Ithaca and London: 1970.

Mandell, Steven L., *BASIC*, second edition; Academic Press, Harcourt Brace Jovanovich; New York: 1982.

The Synthesizer, second edition; Roland Corporation: 1979.

Trythall, Gilbert, *Principles and Practice of Electronic Music*, Grosset and Dunlap, New York: 1973.

Von Foerster, Heinz and Beauchamp, James W., Editors, *Music By Computers*, John Wiley and Sons, Inc., New York: 1969.

Table of Note Values

Note	Decimal	High FQ	Low FQ
Octave 0			
C-0	268	1	12
C#-0	284	1	28
D-0	301	1	45
D#-0	318	1	62
E-0	337	1	81
F-0	358	1	102
F#-0	379	1	123
G-0	401	1	145
G#-0	425	1	169
A-0	451	1	195
A#-0	477	1	221
B-0	506	1	250
Octave 1			
C-1	536	2	24
C#-1	568	2	56
D-1	602	2	90
D#-1	637	2	125
E-1	675	2	163
F-1	716	2	204
F#-1	758	2	246
G-1	803	3	35
G#-1	851	3	83
A-1	902	3	134
A#-1	955	3	187
B-1	1012	3	244
Octave 2			
C-2	1072	4	48
C#-2	1136	4	112
D-2	1204	4	180
D#-2	1275	4	251
E-2	1351	5	71
F-2	1432	5	152
F#-2	1517	5	237
G-2	1607	6	71
G#-2	1703	6	167
A-2	1804	7	12
A#-2	1911	7	119
B-2	2025	7	233
Octave 3			
C-3	2145	8	97
C#-3	2273	8	225
D-3	2408	9	104
D#-3	2551	9	247
E-3	2703	10	143
F-3	2864	11	48
F#-3	3034	11	218
G-3	3215	12	143
G#-3	3406	13	78
A-3	3608	14	24
A#-3	3823	14	239
B-3	4050	15	210

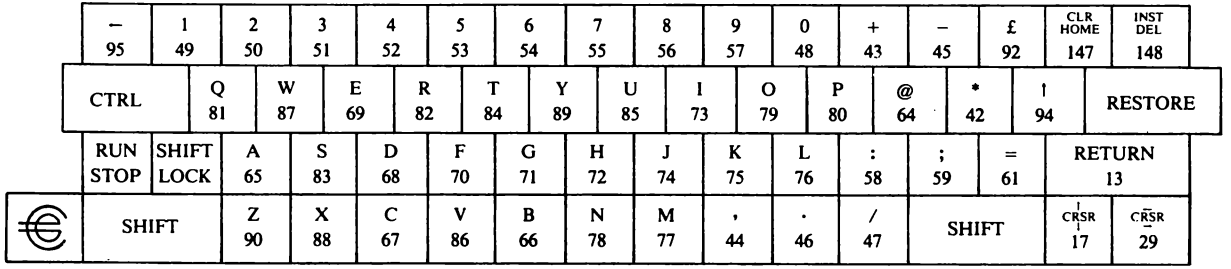
Table of Note Values cont.

Note	Decimal	High FQ	Low FQ
Octave 4			
C-4	4291	16	195
C#-4	4547	17	195
D-4	4817	18	209
D#-4	5103	19	239
E-4	5407	21	31
F-4	5728	22	96
F#-4	6069	23	181
G-4	6430	25	30
G#-4	6812	26	156
A-4	7217	28	49
A#-4	7647	29	223
B-4	8101	31	165
Octave 5			
C-5	8583	33	135
C#-5	9094	35	134
D-5	9634	37	162
D#-5	10207	39	223
E-5	10814	42	62
F-5	11457	44	193
F#-5	12139	47	107
G-5	12860	50	60
G#-5	13625	53	57
A-5	14435	56	99
A#-5	15294	59	190
B-5	16203	63	75
Octave 6			
C-6	17167	67	15
C#-6	18188	71	12
D-6	19269	75	69
D#-6	20415	79	191
E-6	21629	84	125
F-6	22915	89	131
F#-6	24278	94	214
G-6	25721	100	121
G#-6	27251	106	115
A-6	28871	112	199
A#-6	30588	119	124
B-6	32407	126	151
Octave 7			
C-7	34334	134	30
C#-7	36376	142	24
D-7	38539	150	139
D#-7	40830	159	126
E-7	43258	168	250
F-7	45830	179	6
F#-7	48556	189	172
G-7	51443	200	243
G#-7	54502	212	230
A-7	57743	225	143
A#-7	61176	238	248
B-7	64814	253	46

