SOUND EFFECTS & MUSIC ON THE COMMODORE 64

William Turner & Alf Vella



SOUND EFFECTS & MUSIC ON THE COMMODORE 64

Sound Effects and Music on the Commodore 64

William Turner & Alf Vella



Duckworth

To our wives

First published in 1985 by
Gerald Duckworth & Co. Ltd.
The Old Piano Factory
43 Gloucester Crescent, London NW1

© 1985 by William Turner and Alf Vella

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 or otherwise, without the prior permission of the publisher.

ISBN 0 7156 1986 1

British Library Cataloguing in Publication Data

Turner, William
Sound effects & music on the Commodore 64.—
(Duckworth computing)
1. Commodore 64 (Computer)—Programming
2. Computer sound processing
I. Title II. Vella, Alf
001.64'43 QA76.8.C64

ISBN 0-7156-1986-1

Photoset in North Wales by
Derek Doyle & Associates, Mold, Clwyd
Printed in Great Britain by
Redwood Burn Ltd., Trowbridge

Contents

Acknowledgments	7
Preface	9
1. Introduction to Sound	11
2. Strike a Note	21
3. The Musician's Aid	28
4. Play a Tune	32
5. Two Notes are Better than One	39
6. The Sound of Music	50
7. Variations on a Theme	58
8. Picture a Sound	68
9. Instruments of the Orchestra	82
10. That Sounds Like a...	98
11. The Sound of Flight	110
12. Ringing Tones	118
13. Noisy Noise	123
14. A Jungle of Sounds	127
15. Sounds Familiar	140

16. Sounds You Haven't Heard... Yet! 149

Appendixes 161

Further Reading 174

Index of Sounds 175

General Index 176

Acknowledgments

We would like to express our gratitude to Carol Vella for her many helpful suggestions and to Farzana Turner for her assistance with the artwork. Thanks are also due to Oxford Digital Enterprises and Peter Ells for technical support. Finally, we would particularly like to thank David Pringle for first opening our ears to the potential of sound synthesis using the Commodore 64.

Important

If you are one of those people who likes to dip into sections of a book rather than reading through it chapter by chapter, you should read this.

Nearly all the programs in the text are self-contained and just need to be typed into the Commodore and run. However, before trying to use the sound effects programs in the later chapters, you should read Chapter 10 to find out how they are organised.

Most of the programs produce a sound or music and then stop in silence. However, if you press the STOP button while a sound is playing, it will often continue to play! There are three ways in which you can remove the sound:

1. Turn down the volume on your TV.
2. Type POKE 54296,0 and press the RETURN key.
3. Switch off the computer!

Preface

The role played by computers has changed dramatically in just one generation. The two most notable changes are the way computers have become readily available to everybody and, following on from this, the way the great mystery of the computer has been lifted, particularly for young people. However, in all these changes one aspect of computing has been rather overlooked and underdeveloped–the fascinating world of sound.

Until recently home computers had very rudimentary facilities for sound production; indeed the Commodore 64's forerunner, the PET, did not have any sound device at all. Even now most microcomputers do not have anything like the extensive facilities available on the Commodore 64. The Sound Interface Device, or SID (as it will affectionately be called throughout this book), is a very sophisticated piece of hardware enabling you to control three independent voices to produce a wide range of sounds.

Unlike most books on the Commodore 64, in which sound is allocated one chapter and sound effects just a short section with one or two examples, we consider that sound deserves a fuller and complete treatment. So instead of just describing the 29 registers which are available on the SID chip and leaving it to you to work out how they might be used, we have adopted a functional approach. This means that we consider the tasks that you might want to perform, whether creating music or sound effects, and then describe how you can program them using SID.

Throughout the book we have provided full explanations, in non-technical language, although all necessary information has been given. In order to get the most out of this book we suggest that you progress through it step by step, although it is possible to dip into sections of particular interest, out of order if you wish.

Where programs are provided, we have set out to keep them simple but still displaying some style and structure. We have avoided programming tricks and clever short cuts as these tend to get in the way when trying to understand how they work.

The book falls naturally into three parts. First, we have described some basic ideas about the subject of sound itself, but to assist in this we have included many investigations, often using the Commodore 64. You will find fully described the Doppler effect, the cocktail party effect, the chorus effect and even the ghost effect!

Secondly, we deal with music. We describe what makes the characteristic sound of different musical instruments and provide a program to synthesise some conventional instruments as well as some which do not even exist! You will also find a program which converts your computer keyboard into an electronic organ. We show how you can store music for one, two or three voices in a computer program and provide examples. Because this music is programmed you can play it in ways never heard before–twice the speed or even backwards!

Finally, we move to sound effects. We have taken care to cover the subject comprehensively, not simply by providing programs for certain sound effects, although there are plenty of these in the book, but also by describing how these programs were composed so that you can work out how to produce your own. This is an exciting activity, as you will hear.

The world of sound is very exciting, partly because so much of it is unexplored. This book will provide you with a 'sound' basis on which to investigate it. Happy exploring!

W.T. & A.V.

1
Introduction to Sound

There is a well known saying, 'One picture is worth a thousand words' and we certainly wouldn't deny that there is some truth in it. However, we would like to extend it to 'If a picture is worth a thousand words then, with a touch of sound, it becomes even more colourful'.

It is a sad fact that although we have some idea of how early mankind saw the world, through surviving buildings, tools and art, the sounds of those times have now gone forever, and we can only guess at what the music might have sounded like. Certainly no civilisation has yet been discovered in which music does not have an important–even central–part to play.

There is no doubt that our visual sense is more developed and acute than our hearing. Our sight is so keen that we have to switch it off completely by closing our eyes in order to sleep, whereas slight sound does not affect sleep, particularly if the sound is in fact a gentle melody. However, sound is an absolutely necessary part of a complete experience. In the days of 'silent' films there was in fact either a piano or orchestral accompaniment to provide an impression of the sound that should have been there.

It is quite surprising, however, how sound alone can be quite self-contained. People are quite happy to listen to the radio or hi-fi for hours on end without feeling that anything is missing. Try watching TV without sound to find how necessary it is to complete the picture.

Recently many microcomputers have begun to provide facilities for sound generation to enhance their visual display. The Commodore 64 microcomputer has, among its chips, the 6851. This may not sound very impressive until it is revealed that it is through this chip that it can talk. It is properly called the Sound Interface Device, or SID for short. The Commodore's ability to synthesise sound is among the

most advanced available on microcomputers today. Although it has only 29 registers or settings it can independently produce sound and music through three voices and span eight octaves.

You are probably impatient to hear some of these sounds, so we include a few samples in the first chapter. When you run Program 1.1 you will be quite surprised at what you hear.

If you simply want to play with sound effects or hear some music, feel free to dip into the parts of the book that interest you. However, to drive SID effectively is no trivial task, and because of this we have deliberately taken a step-by-step approach to producing sound on the 64. In order to help you understand computer sound, we shall examine the subject of sound itself and, indeed, show you how the computer can be used to demonstrate these ideas. There are several sound experiments and sound investigations in the following chapters–listen out for the Ghost in Chapter 5!

Some sample sounds

Type in Program 1.1 to hear just a sample of what is to come. Note that the heart symbol in line 20 of the listing is obtained by pressing the Home/CLR button while holding down the SHIFT key. Do not forget to turn up the volume on your TV! The program is quite short and simple but provides an example of how sound can be used to enhance graphics, which might be part of a game. It demonstrates the two main aspects of using sound in a program, namely for music and for sound effects.

```
10 REM 1.1  FIRE!
20 B=54272           : PRINT "␣"
30 POKE 1304,81      : POKE 55576,5
40 POKE 1264,85      : POKE 55536,0
50 POKE B+24,15      : GOSUB 160
60 POKE B+5,1+0      : POKE B+6,252
70 POKE B+1,200      : POKE B+4,129
80 POKE B+4,128      : FOR I=1 TO 38
90 POKE 1582-I,31    : POKE 55854-I,7
100 FOR J=1 TO 10    : NEXT
110 POKE 1582-I,32 : NEXT
```

```
120 POKE 1544,31    : POKE 55816,7
130 FOR I=1 TO 300 : NEXT
140 GOSUB 160       : POKE B+24,0
150 END
160 REM PLAY TUNE
170 POKE B+5,34     : POKE B+6,165
180 READ F,T        : IF F<0 THEN RETURN
190 G=F*16.777      : H=INT(G/256)
200 L=G-256*H       : POKE B,L
210 POKE B+1,H      : POKE B+4,33
220 FOR T1=0 TO T*90                : NEXT
230 POKE B+4,32 : FOR T1=0 TO 30 : NEXT
240 GOTO 180
1000 DATA 523,1,523,1,523,2,523,1
1010 DATA 523,1,523,2,523,1,523,1
1020 DATA 698,2,784,2,880,4
1030 DATA 523,1,523,1,523,2,523,1
1040 DATA 523,1,698,2,880,2,784,2
1050 DATA 659,2,523,2
1060 DATA 523,1,523,1,523,2,523,1
1070 DATA 523,1,523,2,523,1,523,1
1080 DATA 698,2,784,2,880,4
1090 DATA 698,1,880,1,1047,3,932,1
1100 DATA 880,1,784,1,698,2,880,2
1110 DATA 698,4,-1,-1
1120 DATA 175,8,175,6,175,2,175,8
1130 DATA 208,6,196,2,196,6,175,2
1140 DATA 175,6,175,2,175,12,-1,-1
```

What is sound?

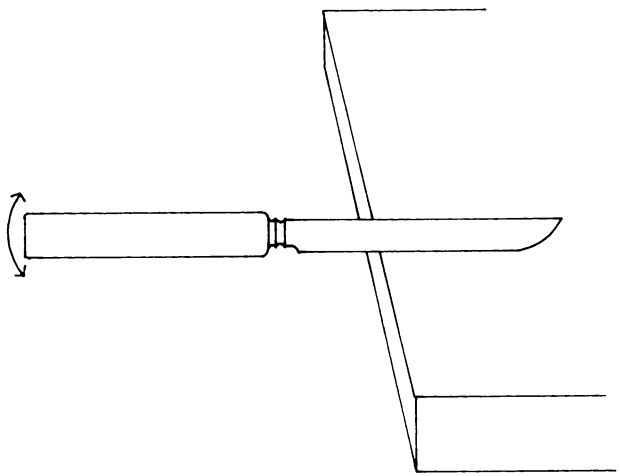
As sound is so familiar to us, an ever-present feature of our lives, we tend to take it for granted. What we are going to do here is 'put sound under the microscope' to reveal some of its secrets.

To make sound there has to be movement, in fact movement of a certain type. Try moving your arm backwards and forwards as fast as you can–you cannot hear anything (unless you happen to knock down your most precious vase!). That is because you cannot do it fast enough for sound to be produced.

Now take a knife (preferably one with a heavy handle and a long but not too sharp blade!) and place it on the edge of a table so that some of the blade is in firm contact with the surface and the handle is suspended in air.

Place the knife so that only a small amount of blade is held in contact with the table top. Holding the blade down firmly with one hand, twang the handle. You can see the knife vibrating, i.e. moving up and down about a central position, and hear it as a sound. This establishes an important fact – sound is one way in which moving energy is changed into another form. There can be no sound if no work is done, i.e. if no energy is put into the system.

Vibrating knife experiment



There are several other ideas related to sound which can be demonstrated by means of this simple but effective sound experiment. If you only put a small amount of energy into the knife blade it moves less and certainly sounds quieter. The volume of the sound heard is a function of the size or amplitude of the vibration. What may not be obvious is that the knife decides the 'note' it is going to produce. It would be more accurate to say that the knife vibrates at its natural frequency. The fact that bodies have a natural frequency at which they vibrate has many consequences, as we shall see later.

Try moving the blade in slightly and setting it in motion. This time you can see the vibrations are faster and can hear a higher note. By changing the amount of material free to vibrate, the natural frequency of the knife is changed. Generally the less material set in motion the faster will be the vibrations and the higher the pitch of the note heard. This can be shown very effectively by a second investigation. Take an empty glass bottle and tap the side fairly gently with a spoon. You will hear the characteristic bottle sound which is a consequence of the bottle vibrating. Now while repeatedly tapping the side of the bottle pour in some water and listen to the sound. As the amount of water inside the bottle increases the sound gets higher and higher. Can you explain why? The water in contact with the bottle stops the bottle vibrating so that as the bottle is filled up the amount of bottle vibrating progressively reduces and the pitch of the sound rises.

The medium

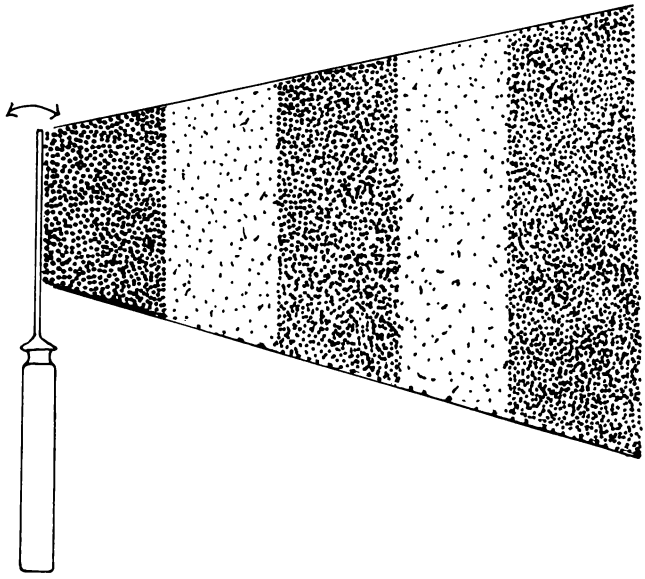
You may have heard the expression 'the silent world of space'. The silence is not due to the absence of noisy neighbours but to the total lack of any material (like the air) to vibrate. If someone bangs a gong in space it will certainly vibrate but there is no matter to convey the vibrations to any other place. This is one of the principal ways in which sound differs from light, and it might be interesting to consider the sun's light and sound in relation to our position on earth.

The light from the sun travels across the 93 million miles to the earth in about eight minutes. Now although space is silent, the sun is not! The sun is one gigantic thermonuclear explosion, and although it is the presence of the earth's atmosphere which protects us from too much dangerous radiation, it is the very absence of any vibrating medium between the sun and the earth's atmosphere which protects us from its sound. Indeed if there was air between the sun and earth we would all be killed by the noise!

So how does sound travel across air? We have already seen that there has to be movement of a body for sound to be produced. The way that sound is transferred to our ears is by the molecules which make up the air also being set in vibrational motion. For example, in our knife experiment, the air at the side of the blade was (fairly rapidly) pushed and pulled. The air on each side of the blade was repeatedly com-

pressed and then released (rarefaction). This is best shown in a diagram.

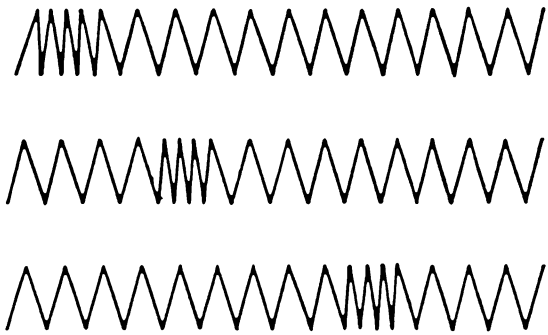
Longitudinal waves



As the blade moves from side to side a set of compressions and rarefactions is produced which together comprise a longitudinal wave. Molecules of air do not move from the knife to our ears but simply move backward and forward in sympathy with the knife. The membrane in our ears is set in further vibrational motion and so the sound can be heard.

Another way in which this idea can be illustrated is by considering a flexible spring between a sound source and your ear. If the spring is given a push at one end, it contracts and then expands. Note that although each part of the spring simply moves backwards and forwards, the wave is able to move along the complete length of the spring.

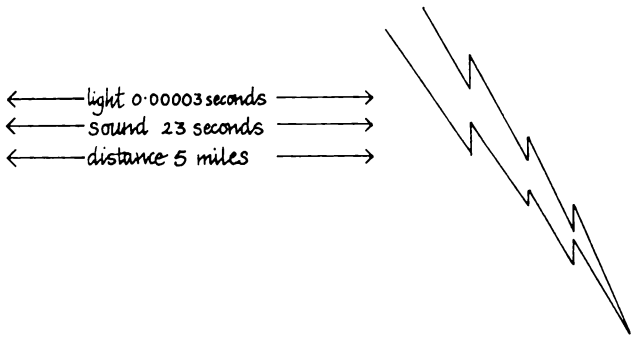
Wave motion in a spring



The fact that sound needs a medium has some strange consequences. One is that it is affected by the wind. If a sound is moving against the wind it meets resistance and will not travel so far. Conversely with a tail wind it can travel further. Another consequence is the Doppler effect which you hear when the source of the sound moves towards you. The sound rises in pitch because more vibrations are fitted into each interval of time.

A final consequence of sound needing a medium is that it takes some time to set molecules in motion. This means that the speed at which sound travels is relatively slow–about 345 metres per second. When you see a ball being struck some distance away there is a considerable gap between what your eyes see and the sound heard. This is because light travels at a speed of about 300 million metres per second, i.e. about a million times faster than sound. A more dramatic example is when there is an electrical discharge in the atmosphere. The lightning is often seen many seconds before the clap of thunder is heard, even though both started life together at the same instant.

The speed of light and sound



Feeling the vibes

Imagine pushing a swing. If your aim is to make it go higher and higher, you automatically give it a push at just the right point in its cycle and so at the same frequency as the swing. You have to keep in step with the natural frequency; it is only then that the amplitude of the swing will increase. When the source of energy is at just the right frequency to match the natural frequency being set in motion, the phenomenon is known as resonance.

There are several ways in which resonance can be experienced. Often car engines running at just the right speed set up noticeable vibrations in cars. Resonance has played an important part in the history of bridges. Soldiers marching across bridges have always been made to break step in case the rhythm of their marching happened to be at the natural frequency of the bridge. An awful resonance disaster was experienced at the Tacoma bridge in 1940 when the wind was sufficiently strong to set the bridge vibrating at its natural frequency and it vibrated to destruction.

We can feel the effect ourselves when in the presence of fairly loud music. You can actually 'feel' the beat of the music in your body as well as hear it in your ears because the natural frequency of the human body is quite low. A more delicate and rather beautiful example of

resonance can be demonstrated with a wine glass. By wetting your finger and lightly wiping the rim of a glass, you can 'excite' it to vibrate at its natural frequency; the resonance is a very pleasant effect. If you can produce a sound of sufficient strength at the natural frequency of the wine glass, then it may resonate to such a degree that the glass will shatter–a somewhat dramatic illustration of the power of sound.

Hearing sound

The environment is full of sounds, but not all of them impinge on us. For one thing much of the sound is beyond the range of our hearing, i.e. the frequency is too high or too low. One wonders if bats would be more or less frightening to people if we could hear their sounds! Even when a sound is within our audible range we have to interpret it. The ability to be selective in hearing is best illustrated by what is known as the 'cocktail party effect'. Although there may be a great buzz of conversation around us it is possible to eavesdrop on other people's conversations at will. Without this ability to focus, sound would be a cacophony. In fact sound becomes noise when we can no longer make sense of it or when it comes from an undesired source.

The sounds heard by different people vary considerably. For one thing the range of detectable frequencies can be quite different. However, the normal highest frequency detectable by human ears is in the region of 20000 cycles per second. The apparent loudness of sounds can also vary between individuals, and to make things worse this varies with frequency. So the sound pictures created within a crowd of people listening to some music may vary considerably from person to person.

Whereas we can shut off light which is too intense by simply closing our eyes or wearing dark glasses, our ears are more exposed. They are, however, very sensitive instruments. A single bar of an electric fire normally has a power of 1000 watts. If in sound energy terms one hundred-thousandth of this power were to fall on our ears we would be made deaf! In fact we are able to detect a sound level of one thousand-million-millionth of the power of an electric fire!

Although you may feel that your ears are in perfect shape, we feel we have a duty to inform you that you are suffering from presbycusis.

Do not be too alarmed as this simply refers to our gradual loss of ability to detect higher frequencies as we age. Be warned that too great a level of sound intensity or even a moderately high level of sound over a long period of time, whether caused by mechanical sounds or by pop music, can permanently damage your ears. Happily the Commodore 64 has only a moderate sound output and so will not harm them.

Sounding off

The way that humans use sound is mainly to convey words and music. However, sound is increasingly becoming a part of the modern world–as a side effect of developing technology. Many of these recent developments produce unpleasant sound: jet aircraft, traffic, road drills, etc. This brings to mind the pertinent observation 'a noisy noise annoys'. But there are also many new applications being found for sound, particularly outside the range of frequencies audible to the human ear. For example, we can now 'see' shoals of fish or faults in metal objects like oil pipes by using sound detection. It is even possible to use sound bleepers to attract fish. A particularly useful application for sound is in the ultrasound scanners that are now used to monitor foetal development in the womb and to detect gall stones. This certainly causes less tissue damage than using X rays, as sound waves have so much less energy.

2
Striking a Note

The way in which we can make the Commodore 64 play a note, let alone a tune, takes some explaining. Sound is controlled wholly by the use of the SID chip, which has 25 locations or addresses into which you can place values – different values affecting different aspects of the sound. For example, one location determines the volume at which notes or sounds are produced, another determines the type of sound that will be produced, and so on. These locations have addresses which run from 54272 to 54299. Values are placed in these locations by means of the POKE instruction. Typing

```
POKE 54296,15
```

puts the value 15 in the location whose address is 54296. In fact this particular address is the register for the volume and the values that can be set range from 0 for zero volume to 15 (maximum volume). This is separate from the volume control on your television; if you have set this at a minimum then nothing you do in a program will make a sound audible! The setting you control in register 54296 is the relative volume within the maximum possible for the setting of the volume control on your television.

The address 54296 is particularly useful when writing and experimenting with programs containing sound because if anything goes wrong and you need to press the STOP key, although the program will stop executing, quite often the sound will continue. Just typing

```
POKE 54296,0
```

will immediately switch off the sound output and your sanity will be restored!

As all the locations to be poked have such high addresses, it would

be useful to refer to them in terms of the address 54272. For example, the volume address would be 54272 + 24. In programs we shall store the 'base' address 54272 in the variable B, so that the volume register now becomes B + 24. This is a neat way of working and is useful because it will be easier to remember what each register does if it relates to a number between 0 and 24.

Having explained that the sound device is made to work by placing values in appropriate registers using POKE, we are now ready to make the chip play notes of different frequencies. Although there are three separate voices in the SID, only Voice 1 will be described here. In order to play a note of a particular pitch or frequency the SID chip requires values to be placed in two separate registers. For Voice 1 these are registers 54272 and 54273 and the values which can be poked into them must be in the range 0 to 255 inclusive.

To see how this works, type in Program 2.1 (you can ignore any spaces, we put them in to make the programs easier to read). Line 20 simply sets the base value for B at 54272 and sets the volume at a maximum. The two pokes in line 30 affect the type of sound heard and will be described in more detail later. The two loops at line 40 and line 50 constantly change the values H and L in the two registers which set the frequency of the note to be played. Line 90 makes sure that nothing is heard when the program is finished.

Program 2.1

```
10 REM 2.1  GLISSANDO
20 B=54272      : POKE B+24,15
30 POKEB+6,240 : POKE B+4 ,33
40 FOR H=20 TO  40
50 FOR L=0  TO 255 STEP 40
60 POKE B,L     : POKE B+1 ,H
70 NEXT
80 NEXT
90 POKE B+24,0 : END
```

When you run this program you will hear a note which starts at a particular frequency or pitch and automatically rises until it is eventually terminated. The STEP 40 in line 50 makes the rise in pitch occur at

a fairly fast rate. Try changing line 50 to

```
50 FOR L = 0 TO 255 STEP 10
```

and you will find that the note rises more slowly. Similarly, changing the step size to 128 using

```
50 FOR L = 0 TO 255 STEP 128
```

will make the note rise in pitch so fast that you will almost be able to hear the steps–it is no longer quite so smooth a rise.

If you change the values set in the loop in line 40, it is possible to extend the range of the notes played. The whole range of frequencies that can be used by the SID will be heard if line 40 is changed to

```
40 FOR H = 0 TO 255
```

When you now run the program it will take a long time to play the whole range of notes as the span is eight octaves.

High and low parts

We have seen how two separate pokes are needed to control the frequency of a single note, one into address B and the other into address B + 1. The SID chip uses the two values together to synthesise the note. In other words, although you may be placing the value L in address B and H in address B + 1, the sound chip will take these two values together to produce the single value L + H*256. The actual frequency of the note produced is related to the internal 'clock' within the computer – the frequency in cycles per second (which is also known as Hertz) can be found by dividing the above value, L + H*256 by 16.777.

To show how this works, add the line

```
65 PRINT "CLR" INT ((L + H*256)/16.777)
```

to your original program. You should now have Program 2.2:

Program 2.2

```
10 REM 2.2  FREQUENCY
20 B=54272      : POKE B+24,15
30 POKEB+6,240 : POKE B+4 ,33
40 FOR H=20 TO  40
50 FOR L=0  TO 255 STEP 40
60 POKE B,L     : POKE B+1 ,H
65 PRINT "␣"INT((L+H*256)/16.777)
70 NEXT
80 NEXT
90 POKE B+24,0 : END
```

When you run the program, the frequency will be shown at the top of the screen. The INT in line 65 simply rounds the actual frequency to a whole number so as to make it more pleasant to look at.

Notes on a scale

So far frequencies have been altered in a fairly continuous way, making the pitch of the note rise smoothly. When playing a musical instrument like the piano only particular notes and therefore frequencies are used. We shall now see how this can be achieved. Instead of poking numbers in addresses B and B + 1 to find what frequency is produced, we will normally want to perform the reverse process. If you want to play Middle C you will need to know what values to poke into the two addresses. One way to obtain these values is to look them up in Appendix 1. Thus to obtain Middle C (C4 in the table), you need to poke 37 into address B and 17 into address B + 1.

POKE B,37 : POKE B + 1,17

Instead of you doing the work of looking up the values, it is quite simple to get the computer to work them out. Type in Program 2.3. The value of F in the program is the frequency of Middle C. The way that this program works should be clear to you because it is quite similar to the previous one. The loop that runs between line 40 and line 80 changes the frequency by multiplying it by 1.06 . Line 50 arranges for

the two poke numbers to be calculated, and in line 60 these are poked into the appropriate frequency registers. Run the program and you will hear the twelve semitones into which Western music divides the octave.

Program 2.3

```
10 REM 2.3  HARMONIC SCALE
20 B=54272: F=262 : POKE B+24,15
30 POKE B+6,240   : POKE B+4 ,33
40 FOR N=0 TO 12  : G=F*16.777
50 H=INT(G/256)   : L=G-256*H
60 POKE B,L       : POKE B+1,H
70 FOR T=1 TO 200 : NEXT
80 F=F*1.06       : NEXT
90 POKE B+24,0    : END
```

As in the previous program, if you add the line

65 PRINT "CLR" INT(F)

you will see the frequencies of each note displayed. If you want to hear more notes then simply alter line 40. For example, changing it to

40 FOR N = 0 TO 24

will give a run of semitones over a range of two octaves.

Scaling heights

In general, when we measure anything we prefer linear scales. One example of this is in measuring temperature, where a one-degree interval at 20 degrees is exactly the same as a one-degree interval at 40 degrees. As we shall see, sound does not operate in such a simple manner.

Type in Program 2.4 and run it. Notice what happens as the pitch of the note increases. For the lower notes the changes sound like the

intervals between the notes on a piano. Later on the difference in pitch seems much less.

Program 2.4

```
10 REM 2.4  STEPPING TONES
20 B=54272      : POKE B+24,15
30 POKEB+6,240 : POKE B+4 ,33
40 FOR F=110 TO 880 STEP 10
50 G=F*16.777   : H=INT(G/256)
60 L=G-256*H    : POKE B,L
70 POKE B+1,H   : PRINT "␣"INT(F)
80 NEXT
90 POKE B+24,0 : END
```

If you look at the frequencies printed on the screen you can see that they increase at a steady rate, ten cycles per second. Moving from A2 (110 cycles per second) to A3 (220 cycles per second) requires a doubling of the frequency, which is a frequency change of 110 cycles per second. However, to move from A4, middle A (440 cycles per second) to the octave above A5 (880 cycles per second) also requires a doubling of the frequency, which is a frequency change of 440 cycles per second. This second change is four times the first. Although the frequency steps are equal, your ears will tell you that a ten cycle per second interval at low frequencies is larger than ten cycles at a higher frequency. Put another way, the time take to span the first octave is much less than the time taken to span the higher ones.

Sound warp

If you listen to a note which is constantly rising in pitch, you eventually reach a frequency–the Critical High Audio Point or CHAP–above which the note is too high for you to hear. The sound may still be present and your dog might actually be enjoying it. However, you perceive silence. Unfortunately you cannot test your CHAP using SID, as its highest frequency is about 4000 cycles per second whereas your CHAP is in the region of 20000 cycles per second.

