

Bells and Whistles on the Amstrad CPC 464

SHIVA's
friendly
micro
series

Jeremy Vine



**Bells
and Whistles
on the
Amstrad**

DEDICATION

For Michael and Rosalind

Bells and Whistles on the Amstrad

Jeremy Vine



Shiva Publishing Limited

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

© Jeremy Vine, 1984

ISBN 1 85014 063 4

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

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

An interface was used to produce this book from a microcomputer disc, which ensures direct reproduction of error-free program listings.

Typeset by Wordsmith Graphics Limited
Printed by Devon Print Group, Exeter

Contents

	Preface	7
1	What Can I Do?	9
3	A Sound Statement	11
3	Sounds Abound!	15
4	A Musical Interlude	25
5	The Sound of Music	33
6	The Volume Envelope	37
7	The Tone Envelope	43
8	Whiz Bang Wallop!	49
9	Music Maestro Please	57
10	Dig that Beat!	63
11	Bells and Whistles	65
	Appendix A Amstrad BASIC Keyword Summary	67
	Appendix B Sound Parameters	75
	Appendix C Volume Envelope Parameters	77
	Appendix D Tone Envelope Parameters	79
	Appendix E Frequency, Pitch and Note Table	81
	Appendix F The Sound Chip—Technical Notes	85

'I wander'd by the brookside,
I wander'd by the Mill,—
I could not hear the brook flow,
The noisy wheel was still:
There was no burr of grasshopper,
No chirp of any bird:
But the beating of my own heart
Was all the sound I heard.'

Richard Monckton Milnes, Baron Houghton (1809–1885)

Preface

The vast appeal of home micros lies in their effects on our senses. Not only do we depend on seeing and hearing to live normal lives, we also find enjoyment in many of the things we see and hear. This is an area in which computers can excel. Arcade and home-computer games rely on their visual effects for their popularity—and the Amstrad is second to none in this. But computer effects are not merely visual. Micros can also produce unusual, varied and exciting sounds. And that is what this book is all about.

The Amstrad CPC 464 has a built-in sound chip which is capable of producing remarkable sounds. The BASIC commands to produce these sounds may on first sight seem complicated and off-putting, but I intend to show how easy it really is to produce sound on the Amstrad—you can achieve amazing results with a little thought. I have made few assumptions about the programming ability of the reader, and all programs have been written with beginners in mind. However, to get the most out of your programming in sound you should at least have a rudimentary knowledge of BASIC.

The fundamentals of sound and music are introduced, and you will discover how to produce special noise effects such as explosions, as well as the sound of different musical instruments. You can also play your Amstrad as a piano keyboard, and programs are included that will play everything from musical scales to ragtime. But the real thrill is when you are able to create your own sounds, and by the end of the book you should be doing just that.

I finish on a personal note. This is my third book, and it is with gratitude that I acknowledge the constant support and encouragement of my family and friends through the long hard weeks. A special mention goes to my friends in the magazine world, and in particular Tony Quinn of *Acorn User*, who published my first few hundred words and has helped me ever since.

About the author

Jeremy Vine was educated at William Ellis Grammar School, Highgate, London, and at the City of London Polytechnic, where he achieved a BSc in Psychology. Jeremy first became involved with computers while studying for his degree and has never since been far away from one. When Jeremy left the polytechnic, he worked freelance for *Acorn User* magazine, before joining Acorn Computers Limited. During this period, he also studied for an MSc in Neurophysiology. Now, having left Acorn, Jeremy is working full-time as a freelance writer. As well as his books for Shiva, he is a regular contributor to several computing magazines, which has brought him a consultant editorship. His spare-time interests are tennis, piano, photography and, of course, home-computing.

1 What Can I Do?

When you first bought your Amstrad, your chief reason for doing so may have been games playing, educational use, word processing or perhaps even business use. But the chances are you didn't buy the CPC 464 solely because it had a sound generator. You were probably aware that it could produce sounds, but may have thought they were limited to beeps and crashes. But having bought this book you will now realise that the Amstrad can do much more than just go 'beep'! In fact, the use of its powerful sound facilities can add an extra dimension to all its other applications.

The Amstrad is a versatile machine, and the sound facilities are every bit as versatile and outstanding as the rest of the machine. But then one couldn't expect less from a hi-fi manufacturer! After all, can you imagine Amstrad building a quiet machine?

The book follows two distinct lines of thought. Firstly it teaches you the rudiments of sound generation and music. If you don't already play a musical instrument or read music, there is an introduction to this which will give you enough basic knowledge to program music into the Amstrad.

The second major aim of the book is to acquaint you thoroughly with the commands that control the sound effects. The area of sound and music is vast, and the commands within the BASIC language have been well designed to enable the BASIC programmer to control every aspect of sound production.

