



M

# MUSIC & SOUND ON YOUR AMSTRAD

Ian Sinclair





**MUSIC & SOUND  
ON YOUR AMSTRAD  
CPC 464**

**Ian Sinclair**



**MELBOURNE HOUSE  
PUBLISHERS**

© 1985 Ian Sinclair

All rights reserved. This book is copyright and no part may be copied or stored by electromagnetic, electronic, photographic, mechanical or any other means whatsoever except as provided by national law. All enquiries should be addressed to the publishers:

IN THE UNITED KINGDOM —  
Melbourne House (Publishers) Ltd  
Castle Yard House  
Castle Yard  
Richmond, TW10 6TF

IN AUSTRALIA —  
Melbourne House (Australia) Pty Ltd  
2nd Floor, 70 Park Street  
South Melbourne, Victoria 3205

ISBN 0 86161 192 6

Printed and bound in Great Britain by Short Run Press Ltd, Exeter

Edition: 7 6 5 4 3 2 1  
Printing: F E D C B A 9 8 7 6 5 4 3 2 1  
Year: 90 89 88 87 86 85

# Contents

<b>CHAPTER 1 — SOUND SENSE</b> .....	<b>1</b>
The shape of the wave .....	5
The language of sound .....	7
Stereo effect .....	11
Sound snags .....	15
<b>CHAPTER 2 — MUSIC HATH CHARMS</b> .....	<b>17</b>
Naming notes and writing music .....	22
<b>CHAPTER 3 — THE AMSTRAD SOUND</b> .....	<b>35</b>
The BASIC beep .....	38
Superior SOUND .....	38
Timing the note .....	42
Make thy musick .....	43
Formula Music .....	49
<b>CHAPTER 4 — HARMONY AND STEREO</b> .....	<b>55</b>
Synchronisation .....	65
How to synchronise .....	70
Stereo Sound .....	71
Rolling your own .....	75
The program action .....	80
Harmoniser .....	81
<b>CHAPTER 5 — WAVEFORMS AND EVELOPES</b> .....	<b>83</b>
Hardware envelopes .....	87
An absolute envelope .....	93
An evelopes program .....	102
Pitch Envelopes .....	106
<b>CHAPTER 6 — USING NOISE</b> .....	<b>111</b>
Pitch envelope effects .....	118
The whole caboodle! .....	123
<b>CHAPTER 7 — LOOSE ENDS</b> .....	<b>127</b>
The PSG registers .....	134
The enable register .....	135
The sound routines .....	138
<b>APPENDIX A — MUSICAL TERMS</b> .....	<b>145</b>
<b>APPENDIX B — MUSICAL INSTRUMENTS</b> .....	<b>149</b>
<b>APPENDIX C — CONNECTING TO OTHER UNITS</b> .....	<b>151</b>

# Preface

One of the most enjoyable developments in computing in the 80's has been the ability of the computer to produce sound. This started as a weak squeak, developed to a mellow bellow, and now allows what we might call a strange range. Modern computers can play all the notes of a very extended musical scale, and many have the ability to carry out a limited amount of sound synthesis. This allows some imitation of other instruments, but more importantly, it permits a vast range of sound effects to be generated. The Amstrad CPC464 and CPC664 machines are rightly renowned for the possibilities that they offer to any programmer who wants to create sound effects, and this book is dedicated to the user who wants to make a start with this fascinating branch of computing with the Amstrad machines.

One of the problems with programming for sound is that the computer owner may not know much about sound, less about musical notation, and nothing at all about sound effects. If that reads like a description of you, then this is your guidebook. Whatever you want to do, a quick snatch of tune to liven up a game, a long piece of music in three-part harmony, or sound effects for a Space Adventure, you'll find all the hints and tips here. In addition, you'll find a detailed description of what sound is, how the computer can produce sound, and how you can control it. All I have assumed is that you have had a little experience of programming your Amstrad machine in BASIC, so that you know how to write program lines. Since few of us can compose music well, I've included lots of detail on how to read sheet music and convert these strange-looking dots into numbers that the computer can use. All this has been written from the point of view of a computer owner rather than that of a music student, though by the time you have finished, you'll certainly know more about music than most people. I have put in a very short section for the machine-code programmer, assuming that anyone who wants to program for sound in this way will already know quite a lot about machine code, and needs only a few hints and tips on the techniques of using the sound routines.

I hope that, whatever your interests in computing for sound, you will find this book a good introduction and a useful source of programs and information. I have included Appendices on musical terms, the pitch ranges of instruments, and how to connect the CPC464 and CPC664 to amplifiers and to tape recorders in order to obtain better quality sound. Since a large number of computer owners are also Hi-Fi enthusiasts, this information should be very useful and is not easy to dig out anywhere else.

Finally, I must thank all at Melbourne House, who saw the potential of this book, and encouraged me in the writing of it. I specially want to thank Alan Giles, who initially reacted favourably to the title, commissioned the book, and who has edited my manuscript into the form that you now see before you.

Ian Sinclair, June '85

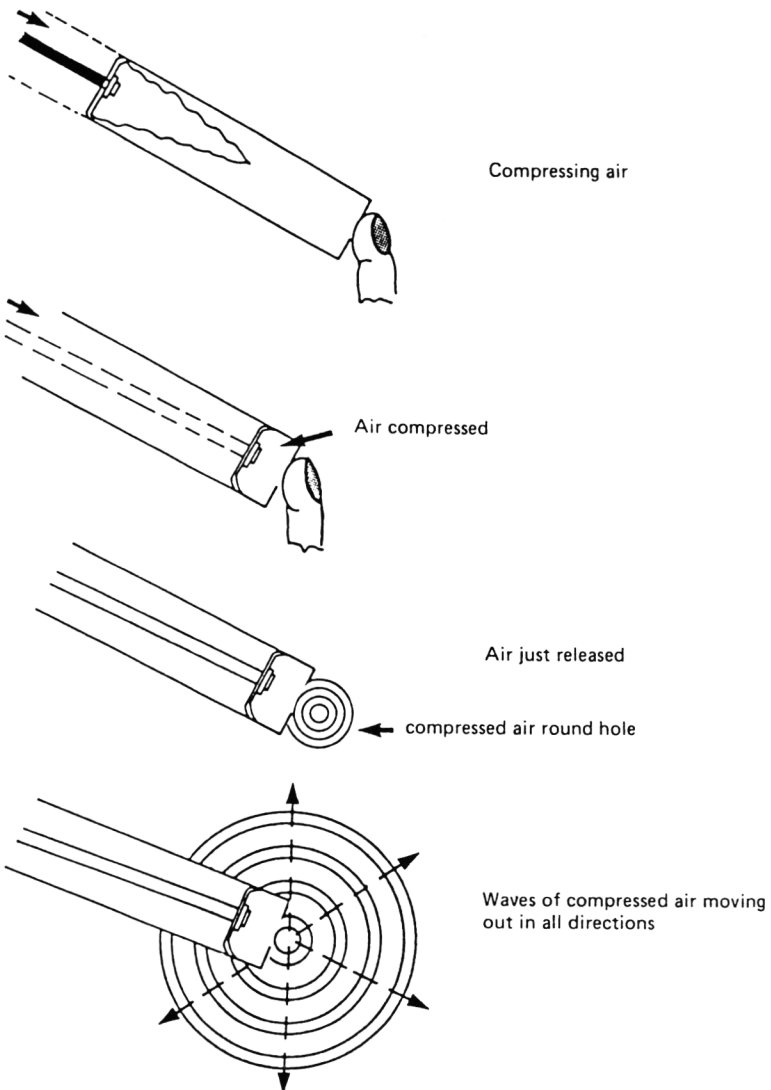
# 1

## Sound sense

Have you ever thought how different sound is as compared to other computer actions? For example, if you have programmed in BASIC for some time, you can look at a program for your CPC464 and have a good idea of what it does. Even if graphics instructions are used, if you have used the graphics of the machine to any extent you can always make a good guess about what you'll see on the screen when the program runs. Sound is very different. Even after a lot of experience, it's still difficult to look at a sound program and be able to say what kind of sound you expect. Part of the problem is that we can't describe sounds easily. You can draw on a piece of paper what a graphics program might produce, and you can say what an accounts program will print on to paper, but you can't mimic the sound that you'll hear when a sound program runs. Sound is different, and to master sound programming on the CPC464 you have to understand sound itself as well as have a reasonable knowledge of programming in BASIC. Your reward for all this effort is the ability to enrich your programs (or any other programs in BASIC) with a vast range of sound effects or musical phrases. If you go on to program your CPC464 in machine code, then what you have learned about sound programming in this book will still be useful to you — the programming details may be different, but the principles are exactly the same, and Chapter 7 contains a special section devoted to the subject of machine coded sound.

Most of the sound that we hear comes to us through the air, so we'll concentrate on that. Air is an elastic material, which means that its volume can be changed by squeezing it. When you block

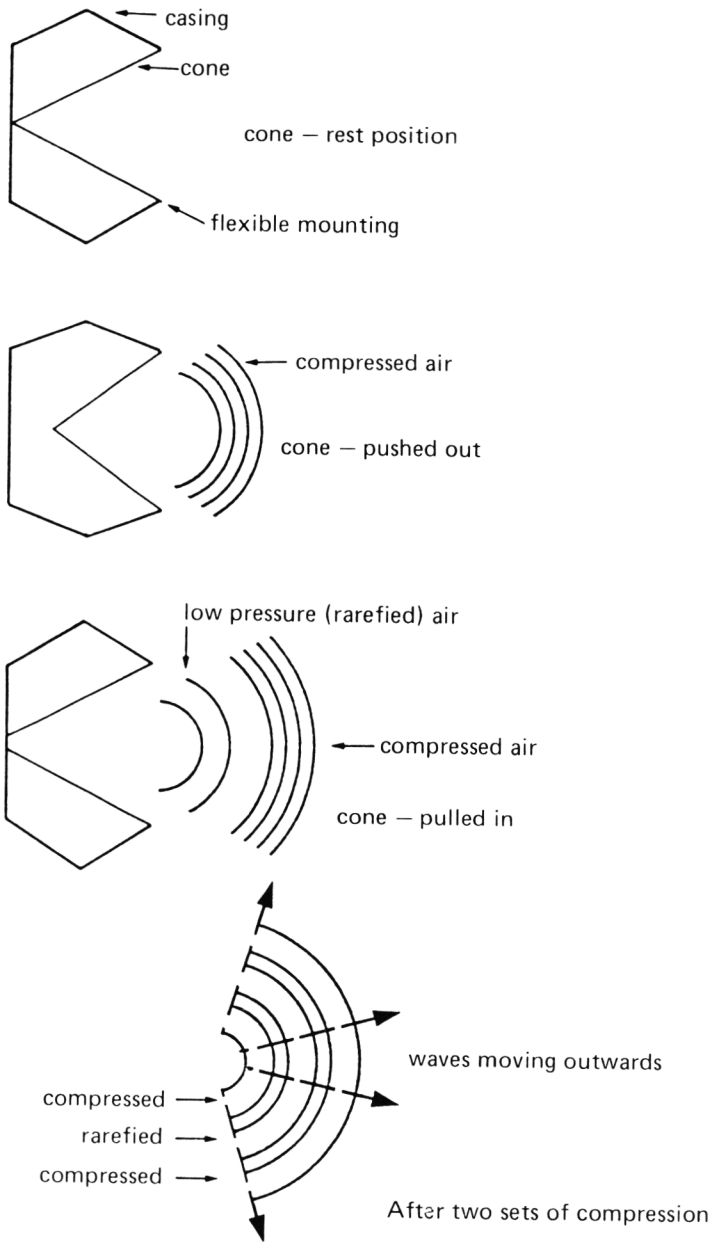




**Figure 1.1** How a wave of sound can be formed. Anything that makes air move will also compress it, and the wave of compression moves outwards as a sound wave.

the outlet of a bicycle pump and push the plunger down, for example, you are squeezing (or **compressing**) the air, and its volume is less; it takes up less space in the pump. A vacuum pump works the other way round, using a plunger that pulls at the air and increases its volume so that it takes up more space. You don't even need to have the air inside a pump to be able to push and pull at it like this. Suppose you compressed some air in a pump, and then released the end? The air in the pump rushes out, and the only way it can get room outside the pump is by compressing the air outside. This lot in turn compresses the air next to it, so that if you could see compressed air, you would see a bunch of compressed air working its way outwards all round the pump (Figure 1.1). It's not the same chunk of compressed air, though. All that has happened is that the air at each place round the end of the pump gets compressed and then recovers its place by compressing the next lot of air. The air behaves like the steel balls in a Newton's Cradle, passing on the push to the next one. This is what is called a sound wave. The name isn't a coincidence, because water waves behave in the same way. When you look at waves, you feel sure that the waves are moving into the shore. If you watch something floating on the water, though, all you see is the up-and-down movement. That's because each patch of water is just moving up and down — the wave that you see is the result of the way that the movement spreads from one patch of water to the next.

How does that affect us? Well, what we hear depends on the vibration of air that reaches our ears. When the sound has been generated by a computer, the air is set into its wave movement by the cone of a loudspeaker (Figure 1.2). As the cone moves forward, it compresses the air, and as it moves back again, the air is de-compressed (the technical term is that the air is 'rarefied'). The result, once again, is a wave which reaches our ears. Inside the ears this air pushes and pulls alternately at the eardrum, vibrating the bones inside the ear and causing electrical signals in the nerves. That's what we call 'hearing'; one of these miracles that we take for granted. All of the things that we hear

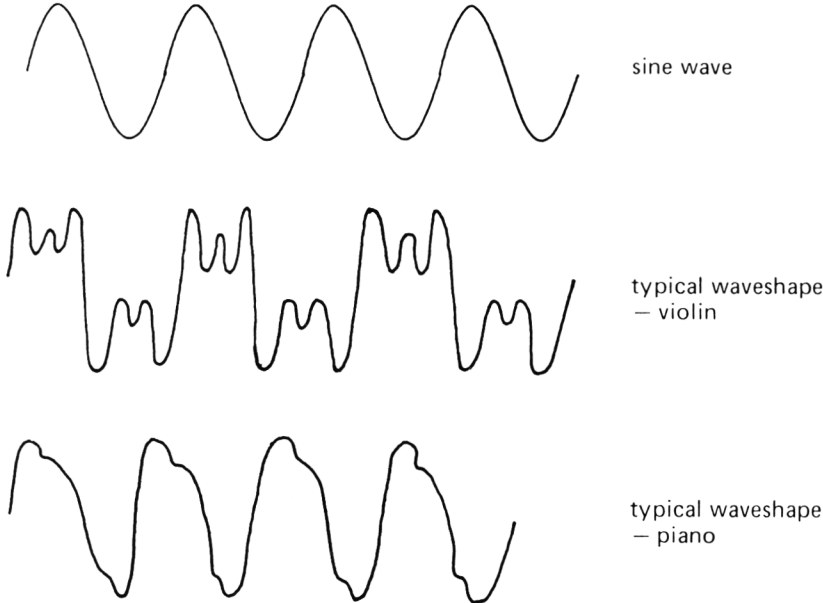


**Figure 1.2 How the moving cone of a loudspeaker creates sound waves.**

are things that vibrate the air. The skin of a drum vibrates, pushing and pulling at the air. The string of a guitar can't push much air, but if it's fastened to a wooden box, the string vibrates the box and the box moves the air. Nowadays the alternative is more complicated — the guitar string moves a magnet which generates an electrical signal which is amplified and works a loud-speaker. Some musical instruments move the air directly, like the flute, clarinet, oboe, bassoon, trumpet, trombone and other wind instruments.

## The shape of the wave

When you look at waves in fairly calm water, you can see a definite shape, a smooth up-and-down appearance of the surface of the water. Sound waves are invisible, but we can make the shape of the wave appear by using electronic instruments. The **oscilloscope** is the instrument that is used, and when it is connected to a microphone, the shape of any sound wave that



**Figure 1.3** Waveshapes — the simplest shape is called the 'sinewave', but the waveshapes from musical instruments are much more complicated.

reaches the microphone can be seen on the screen of the oscilloscope. The simplest type of wave is the smooth up-and-down type, which technically is called a 'sine wave'. It's the shape of wave that you see when you whistle into the microphone, but the waves that musical instruments generate are much more complicated (Figure 1.3). Most musical instruments generate waves that are of a more jagged shape, and the exact shape depends a lot on the type of instrument and how it is being played. That's one of the reasons why you can tell one instrument from another. A note played on a flute does not sound like the same note played on a clarinet, for example, or the same note played on a piano. A lot of electronic instruments can produce notes that do not sound like the notes of any conventional musical instrument, and this is done by using a very different shape of wave.

A sound synthesiser is an electronic instrument that can mimic other instruments by producing waves that are roughly the right shape. Your computer doesn't have this sort of versatility, and the wave shape that it produces is roughly square (Figure 1.4). This is because it happens to be a very easy shape of wave for a computer to generate. It's done by switching a voltage on and off, which produces an electrical wave that is perfectly square. The loudspeaker can't cope with such a wave, however, so what you get from the loudspeaker is rather more like the second drawing in Figure 1.4. If you feel that having just one wave shape is a bit disappointing, remember that a sound synthesiser costs

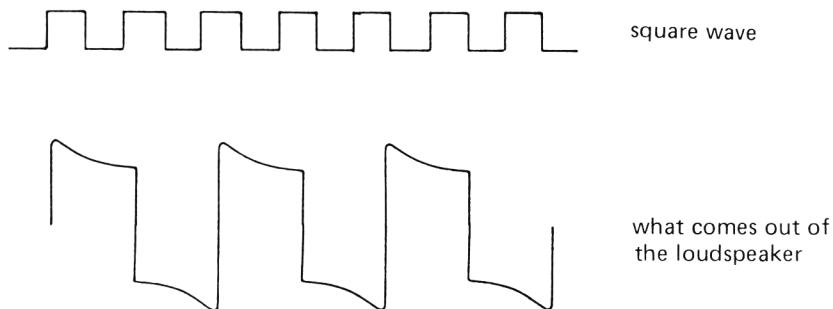


Figure 1.4 The square wave, and what a loudspeaker does to it. The loudspeaker rounds off all the corners, and allows the flat parts of the wave to droop.



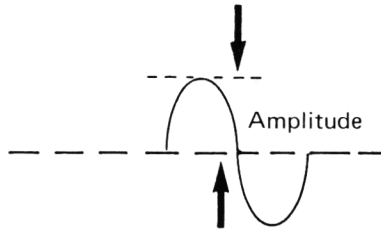
a lot more than a computer and it can't display graphics or work out your income tax.

There's another perfectly good reason for using this shape of wave, however. Because it's a fairly jagged sort of shape, it's closer to the shape of real musical instruments that you could get by using a rounded 'sine' type of wave. As it happens, a lot of synthesisers generate a square wave electrically, and then operate on it to change its shape to what is needed. This requires the use of 'filters' which act to change the shape of a square wave. Once again, you can't do this easily with your computer, but you **can** do quite a number of things that will make the sound resemble a note from another instrument rather more closely. In addition, it's possible to use 'noise'. Noise means a wave with no fixed shape and no pattern to it. It's the musical equivalent of a set of random numbers, and the computer can generate this noise by using signals which are controlled by the random number generator. Noise is the heart of all of the sound effects, like gunshots, puffing trains, space launches and all the other favourites of the games. With the combination of square waves and noise, then, your computer can be programmed to produce some quite impressive sounds.

## **The language of sound**

Nobody has to tell you very much about the language of graphics. You knew what was meant by a line, a circle, and a colour before you started using a computer. Once again, sound is different, and you may not have met the words that are used to describe a sound unless you have taken an interest in Hi-Fi or in music synthesis. That's because the words that we need are about the wave, and you can't see the wave. Unless you have worked with an oscilloscope you will never have seen what the shape of a sound wave looks like, and the words 'amplitude' and 'frequency' won't mean much to you.

The amplitude of a wave means its wave height. For a water wave, you would measure this from the level of still water to the crest of a wave, and on the diagram of a sound wave, we can show the same sort of measurement in Figure 1.5. Since you



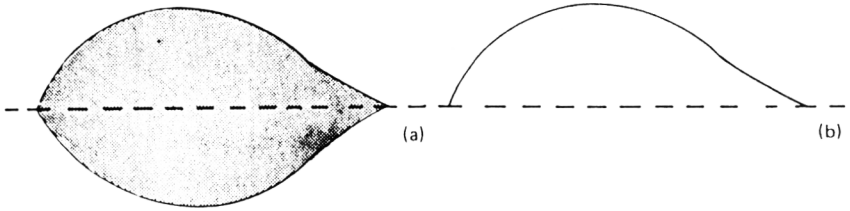
**Figure 1.5** The amplitude of a wave is its size measured from a peak to the centre.

can't normally see this, how does it affect you? The answer is loudness. The greater the amplitude of a sound wave, the louder it sounds to your ear — assuming that you can hear it at all. The connection is not quite so simple as you might think, though. If you listened to a set of different notes, all of the same amplitude, you would not hear them all as being equally loud. That's because the human ear is not equally sensitive to all sounds. The ear is most sensitive to sounds in the range that we make for ourselves when we speak, and it's less sensitive to sounds outside this range. To put it another way, the voice and the ear were designed as a matching pair, and any other source of sound seems to have been an afterthought.

There's another feature of all waves that affects sound. When you look at water waves, you see some that are slow-changing, so that only a few waves pass you in a minute. Others seem to be spreading rapidly, many more in the minute. The number of sound waves that passes a point in each **second** is called the 'frequency' of the sound. The second is chosen as the measure of time because a minute is much too long. Sound waves are packed quite closely together, with several hundred typically passing your ear per second. This frequency of waves corresponds to what we call the **pitch** of a sound. A low pitch corresponds to a low frequency, and a high pitch to a high frequency. Putting some numbers to these ideas, the lowest notes of a church organ might have a frequency of about 30 waves per second, and the highest notes of a piccolo about 4000 waves per second. A frequency of one wave per second is called one **Hertz** (after Heinrich Hertz, who discovered radio waves) so that the range of these musical sounds is from 30 Hertz, written as

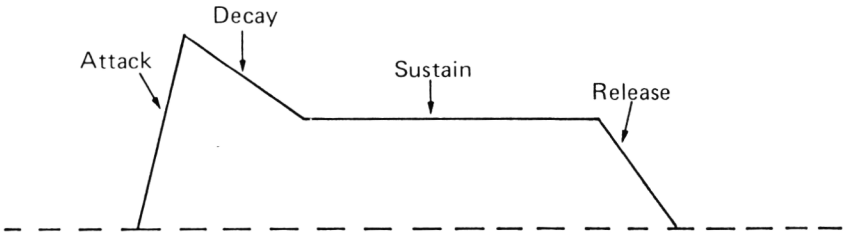
30 Hz, to 4000Hz. To avoid writing a lot of zeros, 4000 Hz is usually written as 4 kHz, where the 'k' means kilo, a thousand. Your ear, if it's in perfect condition, can detect sounds in the range of about 30 Hz to 18 kHz. As you get older, though, the ear becomes less sensitive to the higher notes, and you end up with the top end of the range at about 7 kHz or less. You might think that since the highest note that a musical instrument can make is at about 4 kHz, this doesn't matter. Unfortunately, it does. A simple smooth sinewave at 4 kHz consists only of waves at this frequency, but a wave of any other shape is more complicated. The reason for a wave having a different shape is that it contains higher frequencies which are related to the main frequency. What I mean by that is that if the instrument generates a square shaped note at 4 kHz, then that note also contains frequencies at 8 kHz, 12 kHz, 16 kHz, 20 kHz and so on. These frequencies are twice, three times, four times (and so on) the frequency of the main note. We call these notes 'harmonics' and any wave that is not a perfect sinewave contains harmonics. They make the note sound more interesting to our ears, and they give a note some character, so that we can tell one instrument from another. The square wave which is obtained from your computer is rich in these harmonics, which is why it can be used so successfully in creating sounds.

There's another feature of sound that's just as important, but much more difficult to measure. You can measure amplitude as a size number, and frequency as a number of Hertz, but you can't so easily make a simply measurement of the **envelope** of a musical note. You see, musical instruments don't give out waves that are continuous. Each note of music is separate, so that the waves start, continue for a time, and then stop. The way that musical instruments are constructed, the amplitude of the waves is not constant during the time of a note. Right at the start of a note, the amplitude is low, so that the note starts soft. The amplitude then increases to its peak, falls away, and finally stops. You can examine the shape of a complete note using the oscilloscope, and typically you get something like the shape in Figure 1.6. This is not the shape of the **wave**, remember, it's the



**Figure 1.6 An envelope (a) is the name of the shape that is traced out by the different amplitudes of the waves in a note. We usually show one half of the shape only, with no waves (b).**

shape that shows how the amplitude of the waves changes. This shape, the envelope, has a lot to do with how a sound affects our ears. Explosive or hammering sounds, for example, have envelopes that grow very rapidly and then die away very rapidly. Many musical instruments produce sounds whose envelopes grow fairly rapidly, and die away in two stages, slowly at first and then faster. This type of shape (Figure 1.7) is called an ADSR shape, the letters meaning Attack, Decay, Sustain, Release. The names are given to the four distinct sections of the envelope shape which are shown in Figure 1.7. If you want to synthesise the sounds of music, and in particular if you want to create good sound effects, then you must be able to control the envelope of each note. Fortunately, the Amstrad CPC464 and CPC664 allow you to control this feature of sound very satisfactorily. In addition to this variation of amplitude during a note, the sound of some instruments varies **pitch** throughout a note. This effect can also be reproduced with the Amstrad computers, and it's one that we'll look at later in this book.



**Figure 1.7 Most envelope shapes can be reproduced approximately by using a shape made out of four straight lines. The sections are attack, decay, sustain and release.**

# Stereo effect

There is a very noticeable difference between listening to music from a mono radio and listening to the same performance live. Quite apart from the sound quality, you are always aware that the sound from a radio comes from one small loudspeaker, whereas the live sound comes from a variety of instruments that are spread out in front of you. You can't, however, overcome the difference by listening to the sound from two loudspeakers placed some distance apart **and playing the same sound**. The effect of using two loudspeakers like this is simply that you hear the sound coming from one or the other, with no illusion of space. The essential feature of sound that creates the illusion of a wide-spaced source is that the loudspeakers should each play **slightly different** sounds. Even if you imagine that the musicians at each end of a stage are playing the same notes on the same instruments, it is quite impossible that the soundwaves which they are creating are *identical* in every way. Because of these slight differences, we can locate the positions of sound sources. It was

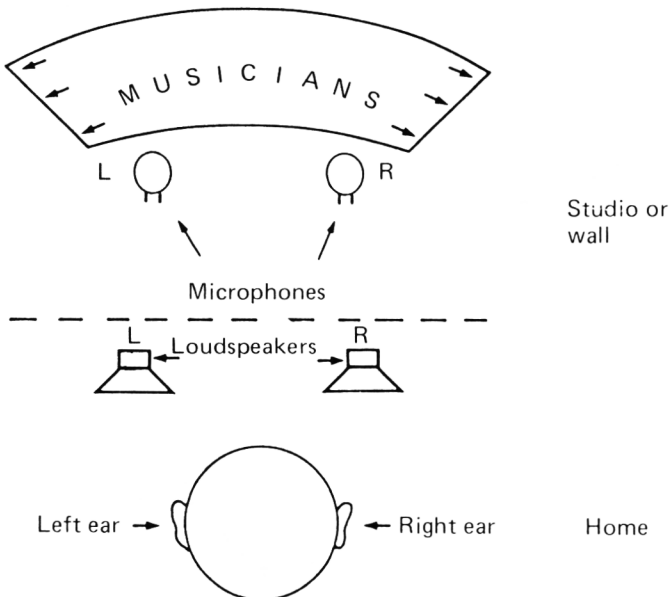


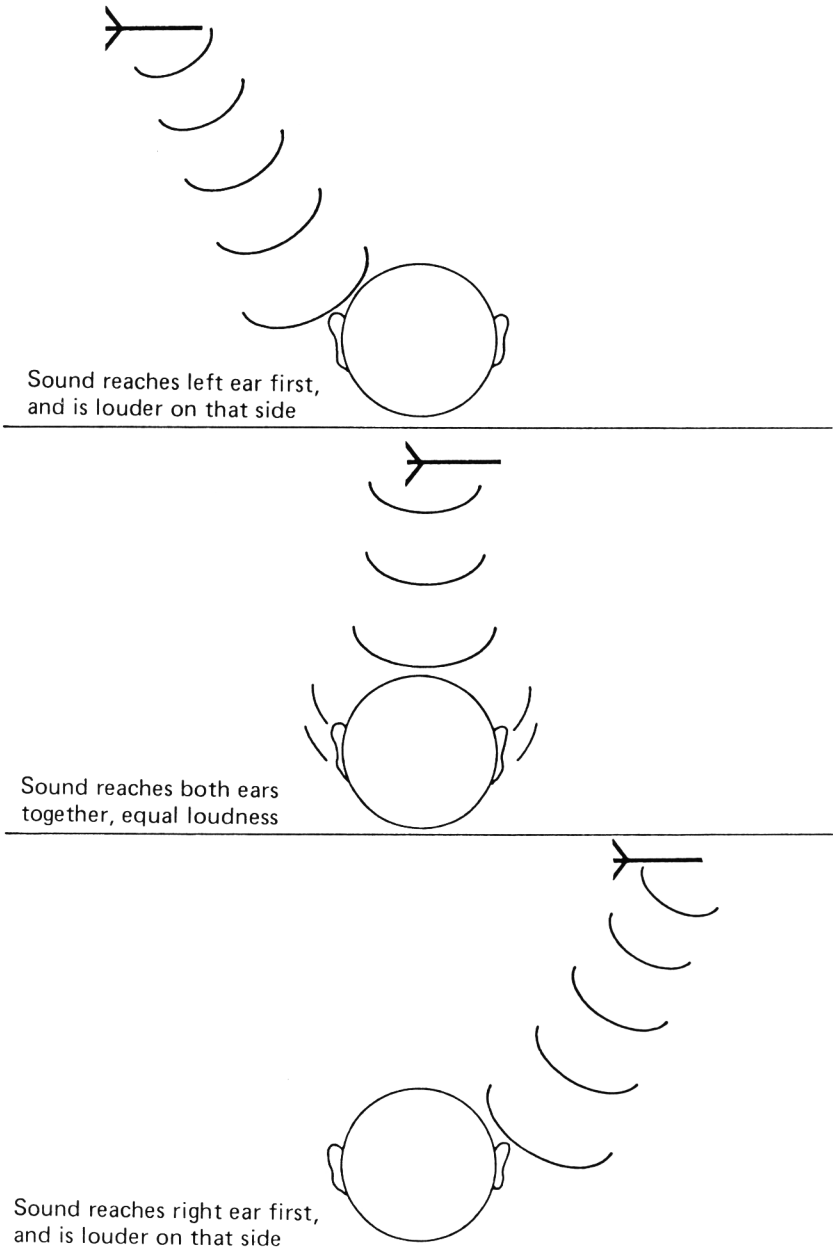
Figure 1.8 Stereo uses two separate microphones picking up different parts of the sound and transmitting them to two separate loudspeakers.



once thought that this was mainly due to possessing two ears, but there is evidence to show that even with only one ear, a human listener can appreciate the difference between stereo and mono sound from loudspeakers.

The vital difference between stereo and mono, then, is not the number of loudspeakers but the sound that emerges from the loudspeakers. If each loudspeaker gives out a slightly different version of the sound, corresponding to what a microphone picks up in part of the place where the recording is made, then the effect will be of stereo sound (Figure 1.8). This means that you will be aware of the positions of instruments, and of a feeling that the sound is coming from a wide source, not from a single point. It's much easier to understand if you think of a sound effect like a passing jet aircraft, rather than a band. Imagine that a jet fighter (all right, an F-15 if you want an example) is crossing from left to right in front of you. The sound will start off faint in the left ear, and fainter at the right ear. As the plane moves, the sound gets louder in both ears, but while the plane is still to the left, it will still seem louder in the left ear. When the plane is in front of you, the sound will be equal in both ears, and as it moves to your right, the sound in the right ear will start to be greater than the sound in the left ear. This continues until the plane is out of earshot. As it happens, because the sound moves so much slower than the light, the plane still sounds on your left when it has passed over to the right, but the sequence is the same, and it's illustrated in Figure 1.9. If you wanted to program this sound in stereo, you would have to reproduce these different levels of sound in the two ears, which means using two different envelopes.

The alternative to loudspeakers for stereo sound is, of course, earphones. Nowadays, the use of large earphones is less common, and the portable stereo player of the Sony Walkman variety is the most usual version of earphone stereo. When earphones are used, the differences between the sounds that are fed to the two ears are very much greater than when loudspeakers are used. The stereo effect is therefore much stronger, and is quite artificial and totally unlike a live performance. For sound effects, this doesn't matter, and you can make use of 'Walkman' type



**Figure 1.9 Both the loudness and the timing of the sound from a moving object cause the ears to hear different 'envelopes' of sound.**

earphones to hear a lot of interesting effects. The Amstrad computers are almost unique in allowing you to hear sounds in stereo. This is done by having two of the sound generators of the computer connected to different points in a standard 'Walkman' earphone jack. If your program can generate different signals to these points, then, the result will be stereo sound, providing that your programming is suitable. The effect is not quite so good for music, but for sound effects like jet fighters flying past, it's devastatingly good. The only problem is that the amplitude of the signals at the headphone jack is not very large, and you have to have the volume of the signals turned well up to hear anything at all. This does **not** mean the volume control on the computer, because it has no effect on the earphone signals. Different earphones have different sensitivity, and some earphones are better than others in this respect. Appendix C shows how you can use a small amplifier for your earphone signals.