The program which follows is designed to examine the opposite

effect–in other words to note the effect of decreasing the frequency of a note. What is interesting is the way that our brains interpret the signals received by our ears. Type in Program 2.5 and run it. It is very similar to Program 2.4 except that this time the frequency is falling, indeed until SID's lowest possible frequency value is reached. (The changes are in lines 40 and 75.)

Program 2.5

```
10 REM 2.5  V-T CUTOFF
20 B=54272      : POKE B+24,15
30 POKEB+6,240 : POKE B+4,33
40 FOR F=110 TO 0 STEP -1
50 G=F*16.777   : H=INT(G/256)
60 L=G-256*H    : POKE B,L
70 POKE B+1,H   : PRINT "␣"INT(F)
75 FOR I=1 TO 200 : NEXT
80 NEXT
90 POKE B+24,0 : END
```

Note the frequency at which the sound is no longer what you would call a 'note'. The only distinction between sounds below and above this point is that we interpret those above this critical value as continuous sound or notes, while those below are discrete sound, more like clicks. This cut-off point is called the V-T threshold.

3

The Musician's Aid

Tuning your musical instrument

As you saw earlier in Program 2.3, to play a particular note we need to provide the frequency and then, using calculated values of L and H, set SID's frequency register. So to tune a musical instrument we simply need to know the appropriate frequencies required and use them in a program to generate a note.

Program 3.1

```
10 REM 3.1  INSTRUMENT TUNER
20 B=54272      : POKE B+24,15
30 POKEB+6,240 : POKE B+4 ,33
40 PRINT " NEXT FREQUENCY ";
50 INPUT F      : IF F=0 THEN 90
60 G=F*16.777   : H=INT(G/256)
70 L=G-256*H    : POKE B,L
80 POKE B+1,H   : GOTO 40
90 POKE B+24,0 : END
```

Program 3.1 asks you to type in a particular frequency and then plays the appropriate note. To find out the notes for various musical instruments, see Appendix 2. If you are tuning a guitar you will need:

Note	Frequency
E2	82
A2	110
D3	147
G3	196
B3	247
E4	330

A metronome

This is a musical tool which makes a ticking sound at regular intervals. The Commodore 64 has an internal clock which starts as soon as you switch on the power.

A metronome is used by musicians for two main purposes. The constant tick can be set to agree with the tempo printed on the music so that we can play a piece of music at exactly the speed specified by the composer. Its other use, especially for someone learning to play an instrument, is to help him or her to keep time. Although music played out of tune is extremely painful to everyone around, when it is played with a varying tempo it can also very quickly lose its attraction!

Program 3.2

```
10 REM 3.2  METRONOME
20 R=60 : B=54272       : POKE B+1,50
30 POKE B+24,15         : POKE B+6,240
40 PRINT "␣"R           : Q=3600/R
50 T=TI : POKE B+4,33 : POKE B+4,32
60 IF TI-T<Q THEN 60
70 K=PEEK(197)          : IF K=64 THEN 50
80 IF K=21 THEN R=R+5 : GOTO 40
90 IF K=13 THEN R=R-5 : GOTO 40
100 IF K<>60 THEN 40
110 POKE B+24,0         : END
```

Program 3.2 sets a metronome ticking at 1 tick per second. By pressing F for faster and S for slower the speed of the ticks can be altered and this speed is shown on the screen in ticks per minute. So now you have no excuse for playing out of time!

Your own electronic organ

Now that you have tuned your musical instrument and have no excuses for not playing in time, we can reveal the Commodore in yet another role–as a keyboard instrument with a span of more than three octaves.

Program 3.3

```
10 REM 3.3  KEYBOARD
20 DIM F(64)
30 B=54272          : FOR J=0 TO 24
40 POKE B+J,0       : NEXT
50 POKE B+24,15     : FOR J=1 TO 42
60 READ F(J)        : NEXT
70 POKE B+6,165     : POKE B+4,33
80 J=PEEK(197)      : K=PEEK(653)
90 IF J=64 THEN POKE B+4,32    : GOTO 80
100 F=F(J)*2^K      : G=F*16.777
110 H=INT(G/256)    : L=G-256*H
120 POKE B,L        : POKE B+1,H
130 POKE B+4,33     : GOTO 80
1000 DATA ,,,,,,,,277.2,261.6,,,293.7
1010 DATA 311.1,,,,329.6,,,349.2,370.0
1020 DATA ,,415.3,392.0,,,440.0,466.2,,
1030 DATA ,493.9,,,523.3,554.4,,,,587.3
```

Type in Program 3.3 . When you run it you will find that the computer keyboard is ready to be played as a musical keyboard, just like an organ. The keys used for notes in the scale are as follows; the ordinary keys

```
  W   E       T   Y   U
A   S   D   F   G   H   J   K
```

have been converted to act like the standard notes on an organ or piano type keyboard, running through an octave:

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

Try playing with the keyboard notes to get the feel of the instrument–it has a better sound quality than many smaller single function organs that you can buy. To obtain all three octaves you need to hold down a second key while you are playing the organ. Holding the SHIFT key down moves all the notes up one octave, while holding down the Commodore key moves all the notes up a second octave. It should not take you long to become quite skilled in using the computer as an organ.

You will soon discover that we have built in a bonus feature. If you press the keys for any length of time, the sound produced is of an electronic organ, but if you quickly tap the keys, you will hear the computer sound more like a banjo!

If you are feeling adventurous you might like to try changing the quality of the sound that the organ is producing. Chapter 9 will give you some ideas on how this can be done.

The ways in which the Commodore has been put to musical use here are very different from the use of the computer to perform calculations. How you feel about your Commodore has probably changed quite a lot now that you have heard its voice.

4

Play a Tune

Now that we know how to play a note we can widen our horizons and play a tune. The aspect we have not had to concern ourselves with yet is time, and in this context there are two points to be considered. One is the length of time each note will be played relative to others, and the other is the overall time taken to play the music, in other words the tempo. In a later chapter you will see how to play music with harmony using more than one of SID's voices at a time but for now we shall look at the simpler problem–playing a tune.

Whatever tune you decide to play, you will have to identify its separate notes. If you simply want to play the scale of C Major then the notes will be

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

Looking at the table of musical frequencies printed in Appendix 1, you will see that you have a choice of octaves. If you decide to start at Middle C, i.e. the fourth octave, the note is labelled C4 and the set of frequencies are:

```
C4    D4    E4    F4    G4    A4    B4    C5
262   294   330   349   392   440   494   523
```

Now we need to be able to play each of these notes in turn. This does not require the frequencies to be stored permanently although that is a useful programming technique. As we are going to play each note in turn we only need to deal with the frequency of one note at a time. The simple way to do this is to READ in each value from a DATA list as it is required. Program 4.1 shows how this can be done.

Program 4.1

```
10 REM 4.1  C MAJOR SCALE
20 B=54272          : POKE B+24,15
30 POKE B+5,85      : POKE B+6 ,165
40 READ F           : IF F<0 THEN STOP
50 G=F*16.777       : H=INT(G/256)
60 L=G-256*H        : POKE B,L
70 POKE B+1,H       : POKE B+4,33
80 FOR T=0 TO 200   : NEXT
90 POKE B+4,32      : FOR T=0 TO 30
100 NEXT            : GOTO 40
1000 DATA 262,294,330,349,392,440
1010 DATA 494,523,-1
```

The first two lines simply put certain values in SID's registers to set the volume level, and so on. When the program executes line 40 the first value is taken from the DATA list. As this value, 262, is the frequency of a note to be played, it must be converted into the two values L and H which will be poked into registers 0 and 1 respectively. Line 80 is a rather strange device because, unlike most computing, it is actually designed to slow things down. It is called a dummy loop because it does precisely nothing 200 times; however, it takes time to repeatedly do nothing and it is this device we shall use to control the duration of each note. If instead of 200 in the loop we put 1000, each note will play for a significantly longer period.

After the first note has been played the program obeys the GOTO and so reads the next note from the DATA list. If you are not familiar with the DATA statement, think of having a marker pointing at the list; after each READ instruction the pointer moves one place down the line. You may have been wondering what happens when the program runs out of data. The answer is not that the computer starts making up its own tune, but that it prints

OUT OF DATA AT LINE 40

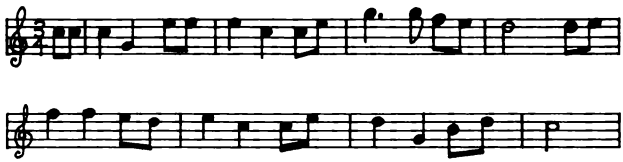
on the screen. This is not a pleasant way to end a tune, even the fairly dull C Major scale. It is for this reason that a 'flag' is placed in the data,

i.e. a value which the program will recognise as an inappropriate frequency. In our program we have used −1 to signal the end of the tune. Line 40 also checks each frequency and, if −1 is encountered, it stops the music.

Storing a tune

As we have already mentioned, if you are going to play a tune, or rather program the computer to play a tune, then you must first find out what notes are to be played. The easiest way is to find the music already written either in stave notation or the simpler C4 , D4... notation. As an example we shall use the following cheerful melody:

Clementine



Our first task is to identify each note in terms of its name and which octave it is in. For this tune we would record the following sequence:

```
C5 C5 C5 G4 E5 E5 E5 C5 C5 E5 G5 G5 F5 E5 D5

D5 E5 F5 F5 E5 D5 E5 C5 C5 E5 D5 G4 B4 D5 C5
```

Now if you look at the music you can see that, unlike the earlier scale, not all the notes are of equal length. If you are not very familiar with music and its notations, the table will give you a simple guide to note lengths.

Note lengths

Note	Note name	Musical name	Time code
𝅝	Whole	Semibreve	16
𝅗𝅥.	Dotted half	Dotted minim	12
𝅗𝅥	Half	Minim	8
♩.	Dotted quarter	Dotted crotchet	6
♩	Quarter	Crotchet	4
♪.	Dotted eighth	Dotted quaver	3
♪	Eighth	Quaver	2
𝅘𝅥𝅯	Sixteenth	Semiquaver	1

The times for the different types of note are very simply related to each other so if we let the 1/16th note have a duration of 1 unit then all the other notes can be written as simple multiples of it.

In coding the tune printed earlier, we now have to look at the music, note by note, and write down the time codes for each note. The first note is an 1/8th note so we write 2 and so on. The complete list of time codes for the tune is as follows:

```
2 2 4 4 2 2 4 4 2 2 6 2 2 2 8 2

2 4 4 2 2 4 4 2 2 4 4 2 2 8 -1 -1
```

We now need to co-ordinate the two aspects of each note, the frequency and the time code. As a check, you should count up how many notes there are in the music and compare them with the number of items you have in your note list and the list of time codes. If they do not agree, it is easier to detect and correct any errors at this stage rather than when all the information is in the computer and the tune sounds bad. Program 4.2 shows how the complete tune may be coded in Basic.

Program 4.2

```
10 REM 4.2  TUNE
20 B=54272      : POKE B+24,15
30 POKE B+5,34 : POKE B+6,165
40 READ F,T     : IF F<0 THEN STOP
50 G=F*16.777  : H=INT(G/256)
60 L=G-256*H    : POKE B,L
70 POKE B+1,H  : POKE B+4,33
80 FOR T1=0 TO T*100  : NEXT
90 POKE B+4,32 : FOR T1=0 TO 30
100 NEXT          : GOTO 40
1000 DATA 523,2,523,2,523,4,392,4
1010 DATA 659,2,659,2,659,4,523,4
1020 DATA 523,2,659,2,784,6,784,1
1030 DATA 698,2,659,2,587,8,587,2
1040 DATA 659,2,698,4,698,4,659,2
1050 DATA 587,2,659,4,523,4,523,2
1060 DATA 659,2,587,4,392,4,494,2
1070 DATA 587,2,523,12,-1,-1
```

Tempo

When you play a tune, the time codes have been used to set the relative times of duration of the separate notes. However, the tune can sound quite ridiculous if it is played at the wrong speed. Just consider how music sounds if you play a record at the wrong speed; it may make an amusing sound but it is certainly not musical!

The overall control of musical timing, known as the tempo, can be set by changing the constant number 100 which multiplies T in line 80 of Program 4.2. Larger numbers slow down the tempo.

This is most conveniently done by changing the constant 100 to a variable, TEMPO, which has to be set earlier in the program. Program 4.3 shows how this can be done.

Program 4.3

```
10 REM 4.3  TEMPO
20 B=54272      : POKE B+24,15
30 POKE B+5,34 : POKE B+6,165
35 TEMPO=25
40 READ F,T     : IF F<0 THEN STOP
50 G=F*16.777   : H=INT(G/256)
60 L=G-256*H    : POKE B,L
70 POKE B+1,H   : POKE B+4,33
80 FOR T1=0 TO T*TEMPO: NEXT
90 POKE B+4,32 : FOR T1=0 TO 30
100 NEXT          : GOTO 40
1000 DATA 523,2,523,2,523,4,392,4
1010 DATA 659,2,659,2,659,4,523,4
1020 DATA 523,2,659,2,784,6,784,1
1030 DATA 698,2,659,2,587,8,587,2
1040 DATA 659,2,698,4,698,4,659,2
1050 DATA 587,2,659,4,523,4,523,2
1060 DATA 659,2,587,4,392,4,494,2
1070 DATA 587,2,523,12,-1,-1
```

Playing with feeling

Depending upon your taste in music, you may or may not like the style in which some music is being played. If you play the tune on a standard musical instrument or even using the keyboard set up in Program 3.3, you can add the 'je ne sais quoi', the something extra which would make your rendition unique. Indeed, when playing a melody a second time you can change the way it sounds. One way of describing this is to say that you can play the music 'with feeling'. Now feeling, as you know, is one characteristic that computers do not have! It is not surprising that when they play, thay sound somewhat mechanical.

Some of the features we almost subconsciously introduce into our playing include accentuating the 'beat', i.e. placing slightly more

emphasis on the first beat of the bar, crescendo and diminuendo (getting louder and softer) and accelerando and rallentando (getting slightly faster and slower) during the playing of sections of the music. While playing we are also listening, and using this feedback affects how we continue to play–we are able to look not just at notes being played one at a time but at phrases within the music and indeed the piece as a whole. So if you want to hear a piece of music played tenerezza (tenderly), you will have to play it yourself.

Have a rest

In music particularly when there is more than one part being played, like a symphony for example, there are quite often rests when no notes are being played by certain instruments. This is not for trumpet players to get their wind back nor to prevent violinists getting cramp! If you want to have SID take a break between two notes, the simplest way is to get it to play a note of frequency 0 for however long the rest is required to last.

Play it again

It is possible that we may want to play a tune over again. Obviously we could type RUN to repeat the program, but this is unnecessary as we can build into our program a means of repeating the tune as often as we like or indeed forever! What we have to do is to have the program read the DATA list from the start again. This can be done simply by using the command RESTORE. The effect of RESTORE is to move the pointer which we described earlier back to the start of the DATA list. So if we amend line 40 of Program 4.3 to

```
40  READ F,T : IF F < 0 THEN RESTORE : GOTO 40
```

when we RUN the program the tune will repeat until you tire of it and take some action. Even the sweetest melody cannot endure forever!

5
Two Notes are Better than One

The colourful world of sound

By now you should have a good idea of what is meant by frequency and its relationship to the pitch of the sound heard when a note is played. Now we shall look at the situation when two notes are played together.

We shall start by comparing the playing of two notes with the effects that can be obtained with two colours. When two colours are placed side by side they 'interact' and introduce a third feature, the boundary. Particular choices of colours can either harmonise together or clash; one side effect of the interaction is that the boundary between them can either be strong or weak. Usually it is considered that shades of red clash when placed adjacent to each other while all shades of blue tend to harmonise, although some of the feelings we have about colours are personal and can vary.

The sound feature closest to the colour situation described above is when one note is followed by another. Once again the two elements give rise to a third, this time called the interval. The first note heard is used to set a 'base' by which the next note is measured. Although for two notes played from within the same scale there is no strong effect that could be called a clash, some intervals certainly have a particular feel.

The octave is recognised as a closely related pair of sounds and other simple intervals like the interval from C to G, or a fifth as it is usually known (because it spans five notes) are normally considered to have a pleasant feel to them. However, although much of sound is related to individual feeling, most people consider a rising interval to be more

exciting. It generates a feeling that something is about to happen, whereas a falling interval is more relaxing.

Perhaps you would like to try out these ideas for yourself. You could experiment with intervals on your own musical instrument, but we have provided Program 5.1 to help you.

Program 5.1

```
10 REM 5.1  RANDOM INTERVALS
20 B=54272        : POKE B+24,15
30 F1=440         : G=F1*16.777
40 H=INT(G/256)   : L=G-256*H
50 POKE B+0,L     : POKE B+1 ,H
60 POKE B+6,200   : POKE B+13,200
70 POKE B+4,33    : FOR I=1 TO 200 : NEXT
80 POKE B+4,32    : FOR I=1 TO  50 : NEXT
90 X=INT(1+RND(0)*8) :Y=INT(1+RND(0)*8)
100 F=F1*X/Y      : G=F*16.777
110 H=INT(G/256): L=G-256*H
120 POKE B+7, L : POKE B+8,H
130 POKE B+11,33: FOR I=1 TO 200 : NEXT
140 POKE B+11,32: FOR I=1 TO 900 : NEXT
150 IF PEEK(197)=64 THEN 70
160 POKE B+24,0 : END
```

When you run the program you will hear pairs of notes; the first note is always A4, but the second note is set randomly above or below the first note. Listen to several pairs of notes and decide for yourself if it is true that

1. The first note sets the character of the interval.
2. A rising interval has a different feel, for example it is more exciting, than a falling interval.
3. Some intervals, whether rising or falling, sound more pleasant than others.

When you have experimented sufficiently with the intervals, try these small modifications to the program.

Line 70 change 200 to 100
Line 130 change 200 to 100
Line 140 change 900 to 300

These alterations reduce the time taken for each note to play and also the time intervals between the pairs of notes. When you run the modified program the sound is very different and quite an enjoyable experience. This is the kind of music a switched on computer likes to play!

We shall follow the comparison with colour further. You know that when two colours of paint are mixed they both disappear and all we see is a single colour. So yellow paint mixed with blue produces what 'looks like' green paint; when they have been well mixed the original constituents can no longer be identified. The situation has changed from two colours to one.

The sound equivalent of colour blending is when two notes are played together; the picture this time has a much more complex web of interaction. Many more things happen with two sounds than with two colours. One obvious effect is that the two source sounds give rise to a new feature, the chord, which to a certain extent has a life of its own. One thing that makes sound different from colour is that when we hear a chord we are also able, particularly with some training and practice, to identify the separate notes as well.

Unlike single notes heard in isolation, where no note or frequency sounds particularly different from any other, every chord of two notes has a 'feel' to it. Certain chords are considered to be harmonious while others are considered less pleasant. It is worth considering whether this is just a matter of taste or whether it has a physical basis.

The found chord

We would probably all agree that the most harmonious chord, although perhaps not the most interesting, is the octave. As we already know, there is a simple relationship between the two frequencies spanning the octave: one is twice the other. So from the A in the fourth octave, which has been set at an internationally agreed frequency of 440 cycles per second, the A in the next octave up has a

frequency of 880 cycles per second, exactly double. It turns out that the pleasant sounding two-note chords all have simple relationships between the frequencies of the constituent notes. To make this clearer, the frequencies of all the semitones in an octave of the scale of A are shown here:

A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A
440	466	494	523	554	587	622	659	698	740	784	831	880

Imagine a chord made of A4 and a note within the octave above. Now if you multiply the frequency of the lower note, 440, by simple ratios or fractions, it can be seen that the most harmonious chords are those obtained when the two frequencies are related in the simplest numerical way. As the numbers in the multiplying fraction get larger the chord progressively becomes more discordant. The sequence of fractions you need to multiply by is 2/1 for the octave, 3/2 for the fifth, and so on. The complete set of multipliers and the named interval are shown in the chord table below. The order of increasing disharmony is:

2/1 3/2 4/3 5/4 5/3 6/5 8/5

You will find the numbers obtained by multiplying 440 by each of these fractions do not always agree exactly with the printed frequency. This is not due to an error in the printed values or in your calculator! The truth is that the frequencies fixed in the internationally agreed scale, the so-called equitempered scale, are just slightly different from what they should be for perfect matches–where Pythagoras wanted to put them!

Chords

Note	Frequency	Ratio	Multiplier
A4	440	1:1	1
A4#	466		
B4	494		
C5	523	6:5	1.2
C5#	554	5:4	1.25

D5	587	4:3	1.33
D5#	622		
E5	659	3:2	1.5
F5	698	8:5	1.6
F5#	740	5:3	1.67
G5	784		
G5#	831		
A5	880	2:1	2

Enough of all this theory. The way to appreciate sound is to listen to it! Program 5.2 should now be typed in. Save the program, as you will need it later on in this chapter. When you run the program you will hear two notes being played; one has a frequency of 440 cycles per second, i.e. A4 or Concert A, and the other has a frequency of 880 cycles per second. The frequencies of the notes are shown on the screen. Notice that, as expected, the second note is one octave higher than A4, indeed it is A5.

Program 5.2

```
10 REM 5.2  CHORDS
20 B=54272          : POKEB+24,15
30 F=880 : F2=440 : DF=10
40 G=F*16.777 : H=INT(G/256) :L=G-256*H
50 POKE B+6,240     : POKE B+13,240
60 POKE B+4,33      : POKE B+11,33
70 POKE B  ,L       : POKE B+1 ,H
80 G=F*16.777 : H=INT(G/256) :L=G-256*H
90 POKE B+7,L       : POKE B+ 8,H
100 PRINT "␣VOICE 1 "F2" VOICE 2 "F
110 A=PEEK(197)     : IF A=64 THEN 110
120 IF A=30  THEN F=F+DF
130 IF A=18  THEN F=F-DF
140 IF A<>60 THEN 80
150 GET A$ : IF A$<>"" THEN 150
160 POKE B+24,0     : END
```

When you press the D key on the keyboard the frequency of the second note falls. Note the sounds of the changing chord as well as the two fundamental notes; as the chord changes it sometimes

sounds quite harmonious, while at other times it is quite discordant and even strident.

Without looking at the frequencies printed on the screen, try to find the particular point other than the octave where the chord is most pleasant. If you overshoot a particular chord then use the U key to increase the second frequency again. When you have 'frozen' the chord as exactly as your ears will permit, work out the ratio of the frequencies–it is very likely to be 3:2 or expressed as a simple decimal 1.5. If you find that the frequencies for the different chords are not those printed in the table, it is quite likely that your ears are correct! As described above, the frequencies in the table have been 'fixed'! Feel proud that Pythagoras would have agreed with your sense of harmony–you are in good company!

A fundamental question

If we consider a single note, say of frequency 220 cycles per second, then it may be called the fundamental or first harmonic. The frequency which is two times this fundamental frequency is called the second harmonic, obviously one octave above the fundamental. By multiplying the original frequency by 3, 4, 5, and so on, we can obtain the third, fourth and fifth harmonics. In a sense all the harmonics can be said to have grown from the root frequency, the fundamental. The sequence of frequencies described above is

Frequency	220	440	660	880	1100
Harmonic	1	2	3	4	5

Now if these frequencies are all played together, one can detect the lowest note and hear the family of harmonics as quite an agreeable sound. However, if the source of the fundamental frequency of 220 cycles per second is removed so that the family of frequencies played is

440 660 880 1100

a very surprising effect can be experienced. This effect, first noted by Fletcher in 1934, is that the missing frequency is recreated by the

brain – you can hear a note which is not being played. Quite mysterious!

Using the same program as before, Program 5.2, set the two frequencies at 440 and 660. Can you hear the extra note?

Getting the beat

If somebody playing a musical instrument is joined by a second person playing the same tune on the same type of instrument, then a third and so on, the difference that you hear is not simply that each note is louder. You hear a much richer sound, indeed the music appears to have depth. This is partly due to what is known as the chorus effect.

This effect is a consequence of slight variations in many of the different aspects of sound, including minor changes in frequency during the life of each note and phase differences. This means that although the sounds are vibrating at the same frequency, they are at a maximum and minimum of their cycles at different times. It is like watching two swings moving backwards and forwards both at the same rate, say 5 seconds for a complete swing, but with high points at different times. They have the same frequency but are out of phase.

When you actually listen to two notes being played together there is a fascinating interaction, which with further investigation leads to several surprises. Change line 30 in Program 5.2 so that it becomes Program 5.3.

Program 5.3

```
10 REM 5.3  BEATS
20 B=54272          : POKEB+24,15
30 F=440 : F2=440 : DF=1
40 G=F*16.777 : H=INT(G/256) :L=G-256*H
50 POKE B+6,240     : POKE B+13,240
60 POKE B+4,33      : POKE B+11,33
70 POKE B  ,L       : POKE B+1 ,H
80 G=F*16.777 : H=INT(G/256) :L=G-256*H
```

```
90 POKE B+7,L      : POKE B+ 8,H
100 PRINT "␣VOICE 1 "F2" VOICE 2 "F
110 A=PEEK(197)    : IF A=64 THEN 110
120 IF A=30  THEN F=F+DF
130 IF A=18  THEN F=F-DF
140 IF A<>60 THEN 80
150 GET A$ : IF A$<>"" THEN 150
160 POKE B+24,0    : END
```

When you run it you should hear the same note, A4, being played in two of SID's voices; they are playing in unison. If you now press the U key once, the frequency of the second voice will increase by 1 cycle per second. The frequencies of the two notes are displayed on the screen. Apart from the two notes being played you will also hear a slow but strong beat effect. This is really an extra note at a frequency which is the difference of the two notes producing it. As the two notes differ by 1 cycle per second, the pulses or beats will occur every second. If you have a stop watch, time 20 pulses to confirm this relationship.

Alternatively you can add the following lines to your program:

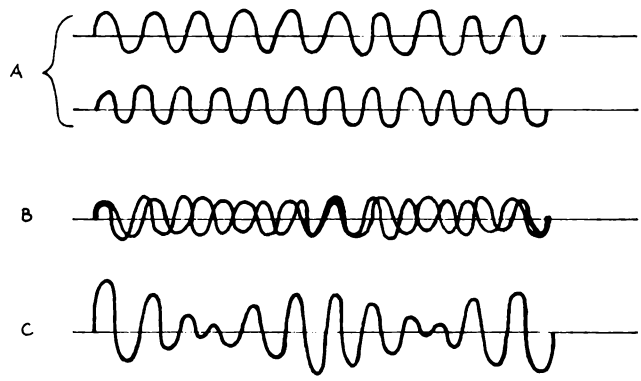
```
55 POKE B+20,245 : POKE B+15,100
91 POKE B+18,32
93 IF TI-T0<600 THEN 100
96 T0=TI : POKE B+18,33
```

Now when you run the program you will hear beeps as well as beats! The beeps occur every ten seconds. Count how many beats there are between two beeps so that, by dividing by 10, you can calculate the numbers of beats per second. You may need a little practice at counting the beats at first. Can you hear 10 beats in the 10 seconds? Make sure that the two frequencies shown on the screen are 440 and 441.

If you move the pitch of the second voice up further you will find the rate at which the beats sound will speed up. At certain stages as you move the frequency up you will find that the beats merge into the general sound. If you overshoot one of these places press D to move the frequency down again. At the point where the beats disappear you should note the frequencies and compare them with those in the table of frequencies. You should find that they coincide with certain

notes within the scale, notably with the special intervals mentioned earlier.

Beat waves



The interaction of two notes of slightly different frequencies is illustrated in the diagram. A shows two waveforms; in the same interval of time the top wave completes 10 cycles while the second completes 12 cycles. In B, the two waves have been superimposed–note that in some parts the two waves almost fit together, while in others they are almost in completely opposite positions. The last part of the diagram shows the overall effect of adding the two waves. Note how the amplitude varies much more slowly than the frequencies of the original waves; indeed there are two complete beat waves in the time interval shown, which is the difference between the numbers of cycles in the original two waves.

As you make the interval between the two notes larger you will notice that whenever there is what seems to us a pleasant sounding chord, a slight shift either side of the interval results in beats being produced. Even when the second frequency reaches close to 880 cycles per second, the complete octave, beats occur although they are not quite as pronounced as those heard at the start of this investigation.

It is interesting to note that the phenomenon of beats has been incorporated into a musical instrument. Organs of any reasonable level of sophistication are provided with a Vox Celeste which by means of two

pipes produces beats at a frequency of between two and eight cycles per second. This provides a pleasant undulating effect to complement the main organ sound. It is normally considered that beats at a frequency of between 15 and 20 cycles per second are unpleasant to the human ear. A second use to which beats are put is in tuning pianos.