SID Chip Control Chart

		Bit	7	6	5	4	3	2	1	0
		Decimal	128	64	32	16	8	4	2	1
	Address	Function	**Voice 1**							
0	54272	Low FQ	FQ 7	FQ 6	FQ 5	FQ 4	FQ 3	FQ 2	FQ 1	FQ 0
1	54273	High FQ	FQ 15	FQ 14	FQ 13	FQ 12	FQ 11	FQ 10	FQ 9	FQ 8
2	54274	Low PW	PW 7	PW 6	PW 5	PW 4	PW 3	PW 2	PW 1	PW 0
3	54275	High PW	/////	/////	/////	/////	PW 11	PW 10	PW 9	PW 8
4	54276	Waveform	Noise	⊓⊔	⩘	△	Test	R mod	Sync	Gate
5	54277	Atk/Dec	Atk 3	Atk 2	Atk 1	Atk 0	Dec 3	Dec 2	Dec 1	Dec 0
6	54278	Sus/Rel	Sus 3	Sus 2	Sus 1	Sus 0	Rel 3	Rel 2	Rel 1	Rel 0
			Voice 2							
7	54279	Low FQ	FQ 7	FQ 6	FQ 5	FQ 4	FQ 3	FQ 2	FQ 1	FQ 0
8	54280	High FQ	FQ 15	FQ 14	FQ 13	FQ 12	FQ 11	FQ 10	FQ 9	FQ 8
9	54281	Low PW	PW 7	PW 6	PW 5	PW 4	PW 3	PW 2	PW 1	PW 0
10	54282	High PW	/////	/////	/////	/////	PW 11	PW 10	PW 9	PW 8
11	54283	Waveform	Noise	⊓⊔	⩘	△	Test	R mod	Sync	Gate
12	54284	Atk/Dec	Atk 3	Atk 2	Atk 1	Atk 0	Dec 3	Dec 2	Dec 1	Dec 0
13	54285	Sus/Rel	Sus 3	Sus 2	Sus 1	Sus 0	Rel 3	Rel 2	Rel 1	Rel 0
			Voice 3							
14	54286	Low FQ	FQ 7	FQ 6	FQ 5	FQ 4	FQ 3	FQ 2	FQ 1	FQ 0
15	54287	High FQ	FQ 15	FQ 14	FQ 13	FQ 12	FQ 11	FQ 10	FQ 9	FQ 8
16	54288	Low PW	PW 7	PW 6	PW 5	PW 4	PW 3	PW 2	PW 1	PW 0
17	54289	High PW	/////	/////	/////	/////	PW 11	PW 10	PW 9	PW 8
18	54290	Waveform	Noise	⊓⊔	⩘	△	Test	R mod	Sync	Gate
19	54291	Atk/Dec	Atk 3	Atk 2	Atk 1	Atk 0	Dec 3	Dec 2	Dec 1	Dec 0
20	54292	Sus/Rel	Sus 3	Sus 2	Sus 1	Sus 0	Rel 3	Rel 2	Rel 1	Rel 0
			Volume/Filters/Misc.							
21	54293	Low FC	/////	/////	/////	/////	/////	FC 2	FC 1	FC 0
22	54294	High FC	FC 10	FC 9	FC 8	FC 7	FC 6	FC 5	FC 4	FC 3
23	54295	Res/Filt	Res 3	Res 2	Res 1	Res 0	Filtex	Filt 3	Filt 2	Filt 1
24	54296	Mode/Vol	3 off	HP	BP	LP	Vol 3	Vol 2	Vol 1	Vol 0
25	54297	Pot 1	7	6	5	4	3	2	1	0
26	54298	Pot 2	7	6	5	4	3	2	1	0
27	54299	Osc 3	7	6	5	4	3	2	1	0
28	54300	Env 3	7	6	5	4	3	2	1	0

Keyboard ASCII Codes



F1	133
F2	137
F3	134
F4	138
F5	135
F6	139
F7	136
F8	140

Other titles of interest

Easy Programming for the Commodore 64 **£6.95**
Ian Stewart & Robin Jones

Commodore 64 Assembly Language **£7.95**
Bruce Smith

Commodore 64 Machine Code **£7.95**
Ian Stewart & Robin Jones

Programming for REAL Beginners: Stage 1 **£3.95**
Philip Crookall

Programming for REAL Beginners: Stage 2 **£3.95**
Philip Crookall

Brainteasers for BASIC Computers **£4.95**
Gordon Lee
'Just the job for a wet afternoon with the computing class'—*Education Equipment*

Computing: A Bug's Eye View **£2.95**
Cosgrove
A collection of the humorous cartoons that have appeared in 'Shiva's Friendly Micro Series'.