You aren't stuck with earphones for your sound, though. Normally, the sound of the CPC464 and 664 comes from a built-in loudspeaker. There's just one speaker, and so you don't get stereo sound from this. You can, however, make up connecting links that will let you connect the headphone socket to the input of any stereo amplifier. Details of such links are given in Appendix C. With this connection made, you can now generate sounds that will strip the paint from the walls if you want it to. With a stereo amplifier attached, you have a better control of volume, and you can also make use of the balance control to adjust the position of sound. Another possibility is to record the sound on a stereo cassette so that you can hear it in all its glory later. Like any other aspect of producing sound with a computer, good stereo effects can be produced only after a bit of experience and a lot of try-it-and-see experiments. Don't be disheartened if your first efforts don't seem to be as good as you expected. If you follow the advice in this book, you will know where to start, and how to get more quickly to what you want. The important point is that you **must** listen to the examples, because it's only by learning to associate a sound with its program that you can make short-cuts when it comes to designing your own sound programs.

## Sound snags

Nothing's perfect, and sound systems are no exception. For one thing, the computer is operated from the mains supply of electricity. This mains supply consists of an electrical wave whose frequency is 50 Hz, and in the course of being turned into the DC that your computer needs to operate it, a frequency of 100 Hz is generated. Now this particular signal is no worry to the computing circuits, because they work at high speeds and ignore anything so low. The sound circuits, however, will pick up this signal and amplify it. It sounds like a rough rasping hum, and because its frequency is fairly low, you don't hear it very obtrusively on the earphones, and certainly not from the built-in loudspeaker. This is because small loudspeakers do not reproduce low frequencies well. If, however, you have connected your Amstrad to a Hi-Fi system, you can expect to hear this hum rather more noticeably. Once again, if you are making use of sound effects, it doesn't matter much, but it limits the amount of amplification that you can successfully use.

Hum is just one of the problems that troubles you when you start to make use of an external amplifier. You can also run up against limitations of amplifiers, loudspeakers — and ears. The sound generator of the Amstrad machines will provide sounds whose frequency can range from 30 Hz to 125 kHz. Now 30 Hz is too low for many amplifiers to cope with, and most loudspeakers give up on frequencies below 80 Hz. You should avoid this sort of frequency, then, unless you are making up programs to test your Hi-Fi. You also have to avoid frequencies that are much above 15 kHz. Once again, though most amplifiers can cope, few loudspeakers can, and no cassette recorders, and in any case you aren't likely to hear anything in this range. Frequencies above 20 kHz are strictly for dogs, cats and bats and of little use to humans. You can't record these frequencies, and unless you have access to an oscilloscope, you can't even detect them in any other way. As a practical range, use the range of 100 Hz to 4 kHz, and your efforts, whether they are musical or not, will at least be **heard**.





# 2

## Music hath charms

All music consists of sound, but not all sounds are musical. The ancient Greeks thought that music was a branch of mathematics, but I can promise you that you won't need much in the way of calculation for this Chapter. What I want to do is to show how music can be made on the Amstrad Micros, and I have to start by assuming that you don't know anything about music. If you do, then a lot of this chapter will be old-hat to you, and you can skip chunks of it. If music is like a closed scorebook to you, then a lot of what follows will be very useful if you want to program your Amstrad to play simple tunes. In the course of this Chapter, there will be several simple programs to enter and listen to. I haven't explained how they work, because that comes later — listen first, understand later is the motto of sound programming! Listening, as I have said before, is important, because that's the only way that you'll learn to associate the program instructions with the sounds that they produce.

To start with, music has been around for a much longer time than computers. Musicians have evolved their own language and their own way of writing music, and unless you intend to struggle along by singing notes and trying to match them with the computer, you really have to know something about this system of musical notes and how they are written down. Start with some fact — by trying the program in Figure 2.1. This plays a note, pauses, plays another note, pauses again, then plays the two notes together. Now these two notes sound very much alike. The second one is at a higher pitch than the first one, but there is a strong similarity to the ear, and when they are played together, they blend perfectly. The reason is that the first note

```

10 SOUND 1,478,200
20 FOR N=1 TO 3000:NEXT
30 SOUND 1,239,200
40 FOR N=1 TO 3000:NEXT
50 SOUND 1,478,300
60 SOUND 2,239,300
70 END

```

**Figure 2.1 Octaves. A note whose frequency is exactly twice the frequency of another is its octave. You sense that there is a pitch difference, but that the notes are otherwise similar.**

was at a frequency of about 261.5 Hz, and the second at about 523 Hz, exactly twice the frequency of the first one. I have taken approximate numbers rather than show several places of decimals, because the important point is that the second frequency is **exactly** twice as much as the first. In ancient times, they couldn't measure frequencies but their ears were sharp enough to tell that there was a relationship between these two notes. Musicians say that the higher note is **an octave above** the lower one. It extends beyond this, because if you listen to a note whose frequency is half of the frequency of the first note, it also seems to fit in perfectly — musicians say that this new note is an octave below the first one. Music is designed around these keynotes, and what we call a **musical scale** is a set of notes that starts at one keynote and ends at the next one above it, the octave above.

How do you divide up so as to get a set of notes for a scale? We might think that a reasonable method would be to have notes whose pitch increased by exactly equal amounts. With the advantage of several centuries of science to help us, we can divide up the notes in this way. The usual system is to have a total of eight notes, counting both keynotes, and that's where the word 'octave' (meaning a group of eight) came from. Perhaps the ancient musicians reckoned there was something special about a set of eight, just as the modern computer designers did. If you divide out the frequencies so that you have seven notes between the keynotes, all with exactly equal frequency differences, the result sounds as in the program of Figure 2.2. It starts off reasonably well with the first three notes, but the later notes sound definitely odd. It would be difficult to write a tune that sounded good using this set of notes. What's worse is that it's difficult to

```

10 FOR J=1 TO 8
20 READ F
30 SOUND 1,F,200
40 FOR N=1 TO 800:NEXT
50 NEXT
60 DATA 478,418,372,335,304,279,257,239

```

**Figure 2.2 A scale which has equal intervals, meaning that the differences of frequency between notes are equal.**

make harmony. Harmony means that the name suggests — that you can sound a couple of notes together with a reasonable chance that they will blend well. With this set of notes, there are precious few harmonies.

Well, that's a bit of egg on the face of mathematics for you. The ancient musicians, knowing nothing about any of this frequency stuff, decided that the easiest way around the problem was simply to adjust some of the notes. Most of them were reduced a bit in pitch, a few increased. This operation was called 'tempering', and an instrument which was tuned to an adjusted scale was called 'well-tempered'. As an example of what they did, try the program in Figure 2.3. It starts with the keynote and ends one octave above, but the notes in-between have been altered so as to make one variety of tempered scale. It certainly sounds more familiar and easier on the ear. More important, its set of notes allows us to make better sounding tunes and to create better harmonies. This particular scale is called the scale of 'C-Major'. The 'C' part of it comes from its keynote, which is the note that is called Middle C. We'll come back to that later. The 'Major' part serves to distinguish this from other scales, because this isn't the only possible group of notes that you can use as a scale.

What is it about this scale that makes it a 'Major' scale? The

```

10 FOR J=1 TO 8
20 READ F
30 SOUND 1,F,200
40 FOR N=1 TO 800:NEXT
50 NEXT
60 DATA 478,426,379,358,319,284,253,239

```

**Figure 2.3 A 'tempered' scale, in which the intervals are *not* equal. This was originally found by trial and error, and it sounds a lot better. This particular variety is a Major scale.**

```

10 FOR J=1 TO 8
20 READ F
30 SOUND 1,F,200
40 FOR N=1 TO 800:NEXT
50 NEXT
60 DATA 506,451,402,379,338,301,268,253

```

**Figure 2.4 Another Major scale, which this time starts a semitone lower.**

answer is in the way that it's arranged. If you listen carefully to this scale, you'll notice that the pitch differences between notes (called intervals) are nothing like equal. Even if we ignore the effects of tempering, there are two quite different sizes of intervals. The interval between note 3 and note 4 is only about half of the interval between the other pair of consecutive notes, and the interval between note 7 and note 8 is another small one. We call the bigger interval a tone and the smaller one a semitone. What makes a scale a Major one is that its semitones are between notes 3 & 4 and 7 & 8, with all the rest of the intervals being tones. You can start a scale on any other keynote, and to make it a Major scale, you have to get the semitone intervals into the same places. Try the scale in the program of Figure 2.4 now. It starts on a note which is a semitone lower than the scale in Figure 2.3, but it's still a Major scale, because it has the same sized intervals in the same places. Whatever the keynote, a major scale gives you a set of notes that are ideal for bright cheerful music.

## Some Minor problems

The Major scale is useful, but not all of the music that we want to play is going to be bright and cheerful. We need scales which contain notes that sound mournful and sad, or angry and bitter. Does that sound far-fetched? Music is something that makes direct contact with your emotions, and the only way we can describe in words how it sounds is by using words for emotions. Since we want to be able to convey the full range of emotions with music, then, we need the scales that are described as Minor. Now though there's one form of major scale, there are quite a lot of minor scales. A lot of these are of interest to the professional musician only, and we'll look at just one, the Natural Minor. Figure

```

10 FOR J=1 TO 8
20 READ F
30 SOUND 1,F,200
40 FOR N=1 TO 800:NEXT
50 NEXT
60 DATA 478,426,402,358,319,301,268,239

```

**Figure 2.5 A Minor scale, in this example, the Natural Minor. The difference is remarkable — the scale sounds sad and mournful.**

2.5 illustrates what this sounds like for a Scale of C-Minor. This has its semitones between notes 2 & 3 and notes 5 & 6. Listen to it, and compare it with the scale of C-Major. You can hear that the notes sound different — it's hard to imagine when you look at the numbers in the DATA line that the two scales could be so different. Just to rub in the gloom, try the slightly different scale whose DATA line is shown in Figure 2.6. This is called the Harmonic Minor, and it sounds even more gloomy than the Natural Minor.

```

10 FOR J=1 TO 8
20 READ F
30 SOUND 1,F,200
40 FOR N=1 TO 800:NEXT
50 NEXT
60 DATA 478,426,402,358,319,301,253,239

```

**Figure 2.6 For even more misery, try the Harmonic Minor scale!**

Not all scales are even of eight notes, one octave, because not all music has come from Europe. You'll find quite different scales used in music from India, from China, from Japan. A lot of our popular music is based on the African scale which came into Jazz music as the Blues Scale. Figure 2.7 illustrates this scale, which uses only seven notes rather than eight, and has a completely different arrangement of intervals. Before your head starts to reel with all these possibilities, though, remember that the

```

10 FOR J=1 TO 7
20 READ F
30 SOUND 1,F,200
40 FOR N=1 TO 800:NEXT
50 NEXT
60 DATA 478,402,358,338,319,268,239

```

**Figure 2.7 The Blues scale, with one note less, and different intervals. This is the great scale of Jazz.**

```

10 FOR J=1 TO 13
20 READ F
30 SOUND 1,F,200
40 FOR N=1 TO 800:NEXT
50 NEXT
60 DATA 478,451,426,402,379,358,338,319,
301,284,268,253,239

```

**Figure 2.8 A Chromatic scale, which uses all of the notes of other scales.**

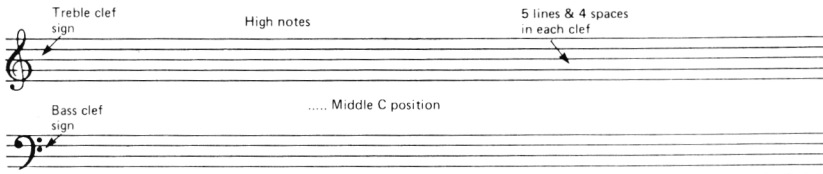
scale is just a convenient way of selecting notes that sound out the message of the music. You will certainly be concerned with selecting a scale if you are composing music, but it's more likely that you will be using music that some-one else has written. The snag here is that you might want to change the music so that it sounds better on your Amstrad, by shifting its keynote up or down. This is called transposing, and to do it correctly, you really need a good ear for the sound of different scales. To round off this business of scales, take a listen to the results of Figure 2.8. This is called a 'chromatic scale', and it consists of thirteen notes. These consist of all the notes that are used by any other scales, and in Western music, these are all the notes that exist between one keynote and its higher counterpart one octave up. Just to make life difficult, this is often called a 'twelve-note scale', because if you count only the first keynote and not the octave note, there are twelve notes in the scale, with the higher keynote in our program acting as number thirteen. Like computer buffs, musicians sometimes count inclusively, and sometimes don't! Some composers have written music using twelve-tone scales, but to me this music has all the appeal of thrashing myself around the face with a wet fish. Come back, Wolfgang Amadeus Mozart, you have never been out of fashion!

## **Naming notes and writing music**

Musicians have one considerable advantage as compared to computer-users — standardisation. If we stick to traditional Western music (no, not Country & Western, Hank, just Western Europe), the method of naming notes and writing music has been pretty standard for several hundred years. If you can read music which is written to this standard, then you can convert it into the form that's needed for your computer. No two computers

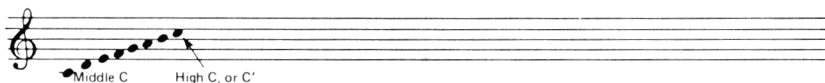
(apart from MSX) deal with music in identical ways, though, and it can be very difficult if you want to convert a music program that was written for one computer into the form that is needed by your Amstrad. For those of you who can work with machine-code, Chapter 7 deals with how to convert the SOUND instructions of the MSX computers (and others) into Amstrad form. Sometimes the only easy way is to convert from the X-brand computer program into written music, and from the written music into Amstrad program form. It may sound a very roundabout method, but it's often quicker and easier. All of that means that we need to be able to read conventionally written music. You don't have to be able to read as fast as a musician might, only fast enough to decide which note is which.

To start with then, music is written on a grid of five lines and four spaces which is called a staff or stave. The pitch of a note is indicated by these lines and spaces, as Figure 2.9 shows. For



**Figure 2.9 Music manuscript. Music is written down by putting marks in a grid of lines and spaces. Each line or space represents one note in the scale of C Major. To make music easier to read, the high notes are put on to the upper part (treble clef) and low notes on the lower part (Bass clef).**

a lot of written music, we use two sets of staves, with markings that indicate the range. The one which is marked with the '&' sign is called the treble stave, and it consists of the higher range of notes. The one that is marked with what looks like a reversed letter 'C' is the bass stave, and it consists of the lower range of notes. These staves are arranged with space for one line between them. This line is never printed in, because it would make the music difficult to read. The note that belongs on this line is called Middle C and it is in the middle, between the staves. When a note is written in this place, a short line is put through it to show that it belongs in this position. Figure 2.9 also shows how these stave positions for notes are named, and these names are the



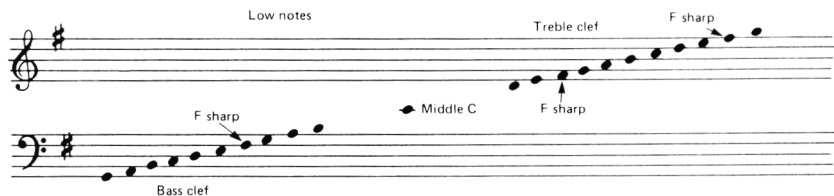
**Figure 2.10** One octave of the scale of C-Major, treble clef, written down. There's nothing to show which intervals are tones and which are semitones.

ones that we use for the notes. The naming system simply uses the letters of the alphabet from A to G.

Now when you look at this, you'll see one flaw in the system. There's absolutely nothing that shows how a scale is arranged. If you look at the scale of C-Major put on to a staff (Figure 2.10), you'll see that there is nothing to show you where the semitones come. That's the way it is, I'm afraid, but it does have the advantage of letting you use the same staff to write with any sort of scale you want to use. At the same time, though, it gives a rather unfair advantage to the scale of C-Major, because this one can be written on the staff with no modifications, just by using the natural positions. It's the only scale that **can** be written in this way. For any other scale, you have to use special markings to indicate notes that are **not** the same as they would be in the scale of C-Major. These marks are called sharps and flats, and when they are used to show a scale which is not the scale of C-Major, the arrangement is called a 'key signature'. If you just want to have an odd note that is sharp or flat, you put a sign next to the note, and this type is called an 'accidental'. The key signature signs are put at the start of each line of music, and they affect each C note that is in their line or space.

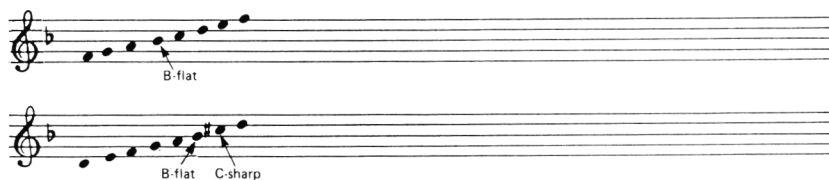
The sharp mark looks like the computer's hashmark, and it means that the note which is sounded is to be a semitone higher than the position of the note for C-Major. Now if you put a sharp sign on the top line of the treble staff, that means that this note will always become a semitone higher. The 'natural' note, with no sharp mark, is F, and by putting a sharp on this top line, you always play F# (pronounced F-sharp) instead of F natural. Now, as it happens, if you play a Major scale that starts with G, you need to play F# in place of F natural. This, then, is a very convenient method of showing which scale you are using as well as which note has to be modified. This particular key signature,





**Figure 2.11 A scale of G Major which extends over three octaves. Each marked *F* is played as *F-sharp*.**

the F# marking, belongs to the key of G-Major. The reason should be clear by now — it's because with the F# permanently in place, we can play a Major scale which starts with G. Figure 2.11 shows the notes of this scale.



**Figure 2.12 The flat sign is used to show that a note one semitone lower than the printed position is to be played. With a B-flat permanently in place, we can play a scale of F-Major (a) or D-Minor (b). The Minor scale needs another change — C# in place of C natural.**

The other mark is the flat, which looks like a droopy letter 'b'. When this mark is in place, then you sound a note which is a semitone **lower** than you would expect in this position. Look, for example, at Figure 2.12. This shows a flat mark on the middle line of a treble staff. This note position is B, and so every B becomes a B-flat, a semitone lower than B. Now if we happen to start a Major scale on the note F — a scale of F-Major, then we need to use B-flat in place of B, and Figure 2.12 (a) shows the notes of this scale. We can also start a natural Minor scale on the note D, and find that this also needs B-flat in place of B. The use of this mark, then, can mean that you are working with the scale of F-Major or the scale of D-minor! The notes of the D-Minor scale are shown in Figure 2.12 (b), and you can listen to the notes of the two scales in the program of Figure 2.13. The scales sound quite different, one major, one minor, and yet they use the same staff. At first sight, this can be confusing, but it's really very useful to be able to write all music with the same set

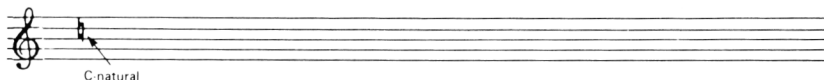
```

10 CLS:PRINT"F-MAJOR"
20 FOR K=1 TO 2
30 FOR J=1 TO 8
40 READ F
50 SOUND 1,F,100
60 FOR N=1 TO 500:NEXT
70 NEXT
80 FOR N=1 TO 3000:NEXT
90 CLS:PRINT"D-MINOR"
100 NEXT
110 DATA 358,319,284,268,239,213,190,179
120 DATA 426,379,358,319,284,268,239,213

```

**Figure 2.13** How these two scales, F-Major and D-Minor, sound on the Amstrad.

of grid lines, and adjust some of the notes with the sharp or flat signs when you need to. There's another sign, the natural (Figure 2.14) which is used if for some reason you have marked a line or space with a sharp or flat and you don't want to use it.

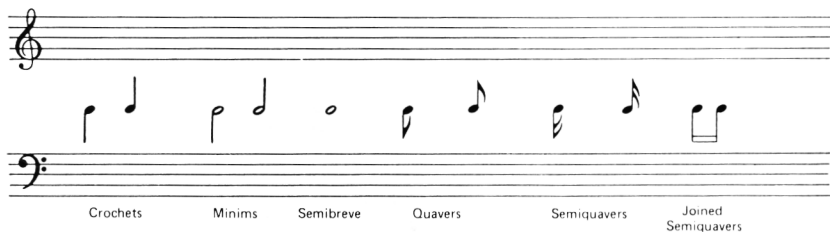


**Figure 2.14** A *natural* sign. This is used to show that the normal note should be played, cancelling the effect of a sharp or flat sign at the start of a line of music.

The most commonly used musical instrument is the human voice, and the staves were originally designed for writing music for singers. The next most common instrument nowadays is still the piano, and we can include with the piano all instruments that use the piano keyboard. The piano is arranged so that it can play any chromatic scale, and any set of notes that belong to a chromatic scale. This means that the note A-sharp is the same as the note B-flat; D-sharp is the same as E-flat and so on. In some Eastern music, these notes would not be the same, and instruments which do not use keys (guitars, violins for example) can play notes which are quarter-tones, so that B, B-flat, A-sharp, A would sound like four different notes. On the piano, these are only the three notes, and for most Western music, the other instruments follow the example of the piano. When you get to the Amstrad musical pitch numbers, you'll see that they deal with sharps only (because the hashmark is easier to get on a keyboard than the flat mark). You can, however, generate notes

that are quarter tones if you want to simulate Eastern music.

That deals with how we map out notes. With this mapping, you can write the pitch of a note just by putting the note in position on a line or space. If the note that you want to use comes above the range of the treble staff or below the range of the bass staff, then you can draw in extra lines above or below. These lines are not put in all the way along a page, but only through the notes that need them. Once again, like the Middle-C line, this is done so that you can see the staves clearly, with no confusion due to extra lines. In addition to pitch, however, a note has duration and volume. We need some method of indicating this in music. Here, too, because the history of written music is so long, things have become standardised, but in the course of this standardisation there has been a bit of confusion. There are, in fact, two ways of indicating the duration of the notes. One way is relative. This means that there is a note of 'standard' duration, and all the other notes are measured relative to this. Once again, there is a curious similarity to computing (machine code, too), because the multiples are all twos. The standard length note is called the *crochet*. It's marked by its shape (Figure 2.15), with a black dot and a tail which can point either up or down. A note which lasts for twice as long as a *crochet* is called a *minim*, and its shape is a circle with a tail. The note which lasts for twice as long as the *minim*, four times as long as the *crochet*, is the *semibreve*, which is marked by a rugby-ball shape, with no tail. There's a note twice as long as this, the *breve*, but you hardly ever come across it. Going to the shorter end of the scale, a note which lasts for only half as long as the *crochet* is called the *quaver*. It looks rather like the *crochet*, but with a hook on the



**Figure 2.15** The durations of notes are indicated by the note shapes. You often find that the short notes (quavers and shorter) are joined by the tails.

tail. Put another hook on the tail, and you have the semiquaver, which lasts only half as long as the quaver, a quarter of the duration of the crochet. Yet another hook gets us to the demisemiquaver, which lasts half as long as the semiquaver. If you like them really brief, there's also a hemidemisemiquaver!

All of this is starting to look like the old problem of how long a piece of string is. The thing that we really need to know is how long a crochet lasts. Unfortunately, that's the slippiest item in the whole of music. The traditional method is to divide music into bars which take equal amounts of time, and to say whether you want the music played fast, medium or slow. You then write what your unit of timing is to be and how many of them you want in each bar. It all looks very technical, but it's useful only to the musician who has learned by experience what these figures (the 'time-signatures') mean. For the rest of us, it looks vague, to say the least. It became a lot less vague around 1823, when the metronome was invented. The metronome is just a piece of clock-work which uses a little swinging pendulum to mark time. The timing is adjustable, and the adjustment is usually marked in terms of the number of crochets per minute. Now you might think that once the metronome had been invented, all the older ways of describing timing would have been abandoned. Don't you believe it! Though the metronome was taken up and used by composers (including Beethoven, who, since he was deaf greatly appreciated the idea), you still don't see all that much music marked with metronome settings. The reason is that musicians feel that a metronome setting ties them down too much. If the only indication of time is a remark like 'walking pace', then the musician can decide what he/she calls a walking pace. That's just one of the reasons why a piece of music that you may have heard a hundred times can still be made to sound fresh and different. Once you fix a metronome reading, you have to keep to its pace, and that limits your scope for using your own ideas about how the music should sound.

What do we do, then, about transferring music to the computer? The best approach is to assume that if the music is to be played at an ordinary pace, then a crochet should last for about half a

```

10 FOR J=1 TO 16
20 READ F
30 SOUND 1,F,50
50 NEXT
60 DATA 478,426,379,478
70 DATA 358,478,319,478
80 DATA 284,478,319,358
90 DATA 379,426,478,478

```

**Figure 2.16 A program for a tune that is played with each crochet lasting for half a second.**

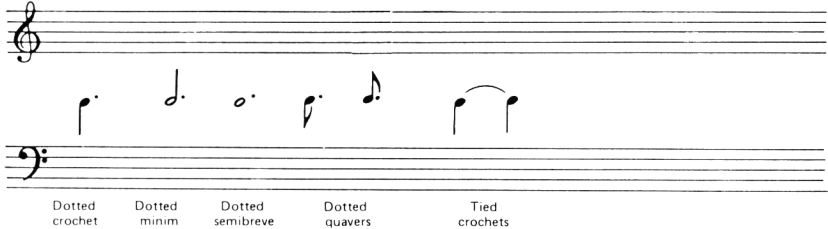
second. This corresponds to a metronome setting of 120. The Amstrad SOUND command gives you a default (meaning what you get if you don't change it) setting of 1/5 second for a note, corresponding to a metronome setting of 300. This is a fairly fast setting. Figure 2.16 uses the half-second crochet in a little set of sixteen notes, just to let you hear the pace. If you remove the last part of the SOUND instruction (which reads ,50), then you can hear the Amstrad default speed. This corresponds to the sort of pace which a musician would label as *allegro*. No, there's no pace called Montego.

## A bit of flexibility

If there's anything that a musician, composer or performer hates, it's being tied too rigidly to a measurement of anything, whether its timing or even pitch of note. As we have seen with the business of the time of a crochet, a bit of flexibility is very strongly valued in music. For those of us who are not dedicated musicians (though I confess to having a clarinet, a piccolo and a trumpet in the computer-room) this refusal to be tied down is a nuisance when it comes to writing music programs for the CPC464 or CPC664. I said at the start of this book that 'sound is different', and this is just one of the differences. You can, with a bit of practice, soon learn to transfer the written notes of a music score into the SOUND code numbers for the Amstrad computers. You'll always be slightly uncertain, however, whether the timing is just right, and only your ear will tell you. This is what I meant when I said that you can't just look at a program for making music and knew what it will sound like. With experience, you can tell what the notes are, and you can even get a pretty good idea of the

timing, but unless you have a really good musical ear, you still don't quite know what it will sound like. Most of us are cursed with putty ears as far as music is concerned, but this is something that you can work at. With a bit of practice, you can adjust your programs here and there and end up with them sounding quite a lot more musical. Before we start on that, though, we'd better look at a few of the ways in which composers and performers make this written music into a more flexible form.

One method is by using dotted notes. The standard note lengths are the ones that were illustrated in Figure 2.15. Every now and again, though, you need a note that is not exactly of one of these lengths. This is done by placing a dot following a note. The effect to the dot is to make the note half as long again. For example, if you count a crochet as a single unit of time, then a dotted crochet counts as one and a half. If a standard minim is two units, then a dotted minim is three. If a standard semibreve is four units, then a dotted semibreve is six. The idea of dotting notes extends to the shorter notes also. Another way of making the length of a note differ from the normal is by ties. If you draw a bowed line that joins two identical notes, it means that the two have to be sounded as one note whose duration is the combined duration of the two notes. Figure 2.17 illustrates this and the dot system of extending note durations.



**Figure 2.17 Dots and ties. Following the note with a dot extends the note by 50%. The dot is put in the space (not on the line) following the note. The tie mark linking two identical notes means that they should be played as one note.**

Another effect in music arises from having rests. Every now and again, you want the notes to stop, and since the time of silence is as important as the time of a note, there must be markings in written music to indicate how long these silences or rests are to be. Figure 2.18 shows the symbols for the rests, which are



**Figure 2.18 The symbols for 'rests' or silences. These use the same timing as notes, and can also be dotted.**

geared to the same relative time settings as the notes. You can use dots and ties on these symbols as well if you want to create times of rests that are not catered for with the ordinary symbols. Anything that is longer than a few notes will be played in 'phrases'. This means that the notes will be grouped, with short silences between the groups of notes, and so the composer must be able to specify the time of the silence. Since the pace of the music will still be set by using vague words like **allegro** and **andante** (which means at a walking pace), the musician can use a bit of discretion about how long the silence is, and so vary the effect of the phrasing.

The musician's main guide to the problem of how long a crochet should be sounded is the 'tempo'. This is Italian for 'time', and the words that musicians use are Italian words. Figure 2.19 lists these timing words, in order of very slow to very fast, and with **approximate** metronome and CPC464 speed figures. I must stress that these numbers are approximate, because musicians

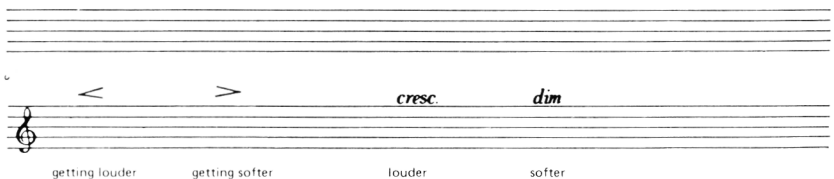
Word	Meaning	Metronome	CPC464 Duration
Largo	slow	60/70	100/85
Lento	slow	60/70	100/85
Grave	slow	60/70	100/85
Andante	rather slow	120/75	50/80
Allegretto	fairly fast	150/120	40/50
Allegro	fast	300/150	20/40
Vivace	lively	300/150	20/40
Con brio	dashing	300/150	20/40
Presto	fast	600/300	10/20
Prestissimo	very fast	1200/400	5/15

**Figure 2.19 Musical speeds. The words that indicate speed are deliberately vague, and these metronome figures and CPC464 timing numbers are approximate only.**

will never be tied down on these figures. If you see these tempo words being used in a score, however, the figures in the list will be a good guide, and will give you a starting point at least. You can then use the judgement of your own ears to decide whether you want the piece played faster or slower, but at least you have something to start you off. Appendix A lists these and other words, Italian and otherwise, which you will find cropping up in written music.

We've said nothing about volume, the amplitude of sound. Once again musicians like a bit of flexibility about this, and you can't really blame them. Music, you see, may be played by a variety of instruments, some of which can make a lot more noise than others. Because of this, you can't have an absolute figure of volume, and it's this that makes the difference between real live music and a lot of synthesiser systems. The traditional method is to use the letter *f* (for 'forte') to mean loud, and *p* (for 'piano') to mean soft. The loudest note your instrument can play is indicated by *fff*, and the softest note that you can achieve by *ppp*. If the music is unmarked, you play at somewhere between these two. Where the composer wants the sound to get louder as you play some notes, there is a 'crescendo' (pronounced creshendo) mark over the notes, or music that gets softer, the opposite mark is the 'diminuendo'. Both of these marks are made by having signs over the staff like the less-than and greater-than signs (Figure 2.20).

To crown it all, there's one effect which is practically unwritable — rhythm. A rhythm comes about when notes are played in groups, and certain notes are stressed. A note can be stressed by playing it slightly louder, or slightly longer, than the others.



**Figure 2.20** The signs that show when music should change volume, either louder or softer.



Written music shows the grouping of notes by drawing vertical lines across the staff. The space between two lines is called a bar, and in a given piece of music, there are always a fixed number of crochets in a bar. This doesn't mean that **all** of the notes are crochets, though. If you music uses four crochets in a bar, you can have a single semibreve in a bar, because its time is the time of four crochets. You could also use two minims, or a dotted minim and a crochet, anything in fact that added up to the correct time of four crochets. To get the rhythm, then, you would stress one of the notes in each bar, usually the first or the second. This can become monotonous, though, so very often the stress can be varied, strong in places, weaker in others. You might also want to syncopate, which means that the stressed note is not always in the same position in each bar. Once again, all this adds up to the fact that even when you have the correct collection of notes, played with the correct times, you still might not have anything that sounds like music! Until you get your ears accustomed to listening to music, you'll find it very difficult to appreciate all this, but in the following Chapters, we'll be looking at programming methods and showing how these subtle effects can be achieved.