The ghost effect

In the earlier investigation the beats were quite easy to hear, particularly for certain pairs of frequencies. The next investigation is more subtle and requires some concentration to hear the effect; however, it is quite surprising and somewhat mysterious. Change line 30 in Program 5.2 as follows to produce Program 5.4:

```
F = 440 : F2 = 440 : DF = 5
```

Program 5.4

```
10 REM 5.4  GHOSTS
20 B=54272          : POKEB+24,15
30 F=440 : F2=440 : DF=5
40 G=F*16.777 : H=INT(G/256) :L=G-256*H
50 POKE B+6,240     : POKE B+13,240
60 POKE B+4,33      : POKE B+11,33
70 POKE B  ,L       : POKE B+1 ,H
80 G=F*16.777 : H=INT(G/256) :L=G-256*H
90 POKE B+7,L       : POKE B+ 8,H
100 PRINT "␣VOICE 1 "F2" VOICE 2 "F
110 A=PEEK(197)     : IF A=64 THEN 110
120 IF A=30  THEN F=F+DF
130 IF A=18  THEN F=F-DF
140 IF A<>60 THEN 80
150 GET A$ : IF A$<>"" THEN 150
160 POKE B+24,0     : END
```

When you run this program you will notice that Voice 2 is now changing its frequency faster. Try to ignore both of the main notes as U is pressed. Keep your finger pressed down because the ghost effect is clearer when the interval between the notes is changing, rather as

some animals remain hidden as long as they keep still but can be seen as soon as they move.

The effect that you hear is made of two parts. In the lower registers you can detect a rise and fall of a very low note. More concentration will enable you to hear a rising and falling very high pitched note as well. This is quite eerie! However there is a simple explanation for where this note comes from. In the same way as an extra note is made from the difference of the frequencies which we called beats earlier, it is also possible to add the two frequencies to produce a third very high pitched note. So if Voice 1 has a frequency of 440 cycles per second and Voice 2 has a frequency of 880 cycles per second, then the high frequency will be at 880 + 440, i.e. 1320 cycles per second, which is about E6, much higher than either of the two original sources of sound.

Both beats and the ghost effect are a consequence of the two notes (which are really just vibrations of the air) affecting the same air. The molecules are under the influence of two sources of motion or vibration. If they are being pulled in the same direction then the move is exaggerated, if in opposite directions then the movements are reduced. Hence playing several notes together produces not just the basic frequencies but also a rich tapestry of sound.

6
The Sound of Music

Rich impurities

Imagine being blindfolded and then listening to the same note being played by a variety of different musical instruments–for example a violin, a trumpet, a drum, a saxophone, and so on. Each instrument can be immediately recognised by its individual and characteristic sound. Although the nominal frequency of the notes is the same for each instrument, something creates the different qualities of the instruments. Obviously this has quite a lot to do with the shapes of the instruments and the ways in which the sounds are produced, whether by objects being struck, strings vibrating, reeds vibrating, or simply air vibrating in a tube.

The question arises, 'If each note is really the air vibrating at the same rate, say 440 cycles per second, what is the physical difference between the sounds produced by the different instruments?' To put it another way, 'How does the ear analyse the separate sounds and know which instrument made them?' The answer lies not in the fundamental frequency played by each instrument, but in the hidden extras tagged along with the fundamental. These are the harmonics of the note.

The sound that a musical instrument produces is not only the fundamental frequency but also a set of overtones–these are the second, third, fourth and higher harmonics that were discussed in Chapter 5. The relationship between the set of harmonics and the set of overtones is simply in their numerical values. This should be made clearer by taking an example.

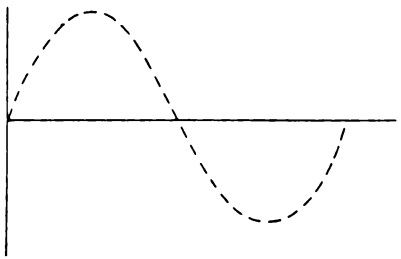
Frequency	440	880	1320	1760
Harmonic	1	2	3	4
Overtone		1	2	3

In relation to 440 cycles per second, the fundamental and nominal frequency of the note, the second harmonic with frequency 880 is also called the first overtone.

The quality or timbre of a musical note played on an instrument which distinguishes it from any other instrument is determined by the relative amounts of the various overtones present with the fundamental. What you hear when you listen to the note A played on a violin is a whole spectrum of different notes.

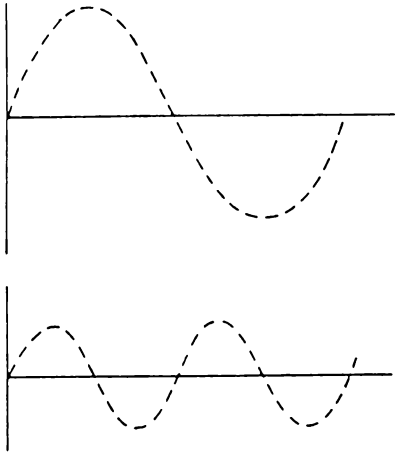
To hear a sound that does not have any overtones you will either have to obtain an audio oscillator or perhaps listen to a tuning fork. If you cannot get hold of either of these, try whistling! This type of sound is pure, with a sinusoidal wave form when viewed on an oscilloscope. See the diagram of a sine wave.

Sine wave



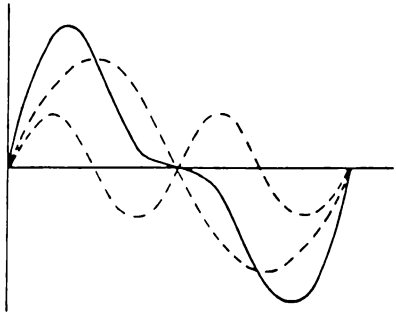
Musical instruments have the same basic fundamental smooth oscillation, but it is enhanced by overtones superimposed on the fundamental. The first overtone is also a sine wave, but of twice the frequency. This is shown clearly in the second diagram.

Fundamental with first overtone



Now these separate oscillations do not lead independent lives but interact together to form a more complex waveform. The third diagram shows how this happens.

Combination of two sine waves

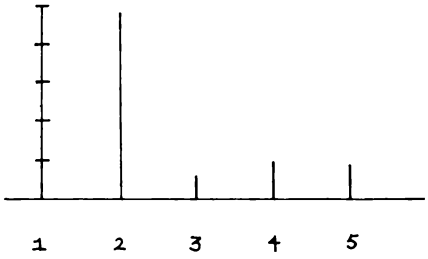


It is obvious that there are two features that determine the quality or tone of the sound heard when a musical instrument is played. These are whether particular overtones are present at all and, for those that

are present, the relative sizes of each contributing overtone. This may be made clearer by comparing the waveform structures of certain instruments.

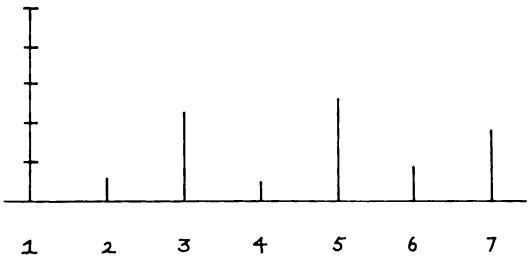
The flute has present in its sound the fundamental and second harmonics in similar proportions but with very small contributions from the higher harmonics.

Flute harmonics



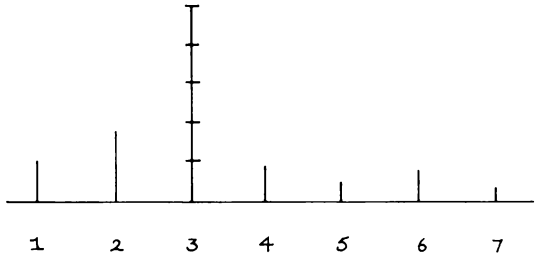
However, with the clarinet, after its principal first harmonic, in order of magnitude, come the fifth harmonic followed closely by the third and seventh harmonics.

Clarinet harmonics



What may be surprising is that for some instruments the harmonic contributing most to the overall waveform is not the fundamental frequency; for example the bassoon's waveform is dominated by the third harmonic, while the first harmonic (which is actually oscillating at the frequency being heard) is only one fifth as strong.

Bassoon harmonics



It is important to realise that there are other differences in the sounds that these instruments make, for example, the way their individual notes start their lives, whether abruptly when something is hit or more slowly as when a violin is bowed. This is caused by changes in volume and how quickly they occur. This important aspect of sound quality will be considered in Chapter 8.

SID's waveforms

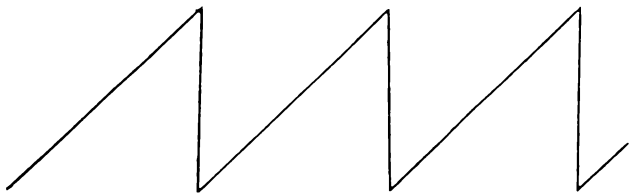
Now that we know that what determines the different qualities of notes is the mixture of their harmonics, we can test these ideas by considering certain recipes. Consider adding all the harmonics of a fundamental note together but in declining proportion:

Sawtooth harmonics

Harmonic	1	2	3	4	5	6	7
Proportion	1	1/2	1/3	1/4	1/5	1/6	1/7

Remember that all the harmonics are separate sine waves oscillating at different frequencies. The composite waveform obtained when they are mixed in the proportion shown in the table has a characteristic shape called the sawtooth.

Sawtooth waveform



The sound heard is usually described as quite 'rich'. This particular waveform is one of four available in SID's repertoire. When a value of 33 is poked into any of SID's waveform registers (4,11,18 for the three separate voices) it is the sawtooth sound which is heard.

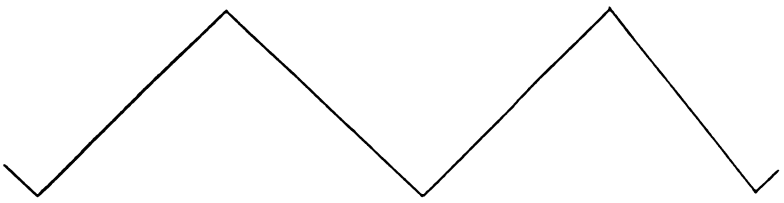
A second recipe for making sound is to take no even harmonics at all but let the odd harmonics decline much faster than before:

Triangle harmonics

Harmonic	1	2	3	4	5	6	7
Proportion	1	0	1/9	0	1/25	0	1/49

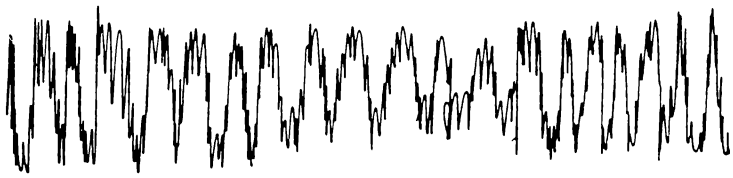
Put in words this means that the odd harmonics decline inversely as the square of the harmonic number. The net result of this mixture is the triangular waveform. This waveform is available when 17 is poked into any of SID's waveform registers 4,11 or 18.

Triangle waveform



Apart from these two waveforms, SID has two more. One is rectangular, and the other is called white noise.

White noise waveform



White noise is very different from musical instrument sounds in that instead of having a fundamental note accompanied by a set of harmonics, it has a complete mixture of frequencies. This waveform will be particularly useful for creating sound effects. If you want to hear this noise then you need to poke 129 into one of SID's waveform registers.

Type in Program 6.1. When you run it, you will hear the same tune being played over and over again. The waveform used to provide the sound is printed on the screen. The program is designed to let you hear different waveforms; pressing the Commodore key automatically changes the waveform while the tune continues.

Program 6.1

```
10 REM 6.1  WAVEFORMS
20 W=1                : TEMPO=200
30 B=54272            : POKE B+24,15
40 POKE B+5,34        : POKE B+6,165
50 W(1)=17            : W$(1)="TRIANGLE"
60 W(2)=33            : W$(2)="SAWTOOTH"
70 W(3)=129           : W$(3)="WHITE NOISE"
80 READ F,T
90 IF F<0 THEN RESTORE : GOTO 80
100 G=F*16.777        : H=INT(G/256)
110 L=G-256*H         : POKE B,L
120 POKE B+1,H        : POKE B+4,W(W)
```

```
130 FOR T1=0 TO T*TEMPO : NEXT
140 POKE B+4,W(W)-1
150 IF PEEK(653)=2 THEN W=W+1
160 IF W>3 THEN W=1
170 PRINT"␣"W$(W)
180 IF PEEK(197)<>60 THEN 80
190 POKE B+24,0   : END
1000 DATA 392,2,523,3,523,1,523,2
1010 DATA 659,2,587,3,523,1,587,2
1020 DATA 659,1,587,1,523,3,523,1
1030 DATA 659,2,784,2,880,6,880,2
1040 DATA 784,3,659,1,659,2,523,2
1050 DATA 587,3,523,1,587,2,659,1
1060 DATA 587,1,523,3,444,1,440,2
1070 DATA 392,2,523,4,-1,-1
```

When you have heard enough of this, press the space bar to terminate the program. If you hit the STOP key then it is possible that the last note played will continue to sound. To remove the sound simply enter

POKE 54296,0

and peace will return. In the next chapter we shall demonstrate a variety of ways in which we can use all of SID's voices to play music.

7
Variations on a Theme

The rule of seven

When we looked at how to store a tune in Program 4.2, we only had to concern ourselves with one voice. When more complicated music is to be played, it is necessary to use more than one of SID's voices. As you will see, this is quite straightforward as each of the voices is organised in exactly the same way. Each of SID's voices has seven registers; these have been arranged so that they are in exactly the same order for each voice. Hence the functions carried out by the first register in each voice will be the same.

In Voice 1, the registers used to create the appropriate frequencies for the note to be played were 0 and 1. It follows that the registers used for the setting of frequencies in Voice 2 and Voice 3 will be seven more than these registers, that is they will be registers 7 and 8 for Voice 2 and registers 14 and 15 for Voice 3.

In a similar way, the waveform registers for the three voices are spaced out by seven. As Voice 1 has its waveform set by register 4, it follows that Voice 2 uses register 11 and Voice 3 uses register 18 to set its waveform. The way that this register numbering is organised is called 'the rule of seven'. You will see it applied throughout the rest of the book.

Appendix 3 shows how all of SID's 29 registers are organised. You can see how the rule of seven operates by looking at the groupings within the first 21 registers in the table. The remaining registers, numbered 21 to 28, will be described in future chapters.

After the fairly heavy theory of waveforms in the last chapter, it is time to listen to some music, indeed music using all of SID's voices.

Just amazing

The first and simplest form of multi-part music to play is with the melody on one of SID's voices and constant notes on the other two voices. You may think that this is going to sound dull, after all the word monotonous derives from the Greek for single tone! However, although the music of bagpipes may be described in a variety of ways it is certainly not dull!

Program 7.1

```
10 REM 7.1  JUST AMAZING
20 B=54272          : POKE B+24,15
30 POKE B+5,34      : POKE B+6,165
40 POKE B+7,143     : POKE B+8,12
50 POKE B+13,112    : POKE B+20,112
60 POKE B+14,97     : POKE B+15,8
70 POKE B+11,33     : POKE B+18,33
80 READ F,T         : IF F<0 THEN 140
90 G=F*16.777       : H=INT(G/256)
100 L=G-256*H       : POKE B,L
110 POKE B+1,H      : POKE B+4,33
120 FOR T1=0 TO T*200 : NEXT
130 POKE B+4,32     : GOTO 80
140 POKE B+24,0     : END
1000 DATA 392,3,523,1,523,8,659,2
1010 DATA 587,1,523,1,659,8,587,4
1020 DATA 523,8,440,4,392,8,392,3
1030 DATA 523,1,523,8,659,2,587,1
1040 DATA 523,1,659,8,587,3,784,1
1050 DATA 784,8,659,3,784,1,784,8
1060 DATA 659,2,587,1,523,1,659,8
1070 DATA 587,4,523,8,440,4,392,8
1080 DATA 392,3,523,1,523,8,659,2
1090 DATA 587,1,525,1,659,8,587,4
1100 DATA 523,16,-1,-1
```

In Program 7.1, the drone is set in Voices 2 and 3 using frequencies 131 and 196 cycles per second. The melody is played in Voice 1. Try listening to what you will surely agree is an amazing tune!

Playing the bagpipe music is not really very different from playing the tune of Clementine which appeared in Chapter 4; it is just a tune being played while the other two voices are playing constant notes. In future musical excursions we will need a slightly different technique.

When you use READ to obtain values in DATA, there is only one pointer and you do not really have much control over it. It either continues to 'pick off' the note values one by one, or, by using RESTORE, you can set it pointing to the first item in the list.

Data list

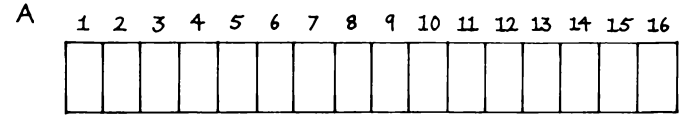
```
DATA 23,46,57,16,23
       ↑
     pointer
```

What we need now is a method of having a programmable pointer, whose position we can control within a program. In fact for the next demonstration of SID's musical talents we shall have several pointers!

First, a little theory. Instead of storing the note values in two variables F and T, which means that as each note is read from DATA, the old values of F and T are lost, what is required is to store each note permanently in variables within the program. This can be done using what are known as arrays.

In case you have not yet come across arrays, they will be described briefly here. Consider a street in which all the houses are numbered. If you want to identify any particular house, it is usually a very simple procedure–you simply give the street name and the number of the house. With arrays, the name of the variable is given followed, in brackets, by the number required.

An array

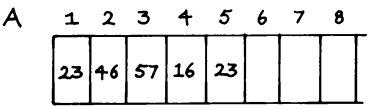


Suppose that the name of the array is 'A'; then A(1) usually refers to the first position in the table, A(2) the next and so on. The great advantage of using arrays to store data is that it can so efficiently be manipulated–you do not have to use the separate variable names A(1), A(2) and so on–they can all be accessed using another variable within the brackets. For example, the following piece of code:

```
10 FOR I = 1 TO 5
20 READ A(I)
30 NEXT I
40 DATA 23,46,57,16,23
```

has the variable I taking values 1, then 2 and so on. As the program goes round the loop each item from the DATA list is stored in a separate position in the array A. The net result of running this short program is to allocate values as shown below:

Storing values in array A



We shall now see how arrays can be used in playing some music.

Around a round

There are many songs created by having many voices singing exactly the same tune but starting at different times. For example, the second voice starts a certain number of bars after the first and then the third joins in after the same interval, and so on. One famous 'round' song

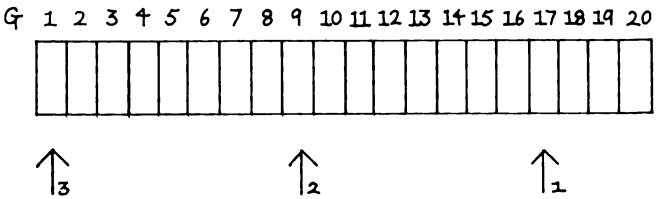
is 'London's Burning', but in Program 7.2 is found the equally famous 'Frere Jacques'.

Program 7.2

```
10 REM 7.2  A ROUND
20 DIM G(99),T(99): N=1
30 B =54272        : B2=B+7
40 B3=B+14         : READ TEMPO,NV
50 POKE B+24,15    : FOR V=1 TO NV
60 READ W(V),X(V)  : NEXT
70 POKE B +5,3     : POKE B +6, 53
80 POKE B2+5,79    : POKE B2+6,240
90 POKE B3+5,88    : POKE B3+6, 89
100 READ F,T(N)    : IF F<0 THEN 120
110 G(N)=16.777*F  : N=N+1 : GOTO 100
120 N=N-1
130 FOR V=1 TO NV  : Y=Y(V)
140 IF Y>0 THEN Y(V)=Y-1   : GOTO 210
150 P=B+7*(V-1)    : POKE P+4,W(V)-1
160 X=X(V)         : X=X+1
170 IF X>N THEN X=1
180 X(V)=X : Y(V)=T(X)-1 : G=G(X)
190 H=INT(G/256)   : L=G-256*H
200 POKE P,L       : POKE P+1,H
210 NEXT
220 POKE B+4,W(1)  : POKE B2+4,W(2)
230 POKE B3+4,W(3): FOR J=1 TO TEMPO
240 NEXT           : GOTO 130
1000 DATA  90,3,17,0,17,1,33,2
1015 DATA 0,16,0,16,0,16
1020 DATA 262,2,294,2,330,2,262,2
1030 DATA 262,2,294,2,330,2,262,2
1040 DATA 330,2,349,2,392,4
1050 DATA 330,2,349,2,392,4,392,1
1060 DATA 440,1,392,1,349,1,330,2
1070 DATA 262,2,392,1,440,1,392,1
1080 DATA 349,1,330,2,262,2,262,2
1090 DATA 196,2,262,4,262,2,196,2
1100 DATA 262,4,-1,-1
```

In this program you will see that a number of arrays have been used. The DIM statement in line 20 simply informs the computer that we shall want to use a certain amount of the available random access memory (RAM) for storing data. This has only been done for arrays which need more than 10 locations. Array W has been used, but it only requires 3 locations so that it has not been declared in line 20. The two arrays named G and T hold the note frequencies (after multiplying by 16.777 as always) and note durations for the tune. The array X stores the position of the pointers when the tune is played. As there are three musical parts to be dealt with at the same time, three pointers will be needed to keep track of where in the tune each voice is. The idea behind the method used is shown below.

Array G with three pointers



The diagram shows that the first pointer is at position 17, the second pointer is at position 9 while the third pointer is at position 1. So, referred to the pointer array X, this means that

X(1) = 17
X(2) = 9
X(3) = 1

where pointer 1 refers to the position in the song where Voice 1 is singing. Obviously, in playing the round, the three pointers will progress through the arrays in sequence.

In any round, the first voice starts while the other voices are silent. This has been arranged by providing a number of 'silent' notes for Voices 2 and 3 at the start.

The use of arrays is one of the most powerful, yet neat, tools the

programmer can use when wanting to store large amounts of information within a program. You will see the ideas of arrays developed further in the musical examples which follow.

Bach's byte

We now come to playing independent melodies using two of SID's three voices. As Bach is considered to be the premier composer of counterpoint it seems a good idea to use his music to illustrate the method. There are several ways in which the computer can be programmed to play a number of melodies simultaneously, some simple, some complex.

There is a maxim in professional computing that 'simplicity is elegance', so the simplest technique is illustrated in Program 7.3. The method used is simply an extension of the design used in Program 7.2, i.e. it uses arrays. The main difference is that as there are two independent voices to control, we now need each array to have two parameters; as they now have two parameters they are called two-dimensional arrays. The first parameter indicates which voice is being considered while the second identifies which note is being played.

Program 7.3

```
10 REM  7.3  BACH
20 DIM G(2,99),T(2,99)
30 B=54272           : B2=B+7
40 POKE B+24,15
50 FOR V=1 TO 2      : N=1
60 READ F,T(V,N)     : IF F<0 THEN 80
70 G(V,N)=16.777*F   : N=N+1 : GOTO 60
80 NEXT
90 POKE  B +5,0      : POKE B +6,240
100 POKE B2+5,0      : POKE B2+6,240
110 FOR V=1 TO 2     : Y=Y(V)
120 IF Y>0 THEN Y(V)=Y-1  : GOTO 180
130 P=B+7*(V-1)      : POKE P+4,32
140 X=X(V)           : X=X+1
150 IF X>78 GOTO 220
```

```
160 X(V)=X           : Y(V)=T(V,X)-1
170 G=G(V,X)         : H=INT(G/256)
180 L=G-256*H        : POKE P,L
190 POKE P+1,H       : NEXT
200 POKE B+4,33      : POKE B2+4,33
210 FOR J=1 TO 100 : NEXT : GOTO 110
220 POKE B+24,0      : END
1000 DATA 294,1,330,1,349,1,392,1
1010 DATA 440,2,587,2,554,2,440,2
1020 DATA 330,2,392,2,370,2,294,2
1030 DATA 523,6,494,1,440,1,494,2
1040 DATA 392,2,330,2,392,2,466,2
1050 DATA 294,2,277,2,440,2,294,2
1060 DATA 392,2,349,4,330,1,349,1
1070 DATA 330,1,349,1,294,1,277,1
1080 DATA 294,1,330,1,370,1,415,1
1090 DATA 440,1,494,1,440,1,494,1
1100 DATA 523,1,587,1,659,2,880,2
1110 DATA 831,2,659,2,494,2,587,2
1120 DATA 554,2,440,2,784,6,740,1
1130 DATA 659,1,740,2,587,2,494,2
1140 DATA 587,2,698,2,440,2,415,2
1150 DATA 659,2,440,2,587,2,523,4
1160 DATA 494,1,523,1,494,1,523,1
1170 DATA 440,1,415,1,440,1,494,1
1180 DATA 523,1,587,1,659,1,698,1
1190 DATA 784,10,-1,-1
1200 DATA 0,16,0,16,0,16,0,16
1210 DATA 523,1,494,1,440,1,494,1
1220 DATA 523,1,494,1,523,1,587,1
1230 DATA 659,9,587,1,554,1,494,1
1240 DATA 554,1,440,1,494,1,554,1
1250 DATA 587,9,523,1,494,1,523,1
1260 DATA 587,1,659,1,698,1,587,1
1270 DATA 659,1,698,1,659,1,587,1
1280 DATA 523,1,494,1,440,1,415,1
1290 DATA 440,2,880,4,831,2,880,2
1300 DATA 440,1,415,1,440,1,494,1
1310 DATA 523,1,440,1,392,10,-1,-1
```

What have they done to my song?

We apologise in advance for this section and warn that it is not for the musically squeamish! We saw earlier how it is possible to use arrays to store notes and time codes; this was introduced so that by using pointers we could access notes in a variety of positions. In this section we are going to develop another application for this idea.

Suppose that instead of moving the array pointers forward, as we have been doing so far, we move them backwards–can you imagine what it will sound like? In Program 7.4 you will find we have again coded up the song 'Clementine'.

Program 7.4

```
10 REM 7.4  REVERSE
20 DIM F(50),T(50)   : B=54272
30 I=0               : POKE B+24,15
40 POKE B+5,105      : POKE B+1,161
50 N=N+1             : READ F(N),T(N)
60 IF F(N)>0 THEN 50
70 I=1               : N=N-1 : DI=1
80 F=F(I)            : T=T(I)
90 G=F*16.777        : H=INT(G/256)
100 L=G-256*H        : POKE B,L
110 POKE B+1,H       : POKE B+4,33
120 FOR T1=0 TO T*75
130 NEXT             : POKE B+4,32
140 FOR T1=0 TO 30   : NEXT
150 IF PEEK(653)=2 THEN DI=-DI
160 I=I+DI
170 IF I<1 THEN I=N
180 IF I>N THEN I=1
190 GOTO 80
1000 DATA 523,2,523,2,523,4,392,4
1010 DATA 659,2,659,2,659,4,523,4
1020 DATA 523,2,659,2,784,6,784,1
```

```
1030 DATA 698,2,659,2,587,8,587,2
1040 DATA 659,2,698,4,698,4,659,2
1050 DATA 587,2,659,4,523,4,523,2
1060 DATA 659,2,587,4,392,4,494,2
1070 DATA 587,2,523,12,-1,-1
```

When you run the program you can arrange for it to play the tune in reverse simply by pressing the Commodore key. How you feel about this new sound will depend on your own taste! However, simply press the Commodore key again and normal service will be resumed. Notice how Clementine sounds quite pleasant, although different, even when played backwards.

The last three chapters have been full of challenging ideas. We now know that any musical instrument is never just playing a note but is also producing the oscillations for a whole range of sounds. We have also seen how playing two notes together produces new entities which have their own lives and characteristics. Do not be afraid of the ghost effect or the beats, they simply point to the startling array of phenomena associated with the world of sound.

What is perhaps most surprising about sound is that when we sit in a concert hall and listen to about fifty instruments playing a collection of different tunes we can not only make some sense out of it, but can use our intellect to convert it into a pleasant and enriching experience.

8

Picture a Sound

When we hear sounds, whether they are musical or from the natural world, we tend to take a lot for granted; often they just form a background to our lives. This can be attributed to a variety of causes. One is that our visual sense seems to be significantly more sensitive than our hearing, while another is that apart from the most dull monotonic note, sound is constantly changing.

When you look at a garden you obtain an immediate overview of the amount of trees, shrubs, plants and lawn in it, whether it is well cared for and so on; however you can focus in on particular parts and 'see', for example, whether the fruit on a particular branch of a tree looks ripe, whether a shrub is leaning over and needs more support and so on. You can control what you want to see by zooming in or panning around. Sound cannot generally be 'frozen' in time; its whole meaning comes from its changing quality as time progresses.

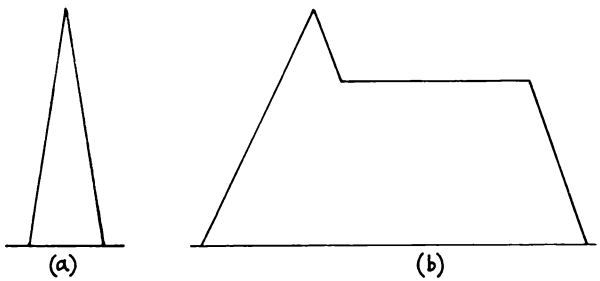
A second difference with sound is that there is frequently a variety of changing effects present at any one time. If you are listening to music then it is perhaps intentional that you listen for the complete effect, but if you try to listen to just one part of the overall sound, say the second violins in an orchestra, it is not a particularly easy task, although of course this skill can be developed.