Easy Programming for the BBC Micro **£5.95**
Eric Deeson
'A beginners guide in the true sense'—*Educational Computing*

Further Programming for the BBC Micro **£5.95**
Alan Thomas
'If you are sinking with Basic on your BBC Micro, this could be a life raft!"—*Educational Computing*

BBC Micro in Education **£6.50**
Eric Deeson

BBC Micro Assembly Language **£7.95**
Bruce Smith
'This book is an excellent beginner's guide'—*Acorn User*

BBC Micro Assembler Workshop **£6.95**
Bruce Smith

Easy Programming for the Electron **£5.95**
Eric Deeson

Electron Assembly Language **£7.95**
Bruce Smith

Electron Assembler Workshop **£6.95**
Bruce Smith

Easy Programming for the ZX Spectrum **£5.95**
Ian Stewart & Robin Jones

'. . . will take you a long way into the mysteries of the Spectrum: is written with a consistent and humorous hand: and shares the affection the authors feel for the computer'—*ZX Computing*

Further Programming for the ZX Spectrum **£5.95**
Ian Stewart & Robin Jones

'Again written with clarity and style'—*Educational Computing*

Spectrum Machine Code **£5.95**
Ian Stewart & Robin Jones

'An excellent grounding in machine and assembly language'—*Sinclair User*

Computer Puzzles: For Spectrum and ZX81 **£2.50**
Ian Stewart & Robin Jones

'What a gem of a book!'—*Education Equipment*

Games to Play on Your ZX Spectrum **£1.95**
Martin Wren-Hilton

Spectrum in Education **£6.50**
Eric Deeson

PEEK, POKE, BYTE & RAM! Basic Programming for the ZX81 **£4.95**
Ian Stewart & Robin Jones

'Far and away the best book for ZX81 users new to computing'—*Popular Computing Weekly*

Machine Code and better Basic **£7.50**
Ian Stewart & Robin Jones

'A beautifully written course in so-called advanced programming—*Education Equipment*

Easy Programming for the Oric-1 **£5.95**
Ian Stewart & Robin Jones

Games to Play on your Oric-1 **£4.95**
Czes Kosniowski

Machine Code for the Atmos and Oric-1 **£6.95**
Bruce Smith

Easy Programming for the Dragon 32 **£5.95**
Ian Stewart & Robin Jones

Further Programming for the Dragon 32 **£5.95**
Ian Stewart & Robin Jones

Dragon Machine Code **£6.95**
Robin Jones & Eric Cowsill

Shiva Software

BBC Micro Utilities 1 **£6.95**
Bruce Smith
A machine code monitor plus a series of machine code subroutines.

Supergraph *Summer 1984*
David Tall
This teaches an appreciation of coordinate geometry and lays the groundwork for calculus (covered in subsequent tapes).

Spectrum Special 1 **£5.95**
Ian Stewart & Robin Jones
A selection of 10 educational games and puzzles, including St. Moritz, Tower of Hanoi, Picasso and Tiger Hunt.

Spectrum Special 2 **£5.95**
Ian Stewart & Robin Jones
Utilities, games and puzzles including: World Map, Automatic Morse, Magic Forest and Brickbat.

Spectrum Special 3 **£5.95**
Ian Stewart & Robin Jones
A further collection of leisure games and puzzles including: Dayfinder, Star Chart, Psychospectrology and Roulette.

Spectrosim **£9.95**
Robin Helmer
A utility program that overcomes the first hurdles of machine code programming on the ZX Spectrum.

The SHIVA First Mathematics Programme
(for children 5–8 years)
Developed by Iris V. Hewett, M. Ed.
Each of the six tapes on numeracy and logic contains five graphically illustrated programs with full documentation.

Lift off with Numbers **Launching Logic**

Additional Fun **Sets and Operators**

Playing with Places **Decisions**

Cassettes are available for the BBC Micro Model B and versions for the Sinclair Spectrum, Electron, Commodore 64, Dragon 32 and RML 480Z microcomputers will follow soon.

Have you ever wanted an organ in your lounge? Or perhaps a banjo or a steam train? If so, then read on!

Here's your chance to make your Commodore 64 sound like anything from a musical instrument to an electronic synthesizer.

It is designed for people with little or no programming experience, and includes the fundamental techniques of musical composition, programming in BASIC and electronic sound synthesis, including:

- multiple voice programming
- sound effects
- real time music
- filters

There are numerous ready to RUN programs which illustrate the range of the 64's 'voice', such as:

- Flight of the Bumble Bee
- Pots and Pans
- Insect Speech
- Oh Suzannah!

Put your stereo away. Find out how to make your Commodore play!

GB £ NET

ISBN 1-85014-019-7

9 781850 140191 00595

0684 NR06078 130,50

SHIVA

Shiva Publishing Limited



This was brought to you

from the archives of

http://retro-commodore.eu