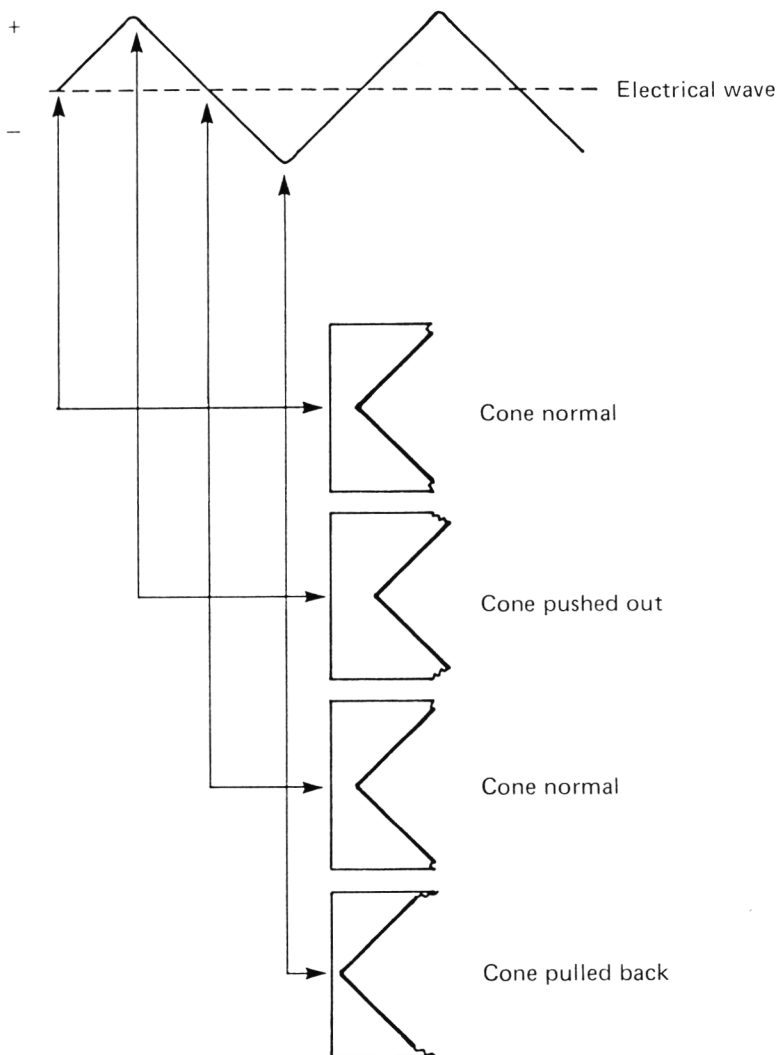
# 3

## The Amstrad Sound

A computer is not a piano, and you'd look rather silly trying to play Harrier Attack on a violin. The computer produces sound by a bit of fiddling with electrical signals which are then used to work a loudspeaker. It can never make a sound that is **exactly** like a piano, or a violin or a clarinet, but you can produce a lot of sounds that are musically pleasing. In addition, you can program your CPC464 to create a lot of sound **effects** that no musical instrument could ever produce. In this Chapter, we'll look at how sound is produced, and how we can control what we get.

To start with, a loudspeaker makes a sound when it gets an electrical signal. This electrical signal consists of a rising and falling electrical voltage. As the voltage rises, the cone of the loudspeaker is pushed forwards, pressing the air outwards. As the voltage falls, the cone is pulled backwards, de-compressing (rarifying) the air. If the cone of the loudspeaker is pushed alternately forwards and backwards 100 times a second, the result is a sound wave whose frequency is 100 Hz. If we use a large electrical voltage, the cone of the loudspeaker moves further, and the amplitude of the sound is greater. The sound that comes from the loudspeaker, then, is a wave which is a copy, as near as is possible, of the electrical wave of voltage, Figure 3.1.

As it happens, everything that happens in your computer is decided by electrical waves, and they all have the same shape — square. A square wave simply means that an electrical voltage is switched on suddenly, held steady for a short time, and then switched off suddenly. The signals that make things happen in the microprocessor of the computer are called 'clock pulses', and they are switched on and off pretty rapidly. In the CPC464,



**Figure 3.1 How the electrical signals to the loudspeaker affect the cone, pushing and pulling it and so creating sound waves.**

in fact, these signals go through a cycle of off-on-off at a rate of 4 MHz. That's four **million** Hertz, which is a long way beyond anything that we can use as sound. We can make this clock operate slower switches, however, just as we can make the ticking of a watch operate a minute hand. By programming, for

example, we could make a voltage stay high for 2000 clock cycles, and low for another 2000 cycles. Since 2000 clock cycles takes a time of half a thousandth of a second, we would in this way create a wave which was at high voltage for half a thousandth of a second, and low for the same time. This gives a wave whose total time is one thousandth of a second, and if we continued with these, we would have a thousand waves per second. This is a reasonable sound frequency of 1 kHz, one thousand Hertz. What it all boils down to is that if we use a routine in machine code, we can feed a number to this routine, and it will use this number to keep a voltage high for that number of clock cycles, and low for the same number. This will generate a wave which, if we choose numbers wisely, will be in the range of sound frequencies that we can hear.

The CPC464 does not use such large numbers as 2000 directly. Instead, the numbers are scaled down, by using a slower clock rate for the sound generating system, and for a frequency of 1000 Hz (1 kHz), the number that is used is 125. The machine code program which actually carries out the work will then multiply this up into the size that is actually needed in the timer. Figure 3.2 shows the formula which allows you to find what pitch number will correspond to a frequency. For example, if you want to use a frequency of 1000 Hz, then 125000/1000 gives 125, which is the pitch number for the SOUND command. What is going on inside, however, is a lot more complicated, as you will have gathered by now.

Tone Period No. = 125000/frequency  
(frequency must be in Hertz. If the frequency is given in kHz, then multiply by 1000.)

Example: Frequency of 1.5 kHz. This is 1500 Hz, and so the tone period is 125000/1500, which is 83 (nearest whole number).

**Figure 3.2 The formula for the CPC464 pitch numbers.**

## The BASIC beep

The simplest sound is the beep that you can use in programs to indicate that some attention is needed. To get this beep (frequency about 1.389 kHz), you simply put `PRINT CHR$(7)` into your program. The time of the beep is fixed, and its frequency and volume are also fixed, so there's not a great deal that you can do with it apart from making it last longer. If you want a BEEP for special occasions, for example, you can use:

```
FOR X=1 TO 20:CHR$(7);:NEXT
```

which will give a long beep. The semicolon following the `CHR$(7)` is needed to stop the computer taking a new line **on the screen** every time the loop goes round. You can also program a 'dotted beep', using:

```
FOR N=1 TO 20:CHR$(7);:FOR J=1 TO 400:NEXT:NEXT
```

which will give you a repeating beep. You can alter the timing of the silence between beeps by changing the number in the J loop. That, however, just about exhausts the possibilities of the BASIC beep.

## Superior SOUND

The sound system of the CPC464 is based around a unit which is called the Programmable Sound Generator (PSG). This is one of the chips inside the case of your Amstrad, and its type number is AY-3-8912, made by General Instruments of the U.S.A. It can generate square waveforms, nothing else, and the shape of this waveform will then be altered only by the loudspeaker. This is mainly because the cone of the loudspeaker cannot move as fast as would be needed for perfectly vertical sides, and cannot remain at rest long enough for the flat top of a square wave. If you feed the sound signals to a high-quality amplifier and loudspeaker system, you will hear a very different type of sound, but no loudspeaker can ever reproduce perfectly square shaped waves. The sound generator chip uses clock signals that have been obtained by dividing the frequency of the clock pulses from the main computer, and the chip action is programmed by

the main computer. It will generate three channels of sound, two of which can be kept separate to use as stereo channels. The third channel feeds into both stereo outputs. There is also a noise-generating circuit that can be connected to any or all of the channels. The chip contains memories (registers, if you know about microprocessors) which are used to store set-up instructions, and in addition, the main memory of the computer can also be used to store a set of commands. You have probably noticed in the examples so far that the computer screen showed the READY prompt long before the sound stopped. This is because complete sound instructions are held in the memory, using a type of system which is called FIFO, meaning First-in, First-out. This memory can hold five sound instructions for each channel, so that if you have a tune that consists of five notes only, the computer processes this almost immediately, putting the information into the memory of the sound chip, and then getting on with other processing. By the time all five notes have played, the computer may be working on something quite different. This can cause you problems if you want the sound to keep in step with something that is appearing on the screen, and that's something that we'll have to look at later. In the examples so far, I have made use of time delays to overcome this type of problem. This FIFO action of memory is referred to in the manuals as the 'sound queue', and this is a good description of how it works.

In this book, we'll keep mainly to programming methods in BASIC, because you can hear the results more quickly and get to grips with it more easily. If you eventually intend to program the sound chip by machine code, then the methods that are needed in BASIC still hold good, but you will have to know how to make use of the registers in the sound chip. That's fairly advanced programming, and you'll need to make use of the Amstrad Firmware Specification manual for details of machine code calls. Advice on machine code programming for readers who have had experience with this type of program is in Chapter 7. Right now, what we're going to look at is how we can work with the straightforward SOUND instructions of the CPC464 BASIC.

SOUND is an instruction word which can be used in either a simple way, or with added complications, and we'll build up to the more complicated uses later. The simplest form of the SOUND instruction uses just two numbers following SOUND. The first of these is the channel number. Now when we are dealing with the simple form of the SOUND instruction, the channel numbers can be 1, 2 or 4. If we call the channels A, B, and C, then using the number 1 gives channel A, 2 gives channel B, and 4 gives channel C. No, Marilyn, you can't use 5 and get Channel No.5. You can, however, use zero, and get no sound at all. The channels refer to the fact that you can have three independent sound generators. This means that you can simulate the effect of three instruments playing all at the same time. You can also add the numbers, so that  $1+2=3$  gives you channels A and B playing,  $2+4=6$  gives you channels B and C,  $1+4=5$  gives you channels A and C, and  $1+2+4=7$  gives you all three. Adding numbers like this forces more than one channel to play the same note.

The second number in the SOUND command is the pitch number. Now it's a little bit misleading to call it a pitch number, because a high number value gives a low pitch, and a low value number gives a high pitch. It would be more accurate to call it a time number, because it affects the timer that generates the square wave. The Amstrad manuals refer to it as a tone period number, and we'll keep to this description from now on. The values that can be used here range from 0 to 4095. If you use zero, there will be no sound, and this can be a useful way of getting a silence. Apart from this value of zero, the numbers which are very low or very high really aren't particularly useful. Very low bass notes simply can't be reproduced correctly by the tiny loudspeaker, so that unless you have a **very** good Hi-Fi system connected, it's not a good idea to use tone period numbers larger than about 1000. You **can** get sounds with higher numbers, but they are not particularly musical. With an amplifier and a good big loudspeaker, you can get really impressive bass notes, but not with the built-in loudspeaker. You won't get very impressive bass notes with earphones either unless you have fairly sensitive earphones.



On the high pitch (low number) side, there isn't much point in programming notes that only a bat can hear. The manual shows values down to 16 in the musical note lists, and this is about as high as you can usefully use. Above this, the loudspeaker doesn't cope very well and you can hardly hear the note. More important, it gets much more difficult to get the exact musical notes that you want when you use these numbers, because the intervals get larger as the numbers get smaller. This is because of the formula,  $125000/(\text{frequency})$ . If we turn this around, it becomes  $\text{Frequency} = 125000/(\text{Tone period})$ . Now for a tone period of 1000, the frequency is 125 Hz. If we use a tone period of 995, just five less, then the frequency is 125.63 Hz. You would really have your work cut out to tell the difference. If you use 20 as the tone period, though, the frequency is  $125000/20 = 6250$  Hz, or 6.25 kHz. Take a tone period of five less, 15, and the frequency is  $125000/15 = 8333$  Hz or 8.33 kHz. So a difference of five in the tone period number makes a difference of 0.63 Hz in a bass note, but a difference of more than 2 kHz in a treble note!

```

10 FOR N=1000 TO 10 STEP -5
20 SOUND 1,N
30 NEXT

```

**Figure 3.3 Using a SOUND instruction in a loop, with the loop counting number acting as the tone period. We would normally use integer numbers in a loop of this type.**

This is made clear in the example of Figure 3.3. This uses the SOUND instruction in a loop, playing notes by using the tone period numbers in the loop. Because the loop uses steps of -5, the frequencies change only very slightly at first, but by the time you get to around Middle C, the notes are almost a tone apart, and at the upper end of the scale the differences are enormous. Using SOUND in a loop with the loop number controlling tone period is not quite so useful! You can, however, get just the opposite — steps of frequency. Because pitch number is  $125000/(\text{frequency})$ , you can program as in Figure 3.4. This takes the loop from frequency values of 100 to 5000 in steps of five. Within the loop, the tone period number is calculated for each frequency by setting  $P=125000$ , and using  $P/N$ . Don't use  $125000/N$  in calculations like this, because it slows the computer

```

10 P=125000
20 FOR N=100 TO 5000 STEP 5
30 S%=F/N
40 SOUND 2,S%
50 NEXT

```

**Figure 3.4 Programming with frequency values rather than with tone period numbers.**

down. Numbers are stored in a coded form when you assign them to a variable like P, but if you keep them as numbers, they have to be converted each time the number is used, and that slows down the loop. You'll find that this loop needs no slowing! This time, the change of frequency is most noticeable at the low frequency end, and the changes are very gradual at the high frequency end.

## Timing the note

If you use only a channel number and a tone period number, then each note will be sounded for one fifth of a second. If you are happy about this, and it's quite useful for a lot of musical effects, then you need use only two numbers in a SOUND instruction. By using a third number, however, also separated by a comma, you can specify a different time of note. For the simple use of the SOUND instruction, the range of this time number is 1 to 500 or so. You can use zero, and you can use negative values, with numbers up to 32767, but for the moment, keep to the simple use of this part of the instruction. When you use a positive number, it is equal to the number of hundredths of a second of duration. Using 100, for example, gives 100/100 of a second, which is one second. A value of 500 gives a five second note, which is longer than you would normally use for music. Going to the other extreme, a value of 1 gives a note which lasts only for 1/100 of a second, and it can be useful in some of these loops. Take a listen to the effect of Figure 3.5, which uses a loop with frequency values from 1000 to 2000, with a note time of one hundredth of a second. This gives a rapidly-rising type of note, and to make it sound more interesting, I have put it inside another loop so that it repeats five times. This is a very good 'disaster-warning' type of note, which you could use

```

10 P=125000
20 FOR J=1 TO 5
30 FOR N=1000 TO 2000 STEP 10
40 SOUND 1,P/N,1
50 NEXT
60 NEXT

```

**Figure 3.5** A note with rapidly rising pitch, using a loop.

in programs to signal something unpleasant about to happen. In general, if you want to use loops with either frequency numbers or tone period numbers, then make the time of each note as short as possible by using a 1 as the third number in the SOUND instruction.

## Make thy musick

It's time we got to grips with the actual business of making music with the CPC464. We'll start with a simple melody, using Yankee Doodle. You'll find that in this book, all the melodies are either traditional or of my own composition. The reason is that a composer's copyright lasts for fifty years after he dies, and copyright fees can cost quite a lot. This is also the reason that you don't hear the latest tunes being played in your computer games either! Getting back to the music, Figure 3.6 shows the score of the part of the tune that we'll use. The unit of time is the crochet, and there are two crochets in each bar. This information comes from the 'time signature' at the start of the line of music. The figure '2' on top means two units in each bar, and the '4' underneath means crochets. An '8' in the lower position would mean quavers, a '2' would mean minims. As it is, the unit is the crochet, and two crochets to the bar means four quavers, which is what most of the melody uses. If you've never seen music script before, you might be wondering why the tails of the quavers are shown joined together. There's no particular reason, except that it looks neater, but in this case it can be used as a rhythm indicator. If



**Figure 3.6** The first part of the score for Yankee Doodle. The vertical lines mark the end of bars, the divisions of the music. The twin vertical bars mark the end of a section.

you place more stress on the first note of each connected pair, the music sounds better.

However, that's going a bit too fast for the moment. What we need to do is to transform this piece of written music into a form that we can use in a program. The easiest way is to ensure that we will read the note pitch values from a DATA line. Practically all of the notes are quavers, and the three that aren't can be dealt with by playing two quavers (same pitch) together to sound like a crochet. The last 'note' is a quaver-length rest — this is because this example uses only the first line of a tune that continues for several lines more — but it's enough for an example. The first thing to do, then, is to count notes. There are 31 quavers, counting each crochet as two quavers, so we need a FOR...NEXT loop that counts from 1 to 31. The tone periods will be read from a DATA line, and we can put them in later. We can make up a skeleton program then, in the form:

```
10 FOR N%=1 TO 31:READ K%
20 SOUND 1,K%:NEXT
```

which will attend to reading the values, and sounding them. Once this has been done, we need to write the DATA lines that will carry the melody. A good rule here is to write a DATA line for each bar. It might look wasteful, but it makes things a lot easier if you are trying to chase a faulty note. There's no wasted time either, because the computer can put notes into the sound queue a lot faster than the sound chip will be playing them. In this first effort, because the notes are quavers, we're going to use the 'default' timing of one fifth of a second.

Now if you read music fluently, it's perfectly easy to convert from the score to the tone period numbers. If you don't, then the safest method is to carry this out in two stages. In the first stage, write the **names** of the notes under the score. The tune starts with the C above Middle C, and you can mark this as C' to distinguish this from Middle C. If you use the apostrophe mark like this for all notes in the octave above the end of the C-Major scale, you won't confuse a high D with a low D or the other way round. With the notes written in, the score looks as in Figure 3.7. Count them



**Figure 3.7** The score with the note names written in. This is always a useful step unless you read music really well. It's particularly useful if there are a lot of sharps or flats in the scale.

again, and remember that you need to show two notes where there has been a crochet. You can now look up these notes in the Appendix of the CPC464 manual, where the tone period number for each note is illustrated. Keeping to four numbers in each line, we need eight lines. The last line ends with a Ø, but this is not actually read by the program. Later, we'll see that this can be a way of ending a program that is operated by a WHILE...WEND loop. The result is in Figure 3.8, and when you run this, you'll hear that the choice of timing is about right, because this is a quick little tune. The only thing that's wrong with the way the computer plays it, in fact, is that the notes which should be separated are not.

```

10 FOR N%=1 TO 31:READ K%
20 SOUND 1,K%:NEXT
30 DATA 239,239,213,190
40 DATA 239,190,213,319
50 DATA 239,239,213,190
60 DATA 239,239,253,253
70 DATA 239,239,213,190
80 DATA 179,190,213,239
90 DATA 253,319,284,253
100 DATA 239,239,239,Ø

```

**Figure 3.8** Yankee Doodle written as a CPC464 program.

Now this isn't quite so easy to deal with. You can, of course, modify the program so that it looks as in Figure 3.9, with a line 30 which will cause a brief silence between notes. This makes most of the tune sound better, with short sharp notes — the correct musical name is **staccato**. The trouble is that it also causes separation of notes that ought to be played together. In a short and simple tune like this, we can overcome the problem quite easily, as Figure 3.10 shows. The places where we don't want the staccato effect occur when N% has values of 13, 14, 15, 16 and 29. By changing line 30 so that the short silence is

```

10 FOR N%=1 TO 31:READ K%
20 SOUND 1,K%
30 SOUND 1,0,2
40 NEXT
50 DATA 239,239,213,190
60 DATA 239,190,213,319
70 DATA 239,239,213,190
80 DATA 239,239,253,253
90 DATA 239,239,213,190
100 DATA 179,190,213,239
110 DATA 253,319,284,253
120 DATA 239,239,239,0

```

**Figure 3.9** Using brief silences between notes to make the notes sound more clearly separated.

```

10 FOR N%=1 TO 31:READ K%
20 SOUND 1,K%
30 IF (N%<13)OR(N%>16 AND N%<29) OR (N%>
29) THEN SOUND 1,0,2
40 NEXT
50 DATA 239,239,213,190
60 DATA 239,190,213,319
70 DATA 239,239,213,190
80 DATA 239,239,253,253
90 DATA 239,239,213,190
100 DATA 179,190,213,239
110 DATA 253,319,284,253
120 DATA 239,239,239,0

```

**Figure 3.10** Joining selected notes by making use of the note counter number.

added only for other values of N%, we can make the notes join up in the few places where we want them joined, and remain staccato for all the other notes. It's not a solution that would be useful for more elaborate tunes, because there would be too many conditions to get into the 'silence' line, but it's a perfectly good solution for this type of example. After all, many of your applications for sound will be to little phrases of music just like this one.

Figure 3.11 shows another example, starting with the score (a) for part of the traditional song, the Keel Row. Now this uses minims, crochets, quavers and semiquavers, and in most of the bars there is a crochet followed by a dotted quaver tied to a

E<sup>4</sup> C<sup>4</sup> E<sup>4</sup> F<sup>4</sup> D<sup>4</sup> F<sup>4</sup> E<sup>4</sup> C<sup>4</sup> E<sup>4</sup> D<sup>4</sup> B<sup>3</sup> G<sup>3</sup> E<sup>4</sup> C<sup>4</sup> E<sup>4</sup> F<sup>4</sup> D<sup>4</sup> F<sup>4</sup> E<sup>4</sup> C<sup>4</sup> D<sup>4</sup> B<sup>3</sup> C<sup>4</sup>  
 - C<sup>4</sup> E<sup>4</sup> G<sup>4</sup> E<sup>4</sup> C<sup>4</sup> G<sup>4</sup> F<sup>4</sup> E<sup>4</sup> C<sup>4</sup> E<sup>4</sup> D<sup>4</sup> B<sup>3</sup> G<sup>3</sup> C<sup>4</sup> E<sup>4</sup> G<sup>4</sup> E<sup>4</sup> C<sup>4</sup> G<sup>4</sup> F<sup>4</sup> E<sup>4</sup> C<sup>4</sup> D<sup>4</sup> B<sup>3</sup> C<sup>4</sup>

```

10 J%=1
20 WHILE J%<>0
30 READ J%,D%
40 SOUND 1,J%,8*D%
50 SOUND 1,0,2
60 WEND
70 DATA 190,4,239,3,190,1
80 DATA 179,4,213,3,179,1
90 DATA 190,4,239,3,190,1
100 DATA 213,3,253,1,319,4
110 DATA 190,4,239,3,190,1
120 DATA 179,4,213,3,179,1
130 DATA 190,3,239,1,213,3,253,1
140 DATA 239,8
150 DATA 239,3,190,1,159,3,119,1
160 DATA 142,4,159,3,179,1
170 DATA 190,4,239,3,190,1
180 DATA 213,3,253,1,319,4
190 DATA 239,3,190,1,159,3,119,1
200 DATA 142,4,159,3,179,1
210 DATA 190,3,239,1,213,3,253,1
220 DATA 239,8,0,0

```

Figure 3.11 The Keel Row arranged for Amstrad. The score (a) has the note names pencilled in, and the program (b) has data lines that have been obtained by looking up the tone period numbers for the notes. On the score, the 'hooks' of the semiquavers are normally shown as stubs when notes are joined as shown.

semiquaver. The time is once again two-four, meaning that the unit is the crochet, and there are two beat units to each bar. The problems here are to get the **relative** timing right, and then to get the **absolute** timing correct so that the tune goes at a good pace. This last part is purely a matter of judgement, and a real Tynesider might think I have made it too slow. We start, as usual, by putting the letter names under the notes. Most of the notes are in the octave above the middle one, with the odd note two octaves above Middle C. The timing will be set in this example

by including a timing number with each pitch number in the DATA lines. For each note, then, we will read J%, which is the tone period, and D%, which is a duration number. We **don't** have to get these duration numbers exact, however, as long as they are **relatively** correct. In this example, I have used a duration number of 4 for a crochet, 8 for a minim, 3 for a dotted quaver and 1 for a semiquaver. That makes all of these notes correctly timed relative to the crochet. To get the **absolute** timing, I have multiplied D% by 8 in line 40 of Figure 3.11(b). This figure of 8 is easily changed — use a smaller value if you want the tune to rip along faster, a larger value if you want it slower. This is much easier than altering the timing numbers in each DATA line, and it's a technique that you should try to use as far as possible. There is quite enough work as it is in writing all these DATA lines! In this example, also, I have avoided the need to count the notes by using a WHILE. . .WEND type of loop. The loop is arranged to detect a value of zero for J%, so we have to start by arranging a value which is not zero, 1 in this example. The loop will be read until it finds an item of data equal to zero in the J% position. This is satisfactory providing that you don't have a zero anywhere used as a silence mark. If your melody contains rests, it's safer to use a value of -1 for the duration number as the terminator for the loop.

The loop is set up, then, with both tone period and duration numbers being read, and used in the SOUND statement in line 40. The silence line, line 50, can be kept constant this time, because we don't need to combine notes to get the timing right in this example. The hard work now is to write the DATA lines. This is always tough going, because it means looking up tables and entering values, and it's more so this time because there are two numbers for each note. It's a drag, but it's not particularly difficult, and when you try it, the effect is quite rewarding. The bonus here is that you can keep the part of the program that plays the music, and write a new tune simply by using different DATA lines.



## Formula Music

The worst part of converting from written music to Amstrad tone periods is having to look up each number in the manual. If you are struggling to learn how to read music, you might not agree with this, but in fact, it's in the looking up that most mistakes are made. Now as it happens, it's not strictly necessary to look up the number for each note. The frequency of each note in the musical scale can be calculated, and the calculated result is so close to the correct value that only a very well-trained musical ear would ever detect the difference. The formula is given in the Amstrad manual, but in my edition it has been printed incorrectly with  $(1\emptyset-N)$  in place of  $(N-1\emptyset)$ . Now you would probably think that using a formula of the kind that is shown in the manual is even worse than looking up the values, but help is at hand. You see, if the frequency and the tone period numbers can be calculated, then the computer can calculate them for us. It would make the process one step easier if you could enter the names of the notes, and have the machine generate a set of tone period numbers for you. Because the notes names use only the letters A to G, we need to be able to specify which octave we want to use. The Amstrad manual numbers the octaves with  $\emptyset$  used for the octave that starts with Middle C, and  $-1$  for the octave below,  $+1$  for the octave above. If we want to keep our notefinder program simple, we will need a rather different numbering system, starting with 1 for the lowest octave (useless if you are using only the built-in loudspeaker), 4 for the octave that contains Middle C, and 8 for the highest octave. We could then enter a string of notes such as: O3BO4CDEF#B and have each note in the string converted to the correct tone period number. This, in fact, is very similar to the method of programming music that some other types of computers, notably the MSX computers, use. The program will have to read each letter in the string, and decide whether it is an octave command or a note. If the letter is 'O', then the number that follows it will have to be read and converted into the correct form for the formula. If the letter is a note letter, then we find from a table a number which acts as the code number for the formula. The numbering system uses 1 for

C, 2 for C#, 3 for D and so on, starting with the note C in each octave. If we assign the numbers for the letters, we can then test by reading the next character to see if it is a sharp or a flat. If it's a sharp, we add 1 to the code number, if the character is a flat, then we subtract 1. Once we have the octave number and the note number, the formula can be used to find the tone period. We can then sound the note and print the number. With the numbers on the screen, you can then use the editing system to type a line number, the word DATA, and then copy down a line of numbers. The space between them allows room for duration numbers and anything else you want to put in. You can then save the DATA lines, and write the rest of your music program. Easy!

The program is shown in Figure 3.12. It starts by clearing the screen, assigning the number 125000 to variable CK, and setting a zone of ten characters wide for printing on the screen. The figure of 125000 is needed for the conversion of frequency values to tone periods, and the use of a ZONE of ten characters makes it easy to print four columns of numbers. This allows you to have four tone period numbers in each DATA line. The lines up to line 80 then print the title and remind you what the program does, and how the music strings are entered. The length of a string has been limited to 80, because otherwise, there will be too many numbers, leaving you no room to edit in DATA lines. The string is input in line 90, and its length measured, with the length test carried out in line 100.

The program from 110 onwards then clears the screen, analyses the string of notes, and prints the results. Line 120 sets the variable V% at 0 so that if you forget to start with an octave setting, the Middle-C octave will be set by default. The main loop then starts in line 130, picking characters out of the string one

```
10 CLS:CK=125000:ZONE 10
20 PRINT TAB(15)"NOTEFINDER"
30 LOCATE 1,3
40 PRINT "Please type music string"
50 PRINT"Use 0 for Octave (1 to 8).":PRI
NT"Midle C starts Octave 4."
```

```

60 PRINT"A to G for notes. Use # for sharp"
70 PRINT" and ! for flat. Example:"
80 PRINT"01BCD#CGA"
90 INPUT Music$:L%=LEN(Music$)
100 IF L%>80 THEN PRINT"Too long- only 80
0 will be printed":L%=80
110 CLS:PRINT TAB(15)"Your notes..."
120 LOCATE 1,2:V%=4
130 FOR N%=1 TO L%:P%=MID$(Music$,N%,1)
140 IF P%="0" THEN GOSUB 1000:GOTO 180
150 IF ASC(P%)>64 AND ASC(P%)<72 THEN GOSUB
2000:GOTO 180
160 PRINT"Error in music string- please
check"
170 PRINT Music$:END
180 NEXT
190 END
1000 N%=N%+1:P%=MID$(Music$,N%,1)
1010 V%=VAL(P%):IF V%<1 OR V%>8 THEN GOT
0 160
1020 V%=V%-4
1030 RETURN
2000 IF P%="C" THEN X%=1
2010 IF P%="D" THEN X%=3
2020 IF P%="E" THEN X%=5
2030 IF P%="F" THEN X%=6
2040 IF P%="G" THEN X%=8
2050 IF P%="A" THEN X%=10
2060 IF P%="B" THEN X%=12
2070 K%=MID$(Music$,N%+1,1)
2080 IF K%="#" THEN X%=X%+1:N%=N%+1
2090 IF K%="!" THEN X%=X%-1:N%=N%-1
2100 GOSUB 3000
2110 RETURN
3000 F=440*(2^(V%+(X%-10)/12))
3010 G%=ROUND(CK/F)
3020 SOUND 1,G%,10
3030 PRINT G%,
3035 FOR J=1 TO 400:NEXT
3040 RETURN

```

Figure 3.12 Converting note names into sound and CPC464 tone period numbers. This program can be used to make the task of conversion much easier.

by one, using N% as a counter. Each character is assigned to variable P\$, and line 140 tests to find if this is 'O' for Octave. If it is, then the program shifts to a subroutine which will set the octave number. The next line, 150, runs only if no octave letter has been found. This tests for a letter being A to G, the correct range of note letters. If this test succeeds, then another subroutine is called to deal with note numbers. If neither of these tests succeeds, then the character must be an illegal one, and lines 160, 170 deal with it by printing an error message and the string. If there is a small error in your string, you can edit it out and use the direct command GOTO 110 to try again.

The meat of the program is, as usual, in the subroutines. At line 1000, the counter is incremented and another character extracted. This is because this routine has been called to deal with an octave number, so the character that follows the 'O' must be the number. This is converted to number form in line 1010 and tested, with a jump to the error report if the value is unacceptable. Line 120 then converts from the 1-to-8 range which we need to use, into the -3 to +4 range that the formula needs. The subroutine which starts at 2000 then deals with the note letters. Lines 2000 to 2060 assign numbers according to the letter that is found, and lines 2070 to 2090 check for the next character being a sharp (#) or flat(!) sign. Since computers do not have the musical flat sign the exclamation mark (also called pling or shriek) has been used. The effect of either of these characters is to modify X%, the note number. The routine then ends by calling up the final subroutine, which prints and sounds the note, and returns.

In line 3000, the formula is used to calculate the frequency of the note, using the octave number and the note number. This formula is based on the agreement that the note 'A' in the Middle octave has a frequency of 400 Hz. In an orchestra, it is the job of the oboe player to tune up, checking with a tuning fork that the instrument will produce this frequency of 'A', and then the other instruments tune from the oboe's 'A'. The formula, which has been in use for rather longer than computers have been around, uses a power of two, another curious similarity between

computing and music. Once the frequency number has been found from this, the tone period can be found by dividing this number into 125000, and this is why the variable CK was assigned early on. The result, G%, in line 3010, is the correct tone pitch number. This number is an integer, and it is not exact, but since the sound generator will not accept fractions, it's as close as we can get. Note that you must not use an integer for F, otherwise a lot of the notes, particularly the low notes, will be seriously out of pitch. Line 3020 then sounds the note, using a fairly short duration, and line 3030 prints the tone period value. This is followed by a comma to force the printing into columns which have been preset by the use of ZONE. Line 3035 then puts in a time delay which is intended to counteract the effect of the sound queue. With the time delay, each number pops onto the screen just as the note sounds, so that you have a chance to check the note against the number. Later, we'll look at another way of performing this synchronisation action. Though this doesn't let you do any more than check the pitch, it's a good guide as to whether you have the right tune or not! You can also, of course, alter the program so that you can use it as a music subroutine, allowing you to specify tunes by using the note names. Help yourself!



# 4

## Harmony and Stereo

Harmony in music means sounding suitable notes together, and a set of such notes is called a **chord**. What are 'suitable' notes? Once again, there is no set answer to this — the suitable notes are the ones that sound good to your ear when they are sounded together. Though the history of music has been long, composers were still inventing 'new' harmonies right into this century. There are a few guidelines, though. Suppose we want a cheerful harmony, the type that is called a **major chord**. Once again, the word 'major' is being used to mean cheerful and bright, and as you might expect, the notes of a major chord are picked from a major scale. The keynote of any major scale will be in harmony with the **third** and the **fifth** notes of the scale. If you want two-note harmony, you can pick either, and the program in Figure 4.1 illustrates a few of these major chords. In this program, each note in a C-major scale is used as a key. The note is played, then the first chord, the second chord and finally the three-note chord.

Looking through this program gives you some idea of how the chords are produced. If you go back to the idea of numbering the notes of a chromatic scale, illustrated in Figure 4.2, then the notes that you want for a chord are numbered 1, 5 and 8 where note 1 is the one that you select as the keynote. For example, if you select the note F as your keynote, you call this note 1, and you count up to note 5, which is A. Note that this is the **third** note of a scale that starts at F. You then count up to note 8, which is C, the **fifth** note of the scale. The notes F, A and C' (the high

```

10 D%=50
20 FOR N%=1 TO 7
30 READ S$,A%,B%,C%
40 PRINT S$;" Major chords."
50 SOUND 1,A%,D%
60 GOSUB 1000:REM DELAY
70 SOUND 1,A%,D%:SOUND 2,B%,D%
80 GOSUB 1000
90 SOUND 1,A%,D%:SOUND 4,C%,D%
100 GOSUB 1000
110 SOUND 1,A%,D%:SOUND 2,B%,D%:SOUND 4,
C%,D%
120 GOSUB 1000
130 PRINT
140 NEXT
150 END
160 DATA C,478,379,319
170 DATA D,426,338,284
180 DATA E,379,301,253
190 DATA F,358,284,239
200 DATA G,319,253,213
210 DATA A,284,225,190
220 DATA B,253,201,169
1000 START=TIME
1010 WHILE TIME < START + 300
1020 WEND
1030 RETURN

```

Figure 4.1 Examples of Major chords, using the first, third and fifth notes of each major scale.