We have started this chapter in this way in order to invite you to focus in on sound, to listen not for the total effect but to hear the constituent parts into which it divides. You may like to go around your house and actually produce some sounds, but it will probably suffice to do the following 'thinking experiment'.

Compare what happens and what you hear if someone strikes an object, say a tin can or a glass, with the sound you hear when someone blows a trumpet. To make life easier, and in a thinking experiment this can easily be arranged, we will assume that the two sounds

have exactly the same pitch. The sound from the struck object rises in volume from zero to peak volume almost in an instant and also fairly quickly fades away again. The trumpet has a slower build up in volume (although it is still quite fast), a peak is reached which then settles at a slightly lower volume as the note is sustained at a fairly constant level. Eventually the note will stop (for the player to take a breath!) and the sound will fall off quite quickly as the pressure is released. This comparison is best illustrated by a diagram showing the amplitudes of the sounds from the two sources.

Sound pictures (a) tin (b) trumpet



The very different qualities of these two sounds can be partially explained by the harmonic content of their waveforms, which we described in Chapter 6. However, the amplitude envelopes shown above are also a major constituent in our perception and interpretation of the sounds. Whether we are mimicking the instruments of the orchestra or making sound effects using the computer, the control of the amplitude envelope will be crucial.

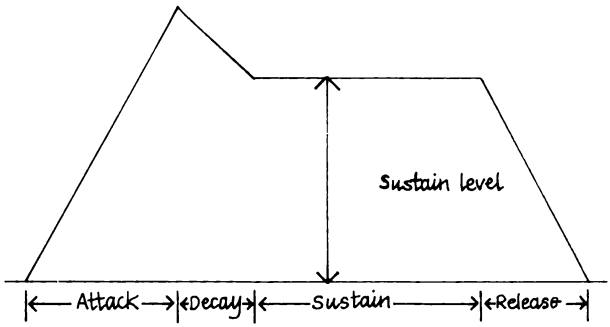
Enveloping the sound

The amplitude cycle for any sound can be divided into four distinct parts, although for some sounds certain of these may be absent. The first is the attack phase, in which the sound rises from zero to peak volume; the quicker this occurs the 'sharper' the sound will be. The second is the decay phase in which the sound falls back from the peak to some intermediate level which is maintained during the third phase, the sustain phase. Although the attack and decay phase typically only

last for a few milliseconds (each millisecond is one thousandth of a second so they occur fairly quickly!), the sound is sustained for however long energy continues to be provided. For musical instruments this will be for however long the trumpeter blows, the violinist bows or the key is held down on an organ. Finally, when the sustain phase is over, the release phase, in which the sound falls away to nothing, occurs. Note that in the tin tapping activity there was a fast attack followed by decay of the sound back to zero amplitude; hence there was a negligible sustain element in this sound cycle.

The complete cycle, Attack, Decay, Sustain and Release will be used frequently in the rest of this book and will be known as the ADSR cycle. It is particularly important in producing sound effects on the computer.

Amplitude phases



SID's envelope

We have already seen how poking various addresses sets appropriate values in some of SID's registers to make a sound. We are now going to use more of its registers to create the ADSR envelope for our sounds.

You might think that as there are four parts to the ADSR envelope, there will be four separate registers to be poked. However, they have been paired up so that for Voice 1 register 5 (poked at 54277) controls the attack and decay parts of the cycle and register 6 (poked at

54278) controls the sustain and release phases. Note that although the attack, decay and release settings for a sound's life are controlled by times, the sustain setting is a measure of level; this means that it determines at what fraction of the peak volume to keep the amplitude set. As will be explained later, the time that the sound is kept at the sustain level will be controlled by the Basic program–it has nothing to do with the SID chip itself.

Each of the A,D,S,R components of the sound envelope can be given a value in the range 0 to 15 inclusive. The effect each value has for A,D and R relates to times taken to complete that phase:

ADSR timings

Setting	Attack rate (msecs)	Decay & Release (msecs)
0	2	6
1	8	24
2	16	48
3	24	72
4	38	114
5	56	168
6	68	204
7	80	240
8	100	300
9	250	750
10	500	1.5 sec
11	800	2.4 sec
12	1 sec	3 sec
13	3 sec	9 sec
14	5 sec	15 sec
15	8 sec	24 sec

So sound with ADSR parameters

Attack,A = 14 Decay,D = 12 Sustain,S = 7 Release,R = 11

will take 5 seconds to rise to its peak and 3 seconds to fall to a level about half the peak. Finally when the sound is released it will take 2.4 seconds to fall from the sustained level to zero.

The way the various components of the amplitude cycle interact can be quite complex. For example, if the sustain value S is set at 15, there can be no actual decay as the decay setting D is a measure of the time taken to fall from the peak amplitude to whatever level has been set by S. Similarly, if S = 0, the note will be sustained at zero amplitude so that when it is ended, the release cycle will not be audible, whatever the setting for R.

Setting ADSR

If you have been wondering how it is possible to store the four values of A,D,S,R within just two locations, the mystery will now be explained. The value stored in register 5 will be 16 times the A value plus the D value. Similarly, the value stored in register 6 will be 16 times the S value plus the value of R; the computer will unscramble this coding. So the way we could arrange for the ADSR setting described earlier would be by

```
POKE B+5,  14*16+12
POKE B+6,   7*16+11
```

Generally in programs we shall write this as

```
POKE B+5, A*16+D
POKE B+6, S*16+R
```

where the variables A,D,S and R will take particular values.

If you set the two registers 5 and 6 by these two pokes you will unfortunately hear nothing! For one thing, the overall volume control for the chip has not been set; however, even if you include

```
POKE B+24,15
```

there will still be no sound. So we are left with two important questions: 'How do you actually make a sound sound?' and 'How do you start the ADSR cycle?'

The ADSR trigger

It should now be clear that the ADSR values are simply settings on two of SID's registers which are there to be obeyed when the sound cycle is triggered. If you think back to the early chapters, we introduced a number of programs which were quite successful at making sounds, so where was the trigger?

You will remember that we said that poking register 4 produces different waveforms: 17 for triangle, 33 for sawtooth, and so on. We must now put our hands on our hearts and tell you that what we told you then was not true, or rather it wasn't 100 per cent true. The whole truth is that 16 is the value to be poked for the triangle waveform, 32 for the sawtooth and the extra 1 is added to trigger the sound. So in all the programs we have met so far, poking the value 33 into register 4 has performed two functions; first it has selected the waveform and secondly it has triggered the ADSR cycle.

As we are now being so honest we ought to come clean and tell you that it is only the ADS part of the cycle which is triggered. The attack and decay will occur and the note will continue playing forever at the selected sustain level until some further action is performed like turning the volume down or switching off the computer! The means by which the release part of the ADSR cycle is effected requires setting register 4 back to the waveform value originally used but with the extra 1 used to trigger the sound removed. So if we use

```
POKE B+4, 17
```

to trigger the note, then

```
POKE B+4, 16
```

will move the ADSR cycle to its release phase.

Only one point remains to be explained before we can listen to ADSR working on the computer and confirm that it behaves exactly as described. If you have a program which has the ADSR settings we described in the example earlier and trigger the sound using the tri-

angle waveform as in:

```
10 POKE B+24 ,15
20 POKE B+5, 14*16+12
30 POKE B+6, 7*16+11
40 POKE B+4, 17
50 POKE B+4, 16
```

sadly all that will be heard will be a click. So much effort for so little result! The problem is that we have turned off the ADS part of the sound in line 50 too quickly after turning it on in line 40, and the cycle is cut off in its prime. As we said earlier, the time that the note plays for is not controlled by SID but externally by your program. What is needed is a pause between the trigger and the switching off. This may be achieved in a variety of ways, the simplest of which uses the dummy loop system already described in Chapter 4.

A second and more precise device to create a time delay within a program uses the internal clock built into the Commodore 64. This may be accessed as the variable TI; it is initialised to zero when the computer is first switched on and then counts in jiffies, of which there are 60 in each second. So to create a pause of 11 seconds we simply need to check for a time lapse of 11*60 = 660 jiffies. This device will be used in the next program.

Licking the ADSR envelope

Type in Program 8.1 and run it several times. The A,D,S and R values within the program are those described earlier so that you should hear the sound increase for 5 seconds, a decay of 3 seconds, the note should continue at a volume level of half the peak for 11 seconds and then decline to zero volume over a period of about 3 seconds. The time when the release phase starts is shown by a message on the screen.

Program 8.1

```
10 REM 8.1  ADSR
20 B=54272              : POKE B+24,15
30 A=14 : D=12          : S= 7 : R=11
40 POKE B+5,A*16+D      : POKE B+6,S*16+R
50 POKE B+1,50          : POKE B+4,17
60 FOR T=0 TO 7000      : NEXT
70 PRINT "RELEASING"    : POKE B+4,16
80 FOR T=0 TO 5000      : NEXT
90 POKE B+24,0
```

The obvious next step is to start to change the four values for A,D,S and R and to listen to the resulting sound. We suggest that you do not change them all at once but rather, keeping three of them constant, just change the fourth. The A,D,S and R settings are going to be one of the main tools in producing sound effects so it is obviously quite important to get a feel for them.

Random envelopes

If, as we do, you think machines should do your work for you, instead of repeatedly typing in different A,D,S and R values in Program 8.1, we can arrange for the changes in the parameters to be automatically altered within a program. You can then sit back and concentrate on how the different settings affect the sound heard.

Program 8.2 plays a short sequence of notes. As there are 16*16*16*16, that is over 65000 different combinations of ADSR settings possible, and the tune takes about 5 seconds to play, even if you continually listened, night and day, it would still take about four days to hear all the possible combinations! The program has been designed to play just some of the possible combinations.

When you run the program, listen out not only for the by now predictable sound envelopes, but also for some sounds which approximate to real musical instruments. We are now near victory in the battle to control the SID chip.

Program 8.2

```
1 REM 8.2  RANDOM
20 B=54272              : POKE B+24,15
30 A=INT(RND(0)*12)     : D=INT(RND(0)*10)
40 S=INT(RND(0)*14)     : R=INT(RND(0)*10)
50 PRINT "A="A"  D="D"  S="S"  R="R
60 POKE B+5,A*16+D      : POKE B+6,S*16+R
70 READ F,I             : IF F<0 THEN 140
80 G=F*16.777 :H=INT(G/256) : L=G-256*H
90 POKE B+1,H :POKE B+4,17  : POKE B,L
100 FOR T=0 TO 25*I*(A+D+1) : NEXT
110 POKE B+4,16
120 FOR T=0 TO 25*R+1: NEXT
130 GOTO 70
140 FOR T=0 TO 300      : NEXT
150 RESTORE             : GOTO 30
1000 DATA 523.25,2,349,5,440,1,523.25,1
1010 DATA 587.33,1,329.63,6,-1,-1
```

When you have heard enough variations on the original theme, you might like to change the waveform of the sound by altering the 17 in line 90 to 33, and the 16 in line 110 to 32. When you run the program again you will hear another collection of interesting sounds. When you have listened to the details of the ADSR envelope cycle for long enough to appreciate the four phases, relax and listen to the whole sound–it has a very relaxing and hypnotic effect!

Look at this sound

So far you have seen some illustrations of the ADSR envelope and been told how the shape has considerable effect on sound production. By now you should be able to picture in your mind the various characteristic shapes of the amplitude envelope for fast and slow attacks, sudden and minimal decays and so on. You may at some time have seen an oscilloscope picture of a waveform being played. What has been missing is the means of seeing the ADSR cycle and its covering envelope on the same machine that has been producing the sound–you can now do just that with your Commodore 64.

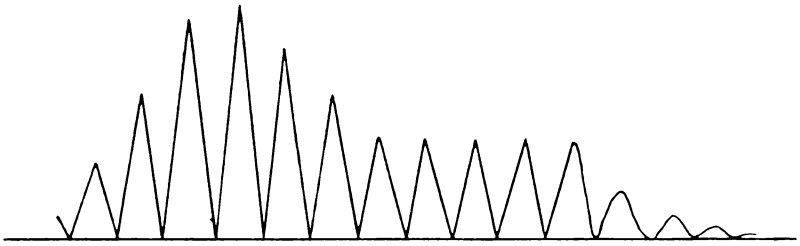
Program 8.3

```
10 REM 8.3  GRAPH
20 DIM Y(500),W(500) : PRINT "␣"
30 B=54272 : B1=B+28 : B2=B+27 : G=8192
40 FOR I=0 TO 24  : POKE B+I,0 : NEXT
50 POKE B+14, 80  : POKE B+19,12*16+11
60 POKE B+24,15   : POKE B+20,16*6+10
70 POKE B+18,17   : FOR X=1 TO 200
80 Y(X)=PEEK(B1)  : W(X)=PEEK(B2): NEXT
90 POKE B+18,16   : FOR X=201 TO 300
100 Y(X)=PEEK(B1) : W(X)=PEEK(B2): NEXT
110 FOR I=1024 TO 2023 : POKEI,3 : NEXT
120 FOR I=0 TO 7999 : POKE G+I,0 : NEXT
130 POKE 53272,PEEK(53272) OR 8
140 POKE 53265,PEEK(53265) OR 32
150 FOR X=1 TO 300
160 Y=199-INT(5+Y(X)*W(X)/512)
170 C=INT(X/8) : R=INT(Y/8) :L=Y AND 7
180 BY=G+R*320+8*C+L   : BI=7-(X AND 7)
190 POKE BY,PEEK(BY) OR (2^BI)
200 NEXT X : POKE B+4,0 : INPUT A$
```

Program 8.3 will use the computer's high resolution graphics capabilities to show the ADSR cycle. Type it in and run it. The picture of the sound is not produced 'instantly' as with an oscilloscope but you will see the familiar envelope pattern develop over a period of time –although it takes up to two minutes to develop, it is worth waiting for.

Look at the diagram on p.78 to see an example of what is produced. We are using such a low frequency that you can see the waveform! The sound, however, cannot be heard as a note. Notice how the overall shape of the picture has the classic ADSR pattern.

The display from Program 8.3



After you have admired the picture for a time, you will want to try varying the program to make the overall envelope clearer. First you have to stop the program. Press RUN/STOP at the same time as the RESTORE key to return control from the high resolution graphic display. Now some slight alterations in lines 151 to 154 will produce the extended Program 8.4.

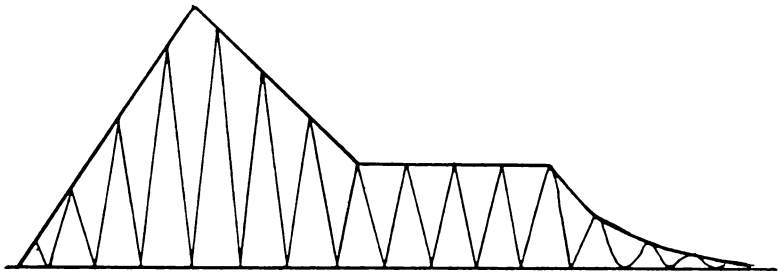
Program 8.4

```
10 REM 8.4  GRAPH MODIFIED
20 DIM Y(500),W(500) : PRINT "␣"
30 B=54272 : B1=B+28 : B2=B+27 : G=8192
40 FOR I=0 TO 24  : POKE B+I,0 : NEXT
50 POKE B+14, 80  : POKE B+19,12*16+11
60 POKE B+24,15   : POKE B+20,16*6+10
70 POKE B+18,17   : FOR X=1 TO 200
80 Y(X)=PEEK(B1)  : W(X)=PEEK(B2): NEXT
90 POKE B+18,16   : FOR X=201 TO 300
100 Y(X)=PEEK(B1) : W(X)=PEEK(B2): NEXT
110 FOR I=1024 TO 2023 : POKEI,3 : NEXT
120 FOR I=0 TO 7999 : POKE G+I,0 : NEXT
130 POKE 53272,PEEK(53272) OR 8
140 POKE 53265,PEEK(53265) OR 32
150 FOR X=1 TO 300
151 Y=199-INT(5+Y(X)/2)
152 C=INT(X/8) : R=INT(Y/8) :L=Y AND 7
153 BY=G+R*320+8*C+L   : BI=7-(X AND 7)
```

```
154 POKE BY,PEEK(BY) OR (2^BI)
160 Y=199-INT(5+Y(X)*W(X)/512)
170 C=INT(X/8) : R=INT(Y/8) :L=Y AND 7
180 BY=G+R*320+8*C+L    : BI=7-(X AND 7)
190 POKE BY,PEEK(BY) OR (2^BI)
200 NEXT X : POKE B+4,0 : INPUT A$
```

When you run this program, you will see not only the same frequency graph as before but also the amplitude envelope superimposed, as shown in the diagram below.

Triangle waveform with envelope



As before, 'break out' of the graphics using RUN/STOP and RESTORE. This time, change the 17 in line 70 and 16 in line 90, which were used for the triangular waveform, to 33 and 32 respectively; these are the values for the sawtooth waveform. Now when you run the program you will see a graphical display of the sawtooth waveform. Note that the vertical drop part of the shape is missing because it occurs too quickly.

Finally it is time to have a peek at the white noise waveform. Stop the program in the usual way, but this time change the waveform values in lines 70 and 90 to 129 and 128 respectively. When you run the program, you will really see white noise, not the simplified version shown earlier in Chapter 6. What you see is more like 'snow' – which of course could be used to explain its name! The picture illustrates the completely random mixture of frequencies.

Notice also that the ADSR envelope is still present, the amplitude of

the note being controlled by the ADSR setting. In all these investigations we have used the following ADSR settings:

A = 12 D = 11 S = 6 R = 10

A final modification for Program 8.4 is to vary these values to obtain different amplitude envelopes. Your friends may need some convincing when you tell them that you have seen sound!

Multiple ADSRs

As there are three voices on the SID chip it is important to realise that it is possible to set each one with a different ADSR. Each of the ADSR cycles will be triggered independently when the waveform is selected so that you have total control over the type of sound produced by each voice. The only feature of the sound for the three voices which cannot be set independently is the overall volume setting in register 24. So the amplitude or volume envelopes controlled by ADSR work independently but within SID's overall volume setting.

By the 'rule of seven' which was described in Chapter 7, the registers for Voice 2 and Voice 3 are 7 and 14 higher than those for Voice 1. Hence if the ADSR values for Voice 1 are set in registers 5 and 6, then the registers for Voice 2 are registers 12 and 13 and for Voice 3 registers 19 and 20. You may find it useful to look at the full register map in Appendix 3, to see how the separate voices are controlled. When playing music you will often want the same type of sound from each of the voices you are using so that the same ADSR can be set in each voice.

One final point; you may have noticed that we managed to produce a variety of sounds in earlier chapters without any need to consider the ADSR envelope. If you look back at Program 5.2, for example, there was no mention of register 5 and for register 6 we used

```
POKE 6, 240
```

As the settings for attack and decay were both zero because no other value had been given to them in register 5, there was 'instant' attack and decay. If you look at Appendix 5 you will see that the shortest

attack and decay SID permits are 2 and 6 thousandths of a second respectively–still quite fast!

Now the 240 poked into register 6 is really 15*16 + 0 which represents a sustain level of 15 and also instant release (again in reality 6 milliseconds). As the sustain level is the maximum possible it is now clear why there is no reduction in volume. Indeed there would have been no difference in sound if instead of the value 240 being poked into register 5, any of the values 241 to 255, giving release settings of 0 to 15 had been used instead. That is because the waveform was not switched off and therefore there could not be any release phase in the ADSR cycle.

It has been quite an effort mastering the amplitude envelope controlled by the ADSR cycle. However, you will find it all to have been worthwhile when you investigate the challenging world of sound effects and synthesised musical instrument sounds in the next chapters.

9

Instruments of the Orchestra

Back in Chapter 6 we saw how the main property that created the easily identifiable distinct sounds of the instruments of the orchestra was the presence of overtones. More accurately, it is the relative sizes of the different overtones present in relation to the fundamental frequency. What we shall do in this chapter is investigate the sounds of various musical instruments, attempt to mimic them using SID and finally create a musical instrument which does not actually exist!

It is not surprising that there are different characteristic sounds when the physical basis of the sources of these sounds is considered. Perhaps it would be useful to note one common element they have; they are all designed to set the air around them in motion vibrating at particular frequencies. Perhaps a second common element they have is that the larger they are, the lower the frequency of note produced. This is particularly noticeable in the wide range of drums, each larger and producing deeper notes than the one before.

One section of the orchestra comprises the violin family or strings; their main problem is to create enough sound to be heard. If the strings were just held taut in air their sounds would be inaudible, the hollow wooden structures are present to create a larger source of vibration and thus more sound. Even then it is noticable that an orchestra requires a relatively larger number of players in the string section, particularly the violins, to balance the sounds produced by the other sections.

The principal way in which the strings are set in motion is by bowing, in which friction between the bow and string sets the latter vibrating. This produces transverse vibrations where the string moves up and down at right angles to its length. Even when bowing it is possible to produce a totally different sound, particularly when the player is a learner; this is due to setting up longitudinal waves along the

string, when the bow moves along the direction of the string instead of across it. These vibrations are at a much higher frequency than the normal transverse waves, and explain how a violin, when bowed wrongly, can make a sound like a cat whose tail has just been trodden on!

A totally different sound can be obtained by pizzicato, i.e. plucking the string; although the same resonator is used, the harmonics set up are very different. Note that apart from the general harmonic structure defining the characteristic sounds of the instruments, there are other important differences. For example, when a string is plucked, it has a very different amplitude envelope from the one that it has when it is bowed. In other words, the two sounds have different ADSR properties.

A large number of orchestral instruments use human wind power to create their sound; however, there are many variations among the different wind instruments. For example, one section uses reeds to set up vibrations–as air is blown over their surface they vibrate at their natural frequency and set the air in the pipe vibrating. By opening or closing different holes along the length of the instrument, whether it be clarinet, oboe or any of the woodwind section, different lengths of air vibrate producing notes of different frequency.

A second set of wind instruments, the brass section, uses lip reeds, which is a polite way of saying that the player has to make the sound by setting up vibrations between his or her lips. A final group of wind instruments are those in which air is blown over an aperture in a pipe. Examples of this type of instrument are the flute and the piccolo. The air has to be blown sufficiently fast to produce turbulence over the hole and thus vibrations. The physics of sound shows that this is equivalent to setting up vibrations of air in an open pipe, which has the effect of producing a natural frequency one octave above that which would be produced if the pipe were closed at one end. This explains why the flute and piccolo produce such high notes.

SID's overture

As you already know, SID has a repertoire of four basic sounds, of which three are quite familiar, the sawtooth, triangle and white noise.

This last contains all frequencies and so is of no use in trying to mimic the sonorous instruments of the orchestra.

If we are to recreate the sound of each instrument of the orchestra then we really need to be able to fix the set of harmonics in appropriate ratios; this approach is not feasible, however, so we shall have to rely on the particular sounds which are available on the SID chip. The implication of this is that the sounds we shall produce will not be exact; but they are in fact quite good approximations.

Of course we have some further control over the sounds which are produced. For example, changing the ADSR envelope operating on a single waveform like the sawtooth will be very effective in changing the sound produced by the computer. It is possible to do some fine tuning using filter techniques; these will be discussed in Chapter 16.

Now to hear some sounds. Type in Program 9.1.

Program 9.1

```
10 REM 9.1  INSTRUMENTS
20 DIM F(64)        : B=54272
30 PRINT"␣"         : FOR J=0 TO 24
40 POKE B+J,0       : NEXT
50 POKE B+24,15     : FOR J=1 TO 42
60 READ F(J)        : NEXT
70 FOR I=1 TO 9
80 READ N$(I),A(I),D(I),S(I),R(I),W(I)
90 NEXT
100 GET A$          : IF A$<>"" THEN  100
110 FOR I=1 TO 9    : PRINT I,N$(I):NEXT
120 PRINT           : PRINT "INSTRUMENT";
130 GET A$          : IF A$="" THEN 130
140 I=ASC(A$)-ASC("0")
150 IF I<=0 THEN END
160 IF I> 9 THEN 120
170 POKE B+5,A(I)*16+D(I)
180 POKE B+6,S(I)*16+R(I)
```

```
190 K=PEEK(653)    : IF K=4  THEN 100
200 J=PEEK(197)    : IF J=64 THEN 190
210 F=F(J)*2^K     : G=F*16.777
220 PRINT "␣"N$(I): H=INT(G/256)
230 L=G-256*H      : POKE B,L
240 POKE B+1,H     : POKE B+4,W(I)
250 IF J=PEEK(197) THEN 250
260 POKE B+4,W(I)-1     : GOTO 190
1000 DATA ,,,,,,,,277.2,261.6,,,293.7
1010 DATA 311.1,,,,329.6,,,349.2,370.0
1020 DATA ,,415.3,392.0,,,440.0,466.2,,
1030 DATA ,493.9,,,523.3,554.4,,,,587.3
2000 DATA BANJO          , 1, 7, 0, 0,33
2010 DATA ORGAN          , 0, 0,15, 0,17
2020 DATA OBOE           ,11,14, 0, 0,17
2030 DATA XYLOPHONE      , 0, 9, 0, 0,17
2040 DATA VIOLIN         , 6, 9, 5, 4,33
2050 DATA HARPSICHORD    , 0, 9, 0, 0,33
2060 DATA FLUTE          , 2, 8, 9, 0,17
2070 DATA ACCORDION      , 6, 6,13,11,33
2080 DATA GUITAR         , 2, 9, 0, 0,33
```

When you run Program 9.1 you will see a 'menu' of instruments which can be selected simply by pressing a number between 1 and 9. After selecting the instrument you want to hear, the computer will act like a keyboard, as we saw in Chapter 3; you can press keys to hear sounds which approximate real musical instruments. For each instrument various of SID's settings have been stored; these include waveform and ADSR settings. When you want to change instruments, press the CTRL key, whereupon the menu will be presented again. To end the program, press 0.

Vibrato and tremolo

At first sight what we are going to look at next may seem rather complicated–we have broken it into stages and used separate programs to demonstrate each point. Vibrato is the effect heard when the person playing a musical instrument, particularly a string instrument like the violin, instead of playing a single note, say A5, actually plays a sweep of notes which oscillate slightly above and below the note. On

a violin this is usually done by rocking the wrist of the left hand while playing a note. At first it may seem strange that a violinist who wants to play in tune would play a wide range of notes around the note being played; however the vibrato effect produces a much richer sound.

What is really happening in this situation is that there is a main frequency of vibration which has added to it a slight additional extra oscillating frequency either side of the principal note. Anything which is done in a regular way to affect the frequency of a note is called frequency modulation. Consider Program 9.2:

Program 9.2

```
10 REM 9.2  ROCKING NOTES
20 B=54272          : POKE B+24,15
30 POKE B+6,240     : POKE B+4 ,17
40 F=440            : FOR N=1 TO 12
50 G=F*16.777       : H=INT(G/256)
60 POKE B+1,H
70 FOR I=1 TO 5     : FOR J=1 TO 20
80 POKE B,J*10      : NEXT
90 FOR J=1 TO 20    : POKE B,200-J*10
100 NEXT            : NEXT
110 F=F*1.06        : NEXT
120 POKE B+24,0     : END
```

Starting at A4 (440 cycles per second) the twelve semi-tones which make an octave are played in sequence. This is done between lines 40 and 110. For each note played, a loop running from line 70 to line 100 sets up 5 cycles of vibrato. Line 80 sweeps the low frequency register 0 with values ranging from 10 to 200 while in line 90 there is a loop which takes its value back down to 0. By changing the value in the low register, number 0, we are affecting the frequency of the note being played; the main factor changing the frequency is the value poked into address 1 in line 60.

This program has produced a vibrato effect by using values produced in the Basic program. We might like to consider other ways of producing the same effect from within the computer itself. What we need to find is a source of a regularly varying oscillation. You might like

to imagine where this source may be found (hint: think about SID's other registers). Before we provide the solution to this problem we shall consider another method of providing a regularly oscillating source of vibration.

One way to add to a basic frequency is to sample from a sine wave. If you imagine taking lots of sine values in sequence and use the values to affect the sound you are listening to, then the variation will be regular and a vibrato effect will be heard. Note that the additional modulating signal must be at a much lower frequency than that of the principal note being heard. This is the driving concept behind FM radio, where to a very high frequency sine wave which carries the signal, slight variations of frequency are added to hold meaningful information like music or the human voice.

On the Commodore 64, the sound waveforms are not sinusoidal but sawtooth, triangle and white noise. However, to produce a vibrato effect we need to add to one of these waveforms an extra modulating frequency, designed to move above and below the original frequency in a slow regular manner. Now it is possible to produce a sine function in Basic and sample it within a program. These values can then be added to the frequency of each note we are playing. This has been incorporated in Program 9.3.

Program 9.3

```
10 REM 9.3  WAVY NOTES
20 B=54272          : POKE B+24,15
30 POKE B+ 6,240 : POKE B+4 ,17
40 F=440            : FOR N=1 TO 12
50 G=F*16.777     : H=INT(G/256)
60 POKE B+1,H     : FOR I=1 TO 10
70 FOR T=0 TO 6 STEP .7
80 POKE B,125*(SIN(T)+1)
90 NEXT
100 NEXT
110 F=F*1.06        : NEXT
120 POKE B+24,0   : END
```

As in the previous program, we are playing an octave of semi-tones beginning at A4. The sampling of the sine function is done at line 80, again producing values in the low frequency register number 0. As the sine function has values in the range −1 to 1 it has had a 1 added to it to change the range to 0 to 2. The values that can be poked into the low frequency register number 1 range from 0 to 255, so that the sampled value is multiplied by 125 to change the range to 0 to 250, which is close to its limit.

The speed of the vibrato is controlled by the step size of the loop in line 70. If the value 0.7 is replaced by 2 the vibrato will sound very fast, while if it is set at 0.1 the sound makes you feel quite dizzy.

A far more elegant approach to the subject of vibrato and frequency modulation is to look around for any other sources of regularly changing vibration. As we hinted earlier, you need look no further than SID itself. It has an oscillator and can produce several regularly changing waveforms – sawtooth and triangle, for example. So how can we access the numerical values of these oscillations without actually playing them? The answer lies in one of the cleverest tools provided for the programmer – PEEK.

In the same way that POKE allows you to place any value in a particular location in the computer's memory (as we have been doing to produce all the sounds so far), PEEK allows you to look at the contents of a memory location without affecting it.

SID has been designed to allow you to peek into the oscillations of Voice 3; if you check with the register index in Appendix 3, registers 25 to 28 can only be peeked – they are read only locations. Register 27 holds the numerical (digitised) value of whatever waveform is being played in Voice 3. If this is a new idea to you then Program 9.4 should help you to understand it.

Program 9.4

```
10 RE.1 9.4   PEEK & PRINT
20 B=54272
30 F=3              : G=F*16.777
40 H=INT(G/256)     : L=G-H*256
```

```
50 POKE B+14,L    : POKE B+15,H
60 POKE B+18,32   : V3=B+27
70 FOR I=1 TO 22 : PRINT PEEK(V3)
80 NEXT           : END
```

A sawtooth waveform is set in register 18 at a very low frequency (3 cycles per second)–it has to be slow so that when you take sample values from it, you can see the pattern of numbers. If it were too high, you would be seeing values from several cycles rather than several values from each cycle. We have arranged for the waveform of register 3 to be peeked (line 70) and are printing the value on the screen.

When you run the program you will see sequences of increasing numbers in the range 0 to 255. If you look back to the pictures of waveforms in Chapter 6, you can see that this is exactly what you would expect from a sawtooth waveform. You will not see any decreasing numbers as the fall occurs so fast. Note that it is this technique of peeking that was used in Chapter 8 to produce the picture of sound.

To make this demonstration even more convincing, change the waveform of Voice 3. By changing line 60 to

```
60 POKE B+18,16 : V3=B+27
```

a triangle waveform is produced in Voice 3. This time the numbers printed on the screen move both up and down as you would expect. Now change the waveform to white noise. This time when you run the program the numbers do not have any pattern–they are random (in the range 0 to 255).

A further way in which this program can be used is to change the frequency of the sound in Voice 3. This can be done by changing the value of F in line 30. You ought to change the waveform back to sawtooth or triangle first–that way you will be able to see the pattern of numbers. With a frequency of 6 cycles per second, the numbers should run through their cycles at exactly twice their original rate.

Now it is time to harness the output from Voice 3 and use it to modify or modulate the frequency of some other waveform. If the vibrato effect is to be heard then the variation of frequency needs to be fairly

slow–hence the frequency in Voice 3 will be set low. Program 9.5 has a triangle waveform and plays the octave of semi-tones in Voice 1 as before, but this time it is modulated by the numerical values of the triangle waveform oscillating in Voice 3. Run the program to hear the very smooth vibrato effect.

Program 9.5

```
10 REM 9.5  PEEK & POKE
20 B=54272          : POKE B+24,143
30 POKE B+ 6,240 : POKE B+4 ,17
40 F3=4             : G=F3*16.777
50 H=INT(G/256)     : L=G-256*H
60 POKE B+14,L      : POKE B+15,H
70 POKE B+18,16     : F=440
80 FOR N=1 TO 12
90 G=F*16.777       : H=INT(G/256)
100 POKE B+1,H      : FOR I=1 TO 100
110 POKE B,PEEK(B+27)/2
120 NEXT
130 F=F*1.06        : NEXT
140 POKE B+24,0     : END
```

There are three main parts of the sound which can be alter the vibrato heard in the program. One, of course, is the basic note. We have designed the program to play the complete octave of semi-tones but by simply poking in the appropriate values in frequency registers 0 and 1 any other note can be played.

The second feature which can be altered is the amount of movement built above and below the basic frequency. The program allows for a movement of 4 cycles per second either side of the principal note, but you should try some changes to hear the effect. This can be done by changing the divisor in line 110. Changing the 2 to a 1 will produce a movement of 8 cycles per second.

The last feature that can be altered is the speed at which the vibrato occurs. This is determined by the frequency set in line 40. Try changing the value of F3 to 8; the effect is a much speedier vibrato. If you go even higher, say to 30, it sounds more like two notes.

In the same way as vibrato is the modulation or variation of a basic frequency of a tone, modulation can also be made to the volume– this is called amplitude modulation. If a sound plays at a certain volume and it is made to vary regularly slightly up and down, a tremolo effect can be heard. The Commodore does not allow much scope for varying the volume of a sound, having only 15 settings on volume register 24, so SID's ability to produce a convincing tremolo effect is quite limited.

To finish off our investigation into the vibrato effect, here is Program 9.6.

Program 9.6

```
10 REM 9.6  VIBRATING INSTRUMENTS
20 DIM F(64)        : B=54272
30 PRINT"␣"         : FOR J=0 TO 24
40 POKE B+J,0       : NEXT
45 POKE B+14,85     : POKE B+18,16
50 POKE B+24,143    : FOR J=1 TO 42
60 READ F(J)        : NEXT
70 FOR I=1 TO 9
80 READ N$(I),A(I),D(I),S(I),R(I),W(I)
90 NEXT
100 GET A$          : IF A$<>"" THEN  100
110 FOR I=1 TO 9    : PRINT I,N$(I):NEXT
120 PRINT           : PRINT "INSTRUMENT";
130 GET A$          : IF A$="" THEN 130
140 I=ASC(A$)-ASC("0")
150 IF I<=0 THEN END
160 IF I> 9 THEN 120
170 POKE B+5,A(I)*16+D(I)
180 POKE B+6,S(I)*16+R(I)
190 K=PEEK(653)     : IF K=4  THEN 100
200 J=PEEK(197)     : IF J=64 THEN 190
210 F=F(J)*2^K      : G=F*16.777
220 PRINT "␣"N$(I): H=INT(G/256)
230 L=G-256*H       : POKE B,L
240 POKE B+1,H      : POKE B+4,W(I)
250 IF J<>PEEK(197) THEN 260
```

```
255 POKE B,PEEK(B+27)/2 : GOTO 250
260 POKE B+4,W(I)-1     : GOTO 190
1000 DATA ,,,,,,,,277.2,261.6,,,293.7
1010 DATA 311.1,,,,329.6,,,349.2,370.0
1020 DATA ,,415.3,392.0,,,440.0,466.2,,
1030 DATA ,493.9,,,523.3,554.4,,,,587.3
2000 DATA BANJO          , 1, 7, 0, 0,33
2010 DATA ORGAN          , 0, 0,15, 0,17
2020 DATA OBOE           ,11,14, 0, 0,17
2030 DATA XYLOPHONE      , 0, 9, 0, 0,17
2040 DATA VIOLIN         , 6, 9, 5, 4,33
2050 DATA HARPSICHORD    , 0, 9, 0, 0,33
2060 DATA FLUTE          , 2, 8, 9, 0,17
2070 DATA ACCORDION      , 6, 6,13,11,33
2080 DATA GUITAR         , 2, 9, 0, 0,33
```

This is a modified version of the first program in this chapter. The changes are

```
45 POKE B+14,85 : POKE B+18,16
50 POKE B+24,143 : FOR J=1 TO 42
250 IF J()PEEK (197) THEN 260
255 POKE B,PEEK(B+27)/2 : GOTO 250
```

The change in line 50 switches off the output from Voice 3. Appendix 4 shows how by poking an extra 128 beyond the normal 15 we use for a volume setting, it can be turned off. If you want to change the speed of the vibrato it may be done by changing the value of register 14 in line 45; it can take any value between 0 and 255.

Rectangular or pulse waveforms

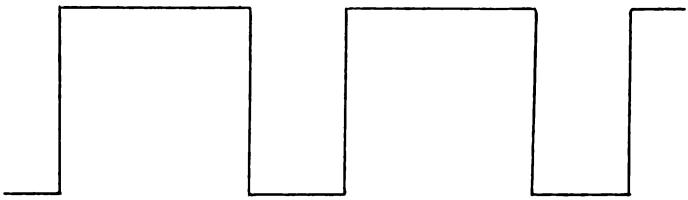
The sounds you heard using the keyboard program at the start of this chapter were first approximations to the true instrumental sounds. There are several ways they can be improved, for example by the inclusion of filters. However what we shall do now is to introduce a fourth waveform which will extend our horizons considerably, not only to mimic existing instruments but to produce really exciting musical sounds from instruments which do not even exist!

By now you should be familiar with the idea that the sound produced by SID in Voice 1 is controlled by

1. The volume, using register 24 and the ADSR cycle (registers 5 and 6).
2. The frequency at which it plays (registers 0 and 1).
3. The waveform (register 4).

In the same way that the triangle and sawtooth waveforms are produced by suitable combinations of the harmonics of a note, a different combination can lead to a whole family of waveforms called rectangular or pulse. The reason why we have used the word 'family' can be understood by looking at the following diagram:

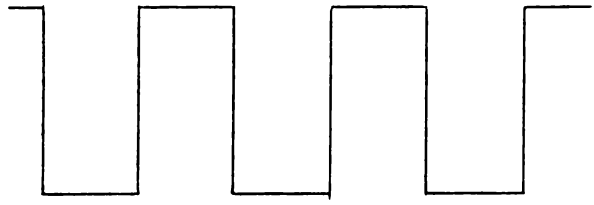
Rectangular (pulse) waveform



A complete cycle of this waveform comprises one high part and one low part. The number of cycles per second represents the frequency of the note in exactly the same way as the other waveforms. However the exact shape of this waveform can be changed in the same way as ordinary rectangles can have a variety of shapes.

Each member of the family is defined by the ratio of high part in each cycle to low part. If the high part occupies 90 per cent of the complete cycle (and the low part 10 per cent), the particular waveform is defined by the ratio 90 per cent. If the two parts of the waveform are equal in length then the ratio is 50 per cent and it is called a square wave.

Square waveform



In order to use the pulse waveform in SID, Appendix 4 shows that register 4 must be set at 65 to trigger the sound and 64 to release it. However, SID also needs to be informed of which particular member of the pulse family we want. This is worked out as follows:

The ratio defining the pulse is multiplied by 40.95 to produce the pulse width number, PW. This is now converted into two numbers, very much in the way we are used to doing for frequency settings. So 'high' and 'low' values are obtained by the following calculations:

```
HP = INT(PW/256)
LP = PW - 256*HP
```

The high pulse should be set in register 3 and the low pulse in register 2. This is done by using:

```
POKE B + 2,LP
POKE B + 3,HP
```

Now we know how to produce the pulse waveform, it might be useful to hear it played. Program 9.7 is a revitalised form of the original keyboard program, and the pulse width has been set to produce a piano-like sound. The pulse ratio used is 5 per cent.

Program 9.7

```
10 REM 9.7  PULSATING  KEYBOARD
20 DIM F(64)        : B=54272
30 FOR J=0 TO 24    : POKE B+J,0 : NEXT
40 FOR J=1 TO 42    : READ F(J)  : NEXT
50 POKE B+2,200     : POKE B+24,15
60 POKE B+5,16*1+9: POKE B+6,1
70 J=PEEK(197)      : K=PEEK(653)
80 IF J<>J1 THEN POKE B+4,64
90 IF J=64 THEN 70
100 F=F(J)*2^K      : G=F*16.777
110 H=INT(G/256)    : L=G-256*H   : J1=J
120 POKE B,L        : POKE B+1,H
130 POKE B+4,65     : GOTO 70
1000 DATA ,,,,,,,,277.2,261.6,,,293.7
1010 DATA 311.1,,,,329.6,,,349.2,370.0
1020 DATA ,,415.3,392.0,,,440.0,466.2,,
1030 DATA ,493.9,,,523.3,554.4,,,,587.3
```

Try playing a few tunes; do not forget that to shift the keyboard up one or two octaves, you keep the caps shift or Commodore keys depressed. When you have heard enough you might like to experiment to hear how different the sound is when a different ratio is used. The value set in register 2 at line 50 can be set anywhere between 0 and 255. If you want a more dramatic change in sound, change line 50 to

```
50 POKE B+3,1 : POKE B+24,15
```

Now by varying the setting in register 3 anywhere from 0 to 15, the total range of different pulses can be heard. You should find that close to the centre of the range, around 7 and 8, the sound is quite mellow, but that at either extreme the sound is more sharp or spiky.

To make investigating the pulse waveform easier for you we include Program 9.8, which can be considered a development tool. When you run it the same note is played 30 times. That is what is happening but

it is not what you hear! The pulse width is changed for each note so that each one sounds quite different. Try changing the frequency of the note by setting register 1 to any value in the range 1 to 255 at line 50.

The other way this development program can be used is by changing the time delay between each of the notes. That means changing the loop counters in lines 80 and 120. If the time delay is made quite small, the effect is of sweeping the pulse, that is, running quickly through the range of pulses available. This idea will be exploited further when we come to sweep up.

Program 9.8

```
10 REM 9.8  FEELING THE PULSE
20 B=54272
30 FOR I=0 TO 24:POKE B+I,0:NEXT
40 POKE B+6,240      : POKE B+24,15
50 POKE B+1,20       : POKE B+4,65
60 TM=1              : FOR PH=1 TO 15
70 POKE B+3,PH
80 FOR T=1 TO TM     : NEXT
90 NEXT
100 FOR PH=15 TO 1 STEP -1
110 POKE B+3,PH
120 FOR T=1 TO TM    : NEXT
130 NEXT
140 POKE B+24,0      : END
```

Sweeping up

We promised earlier that we would produce the sound of a musical instrument which does not exist. We shall keep this promise by using the pulse waveform, not played constantly as in the previous program but by changing or sweeping the pulse width while the note is playing! The effect is something like what you would expect if you dropped each note played by a piano into a pond!

Type in Program 9.9. It can obtained from Program 9.7 by adding lines 75 and 76.

Program 9.9

```
10 REM 9.9  RENRUT KEYBOARD
20 DIM F(64)        : B=54272
30 FOR J=0 TO 24    : POKE B+J,0 : NEXT
40 FOR J=1 TO 42    : READ F(J)  : NEXT
50 POKE B+2,200     : POKE B+24,15
60 POKE B+5,16*1+9: POKE B+6,1
70 J=PEEK(197)      : K=PEEK(653)
75 POKE B+3,P       : P=P+1
76 IF P>15 THEN P=0
80 IF J<>J1 THEN POKE B+4,64
90 IF J=64 THEN 70
100 F=F(J)*2^K      : G=F*16.777
110 H=INT(G/256)    : L=G-256*H   : J1=J
120 POKE B,L        : POKE B+1,H
130 POKE B+4,65     : GOTO 70
1000 DATA ,,,,,,,,277.2,261.6,,,293.7
1010 DATA 311.1,,,,329.6,,,349.2,370.0
1020 DATA ,,415.3,392.0,,,440.0,466.2,,
1030 DATA ,493.9,,,523.3,554.4,,,,587.3
```

We feel sure that you have not heard this musical instrument before. This is one of the most exciting aspects of home computing today: you can do what perhaps nobody else has ever done before. Using this program, by changing register 3, you can develop your own unique musical instrument!

10

That Sounds Like a...

Sound effects

Now that we have managed to play some quite melodious tunes and also taken a trip around and even beyond the instruments of the orchestra, it is time to reveal the real eye (or should it be ear?) opener. The same chip, SID, will now be used to create a very wide range of sound effects. Instead of simply providing you with some programs, each one of which would produce just one sound effect, we shall tackle the subject in a more methodical way. That means wherever possible analysing the common elements in different sounds and finding an efficient way of catering for them. Apart from anything else this will help to keep your programs looking neat.

When creating a sound, it is important to be able to identify the components from which it has come. One aspect of this process is to note whether the whole sound is simply a sequence of repeats of a simpler sound. For example, the sound of a frog croaking requires a single croak repeated, perhaps with some minor variations, many times. One surprising aspect of making sound effects is that quite often it is possible to identify and produce the appropriate elements of particular sounds, but when played the effect sounds completely wrong because it occurs much too fast. This is like playing the C Major scale in Chapter 2 without the time delays and thus so fast that you cannot hear the individual notes.

So sound effects programming needs a mechanism to space out the components and so slow down the complete process. Again using the frog as an example, if the component 'clicks' of a croak play too fast the sound loses its frog-like character and simply sounds like an extended click! The slowing down is achieved by incorporating time

delays within the sound production. Note also that there needs to be some time, not just within the croaks but also between the croaks.

Now it is obviously inefficient to create new lines within a sound effects program when nearly every sound effect will need the same or very similar lines. That is like reinventing the wheel every time you want to make a new car. So the approach that we shall follow is to provide some basic building blocks which can be used to form secure foundations for all your future investigations into sound effects.

A tool kit

Instead of providing lines of Basic program to be fitted within the code of your programs, we shall introduce our tools as a neat and ordered collection which can be kept separate from your programs but will be available as a resource whenever you need them. The device we shall use for this is called the subroutine.

A subroutine is a section of program code which can be entered through its first line and is left when the word RETURN is met. This idea is best illustrated by an example. In sound effects programming, as we have already seen, there is a need for time delays. In Chapter 8 we saw a means of providing a pause using the Commodore TI jiffy timer. Consider the following lines:

```
1100 REM WAIT T JIFFIES
1110 TS = TI
1120 IF TI-TS < T THEN 1120
1130 RETURN
```

This is a subroutine designed to provide a time delay or pause of T jiffies (remember that 60 jiffies equal one second). At any point in a program we can now write:

```
GOSUB 1100
```

and the subroutine will cycle around line 1120 until the time delay (that is the current time TI less the start time TS) is greater than the time delay value T. The useful point about subroutines is that they can be general purpose–if you want a delay of one second simply set T as

60 and write GOSUB 1100, if you want 5 seconds then set T as 300, and so on. When the RETURN command is met, the program flow returns to the next instruction after the GOSUB. So if you have a multiple statement using the : as in

```
100 GOSUB 1100 : PRINT "ADSR"
110 PRINT "CROAK"
```

the message ADSR will first be printed then the message CROAK.

If in a program you are going to develop a collection of sound effects, each requiring its own SID register settings, you will want to be able to clear all the registers at a stroke; this process of setting start values for variables and registers is called initialisation. The following subroutine is designed to set the base address for SID in variable B and clear the 25 registers into which you can poke values to zero.

```
1000 REM INITIALISE SID
1010 B=54272
1020 FOR IR=0 TO 24
1030 POKE B+IR,0
1040 NEXT
1050 RETURN
```

As before, this subroutine can be called from any point in your main program simply by writing

GOSUB 1000

and the return from this subroutine will again be to the next instruction following the GOSUB command.

There are two more basic tools you should find useful. First we need to set up A, D, S and R values in the appropriate registers. We suggest using the following subroutine:

```
1200 REM ADSR SETTINGS
1210 POKE B+5 ,A*16+D : POKE B+6 ,S*16+R
1220 POKE B+12,A*16+D : POKE B+13,S*16+R
1230 POKE B+19,A*16+D : POKE B+20,S*16+R
1240 RETURN
```

So in the main program values will need to have been given to the variables A, D, S, and R.

Some comments need to be made. First, in order for this subroutine to work correctly B must be set at 54272. This means that the initialising subroutine 1000 must have been called before subroutine 1200.

A second point relates to SID's multiple voices. Subroutine 1200 sets all of SID's voices to the same A, D, S and R values in its registers –which initially may seem rather strange. If you only wish to use one of the voices then the fact that all three voices have been given ADSR settings does not matter–remember that it is the waveform register that triggers the sound, nothing else. With multiple voice programming the same ADSR settings are frequently required for each voice, which is why subroutine 1200 is so useful. If, however, one voice, say Voice 2, needs to have a different set of ADSR values then you can still use subroutine 1200 first and then reset the ADSR registers for Voice 2 back in the main program.

The final basic tool will now be developed. Any sound that is to be produced using SID must have a frequency. You will recall the somewhat messy way this is done by converting frequency values in cycles per second into the two values L and H, which are then used to set SID's frequency registers. As this calculation is repeated so many times it is useful to consider it also as a basic building block. The following subroutine provides this tool:

```
1300 REM FREQUENCY CONVERTER
1310 G=F*16.777
1320 H=INT(G/256) : L=G/256*H
1330 RETURN
```

We have now established four basic utility subroutines for our Basic programs. We recommend that they are assembled together into a tool kit which can be used in all programs which are to contain sound. If you already have a program that uses line numbers around 1000 to 1300 then to incorporate these subroutines into the program simply place a 5 in front of each line number, so that the four subroutines will be as follows:

```
51000 REM INITIALISE SID
51100 REM WAIT T JIFFIES
51200 REM ADSR SETTINGS
51300 REM FREQUENCY CONVERTER
```

Two basic principles

When you listen to any sound, other than a monotone, it will contain features which are changing. It is the manner in which changes occur that defines any particular sound. There are two principal ways in which changes can be engineered: by amplitude control and by frequency control. Put simply, sound effects often have a sweep of amplitude/volume (increasing or decreasing) and sweeps of frequency similar to the glissando effect introduced in Chapter 2.

As these two processes are such common features of sound effects programming, it is useful to make them available as subroutines like the basic utilities already described. In designing the subroutine to sweep the volume we must first specify all the uses to which it could be put. We have to allow for either a rise or a fall, we have to allow for this being in large or small increments or steps, and we must allow for a time delay between changes in volume settings. The following subroutine satisfies this specification:

```
2000 REM VOLUME SWEEP
2010 FOR SW = SV TO FV STEP DV
2020 POKE B+24,SW
2030 GOSUB 1100
2040 NEXT
2050 RETURN
```

Here SV and FV are the start and finish volume settings and DV is the amount of change required at each stage. Note the use of the time delay subroutine in line 2030. Variable T needs to be given a value before subroutine 2000 is called.

If in calling the subroutine

SV = 4, FV = 10, DV = 2 and T = 60

the volume will rise from level 4 to volume settings 6, 8 and 10 with time delays of 1 second between each change. If, however,

SV = 15, FV = 0, DV = −1 and T = 30

the volume will fall from 15 (maximum) to zero in single unit steps at half second intervals. If no time delay is required, T is set to zero before calling the subroutine.

A similar idea will enable us to construct a frequency sweep routine. First we have to decide what it may be required to do; it will need to be able to work for a rise in frequency or a fall, permit small or large incremental changes and of course incorporate a possible time delay. The following subroutine satisfies this specification:

```
2100 REM FREQUENCY SWEEP
2110 FOR F = SF TO FF STEP DF
2120 GOSUB 1300
2130 POKE B,L :POKE B+1,H
2140 GOSUB 1100
2150 RETURN
```

Here SF and FF are the start and finish frequency settings and DF is the amount of frequency change required at each cycle of the loop. Note that this not only calculates the stepped frequencies required but pokes the appropriate L and H values into the frequency registers of Voice 1. It follows that if you want to sweep frequencies on any other voice then you will either have to write your own code (using the above subroutine scheme) or amend the subroutine to allow you to indicate which voice is to have its frequency registers swept. A more efficient approach is to always use Voice 1 when a frequency sweep is required in programs containing sound effects.

A word of warning. When you call a subroutine from your program beware of using the same variables in your main program as in the subroutines, otherwise the consequences are likely to be disastrous. To help you we have tried to keep away from common variable names like I, J, K, X and Y in the subroutines. You will find a complete glossary of the variables used in these subroutines in Appendix 7. You don't have to follow this scheme but if you do you will find the dis-

ciplined approach rewarding in the reduced number of program errors you will encounter.

A trip to the library

The four basic utility subroutines plus the last two sweep devices together create a useful programming resource. You will find that you want to call upon them for most of the sound-effect programs to be developed in this and the following chapters. To show the complete subroutine library, it will now be printed as a whole.

```
1000 REM INITIALISE SID
1010 B=54272
1020 FOR IR=0 TO 24
1030 POKE B+IR,0
1040 NEXT
1050 RETURN
1055 .
1100 REM WAIT T JIFFIES
1110 TS=TI
1120 IF TI-TS<T THEN 1120
1130 RETURN
1131 .
1200 REM ADSR SETTINGS
1210 POKE B+5 ,A*16+D :POKE B+6 ,S*16+R
1220 POKE B+12,A*16+D :POKE B+13,S*16+R
1230 POKE B+19,A*16+D :POKE B+20,S*16+R
1240 RETURN
1241 .
1300 REM FREQUENCY CONVERTER
1310 G=F*16.777
1320 H=INT(G/256)      : L=G-256*H
1330 RETURN
1331 .
2000 REM VOLUME SWEEP
2010 FOR SW=SV TO FV STEP DV
2020 POKE B+24,SW
2030 GOSUB 1100
2040 NEXT
2050 RETURN
```

```
2051 .
2100 REM FREQUENCY SWEEP
2110 FOR F=SF TO FF STEP DF
2120 GOSUB 1300
2130 POKE B,L          : POKE B+1,H
2140 GOSUB 1100        : NEXT
2150 RETURN
```

It is recommended that you type this in and make at least one copy of it; when you start writing a program which is to include sound effects this library will provide a basic foundation upon which to build.

Now hear this

By now you are probably getting a bit impatient; so much programming effort and from the computer not a peep! Now for the reward. Obviously if you want 'off the peg' sound effects you can look at the listings in the following chapters–if you haven't already! What we shall do here is to give some indication of how to create the appropriate lines of program for any particular sound effect.

Suppose the sound effect required is the breaking of waves on the sea shore accompanied by the squawking of seagulls sailing in the wind. As we said before the skill of writing sound effects is to be able to dissect the global sound into parts and then identify what is happening in each part. This is quite challenging when you listen to real sound, and perhaps surprisingly it is not much more difficult when you have to imagine the sound in your head. So relax and concentrate on a sea shore scene. Decide how long the sea shore effect is to last or, perhaps more appropriate from the programming point of view, how many wave cycles you want to experience. Let us suppose it is five. That dictates that there should be a loop arranging for the sound production to be executed five times:

```
FOR I = 1 TO 5
.....
.....
.....
.....
NEXT
```

Now in the sound picture in your head you must 'listen' to the dynamics of each wave–how long does it take to crescendo to a peak, how long is it sustained, how does the sound fall away? For this particular sound effect the timings are not critical, as slight variations do not significantly alter the overall effect. Let us now consider the two main components, the sea and the seagull, separately.

To make the sea sound, as with many non-musical sounds, we shall use the white noise waveform which is obtained by setting the waveform register to 129. If the sea is not too rough, you might expect a fairly slow build up, say about 5 seconds, no decay and after a sustained period, a slow decline phase as the sea recedes. This seems to indicate the following ADSR values, using the table in Appendix 5:

A = 14 D = 0 S = 15 R = 13

Remember that the release phase will only be produced when 128 is set in the frequency register. So it seems that the wave effect can be created solely by the ADSR envelope with an appropriate time delay for the sustain phase. Do not forget that the 15 value for S is for the level, not the duration.

The squawk of a seagull can vary considerably from squawk to squawk! To keep things simple we shall just develop a standard squawk; to make the sound even more realistic you can include random elements in the sound. This can be done in two ways. You can vary the sound of a squawk as already mentioned or more simply let it sound at different times in relation to the wave effect. If you are trying to make realistic sound effects it is important that the squawk is not always heard at the same part of the wave cycle! The use of random numbers in sound-effect production will be dealt with in a future chapter.