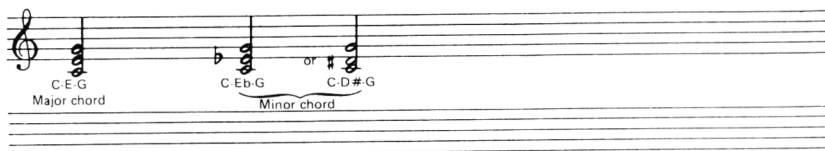
Figure 4.2 The chromatic scale numbered, starting at Middle C.

C above F) are then the three notes for a major chord. You can sound two-note chords with F and A or F and C, or the complete three-note chord with F, A and C together. This is what the program does for each of seven notes. The name of the keynote is read from a DATA line, along with numbers for the keynote and its two chord notes. The keynote name is printed, then the



keynote is sounded, using Channel 1. After a time delay, the keynote and its 'third' (the third note of the major scale in which the keynote is number 1) are sounded together. This gives one harmony. After another delay, the keynote and its fifth are sounded together, which gives a rather different sounding harmony. Another delay, and then all three notes are sounded together. This is then repeated for all the notes of the C-major scale, so that you can hear what these chords sound like for each keynote.

Once again, not all music wants these particular happy contented notes, and for any major chord of three notes, we can



**Figure 4.3 Major and minor chord of C. The flattened note can be written as E-flat or as D-sharp.**

```

10 D%=50
20 PRINT"MAJOR CHORD"
30 SOUND 1,478,D%:SOUND 2,379,D%:SOUND 4
,319,D%
40 GOSUB 1000
50 PRINT"MINOR CHORD"
60 SOUND 1,478,D%:SOUND 2,402,D%:SOUND 4
,319,D%
70 GOSUB 1000
80 GOTO 20
1000 START=TIME
1010 WHILE TIME < START + 300
1020 WEND
1030 RETURN

```

**Figure 4.4 The program which demonstrates these chords.**

make small changes which will create quite different effects. You can change a major chord into a minor chord by flattening the third. In plain language, that means reducing by one semitone the pitch of the note which is number three in the scale, the middle note of the chord. In musical notation, Figure 4.3 shows

what this amounts to, and the short program of Figure 4.4 lets you hear the effect. This simply plays the major and then the minor chord of C, and repeats until you press the ESC key. You'll hear how very different the minor chord sounds, sad and depressed compared to the cheerful optimism of the major chord.

Now try another type of change. This time, the major chord is played, followed by an augmented chord. The augmented chord is one in which the fifth is sharpened by one semitone. The effect is sounded in Figure 4.5, and you can hear that the effect is to make the chord into one which carries a sense of something about to happen, and also with even more of a feeling of optimism than the ordinary major chord. Figure 4.6 shows how this looks

```

10 D%=50
20 PRINT"MAJOR CHORD"
30 SOUND 1,478,D%:SOUND 2,379,D%:SOUND 4
,319,D%
40 GOSUB 1000
50 PRINT"AUGMENTED CHORD"
60 SOUND 1,478,D%:SOUND 2,379,D%:SOUND 4
,301,D%
70 GOSUB 1000
80 GOTO 20
1000 START=TIME
1010 WHILE TIME < START + 300
1020 WEND
1030 RETURN

```

Figure 4.5 The sound of an augmented chord.

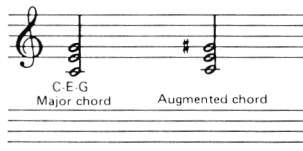


Figure 4.6 How the augmented chord looks on paper.

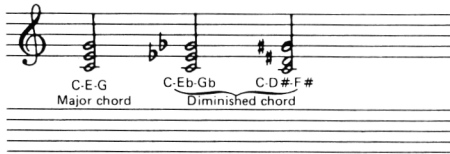
in written music. In use, you always expect to hear another chord follow an augmented chord like this, and you **never** end a tune on such a chord. Figure 4.7 illustrates another tinkering with the chord, this time lowering both the third and the fifth by a semitone

```

10 D%=50
20 PRINT"MAJOR CHORD"
30 SOUND 1,478,D%:SOUND 2,379,D%:SOUND 4
,319,D%
40 GOSUB 1000
50 PRINT"DIMINISHED CHORD"
60 SOUND 1,478,D%:SOUND 2,402,D%:SOUND 4
,338,D%
70 GOSUB 1000
80 GOTO 20
1000 START=TIME
1010 WHILE TIME < START + 300
1020 WEND
1030 RETURN

```

**Figure 4.7** The sound of a diminished chord.



**Figure 4.8** The diminished chord written down.

each. Again, this makes the bright chord into something quite different, depressed, and waiting for something unpleasant to happen. Figure 4.8 shows the notes in standard musical notation.

All of these harmonies have made use of the three channels, obtained by using the channel numbers 1, 2 and 4. The next step is to show how simple melodies with harmony can be put into computer form by making use of these channels. For a moment, we'll stick to comparatively simple methods, ignoring awkward problems of the sound queue and synchronising channels. The first point is how to get hold of music, assuming that you will not be writing your own. Piano music, or music for the various types of small electronic organs, is the best to work with. Don't on any account use music for instruments like the clarinet or trumpet, because these are 'transposing instruments' (see Appendix B). What that means is that the sounds you get don't

correspond to the written notes. For example, when the written music shows C, a B-flat clarinet will give you B-flat (yes, that's why it's called a B-flat clarinet!). This is deliberately arranged as a way of making life easy for orchestral composers, but it makes it very awkward for anyone who wants to know what a score for one of these instruments sounds like. If you keep to piano and organ music, you'll have none of these problems at least. If you compose your own music, your problems will be quite different! Whatever you do, be careful about copyright. Unlike programmers, composers have the full weight of the Law behind them, and a very effective organisation called the Performing Rights Society. The aim of both is to ensure that the composer gets a royalty each and every time his/her music is played in public. This means every time it's played in public on a radio, from a disc or tape — or from a computer. Places where music is played, including pubs, restaurants, clubs, shops, cinemas, all have to take out licences and pay money for the music that they play. Not all do, but they pay a lot more if they are caught out! If you are going to use music in a program which you will then sell or perform in public, then you must either take out a licence with the Performing Rights Society, or make sure that none of the music that you use is in copyright — which means composing your own or sticking to music that was written more than a hundred years ago. I just hope that musicians are as fussy about observing the copyright of any computer programs that **they** use! If you are going to put music into a program of your own for your own use, there is nothing to hinder you buying the sheet music for the latest hit and using it.

With that warning, let's take a look at a tune that uses harmony in a simple way. Figure 4.9 shows the score for a slow-moving piece of music — the original is marked 'Cantabile' which means with a singing tone, not in sharp bursts like a staccato piece. This uses minims for practically all of the notes except the last ones, which are semibreves. As before, we can fake the semibreves by playing two minims together, but since the loop is not a FOR...NEXT loop, we can't, in this example, run the two com-

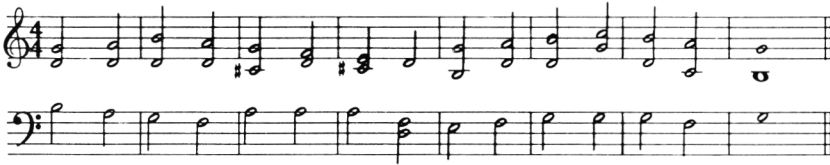


Figure 4.9 A harmony example. The original includes repeats and some crochets, but this has been adapted for simplicity.

```

10 A%=1:D%=70:N=650
20 WHILE A%>0
30 READ A%,B%,C%
40 SOUND 1,A%,D%:SOUND 2,B%,D%:SOUND 4,C
%,D%
50 FOR J=1 TO N:NEXT
60 WEND
70 DATA 426,319,506
80 DATA 426,284,568
90 DATA 426,253,638
100 DATA 426,284,676
110 DATA 451,319,568
120 DATA 426,338,568
130 DATA 451,379,568
140 DATA 676,426,851
150 DATA 506,319,758
160 DATA 426,284,676
170 DATA 426,253,638
180 DATA 319,239,638
190 DATA 426,253,638
200 DATA 478,284,676
210 DATA 506,319,638
220 DATA 506,319,638
230 DATA 0,0,0

```

Figure 4.10 The program for the music for Figure 4.9.

pletely together. The program corresponding to this is in Figure 4.10. It uses the type of reading instructions that you should be familiar with by now, with a WHILE. . .WEND loop used to read data until the first of a set is a zero. The DATA lines have been taken direct from the written music, using the table of notes in the Amstrad manual. The times of the notes have been set with D% equal to 70 to make the music slow, and the delay loop uses a value of N=650. This last value has been evolved by trial and

error. When you are trying out a tune like this, it's always a good idea to use a delay loop that is much too long, such as you get with  $N=1000$  or even more. That way, when the tune plays, you will hear each chord separately, which makes it a lot easier to hear if one of them sounds a bit off. In this example, I did just that, and then reduced the value of the counter  $N$  until the chords were **just** beginning to run into each other. This is what produces a reasonable 'singing' note. If you alter the value of the note length,  $D\%$ , then you will also have to alter the value of the loop count  $N$ , so it's advisable to get the note length right first.

Now it's not exactly like the latest rock hit, but we have to move slowly when we are learning the ropes. It does at least play something, and that's what we want it to do. The next thing we can check out is whether we can improve on this. As it stands, it's a bit drab and featureless, just a set of chords. Suppose, for starters, that we emphasised the melody a bit. The melody is in channel 2, the  $B\%$  numbers, and we could emphasise it by making it slightly louder. Now for the simple use of the SOUND instruction, you can specify volume by adding another number following the duration number. The permitted range is  $0$  to  $7$  — later we'll look at the circumstances in which this can be extended to  $15$ . If you don't specify any volume number, then you get the volume that corresponds to level  $4$ . Suppose that we change the volume in the  $B\%$  numbers to  $6$ . We can do this by altering line  $40$  so that it reads:

```
40 SOUND 1,A%,D%:SOUND 2,B%,D%,6:SOUND
4,C%,D%
```

— with the change of volume only in the middle channel, the one which carries the melody. If you play this one now, you'll find that the melody sounds much clearer, and the effect is better. You can make another improvement by jacking up the volume of the bass notes, but you won't notice so much change unless you are using a Hi-Fi system, or sensitive earphones. A volume figure of  $5$  in the  $C\%$  SOUND instruction will boost the bass nicely, enough to be noticeable on these systems, but not if you are using just the built-in loudspeaker. If you are using headphones, incidentally, you'll find that the notes are separated, with

the bass appearing mainly in the right-hand earpiece. We'll be looking at stereo sound in more detail later but for the moment, it's worth noting that if you are likely to be writing music for stereo output, you should always try to put the melody into the middle, 'B', channel rather than in the A or C channels, and that's why it has been done in this example.

We can go rather further in making this sound better, though. One of the things that still makes it seem a bit featureless is that the notes all carry pretty much the same weight, there's nothing that conveys a rhythm. Now this can be remedied if we emphasise the first note in each bar which, in this case, means the first in each two notes. We can emphasise this note either by making it fractionally longer or by making it louder. Suppose we try making it louder. If we numbered the notes 1, 2, 3 and so on, the loud ones would be the odd numbered ones. We can number the notes by returning to using a FOR. . .NEXT loop for reading the data, and testing the number which is the loop counter. If this number divides evenly by 2 we can play the note softly, because it will be an even numbered note. If the number does not divide evenly by two, then the number is odd, it's the first note in a bar, and we make it loud!

This method of putting a bit of rhythm into a tune is illustrated in Figure 4.11. The FOR. . .NEXT loop is arranged to read in all the notes from the DATA lines, and also to produce the emphasis. The method that is used may be new to you, however. MOD is a word that does not appear in the list of BASIC keywords in the manual, because it is one of the number operators, like +, -, \*, and /. The effect of MOD is to produce the remainder of an integer division, and the MOD is placed between the numbers just as you would place a division sign, but with the important difference that there **must be a space between MOD and each number**. For example, if you write  $X \% MOD\ 3$ , then this gives 1 if  $X \% = 1$ , 2 if  $X \% = 2$ , and 0 if  $X \% = 3$ . It will then give 1 again if  $X \% = 4$ , 2 again if  $X \% = 5$ , 0 again if  $X \% = 6$  and so on. These numbers are the remainders after the value of  $X \%$  has been divided by three. In our example, we want to detect the odd notes, and this means that there will be a remainder when we

```

10 D%=70:N=650
20 FOR J%=1 TO 16
30 IF (J% MOD 2)=0 THEN V%=5 ELSE V%=7
40 READ A%,B%,C%
50 SOUND 1,A%,D%:SOUND 2,B%,D%,V%:SOUND
4,C%,D%,5
60 IF J%<16 THEN FOR J=1 TO N:NEXT
70 NEXT
80 DATA 426,319,506
90 DATA 426,284,568
100 DATA 426,253,638
110 DATA 426,284,676
120 DATA 451,319,568
130 DATA 426,338,568
140 DATA 451,379,568
150 DATA 676,426,851
160 DATA 506,319,758
170 DATA 426,284,676
180 DATA 426,253,638
190 DATA 319,239,638
200 DATA 426,253,638
210 DATA 478,284,676
220 DATA 506,319,638
230 DATA 506,319,638

```

**Figure 4.11 Putting rhythm into a tune by emphasising the first note in each bar.**

divide the counter number  $J\%$  by 2. If we test for  $(J\% \text{ MOD } 2)=0$ , then this will be true when  $J\%$  is even, false when  $J\%$  is odd. We could, of course, just as easily test for  $(J\% \text{ MOD } 2)=1$  which would be true when  $J\%$  was odd. In the program of Figure 4.11, this test has been made in line 30. If  $J\%$  is odd, then the volume number  $V\%$  is 5, but if  $J\%$  is even, the volume number 7 is used. This gives a good emphasis to the first note in each bar. The test in line 60 also allows us to combine the last two minims into a complete semibreve, which sounds more appropriate for the end of the phrase. Finally, Figure 4.12 shows the other method. The same test has been used, but this time, the first note in each bar has been made slightly longer. This, of course, makes a bit of a mess of the composer's timing, and it's not something that you would use unless you thought that the effect warranted it. The technical name for this sort of time distortion is 'rubato', meaning literally 'robbery'. You have robbed the second note



```

10 D%=70:N=550:V%=5
20 FOR J%=1 TO 16
30 IF (J% MOD 2)=0 THEN D%=50 ELSE D%=70
40 READ A%,B%,C%
50 SOUND 1,A%,D%:SOUND 2,B%,D%,V%:SOUND
4,C%,D%,5
60 IF J%<16 THEN FOR J=1 TO N:NEXT
70 NEXT
80 DATA 426,319,506
90 DATA 426,284,568
100 DATA 426,253,638
110 DATA 426,284,676
120 DATA 451,319,568
130 DATA 426,338,568
140 DATA 451,379,568
150 DATA 676,426,851
160 DATA 506,319,758
170 DATA 426,284,676
180 DATA 426,253,638
190 DATA 319,239,638
200 DATA 426,253,638
210 DATA 478,284,676
220 DATA 506,319,638
230 DATA 506,319,638

```

Figure 4.12 The other method of emphasising a note by making it slightly longer.

in each bar of some of its time in order to extend the first. The delay loop has been slightly altered to accommodate this change of timing.

## Synchronisation

Suppose we have a piece of music which looks something like the example in Figure 4.13. The treble melody line uses crochets and quavers, but the bass line uses minims and quavers. In each bar, a minim in the bass plays for the same time as two

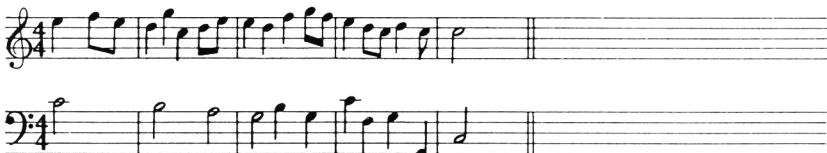


Figure 4.13 A slow-moving melody which uses a mixture of note times.

of the crochets (or a crochet and two quavers) in the treble, and the important point is that these notes must be synchronised. Now for a **very short** tune, this does not need any special effort on your part. You can use a slight modification of the methods that we have used so far, with a channel number included as part of the DATA. If you read for each note the channel number, tone period number, and a duration number, then you can place the data lines in order. The order is the order of playing, remembering that the data will be read from the sound queue into the channel each time a channel is free. Take a look, for example, at the first bar of this tune which uses a crochet and two quavers in the treble, and a single minim in the bass. The most straightforward way of programming this is to put in the data for the minim, using channel A, and then three data lines for the treble, the crochet and the two quavers. Providing that the timing is correct, this should sound completely acceptable. The data for the minim is read, goes at once into the head of the sound queue for Channel A, and the data for the three treble notes is also read, this time into the sound queue for Channel B with the data for the crochet going at once to the head of the queue. There is very little difference in the time between starting the minim and starting the crochet, so the two appear to make a good chord. Whenever the crochet finishes, the next quaver is taken from its queue, and when this quaver is finished, the second quaver is put from its sound queue into its channel.

For a few notes, even for a few bars, this is reasonably satisfactory, but after some time, the synchronisation falls apart. The reason is that taking data from the sound queue into the sound channels is not absolutely instantaneous. There is always a slight delay caused by this transfer, and when you feed in a lot of notes in sequence like this, these delays build up. The effect of this cumulative delay is to make the treble and the bass out of step after several bars. When a tune consists of a stream of notes with no rests, and when you can use an organ-note effect with no separation between notes, the lack of synchronisation is not really important, because you won't notice it unless you are working with really long pieces. Try, for example, the tune in Figure 4.14, which is the Amstrad version of the piece of music

```

10 L%=40:C%=1
20 WHILE C%<>0
30 READ C%,N%,D%
40 IF C%=2 THEN V%=7 ELSE V%=5
50 SOUND C%,N%,D%*L%,V%
60 WEND
70 DATA 1,478,4
80 DATA 2,190,2
90 DATA 2,179,1
100 DATA 2,190,1
110 DATA 1,506,4
120 DATA 2,213,2
130 DATA 2,159,2
140 DATA 1,568,4
150 DATA 2,239,2
160 DATA 2,213,1
170 DATA 2,190,1
180 DATA 1,638,4
190 DATA 2,190,2
200 DATA 2,213,2
210 DATA 1,506,2
220 DATA 2,179,2
230 DATA 1,638,2
240 DATA 2,159,1
250 DATA 2,179,1
260 DATA 1,478,2
270 DATA 2,190,2
280 DATA 1,716,2
290 DATA 2,213,1
300 DATA 2,239,1
310 DATA 1,638,2
320 DATA 2,213,3
330 DATA 1,1276,2
340 DATA 2,239,1
350 DATA 1,956,4
360 DATA 2,239,4
370 DATA 0,0,0

```

**Figure 4.14** The melody programmed for the CPC464. If you can try this with output from a good stereo system, it sounds quite impressive.

in Figure 4.13. This has been programmed with data lines which contain the channel number, the tone period, and the duration. The SOUND instruction then uses these values, with a multiplier L% used at the start of the program to set the timing. This has

been set so as to give about the right pace. The original music (by Handel) is marked 'Andante', which means 'at a walking pace'. If you imagine your legs moving in time to the beat of the music, you should find that the tempo is, as it suggests, a walking pace. The melody line is emphasised by making this channel (Channel B, code 2) play louder. This has been done by testing at the reading stage, and if a '2' has been read for the channel number, then the volume number V% is set to 7, otherwise to 5. When each note is put into a data line like this, control of volume becomes easier. You could, if you liked, even control the volume of each note separately by having a V% entry into each data line, but it's unusual to have to change the volume so often. A more useful method is to use a FOR. . .NEXT loop to read in the data, and to change the volume at various places by checking the value of the counter variable.

This little tune, then, plays acceptably with no problems of synchronisation, provided that we accept its limitations of no spaces between notes. When we want to make the notes of the melody sound separated, however, we are likely to run into trouble. If we simply use a delay loop after each melody note, then both melody and accompaniment seem to suffer from hiccups, and the effect is not pleasing. A neater method is to detect a note which uses C%=2, and to insert a SOUND instruction which plays a short space. This can be done by adding a line 55 which reads:

```
55 IF C% =2 THEN SOUND 2,0,5,0
```

which will give a short space, only two units, between notes of the melody. The normal crochet space is 80 (with L%=40 and D%=2), and a gap of 5 units doesn't seem too drastic. When you play this one, the notes of the melody are nicely separated, but the accompaniment is badly out of synchronisation. This shows itself after a few notes, when you can hear the bass note change at the wrong time, and you find that bass and treble are not in step at the start of any bar.

Getting out of this one involves using the synchronisation codes of the Amstrad SOUND instruction. Try it out on your ears first,

```

10 L%=40:C%=1:J=600
20 WHILE C%<>0
30 READ C%,N%,D%
40 IF C%=2 THEN V%=7 ELSE V%=5
50 SOUND C%,N%,D%*L%,V%
55 IF C%=2 THEN SOUND 2,0,5,0
60 WEND
70 DATA 17,478,4
80 DATA 10,190,2
90 DATA 2,179,1
100 DATA 2,190,1
110 DATA 17,506,4
120 DATA 10,213,2
130 DATA 2,159,2
140 DATA 1,568,4
150 DATA 2,239,2
160 DATA 2,213,1
170 DATA 2,190,1
180 DATA 17,638,4
190 DATA 10,190,2
200 DATA 2,213,2
210 DATA 1,506,2
220 DATA 2,179,2
230 DATA 1,638,2
240 DATA 2,159,1
250 DATA 2,179,1
260 DATA 17,478,2
270 DATA 10,190,2
280 DATA 1,716,2
290 DATA 2,213,1
300 DATA 2,239,1
310 DATA 1,638,2
320 DATA 2,213,3
330 DATA 1,1276,2
340 DATA 2,239,1
350 DATA 17,956,4
360 DATA 10,239,4
370 DATA 0,0,0

```

**Figure 4.15** The new version of the melody, with spaces and with synchronisation added.

by playing the program in Figure 4.15. This is very much the same as Figure 4.14, but with some cunning alterations to the

Number	Effect
8	Synchronise with note in Channel A
16	Synchronise with note in Channel B
32	Synchronise with note in Channel C
In addition, we have the following —	
64	Hold note until RELEASE
128	Remove all notes from queue.

**Figure 4.16 The essential synchronisation numbers.**

DATA lines. You'll see that some of the channel numbers have been changed, and when you play the program, your ear will tell you that the synchronisation is much better. What the changes have done is to force the channels to synchronise at the start of each bar. This still allows the notes to drift apart slightly at the end of the bar, but the difference is small and acceptable. Provided the start of each bar is synchronised, then the effect is a great improvement on the version of Figure 4.14 which included the 'silence' line to separate the melody notes.

## How to synchronise

Getting sounds to synchronise, or 'rendezvous' as the manual puts it, is done by adding numbers to the channel numbers. The numbers which can be added are shown in Figure 4.16. These numbers must be added to the channel number of **each channel that is to be synchronised**, not just to one channel. In our tune, we want the first note in Channel B to synchronise with the first note in Channel A. The first DATA line contains the numbers for the minim that is the first note in channel A, code number 1. To make this synchronise with Channel B, the melody channel, we must add 16 to the channel code number, making it 17. This will have no effect unless we also mark the note in Channel B which is to sound at the same time. To do this, we add 8 to the Channel B code of 2, making 10. If we had added 8 to another note in Channel B, then the synchronisation would have occurred on that note instead of the first one. The program has then been altered by picking out the lines that contain the first notes of each bar in the two channels. These have their C% numbers

altered, making each 1 into 17 and each 2 into 10. This ensures that the channels are synchronised at the start of each bar. We could, if we liked, synchronise other notes but, as I have said, it's quite reasonable to synchronise just the starting notes like this. The effect is certainly a lot better than you get normally **when** there is a set of short silences in the melody line. I must emphasise that for short pieces, played in organ style with no silences, you don't have to worry about synchronisation. Where you want to use separated notes in the melody, or where there is a complicated set of silences in a tune, and particularly if the tune is a long one, then you **do** have to worry about synchronisation. If you are using three channels, then the third channel is synchronised in the same way, using the appropriate code numbers. The best scheme is to use Channel B as the melody channel, and to make the other two synchronise with it at the start of each bar. It may be, of course, that some bars will start with a silence in one channel. You can still write a DATA line for a silence, however, and synchronise with it. The alternative is to synchronise the other channels, and then synchronise again when a note is played after the silence.

Synchronisation really becomes essential, however, when we start to use 'envelopes' to create notes that have a pattern like that of most musical instruments. This is something that we'll be looking at in the following Chapter, but as an introduction, the use of an 'envelope' means that the volume of a note can be made to vary during the time it is sounded. This makes it possible to have notes that sound separated without the use of silences, but it also means that control of duration is less easy. In turn, this makes synchronisation very difficult unless we make use of these extended Channel numbers to ensure that notes start out in step at the start of each bar. In some cases, synchronisation may be needed more than once per bar. The effects can be very pleasing — so now you have something to look forward to!

## **Stereo Sound**

We've met the idea of stereo sound briefly, but it's now time to take a look at how the CPC464 can produce this type of sound.

It's done simply by using the channels. Channel A, code 1, feeds the left-hand socket. If you type:

```
SOUND 1,239,5000,7
```

and plug in stereo headphones, you can hear the effect. You will need to turn the volume control down as far as possible. This volume control affects only the built-in loudspeaker, **not** the headphone socket. When you press ENTER, you'll hear the sound — but in the left ear only. Now type:

```
SOUND 4,200,5000,7
```

and press ENTER, and you'll hear this other note in the right ear. This is Channel C in use, code 4. If you now type:

```
SOUND 2,300,5000,7
```

then you'll hear the sound equally in each ear, because the sound of Channel B is fed equally to both parts of the output socket. This is why, up until now, I have advocated using Channel B for the melody line. When you use the built-in loudspeaker of the CPC464 (or the TV loudspeaker when you use a TV receiver and power-pack in place of a monitor), the stereo signals are combined, and it is this combined signal that is controlled by the volume control. When you use earphones, the only control that you have over volume is by way of the volume number, 0 to 7, in the SOUND instruction. You can, of course, follow the hint in Appendix C and attach a low-power stereo amplifier to the output so that you have complete control over the earphone output.

By way of introduction to the idea, give a listen to the scrap of waltz in Figure 4.17. In this example, the melody is kept in the left stereo channel and the accompaniment in the right channel. On headphones or with a stereo amplifier, the separation is very noticeable, though there is no hint of it when you listen with the internal loudspeaker. Separation like this is entirely artificial, and you would do this kind of thing only when you particularly wanted the effect of separation. A more interesting effort is illustrated in Figure 4.18, which uses the same tune and the same main program, but with some considerable alterations to the DATA lines.





```

10 L%=20
20 FOR N%=1 TO 31
30 READ C%, T%, D%
40 IF C%=31 THEN V%=7 ELSE V%=5
50 SOUND C%, T%, L%*D%, V%
60 SOUND 1, 0, 2, 0: SOUND 4, 0, 2, 0
70 NEXT
80 DATA 1, 284, 2, 1, 239, 2, 1, 213, 2
90 DATA 33, 190, 6, 12, 568, 2, 4, 478, 2, 4, 478,
  2
100 DATA 12, 379, 6, 33, 190, 2, 1, 213, 2, 1, 239
  , 2
110 DATA 33, 213, 6, 12, 506, 2, 4, 426, 2, 4, 426
  , 2
120 DATA 12, 379, 6, 33, 253, 2, 1, 213, 2, 1, 190
  , 2
130 DATA 33, 179, 6, 12, 506, 2, 4, 426, 2, 4, 426
  , 2
140 DATA 12, 379, 6, 33, 179, 2, 1, 190, 2, 1, 213
  , 2
150 DATA 33, 239, 6, 12, 568, 2, 4, 478, 2, 4, 478
  , 2

```

Figure 4.17 (a) A scrap of waltz and a program (b) which will give the sound in stereo.

In this example, we shall put the melody line in Channel B, and split the accompaniment into channels A and C. The way this will be done is fairly typical of stereo music programs. The tune is a waltz, and each alternate bass line is the characteristic 'rum-pum-pum' of waltz-time. The first of these three notes, which is usually the lowest pitch, will be put into Channel C, and the other two in Channel A. When the bass line consists of only one note, then it will be played in both A and C. In all other respects, the program is the same as that of Figure 4.17.

The synchronisation, however, is far from straightforward. No synchronisation is needed in the first DATA line, because the

```

10 L%=20
20 FOR N%=1 TO 38
30 READ C%,T%,D%
40 IF C%=31 THEN V%=7 ELSE V%=5
50 SOUND C%,T%,L%*D%,V%
60 SOUND 1,0,2,0:SOUND 4,0,2,0:SOUND 2,0
,2,0
70 NEXT
100 DATA 2,284,2,2,239,2,2,213,2
110 DATA 34,190,6,20,568,2,12,0,4,33,478
,2,1,478,2
120 DATA 17,379,6,20,379,6,42,190,2,2,21
3,2,2,239,2
130 DATA 34,213,6,20,506,2,12,0,4,33,426
,2,1,426,2
140 DATA 17,379,6,20,379,6,42,253,2,2,21
3,2,2,190,2
150 DATA 34,179,6,20,506,2,12,0,4,33,426
,2,1,426,2
160 DATA 17,379,6,20,379,6,42,179,2,2,19
0,2,2,213,2
170 DATA 34,239,6,20,568,2,12,0,4,33,478
,2,1,478,2

```

**Figure 4.18 Using further stereo effects to emphasise the waltz time.**

notes of the first bar are unaccompanied. The next DATA line is slightly more complicated. The melody note, a dotted minim, is to be in Channel B (code 2), and must synchronise with the first accompanying crochet, which will be in Channel C. Since the synchronisation number for C is 32, this adds up to 34 for this channel number. The next note in the DATA line is the first crochet of the accompaniment. This is in Channel C (code 4), and must synchronise with the melody in Channel B (sync. code 16). Adding, we get the code of 20 to use as channel number for this note.

Now things get more complicated. The next two notes of the accompaniment are to be played in Channel A. If we put them in with no synchronisation, they will be played at the same time as the other notes. What can we synchronise them with, when no other note starts at the same time? The answer is to 'play' a silent note in Channel C for two beats, and synchronise to this.

Channel C (code 4) synchronised to A (code 8) gives 12, and we need a tone period of 0 to get silence, with a length number of 4 for two beats. We can now put the second note of the accompaniment into Channel A, and synchronise to this 'dummy' note in C. Channel A (code 1) synchronised to C (code 32) gives 33, which is therefore the channel number for this note, and it is then repeated, this time with the Channel number code of 1 because the last note needs no synchronisation.

Now take a look at DATA line 120, which contains the next lot of surprises. We start with the first accompaniment note in Channel A, synchronised to channel B with the number 17 (= 1 + 16), and then code 20 is used to play the bass note also in Channel C, synchronised to Channel B. What we want now is to play the first melody note in Channel B, synchronised to **both A and C**. If you try to synchronise to one or the other, the program will hang up, waiting for something to start first! The note is to be in B, code 2, with sync. to A (code 8) and to C (code 32). The sum of 2+8+32 is 42, and that's the number to use. After this hurdle has been dealt with, the other two notes in the melody line are dealt with in the normal way. The remaining lines now follow the pattern of either line 110 or line 120. The important point here is that you may have to synchronise a note to **two** other channels. If you find that a music program which uses synchronisation hangs up, then print out the values of channel and note numbers after pressing ESC twice. You will usually find a synchronisation error at a point in the data just before the place where reading stopped. It's not always easy for you to spot when this is needed, but the machine certainly can!