Another variation which would add realism to this effect would be to have sometimes one squawk and sometimes two. Nature is somewhat unpredictable, so some elements of uncertainty should be built into the program if at all possible. The standard squawk will be a frequency drop between two fairly high frequencies–obviously a job for frequency sweep routine 1300. We shall use:

SF = 1186 FF = 1142 DF = −3

When you run the program, try varying these values to hear the effect. We haven't yet considered the ADSR cycle for the gull squawk. The squawk is not a sequence of separate sounds requiring separate ADSR values but a continuous sustained sound. It follows that we shall not be interested in the attack and decay sections of the cycle but more the sustain and release phase. So again reference to Appendix 5 leads to the following values:

A = 0 D = 0 S = 15 R = 5

Now that we have an idea of how the two main component sounds can be produced, it is time to think about combining them in a single program. The first decision is which voice to use for which sound; the answer is dictated by the fact that the gull has a frequency sweep and so by the design we have used in the subroutine library should be in Voice 1. Hence Voice 2 will be used for the sea. The power of the SID chip is revealed in its ability to play the two separate components together without one affecting the other. It is time to reveal the whole program; notice how the sound library of subroutines described earlier is tagged on at the bottom.

Program 10.1

```
10 REM 10.1  SEA AND GULLS
20 GOSUB 1000 : POKE B+24,15
30 A=0  : D=0 : S=15 : R=5 : GOSUB 1200
40 A=14 : D=0 : S=15 : R=13
50 POKE B+12,A*16+D  : POKE B+13,S*16+R
60 POKE B+8 ,18
70 REM WAVE CYCLES
80 FOR I=1 TO 5       : POKE B+11,129
90 T =200 : GOSUB 1100 : GOSUB 500
100 POKE B+11,128
110 T=200 : GOSUB 1100 : GOSUB 500
120 T=200 : GOSUB 1100
130 NEXT
140 T=200 : GOSUB 1100
180 END
190 .
500 REM GULL SQUAWK
```

```
510 POKE B+4,33
520 SF=1186: FF=1142   : DF=-3
530 T=0     : GOSUB 2100
540 POKE B+4,32
550 RETURN
560 REM
1000 REM INITIALISE SID
1010 B=54272
1020 FOR IR=0 TO 24
1030 POKE B+IR,0
1040 NEXT
1050 RETURN
1055 .
1100 REM WAIT T JIFFIES
1110 TS=TI
1120 IF TI-TS<T THEN 1120
1130 RETURN
1131 .
1200 REM ADSR SETTINGS
1210 POKE B+5 ,A*16+D :POKE B+6 ,S*16+R
1220 POKE B+12,A*16+D :POKE B+13,S*16+R
1230 POKE B+19,A*16+D :POKE B+20,S*16+R
1240 RETURN
1241 .
1300 REM FREQUENCY CONVERTER
1310 G=F*16.777
1320 H=INT(G/256)      : L=G-256*H
1330 RETURN
1331 .
2000 REM VOLUME SWEEP
2010 FOR SW=SV TO FV STEP DV
2020 POKE B+24,SW
2030 GOSUB 1100
2040 NEXT
2050 RETURN
2051 .
2100 REM FREQUENCY SWEEP
2110 FOR F=SF TO FF STEP DF
2120 GOSUB 1300
```

```
2130 POKE B,L              : POKE B+1,H
2140 GOSUB 1100            : NEXT
2150 RETURN
```

After playing the sound effects program a few times, it is quite likely that you will have some ideas for changing it. Perhaps you want more than one squawk to play in sequence, perhaps squawks that rise and fall in pitch, perhaps more gulls by using Voice 3, more variation in the length of each wave. The variations you can include are endless. You may want to extend the printed program in some way to make a different sound picture. Have you thought about introducing a different sound effect from a later chapter (perhaps a rifle shot in Voice 3!). If so it might be useful to make a copy of this program.

Now that you have heard this sound picture you will no doubt be keen to produce further scenes yourself. In the next chapters we will describe sounds that can be grouped together because of a logical connection between them. However there will also be several new techniques not so far described which will further extend your abilities to create realistic sound effects. The only limit to SID's capabilities is your imagination!

11
The Sound of Flight

In this chapter we set out to cover the complete history of flight–in sound! We start with an early monoplane journey with a fateful ending, examine jet air travel, and finally launch a rocket into space!

Yesterday

First, back to earth. We shall develop a monoplane 'story' which will cover the folowing episodes:

firing the engine (with two faulty starts)
taxying for take-off
take-off
cruising flight
running out of fuel
nose dive
crash

The complete story is set out in Program 11.6, but it will be useful to develop the story in stages. The first sound to produce is the cruising flight–in setting this up and experimenting a little to get the sound just right, we are likely to discover many ideas which will be useful in developing the other parts of the story.

If you set the volume to a maximum and then repeatedly trigger and release the white noise waveform, the sound produced is very much like an early aeroplane engine when running at full speed. Program 11.1 shows how this can be done.

Program 11.1

```
10 REM 11.1  AIR JOURNEY - FLIGHT
20 GOSUB 1000
30 POKE B+24,15     : POKE B+1,255
300 REM IN FLIGHT
310 FOR I=1 TO 800 : POKE B+4,129
320 POKE B+4,128    : NEXT
1000 REM INCLUDE LIBRARY HERE
```

Lines 20 and 30 initialise the program and the 'cruising' loop runs between lines 310 and 320. No ADSR values are needed as the sound switches on and off so quickly. We have started the 'in-flight' section at line 300 so that the complete program can be built up in stages–we have to fit in the taxying and take-off before the cruising!

The first part of the journey will now be developed. Let us plan that this stage should start with two faulty attempts at 'firing' before the engine 'catches', with pauses between. Then we need to establish a taxying section followed by the acceleration of the engine ready for take-off. Perhaps it will come as a surprise to find that the speed of the engine is not produced by the frequency of the white noise used, but by the speed with which the separate on/off cycles occur. In the cruising part of the journey there was no time delay between triggering the sound (using value 129) and releasing it (using 128)–this then is the maximum engine speed possible. By putting a very short time delay in the trigger cycle we can control the engine speed.

In Program 11.2, line 180 is very important in controlling the engine speed–it is a dummy loop and so fills in time. However, as the program runs round the loop between lines 170 and 190 one thousand times, the value TM gets smaller so that the time delay reduces and the engine speeds up for take-off.

Program 11.2

```
10 REM 11.2  AIR JOURNEY - TAKE OFF
20 GOSUB 1000
30 POKE B+24,15     : POKE B+1,255
100 REM TAKE OFF
110 FOR K=1 TO 3    : FOR I=1 TO 6*K
120 T=4             : GOSUB 1100
130 POKE B+4,129    : POKE B+4,128 :NEXT
140 IF K=3 THEN 160
150 T=120           : GOSUB 1100    :NEXT
160 TM=20
170 FOR I=1 TO 800 : TM=TM-.02
180 FOR T=0 TO TM  : NEXT
190 POKE B+4,129    : POKE B+4,128 :NEXT
1000 REM INCLUDE LIBRARY HERE
```

The three attempts at firing the engine, two of them doomed to failure, are in the loop between the lines 110 and 150. The time delay in line 120 keeps the engine speed low while the longer delay in line 150 is for the time between attempted firings. The I loop which runs from line 110 is made to increase for each attempted firing so that they last for progressively longer periods. Line 140 is used to escape from the attempted firings and move to the taxying and take-off stage started at line 160.

So far we have started the engine, taxied, taken off and cruised. Sadly the fuel is now running out. This has two effects. First the engine cuts back slightly in its speed and then begins to cut out completely so that only bursts of engine noise are heard. Program 11.3 shows how this can be achieved.

Program 11.3

```
10 REM 11.3  AIR JOURNEY - PANIC
20 GOSUB 1000
30 POKE B+24,15     : POKE B+1,255
400 REM PANIC
```

```
410 FOR I=1 TO 800
420 FOR T=0 TO I/60: NEXT
430 POKE B+4,129    : POKE B+4,128 :NEXT
440 FOR K=3 TO 1 STEP -1
450 FOR I=1 TO 10*K
460 FOR T=0 TO 8    : NEXT
470 POKE B+4,129    : POKE B+4,128 :NEXT
480 T=80            : GOSUB 1100
490 NEXT
1000 REM INCLUDE LIBRARY HERE
```

The loop running from line 410 to line 430 sets up 800 more triggers of the engine. However, the time delay built into line 420 becomes longer at each cycle of the I loop so that the engine slows gradually. The K loop between lines 440 and 490 is to allow for three engine failures. The delay in line 480 creates the silent time when the engine has cut out. Notice how as K gets smaller, the I loop also reduces so that each burst of engine noise lasts for a shorter time.

Eventually the engine cuts out for the last time and the plane begins to fall to earth. Initially there is no sound, but after a short time the plane is moving so fast that you can hear it falling! This has been accomplished in Program 11.4 by having a high pitched whistle getting louder and also falling in frequency.

Program 11.4

```
10 REM 11.4  AIR JOURNEY - DIVE
20 GOSUB 1000
30 POKE B+24,15     : POKE B+1,255
500 REM DIVE
510 A=15   : S=15   : GOSUB 1200
520 POKE B+4,17     : T=0
530 SF=3000         : FF=1500
540 DF=-8           : GOSUB 2100
1000 REM INCLUDE LIBRARY HERE
```

To have the sound slowly increase in volume requires a very slow attack rate. A look at the envelope rates shown in Appendix 5 shows that an attack of 15, the maximum, takes 8 seconds to reach a maximum; this attack value is set at line 510. To decrease the pitch of the note when the plane is falling, subroutine 2100 from the library has been used. Finally, to obtain the whistle sound a triangle waveform, 17 has been set in register 4.

For the last tragic episode of this sound story there is an explosive crash to program. When the plane hits earth, there is a fast attack of sound which is sustained for a while until it falls away fairly quickly. This suggests the following ADSR values

A=2 D=4 S=15 R=11

The explosion is programmed by using white noise, which is triggered in line 620.

Program 11.5

```
1 REM 11.5  AIR JOURNEY - CRASH
20 GOSUB 1000
30 POKE B+24,15     : POKE B+1,255
600 REM CRASH
610 A=2 : D=4 : S=15 : R=11: GOSUB 1200
620 POKE B+4,129    : T=3
630 GOSUB 1100      : POKE B+4,128
640 T=100           : GOSUB 1100
650 POKE B+24,0     : END
1000 REM INCLUDE LIBRARY HERE
```

As the last sounds of the unfortunate plane die away, the sound is switched off in line 650. Each of the segments of the story has been developed independently as a separate, but complete, program. To hear the complete story of early flight, they need to be incorporated into one program. To make it easy for you we are presenting the complete sound story of the air journey as a single program:

Program 11.6

```
10 REM 11.6  COMPLETE AIR JOURNEY
20 GOSUB 1000
30 POKE B+24,15     : POKE B+1,255
90 GOTO 400
100 REM TAKE OFF
110 FOR K=1 TO 3    : FOR I=1 TO 6*K
120 T=4             : GOSUB 1100
130 POKE B+4,129    : POKE B+4,128 :NEXT
140 IF K=3 THEN 160
150 T=120           : GOSUB 1100    :NEXT
160 TM=20
170 FOR I=1 TO 500 : TM=TM-.04
180 FOR T=0 TO TM   : NEXT
190 POKE B+4,129    : POKE B+4,128 :NEXT
300 REM IN FLIGHT
310 FOR I=1 TO 800 : POKE B+4,129
320 POKE B+4,128    : NEXT
400 REM PANIC
410 FOR I=1 TO 800
420 FOR T=0 TO I/60: NEXT
430 POKE B+4,129    : POKE B+4,128 :NEXT
440 FOR K=3 TO 1 STEP -1
450 FOR I=1 TO 10*K
460 FOR T=0 TO 8    : NEXT
470 POKE B+4,129    : POKE B+4,128 :NEXT
480 T=80            : GOSUB 1100
490 NEXT
500 REM DIVE
510 A=15   : S=15   : GOSUB 1200
520 POKE B+4,17     : T=0
530 SF=3000         : FF=1500
540 DF=-8           : GOSUB 2100
600 REM CRASH
610 A=2 : D=4 : S=15 : R=11: GOSUB 1200
620 POKE B+4,129    : T=3
630 GOSUB 1100      : POKE B+4,128
```

```
640 T=100              : GOSUB 1100
650 POKE B+24,0        : END
1000 REM INCLUDE LIBRARY HERE
```

Today

Now for a leap forward in time, to the noisy aeroplanes of today–powered by jet engines. The sound of a jet aeroplane flying overhead has two parts–the approach and the (generally noisy) retreat. Program 11.7 achieves the approach and retreat effects by a mixture of two separate features.

First, the sound volume increases slowly from nothing to a peak and then reduces again fairly gradually. This suggests an ADSR setting of

A = 15 S = 15 R = 14

The values are set in line 30.

Program 11.7

```
10 REM 11.7  JET
20 GOSUB 1000       : POKE B+24,15
30 A=15  : S=15     : R=14  : GOSUB 1200
40 POKE B+4,129
50 X=400 : Y=3900 : Z=25  : T=0
60 SF= X : FF= Y  : DF= Z : GOSUB 2100
70 SF= Y : FF= X  : DF=-Z : GOSUB 2100
80 POKE B+4,128   : T=900 : GOSUB 1100
90 POKE B+24,0    : END
1000 REM INCLUDE LIBRARY HERE
```

Secondly, the frequency of the sound changes during the journey–it rises and then falls. As the library contains a frequency sweep subroutine, this is used to vary the pitch of the sound. As the movement of frequency is in both upward and downward directions, the subroutine is used twice: in lines 60 and 70. Notice that the frequency values are stored as variables–it makes the program look neater.

Tomorrow

If you have not been deafened by the last section, you might like to set up the sound of a rocket being launched. Unlike the previous sound, which had a slow build-up of volume and then a slow decline, a rocket 'explodes' into its sound. It falls back rapidly to what is still a very loud noise which lasts for some time until it eventually fades. This suggests an ADSR setting of

A=2 D=0 S=15 R=14

A further component of the sounds which have been included in Program 11.8 is a change of frequency while the sound is playing. To create a feeling of turbulence, this has been repeated fifteen times using the loop started in line 50. As with the jet sound, we shall use frequency sweep subroutine 2100 but this time only upwards–the rocket should only have a one-way flight!

Program 11.8

```
10 REM 11.8  ROCKET
20 GOSUB 1000     : POKE B+24,15
30 A=2    : D=0   : S=15  : R=14
40 GOSUB 1200     : POKE B+4,129
50 FOR J=1 TO 15 : T=15/J
60 SF=40 : FF=50 : DF=2  : GOSUB 2100
70 NEXT
80 POKE B+4,128  : T=900 : GOSUB 1100
90 POKE B+24,0   : END
1000 REM INCLUDE LIBRARY HERE
```

Once the rocket has reached space, silence should reign. If, however, we meet up with any extra-terrestrials, there is a whole galaxy of sound effects you can use to describe the meeting. These are developed in Chapter 16.

12

Ringing Tones

After the potted sound history of flight, we can now sit back and listen to some very pretty sounds–ranging from the sound of a tapped wine glass, through bells and gongs, to the telephone. The method used to obtain some of these sounds requires a new technique called ring modulation. Perhaps we can leave the 'how it works' description until you have listened to a sample of the sound.

Type in Program 12.1 . When you run the program, you will hear a very realistic gong sound. Some of the register values used in the program should be quite familiar to you, but others may look rather strange. Let us start with the easier parts.

Program 12.1

```
10 REM 12.1  GONG
20 GOSUB 1000     : POKE B+15,20
30 D=12           : GOSUB 1200
40 POKE B+24,15   : POKE  B+4,21
50 POKE B+1,30    : T=155 : GOSUB 1100
60 POKE B+4,20    : GOSUB 1100
70 POKE B+24,0
80 END
1000 REM INCLUDE LIBRARY HERE
```

The gong is a percussion instrument, so in designing the ADSR settings we need a very fast attack, as in all struck objects. However, an important feature of a gong sound is that it continues to reverberate for some time after the initial blow–in other words there is a slow decay. It follows that we need an ADSR setting as follows:

A=0 D=12 S=0 R=0

The ADSR setting is made in line 30. The delay built into line 50 is to allow the decay time to fall to zero. Now to explain how the reverberating gong sound is produced.

When we were looking at the ways separate sounds interact in Chapter 5, we noted the spectacular beat and ghost effects. We also saw in Chapter 6 how different waveforms are produced as a result of combining harmonics of a fundamental frequency in certain proportions. What results when a gong or bell is struck is also a mixture of frequencies, but in a much more complex way than those noted earlier. The phenomenon is known as ring modulation–for obvious reasons. It is produced by taking the sounds from two oscillators and producing two tones which are the sum and difference of the originals. The original two notes do not sound and then react in the ways described in the earlier chapters–they are mixed in the synthesiser to produce the ring modulation effect.

The way this may be made to work on the Commodore is quite simple. The two sound oscillators we have used are in Voices 1 and 3. In Voice 3 only the frequency needs to be set, both waveform and ADSR can be omitted. In line 20 of the program we have set register 15, part of the frequency register of Voice 3, to a value of 20 (check Appendix 3 for SID's register addresses).

In Voice 1 we use the triangle waveform which is normally triggered by poking 17 into register 4. By adding 4 to this value, we can also establish the ring modulation feature. In the bit map printed in Appendix 4, ring modulation is in the third (from the right) location of register 4. Placing a one in this position is equivalent to adding a 4 to the number as the registers operate using the binary system. So using

```
POKE B+4,21
```

in line 40 triggers ring modulation so that the frequency output from Voice 1 interacts with that from Voice 3. The frequency for Voice 1 is set in line 50.

The sound is more interesting if it is repeated. Add the following two lines to the program to repeat the sound 12 times:

```
35 FOR I=1 TO 12
65 NEXT I
```

Shortly we shall provide a 'development' program to enable you to hear a wide range of ring modulation effects, but you might like to try these two effects first. Change lines 20 and 50 as follows:

```
20 GOSUB 1000 : POKE B+15,10
50 POKE B+1,255 : T=155 : GOSUB 1100
```

and your 'grandfather clock' will strike 12 o'clock!

Change lines 20 and 50 as follows:

```
20 GOSUB 1000 : POKE B+15,40
50 POKE B+1,240 : T=155 : GOSUB 1100
```

and your 'carriage clock' will also strike 12 o'clock!

Now you should listen to the variety of different effects that ring modulation can produce. Keeping the frequency setting of Voice 3 constant at 10, we can arrange for the frequency of Voice 1 to vary by using a loop. Type in Program 12.2.

Program 12.2

```
10 REM 12.2  RING MODULATION
20 GOSUB 1000     : POKE B+15,10
30 D=12           : GOSUB 1200
35 FOR I=1 TO 255: PRINT I
40 POKE B+24,15   : POKE  B+4,21
50 POKE B+1,I     : T=155 : GOSUB 1100
60 POKE B+4,20    : GOSUB 1100
65 NEXT
70 POKE B+24,0
80 END
1000 REM INCLUDE LIBRARY HERE
```

Note that the only changes from the original program are the loop running between lines 35 and 65 and the frequency set in line 50. Running the program will take quite a long time as there are 255 different effects! Notice that the loop counter I is printed on the screen so that if you hear a particularly pleasant, unusual or mysterious sound you can find out what produced it!

The 255 effects so far all relate to a frequency register setting of 10 in Voice 3. To do full justice to ring modulation, you should change this to other values and again run Voice 1 through its loop. Altogether there are 255×255, that is over 65000 possible combinations!

Telephone

The number of different telephone sounds is quite large, ranging from bell-like sounds to the more electronic warble sounds. To make the bell-like sound, the use of ring modulation is recommended. What we shall develop here in Program 12.3 is the warble telephone sound; the final sound produced is very realistic and may cause some reaction from members of your family!

Program 12.3

```
10 REM 12.3  TELEPHONE
20 GOSUB 1000      : POKE B+24,15
30 S=15            : GOSUB 1200
40 POKE B+4,17     : FOR K=1 TO 5
50 FOR J=1 TO 2    : FOR I=1 TO 10
60 FOR V=250 TO 0 STEP -50
70 POKE B+1,V      : NEXT
80 NEXT
90 T=15            : GOSUB 1100
100 NEXT J
110 T=60           : GOSUB 1100
120 NEXT K
130 POKE B+24,0    : END
1000 REM PUT LIBRARY HERE
```

The basic warble sound can be produced by a fast slide in frequency in a sustained note. As the note is to be sustained, we simply need to trigger it and then not release it, hence the ADSR values

A=0 D=0 S=15

are required. These are set in line 30. To obtain the frequency slide, you might think that the subroutine 2100 should be used. However, as that subroutine is relatively slow, mainly because it calls a separate subroutine to perform calculations, it will not produce the required sound. To produce a really fast change of frequency, actual frequencies will not be used but just the high frequency register, which can take values of between 0 and 255. The loop between line 60 and line 70 changes the value of V from 250 to 0, in reducing steps of 50.

Having decided how to produce the basic warble sound, all that is now required is to decide how long to play it for–this can be adjusted using the I loop starting in line 50. The telephone program has been arranged to repeat the basic ringing tone 5 times using the loop at line 40; the delay at line 110 is used to create a one-second (60 jiffies) delay between the rings. Finally each warble has been divided into two parts using the loop line 50 and the fairly short delay at line 90.

The waveform used to produce the sound is the triangle. If you change the 17 to 33 in line 40, to produce a sawtooth waveform, the telephone sound produced is slightly different, but still very convincing. The sound is turned off at line 130.

After the rather pleasant ringing tones of this chapter it is now time to embark on a less relaxing sound resource–noise!

13
Noisy Noise

Bang, bang, bang!

Any explosive sound requires a very fast attack and white noise. However, the variations of sound which can be produced by using different frequencies are very large. Program 13.1 has an 'instant' attack (the attack value of zero in fact takes 2 milliseconds–see Appendix 5) so that the significant part of the ADSR cycle is the decay. It has been set at 9; when you experiment with other values you will find that this setting is quite critical if an explosive sound, rather than a damp squib, is to be produced.

Program 13.1

```
10 REM 13.1  SHOT
20 GOSUB 1000     : POKE B+1,18
30 D=9            : GOSUB 1200
40 POKE B+24,15   : POKE  B+4,129
50 T=120          : GOSUB 1100
60 POKE B+24,0    : END
1000 REM INCLUDE LIBRARY HERE
```

The biggest problem is to know what frequency to set in SID's registers. The value in the program is about 280 cycles per second which Appendix 1 shows requires a setting in register 1 of 18. A two-second time delay is included in line 50 so that the decay has a chance to operate–without the delay the volume would be switched off and nothing would be heard. By changing the frequency a wide range of different sounds may be obtained, from revolvers to rifles and even larger weapons.

A hammering effect is obtained by repeating the 'explosions'. This can be done by including a loop to repeat the bangs. However, other changes need to be made to the program because each bang needs to be triggered within the loop. This is normally done by setting register 4 to the value 129; this will only work if the trigger has been released by poking 128 into register 4 beforehand. Program 13.2 incorporates all these points.

Program 13.2

```
10 REM 13.2  HAMMER
20 GOSUB 1000     : POKE B+1,80
30 D=9            : GOSUB 1200
40 FOR I=1 TO 10
50 POKE B+24,15   : POKE  B+4,129
60 T=40           : GOSUB 1100
70 POKE B+4,128
80 NEXT
90 POKE B+24,0    : END
1000 REM INCLUDE LIBRARY HERE
```

Notice that a much higher frequency is now being used–about 1200 cycles per second–and the time delay value T has been reduced to 40. The loop arranges for 10 bangs to be sounded before the volume is switched off. The sound is a very realistic hammering effect which, if your TV volume is set high, will make you sit up! It is more like a percussion riveting hammer than the light tapping of a tack!

This latest program can be quite useful as a development tool to investigate the sounds it is possible to obtain with different frequencies. Simply add this line:

```
45 POKE B+1,I : PRINT I
```

and change line 40 to

```
40 FOR I = 1 TO 250 STEP 5
```

and you will be able to hear the range of different percussion effects as the frequency is changed. Check with the screen to see what setting for register 1 produces each sound.

Thunder

The sound of thunder varies from a low distant rumble to a positive 'end of the world' boom, with lots of variation in between. We shall aim at a sound somewhere between the extremes in Program 13.3 .

We want to explode into sound and obviously use white noise–thunder is certainly not music! After the initial crack, the sound continues until it eventually fades away. This suggests an ADSR setting of

A = 0 D = 0 S = 15 R = 15

Program 13.3

```
10 REM 13.3  THUNDER
20 GOSUB 1000
30 S =15 : R=15   : GOSUB 1200
40 F=40           : GOSUB 1300
50 POKE B+1,H     : POKE B,L
60 POKE B+4,129
70 T=100          : GOSUB 1100
80 POKE B+4,128   : T=10
90 SV=15 : FV=0   : DV=-1
100 GOSUB 2000    : POKE B+24,0
110 END
1000 REM INCLUDE LIBRARY HERE
```

We have used a very low frequency: 40 cycles per second. This is set in line 50. An approximate two-second time delay is used (in line 70) to allow the sound to attack and sustain for a while. The sound is then released in line 80 and using subroutine 2000, the volume reduces gradually to zero.

Wind

To produce the sound of wind we shall once again call upon the richest source of sound effects – white noise. It is much easier to produce a strong wind than a light wind, as you will see (and hear!). To try to stop your roof from blowing off, we shall keep SID's volume in register 24 at about half its maximum setting.

A strong attack is not required, but as the sound should play over a reasonable length of time, it needs to be sustained. This suggests an ADSR setting of

A=11 D=8 S=8 R=0

Program 13.4

```
10 REM 13.4  STRONG WIND
20 GOSUB 1000     : POKE B+24,8
30 A=11  : D=8    : S=8    : GOSUB 1200
40 POKE B+4,129   : T=3
50 FOR J=1 TO 3
60 SF=40 : FF=70 : DF= 1 : GOSUB 2100
70 SF=70 : FF=40 : DF=-1 : GOSUB 2100
80 NEXT
90 POKE B+24,0    : END
1000 REM INCLUDE LIBRARY HERE
```

In order to produce the turbulence of the rushing wind, the frequency of the white noise will be swept. This is done 3 times using the loop set up in line 50. The library subroutine 2100 is used to sweep the frequency up and down while the sound continues.

After suffering the assorted noises of this chapter, it will come as a relief to take a trip to the jungle and meet the fairly strange animals which inhabit it.

14
A Jungle of Sounds

In the same way as when you play music you put in small features that make your playing unique, when you listen to natural sounds they rarely sound the same when they are repeated. When waves break on the sea shore, they have different volumes, different times between waves, and so on. With animals this is just as true. So to make realistic sound effects related to animal sounds, it is necessary to build in some random elements.

In this chapter we set up a wide variety of animal and insect sound effects. Note that many of them use the subroutine library developed in Chapter 10, so you should develop your programs with the library already loaded into the computer's memory. Some of the sounds are very close to the natural ones–you will feel quite itchy when listening to the mosquito! Others need the addition of further synthesiser techniques like the use of the filter, to be discussed in Chapter 16. Many of the sounds produced here have a fun element to them as you will see. Enjoy the varied sounds from our menagerie.

Frog

The variety of sounds made by frogs is quite large; the one developed here has a fairly deep fruity sound. The main technique used to produce the sound of a croaking frog is to arrange the very fast repeated playing of a note, but with a slight delay between each one. Program 14.1 shows how this can be done.

Program 14.1

```
10 REM 14.1  FROG
20 GOSUB  1000
30 POKE B+24,15
100 FOR K=1 TO 2    : FOR I=1 TO 10
110 T=0     : GOSUB 1100
120 POKE B+1,25+I   : POKE B+4,17
130 POKE B+4,16     : NEXT
140 T=10    : GOSUB 1100
150 NEXT
160 POKE B+24,0     : END
1000 REM INCLUDE LIBRARY HERE
```

Although the delay set in line 110 is zero, the time taken to go to subroutine 1100, do nothing and return occupies just enough time to have an effect. Here, Voice 1 is turned on in line 120 and turned off in line 130. This obviously takes very little time. The ADSR settings have not been set in the program and are thus all zero–this means there is extremely rapid attack and decay and only a minute sample of sound is heard. When these 'bits' of sound are played rapidly together, the resulting sound is very much like the croaking of a frog.

In the program there are just two croaks of the frog arranged by the K loop in line 100, with a small time delay (line 140) between them. Obviously the time delay could be increased; by putting T = 60 in line 140 a delay of one second will occur. However, if you want the frog to play for a considerable length of time, then, as described before, you should introduce some random elements into the sound. The simplest way to do this is by altering the time delay between the croaks. Changing line 140 to

```
140 T=60 + 120*RND(0) : GOSUB 1100
```

will do this. As RND(0) produces a value between 0 and 1, multiplying by 120 will create a number between 0 and 120. By adding 60, this increases to the range 60 to 180 which in subroutine 1100, using the jiffy clock, produces variable pauses between one and three seconds long.

Affectionate frogs

When we listened to the croaking of the frog we thought that he sounded very lonely. As his croaking was bound to sound attractive to some other (female) frog, we decided to answer his call. In Program 14.2 we have included a reply!