## Rolling your own

If you seriously intended to become a composer, then using a computer is the hardest way of going about it, and you would be better off using one of the range of Casio or Yamaha keyboards. I'm saving my pennies for one right now. If, however, you just want to knock off the odd phrase to liven up a game that you have devised, or you want to use the sound as an aid to a blind user, or you want to use some sound in a business

```

10 D%=30
20 FOR N%=22 TO 71
30 IF INKEY(N%)=0 THEN X%=N%-21:ELSE 50
40 ON X% GOSUB 80,100,110,120,140,160,18
0,200,220,240,260,280,300,320,340,360,38
0,400,420,440,460,480,500,520,540,560,57
0,590,610,630,650,670,690,710,730,750,77
0,790,810,830,850,870,890,910,930,940,96
0,970,990,1000
50 NEXT
60 IF INKEY(79)=0 THEN FOR J%=1 TO 4:PRI
NT CHR$(8);CHR$(16);:NEXT
70 CALL &BB03: GOTO 20.
80 SOUND 2,100,D%,7:PRINT"100 ";
90 RETURN
100 RETURN
110 RETURN
120 SOUND 2,119,D%,7:PRINT"119 ";
130 RETURN
140 SOUND 2,113,D%,7:PRINT"113 ";
150 RETURN
160 SOUND 2,142,D%,7:PRINT"142 ";
170 RETURN
180 SOUND 2,106,D%,7:PRINT"106 ";
190 RETURN
200 SOUND 2,134,D%,7:PRINT"134 ";
210 RETURN
220 SOUND 2,127,D%,7:PRINT"127 ";
230 RETURN
240 SOUND 2,159,D%,7:PRINT"159 ";
250 RETURN
260 SOUND 2,150,D%,7:PRINT"150 ";
270 RETURN
280 SOUND 2,190,D%,7:PRINT"190 ";
290 RETURN
300 SOUND 2,179,D%,7:PRINT"179 ";
310 RETURN
320 SOUND 2,225,D%,7:PRINT"225 ";
330 RETURN
340 SOUND 2,169,D%,7:PRINT"169 ";
350 RETURN
360 SOUND 2,213,D%,7:PRINT"213 ";
370 RETURN

```

```
380 SOUND 2,253,D%,7:PRINT"253 ";
390 RETURN
400 SOUND 2,201,D%,7:PRINT"201 ";
410 RETURN
420 SOUND 2,239,D%,7:PRINT"239 ";
430 RETURN
440 SOUND 2,301,D%,7:PRINT"301 ";
450 RETURN
460 SOUND 2,284,D%,7:PRINT"284 ";
470 RETURN
480 SOUND 2,358,D%,7:PRINT"358 ";
490 RETURN
500 SOUND 2,338,D%,7:PRINT"338 ";
510 RETURN
520 SOUND 2,268,D%,7:PRINT"268 ";
530 RETURN
540 SOUND 2,319,D%,7:PRINT"319 ";
550 RETURN
560 RETURN
570 SOUND 2,379,D%,7:PRINT"379 ";
580 RETURN
590 SOUND 2,478,D%,7:PRINT"478 ";
600 RETURN
610 SOUND 2,568,D%,7:PRINT"568 ";
620 RETURN
630 SOUND 2,451,D%,7:PRINT"451 ";
640 RETURN
650 SOUND 2,426,D%,7:PRINT"426 ";
660 RETURN
670 SOUND 2,536,D%,7:PRINT"536 ";
680 RETURN
690 SOUND 2,402,D%,7:PRINT"402 ";
700 RETURN
710 SOUND 2,506,D%,7:PRINT"506 ";
720 RETURN
730 SOUND 2,602,D%,7:PRINT"602 ";
740 RETURN
750 SOUND 2,758,D%,7:PRINT"758 ";
760 RETURN
770 SOUND 2,716,D%,7:PRINT"716 ";
780 RETURN
790 SOUND 2,902,D%,7:PRINT"902 ";
800 RETURN
```

```

810 SOUND 2,851,D%,7:PRINT"851 ";
820 RETURN
830 SOUND 2,676,D%,7:PRINT"676 ";
840 RETURN
850 SOUND 2,638,D%,7:PRINT"638 ";
860 RETURN
870 SOUND 2,804,D%,7:PRINT"804 ";
880 RETURN
890 SOUND 2,1204,D%,7:PRINT"1204";
900 RETURN
910 SOUND 2,956,D%,7:PRINT"956 ";
920 RETURN
930 END
940 SOUND 2,1135,D%,7:PRINT"1135";
950 RETURN
960 RETURN
970 SOUND 2,1073,D%,7:PRINT"1073";
980 RETURN
990 RETURN
1000 SOUND 2,1012,D%,7:PRINT"1012";
1010 RETURN
1020 RETURN

```

**Figure 4.19 A program which allows you to use the keys of the computer as a musical keyboard.**

program, then it's entirely reasonable to do your own compositions on the computer. To aid you, here are two short but useful programs. The first, Figure 4.19, turns the keys of the CPC464 into a music keyboard. It's not exactly like a piano keyboard, because the positions of the notes are rather unorthodox, but it does allow a fair range of notes to be produced, and shows the tone period number for each note on the screen unless you delete it. The keys that are used are shown in Figure 4.20, with the note that each key can sound. It would be possible to get even more notes by making use of the SHIFT and CTRL keys along with the character keys, but for the sake of (relative!) simplicity, this has not been done. There are 44 active keys, which is the same as some of the smaller electronic keyboards on the market. The duration of the note is fixed, but because the key action is contained in a loop, a key will sound for at least as long as you press it, and usually quite a lot longer. You can control this to some extent by altering the value of D% in line 10,



but if you make D% much smaller than 20, the repeat action will cause a set of beeps rather than a continuous note. Large values of D% give a note that plays for much too long after the key has been released. When the program is used, only the keys noted have any musical effect. The DELETE key will wipe out the note numbers on the screen, so that you can delete a note which you didn't intend or don't like. The ESC key will allow you to terminate the program. If you use a RETURN in place of the END in this line (line 930) then the program will be interruptable only by pressing CTRL SHIFT ESC, and the program will be wiped when this is done. You should save the program **before** testing it in case any mis-typing causes a lock-up of this kind. Once you have played a melody, wiping out any notes that you don't like, you can leave the program, type a high line number and the word DATA, and then use the copy action of the CPC464 editor to copy down the tone period numbers for your melody. You can then wipe the program lines, leaving you with one or more DATA lines which you can record for use later. Talk about instant composing!

## The program action

The program works by checking the keyboard in a loop. The INKEY(N%) action is to return -1 if key number N% is not pressed, and a value of 0 if the key is pressed. By testing each key number from 22 to 71 in a loop, each of the keys in the set shown in Figure 4.20 is tested, along with some others. When one of these keys is pressed, its keyboard number N% has 21 subtracted, so that the range of the result, X%, is 1 to 50. This number can now be used in an ON X% GOSUB line to cause one of fifty subroutines to be run. In each subroutine, there is a SOUND 2 statement which gives the correct pitch note, the duration set by D%, and volume 7. There is also a PRINT statement which will print the pitch number. The next line is the RETURN line. For the keys which are not used for notes, the subroutine line contains only a RETURN, the exception being the ESC key, which ends the program. The DELETE key is tested for in line 60, and if it has been pressed, then four backspace and delete actions are carried out. This is enough to wipe clear



any of the numbers that has been printed on the screen. Three-figure numbers are printed with a space between them, but the few four-figure numbers will be butted close to each other — this should not cause too much confusion. The CALL in line 70 clears the keyboard buffer, so that you do not see a set of garbage characters on the screen when you break the program.

## Harmoniser

The program in Figure 4.21 is a chordfinder. The principle is that you can input a tone number, and the note that corresponds to this number will be played. You can then input another tone number for a harmony. After a short delay, this too will be sounded. If you like the result, you press the ENTER key and you will be prompted for another note. If you don't like a harmony, then pressing the spacebar will allow you to try another number. After you have accepted the second harmony, the chord is timed and will end when the program ends. Note that when you enter the second harmony, you have to press ENTER to input the tone number, and then press ENTER again to show that you approve of the result. The note times have been set very long to allow you some time to make up your mind about the harmonies. The

```
10 REM CHORDFINDER
20 INPUT "Melody number ";A%
30 SOUND 1,A%,30000,6
40 INPUT"Harmony 1 ";B%
50 SOUND 130,0,0
60 SOUND 2,B%,30000
70 K$=INKEY$: IF K$=""THEN 70
80 IF K$=CHR$(32) THEN 40
90 INPUT "Harmony 2 ";C%
100 SOUND 132,0,0
110 SOUND 4,C%,30000
120 K$=INKEY$: IF K$=""THEN 120
130 IF K$=CHR$(32) THEN 90
140 START=TIME
150 WHILE TIME< START+2000:WEND
160 CLEAR
170 END
```

Figure 4.21 A chordfinder program. This allows you to make up your own chords, listen to them, and edit them!

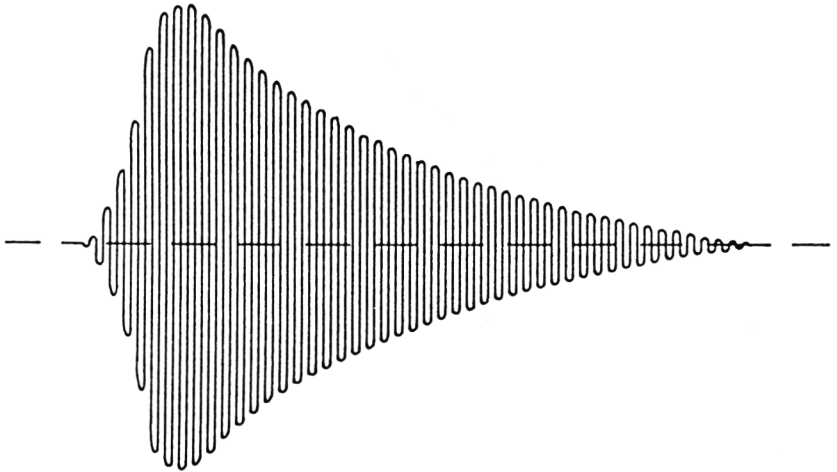
program action is straightforward, and the only point which needs some explanation is the use of CLEAR in line 160. This clears out all the sound queues, among other things, so that the sound stops. Without this statement, the sound can continue for a long after the program ends.

# 5

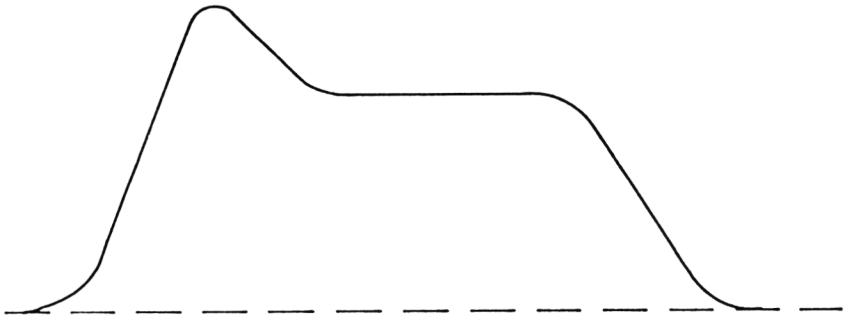
## Waveforms and Envelopes

If you cast your mind (and your ear) back to the start of this book, you'll remember that I said that two things in particular affected the **quality** of a note. By quality, I mean the peculiarity of sound that allows us to tell if a note is being played by a violin, a clarinet, a guitar, or whatever else. These notes might contain waves of the same amplitude and pitch, but the way that they sound in our ears is quite different. The two reasons, remember, are waveform and envelope. The waveform of a note from a musical instrument is normally a complicated and jagged one, not one that can be easily reproduced by the comparatively simple sound generating system that is used by most computers. The envelope is something quite different. A note from an instrument consists of a lot of sound waves, often several hundred. Now whatever way we create a note on an instrument, it doesn't start instantaneously, and it never carries on unchanged. When you start a note, the first few waves are of lower amplitude than the rest, because they form the build-up to the final amplitude. This part of a note is called the 'attack', it's the part in which the amplitude of each wave is considerably greater than the amplitude of the one that went before it.

What happens after that depends a lot on what sort of musical instrument makes the note. If the instrument is one that is struck or plucked, like drum, piano or guitar, then the note reaches its maximum amplitude just after the striking or plucking has stopped. From then on, the amplitude fades away again to zero. The shape, which we call the 'amplitude envelope' of the note,

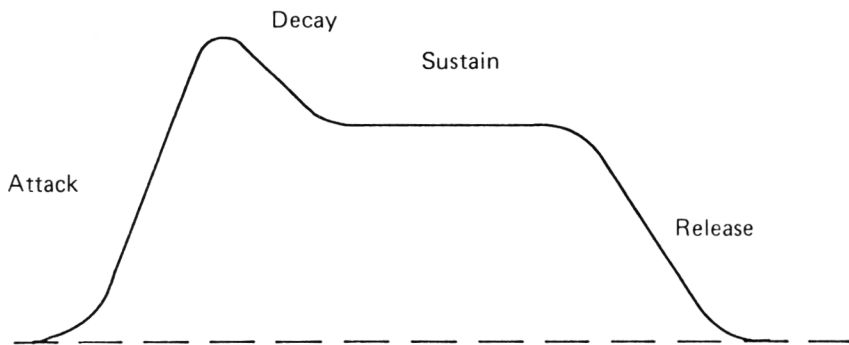


**Figure 5.1** A typical envelope shape for a plucked or struck instrument. The note starts quickly and then dies away.



**Figure 5.2** The rather different envelope shape for an instrument which is 'continually excited', as distinct from being struck.

is rather like the one in Figure 5.1. The shape is symmetrical around the time axis — in other words, if you folded it around the horizontal line that runs through the middle, the two halves would match. When we draw these envelope shapes, then, we can concentrate on one half, the top half, and we don't have to draw the waves either. Figure 5.2 shows the rather different type of envelope that you can expect when an instrument produces a note by 'continual excitation'. Unlike instruments that are plucked or struck, some instruments are bowed or blown for all the time that a note is to be sounded. This makes the envelope



**Figure 5.3 The classic Attack, Decay, Sustain, Release pattern.**

have an attack, then a section in which the amplitude decreases, the 'decay' section. When the amplitude of the note has settled down like this, it can be steady for a little while, and this part is called the 'sustain' section. Finally the player stops blowing or bowing and the note dies away — this is the 'release' section. Figure 5.3 shows the four sections for an imaginary envelope.

This type of envelope is an 'amplitude envelope', because it deals with the way that the different waves of a note change amplitude. The way some instruments are played, however, causes another type of envelope, a 'pitch envelope'. If you watch a violin player at work, you'll notice that the hand whose fingers press on the strings is shaken to and fro while a note is being played. This rolls the fingertips slightly over the strings that are being held down, and it makes the pitch of the note increase and decrease very slightly in time with the movement of the hand. This action is called 'vibrato', and it's a feature of instrumental playing that has developed over the last couple of hundred years. The aim is to make notes sound more interesting, richer, more intense. Too much vibrato has just the opposite effect, it makes the notes sound wavering and undecided. Vibrato cannot be produced so easily on other instruments, apart from the slide trombone, and the corresponding effect for these other instruments is 'tremolo'. Tremolo in wind instruments can be produced by variations in blowing, and it consists of amplitude variations rather than frequency variations.

If the computer is to be able to make a good job of producing

sounds, then, it needs to be able to work with both an amplitude envelope and a pitch envelope. Both of these effects are produced by adding more data to the SOUND command, and so we'll take a look now at what is needed to specify an envelope. To start with, your SOUND instruction **must** have numbers for channel, tone, duration and volume. The duration number **must** be zero, and the volume number **must** have some value, normally zero. If you are trying to reproduce the sound of a musical instrument, the volume number should always be zero, because the note of any instrument will start from zero and increase in amplitude. Other values for the volume number (which specify the starting volume) should be used only for special effects. The range of volume numbers, which is normally 0 to 7, becomes 0 to 15 when an envelope is being used. This allows you a rather better choice of shape for the envelope than would be possible otherwise. The volume number is then followed by an amplitude envelope (or **volume** envelope) number, which will normally be in the range 1 to 15. If you use a zero here, you will get a two-second note of constant volume.

The next thing, then, is to create an envelope shape. The SOUND instruction will only specify that an envelope is to be used, with the choice of fifteen different types. Until you define what each of these envelopes looks like, the SOUND instruction cannot use the envelope. You don't of course, have to specify 15 different envelopes each time you program a sound that needs an envelope. You very often want only one envelope to be used, and so you will pick one number, usually 1. The amplitude envelope shape is then created by using the ENV statement, specifying the number of this envelope right at the start. This ENV statement is one of the most complicated instructions in Locomotive BASIC, but one which is very rewarding to master.

## **The ENV statement**

To start with, there's no way that you can produce an envelope which is precisely like one that you would get from a musical instrument. A genuine amplitude envelope has a pretty complicated shape, and the division into attack, decay, sustain and release is an approximation only. In addition, the real envelope

outline would be a complicated curve, which is very difficult to specify. The nearest that we can get is the use of horizontal or vertical straight lines. This means that sloping lines have to be simulated by 'staircase' shapes, and a curve by a set of straight lines. Nevertheless, we can get sufficiently close to the true envelope shape by these methods to make a very great improvement to the sound of any note from the CPC464. The approximations, however, make it very much easier to specify the shape that we want.



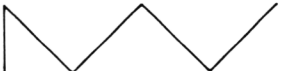
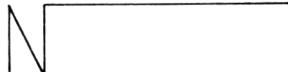

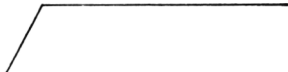


To start with, there are three ways in which we can specify envelopes. The three methods are described as **hardware**, **software relative** and **software absolute**. The Amstrad manual describes in detail only the software relative method, which is the most complicated method. For a lot of purposes, however, you might want to try the other two, and that's what we shall start with. The hardware envelope uses 'built-in' envelope shapes which are part of a ROM that belongs in the sound chip. You can put up to five of these 'hardware' envelope pieces into one single envelope if you want, though for most purposes you will probably want only one. You can also mix a hardware section of envelope with a software section, and that's something that we'll look at later. The form of the ENV instruction for a hardware envelope is:

ENV N%, =H%,P%

using just three numbers and the equals sign. Of these three numbers, the first is the number of the envelope, 1 to 15. The second number, range 8 to 15, decides which hardware envelope will be used, and the third number (range 0 to 65536) sets the period of the envelope.

## Hardware envelopes

The hardware envelopes of the CPC464 allow a considerable number of interesting sounds to be generated, and it's quite a task just to become acquainted with all the possible combinations. To start with, there are eight possible hardware envelopes, whose shapes are sketched in Figure 5.4. The trouble with these sketches is that it's often very difficult for you to associate them

Number	Shape	Description
8		Fast up, slow down and repeat
9		Fast up, slow down, then hold at zero volume
10		Fast up, slow down then repeated slow up, slow down
11		Fast up, slow down, fast up and hold at maximum
12		Slow up, fast down, then repeat
13		Slow up then hold at maximum volume
14		Slow up, slow down and repeat
15		Slow up, fast down and hold at zero volume

**Figure 5.4 The hardware envelope shapes of the CPC464. Numbers 0 to 7 give the envelopes of 9 and 15.**

with the sounds that you hear. That's because the effect that you hear depends critically on the time period of the envelope as well as on its shape. If you choose a very short time period, then any envelope will be played very quickly. That can mean that envelope shapes such as 9 and 15 are almost unheard, and you don't hear the start of envelopes 11 and 13. On the other hand, if you pick a time period that is too great, the time that is taken to change volume is so long that you hear only one section of the envelope. A time period figure of around 10000 is usually the



```

10 FOR W%=8 TO 15:RESTORE
20 PRINT"Waveform No. ";W%;
30 FOR J%=1 TO 4
40 READ N%:PRINT" Period ";N%
50 ENV 1,=W%,N%
60 SOUND 2,239,0,0,1
70 FOR X%=1 TO 6000:NEXT
80 NEXT:NEXT
100 DATA 10,100,1000,10000

```

**Figure 5.5 A program which lets you hear the hardware waveforms.**

```

10 ENV 1,=8,10
20 ENV 2,=8,100
30 SOUND 2,20,0,0,1
40 SOUND 2,50,0,0,2

```

**Figure 5.6 Creating blended notes with hardware envelopes.**

ideal one to allow you to hear what is going on. So that you can judge this for yourself, the program in Figure 5.5 runs over all the possible waveform shapes, and plays each with time numbers of 10, 100, 1000, and 10000. With the time of 10, the effects are hardly noticeable. At 100, you hear a sharp click for waveforms like 9 and 15, and a rasping sound for the repeating waveforms like 8 and 12. This rasp is the result to mixing two frequencies, one being the sound tone that you specify in the SOUND instruction, the other being the fast repetition of the envelope. You can certainly create some interesting effects in this way! Try, for example, the two in Figure 5.6. The use of a short period in the envelope with a high pitch SOUND produces a blend of two notes. With a longer period in the envelope, the sound becomes a warble. Neither has much application to music, but they can both be useful sound effects.

At this point, it's desirable to clear up what is meant by the period number in these envelopes. This is a slightly misleading name, because it suggests that it might be setting the total time of the envelope. In fact, what it sets is the time that is needed for each step of the **changing** part of a waveform. Take, for example, waveform number 13. This consists of a sloping section, called the ramp, and a steady part. The sloping section is represented by sixteen steps, and the time between steps is set by the 'period'

number. According to the Amstrad hardware manual, the units of this number are steps of 128 **microseconds**. Since a microsecond is a millionth of a second, you can also write this number as 0.128 **milliseconds**, where a millisecond is a thousandth of a second. For example, if we pick a period number of 100, it means that each of the sixteen steps will require a time of  $0.128 \times 100$  milliseconds, which is 12.8 milliseconds. This means that all sixteen steps will be completed in  $16 \times 12.8$  milliseconds, which is 204.8 milliseconds. In seconds, this is 0.204, less than a quarter of a second. This is why the low number produces so little effect on our ears. A reasonable approximation is that if you multiply the period number by two, that's the time for the ramp in milliseconds. The manual, however, states that the period number is approximately the time in seconds for a ramp to be completed. Tests with envelope 13, however, suggest that a value of around 5000 for the period number produces a ramp that lasts for one second. Either the manual is misleading, or my ears need a 10,000 decibel service. In any case, whatever number you choose will decide ramp time, but the overall length of time is **always** about two seconds. This is because the envelope instruction automatically puts in a two second pause after any hardware envelope.

This pause can make it appear to be impossible to use the hardware envelopes in music. If you write a tune program and specify a hardware envelope for each sound, then the timing is determined entirely by the envelope, and the tune will go rather slowly. You can, however, get round this. The ENV instruction allows you to put in up to five sections in each complete ENV. Now it would be unusual to want to have five hardware sections, unless you were constructing an envelope using the hardware shapes as pieces. There are easier ways of doing this, as we shall see. If, however, you make the first section a hardware envelope, and then make the second section a short pause, you will be able to get some control over the use of the hardware envelope. To create the pause, you need to add the numbers 1,0,X to the ENV statement, where X is the pause number. This should be a number between 0 and 255. Numbers 1 to 255 give pauses in units of hundredths, so that 255 gives a delay of 2.55

```

10 ENV 2,=9,1000,1,0,40
20 ENV 4,=9,8000,1,0,40
30 FOR N%=1 TO 30
40 READ F%,E%
50 SOUND 2,F%,0,0,E%
60 NEXT
70 DATA 253,2,213,2,213,2,213,2
80 DATA 253,2,213,2,213,2,213,2
90 DATA 190,2,239,2,284,2,239,2
100 DATA 239,2,253,2,253,4
110 DATA 253,2,213,2,213,2,213,2
120 DATA 253,2,213,2,213,2,213,2
130 DATA 190,2,239,2,284,2,338,2
140 DATA 190,2,213,2,213,4

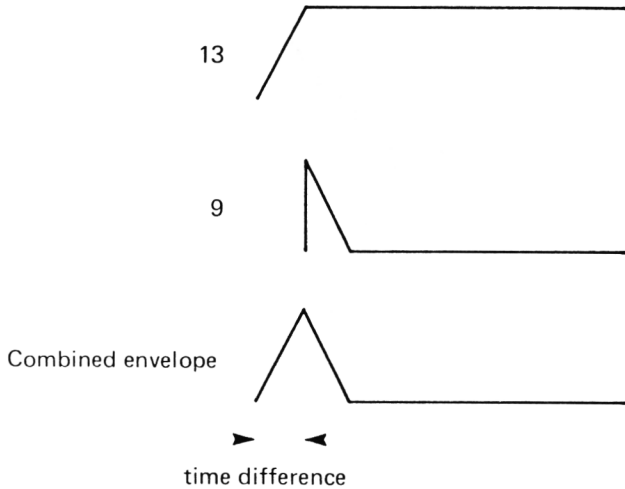
```

**Figure 5.7 A melody which uses an envelope for each duration of note, including a silence.**

seconds. Zero, however, gives the **maximum** delay, of 2.56 seconds. Pauses of 10 to 50 are normal for music. Take a look, for example, at the little tune in Figure 5.7.

This is programmed in what looks like the usual way, with DATA lines that consist of note numbers and what you might think were note duration numbers. It's not quite so simple as that. Because you cannot use a note duration number with an envelope, you have to program one envelope for **each** duration of note. The numbers 2 and 4 in this program are therefore the numbers of different envelopes. I chose these particular numbers simply because they were a convenient reminder of the note durations. Each envelope has been given a period which is appropriate to its note duration, but you have to find these by trial and error. You might expect that a value of 2000 would give a note that lasted for twice as long as a value of 1000, but it doesn't work out this way because these are ramp times, not note durations. The pauses decide the times and use a pause-number of 40, because this suits the note times. Once again, this is something that you have to experiment with, and adjust to whatever you think sounds right. The pause number and the period number for the envelope have to be adjusted together.

Once you have tried out that example, which uses envelope 9, try a modification. This time, the envelope will be made up from



**Figure 5.8** Creating a new envelope pattern from two hardware envelopes. A flat top can be added by putting in a delay.

envelope 13, a ramp rise, followed by envelope 9, which is a ramp fall. The effect of combining the two is to give a waveform which has a triangular shape (Figure 5.8). If we programmed a short delay between them, of course, the result would be a rise, a flat top, then a fall. Try first of all replacing the existing ENV lines by:

```
10 ENV 2, =13,500, =9,500,1,0,40
20 ENV 4, =13,4000, =9,4000,1,0,40
```

and playing this one. The sound is now different because of the different envelope shape. To try out a flat-topped envelope, try:

```
10 ENV 2, =13,500,1,0,7, =9,500,1,0,40
```

— you can hear the effect of the flat top quite clearly in this example, but it depends a lot on choosing the correct number for the pause in the middle. Using 10 gives too long a pause, and 5 is too short. It's because these numbers are so critical, and their effect found only by a lot of trial and error that a lot of Amstrad users tend to be put off the ENV statements.

The best way to regard the hardware envelopes is as a useful set of Lego parts for an envelope. Sometimes you can make use

of one of these envelope parts directly, as I have demonstrated, but it's more likely that you will want to use the software control over envelopes that the CPC464 provides. This consists of two types, the absolute software envelopes and the relative software envelopes. The difference between the two is that the absolute envelope is a rather more crude one, with few changes of volume, whereas the relative software envelope can change volume more smoothly. The programming methods, however, are very similar, and we can take them together.

To start with, we can specify a number of sections of the envelope, up to a maximum of five, just as we could for a hardware envelope. In each of these sections, you can have an outline which consists of a horizontal straight line, or a set of steps that follow the angle of a sloping line. This makes it quite easy, for example, to simulate the classic attack, decay, sustain, release (ADSR) shape with straight lines, and still have one section in reserve. The absolute variety of envelope allows you only one volume setting in each part of the envelope, so that the shape of the envelope is made out of horizontal straight lines with steps up or down only where one section meets another. You have to specify three numbers in each section, a step count, a step size, and a pause time. The step count can be any whole number in the range 0 to 127. If you use zero, then the envelope is an absolute one. The step size gives the amount by which the amplitude is changed in each step. This can be positive (rising amplitude) or negative (falling amplitude). Numbers of -128 to +127 can be used, but in practice since the total range of amplitude is only 0 to 15, it makes little sense to use numbers of more than 2 or 3. Finally the period number is one that we have come across already, it can be in the range 0 to 255, with zero giving a pause of 2.56 seconds.

## **An absolute envelope**

When you use these software envelopes, you have to design each envelope, and the absolute envelopes are simpler from this point of view. Design is best done on graph paper, with centimetre divisions that you can rule off. Figure 5.9 shows an

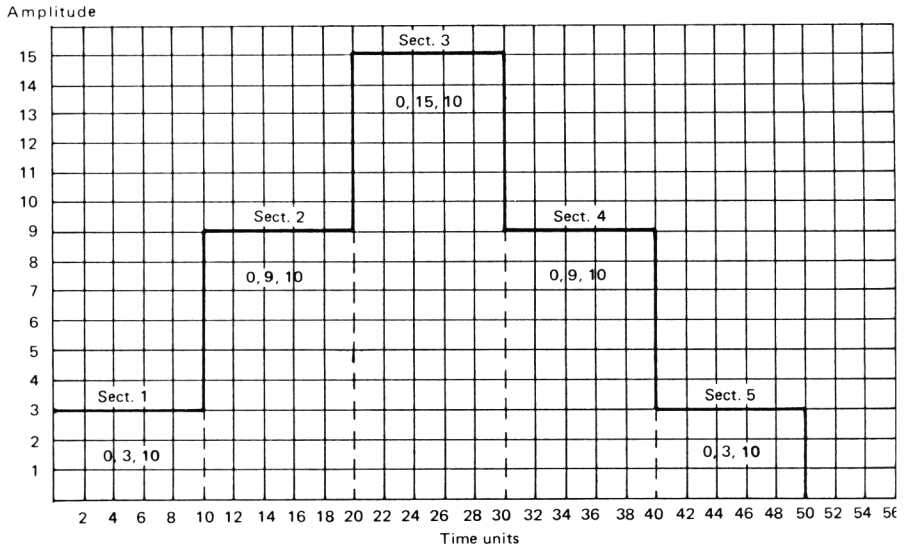


Figure 5.9 Using graph paper to design envelope shapes.

```

10 ENV 2,0,3,10,0,9,10,0,15,10,0,9,10,0,
3,10
20 ENV 4,0,3,20,0,9,20,0,15,20,0,9,20,0,
3,20
30 FOR N%=1 TO 30
40 READ P%,E%
50 SOUND 2,P%,0,0,E%
60 NEXT
70 DATA 253,2,213,2,213,2,213,2
80 DATA 253,2,213,2,213,2,213,2
90 DATA 190,2,239,2,284,2,239,2
100 DATA 239,2,253,2,253,4
110 DATA 253,2,213,2,213,2,213,2
120 DATA 253,2,213,2,213,2,213,2
130 DATA 190,2,239,2,284,2,338,2
140 DATA 190,2,213,2,213,4

```

Figure 5.10 A tune played with notes that use absolute envelope shapes.

example of graph paper with an absolute envelope designed on it. The vertical lines represent volume levels, numbered 0 to 15. The horizontal line shows the times and the number of steps, which for an absolute envelope can be only up to 5, one step in each section. In each step, you have a fixed volume number, which is the second number in the set. The last number then

settles the time for which the section is played. Figure 5.10 shows what happens when the shape of Figure 5.9 is used for the tune that we illustrated earlier. This isn't exactly impressive, and absolute envelopes need some care if the results are to be useful. For one thing, each note follows the next with no pause. That's because we haven't programmed any time at the end of the envelope in which the volume is zero. By using a long pause at the end, and a volume number of zero, we can program the time between notes. This can make the use of absolute envelopes useful just for notes of constant volume, or notes which do not change volume much. The envelope lines in Figure 5.11 show an improved version. In this one, the volume starts high, and is changed by only a count of one on each section. The last section is a silence, longer than the ordinary note sections. The envelope of the crochet uses a **longer section** in the last part of the note, so that the whole note will play for longer — it doesn't work if you just make the silence longer!

One particularly good use for the absolute envelope is the simple creation of an echo effect. If you make the first section of the note have a volume which is large, follow this by a silence, and then by another section which gives a lower volume, you can create an echo effect which is very useful for some purposes.

```

10 ENV 2,0,15,4,0,14,4,0,13,4,0,12,4,0,0
,20
20 ENV 4,0,15,4,0,14,4,0,13,4,0,12,30,0,
0,20
30 FOR N%=1 TO 30
40 READ P%,E%
50 SOUND 2,P%,0,0,E%
60 NEXT
70 DATA 253,2,213,2,213,2,213,2
80 DATA 253,2,213,2,213,2,213,2
90 DATA 190,2,239,2,284,2,239,2
100 DATA 239,2,253,2,253,4
110 DATA 253,2,213,2,213,2,213,2
120 DATA 253,2,213,2,213,2,213,2
130 DATA 190,2,239,2,284,2,338,2
140 DATA 190,2,213,2,213,4

```

Figure 5.11 Using a better envelope shape, with a silent section.

```

10 ENV 2, 0, 15, 10, 0, 0, 10, 0, 7, 10, 0, 0, 20
20 ENV 4, 0, 15, 20, 0, 0, 20, 0, 7, 16, 0, 0, 16
30 FOR N%=1 TO 30
40 READ P%, E%
50 SOUND 2, F%, 0, 0, E%
60 NEXT
70 DATA 253, 2, 213, 2, 213, 2, 213, 2
80 DATA 253, 2, 213, 2, 213, 2, 213, 2
90 DATA 190, 2, 239, 2, 284, 2, 239, 2
100 DATA 239, 2, 253, 2, 253, 4
110 DATA 253, 2, 213, 2, 213, 2, 213, 2
120 DATA 253, 2, 213, 2, 213, 2, 213, 2
130 DATA 190, 2, 239, 2, 284, 2, 338, 2
140 DATA 190, 2, 213, 2, 213, 4

```

**Figure 5.12 Designing absolute envelopes so as to create an echo. This can be very impressive!**

The timing of the notes should not be too short, and the echo should not be too quiet, otherwise it's easy to miss the effect. Figure 5.12 shows the melody of 5.11 used with an echo on each note. Four sections have been used in each envelope, and the silence has been programmed as usual simply by using a section with zero volume. Give a listen to this one, because the principle of the echo is an important one that you might want to come back to when we deal with sound effects.