Program 14.2

```
10 REM 14.2  FROGS
20 GOSUB  1000
100 REM ALONE
110 FOR L=1 TO 2
120 HF=25  : V=15  : GOSUB 500
130 T=240  : GOSUB 1100
140 NEXT
200 REM ANSWER
210 FOR L=1 TO 7
220 HF=25  : V=15  : GOSUB 500
230 T=240-30*L     : GOSUB 1100
240 HF=55  : V=2*L : GOSUB 500
250 T=240-10*L     : GOSUB 1100
260 NEXT
300 REM TOGETHER
310 FOR L=1 TO 10
320 HF=25  : GOSUB 500
330 HF=55  : GOSUB 500
340 NEXT
350 END
500 POKE B+24,V
510 FOR K=1 TO 2   : FOR I=1 TO 10
520 T=0    : GOSUB 1100
530 POKE B+1,HF+I  : POKE B+4,17
540 POKE B+4,16    : NEXT I
550 T=10   : GOSUB 1100
560 NEXT   : RETURN
1000 REM INCLUDE LIBRARY HERE
```

Now the main feature we had to ensure was that the female frog would be quite distinct from the original. This has been achieved by setting the frequency of the note to a higher level, 55–the original was 25 in the high frequency location, register 1.

We could not resist telling a story, especially one with a romantic flavour, so we designed the following sequence. First the original frog should croak alone a few times. Then the reply should be heard–faintly. Then, as they find their calls so attractive, they should get progressively closer; actually the female frog does all the walking (or should it be hopping!) to get to the original frog. Finally they croak together in unison.

This story could use up a great number of lines of program code, but by using the subroutine device we can write it in a neat way and keep it short. Basically we can set up a subroutine for each croak, whether male or female. What needs to be changed for each one is the pitch of the note and the volume at which the sound is to be played. The subroutine in line 500 needs an HF value (25 for the male and 55 for the female) for frequency register 1 and a volume setting V for register 24.

Lines 110 to 140 produce two male croaks with a four-second delay between them. In the second part, starting at line 210, there are seven cycles where the times between responses are reduced as loop counter L increases. For the male, the volume setting is permanently the maximum 15, for the female it increases each time round the loop (line 240). Finally they are together in line 310 and sing in harmony. Who said computers could not be romantic?

Jumpy insects

You may be quite surprised to find that unlike the real world, we can change one animal into another by using the computer. What we shall now do in Program 14.3 is change the frog into a grasshopper!

Program 14.3

```
10 REM  14.3  HOPPER
20 GOSUB 1000          : POKE B+24,15
30 FOR K=1 TO  10      : FOR I=1 TO 20
40 POKE B+1,220+I      : POKE B+4,33
50 POKE B+ 4,32        : NEXT
60 T=10*RND(0)+5       : GOSUB 1100
70 NEXT
80 POKE B+24,0         : END
1000 REM INCLUDE LIBRARY HERE
```

There are some minor changes required in the original program, of course; for example, the waveform is changed to sawtooth and the frequency is set with a base of 220 in register 1. However, the basic principle of repeatedly playing very small extracts of sound is used successfully to produce the sound of a grasshopper.

The random element in line 60 has been added to provide more realism. By changing the frequency base and the number of cycles in the I loop in line 30 you can further extend this basic idea and obtain the sounds of crickets and other insects.

Buzzing insects

It seems that quite a lot of different sounds can be obtained by repeatedly switching a waveform on and off. However, the method needed to obtain an insect buzz is to use a pulse waveform, first introduced in Chapter 9. In Program 14.4, the pulse width has been set very low which means that only a small amount of each cycle is set high. When played at about 500 cycles per second the sound obtained is a buzz.

Program 14.4

```
10 REM  14.4  MOSQUITO
20 GOSUB 1000          : POKE B+24,15
30 POKE B+2,80         : POKE B+5,255
```

```
40 JF=500
50 FOR I=1 TO 10      : POKE B+4,65
60 FOR K=1 TO 8
70 EF=JF+20+RND(0)*40
80 SF=JF    : FF=EF    : DF=2 : T=0
90 GOSUB 2100
100 JF=EF-20-RND(0)*40
110 SF=EF   : FF=JF    : DF=-2: T=0
120 GOSUB 2100         : NEXT
130 POKE B+4,64        : NEXT
140 POKE B+24,0        : END
1000 REM INCLUDE LIBRARY HERE
```

Having established the basic building block for the buzz sound, we must now try to set it within a framework which will produce a realistic sound effect. If you have ever had the misfortune of trying to sleep in the presence of a mosquito, you will know that the sound appears to change slightly in pitch and also that its volume fluctuates considerably–silence meaning either that it has found somebody else to pester or that it is tapping into your vital juices!

It is possible to incorporate both frequency and volume changes within a program. To vary the volume a slow attack and decay have been set in register 5; they are both 15–the maximum possible. Hence the sound will gradually get louder then softer, and as this whole cycle is repeated each time the waveform is triggered in line 50, the effect is simply a continuously changing volume.

The pitch control needs to be more subtle. If it is programmed to rise and fall in a regular manner, the effect is likely to be more like a siren. What we have done in the program is to vary the ranges of the increases and decreases of pitch using random numbers. This sound effect is extremely successful–just as you think the mosquito has disappeared, it comes back to torment you!

Cat

If you are feeling rather itchy after listening to the sound of the mosquito, relax to the miaowing of our cat. Program 14.5 uses the sawtooth waveform, 33 in line 100 and the cat sound is made by using

frequency sweep subroutine 2100 in a falling direction. The fall is quite small – from 880 to 873 cycles per second.

Program 14.5

```
10 REM 14.5  CAT
20 GOSUB  1000      : POKE B+24,15
30 A=13  :D=9 :S=10: R=9 : GOSUB 1200
100 POKE B+4,33
110 T=6  : SF=880  : FF=873
120 DF=-1: GOSUB 2100
130 POKE B+4,32
140 T=45 : GOSUB 1100
150 POKE B+24,0     : END
1000 REM INCLUDE LIBRARY HERE
```

In the same way as was done with the frog – repetition can be built in and also some random elements. This could be in the time delays and also in the frequencies used.

Snake

We do not normally consider the snake to be one of creation's noisiest animals! What we have arranged in Program 14.6 is five 'bursts' of white noise to produce the hissing sound. The frequency setting in register 1 is set at its highest level, 255.

Program 14.6

```
10 REM 14.6  SNAKE
20 GOSUB 1000        : POKE B+24,15
30 A=15 : D=8 : S=4 : R=9 : GOSUB 1200
40 POKE B+1,255
100 FOR I=1 TO 5     : POKE B+4,129
110 T=60             : GOSUB 1100
120 POKE B+4,128
130 T=180 : GOSUB 1100
140 NEXT
```

```
150 POKE B+24,0      : END
1000 REM INCLUDE LIBRARY HERE
```

To keep the sound quiet we have used the slowest attack rate, 15. A check with Appendix 5 will show that the attack takes 8 seconds to reach a peak; in line 110 we only use the first second before cutting off the sound in line 120–the sound is fairly quiet. It is better to use the ADSR settings to control volume rather than the rather crude volume setting in register 24 which only has 15 separate steps, is not smooth and controls all three of SID's voices. To make the sound of the snake more 'hissy' you will need to use a filter.

We have added an extra dimension to the snake sound by developing Program 14.7. Here a hypnotic tune is played on a flute to charm the snake.

Program 14.7

```
10 REM 14.7  CHARMING SNAKE
20 DIM F(24),D(24)
30 FOR I=1 TO 24 : READ F(I),D(I) :NEXT
40 GOSUB 1000    : POKE B+24,15
50 A=14 : S=15   : R=2 : GOSUB 1200
60 POKE B+12,0   : POKE B+1,255
70 POKE B+13,15*4: POKE B+11,33
80 C=1
100 FOR J=1TO 2
110 FOR I=1 TO 24: C=C+1
120 F=F(I)       : GOSUB 1300
130 POKE B+7,L   : POKE B+8,H
140 T=D(I)*5     : GOSUB 1100
150 IF C=8 THEN POKE B+4,128 : C=1
160 IF C=7 THEN POKE B+4,129
170 NEXT         : NEXT
180 POKE B+24,0  : END
500 DATA 466,4,494,4,622,4,659,4
510 DATA 784,1,831,1,784,4,659,4
520 DATA 622,1,659,4,622,4,554,4
530 DATA 466,4,494,4,  0,4,466,4
540 DATA 494,4,587,4,622,4,740,1
```

```
550 DATA 784,1,740,4,659,4,622,6
1000 REM INCLUDE LIBRARY HERE
```

The note frequencies and durations are stored in arrays F and D, using ideas developed in earlier chapters. The values are read in and stored in line 30 so that the notes can be played in Voice 2. The triangle waveform is used for the tune, and line 70 sets the sustain level in register 13. The hissing sound for the snake is set in line 160 to occur after every 7 notes using counter C. After this trip to the mystic East, we shall extend our journey–this time to a land of fantasy.

Dragon

One good aspect of making a fire-breathing dragon is that nobody can compare the sound with a real one! To produce a dragon sound in Program 14.8, we have again used white noise (129 in register 4) but with more time (2 seconds in line 120) to allow the volume to build up. Also we have lowered the pitch of the sound by using a setting of 150 in register 1.

Program 14.8

```
10 REM 14.8  DRAGON
20 GOSUB  1000        : POKE B+24,15
30 A=12  : D=10       : GOSUB 1200
40 POKE B+1,150
100 FOR I=1 TO 5      : POKE B+4,129
110 T=120             : GOSUB 1100
120 POKE B+4,128
130 T=45 : GOSUB 1100
140 NEXT
150 POKE B+24,0       : END
1000 REM INCLUDE LIBRARY HERE
```

By changing the ADSR setting in line 30, you can arrange to make the dragon either more aggressive (smaller values for A in line 30) or more relaxed (larger values). The program allows for 5 bursts of dragon fire by using the I loop set up in line 100. Stand well back!

Wolf

The cry of the wild can be heard in Program 14.9. We have arranged two wolf cries using the loop set up in line 100. The main cry is produced by a fall in frequency as was used with the cat. However, there are some significant differences. The cry has been divided into two parts–this is to allow for the drop in frequency to increase in the later stage.

Program 14.9

```
10 REM 14.9  WOLF
20 GOSUB 1000          : POKE B+24,15
30 A=2: S=15: R=3      : GOSUB 1200
100 FOR I=1 TO 2
110 POKE B+4,33        : T=0
120 SF=1175 : FF=1125: DF=-2
130 GOSUB 2100
140 SF=1125 : FF=999 : DF=-10
150 GOSUB 2100
160 POKE B+4,32
170 T=60     : GOSUB 1100
180 NEXT
190 POKE B+24,0        : END
1000 REM INCLUDE LIBRARY HERE
```

The frequency sweep subroutine 2100 is initially used to let the sound fall from 1175 to 1125 cycles per second in steps of 2 (line 120) while in the second part (line 140), the frequency falls further to 999 cycles per second but at a faster rate (10 cycles per second).

Birds

After the call of the wild, we shall introduce the more dulcet tones of bird song–one of the most realistic sounds we can produce and one in which a large amount of variation is possible. In Program 14.10

the basic bird chirp has been achieved by a rapid rise in frequency.

Program 14.10

```
10 REM 14.10  BIRDS
20 GOSUB 1000          : POKE B+24,15
30 A=4 : D=4 : S=15 : GOSUB 1200
100 FOR I=1 TO 25
110 POKE B+4,33
120 FOR H=150 TO 200 STEP 3
130 POKE B+1,H         : NEXT
140 POKE B+4,32        : T=3+6*RND(0)
150 GOSUB 1100         : NEXT
180 POKE B+24,0        : END
1000 REM INCLUDE LIBRARY HERE
```

Normally we would use subroutine 2100 to arrange for a frequency sweep. However, that is when the changes in frequency are not too fast – for bird song we want the ability to go through a frequency change at a very fast rate. This has been arranged by just changing the value in the higher frequency register, number 1. The frequency sweep loop is in lines 120 and 130.

To make the sound more realistic we have arranged a variable time pause between chirps–using random numbers in line 140. There are several ways this program can be altered. You could change the H values in line 120, either raising or lowering the range of frequencies. You could allow the pitch to fall instead of rise using a negative step size. Another variation could be produced by having a rapid rise and fall of pitch using two loops.

Owl

We thought this sound so pleasant that it was worth presenting separately. Program 14.11 really sounds like the 'twit-twoo' of an owl with a delightful mellow tone.

Program 14.11

```
10 REM 14.11  OWL
20 GOSUB 1000    : POKE B+24,15
30 S=7           : GOSUB 1200
100 FOR I=1 TO 4
110 F=523        : GOSUB 1300
120 POKE B,L     : POKE B+1,H
130 POKE B+4,17
140 T=1          : GOSUB 1100
150 F=659        : GOSUB 1300
160 POKE B,L     : POKE B+1,H
170 GOSUB 1100   : POKE B+4,16
180 T=24         : GOSUB 1100
190 POKE B+4,17
200 F=659        : GOSUB 1300
210 POKE B,L     : POKE B+1,H
220 T=0          : GOSUB 1100
230 SF=659       : FF=590 :DF=-4
240 T=0          : GOSUB 2100
250 POKE B+4,16
260 T=120        : GOSUB 1100
270 NEXT
280 POKE B+24,0 : END
1000 REM INCLUDE LIBRARY HERE
```

Although a maximum attack and decay have been used, the sustain level in line 30 has been set at about half its maximum setting. The 'twit' part of the sound is achieved by an almost immediate switch of frequencies between 523 cycles per second and 659 cycles per second–lines 110 to 160. The second part of the sound sets up another attack in line 190 and then there is a fall in frequency down to 590 cycles per second using the frequency sweep subroutine 2100.

This chapter has shown how subroutines can be used efficiently and neatly to perform programming tasks. The last program has shown that by breaking a sound into parts, which quickly follow each other, a composite sound can be obtained. The only problem is the amount of program required.

After this trip to the jungle, it is time to return home and listen to some more familiar sounds in the next chapter.

15
Sounds Familiar

Something from nothing

We shall now take a closer look at some sounds closer to home. Generally with sound effects we have to concentrate on which register settings to use; for example we have to select the most suitable frequency, the ADSR values and the waveform. If none of these is set and the volume in register 24 is switched on and off you might have thought that nothing would be heard. However, although in the real world the absence of sound is silence, when using the Commodore this is not quite true–what is heard is a click. This is produced at switch off and is not controllable. Do not forget to load the subroutine library we developed in Chapter 10.

The question arises 'can we exploit this effect?'. In program 15.1 we have set up a loop in line 30 to switch the volume on and off repeatedly. If the loop runs once you hear a click so we might consider changing the upper limit of the loop to 2, 3, 5, 10, 100 and so on.

Program 15.1

```
10 REM 15.1  SNAP
20 GOSUB 1000
30 FOR I=1TO 1
40 POKE B+24,15  : POKE B+24,0
50 NEXT
60 END
1000 REM INCLUDE LIBRARY HERE
```

When the loop runs for 3 cycles, the click is like the snap of a twig, but as we increase the number of cycles, the sound becomes too prolonged for a 'snap'. As the number of cycles is increased to several hundred the sound becomes more like a note, which is simply an example of the V-T effect described in Chapter 2.

A second way in which we can develop the switch-off click effect is by putting a pause between the separate clicks. Program 15.2 is very similar in concept to the previous one except that a delay has now been introduced; the T variable, in line 50, carries the time pause required to the delay subroutine 1100. As this works in jiffies, putting T = 60 produces a sequence of one-second ticks lasting, in total, for 20 seconds.

If T is reduced too far then the program becomes identical to the previous one; however, set at 3 it produces a sound quite like a motor engine noise. If you want to make the engine noise more realistic you will need to increase the number of repetitions contained in the loop at line 30.

Program 15.2

```
10 REM 15.2 CLICK
20 GOSUB 1000
30 FOR I= 1 TO 20
40 POKE B+24,15    : POKE B+24,0
50 T= 60           : GOSUB 1100
60 NEXT
70 END
1000 REM INCLUDE LIBRARY HERE
```

If you would like to make the sound more interesting, consider changing the constant time delay contained in the loop; If you let the delays become progressively longer between the clicks by changing line 50 to

```
50 T=5 + I / 2 : GOSUB 1100
```

then the effect heard is very much like a door creaking.

Although the door creak sound is quite convincing–you can certainly

hear the creaky part of the sound–it could be improved by having more depth or tone in it. We must not forget that it has been produced with minimal effort–just switching the volume off in a certain way. We shall now look at how the extra depth can be given to the sound by using the range of SID's registers we normally use for sound effects.

By using the sawtooth waveform at fairly low frequencies, so that the sound is more granular than a continuous note, we can produce the same kind of clicking as before, but this time with extra tone and depth. Program 15.3 goes straight to the sustained part of the sound at line 40 (using minimum settings for attack and decay) and line 50 (using subroutine 2100) arranges for a downward sweep of low frequencies. The time delay value set in line 40 is also carried over to the frequency sweep subroutine.

Program 15.3

```
10 REM 15.3  CREAK
20 GOSUB 1000     : POKE B+24,15
30 S=15           : GOSUB 1200
40 POKE B+4,33    : T=5
50 SF=23 : FF=5   : DF=-2 : GOSUB 2100
60 POKE B+24,0    : END
1000 REM INCLUDE LIBRARY HERE
```

By a small extension to the program we can make the sound that you hear when you tread on a creaky floorboard. This time you will want to add a rising frequency sweep–this can be done simply by adding the original sweep in reverse and using the same subroutine. So adding this line:

```
55 SF=3 : FF=23 : DF=2 : GOSUB 2100
```

produces the complete floorboard effect. You can now see why subroutines are so useful in writing, developing and extending programs.

Something from everything

Having produced so much from next to nothing in the last section, we can now set about trying to produce some worthwhile results from the 'richest' sound available–white noise. It is rich not in its sound, but in the sense that it contains a complete mixture of frequencies. White noise is undoubtedly our most useful tool when producing sound effects. We have already put it to good use in Chapter 10 to produce the effect of waves breaking on the sea shore, in Chapter 11 to produce the crash, the jet and the rocket and another selection of sound effects in Chapter 13. We shall return to it in Chapter 16 producing a selection of 'space' like sounds. As it is so well represented elsewhere, we shall restrict ourselves to just a few white noise sound effects here–although you can easily develop many more familiar sounds using it.

Program 15.4 creates the effect of somebody walking towards us–obviously a sound which requires an increase in volume. Now the attack part of ADSR can normally be used to increase the volume but it is not suitable here as each footfall requires a separate attack. The volume is increased in the loop running from line 40 to line 80.

Program 15.4

```
10 REM 15.4  STEP FORWARD
20 GOSUB 1000        : POKE B+24,15
30 A=1 : D=8 : R=9 : GOSUB 1200
40 POKE B+1,15       : FOR I=1 TO 15
50 POKE B+24,I       : POKE B+4,129
60 T=48              : GOSUB 1100
70 POKE B+4,128      : T=30
80 GOSUB 1100        : NEXT
90 T=72              : GOSUB 1100
100 POKE B+24,0      : END
1000 REM INCLUDE LIBRARY HERE
```

You will have noticed that it is possible for a sound to become a useful sound effect simply by having its frequency swept or its volume

changed. In Program 15.5, we attempt to produce the sound of a fairly noisy toilet being flushed. As the sound is not explosive we do not need a fast attack value. Also the sound dies away relatively slowly. This suggests an ADSR setting of

A = 11 D = 15 S = 5

which is set up in line 30.

Program 15.5

```
10 REM 15.5  FEELING FLUSHED
20 GOSUB 1000  : POKE B+24,15
30 A=11 : D=15 : S=5  : GOSUB 1200
40 FOR I=0 TO 14 STEP 7
50 POKE B+I+1,18-I/7  : POKE B+I+4,129
60 T=120        : GOSUB 1100
70 NEXT
80 SV=15        : FV=0 : DV=-2 : T=10
90 GOSUB 2000
100 POKE B+24,0         : END
1000 REM INCLUDE LIBRARY HERE
```

The difficulty with the toilet flushing sound is that a 'swirling' effect is required for the running water. We have decided to do this by using all three of SID's voices–not by playing them in unison (which would simply make them louder) but by arranging for them to have different frequencies. This is done within the program by using the loop started in line 40.

You will remember that the 'rule of seven' described how SID's three voices are arranged as three parallel sets of registers (see Appendix 3). For any register in Voice 1, there are equivalent registers in the other two voices, 7 and 14 registers higher. In line 50 of the program, variable I is used to trigger the white noise waveform (129) and three different frequencies in the three voices.

The dying away of the sound at the end of the flush is achieved by using the volume sweep subroutine at line 2000 in the library. The sound is rather dramatic, perhaps most like a fairly ancient toilet. You

could try to 'tame' the sound by reducing the overall maximum volume setting at line 20. Now that you are feeling flushed with success, it is time to present the white noise waveform in small packets.

Drum roll

We have seen how white noise appears to be the key to producing many very convincing sound effects. What we are now going to look at is the making of a side-drum 'tune' or roll, which often accompanies marching. In creating this sound we must, as usual, break it down into its separate parts. The unit of the sound effect is a single roll, which can be obtained by using white noise switched on and off fairly quickly.

If we are going to play a tune then the core of the program will be the playing of the single drum roll for a controlled length of time. As the sound is a sequence of fast repeats, the ADSR cycle needs to have a fast attack, negligible decay and fast release; this suggests the following ADSR values:

A=1 D=15 S=15 R=0

The pitch of a side-drum is quite high, so a high frequency is called for; however, you will find by experimenting that the sound is switched on and off so quickly that the frequency is not critical.

Program 15.6

```
10 REM 15.6  DRUM ROLLS
20 DIM D(15)
30 FOR I=1 TO 15    : READ D(I) : NEXT
40 GOSUB 1000       : POKE B+24,15
50 F=2050           : GOSUB 1300
60 POKE B+1,H       : POKEB,L
70 A=1 : D=15       : S=15 : GOSUB 1200
80 FOR I=1TO 2      : FOR J=1 TO 15
90 FOR K=1 TO D(J)  : POKE B+4,129
100 T=1             : GOSUB 1100
110 POKE B+4,128    : T=2
```

```
120 GOSUB 1100       : NEXT
130 T=3              : GOSUB 1100
140 NEXT
150 NEXT
160 END
500 DATA 4,4,10,4,4,10
510 DATA 4,4,2,2,2,2
520 DATA 4,4,10
1000 REM PUT LIBRARY HERE
```

There are three short delays built into the program. In line 100 the delay is designed to allow the attack to start before being cut off in line 110, while in line 120 it is designed to allow the release to occur. The delay in line 130 is to separate each 'note'.

As there is a sequence of rolls to be played, we need to store the length of the individual rolls using DATA. The rhythm used to make the 'tune', can be worked out by tapping a table with your hands. Let the shortest tap have a length of 1, and work out how long each of the other taps is in relation to the unit. We have seen in earlier chapters that the DATA can either be READ in with each note being played immediately, or the complete set can be stored in an array–which is the method used here.

As an example we have chosen the rhythm

```
4  4  10  4  4  10
4  4   2  2  2   2
4  4  10
```

Try tapping out the rhythm before typing in the program. The note values are stored in array D. The loop between lines 80 and 150 is a repeat for the tune–the whole tune plays twice. The loop from 80 to 140 sets up the playing of 15 notes, the number in the chosen tune. Lines 90 to 120 form the loop which produces the roll; the length of the loop depends on the duration of each note to be played.

After you have tried this drum roll you might like to try a different rhythm. Here is another familiar drum roll:

```
4  2  2  8
4  2  2  8
4  2  2  2  2  2  2
4  4  8
```

As there are 18 'notes' in this tune, you should change the 15 to 18 in lines 20, 30 and 80. Also, of course, you will need to change the original DATA to the above values.

Hubble bubble

As you go into the kitchen and hear the soup boil or indeed into the woods to discover witches assembled around their cauldron, you can hear the bubbling of liquid. At first this may seem like a difficult sound to reproduce–however, it can be done simply by rushing through frequencies. In Program 15.7, the loop in lines 80 and 90 sweeps Voice 1 frequency register very quickly to produce a single bubble sound.

Program 15.7

```
10 REM 15.7  CAULDRON
20 GOSUB 1000     : POKEB+24,15
30 S=15           : GOSUB 1200
40 POKE B+4,17
50 FOR I=1 TO 100
60 POKE B+1,0     : POKE B+4,17
70 FOR J=1 TO 300*RND(0) : NEXT
80 FOR H=0 TO 70 STEP 5
90 POKE B+1,H     : NEXT
100 POKE B+4,16
110 NEXT
120 POKEB+24,0     : END
1000 REM INCLUDE LIBRARY HERE
```

The attack and decay parts of ADSR are not used, except that in line 30 the sustain level is set at the maximum, 15. The loop in line 70 sets up a randomly changing time delay. You may be wondering about setting register 1 to zero at the start of each bubble in line 60; this is necessary because without it the value remaining at the end of the

loop in lines 80 and 90 will spoil the bubble sound. Remove it to see what we mean.

Now to leave the world of familiar sounds and take a leap forward into space!

16

Sounds You Haven't Heard...Yet

In this chapter we start with a fairly traditional technique – that of rapidly changing the frequency of a sound. There is a wide collection of fairly bizarre sounds which can be produced in this way. However, later in the chapter we shall introduce several new techniques which produce some really weird effects – the main problem being to find adequate words to describe them. Do not forget to start by loading the subroutine library developed in Chapter 10, particularly for the early programs in this chapter.

Phasers

It may seem that Program 16.1 is fairly conventional, in that it simply arranges for a fast downward sweep of frequency in register 1. As the range is from 250 to 0, it is almost the complete range of frequencies possible. The large step size of 25 is arranged so that the whole sweep is completed very quickly. We cannot use the frequency sweep subroutine 2100 as the time taken to change frequencies is too long. The basic element of this sound effect is the rapidity of the sweep.

Program 16.1

```
10 REM 16.1  PHASER
20 GOSUB 1000        : POKE B+24,15
30 S=15              : GOSUB 1200
40 POKE B+4,17
50 FOR I=1 TO 20
60 FOR H=250 TO 0 STEP -25
70 POKE B+1,H        : NEXT
80 NEXT
90 POKE B+24,0       : END
1000 REM INCLUDE LIBRARY HERE
```

The loop running between line 50 and line 80 determines the number of phaser shots.

Having developed this fairly short program, you will be pleasantly surprised to discover that it is extremely useful for producing further sounds. Initially we shall consider changing the speed of the sweep. If the step size of −25 in line 60 is changed anywhere in the region from −30 to −10, different phaser weapon sounds can be produced.

For the next stage you should increase the limit of the I loop in line 50 to 100 so that you can hear more of the sound. If the sweep step size in line 60 is changed to −40, the sound is more like a machine gun, while at −50 it becomes very like a telephone. Finally when it is −100, the sound is surprisingly like wood being tapped. When the step size is at this value, the only values set in register 1 (line 70) are 250, 150 and 50, which is a very small sample from the 7 octaves being swept!

If you now decrease the step size in line 60 another range of effects can be heard. Values around −4 or −5 make the sound of a police siren while a value of −3 or −2 will also produce an alarm but with a greater air of urgency. Finally when the step size is changed to −1, the sound is simply of something falling–the same idea was used in Chapter 11 to produce the fall of the aeroplane from the sky.

All these effects have been produced by simply changing one value in the program–the waveform has been left constant at triangle, 17 at line 40. If you change it to 33 for the sawtooth waveform, another collection of sounds can be produced. These are different from those heard earlier so you are recommended to try them. Last, you might like to try the white noise waveform by placing 129 in register 4. A step size of −30 produces a sound something like a road drill, while at −5 some different space weapon sounds can be heard.

Dive, dive, dive

Program 16.1 has been of real value with such a rich selection of sounds. Believe it or not there are still more to come! By simply switching the direction of the sweep in line 60, i.e. setting the phaser in reverse, a new range of sounds can be heard. First make sure that

you are using the triangle waveform (17) in register 4 (line 40). Change line 60 to

```
60 FOR H=0 TO 255 STEP 1
```

and the sound heard is of the alarm used when a submarine has to dive.

Changing the step size to 20 produces another space-weapon sound. Change the waveform to sawtooth and use a step size of 9 to hear yet another weapon. It is hoped that you will not get carried away with all these sounds of weapons. A more exciting range of sounds can be heard in the remaining sections of this chapter.

Arcade

Almost unbelievably, we can exploit the power of Program 16.1 even further! Change the waveform value to 17 in line 40 and change line 60 to

```
60 FOR H=250 TO 60 STEP -30
```

This is similar to the original phaser program. Now include the following two lines:

```
55 FOR H=60 TO 250 STEP 30
56 POKE B+1,H : NEXT
```

What we are doing is producing a rapid upward sweep of frequencies followed immediately by a rapid downward sweep. The sound is one of those built into arcade games.

Once again you can experiment with step sizes in lines 55 and 60 to produce a range of other useful sounds. Similarly, changing the waveform in register 4 (line 40) further extends the range of effects.