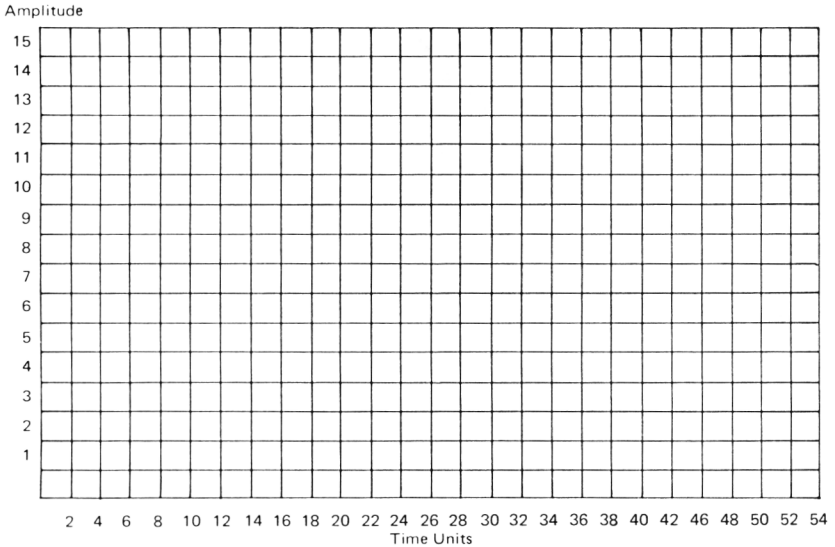
## **The relative ENV**

When you specify a software relative envelope, you are creating an envelope shape which is considerably more complicated than any that you can make up from the hardware shapes, or by the use of software absolute methods. Nevertheless, now that you have used a software absolute envelope, the creation of a software relative envelope is not such a large step into the unknown. To start with, we can specify a number of sections of the envelope, up to a maximum of five. In each of these sections, you can have an outline which consists of a horizontal or vertical straight line, or a set of steps that follow the angle of a sloping line. This makes it quite easy, for example, to simulate the classic attack, decay, sustain, release (ADSR) shape with straight lines, and still have one section in reserve for a silence at the end of the note. Alternatively, you can simulate a curve which does not



conform to the classic ADSR shape at all. The ENV instruction word is followed as usual by the number 1 to 15 which is the reference number for the envelope. You then have three numbers for each section of the envelope. The first of these numbers is the step count number. For an absolute envelope, this number would be zero, but for a relative envelope it gives the number of steps of volume change which will be used in this section of the envelope. Remember that the starting volume will be decided by the number that is used in the SOUND statement. If you make the starting volume equal to zero, as is usual for envelope control, then the first section of the envelope will be used to specify the number of volume steps up to some volume level which will form the 'attack' section of the note. The number of steps for a relative envelope can be 1 to 127, but the practical range is 1 to 15, because if the volume changes by one unit in each step, then there is no point in having more than 15 steps. The second number in each section is the step size, as it was for the absolute envelope. Here again, you are permitted numbers in the range -128 to +127, but since there are only volume steps of 0 to 15, you must choose sensible figures. The final number, as for the absolute envelope, is the pause time, in units of 1/100 of a second. The number range is 1 to 256, with 0 giving the effect of the number 256 (2.56 seconds).

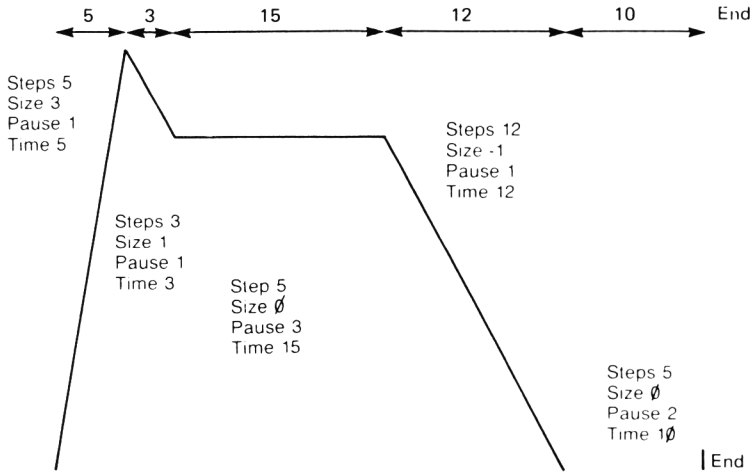
What you need to watch is how these numbers interact with each other. The step count multiplied by the pause time, for example, gives the total time for the section. This should not be ridiculously short, otherwise your ear simply will not detect the effect. It should not be too long either, otherwise each note will last for so long that it will be impossible to use for a melody unless a very slow tempo is wanted. Suitable values are something that you simply have to learn by trial and error, and the examples in this book should be a good starting-point for you. The other interaction is that the step time multiplied by the step size gives the volume change. If you have ten steps, each of size one unit, for example, then your volume change in the section will be ten volume units. If the volume started at zero (in the SOUND instruction), then it will end up at level 10. If the volume started at level 5, it will end up at 15, the maximum. You might, of course, want the volume



**Figure 5.13 Graph paper set out to design a relative envelope.**

to increase more sharply, such as by having five steps, each of size 3 which would give a volume change of 15. This, however, is a rapid change only if the pause time is short. If a long pause time has been chosen, all this would do would be to make a fairly large volume change five times over a comparatively long period. For a really rapid rise, you would require one step of 15 volume units, with one time unit.

Since you are allowed five sections, the most obvious way of using the ENV instruction is by specifying ADSR sections, with a silence at the end specified by the fifth section. Let's hear how that sounds with an example, which will also show how we can plan out these envelope shapes. We start, as always, with graph paper, and Figure 5.13 shows how this should be marked out. I use graph paper for convenience, simply because it's marked with a grid of lines. It's a good idea **not** to draw your envelopes directly on to the graph paper, but to work instead on tracing paper with pencil. This is because you'll inevitably make a lot of mistakes at first, and draw a lot of envelopes that either can't be programmed or which don't produce anything like the results you are looking for. By using tracing paper and pencil, you can



**Figure 5.14 An envelope designed by placing tracing paper over the graph paper.**

experiment as much as you like, rubbing out the unsatisfactory envelopes, and inking in the ones that produce good results. You should, incidentally, be quite careful about this. It's never obvious from the shape of an envelope just how the sound will be, and you can make your work a lot more durable if you write on the sheet which contains the envelope shape the ENV program line, and also what it sounded like. The drawback about using graph paper is that you tend to think that you have to work with units of the same size. You needn't, however, use the same step size, number of steps or pause time in each section. Figure 5.14 shows an envelope designed on the lines of Figure 5.12. Don't attempt to draw the steps, because this is time-wasting and difficult. Represent rises or falls by sloping lines, and show the total time of each section, because this is important, particularly when you want to make any changes in the note. When you add the times for these sections, you will get the total time for the note. In this example, the attack consists of 5 steps of 3 units each to the total volume of 15. This is a comparatively slow attack, much slower that you would get from a plucked or struck instrument. The total time for this attack is 5 units, because the pause has been specified as one unit. The decay is then of three steps,  $-1$  volume units and time 1 unit. This makes the

```

10 ENV 2,5,3,1,3,-1,1,5,0,3,12,-1,1,5,0,
2
20 ENV 4,5,3,2,3,-1,2,5,0,6,12,-1,2,5,0,
2
30 FOR N%=1 TO 30
40 READ P%,E%
50 SOUND 2,P%,0,0,E%
60 NEXT
70 DATA 253,2,213,2,213,2,213,2
80 DATA 253,2,213,2,213,2,213,2
90 DATA 190,2,239,2,284,2,239,2
100 DATA 239,2,253,2,253,4
110 DATA 253,2,213,2,213,2,213,2
120 DATA 253,2,213,2,213,2,213,2
130 DATA 190,2,239,2,284,2,338,2
140 DATA 190,2,213,2,213,4

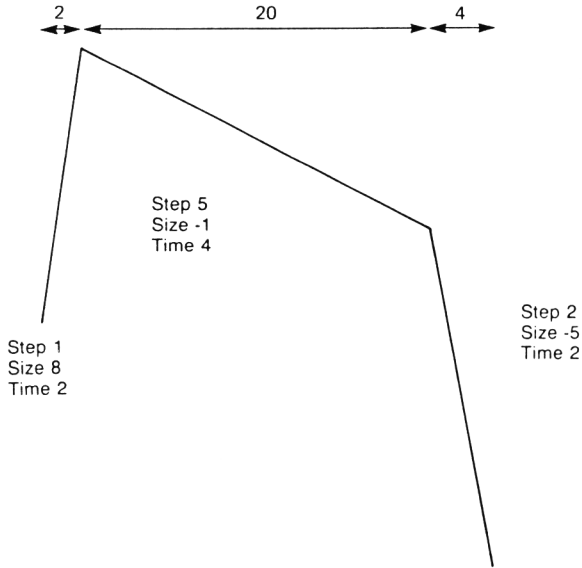
```

**Figure 5.15 A tune played using slow attack and decay.**

decay go down in volume from 15 to 12, and take three time units.

The sustain section uses 5 steps of zero volume change, with pause 3. This makes the time equal to 15 units, with no change in the volume. The release is then slow, using 12 steps of size  $-1$  to get the volume back from level 12 to zero. The pause is 1, so that the time is 12 units. Finally, the volume is held steady (at zero) for five steps of pause 2, a total of 10 time units. The total time for the whole note is 42 units, corresponding to 0.42 seconds. The envelope is programmed by the lines of Figure 5.15, using longer times for the minim envelope, and when you use this in the melody, you'll hear the typical sound of an envelope which has comparatively slow attack and decay. Each note sounds as if it's afraid to come out! Sounds like this have their uses when you can also control the waveshape, but for the square wave pattern that the CPC464 produces, such an envelope is not a particularly good one, though it can produce interesting piano-accordion tunes. For more interesting sounds we have to look at much sharper attack times in particular.

There are two ways in which we can produce steeper attacks. One is simply to use fewer steps with greater amplitude change, and minimum pause time. The other is to use some starting



**Figure 5.16 Using a much sharper attack with the amplitude starting at level 7.**

amplitude in the SOUND instruction. If the SOUND starts with a volume of 5, for example, it's easier to make a sharper attack. The ultimate is to start the volume at 15, and use the envelope to provide only the D, S and R sections. This provides the sharpest attack that you can get with this system. Figure 5.16 shows a plan for a shorter note with steeper attack. This has been achieved by starting the SOUND volume at a figure of 7, and increasing this to 15 in one step with a duration of two units. The envelope then decays and releases in two further sections, with no steady hold portion. This, as you can hear from Figure 5.17, gives a reasonably good 'plucked string' sound, with a sharpness to it that is good for picking out a melody. If you alter the envelope so that it consists of the attack, followed by a decay that is obtained from the numbers 15, -1, 1 then you will get an even sharper 'plucked string' type of note. The tempo of the music will, however, be faster because of the decreased length of the whole note. When you are designing envelopes for musical effects, you have to be very careful about total length because, as we have seen, you need one envelope for each note-length. Any alteration that you make to any section of an envelope is

```

10 ENV 2, 1, 8, 2, 5, -1, 4, 2, -5, 2
20 ENV 4, 1, 8, 2, 5, -1, 8, 2, -5, 4
30 FOR N%=1 TO 30
40 READ P%, E%
50 SOUND 2, F%, 0, 7, E%
60 NEXT
70 DATA 253, 2, 213, 2, 213, 2, 213, 2
80 DATA 253, 2, 213, 2, 213, 2, 213, 2
90 DATA 190, 2, 239, 2, 284, 2, 239, 2
100 DATA 239, 2, 253, 2, 253, 4
110 DATA 253, 2, 213, 2, 213, 2, 213, 2
120 DATA 253, 2, 213, 2, 213, 2, 213, 2
130 DATA 190, 2, 239, 2, 284, 2, 338, 2
140 DATA 190, 2, 213, 2, 213, 4

```

**Figure 5.17** The program which gives the envelope of Figure 5.16.

likely to affect the total length, and so throw out your timing. You can make up for this by altering the timing of a silent section, or by juggling steps between attack and decay sections. It's always a 'cut and fit' business, seldom the matter of precisely planning a good-looking envelope that you might expect.

## **An envelopes program**

Many envelope-planning programs exist, but a lot of them are not really helpful unless you have considerable experience. The trouble with planning envelopes is that if you have an immense amount of choice, you end up simply playing aimlessly with the sound. You can certainly discover a lot of interesting sounds in this way, but if you are trying to obtain some specific effect, the method can be very time-consuming and frustrating. What follows is a very simple envelope planner which I have found very fast and useful. The principle is to restrict your choice to values which give reasonable results, so that you can alter values easily and quickly, listening to the note that is produced. You can also go back down your list of choices, making any changes that you want to, and being reminded of the previous choice. This is a very much quicker method of editing an envelope than one which requires you to input or draw each part of the envelope for yourself, and it can save a lot of time when you are chasing that elusive sound. When you end the program, the ENV statement

is printed out so that you can use the editor to make a copy. You can, of course, make any further changes that you want in the way of 'fine-tuning' on the ENV line.

The listing is shown in Figure 5.18. The program consists of a large main loop which starts in line 40 and which will repeat until you press the 'Q' key at the end of the loop, or use the ESC key

```
10 CLS:M$="Mistake - please try again"
20 PRINT TAB(16)"ENVELOPE"
30 LOCATE 1,2
40 WHILE K$<>"Q"
50 PRINT"Attack - Fast, Medium, Slow (FM
S) ":X$=A$
60 GOSUB 470:IF R$="P" THEN PRINT"-";X$:
GOTO 110:ELSE A$=R$
70 IF A$="F" THEN A1%=1:A2%=15:A3%=1
80 IF A$="M" THEN A1%=3:A2%=5:A3%=1
90 IF A$="S" THEN A1%=5:A2%=3:A3%=2
100 IF INSTR("FMS",A$)=0 THEN PRINT M$:G
OTO 50
110 PRINT"Decay -None, Fast, Medium, Tot
al (NFMT) ":X$=B$
120 GOSUB 470:IF R$="P" THEN PRINT"-";X$
:GOTO 180:ELSE B$=R$
130 IF B$="N" THEN A4%=1:A5%=0:A6%=1
140 IF B$="F" THEN A4%=1:A5%=-3:A6%=1
150 IF B$="M" THEN A4%=3:A5%=-1:A6%=2
160 IF B$="T" THEN A4%=15:A5%=-1:A6%=2:EN
V 1,A1%,A2%,A3%,A4%,A5%,A6%,10,0,10:GOTO
400
170 IF INSTR("NFMT",B$)=0 THEN PRINT M$
:GOTO 110
180 PRINT"Sustain - None, Short, Medium,
, Long (NSML) ":X$=C$
190 GOSUB 470:IF R$="P" THEN PRINT"-";X$
:GOTO 250:ELSE C$=R$
200 IF C$="N" THEN A7%=1:A8%=0:A9%=1
210 IF C$="S" THEN A7%=2:A8%=0:A9%=2
220 IF C$="M" THEN A7%=3:A8%=0:A9%=4
230 IF C$="L" THEN A7%=5:A8%=0:A9%=5
240 IF INSTR ("NSML",C$)=0 THEN PRINT M$
:GOTO 180
```

```

250 PRINT"Release- Quick, Medium, Slow (
OMS) ":X$=D$
260 V%=A1%*A2%+A4%*A5%
270 GOSUB 470:IF R$="P" THEN PRINT"-";X$
:GOTO 320:ELSE D$=R$
280 IF D$="Q" THEN A10%=1:A11%=-V%:A12%=
1
290 IF D$="M" THEN A10%=2:A11%=-V%/2:A
12%=2
300 IF D$="S" THEN A10%=3:A11%=-V%/3:A12
%=3
310 IF INSTR("OMS",D$)=0 THEN PRINT M$:G
OTO 250
320 PRINT"Silence time, Long, Medium, Sh
ort (LMS) ":X$=E$
330 GOSUB 470:IF R$="P" THEN PRINT"-";X$
:GOTO 380:ELSE E$=R$
340 IF E$="L" THEN A13%=10:A14%=0:A15%=1
0
350 IF E$="M" THEN A13%=6:A14%=0:A15%=6
360 IF E$="S" THEN A13%=3:A14%=0:A15%=5
370 IF INSTR("LMS",E$)=0 THEN PRINT M$:G
OTO 320
380 ENV 1,A1%,A2%,A3%,A4%,A5%,A6%,A7%,A8
%,A9%,A10%,A11%,A12%,A13%,A14%,A15%
390 SOUND 130,0,0
400 SOUND 2,239,-20,0,1
410 PRINT"Press Q KEY to QUIT, any other
key to":PRINT"repeat choice. Pressing t
he P KEY":PRINT"lets you skip that stage
and":PRINT"shows the previous choice."
420 K$=INKEY$:IF K$=""THEN 420
430 WEND
440 PRINT"Envelope parameters are:"
450 PRINT"ENV 1,";A1%;", ";A2%;", ";A3%;",
";A4%;", ";A5%;", ";A6%;", ";A7%;", ";A8%;",
";A9%;", ";A10%;", ";A11%;", ";A12%;", ";A13
%";", ";A14%;", ";A15%
460 END
470 R$=INKEY$:IF R$=""THEN 470 ELSE RETU
RN

```

Figure 5.18 The envelope-editor program. This uses a restricted range of preset values to make your selection quicker and easier.



at any other point. You will, however, see the ENV printout only if you quit the program with the Q key. You are asked first for a choice of attack, and the choices are fast, medium or slow only. If you type F, M, or S and ENTER, values are assigned to the first section of the envelope, using integer number variables A1%, A2% and A3%. Each selection will cause the volume level to go from zero to 15, but the rate of rise is different. You should not type 'P' at this stage, and if you have typed any letter other than P, F, M or S, then line 100 will catch the error and ask for a re-run. Lines 110 to 170 then use a similar method to obtain your choice of decay. In this set, however, you have the choice of Total decay, meaning that the amplitude will decay to zero, with no sustain or release sections. This requires a rather different treatment, because for this choice, the envelope contains only an attack and a decay section. A silence section is added, and the envelope is played. The other choices are more orthodox, and they lead to values for the decay section of A4%, A5% and A6% and then proceed to the sustain section.

The sustain and release sections are dealt with similarly, and then you are finally asked to choose a silence time to separate repeated notes. This will also help to determine the rate at which notes are played. The number parameters A1% to A15% are then assigned to the ENV statement in line 380, assuming that the Total decay choice was not made. The sound channels are then cleared by line 390 (adding 128 to a channel number clears the sound queue), and the note is played by the SOUND instruction in line 400. This uses a duration number of -20. When a negative duration number is used, the number is taken as a number of repetitions for the envelope. Using a duration time of -20 will then ensure that the envelope is repeated twenty times. If, of course, you have made the envelope a very long one, and have specified a short silence, you will hear only a long pulsing note instead of the repetition. As always, you have to make reasonably consistent choices and using a short silence along with a long envelope is the only one that can cause confusion. The quantities that are used for amplitude changes have been organised so that any combination of ADSR choices will produce

a note. This means that when you choose a decay, it is always from 15 to 12, because the volume figure of 12 can be released completely in 2, 3, or 4 steps. This would not be possible for any other choice of number other than 6, and though the program works if other decay amplitudes are chosen, the note sounds different on each repetition because the volume number is not being returned to zero on each repetition. Alter the program by all means to suit yourself, but don't ignore the problems that this could cause! One minor point is that if Total decay is chosen, the ENV printout is not completely accurate, because it shows a set of zeros for the sustain, release and silence sections. These zeros can, of course, be edited out.

## Pitch Envelopes

As well as the amplitude envelope, which decides the way that amplitude changes during the time of a note, you can also alter the pitch of a note while it is being sounded. Now for musical notes, this is a way of getting vibrato, but it must conform to rules if it is to sound successful. The rate of vibrato has to be carefully chosen, and should be faster for high notes than for low notes. The amount of vibrato also has to be regulated. It is normally less than a semitone, because large amounts of pitch change sound ridiculous — excessive vibrato is the main curse of the type of amateur singer who sounds as if someone was using his/her chest as a drum skin. We shall want to keep to small changes for this chapter, but for special effects discussed in the following chapter, we can make use of much greater vibrato and other pitch envelope effects. The use of a pitch envelope requires an extra number tacked on to the SOUND instructions, and the use of an ENT statement to define the pitch envelope.

The ENT statement follows so closely the pattern of the ENV statement that we can dispose of it quite quickly. As usual, following ENT, we have the envelope number with the usual range of 1 to 15. Choosing 0 will have the effect of leaving the note as a steady one. There is a difference here, however, because you can use a negative number as the envelope number. If you do so, then the pitch variation will be repeated until the note ends. The end of the note will be decided by its amplitude

envelope or by the duration number in the SOUND statement. Following the envelope number, you can then define up to five sections of pitch change, each of which can be an **absolute** setting or a **relative** one. The difference is that the absolute setting decides a pitch number which is unchanged for a specified time, but the relative setting can specify a rate of change and an amount of change of pitch.

An absolute section of pitch envelope is obtained rather in the way that a hardware amplitude envelope is obtained. Following the envelope number, or the previous section, there is a comma, then an equality sign, and then a tone period number, comma, and a pause time number. The tone period will be a number in the normal range (0 to 4095 permitted, 20 to 2000 more realistic), and the pause time is a number in the range 0 to 255, with 0 providing the **longest** pause of 2.56 seconds. In such an absolute section, the tone period number gives the note that will be sounded, and the pause time gives the time in hundredths of a second for which it will be sounded. If the first section of a tone envelope is an absolute section, the tone period number in the SOUND instruction will be ignored. For example, using ENT 1, = 239,50 will sound a C' note for half a second. Try as a quick way of getting acquainted with it, the lines:

```
10 ENT -1, =239,100, =190,100
20 SOUND 2,0,500,7,0,1
```

which will have the effect of sounding the notes C' and E' alternately. Now cut down the pause time in the ENT line to 1 from 100, and listen to the result. The absolute tone envelope is a very useful way of programming short phrases, particularly if they have to be repeated, and it can often be much simpler than the use of a SOUND instruction in a loop with data read in. This is particularly true if you only want a rapid jingle for a game, or a theme for each character in an adventure, or a different warning tune for different actions in a business program.

The relative type of envelope uses three numbers in each section. The first number is the number of steps, the second is the step size, and the third is the pause time. The step size, however, is

```

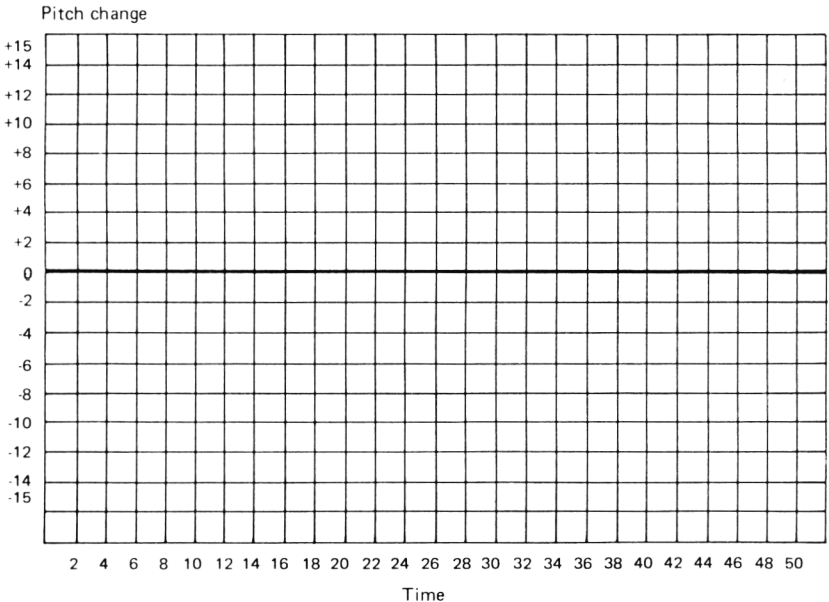
10 ENT 1,4,1,1,8,-1,1,4,1,1
20 SOUND 2,239,100,15,0,1

```

**Figure 5.19** A short illustration of vibrato, using a pitch envelope.

not the size of an amplitude step but of a pitch step, and it corresponds to a change in the note number. For example, if you are playing note 279, then a step of +1 means a change to 280, which is a **lower** pitch. These step sizes can be positive (lower pitch) or negative (higher pitch). The pause numbers are in the usual units of 1/100 second intervals. Figure 5.19 illustrates a note which starts with a bit of vibrato. The pitch changes in four steps, each of one unit, positive, then eight negative, then four positive again, ending with the same note. In the SOUND portion of the program, the usual numbers are specified, but with 0 for amplitude envelope. This is because there must always be an amplitude envelope number preceding the ENT number, and if you have no amplitude envelope specified, then a zero must be used in this position.

As usual, it helps to some extent if we can draw the pitch en-



**Figure 5.20** A planner for pitch envelope shapes — this is more useful for special effects.

For a musical vibrato:

1. Keep the pitch change number to 1 for high notes, 2-3 for low notes.
2. Make the period number 1 for high notes, 2-3 for low notes.
3. The step count should be small, starting with 1-2 in the first section.
- 4 Use three sections. The step count in the middle section should be negative and twice the size of the step count for the first section. The step count in the third section should be equal to the step count for the first section.

**Figure 5.21 Designing ENT values for vibrato — this guide is more useful for music.**

velope, and Figure 5.20 shows a planner for ENT shapes. This is more useful for special effects, however, and for musical notes, it's easier to follow the guidelines of Figure 5.21. Now since most musical notes that use vibrato tend to have the vibrato used for as long as they are sounded, it's normal to use the negative envelope numbers. Notice that the number is negative **only in the ENT statement** — you must not use a negative ENT number in the SOUND statement. When you make the ENT number negative in the program of Figure 5.19, you find that the vibrato is rather excessive. The remedy is a smaller number of steps, since you have already fixed the minimum possible pitch change. You will then have to juggle with the number of sections and the pause times to get the rate of vibrato that you want. Unlike amplitude envelopes, your choices are very much more restricted if you want to use ENT for musical notes, because only a very limited range of ENT effects sounds reasonably musical. It's quite different when we get to special effects, as you'll see.

As an example of how notes can sound with both amplitude and pitch envelopes, try the scale in Figure 5.22. This uses an amplitude envelope which gives fast attack, small decay, medium sustain and fast release, with a medium length silence. The pitch envelope makes the pitch number rise in three steps, fall by six, then rise three again to give the normal zig-zag pitch envelope shape. The total time for the pitch envelope is 12 hundredths of a second, and this is a figure which is 'average' for pitch envelopes for notes in this range. Faster changes should be used

```

10 ENV 1, 1, 15, 1, 2, -3, 1, 5, 0, 5, 1, -9, 1, 5, 0,
4
20 ENT -1, 3, 1, 1, 6, -1, 1, 3, 1, 1
30 FOR N%=1 TO 8
40 READ F%
50 SOUND 2, F%, 0, 0, 1, 1
60 NEXT
100 DATA 478, 426, 379, 358, 319, 284, 253, 239

```

**Figure 5.22 A scale of notes which have both amplitude and pitch envelopes.**

for notes which are around one octave higher. Don't feel that because a pitch envelope can be used that it must be used. A lot of music sounds better without vibrato, and if you want to use a melody and accompaniment, it's better to confine any vibrato to the melody.

At this point, then, we turn from strictly melodic musical effects to something quite different — the use of noise. This will lead us into rhythm instruments, such as drums, and also in the realm of more advanced sound effects. Time, I think, for a new chapter.

# 6

## Using Noise

What we call noise is a kind of random sound. Musical notes have definite values of pitch and amplitude, noise has not. We can generate electronically what is called 'white noise', meaning that the sound contains a mixture of all frequencies and a large range of amplitudes. When you see such white noise displayed on the screen of an oscilloscope, it looks like a band of thick cloud, with no distinct pattern or clear edges. The closest that you usually get to hearing noise of this type is when you tune between stations on an FM radio. Natural noises, however, aren't like this. The noise that you get from knocking two bits of wood together, for example, is not anything like white noise. It has an amplitude envelope with very fast attack and decay, and the range of frequencies is quite limited. The frequency range of the noise, in fact, is centred round a definite note which is related to the length of each piece of wood. Similarly, when you strike a drum skin, the sound is closer to noise than to a musical note, but once again, the frequencies that are produced are centred around one definite value, which depends on the size and tightness of the drum skin. All natural noises, from raindrops through surf on the shore to thunder are of this type.

It's at this point, too, that descriptions become difficult, and you have to use your ears carefully. Up to now, we could say that a program produced a scale of C Major, and you knew what to expect, no matter what kind of fancy work was being done in the way of amplitude or pitch envelopes. When noise is used, however, it's a lot more difficult to know what a program will produce, and small changes can often make the sound very different to the ear. The examples in this chapter are all starting

points, a basis for you to make changes and listen to what you get. The possibilities are so great that no-one could claim to have created, listened to and catalogued every possible sound effect. We'll look at programs which create some noises that many programmers want to have available for games and other purposes, but you can always modify these programs to produce some dazzling effects of your own.

The CPC464 allows you to specify noise of 15 different 'centre tones'. This is done by adding a noise number, range 0 to 15, following the tone envelope number. Using zero as the noise number will, as usual, produce no noise. Noise can be produced alone or in conjunction with other sounds. If you specify a pitch number of zero for the sound channel, then no musical note will be produced, allowing the noise only in that channel. Note, however, that there is only one noise generator, and you cannot have different noises in each channel. The best place to start is by listening to straightforward noises to hear the effect of the noise 'pitch' numbers 1 to 31. Note that the CPC464 manual specifies 0 to 15, but the Complete Basic Specification manual reveals that 0 to 31 can be used. Figure 6.1 illustrates this by cycling through the effects of these numbers from 1 to 31. The noise is rather like a release of steam, and the 'pitch' starts high and ends low. Even with this comparatively simple type of statement, you can produce a lot of interesting effects. By cycling through these noise pitches more rapidly, for example, you can produce a 'raging surf' sound, ideal for these tropical beaches. This one is illustrated in Figure 6.2. The natural surf sound varies in pitch

```
10 FOR N%=1 TO 31
20 SOUND 2,0,50,15,0,0,N%
30 FOR J=1 TO 1000:NEXT
40 NEXT
```

**Figure 6.1 The various noise pitch numbers and how they sound.**

```
10 FOR X%=1 TO 10
20 FOR N%=1 TO 31
30 SOUND 2,0,15,15,0,0,N%
40 NEXT:NEXT
```

**Figure 6.2 The 'raging surf' sound produced by cycling rapidly through the noise numbers.**



```

10 FOR J%=0 TO 79
20 IF J% MOD 4=0 THEN V%=15 ELSE V%=5
30 SOUND 2,0,10,V%,0,0,5
40 FOR X=1 TO 150:NEXT
50 NEXT

```

**Figure 6.3 The steam locomotive sound, using emphasis on the first noise of each group of four.**

from low to high because the size of the wave that produces it is reduced as the wave breaks. In this program, the use of noise pitch numbers 1 to 31 produces the correct variation in the predominant pitch. Another simple use of noise is illustrated in Figure 6.3, which produces the much-loved steam locomotive note. This is not such a good loco simulation as would be produced by using an envelope, but for a simple routine, it is quite effective. The noise pitch is fixed at 5 and duration at 10, with the volume of one in each four being boosted. This gives the correct rhythm to the effect which would otherwise be rather unconvincing.

Noise, however, really comes into its own when you can make use of an amplitude envelope along with the noise. This does not need to be an elaborate relative software envelope, because impressive effects can be obtained from the range of hardware envelopes. The hardware envelope 9 is particularly useful for a variety of sounds that have a steep attack, and Figure 6.4 illustrates gunshots in stereo. A useful tip here is to make the noise pitches slightly different. In this example, one pitch is 10, the other 12. This produces the effect of one gun being further away than the other or in different surroundings, and combined with the stereo effect gives a good impression of a running battle.

```

10 ENV 1,=9,1000
20 FOR N%=1 TO 12
30 SOUND 4,0,0,0,1,0,10
40 FOR J=1 TO 1000:NEXT
50 SOUND 132,0,0
60 SOUND 1,0,0,0,1,0,12
70 FOR J=1 TO 1000:NEXT
80 SOUND 129,0,0
90 NEXT

```

**Figure 6.4 Gunshots in stereo — try this on a stereo system!**

Note that this program is not entirely straightforward because of the requirement for stereo. If you simply feed the sound instructions into the sound queues, it's very difficult to stop them from synchronising, with the effect that you completely lose the stereo effect, and both guns sound in the middle. By using a time delay, with the queue for that channel flushed clear immediately afterwards, you can ensure that the signals on the different channels are well separated. Unwanted synchronisation is the curse of stereo sound effects, and it can often be very difficult to avoid.

Meanwhile, back to the sound effects. The repeating hardware envelopes are a lot more useful for sound effects than for music. One particularly useful application is to sound effects which consist of fast rhythmic noises, like old-style aero engines with pistons. By using a fairly small time number in the hardware envelope, along with shape 9, you can create quite a convincing noise for a World War I Sopwith Camel (all right, SE5A if you prefer it) warming up, as Figure 6.5 illustrates. Just to show how versatile this system is, try changing to hardware shape 14 in this program. You'll now hear the characteristic swishing of helicopter blades. Using ENV 1,=12,200 is also a very good helicopter blade sound. The combination of hardware shapes can also be usefully employed, particularly shapes 13 and 9. Used with short times, these give gunshot sounds, but with long times, you can get the sort of steam hammer effect that you can hear in Figure 6.6.

Obviously, you can make use of software relative envelopes to

```

10 ENV 1,=8,100
20 FOR N%=1 TO 12
30 SOUND 2,0,0,0,1,0,10
40 NEXT

```

**Figure 6.5** An envelope which gives the effect of an old-time aero engine.

```

10 ENV 1,=13,20000,=9,20000
20 FOR N%=1 TO 12
30 SOUND 2,0,0,0,1,0,10
40 NEXT

```

**Figure 6.6** A steam-hammer noise — add a few clanks and rumbles and you have the Victorian iron foundry effect!

```

10 FOR NZ=1 TO 20
20 ENV 1,5,5,2,4,-2,1,5,0,3
30 SOUND 2,0,0,0,1,0,31
40 NEXT

```

**Figure 6.7 Running feet on gravel — or joggers in the park!**

control a noise, and once we get into this work, the sky's the limit! Take a listen to Figure 6.7, which is the sound of running feet on a gravel path. You can try altering the timing of the various sections of the envelope, and the predominant frequency of the noise, to produce a huge range of effects. It's because there are so many parameters that can be altered that you find yourself often spoiled for choice. If you modify the envelope generator program so that the SOUND statement gives noise effects rather than musical notes, however, you can experiment endlessly with new sounds which use only noise in their composition.

Another dimension can be introduced by adding pitch envelopes to noise. The effect of this is **not** to alter the predominant pitch of the noise, because this is set by the last number in the SOUND instruction, but to **add** a changing musical note. If you use a positive step size in the ENT statement, then the note will start at high pitch and change to lower pitch. This can make very impressive sound effects for spaceships, zap-guns and all the other items of games programs. Take a listen, for example to Figure 6.8, which has been created by combining a hardware envelope number 13 with a whistling sound that moves down the scale. The hardware envelope creates a sound that becomes steadily louder, and the ENT statement provides two hundred steps of pitch change. The two together make a most impressive effect. Now change the number of steps to 200, and the step

```

10 ENV 1,=13,6000
20 ENT 1,200,1,1
30 SOUND 2,0,0,0,1,1,10

```

**Figure 6.8 Combining a whistling pitch envelope with noise.**

```

10 ENV 1,=11,6000
20 ENT 1,200,1,1
30 SOUND 2,0,0,0,1,1,10

```

**Figure 6.9 How the effect of Figure 6.8 can be changed by using a different envelope.**

```

10 ENV 1,=8,300
20 ENT 1,200,1,2
30 SOUND 2,0,0,0,1,1,31

```

**Figure 6.10 The falling-down-a-ladder sound.**

```

10 ENV 1,=14,5000
20 ENT 1,200,-4,2
30 SOUND 2,500,0,0,1,1,1

```

**Figure 6.11 A good sound effect for a rapid disappearance.**

size to 2, and hear the difference. Make the period number in the ENT statement also equal to 2, and the sound changes again. Another change brings us to Figure 6.9, which creates two rather different sounding effects in one statement. Because hardware envelope 11 has been used, the sound starts off rather like the earlier version, but the volume drops to zero. When the volume takes up again, the pitch of the whistle has changed considerably, giving the impression of two quite different sounds. This is another good one for your 'space wars' library of sounds.

Now try the combination of a repetitive hardware envelope with a pitch envelope in Figure 6.10. This is one for looney miners falling down ladders! The ENT statement does not have to be confined to descending notes either. If you want an ascending note, though, you must put a starting value into the SOUND statement. Figure 6.11 illustrates this idea, with a good 'disappearing out of sight' sound. This uses a negative step size in the ENT statement, with a starting tone number of 500 in the SOUND line. A repetitive hardware amplitude envelope is used, and the result, well listen to it for yourself!

Another fruitful field for experiments is in rhythm noises for accompanying music. Figure 6.12 gives a passable imitation of a cymbal and drum rhythm. This uses two envelopes, one with a ramp up and quick release, the other with a fast attack and

```

10 ENV 1,15,1,1,1,-15,1,5,0,4
20 ENV 2,1,15,1,15,-1,1,5,0,5
30 SOUND 2,0,0,0,1,0,20
40 SOUND 2,0,0,0,2,0,25
50 SOUND 2,0,0,0,2,0,30

```

**Figure 6.12 Synthesising the effects of cymbals and drums.**

```

10 ENV 1,15,1,1,1,-15,1,5,0,4
20 ENV 2,1,15,1,15,-1,1,5,0,5
30 FOR N%=1 TO 8
40 READ A%,B%,C%
50 SOUND 17,A%,40,7
60 SOUND 10,0,0,0,1,0,20
70 SOUND 17,B%,30,6
80 SOUND 10,0,0,0,2,0,25
90 SOUND 17,C%,30,6
100 SOUND 10,0,0,0,2,0,20
110 NEXT
120 DATA 239,213,190,213,239,284
130 DATA 284,253,239,284,319,358
140 DATA 319,284,253,239,213,190
150 DATA 213,190,239,213,213,213

```

**Figure 6.13 A melody of sorts with rhythm accompaniment.**

slow decay. The drumbeats use different noise numbers so that they sound slightly different. This can be combined with a melody in another channel, and using synchronisation. Synchronisation is essential in programs of this type, because otherwise the rhythm and the melody would inevitably get out of step. The result is shown in Figure 6.13 — not exactly hit-parade material, but it's a start! Obviously you can synthesise any drum rhythm you like, using suitable combinations of envelopes — after all, you have fifteen to play with. All but the most complicated rhythm patterns should be attainable, but as usual, it's hard work, and you need a keen ear to check the results. Keep to reasonably easy rhythms at first, until you get used to the techniques, particularly of putting in silences. This is always the trickiest part, and it's the bit that makes software relative envelopes much superior to other types for this sort of activity.

Some types of drumming and other noise have pitch changes of noise during the time of the sound. These are particularly difficult to synthesise, and the best way is to queue up a number of SOUND instructions with different noise numbers. Figure 6.14 gives a passable imitation of a snare-drum by using an envelope with medium attack and a loop which carries out SOUND instructions with different noise numbers. This is followed by a delay, then two sharp bursts of noise. The result is the drum roll

```

10 FOR J%=1 TO 3
20 ENV 1,3,5,1
30 FOR N%=23 TO 15 STEP -2
40 SOUND 2,0,0,0,1,0,N%
50 NEXT
60 FOR X=1 TO 400:NEXT
70 FOR K%=1 TO 2
80 SOUND 2,0,10,7,0,0,5
90 FOR X=1 TO 400:NEXT
100 NEXT
110 FOR X=1 TO 500:NEXT
120 NEXT

```

Figure 6.14 Drum roll and taps program.

and two taps which you hear repeated three times. If you want to change the noise number in a loop like this, with the sound appearing to be continuous, you need to use **fairly** sharp attacks, because if the envelope takes too long, you will not hear the result as a single sound, simply as a series of sounds.

## Pitch envelope effects

You can create a rich selection of sound effects by making use of the pitch envelope along with musical notes. Normally for musical effects, the pitch envelope is used only to a small extent in creating vibrato, but when the pitch envelope is used to give large pitch changes, the effects can be really interesting. We have had some flavour of this already when using the pitch envelope with noise. Take, for example, the warning note in Figure 6.15. This is very simply achieved by using a pitch envelope which specifies 200 steps of pitch change -5 (upwards) and a delay of 2 units. When this is applied to a sound that starts at note 500, the result is a sound that changes rapidly to higher pitch. You have to choose your sound duration number carefully here, otherwise you will find odd effects caused by the pitch control going off scale after the pitch envelope has finished. In

```

10 ENT 1,200,-5,2
20 FOR N%=1 TO 20
30 SOUND 2,500,100,7,0,1
40 NEXT

```

Figure 6.15 A warning note which is created with a pitch envelope.

```

10 ENT 1,10,-1,1
20 FOR N%=1 TO 20
30 SOUND 2,20,10,7,0,1
40 NEXT

```

**Figure 6.16 A very piercing warning note which is obtained using the low values of tone period.**

this example, the whole sound instruction has been put into a loop so that it repeats twenty times. This is a good one to signal urgent evacuation of the working area when your reactor is about to blow up. Incidentally, on past record, the oxygen/carbon reactor (usually known as a coal-fired boiler) is a darn sight more likely to blow up than the nuclear variety!

You can make really earsplitting warning notes of this kind if you use the higher pitches. Figure 6.16 is an example of the sort of thing that really knocks the wax out of your ears. The starting note this time is of pitch number 20, a very high note, and the ENT specifies 10 steps of -1 change, pause 1. This is a fast change of pitch, and when the whole lot is put into a loop, the results cannot be ignored! If you add to the SOUND instruction a noise number, around 20, at the end, you get back to another of these helicopter-blade sounds. This is just another example of how a comparatively small change in an instruction can make a big difference to what you hear. Another point to remember is that you often don't need to program any loops if you want to repeat an envelope. By using a negative envelope number in the ENT line (but **not** in the SOUND instruction), the pitch envelope will be repeated for as long as the duration of the note specified in the SOUND line. This can be very useful if you want notes that wail up and down in pitch, as Figure 6.17 demonstrates. This uses a pitch envelope which specifies a change of 50 steps down, then 100 up, then 50 down again. When this is used in conjunction with the negative envelope number, the result is a wailing that can be used for all kinds of siren effects. You can also do some tricks with this type of envelope, by making the sections non-symmetrical. By that I mean that the pitch at

```

10 ENT -1,50,2,2,100,-2,2,50,2,2
20 SOUND 2,200,5000,7,0,1

```

**Figure 6.17 A wailing note which is created using a repeating pitch envelope.**

```

10 ENT -1,50,-2,1,90,2,1,50,-2,1
20 SOUND 2,200,1000,7,0,1

```

**Figure 6.18** A modification to the wailing pitch envelope which makes the pitch pattern change.

```

10 ENT -1,50,-2,2,50,0,2,50,2,2
20 SOUND 2,200,1000,7,0,1

```

**Figure 6.19** A wailing pattern which has a period of steady tone.

the end of the envelope is not the same as the pitch at the beginning. In the example of Figure 6.18, the envelope forces the pitch to rise by 50 steps, but it then descends by only 90 steps and rises another 50. This leaves the pitch ten steps higher at the end of the envelope than it was at the beginning, and the effect is rather interesting — it's a good one for the '1, 2, 3, . . . and go' type of sound-effect. Remember also that you can write pitch envelopes that have portions of steady pitch. This is done by a section in which the pitch change number is zero, as Figure 6.19 shows. This gives a wailing sound in which the high note is maintained longer than the low note.

The next obvious step is to combine amplitude and pitch envelopes with musical notes. With the sort of choice that you have of amplitude envelopes, added to the choice of pitch envelopes, this offers you a lifetime with never a dull moment. Just as a starter, listen to the result of Figure 6.20. This uses hardware envelope 9 with a long duration, and a pitch envelope that descends and then rises again. Now if the length of the note is carefully trimmed so that the pitch variation has **just** finished as the note fades away, you get this rather splendid effect. It's ideal for the rebound of the unfortunate Tom from Jerry's stretched rope! Try it with different starting pitches, too. You'll find that the higher pitches give more useful effects than the lower ones, because a pitch change of one unit has much less effect on the lower notes. Consider also what happens when you have a pitch cycle combined with one of the repetitive hardware envelopes.

```

10 ENV 1,=9,6000
20 ENT 1,50,1,1,50,-1,2
30 SOUND 2,200,0,0,1,1

```

**Figure 6.20** A splendid 'rebound' sound effect.



```

10 ENV 1,=8,200
20 ENT -1,20,1,1,40,-1,1,20,1,1
30 SOUND 2,200,0,0,1,1

```

**Figure 6.21 The weird effect of combining a repetitive hardware amplitude envelope along with a pitch envelope.**

```

10 ENV 1,=10,1000
20 ENT -1,30,-1,1,5,6,1
30 SOUND 2,200,0,0,1,1

```

**Figure 6.22 The 'flying insect' effect.**

Figure 6.21 illustrates the sort of thing that you can get. This sound can be greatly changed by using lower values of the hardware envelope time. Try, for example, using a figure of 30 for the period of the hardware envelope. Lower numbers will give even more weird sounds which are a mixture of steady tone and fluctuating note. Another weird combination consists of a pitch envelope which changes slowly in one direction and then fast in the other, combined with a repeating amplitude envelope. This is illustrated in Figure 6.22. On this one, try also making the amplitude envelope duration number 10000 instead of 1000. For the noise of a demented bee, try a duration number in ENV of 1000, and a note number in SOUND of 800. Once again, you can get as many sound effects as you care to try out. How about an American police car speeding towards you? This is the sound of Figure 6.23. It's been achieved by using hardware envelope 13, which ramps up and then keeps maximum volume, along with a pitch envelope which changes up and down, but gets lower (the car is slowing down and stopping near you). As a variation on the same kind of theme, you can use a repetitive envelope with a short time period, and with greater pitch change to give the 'spiralling out of control' kind of sound that is illustrated in Figure 6.24.

Now for something completely different. More versatile sounds can be obtained, as you might expect, if you make use of software relative amplitude envelopes. Using an amplitude envelope

```

10 ENV 1,=13,12000
20 ENT -1,28,-1,1,5,6,1
30 SOUND 2,80,0,0,1,1

```

**Figure 6.23 The American police car siren sound.**

```

10 ENV 1,=14,100
20 ENT -1,26,-1,1,5,6,1
30 SOUND 2,60,1000,0,1,1

```

**Figure 6.24 Von Richthoven out of control — a good one for aerial combat!**

```

10 FOR N%=1 TO 10
20 ENV 1,1,15,1,15,-1,3
30 ENT 1,10,-1,2
40 SOUND 2,1000,0,0,1,1
50 NEXT

```

**Figure 6.25 The basic 'boing' sound — you can do a lot with this one!**

which has fast attack and slower decay, along with a rising pitch envelope gives the traditional 'boing' sound which is well illustrated in Figure 6.25. At lower pitches, this is good for 'giant's footsteps' (try it with an echo, too), and at higher pitches gives quite a reasonable plucked string note. This is only because the time of the amplitude envelope does not give time for more than a fraction of the pitch variation, however. You always have to remember how these envelopes are linked, with the amplitude envelope deciding how long the note plays, irrespective of whether or not the pitch envelope has finished. If you find that a set of envelopes does not do what you expect, it's often a good idea to look at the total times of the envelopes.

Remember that if you have a slow changing amplitude envelope and a fast pitch envelope, all of the pitch changes could be over before the amplitude is loud enough to hear. If you get it the other way round, the note could be too short for you to notice any pitch change. As always, this is something that you have to work at, but remember that you can always make the pitch envelope repeat so as to fill in time in the amplitude envelope.

You can get a nice set of twittering sounds if you use high notes, with fast attack and release, and also fast changes of pitch. Once again, you have to match the times reasonably carefully, and Figure 6.26 uses 41 time units in its amplitude envelope with 20 units in its pitch envelope. If you don't want this one to end

```

10 ENV 1,1,15,1,5,0,5,1,-15,1
20 ENT 1,1,5,5,1,-5,5,1,5,5,1,-5,5
30 SOUND 2,20,0,0,1,1

```

**Figure 6.26 A twittering sound, of alien birds perhaps?**

```
10 ENV 1,1,15,1,6,0,6,15,-1,4
20 ENT 1,4,0,4,10,-1,1,10,0,2
30 SOUND 2,800,0,0,1,1
```

**Figure 6.27 A bell note — this pattern can be used with a lot of different tone periods.**

```
10 ENV 1,3,5,1,10,0,5,5,-3,1
20 ENT 1,30,-2,5
30 SOUND 2,3000,0,0,1,1,1
```

**Figure 6.28 The rattlesnake — file it away with the gunshots.**

with the steady note, then just make the ENT number negative in the usual way. As always, you can play with the pitch number in the SOUND line to discover a huge range of sounds for different purposes. Finally in this section, try some bells. A bell note needs sharp attack and long sustain and release sections in its amplitude envelope, with some pitch change just at the start of the note. Figure 6.27 illustrates this type of thing. The sound, oddly enough, is rather more convincing with the small built-in loud-speaker than it is with stereo headphones or a hi-fi setup. This adds yet another dimension to the complications of sound programs!

## The whole caboodle!

You can now consider what can be done if you use amplitude envelopes, pitch envelopes, tones and noise all in one sound. It would be unreasonable to try to run through all of the possibilities that this set allows you, and all that we can do here is to consider the main types of sound that can be obtained. If you use very low notes, the effect is of bursts of noise, and the pitch envelope does not have much real effect. This is illustrated in Figure 6.28, which is a good one for a rattlesnake effect. File this one near to your gunshots for that Wild West adventure you always wanted to write. If you alter the ENT line so as to read:

```
ENT 1,200,-50,1
```

you will get noticeable pitch changes that remind me, at least, more of a snare-drum. On the other hand, the combination of whistle and rattle in Figure 6.29 reminds me of nothing I have ever heard at all! Perhaps you could use it for the sound of the

```

10 ENV 1,3,5,1,10,0,5,5,-3,1
20 ENT 1,200,-1,1
30 SOUND 2,30,0,0,1,1,1

```

**Figure 6.29 A combination of whistle and rattle.**

```

10 ENV 1,15,1,1,10,0,10,1,-15,2
20 ENT -1,1,5,1,2,-5,1,1,5,1
30 SOUND 2,30,0,0,1,1,1

```

**Figure 6.30 Sound effect for arrival of aliens.**

Venusian Swamp-swallow. With more rapid pitch changes, a high pitch note, and a fast attack long sustain amplitude envelope, you get the sound of alien monsters approaching in Figure 6.30. Altering the pitch of the musical note and the pitch of the noise will both change this effect considerably.

You can get a lot of mileage, in particular, with really long envelopes. A particularly good example is the envelope with a very slow attack, so that the sound volume increases over a long period. There are limits to this, because with only 16 steps of volume (0 to 15) you will get a rather uneven effect if you stretch out the envelope too much. As a 'gradual approach' type of effect, however, it can be very impressive. Try the one in Figure 6.31 to start with. Making the tone number 200 in the SOUND line is also quite useful, giving a mysterious ringing with the noise. You can then try giving it a bit more pitch wobble by making the change of pitch numbers 10 and -10. Different noise numbers can also change the pattern of the sound quite considerably. Just to illustrate how much a sound depends on its amplitude envelope, though, look at the example in Figure 6.32. The amplitude envelope has been changed to one with fast attack and slow decay, but the effect now is of a gunshot with a long echo!

```

10 ENV 1,15,1,100
20 ENT -1,2,2,2,4,-2,2,2,2,2
30 SOUND 2,50,0,0,1,1,1

```

**Figure 6.31 A good sound for gradual approach of something sinister.**

```

10 ENV 1,1,15,1,15,-1,10
20 ENT -1,2,10,2,4,-10,2,2,10,2
30 SOUND 2,200,0,0,1,1,31

```

**Figure 6.32 Gunshot with long echo.**

The sound is entirely different if you use a noise number of 1 in place of 31. Try also using a large note pitch number, around 4000, and once again, the type of sound changes completely. This can be very frustrating if you are looking for a specific sound, but fascinating if you just want to explore. The trouble with exploring, however, is that you often forget just what arrangement gave you some effect. If you want to be really methodical, number your sound programs, and keep a cassette which has just the sounds recorded on it, with your voice announcing the numbers. You'll find out how to record sounds in Appendix C.

This brings us to the end of this section, and in the next (and last) Chapter, we'll be looking at some other sound instructions which are not quite so often used, and also, briefly, at sound for the machine code programmer. There is no particular advantage in turning to machine code for sound effects unless you are writing a complete program in machine code, and for this reason, I shall not go into a lot of detail about the use of machine code. The remaining BASIC commands that concern the sound queue, however, are more likely to be of interest to the programmer of musical sounds, particularly when the sounds are used as part of another program of text or graphics displays.



# 7

## Loose ends

By this stage, you have seen all of the techniques that are needed to create both music and sound effects, and all that we shall do in this last chapter is to tie up some of the loose ends of topics that have been introduced in one way or another earlier. We shall start with the sound queues, because this feature probably causes more frustration than any other. As you have seen already, failure to control the sound queues correctly will make it impossible to obtain synchronised melody and accompaniment, and can wreck stereo effects. The principle is illustrated very clearly in the CPC464 manual, but it's worth repeating the points. When you program SOUND instructions, the machine will read in up to five notes **from each channel** into the sound queues. These notes will be played in order in each channel, obeying the pitch, duration, volume and other controlling numbers in the SOUND instructions. Because all of the channel queues fill up, though, the notes which are at the head of the queue in each channel will normally always be played together, whether you want them to be or not. We have seen already how this problem can be overcome by using the synchronising numbers.

One action of the queues that we have not looked at is the hold-and-release facility. If you add the number 64 to a channel number, the effect is to hold that channel inactive until the RELEASE instruction occurs in a program. If you hold the channel inactive, of course, then because the SOUND instruction cannot be obeyed, the computer will hang up its sound chip, waiting for you. The RELEASE instruction has to be followed by a channel number, which can be any number from 1 to 7 depending on whether you are releasing one channel or a number of channels — as

```

10 FOR N%=1 TO 8
20 READ F%
30 SOUND 65,F%,50,7
40 PRINT"Press a key to hear a note"
50 K$=INKEY$: IF K$="" THEN 50
60 RELEASE 1
70 FOR J=1 TO 500:NEXT
80 NEXT
90 DATA 239,213,190,179,159,142,127,119

```

**Figure 7.1 Using the RELEASE instruction to remove the hold on a channel.**

usual you add up the numbers of the channels that you want to control. Figure 7.1 shows RELEASE in action with a program which plays one note each time a key is pressed. This could be a good starter for a 'name that tune' type of game. The SOUND instruction uses the channel code of 65, meaning the hold code of 64, plus the channel number 1. This will cause the machine to hang up on the SOUND instruction until a RELEASE is issued. By 'hang-up', I don't mean that it will stop operating, because it obviously reaches line 50. The SOUND instruction cannot be played, however, and other SOUND instructions cannot be read until the RELEASE has been executed. This is done following the 'Press any key' step. The RELEASE number is followed by 1, because we are using channel 1 — you do **not** use the number 65 which was put into the SOUND instruction. It's sometimes useful to use RELEASE 7 whether you need it or not — this releases all channels, and it can save having to remember which channel you had put the hold on. If you specify a release on a channel which has not been held, there is no effect of any kind. You can, of course, hold a note in one channel while notes are playing in the other channels.

If the sound queue consists of only up to four notes, a note that is held at the start of the queue does not cause any hang-up **except in the** execution of sound. This allows notes to be queued right at the start of a program, providing that no more than 4 are queued. Try, for example, Figure 7.2 in which four notes are read into a sound queue. Because of the position of the NEXT in this program, all of the notes have been read before the 'Press any key' message appears. When you press a key, as before, a note



```

10 FOR N%=1 TO 4
20 READ P%
30 SOUND 65,P%,50,7
40 NEXT
50 PRINT"Press a key to hear a note"
60 K$=INKEY$:IF K$=""THEN 60
70 RELEASE 1
80 FOR J=1 TO 500:NEXT
90 GOTO 50
100 DATA 239,213,190,179,159,142,127,119

```

**Figure 7.2 Using RELEASE with a short sound queue.**

plays. In this example, however, there's nothing that can detect when the program should end. You play your four notes, and after that, pressing a key has no effect, and you have to use ESC to get out of the loop. Now if you make the first line read N% =1 TO 5 instead of 1 TO 4, then the program hangs up **completely**. The sound queue will not accept another note while there is a hold on the first one, and so the FOR. . .NEXT loop cannot proceed. This is not quite sufficiently stressed in the manual.

You can test the state of the sound queue with the instruction SQ (short for Sound Queue). SQ is a number which is coded as shown in Figure 7.3. When there is a hold on a note, then SQ will take a value which is 64 + the number of spaces in the sound queue. By spaces, I mean places that are not occupied. The

To analyse the SQ number, proceed as follows:

1. Use the computer in direct mode to get the binary code for SQ. You can do this with PRINT BIN\$(SQ,8). The result will contain eight digits, 0 or 1.
2. Analyse the positions of the 1's in the binary number as shown in the table below. A zero means no action.

1	1	1	1	1	1	1	1
Channel playing	Hold	Sync. to C	Sync. to B	Sync. to A	4 free spaces	2 free spaces	1 free space

For example, the number 00101011 would mean no channel playing, no hold, sync with C and A, and three free spaces in the sound queue.

**Figure 7.3 Making use of SQ.**

place numbers range from 0 to 4, so that the number  $64 + 4 = 68$  would mean that the sound queue was completely empty. To see this in action, put a new line:

```
35 PRINT SQ(1)
```

into the program of Figure 7.2, and run again. This time, you will see the numbers:

```
67  
66  
65  
64
```

appear before the 'Press any key' notice. This shows the state of the sound queue after each SOUND instruction. The number 67 is  $64 + 3$ , meaning that there are three blank spaces in the queue. The other numbers indicate that the spaces are being filled, and the final number 64 shows that there are no blank spaces remaining. This analysis of the sound queue can be used to make the program end correctly. Figure 7.4 illustrates this with the test in line 90 checking the SQ(1) number. If this is more than 64, then there are still notes in the queue, and the program loops back. A similar test could have been made using a WHILE...WEND loop. The SQ number can also show the existence of synchronising commands, or a channel which is currently playing. The SQ test is therefore used by sound programmers in rather the same way as the TEST instruction is used for graphics. There is one type of SQ instruction, however, which is particularly useful and which does not correspond to any use of TEST in graphics.

```
10 FOR N%=1 TO 4  
20 READ P%  
30 SOUND 65,P%,50,7  
40 NEXT  
50 PRINT"Press a key to hear a note"  
60 K%=INKEY$: IF K%="" THEN 60  
70 RELEASE 1  
80 FOR J=1 TO 500:NEXT  
90 IF SQ(1)>64 THEN 50  
100 DATA 239,213,190,179,159,142,127,119
```

**Figure 7.4 Using SQ to terminate the program correctly.**

```

10 FOR N%=1 TO 4
20 READ P%
30 SOUND 65,P%,50,7
40 NEXT
50 PRINT"Press a key to hear a note"
60 K$=INKEY$: IF K$="" THEN 60
70 RELEASE 1
80 FOR J=1 TO 500:NEXT
100 ON SQ(1) GOSUB 130: IF SQ(1)>64 THEN
50
110 END
120 DATA 239,213,190,179,159,142,127,119
130 PRINT"How did you like that one?"
140 RETURN

```

Figure 7.5 A simple example of ON SQ GOSUB in action.

## ON SQ GOSUB

The instruction, ON SQ GOSUB allows a subroutine to be run whenever a note finishes playing, whether there are other notes in the queue or not. As before, SQ has to be followed by a channel number placed within brackets. The subroutine can be of any type, and in the example of Figure 7.5 the subroutine simply prints a phrase. The point of using ON SQ(1) GOSUB like this is that the timing of the end of the note is exact. It is often possible to make messages appear at the end of a note, and there have been several examples in this book. Methods that are based on time delays, however, require quite a lot of cut-and-dry programming, which is completely obviated if the ON SQ GOSUB type of instruction is used. This can be particularly useful if you want to program messages or graphics immediately following a sound effect that uses an envelope. The ON SQ GOSUB instruction works only when it has been executed. In other words, carrying out an ON SQ GOSUB will detect the end only of the first note that plays in that channel. If you want to use the instruction again, you need to put the ON SQ GOSUB into a loop, as was done in the example. You should not put the ON SQ GOSUB instruction in with any loop that is reading in sound parameters which are being held, because unless a note is actually sounding, the SQ routine will operate.

```

10 ON ERROR GOTO 150
20 GOSUB 100
30 PRINT "Press a key to hear a note"
40 K$=INKEY$: IF K$="" THEN 40
50 RELEASE 1
60 FOR J=1 TO 500:NEXT
70 IF SQ(1)>64 THEN 30 ELSE GOSUB 100:GO
TO 30
80 END
90 DATA 239,213,190,179,159,142,127,119
100 FOR N%=1 TO 4
110 READ P%
120 SOUND 65,P%,50,7
130 NEXT
140 RETURN
150 END

```

**Figure 7.6 A more elaborate note reading routine which avoids over-filling the queue.**

What do you do, then, if your 'name that tune' routine needs more than four notes? Figure 7.6 shows a way out. In this example, the notes are read in by a subroutine, and in line 70, the test of SQ will call the subroutine if the sound queue becomes empty. Doing this would normally cause an error message (Out of DATA) when there are no more notes to read, but this can be avoided by using the ON ERROR GOTO line which makes the program end when an error has been detected. When you try this program, omit the ON ERROR GOTO line at first until you are sure that you have no mistypings. The reason is that ON ERROR GOTO used in this way does not discriminate one error from another, and a syntax error will also cause the program to end with no error report. In a genuine program, you would want to use ERR to trap the Out of DATA error only, and keep the normal error trapping routines for anything else.

## **Machine-code SOUND**

In the past, some computers permitted the sound routines to be programmed only by machine code. If you had no interest in machine code, too bad, you couldn't use sound. The CPC464 and CPC664 family allow you full control over the sound by using BASIC keywords, and there is never any need to use machine

code to achieve any effect. The only possible reason for using machine code for sound instructions, then, is because you are writing a program in machine code and you need sound in the program. If you have no interest in machine code, then what follows is not for your eyes. If you want to learn about machine code on the CPC464, then there are several excellent books to guide you (modestly naming no names). What follows, then, is purely for the machine-code buff who knows what it's all about with no detailed explanations.

To start with, you will need to have the excellent Amstrad tome called the Complete Firmware Specification, coded SOFT 158 for the CPC464 — I don't know the corresponding number for the CPC664 at the time of writing. This contains details of all the ROM routines that you will need to call in the course of sound programming, and shows which registers have to be used, and which saved on the stack to avoid corruption. Without this book, you are really working in the dark, and Amstrad are to be congratulated on making this information available. Computer manufacturers who keep this sort of thing a secret, or who wait until someone else provides it, tend to go out of business these days!

The programmable sound chip is the General Instruments AY-3-8912. This is one of a family of fairly similar chips, one of which (AY-3-8910) is used in the MSX computers, and earlier varieties of which were in the Colour Genie, Spectravideo, Dragon and other machines. If you have programmed sound in machine code for any of these machines, you will find it fairly easy to deal with the system of the CPC464. There are complications, however. In the CPC464, the sound chip is accessed through the port, and if you try to address the port directly you can run into considerable difficulties because of the use of the port also for the keyboard. If you are absolutely determined to get direct access, then you should study the code in the ROM of the CPC464 from address #0826 onwards. The better and much more trouble-free way of gaining access to the registers of the PSG is through the call to #BD34 (which calls to #0826). Before calling, you must place the register number into the accumulator, and load register C with the data that is to be sent. The routine corrupts

AF and BC, so that these registers should be put on to the stack if you want to use their contents later. In fact, it's better to place these registers on the stack in any case, because corrupting these registers can often make it impossible to return correctly to BASIC.

## The PSG registers

The PSG makes use of 14 registers, labelled R0-R13. Not all of these registers are eight-bit, and in some the numbers that you use are bit-significant, meaning that different actions can be programmed by using different bits in the register. The general arrangement is illustrated in Figure 7.7. Registers R0-R5 control the tone periods of the three channels. Of these six registers, the even numbered ones are eight-bit registers which contain a 'fine-tune' bit. This is used to correct the number so that the note will be in reasonably exact pitch. The coarse tune registers have odd numbers, and are four-bit only. In other words, you have a range of 15 selection numbers for your range of notes, with 256 adjustment numbers to get the notes in each range. This pair of numbers corresponds to the number range of 0 to 4095 which you can use as the tone number in BASIC. For a lot

Register	Bits							
	7	6	5	4	3	2	1	0
R0 Tone A fine	All eight bits used							
R1 Tone A coarse	Bits 3 — 0 only							
R2 Tone B fine	All eight bits used							
R3 Tone B coarse	Bits 3 — 0 only							
R4 Tone C fine	All eight bits used							
R5 Tone C coarse	Bits 3 — 0 only							
R6 Noise period	Bits 4 — 0 only							
R7 Enable	IN/OUT		Noise		Tone			
	C	B	A	C	B	A		
R8 A amplitude	Bits 4 — 0 only							
R9 B amplitude	Bits 4 — 0 only							
R10 C amplitude	Bits 4 — 0 only							
R11 Envelope fine	All eight bits							
R12 Envelope coarse	All eight bits							
R13 Envelope shape	Bits 3 — 0 only							

Figure 7.7 The registers of the PSG.

of sound effects in machine code, you would use either the coarse register or ignore tones and make use of noise.

Register R6 is the noise period register. This is a 5-bit register, hence the range of 0 to 31 in BASIC for the noise period. The number which is put into this register picks a predominant frequency, and the noise signal makes use of 'random' numbers which are centred round this tone. Ignoring R7 for the moment, registers R8-R10 are responsible for amplitude control. Using three bits of these registers allows the normal range of 0 to 7 for amplitude, but if the fifth bit is set, then four bits (range 0 to 15) can be used for amplitude control by an envelope. Registers R11 and R12 will then control the envelope period. These are both eight-bit registers, with R12 containing the more significant bit so that the range of numbers is 0 to 65535. R13 then holds the envelope shape pattern, and the numbers in this register correspond to the hardware envelope numbers that we have looked at already. When you program directly, only these envelope patterns are available, and if you want to use software envelopes, you must program them for yourself or make use of the calls to the ROM which will carry out this action.

## The enable register

Register R7 is an enable register whose contents are bit significant. Bits 0, 1 and 2 are used to control tones, and the bits are used inverted. In other words, a 1 in a bit position suppresses a tone in the channel, a 0 produces tone. Since the bit positions correspond to the channels, then the number binary 111 will shut off all of the tone channels, and 000 will activate all of them. Bits 3, 4 and 5 similarly control the noise output to the three channels, using the same convention that a 0 permits sound and a 1 suppresses it. Bits 6 and 7 are used to control input and output, and the normal setting is bit 7 set, bit 6 reset. Figure 7.8 shows the numbers that should be placed in this register in order to switch tones and noise into the various channels. The correct programming of this register is essential if you want to program the chip directly, and an illustration will probably be helpful at this point. The assembly language program is shown in Figure

Channels	Tone number	Noise number
Open		
A, B & C	0	0
B & C	1	8
A & C	2	16
C only	3	24
A & B	4	32
B only	5	40
A only	6	48
None	7	56

Add 128 to the sum of these control numbers. For example, if you want to use A & C for tone, and B for noise, then use  $2 + 40 + 128 = 170$ .