Leaps

So far we have used sweeps of frequency in regular steps both up

and down. You might think that this basic concept has been so useful in producing sound effects that even more could be gained by using non-constant jumps in our step size. Program 16.2 allows you to experiment with a varying step size.

Program 16.2

```
10 REM  16.2 WHATS NEW
20 GOSUB 1000          : POKEB+24,15
30 S=15                : GOSUB 1200
40 POKE B+4,17         : T=10
50 FOR I=1 TO 10
60 DV=1                : V= 5
70 POKE B+1,V          : DV=DV+DV
80 V=V+DV : IF V>0 AND V<256 THEN 70
90 GOSUB 1100          : NEXT
100 POKEB+24,0         : END
1000 REM INCLUDE LIBRARY HERE
```

Once again we are only using frequency register 1, but the value V is changing rapidly. Initially the value of DV is 1, but as it is repeatedly added to itself in line 70 it takes values 1, 2, 4, 8, 16, 32, and so on. Listen to the sound to hear the effect.

Frequency modulation

One of our uses for Program 16.1 was to develop a siren sound by sweeping the frequency up and down. In Chapter 9 we looked at frequency modulation, by which a principal wave had its frequency modulated by a second waveform. We used Voice 3 as a source of modulation using the PEEK facility in register 27. In Chapter 9 we used a very small frequency change for the Voice 3 modulating source, because we were producing a vibrato effect; we only wanted to vary the main note by a small amount. What we shall now do in Program 16.3 is to modulate the sound of Voice 1 with a modulating signal varying over a greater range.

Program 16.3

```
10 REM 16.3  FREAKY MODULATION
20 B=54272          : POKE B+24,143
30 POKE B+6,240     : POKE B+4 ,17
40 B3=B+27          : POKE B+18,17
50 X=.05 : TM=0     : IM=200
60 Y=50  : F=2      : G=F*16.777
70 H=INT(G/256)     : L=G-H*256
80 POKE B+14,L      : POKE B+15,H
90 FOR I=1 TO IM
100 POKE B+1,PEEK(B3)*X+Y
110 FOR T=0 TO TM : NEXT
120 NEXT
140 POKE B+24,0     : END
```

The basic sound used for Voice 1 has a triangle waveform (register 4). The modulating waveform in Voice 3 will also be triangular (register 18). You will recall that the shape of the triangle waveform regularly increases and decreases in each cycle. You can hear that the pitch is moving evenly both upwards and downwards to produce a different siren sound.

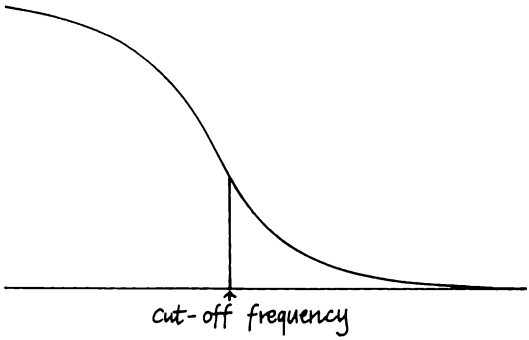
By changing the value in register 18 (line 40) you can set a sawtooth waveform for the modulating signal. As the shape of the sawtooth rises continuously and falls again, at a stroke, the pitch heard moves upwards with a whooping effect–the sound repeatedly rises smoothly and falls abruptly.

Filtering

We also saw in Chapter 6 that a major feature producing the different sounds of musical instruments comes from their mixture of harmonics. It seems reasonable to suppose that changing the contributions of the harmonics would have a considerable effect on the sounds heard. The removal, or at least the significant reduction of the amounts of different frequencies contained in a sound is called filtering.

There are four types of filter available on the Commodore 64. The first, the low-pass filter, while not affecting low frequencies, severely reduces the sound at frequencies above a certain point called the cut-off frequency.

Low-pass filter



Program 16.4 allows you to hear the same note, A4 (440 cycles a second) played a number of times. A low-pass filter has been built in and each time you hear the note, the filter is in a different position; it is moving up in frequency. The position of the filter is set in register 22 which is controlled using the loop starting in line 90.

Program 16.4

```
10 REM  16.4  WHAT A NOTE!
20 B=54272            : FOR I=0 TO 24
30 POKE B+I,0         : NEXT
40 POKE B+6,240
50 POKE B+23,15*16+1: POKE B+24,15+16*1
60 F=440              : G=16.777*F
70 H=INT(G/256)       : L=G-H*256
80 POKE B,L           : POKE B+1,H
90 FOR HF=0 TO 255 STEP 20
100 POKE B+22,HF      : POKE B+4,33
110 FOR T=0 TO 500    : NEXT
120 POKE B+4,32       : NEXT
130 POKE B+24,0       : END
```

As the filter moves progressively across the note being played there is a very noticeable variation in the quality of the sound heard. Although the original note is 440 cycles a second, it must be borne in mind that it contains a whole series of harmonics so that the filtering can have an effect.

The severity of the cut-off, i.e. the speed at which the higher frequencies are attenuated or reduced, is controlled by the resonance built into the filter. Putting this in simple terms, we are talking about the sharpness of the 'knee' in the diagram. Without going into a lot of detail, we shall just set the sharpness at a maximum, so that you can really hear the effect of the filter.

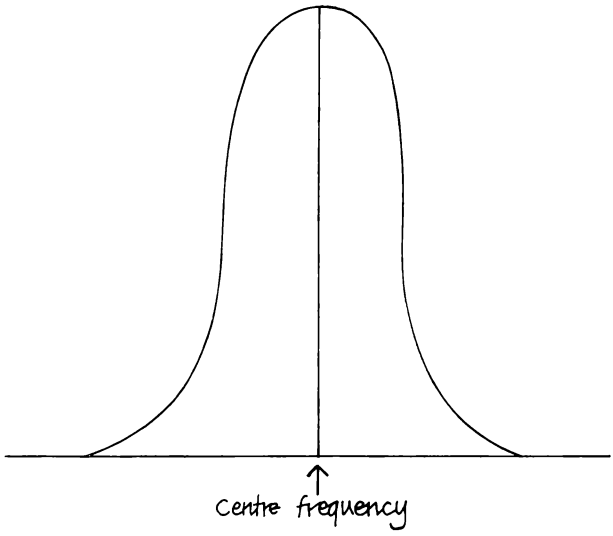
We need to describe briefly some new SID registers. If you look at the SID Bit Map in Appendix 4, you will see that register 23 allows you to set filters on any combination of the three voices. Setting a 1 in register 23 activates a filter on Voice 1, while a setting of 3 will activate a filter on both Voices 1 and 2. The highest 4 bits of the register set the resonance of the filter so that we shall need to add 15*16, that is 240, to the setting of register 23 to achieve a maximum sharpness of the knee. So the value 241 set in register 23 at line 50 produces maximum resonance to the filtering of Voice 1.

To hear the effect of the low-pass filter on other waveforms, change the 33 in register 4 to 17 for triangle and 129 for white noise.

Other filter types are available. The high-pass filter, as might be expected, operates in the reverse way so that lower frequencies are filtered out. To hear the effect of a high-pass filter, set register 24 in line 50 of Program 16.4 to 15 + 16*4 and run the program. To understand how the different types of filter can be set, have another look at Appendix 4. Register 24 has three bits related to filter types. As they are all in the left 'nibble' of the register (each nibble is half the complete eight bit register), they are multiples of 16. So 16*1 set in register 24 activates the low-pass filter while 16*4 sets the high-pass filter.

A third type of filter, called a band-pass filter, operates by leaving central frequencies unchanged but attenuating frequencies outside the interval.

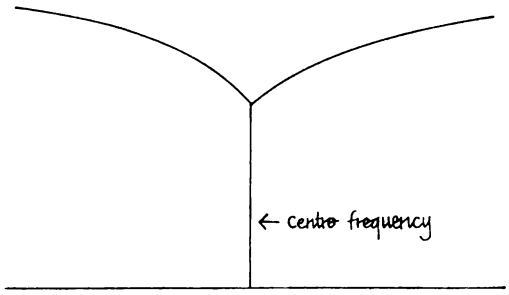
Band-pass filter



The band-pass filter can be heard by placing a 2 in the high nibble of register 24 in line 50.

The final type of filtering possible using the SID chip couples together the high-pass and low-pass filters. By this means only frequencies in the centre are attenuated, while the ranges outside this central band are unchanged. This is called a notch or band-reject filter.

Band-reject filter



As the notch filter is a combination of the high-pass and low-pass filters, you need to activate two of the bits in the high nibble of register 24. Hence the register should be set at 15+16*5 to hear this filter.

Having learnt the theory of filters, it is now time to hear some of the quite bizarre sounds they can produce. Program 16.5 plays a sequence of notes with changing pitch, and as each note is played the filter is rapidly moved upwards across the note.

Program 16.5

```
10 REM  16.5  BOINGY
20 B=54272            : FOR I=0 TO 24
30 POKE B+I,0         : NEXT
40 POKE B+6,240       : POKE B+4,17
50 POKE B+23,15*16+1: POKE B+24,15+16*2
60 FOR H1=0 TO 255  : POKE B+1,H1
70 PRINT "␣"H1
80 FOR HF=0 TO 255 STEP 9 :POKE B+22,HF
90 FOR T=0 TO 1       : NEXT
100 NEXT
110 NEXT
120 POKE B+24,0       : END
```

The program has a low-pass filter set in register 23 at line 50 initially using the triangular waveform set in register 4 (line 40). The filter sweep loop is contained in lines 80 and 100. The sound heard varies dramatically for different basic frequencies. Initially it sounds like a double bass but it later changes to a 'Sparky' sound. There are several ways in which this program can be used.

First, you can change the type of filter by placing different values in register 24. Secondly, you can change the speed at which the filters sweep across each note. This can be done by changing the step size in line 80. A third element that you can alter is the waveform set in line 40. If all this is not enough, try sweeping the filter downwards by changing line 80 to

```
80 FOR HF=255 TO 0 STEP -9 : POKE B+22,HF
```

or upwards and downwards. Finally, a new range of sounds can be made by changing the dummy loop in line 90.

Vice versa

Having heard the variety of sounds that can be heard by sweeping the filter across a note, it seems that there is scope for reversing the process. Program 16.6 sets up the band-pass filter in register 24 and, for different filter frequency settings, sweeps a note across it.

Program 16.6

```
10 REM  16.6  GYBOIN
20 B=54272              : FOR I=0 TO 24
30 POKE B+I,0           : NEXT
40 POKE B+6,240         : POKE B+4,17
50 POKE B+23,15*16+1: POKE B+24,15+16*2
60 FOR HF=0 TO 255   : POKE B+22,HF
70 PRINT "␣"HF
80 FOR H1=0 TO 255 STEP 9 : POKE B+1,H1
90 FOR T=0 TO 5         : NEXT
100 NEXT
110 NEXT
120 POKE B+24,0         : END
```

The sounds produced are very lively and bouncing as you will hear. In the same way as before, you can vary waveform, step size for the sweep, and so on.

Heavenly sounds

We are now ready to introduce the last of our special effects by using a technique called synchronisation. Program 16.7 plays a triangle waveform (17) which is set in register 4 but, as you can see in Appendix 4, the extra 2 which has been added to it in line 50 has the effect of setting up synchronisation with Voice 3.

Program 16.7

```
10 REM  16.7  HEAVENLY
20 B=54272          : FOR I=0 TO 24
30 POKE B+I,0       : NEXT
40 POKE B+6,240     : POKE B+24,15
50 POKE B+4,2+17
60 FOR H3=0 TO 255 : POKE B+15,H3
70 PRINT "␣   "H3
80 FOR H1=0 TO 255 : POKE B+1 ,H1
90 FOR T=0 TO 0     : NEXT
100 NEXT
110 NEXT
120 POKE B+24,0     : END
```

The sounds produced by this program are really strange but quite beautiful – it is not easy to describe them in detail but we think 'heavenly' is quite appropriate. A frequency is set in Voice 3 (line 60) while the loop in lines 80 and 100 sweeps with Voice 1 playing in synchronisation.

As the sounds vary considerably we have arranged to print the frequencies of Voice 3. If you hear a particularly interesting sound you can identify where it came from! As always, there is a wide range of variations to pursue. Try reversing the loop in line 80 to sweep downwards and change the waveform in Voice 1 (line 50).

At the beginning of this book we boasted of the sophistication of the sound synthesis available on the Commodore computer. After listening to the truly astounding sounds played in this chapter, we feel sure you will agree that the possibilities for using the facilities provided by the SID chip are almost without bound.

Appendix 1

Note frequencies (cycles per second)

Note	Frequency	H	L
C0	16.35	1	18
C0#	17.32	1	34
D0	18.35	1	51
D0#	19.44	1	70
E0	20.6	1	89
F0	21.82	1	110
F0#	23.12	1	131
G0	24.49	1	155
G0#	25.95	1	179
A0	27.5	1	205
A0#	29.13	1	232
B0	30.86	2	5
C1	32.7	2	36
C1#	34.64	2	69
D1	36.7	2	103
D1#	38.89	2	140
E1	41.2	2	179
F1	43.65	2	220
F1#	46.24	3	7
G1	48.99	3	54
G1#	51.91	3	102
A1	55	3	154
A1#	58.27	3	209
B1	61.73	4	11

C2	65.4	4	73
C2#	69.29	4	138
D2	73.41	4	207
D2#	77.78	5	24
E2	82.4	5	102
F2	87.3	5	184
F2#	92.49	6	15
G2	97.99	6	108
G2#	103.82	6	205
A2	110	7	53
A2#	116.54	7	163
B2	123.47	8	23
C3	130.81	8	146
C3#	138.59	9	21
D3	146.83	9	159
D3#	155.56	10	49
E3	164.81	10	205
F3	174.61	11	113
F3#	184.99	12	31
G3	195.99	12	216
G3#	207.65	13	155
A3	220	14	106
A3#	233.08	15	70
B3	246.94	16	46
C4	261.62	17	37
C4#	277.18	18	42
D4	293.66	19	62
D4#	311.12	20	99
E4	329.62	21	154
F4	349.22	22	227
F4#	369.99	24	63
G4	391.99	25	176
G4#	415.3	27	55
A4	440	28	213
A4#	466.16	30	140
B4	493.88	32	93

C5	523.25	34	74
C5#	554.36	36	84
D5	587.32	38	125
D5#	622.25	40	199
E5	659.25	43	52
F5	698.45	45	198
F5#	739.98	48	126
G5	783.99	51	97
G5#	830.6	54	111
A5	880	57	171
A5#	932.32	61	25
B5	987.76	64	187

C6	1046.5	68	149
C6#	1108.73	72	169
D6	1174.65	76	251
D6#	1244.5	81	143
E6	1318.51	86	104
F6	1396.91	91	140
F6#	1479.97	96	253
G6	1567.98	102	194
G6#	1661.21	108	222
A6	1760	115	87
A6#	1864.65	122	51
B6	1975.53	129	119

C7	2093	137	42
C7#	2217.46	145	82
D7	2349.31	153	246
D7#	2489.01	163	30
E7	2637.02	172	209
F7	2793.82	183	24
F7#	2959.95	193	251
G7	3135.96	205	132
G7#	3322.43	217	188
A7	3520	230	175

A7#	3729.31	244	102
B7	3951.06	258	239

Appendix 2

Instrument tuning guide

The values given in the table are the notes for each string on particular instruments. Underneath each note is given its frequency in cycles per second. You can use these values with Program 3.1 to help you tune your instrument.

Guitar	E2	A2	D3	G3	B3	E4
	82	110	147	196	247	330
Ukulele	D4	F4	A4	B4		
	294	349	440	494		
Banjo	C3	G3	D4	A4		
	131	196	294	440		
Mandolin	G3	D4	A4	E5		
	196	294	440	659		
Violin	G3	D4	A4	E5		
	196	294	440	659		
Viola	C3	G3	D4	A4		
	131	196	294	440		
Cello	C2	G2	D3	A3		
	65	98	147	220		
Double Bass	E1	A1	D2	G2		
	41	55	73	98		

Appendix 3

SID's registers

Voice 1	0 L value for frequency
	1 H value for frequency
	2 Low pulse rate
	3 High pulse rate
	4 Waveform and Trigger
	5 Attack & Decay
	6 Sustain & Release
Voice 2	7 L value for frequency
	8 H value for frequency
	9 Low pulse rate
	10 High pulse rate
	11 Waveform and Trigger
	12 Attack & Decay
	13 Sustain & Release
Voice 3	14 L value for frequency
	15 H value for frequency
	16 Low pulse rate
	17 High pulse rate
	18 Waveform and Trigger
	19 Attack & Decay
	20 Sustain & Release
Filters	21 High cut-off
	22 Low cut-off
	23 Voice select & Resonance
	24 Volume & Filter type

	25 Pot x
Miscellaneous	26 Pot y
(Read only)	27 Voice 3 digitised
	28 Voice 3 envelope

Appendix 4

SID bit map

REGISTER	7	6	5	4	3	2	1	0		
0	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	
1	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	
2	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	VOICE
3					PW_{11}	PW_{10}	PW_9	PW_8	PW HI	1
4	NOISE	⎍⎍⎍	⩘	⩘	TEST	RING MOD	SYNC	GATE	WAVE	
5	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	
6	SUS_3	SUS_2	SUS_1	SUS_0	REL_3	REL_2	REL_1	REL_0	SUSTAIN/RELEASE	
7	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	
8	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	
9	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	VOICE
10					PW_{11}	PW_{10}	PW_9	PW_8	PW HI	2
11	NOISE	⎍⎍⎍	⩘	⩘	TEST	RING MOD	SYNC	GATE	WAVE	
12	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	
13	SUS_3	SUS_2	SUS_1	SUS_0	REL_3	REL_2	REL_1	REL_0	SUSTAIN/RELEASE	
14	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	
15	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	
16	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	VOICE
17					PW_{11}	PW_{10}	PW_9	PW_8	PW HI	3
18	NOISE	⎍⎍⎍	⩘	⩘	TEST	RING MOD	SYNC	GATE	WAVE	
19	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	
20	SUS_3	SUS_2	SUS_1	SUS_0	REL_3	REL_2	REL_1	REL_0	SUSTAIN/RELEASE	
21						FC_2	FC_1	FC_0	LO	
22	FC_{10}	FC_9	FC_8	FC_7	FC_6	FC_5	FC_4	FC_3	HI	FILTER
23	RES_3	RES_2	RES_1	RES_0	FILT EX	FILT 3	FILT 2	FILT 1	RES/FILT	
24	3OFF	HP	BP	LP	VOL_3	VOL_2	VOL_1	VOL_0	VOL/MODE	
25	PX_7	PX_6	PX_5	PX_4	PX_3	PX_2	PX_1	PX_0	POT X	MISC.
26	PY_7	PY_6	PY_5	PY_4	PY_3	PY_2	PY_1	PY_0	POT Y	
27	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0	OSC 3/RANDOM	
28	ENV_7	ENV_6	ENV_5	ENV_4	ENV_3	ENV_2	ENV_1	ENV_0	ENV 3	

Explanatory notes

The bit map is based on the binary code (using just 0 and 1) so that each position in a register has a specific base 2 value. The values for each of the column positions are:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

or

128	64	32	16	8	4	2	1

There are several ways in which the various registers can be set, depending on their structure. The first way is to place (POKE) a single value in a location to perform some function. For example, in register 4, if a 1 is placed in column position 5 (corresponding to the value 2^5 or 32), the sawtooth waveform is selected for Voice 1.

The second way is to use a combination of positions in a register to produce a value to provide some effect. For example, the column positions 0 to 3 in register 5, produce a decay value corresponding to the decay parameters shown in Appendix 5. So if a value of 12 is set in register 5 then in binary this is 1100, so that bits 2 and 3 are set to 1 and bits 0 and 1 are set to zero. By this mechanism some registers are used for more than one function. In bit positions 4 to 7, the value for the attack is set. For an attack value of 2 (corresponding to a time of 16 milliseconds–see Appendix 5) bits 4, 6 and 7 are set to zero and bit 5 is set to 1, i.e. 0010 in binary. Note that the 2 only applies to the upper half of the register–really 00100000 represents the number 32. The separate components are added so that the register setting for Attack = 2 and Decay = 12 is 32 + 12 = 44. You will find that many of the programs in this book use 16*A + D to illustrate this structure.

Finally, the third use of registers requires that more than one register is sometimes used to produce a setting. For example, the frequency for Voice 1 is set using a combination of registers 0 and 1. If 27 is set in register 0 and 3 is set in register 1, the combined value produced is 256*3 + 27 or 795. This number is a 'fudge' value–it does not correspond to the actual frequency in cycles per second (Hertz). See Chapter 2 for details of the conversion of 'real' frequencies to values to be set in the two frequency registers.

Appendix 5

Envelope Rates

Setting	Attack rate (msecs)	Decay & Release (msecs)
0	2	6
1	8	24
2	16	48
3	24	72
4	38	114
5	56	168
6	68	204
7	80	240
8	100	300
9	250	750
10	500	1.5 sec
11	800	2.4 sec
12	1 sec	3 sec
13	3 sec	9 sec
14	5 sec	15 sec
15	8 sec	24 sec

Appendix 6

ADSR values for musical instruments

	Waveform	A	D	S	R
Accordion	33	6	6	13	11
Banjo	33	1	7	0	0
Flute	17	2	8	9	0
Guitar	33	2	9	0	0
Harpsichord	33	0	9	0	0
Oboe	17	11	14	0	0
Organ	17	0	0	15	0
Violin	33	6	9	5	4
Xylophone	17	0	9	0	0

Appendix 7

Glossary of variables

If you are going to include the sound effects library of subroutines developed in Chapter 10 as part of a program, you should avoid using any of the variables found there in your program code. To help you we list here all the variables found in the library.

Variable	Description
B	Base address for SID's registers
T	Time for delay in jiffies
TI	System variable for current time
TS	Start time
A,D	Attack and Decay settings
S,R	Sustain and Release settings
F	Frequency in cycles per second
G	SID's frequency value equivalent to F
SV,FV,DV	Start, finish and intervals of volume
SF,FF,DF	Start, finish and frequency steps
F,SW	Frequency and volume counters
IR	Initialising variable

Appendix 8

Musical note time codes

Note	Note name	Musical name	Time code
𝅝	Whole	Semibreve	16
𝅗𝅥.	Dotted half	Dotted minim	12
𝅗𝅥	Half	Minim	8
♩.	Dotted quarter	Dotted crotchet	6
♩	Quarter	Crotchet	4
♪.	Dotted eighth	Dotted quaver	3
♪	Eighth	Quaver	2
𝅘𝅥𝅯	Sixteenth	Semiquaver	1

Further Reading

Commodore 64 Programmer's Reference Guide, CBM, Wayne, 1983

Rudiments and Theory of Music (Associated Board of the Royal Schools of Music), London, 1958

A. H. Benade, Fundamentals of Musical Acoustics, OUP, New York, 1979

A. Wood, The Physics of Music, Wiley, New York, 1974

J. Askill, Physics of Musical Sounds, D. Van Nostrand, New York, 1979

J. Backus, The Acoustical Foundations of Music, J. Murray, London, 1970

W. M. D. Turner, Learning to Use the Commodore 64 Computer, Gower, Aldershot, 1983

Index of sounds

Numbers in bold type refer to program numbers, e.g. **1.1**. All other references are to page numbers.

accordion, **9.1**, **9.6**
airplanes, 110
arcade game, 151
arrow, **1.1**

bagpipes, **7.1**
banjo, **9.1**, **9.6**
beats, **5.3**
birds, **10.1**, **14.10**, **14**, **11**
boingy, **16.5**
bouncing, **16.6**
bubbling, **15.7**

cat, **14.5**
carriage clock, 120
cauldron, **15.7**
chords, **5.2**
click, **15.2**
clock, 120
counterpoint, **7.3**
crash, **11.5**
creak, **15.3**

door, **15.3**
double bass, **16.5**
dragon, **14.8**
drum, **15.6**

engine, 110
explosion, 123

falling, **11.4**
floorboard, **15.3**
flute, **9.1**, **9.6**
footstep, **15.4**
frog, **14.1**, **14.2**

ghost effect, **5.4**
glissando, **2.1**
gong, **12.1**
grandfather clock, 120
grasshopper, **14.3**
guitar, **9.1**, **9.6**

hammer, **13.2**
harp, **16.7**
harpsichord, **9.1**, **9.6**
heavenly, **16.7**

insects, **14.3**, **14.4**
intervals, **15.1**

jet, **11.7**
jungle, 127

keyboard, **3.3**, 82

machine gun, 150
metronome, **3.2**
monoplane, 110
mosquito, **14.4**
musical instruments, 82

oboe, **9.1**, **9.6**
organ, **9.1**, **9.6**
owl, **14.11**

percussion, 123, 145
phaser, **16.1**
police, 150
pulse, **9.7**, **9.8**, **9.9**

revolver, 123

rifle, 123
road drill, 150
rocket, **11.8**
rounds, **7.2**

scale, **2.3**, **4.1**
sea gull, **10.1**
shot, 123
siren, 150
snake, **14.6**
snake charmer, **14.7**
snap, **15.1**
sparky, **16.5**
submarine, 150

tapping, 150
telephone, **12.3**, 150
thunder, **13.3**
toilet, **15.5**
tunes, **1.1**, **4.2**, **4.3**, 59
twig, 140

vibrato, **9.2**, **9.3**, **9.5**
violin, **9.1**, **9.6**

walking, **15.4**
water, **10.1**, **15.7**
waves, **10.2**
waveforms, **6.1**
weapons, **16.1**, 150
whooping, 151
wind, **13.4**
wolf, **14.9**

xylophone, **9.1**, **9.6**

General Index

ADSR, 68, 84, 106, 115, 118, 134, 144
ADSR trigger, 73
amplitude, 14, 70
amplitude modulation, 91
arrays, 61, 135, 146
attack phase, 69

band pass filter, 155
band reject filter, 156
beats, 45, 119

CHAP, 26
chord, 39
chorus effect, 45
cocktail party effect, 19

DATA, 32, 66, 146
decay phase, 70
DIM statement, 63
Doppler effect, 17
dummy loop, 33, 111

envelope, 69, 75, 79

filters, 84, 127, 134, 153
flag, 33
frequency, 22, 32, 46
frequency modulation, 86, 152
frequency sweep, 116, 133, 136, 142, 143
fundamental frequency, 44, 50

ghost effect, 48, 119

harmonics, 44, 50
high pass filter, 155

instruments, 28, 82
internal clock, 23
interval, 41

jiffies, 74, 99

longitudinal waves, 15
low pass filter, 155

multipart music, 58
music, 11, 12, 32, 58

natural frequency, 14
noise, *see* white noise
notch filter, 156
note lengths, 35

octave, 12, 23, 31, 32
overtones, 50, 82

parameters, 64, 75
PEEK, 88
pitch, 15, 22
POKE, 21
programmable pointer, 60, 63
pulse, 92, 131

READ, 32, 66, 146
read only locations, 88
rectangular waveform, 56, 93
registers, 12, 21, 58
release phase, 70
resonance, 18, 155
restore, 38
ring modulation, 118
rule of seven, 58, 80, 144

sawtooth waveform, 54, 79, 84, 89, 122, 132, 142
semitones, 25
SID, 11, 55, 58, 93
sine wave, 51
sound chip, 11
sound effects, 12, 98, *see also* Index of sounds
step size, 149, 150
subroutines, 99, 127, 130, 140, 144
sustain phase, 70
sweeping pulse width, 96
sweeping the filter, 157
switch off effect, 140
synchronisation, 158

tempo, 29, 36
timbre, 50
tremolo, 91
triangular waveform, 55, 79, 89, 119, 122

ultrasound, 20

vibrato, 85
voices, 12, 58
vox celeste, 47
V/T effect, 27, 141

waveforms, 50, 119
white noise, 56, 79, 89, 106, 110, 125, 133, 143

Duckworth Home Computing

SOUND EFFECTS AND MUSIC ON THE COMMODORE 64

William Turner & Alf Vella

The sound chip in the Commodore 64 is one of the most advanced available on any microcomputer, with its three independent voices, four waveforms, filters, and so on. This book explains how to make all these features work, illustrating each step with example programs. The chapters on music progress gradually from synthesising a simple tune to playing rounds, bagpipes and multi-part music, and include a program to set up the 64 keyboard as a synthesiser. If you are interested in sound effects, you will find program listings for such diverse sounds as birds, explosions, frogs, aeroplanes, creaking floorboards, rockets, sirens and many more.

William Turner is a senior lecturer at Oxford Polytechnic. He also works as a computer consultant, is the author of several books on computing and statistics, and programmed the sound effects in the popular adventure game *Macbeth.*

Alf Vella worked as a scientific programmer at the Open University. He is now a senior lecturer at North Staffs Polytechnic and has written articles on mathematics and computing.

ISBN 0-7156-1986-1

9 780715 619865

Duckworth
The Old Piano Factory
43 Gloucester Crescent, London NW1

ISBN 0 7156 1986 1

IN UK ONLY £6.95 NET



This was brought to you

from the archives of

http://retro-commodore.eu