**Figure 7.8 How register R7 is used.**

7.9. The register pairs AF and BC are pushed on to the stack. This may not always be necessary, but I found that my assembler program (the Zen assembler) was corrupted if I did not preserve these registers. The A register is then loaded with 7, to operate the channel select system, and the C register with 183. This latter number is made up from 7 (no tones selected) plus 48 (noise in channel A) plus the statutory 128 (selecting **out**). The forwarding address of OBD34H is then called in order to place the byte into the register. Channel A loudness is then set to its maximum by using the number 15 in register 8, and the next set of steps comprise a loop. The aim of the loop is to put numbers, starting at 31 and cycling down to zero, into the noise frequency register, R6. This is done by loading B with 31 and pushing the BC pair on the stack. The A register is then loaded with 6, and the C register from B to get the noise number. After calling the access address of OBD34H, a delay is called so as to prolong the sound. This is very important in a machine code program, because without suitable delays, the sound is finished so soon after it has started that you hear only a click! In this case, the delay subroutine loads the maximum two-byte number of FFFFH into the HL pair, and counts this down. This gives a rather short delay, and for many sound effects, it would be better to use a longer delay, with the DE registers as well as the HL registers. Using, for example, the number FFFFH in HL and numbers in



```

    ORG 07000H
    LOAD $
    PUSH AF
    PUSH BC
    LD A,7
    LD C,183
    CALL 0BD34H
    LD A,8
    LD C,15
    CALL 0BD34H
    LD B,31
    PUSH BC
    LD A,6
    LD C,B
    CALL 0BD34H
    CALL DELY
    POP BC
    LDIR
    LD A,8
    LD C,0
    CALL 0BD34H
    POP BC
    POP AF
    RET
DELY: LD HL,0FFFFH
LOOP: DEC HL
      LD A,L
      OR H
      JR NZ,LOOP
      RET
      END

```

**Figure 7.9 A simple assembly language program which programs the PSG directly.**

the range 1 to 15 loaded into DE, you could program different delays by loading DE before calling the routine. After the delay has ended, BC is popped to restore the count number in B, and the LDIR completes the loop. The volume register is then zero'd and the program returns. The use of RET here assumes that you are returning either to BASIC or to another machine code routine.

This example illustrates the techniques that are needed if you are intending to program the sound chip directly. I must em-

phasise that you really don't need to, because the ROM routines for all the standard operations are available to you, but if you feel that direct programming is necessary, then Figure 7.9 shows how. Whatever the complexity of the program, the general methods of loading A and C, followed by the call to BD34H remain the same. A good guide to suitable programs can be obtained in books that deal with the sound system of the MSX machines. The MSX machines use an alternative sound command **SOUND**, which is followed by a PSG register number and a data number. Thus **SOUND 7,190** means that PSG register 7 is to be loaded with 190, and in assembly language on the CPC464 this would be programmed as:

```
LD A,7
LD C,190
CALL 0BD34H
```

as we have seen. You have therefore a rich source of potential sound-effect programs available to you!

## The sound routines

If you do not program the PSG directly, you can make use of the ROM routines of the CPC464 so that the same range of actions that you have in BASIC becomes available to you also in machine code. The snag here is that a considerable amount of setting up is required in the form of data blocks. This means that a set of numbers such as you might use in specifying an amplitude envelope is stored in the RAM, and the starting address has to be loaded into HL before calling the routine that will deal with the numbers. Figure 7.10 shows an example of this type of thing, with an amplitude envelope described, and a note put into the sound queue. Once again, this illustrates how the routines are used, and you can take it from here for yourself once you know from using the BASIC instructions how the routines are used. In the example, the registers AF and BC have been pushed on to the stack as a precaution, and the call to BCA7H resets the sound system. This also is purely a precaution, and it has the effect of shutting off the sound chip, clearing any sound queues, and preparing for a new sound. This routine should be used to

```

ORG 07000H
LOAD $
PUSH AF
PUSH BC
CALL 0BCA7H
LD HL,BLOK
LD A,1
CALL 0BCBCH
LD HL,QUE
CALL 0BCAAH
POP BC
POP AF
RET
BLOK: DB 2,1,15,1,10,0,10
QUE: DB 1,1,0,2,5,0,7,0,0
END

```

**Figure 7.10 An assembly language program which creates an envelope and calls a SOUND.**

The amplitude envelope data block can use up to 16 bytes, coded as shown.

Byte 0	—	Number of envelope sections.
Bytes 1-3		First section
Bytes 4-6		Second section
Bytes 7-9		Third section
Bytes 10-12		Fourth section
Bytes 13-15		Fifth section

In each section the first byte is the step count, the second is the step size and the third is the pause time.

**Figure 7.11 The amplitude envelope data block.**

ensure that nothing from a previous sound call is still lurking in the PSG registers.

The next step is LD HL,BLOK. This means that the HL pair is loaded with the address BLOK, which is the first of two addresses for data blocks. In this case, the block contains the amplitude envelope data. Figure 7.11 shows what has to be stored in a block of this type, which can use up to 16 bytes to describe a complete envelope. In this example, a fairly simple block has been used. The assembler will place bytes into memory using the DB instruction, and the bytes in this case are shown in line

14. The first byte, 2, is the number of sections. This makes it unnecessary to have any terminator for the data, but you must be careful to use only numbers 0 to 5 in this place if you are specifying a software envelope. If the number 0 is used, the envelope will consist of a steady tone which lasts for two seconds. The routines do **not** make any check for an impossible number in this position, and if you are programming in machine code, you simply have to be careful about this. The remaining bytes then contain the normal envelope numbers for step count, step size and pause time — remember if you want to use negative step size how this will be represented as a single-byte number. There is a special significance to the first byte of the block if the most significant bit is set. This means that a hardware envelope is to be selected, and in this case, the range of numbers in the rest of the byte is the usual 8 to 15 for hardware envelopes. The next two bytes then contain the period numbers, low-byte first. You would normally follow such a hardware section with a software one to give a time delay.

With the block set up for the amplitude envelope, the envelope number now has to be allocated. This is done by loading the number into register A before calling the routine. The range of permitted numbers is 1 to 15, and no envelope will be used if a number outside this range is loaded. Once the HL address and the envelope number have been loaded, the routine at BCBCB is called to set up the envelope ready for use. A similar method can then be used to set up a tone envelope. You must then put the note into the sound queue, by using the machine code equivalent of the SOUND statement. This is done by once again loading HL with the address for another block of data, and calling the routine at BCAAH. In this example, the address QUE contains the sound data, and Figure 7.12 shows how the bytes are used. You need to be careful here, because it's easy to assume that the numbers will be put in using the same order as for the SOUND statement. As you can see, they are not, apart from the first byte. In the example, the first byte of 1 is used to specify Channel B, and the following 1 specifies amplitude envelope number 1 which has already been set up. The next zero is used because no tone envelope is prepared, and the following

Byte 0	Channel code
Byte 1	Amplitude envelope number
Byte 2	Tone envelope number
Byte 3	Tone period fine
Byte 4	Tone period coarse (4 bits only)
Byte 5	Noise period
Byte 6	Amplitude
Byte 7	Duration fine
Byte 8	Duration coarse

The channel code number in Byte 1 is coded in the same way as the normal SOUND number, including any sync., hold or flush codes. If bit 7 of byte 0 is set, then the envelope is a hardware one. The first byte then determines shape, and the following two bytes determine period.

**Figure 7.12 The sound data block.**

numbers 2,5 set the tone period. This corresponds to the number  $2 + 256 \cdot 5 = 1282$ . The next zero specifies no noise number, then 7 sets the initial amplitude. The last two bytes specify duration, and by using zero in both places, the control of duration is passed to the amplitude envelope. Calling BCAAH then sounds the note.

The amplitude envelope uses a fast rise, and then a steady sustain, and since the numbers correspond exactly in position and significance to the numbers that are used in the ENV statement, there's little here to cause bother. There is more to worry about in the sound queue list. The first byte, which is the channel select byte, can use exactly the same range of numbers as the first byte of the SOUND statement. In other words, you can set synchronisation numbers, holds and flush actions by adding the appropriate numbers to the channel codes. Any sound that is sent to more than one channel will **automatically** be synchronised with itself, something that can sometimes cause odd effects. The sound time bytes (7 and 8) use the most significant bit to indicate sign. If this bit is 0, then the number is treated as a duration number. If the msb is 1, then the remainder of the number is treated as the repeat number for the sound.

A tone envelope is set up with the call to BCBFH. This requires the A register to be loaded with its envelope number, and the

Byte 0	Number of sections in envelope
Bytes 1-3	First section
Bytes 4-6	Second section
Bytes 7-9	Third section
Bytes 10-12	Fourth section
Bytes 13-15	Fifth section

Each section consists of step count byte, step size byte and pause time byte. If the step count is on the range 240 to 255 then the section is an absolute one. Bits 3-0 of the step count number are used as the msb of tone period, and the step size number is used as the lsb.

**Figure 7.13 A tone envelope data block.**

HL pair to be loaded with an address for a block of data. Figure 7.13 shows what must be provided in this data block — once again, this corresponds exactly to the type and order of data in the ENT statement. Byte 0 is the section count, and if the msb of this byte is set, then the tone envelope will repeat for the duration of the note. Step count numbers of 240 to 255 will select an absolute envelope. The tone period is then a number which is made out of the step count number and the step size number. The low four bits of the step count become the high-byte of the tone period, and the step size is the low-byte.

There are routines which correspond to the use of SQ (CALL BCADH) and RELEASE (CALL BCB3H), and also ON SQ GOSUB (CALL BCB0H). In addition, the ROM routines include calls which can allow a closer control over some actions. One of these is SOUND HOLD, called by BCB6H which stops all sound. A sound which has been stopped by using this call can be restarted by a SOUND CONTINUE call (to BCB9H), or by the use of an entry into the sound queue or a call to sound release. Of more interest to advanced programmers are the address calls. These allow you to find out whereabouts in the memory various items are stored. The call to BCC2H, for example, obtains the address of an amplitude envelope. Before making the call, the accumulator is loaded with the envelope number, and following the call the address of the amplitude envelope data block will be in HL, with carry set, and with the length of the envelope in BC. If no envelope

of this number exists, then the carry bit is reset, and the HL register will be corrupted. The address of tone envelope data can be found in a very similar way, using a call to BCC5H.

I must emphasise that the information on machine code sound techniques in this Chapter, while being a useful introduction, does not allow you to dispense with the Firmware Specification manual. The sort of detailed information that Amstrad have provided is invaluable to the machine code programmer, and if, in particular, you want to write machine code sound utilities, you cannot proceed very far without this Manual. The machine code programmer who has not attempted to program sound previously will, however, be able to make a start with the help of the information in this Chapter, and that is exactly why I have it. Here's to you, may it all sound marvellous!





# Appendix A

## Musical Terms

Musicians are familiar with the use of Italian words in musical terms, but if you are learning music script along with CPC464 sound programming, you will probably be baffled by the words that appear written over lines of a musical score, or which are used to describe the music. This Appendix lists the more common expressions, not all of which are Italian in origin. These are written in full, although you will often see them abbreviated. I have not made any attempt to create a full list of terms, because many terms are of interest only to professional musicians, either performers or composers. Instead, I have concentrated on the terms that you are likely to meet in the course of working with music on your computer.

**Accelerando** Getting quicker.

**Accidental** A sharp, flat or natural pitch sign which is not part of a key signature (not at the start of the set of lines).

**Accompaniment** The notes that are played to form a harmony to the melody.

**Ad lib** means that you need not keep to strict time.

**Adagio** A slow speed.

**Air** A simple tune, such as a folk-song.

**Allegretto** Slower than Allegro.

**Allegro** Fast, at a running pace.

**Andante** At a walking pace.

**Animato** In a lively style, bright, animated.

**Arpeggio** A way of playing the notes of a chord singly in order rather than together.

**Bar** A division in music which emphasises the rhythm. The end of a bar is marked by a vertical line.

**Beat** The rhythm of music.

**Breve** A very long note, equal to the time of two semibreves, and hardly ever used.

**Cadence** The last phrase of a piece, usually ending on the key note.

**Canon** A way of using a melody as its own accompaniment, by starting it in one channel, and later in another. Also known as a 'round'. Well known canons include Frère Jaques, and Pachelbel's Canon.

**Cantabile** Literally means like a song. The music should be played in a flowing way, smoothly.

**Chord** A combination of notes that are all sounded together.

**Chromatic** A scale that has every interval equal to a semitone, so that there are 12 notes from a keynote to its octave note.

**Clef** The sign in a stave which distinguishes treble from bass.

**Coda** An end section of music.

**Common time (C)** Means 4/4 time, with four crochets in each bar.

**Compass** The range of notes that an instrument can produce.

**Compound time** Any time that does not use two or four beats in each bar.

**Con Brio** Played in a very energetic and dashing style.

**Counterpoint** The combination of two (or more) melodies to form pleasing chords.

**Crescendo (Cresc.)** Getting louder to a climax.

**Crochet** Unit of note duration.

**Da Capo** from the beginning again. Often written as DC.

**Diatonic** Using the notes of a major or minor scale, not chromatic.

**Diminished** Flattened by a semitone.

**Diminuendo (Dim.)** Getting softer.

**Discord** Notes which sound unpleasant together.

**Dominant** The fifth note of a scale, counting the key note as 1.

**Dot** A marking which alters a note. When placed following the note, lengthens the note by 50%. When placed above the note means staccato.

**Double bar** Two vertical lines which mark the end of a piece of music.

**Duple time** A time signature which contains either two or four beats in the bar.

**Dynamics** The range of amplitude from soft to loud.

**Expression marks** The symbols that are marked on music by the composer to indicate how the music is meant to be performed. Most conductors prefer to think that they know better.

**Figure** A short repeated piece of tune.

**Flat** Lowering of pitch by one semitone.

**Forte** Loud, usually written as **f** on the music. Louder is written as **ff**, and the loudest possible as **fff**.

**Fugue** A composition which makes use of counterpoint.

**Gavotte** An old-fashioned dance in 4/4 time which begins on the third beat in the bar.

**Giusto** correct, strict time.

**Glissando** A rapid ascent or descent of a scale, played smoothly.

**Grace note** A note added, often by the player, to make the change from one note to another more smooth. Found particularly in bagpipe music for technical reasons.

**Grave** A slow tempo, almost funereal.

**Grazioso** Played with grace, elegantly.

**Harmony** Sounding together notes that give a pleasing effect.

**Improvise** To play notes that are not written in the score, to your own fancy.

**In Alt** Means the octave above the treble clef.

**In Altissima** Two octaves above the treble clef.

**Interval** The pitch difference between two notes.

**Key** A note which lies at the start of a scale, its keynote.

**Key signature** Sharps or flats which are placed at the start of each line of a piece of music to indicate from which scale the notes are taken. If there is no key signature, the scale of C Major is used.

**Largo** Slow, about one crochet per second.

**Legato** Smoothly performed.

**Lento** Slow, not quite so slow as Largo.

**Maestoso** Majestic, stately — and slow.

**Major** A type of scale which has semitones placed between the 3rd and 4th, and 7th and 8th notes.

**Metronome** A clockwork (or electronic) machine which keeps time by means of a swinging pendulum or by flashing lights. Metronome markings on music refer to the number of crochets played per minute.

**Minim** A note which is played for the time of two crochets.

**Minor** A form of scale which has the 7th note sharpened. Some forms have the 6th note sharpened also.

**Modulate** To change from one key to another. Some singers do this unintentionally!

**Molto** means much or very, so molto maestoso is very majestic.

**Movement** A section of a large musical work, usually with a key and timing that sets it apart from the rest.

**Natural** A note which has no sharp or flat, like all the notes of the C Major scale.

**Ninth** An interval of nine notes.

**Notation** The script of music.

**Note** The unit of music, a complete musical sound.

**Octave** A set of eight notes, with the top note having the same note name as the bottom note.

**Opus** Literally 'work'. The composer's method of numbering his efforts.

**Pavane** A very ancient slow dance time.

**Piano** Soft, usually written as p in the music. Softer is indicated by pp and as soft as possible by ppp.

**Pitch** Musical term for the frequency of a sound.

**Pizzicato** The sound produced by plucking strings rather than using a bow.

**Presto** Very fast.

**Prestissimo** Very very fast!

**Quaver** Note which is sounded for half the time of a crochet.

**Rallentando** Slowing down, as often is required near the end of a tune.

**Recapitulation** A repeat of a phrase or complete passage.

**Rest** Musical silence which is timed like a note.

**Retrograde** A melody played backwards.

**Rhythm** The beat, timing, and accentuation of musical notes.

**Rubato** Lengthening one note at the expense of another, not keeping strict time.

**Scale** A set of notes which forms the building blocks of a melody.

**Score** A piece of music which consists of several parts for different instruments.

**Second** An interval of two notes in a scale.

**Semibreve** A note which is played for a time of four crochets, or two minims.

**Semitone** The smallest pitch interval in Western music.

**Seventh** An interval of 7 notes in a scale.

**Sforzando** Means forced, with a lot of effort.

**Sharp** A pitch that has been raised by one semitone.

**Simple time** A time signature that uses two or four beats in each bar.

**Sostenuto** Literally means sustained, the note is hung on to.

**Staccato** Each note sounded separately, not merged with the next.

**Staff, Stave** The set of lines on which music is written.

**Syncopation** Shifting the stress or beat of music.

**Tempo** The speed of music, which can be specified by a metronome reading.

**Third** an interval of three notes in a scale.

**Time** The division of music for purposes of rhythm.

**Tone** The standard pitch interval in a scale.

**Transcribe** To arrange music for a different instrument.

**Transpose** To arrange music into a different key.

**Tremolo** Rapid to and fro change of amplitude.

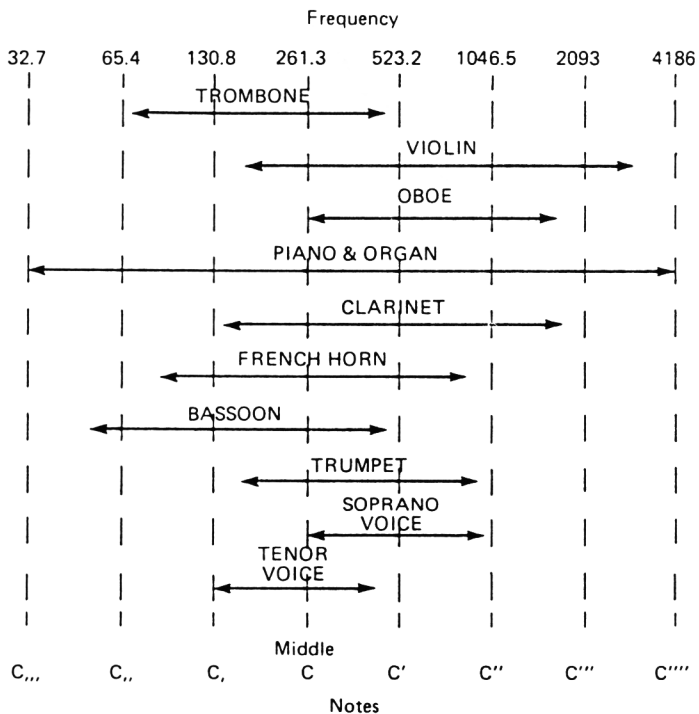
**Vibrato** Rapid to and fro change of pitch.

**Vivace** Means lively, a merry pace.

# Appendix B

## Musical Instruments

Figure B1 shows the approximate pitch ranges of orchestral instruments in terms of the conventional piano scale. For transcribing to computer program form, the best scores are for voices and instruments in the upper ranges, because the bass notes of the computer sound good only on Hi-Fi equipment. Figure B2 shows how instruments make use of written music. These instruments for which the 'Played' column reads 'As written' play the notes that are shown on the music. In order to use the same set of musical staves for all instruments, however, some instruments transpose. It would be ridiculous, for example, to show each note for the double bass with lines drawn under the bass clef, and with nothing in the treble clef. The player of the double bass therefore uses music which shows the notes written one octave higher, and he/she plays notes that are an octave lower than the notes which are printed. Similarly, the piccolo player sounds notes one octave higher than the ones in the music. You will need to bear this in mind if by any chance you are working from music for these instruments. Some instruments require different transpositions, and these are shown in the chart. These transpositions are used to allow the player to use two instruments which are in different natural keys, like the A-clarinet and the B-flat clarinet. The idea is that the music for either clarinet looks the same, and the player presses the same keys, but the notes that are produced are different. Don't use music for these instruments unless you are confident of being able to transpose it back to normal.



**Figure B1 The approximate pitch ranges for orchestral instruments.**

Instrument	Played	Clef(s)
Piano/Organ	As written	Treble & Bass
Violin	As written	Treble
Cello	As written	Bass & Treble
Double Bass	Octave lower	Bass
Clarinet (Bb)	1 Tone lower	Treble
Oboe	As written	Treble
Bb Trumpet	1 Tone lower	Treble
Trombone	As written	Bass & Treble
Piccolo	1 Octave higher	Treble

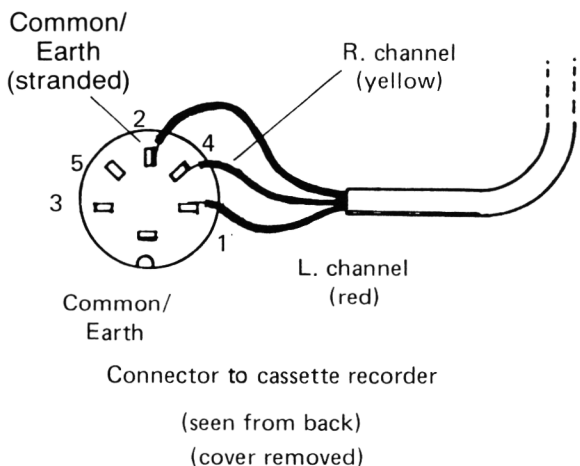
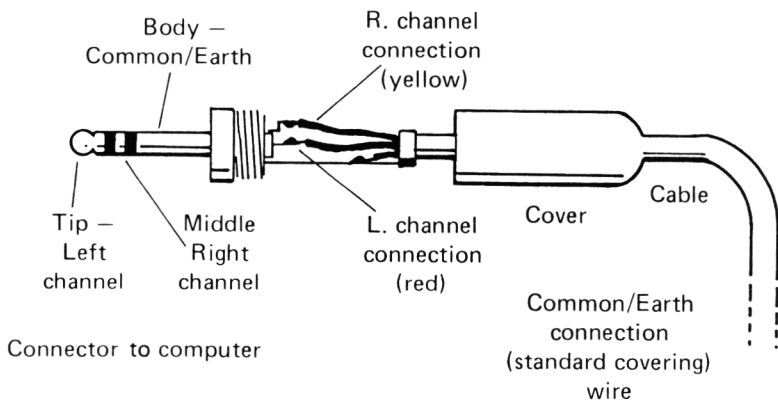
**Figure B2 How music is written for different instruments. Keep clear of music for transposing instruments.**

# Appendix C

## Connecting to other units

The signal from the output socket at the left-hand rear of the CPC464 is rather faint for most types of stereo headphones. You can obtain louder results in two ways. One is to use a small stereo amplifier between the computer and the headphones. A suitable type is the Omega 3-Watt stereo amplifier, but only if you are able to solder connections to it, and put it into a suitable box. A few manufacturers supply small loudspeakers which are complete with amplifiers that are battery powered. These are intended for use with portable stereo players, and so they are equipped with the correct connecting jack and are ideal for use with the CPC464. One useful source of such amplifiers that I have found is B.G. Audio & TV Ltd, of 8 Hatter St., Bury St. Edmunds, Suffolk. (Tel: Bury St. Edmunds 5227.)

Another possibility is to record the music with a stereo cassette recorder. If you have such a recorder as part of your Hi-Fi system, you will need a suitable connector. You may be able to persuade your local Hi-Fi store to make up a cable for you, but if you can solder neatly, or know someone who can, the diagram of Figure C1 shows what is needed. The cable is single way stereo, screened, and has two connectors. One is a 3.5 mm stereo jack plug, the other is a standard five-pin DiN plug. When the cable is made up, plug the jack into the CPC464, and the DiN plug into the cassette recorder. Switch on the recorder, insert a blank tape, wound on to a suitable starting place, and select RECORD, but do not start the tape moving. Now run the computer program, and adjust the volume controls of the recorder until the reading



**Figure C1 The connections for a cable which will allow stereo recording.**

is just hitting the peak marks on the recording level meter. A few recorders use automatic volume control, but most Hi-Fi recorders depend on correct manual setting like this. When the volume controls are correctly adjusted, type RUN, start the tape, and then press ENTER. Stop the recorder after the music has finished. Now take out the cassette, rewind it, and prepare to have your ears blasted out when you replay it on a good setup. The lowest bass notes of the CPC464 can shake my floors beautifully (with 100W per channel!).



# Index

- Absolute pitch envelope . . . 107
- Absolute software envelope . . . 93
- Accidental . . . 24
- ADSR . . . 10
- ADSR shape . . . 96
- Aero engine . . . 114
- Air . . . 1
- Alien monsters . . . 124
- Allegro . . . 29
- Amount of vibrato . . . 106
- Amplifier . . . 14
- Amplitude . . . 8, 32
- Amplitude envelope . . . 83
- Amplitude envelope block . . . 140
- Andante . . . 68
- Assembly language, sound routine . . . 138
- Assembly language example . . . 136
- Attack . . . 10
- Augmented chord . . . 58
- Bar . . . 33
- Bass stave . . . 23
- Beep . . . 38
- Bells . . . 123
- Blues scale . . . 21
- Boing sound . . . 122
- Breve . . . 27
- C-Major . . . 21
- Calculating frequency . . . 49
- Cantabile . . . 59
- Centre tone of noise . . . 112
- Chord . . . 55
- Chordfinder program . . . 81
- Chromatic scale . . . 22
- Clock pulses . . . 35
- Coarse-tune registers . . . 135
- Combining envelopes . . . 92
- Complete firmware specification . . . 133
- Composing . . . 75
- Compressing air . . . 2
- Cone of loudspeaker . . . 3
- Connecting to other units . . . 151
- Copyright . . . 43, 60
- Crescendo . . . 32
- Crochet . . . 27
- Cymbal & drum . . . 116
- Decay . . . 10, 85
- Default speed . . . 29
- Default timing . . . 44
- Demisiquaver . . . 28
- Designing envelopes . . . 93, 99
- Diminuendo . . . 32
- Direct programming . . . 135
- Disappearing sound . . . 116
- Disaster warning . . . 42
- Dotted beep . . . 38
- Dotted notes . . . 30
- Drum skin . . . 5
- Duration . . . 27
- Earphone jack . . . 14
- Earphones . . . 14, 72
- Echo effect . . . 95
- Electrical wave . . . 35
- Emphasise melody . . . 62
- Enable register . . . 135
- ENT guidelines . . . 109
- ENT planner . . . 109
- ENT statement . . . 106
- ENV statement . . . 86
- Envelope . . . 10, 71, 83
- Envelope period registers . . . 135
- Envelope planner . . . 102
- Falling noises . . . 116
- FIFO memory . . . 39
- Filters . . . 7
- Fine-tune registers . . . 134
- Flats . . . 25
- Flattened third . . . 57
- Forfe . . . 32
- Frequency . . . 7
- Frequency formula . . . 49
- Frequency range . . . 15
- Giants' footsteps . . . 122
- Gradual approach . . . 124
- Guitar string . . . 5
- Gunshot with echo . . . 123
- Gunshots in stereo . . . 113
- Hardware envelope . . . 87
- Harmonic minor . . . 21
- Harmonics . . . 9
- Harmony . . . 18, 55
- Hearing . . . 3
- Helicopter blade . . . 114
- Hertz . . . 8
- Hold & release . . . 127
- Hum . . . 15
- Human ear . . . 8
- Insect effects . . . 121
- Interval . . . 21
- Jazz . . . 21
- Jogging on gravel . . . 115
- Key signature . . . 24
- Keynote . . . 19

- Kilo . . . 9
- Language of sound . . . 7
- Loudness . . . 8
- Loudspeaker . . . 35
- Loudspeaker cone . . . 3
- Machine-code . . . 132
- Major chord . . . 55
- Major scale . . . 19
- Merging notes . . . 46
- Metronome . . . 28
- Middle C . . . 19, 23
- Minim . . . 27
- Minor chord . . . 58
- Minor scale . . . 20
- MOD effect . . . 63
- Modified wails . . . 120
- Mono . . . 12
- MSX programs . . . 23
- Music . . . 17
- Music keyboard program . . . 78
- Music program . . . 22
- Music subroutine . . . 53
- Musical phrases . . . 1
- Musical terms . . . 145
- Name to number program . . . 49
- Names of notes . . . 23, 44
- Natural minor . . . 21
- Natural noises . . . 111
- Noise . . . 6, 111
- Noise circuit . . . 38
- Noise period register . . . 135
- Noise pitch numbers . . . 112
- Note duration . . . 42
- Notefinder . . . 49
- Octave . . . 18
- ON SQ GOSUB . . . 131
- Other routines . . . 142
- Pause . . . 95
- Period number . . . 89
- Piano . . . 32
- Piano keyboard . . . 26
- Pitch envelope . . . 85, 106
- Pitch number . . . 37
- Pitch ranges of instruments . . . 149
- Pitch variation . . . 9
- Plucked string sound . . . 101
- Police car siren . . . 121
- Port . . . 133
- Programmable Sound Generator . . . 38, 134
- PSG . . . 38, 134
- PSG registers . . . 134
- Quaver . . . 28
- Raging surf sound . . . 112
- Range of tone period . . . 41
- Rarefied air . . . 3
- Rate of vibrato . . . 109
- Rattlesnake . . . 123
- Reading time and pitch . . . 48
- Rebound effect . . . 120
- Registers . . . 39
- Registers on stack . . . 134
- Relative envelope . . . 96
- Relative pitch envelope . . . 107
- Release . . . 10, 85
- RELEASE . . . 128
- Remainder of division . . . 63
- Rendezvous . . . 70
- Repeating beep . . . 38
- Rests . . . 30
- Rhythm . . . 32, 63
- ROM routines . . . 133
- Rubato . . . 64
- Scale . . . 18
- Semibreve . . . 27
- Semiquaver . . . 28
- Semitone . . . 20
- Shape of wave . . . 5
- Sharps . . . 24
- Sheet music . . . 59
- Silence between notes . . . 45
- Silent note . . . 74
- Sine wave . . . 5
- Singing note . . . 62
- Skeleton program . . . 44
- Snare-drum . . . 117
- Sound channels . . . 39, 40
- Sound effects . . . 1, 7
- SOUND instruction . . . 40
- Sound program . . . 1
- Sound queue . . . 39, 66, 127
- Sound queue block . . . 140
- Sound routines . . . 138
- Sound synthesiser . . . 6
- Sound wave . . . 3
- Space war sound . . . 116
- Spiral out of control . . . 121
- SQ use . . . 130
- Square wave . . . 6
- Staccato . . . 45
- Staff . . . 23
- Staircase shapes . . . 87
- Stave . . . 23
- Steam hammer . . . 114
- Steam loco sound . . . 112
- Steeper attacks . . . 100
- Stereo . . . 12

Stereo amplifier . . . 14, 151  
Stereo cassette recorder . . . 12  
Stereo output . . . 63  
Stereo sound . . . 71  
Stereo synchronisation . . . 73  
Stereo gunshots . . . 114  
Stressed note . . . 32  
Synchronisation with drum . . . 116  
Sustain . . . 10, 85  
Symbols for rests . . . 30  
Synchronisation . . . 65  
Synchronisation codes . . . 68  
Synchronising graphics with sound  
. . . 130  
Tempo . . . 31  
Testing SQ number . . . 129  
Tie line . . . 30  
Time of note . . . 42  
Time periods . . . 87  
Time signature . . . 28, 43  
Timing . . . 28  
Tone envelope block . . . 141  
Tone period . . . 40  
Tone period formula . . . 41  
Total decay . . . 105  
Transposing instruments . . .  
60, 149  
Treble stave . . . 23  
Tremolo . . . 85  
Tune with harmony . . . 60  
Twelve-note scale . . . 22  
Twittering . . . 122  
Two loudspeakers . . . 11  
Using SQ . . . 129  
Using five sections . . . 98  
Vibrato . . . 85, 106  
Volume number . . . 62  
Wailing . . . 119  
Walking pace . . . 28  
Walkman . . . 12  
Waltz . . . 72  
Warning note . . . 118  
Water wave . . . 3  
Wave height . . . 7  
Wave shape . . . 5  
WHILE..WEND loop . . . 48  
Whistle . . . 5  
White noise . . . 111  
Yankee Doodle . . . 43





The Amstrad CPC464 contains one of the most versatile sound generating chips that can be found in any of today's computers.

This guide has been designed for you, the Amstrad user, to extract the maximum benefit from the extraordinary sound capabilities of the CPC464.

With the Amstrad's very powerful BASIC, you won't even need to know about machine language to create music and sound effects to delight you.

Beginning with the first principles of sound you will be shown, step-by-step, how to create melodies, rhythm and fantastic sound effects to liven up your own programs.

Along with the aid of the many program examples you can master the Amstrad sound chip and tap your own creativity.

**£7.95**



Melbourne  
House  
Publishers

ISBN 0-86161-192-6



**WORLDWIDE JOURNALISM & COMMUNICATIONS**

**YOUR COURSE ADVISOR**

**STRAVANTIA SINCLAIR**





Document numérisé avec amour par

# AMSTRAD

CPC 

# MÉMOIRE ÉCRITE



<https://acpc.me/>