So, what can you do? The answer is almost anything—music of all kinds is at your fingertips. For those who prefer noises and sound effects, the Amstrad has ample ability to mimic the sounds we hear around us every day. Examples are given in the book of telephones ringing, alarms, explosions—and whistles! These and many other effects are covered, with the aim of encouraging you to experiment and discover your own special effects.

All the program listings show the BASIC keywords in upper-case (capital) letters, but you can type all your listings in

lower case if you like, as the Amstrad will convert the keywords to upper case automatically.

And now the fun begins. Sit back, relax and switch on your Amstrad. Soon you will hear, not just the tap of the keyboard as you use the computer, but music. Your adventures in sound are about to begin!

2 A Sound Statement

Creating a sound on the Amstrad is really very easy, but to get the most out of the sound generator built into the machine you need a detailed knowledge of all the relevant sound commands. These commands are admittedly quite complicated, but they are well worth the effort.

Before we tackle the first keyword, an introduction to the fundamental principles of sound is needed. This will give you a firm footing from which to understand what is happening within each part of a sound command.

SOUND PRINCIPLES

Sound is all around us. Imagine any sound you like—a loud bang, a soft beep, or something more interesting! Whatever the sound is, it will consist of features common to all sounds. These features are the basic building blocks of sound: the amplitude, pitch and duration.

Amplitude

Amplitude is the volume level of a noise or sound. On the Amstrad this volume can be controlled by the user and is set in the range 0 to 7, from soft to loud. But merely setting the whole sound to loud or soft is not enough. It is easy to tell if a single steady note is loud or not, but within most sounds there is also variation in amplitude, although this is not always as noticeable. Now this is not as difficult as it might seem.

When a piano key is struck the sound heard has a more complicated amplitude than just loud or soft. At the beginning of the note there is an increase of volume, called the attack phase of the sound. At the end of the note there is a decrease in the

volume level, known as the decay phase. Other instruments produce their distinctive sound by building up the volume more slowly or quickly (a lesser or greater attack rate) before reaching their maximum amplitude. You need not worry about this at present, as we will cover it in more detail in Chapter 6, when we look at the ENV statement, which is responsible for altering the attack and decay rates of a given sound.

So the amplitude of a sound is not as simple as it seems at first. And there is more! The volume level is also affected by other factors. The duration of a sound can vary, and this in itself can alter the volume perceived by the listener. A sound which lasts for only a fraction of a second will seem quieter than a much longer sound. It is by altering parameters in this and other ways that we can create a range of sounds.

Pitch

Simply speaking, this is whether a sound is high or low. Without getting too technical the pitch of a note is defined as its frequency—that is, the number of sound waves per second. Sounds are all about us, but as humans we can only hear sounds within a certain range of frequencies. A classic example of a real sound that we cannot hear is a dog whistle. Because dogs can hear sounds of a higher frequency than humans, they hear the sound of the whistle even though we cannot hear it ourselves. So just because we cannot hear a sound does not mean it doesn't exist!

When we talk about pitch we are usually referring to frequency in terms of musical scales, like those of the high and low notes of a piano keyboard. Now the notes on a piano are fixed at given pitches and intervals, but on the Amstrad we have to define the pitches ourselves. Using the ENT statement, pitch can be controlled very exactly—and you can even define how it is to vary! Yet again think of a musical instrument. When a note is played, on some instruments the pitch can be altered by means of vibrato (a pulsating effect). The tone envelope (ENT) allows you to do something similar. In this way we can create a range of pitch effects covering a spectrum of sound. We will return to pitch in later chapters when we make our own sounds and explain musical effects.

Duration

The duration of a note is simply the length of time it is played. All pieces of music have rules as to the length of the notes to be

played (see Chapter 4) and the speed at which the music will be heard. It is these variations in duration that produce the different forms of rhythm we are accustomed to, from waltzes to rock and roll.

With the sound commands of the Amstrad, the length of a note can be specified or changed.

With all these facilities at our fingertips we have the basis for generating a wide range of sounds. However, the sound generator of the Amstrad has more to offer than just the ability to alter these three main variables.

Sound channels

So far we have spoken of sound as a single note or series of notes. But when we listen to a piano being played it is not a series of single notes that we hear. The pianist will usually play a number of notes at the same time, in synchronisation, to form chords and harmonies. And this facility is also available to us on the Amstrad.

The Amstrad has not just one channel on which sound can be produced, but three. The programmer can determine whether these channels are played in unison with each other or separately. These three channels are referred to as A, B and C. As we shall see later, we can not only use them to play in three synchronised parts, but also assign a different sound to each channel.

Besides generating musical notes the Amstrad can produce other sounds, namely so-called 'white noise' effects. It is these sounds which often form the basis of the 'space invader' explosion effects so commonly heard within arcade games. These sounds can also be used to create percussion effects as a background to music.

White noise is simply random noise, containing all the available frequencies mixed together. One channel can be assigned to this sound effect. Whereas you can play three different musical notes at the same time by assigning each to a separate channel, you can use only one noise at a time.

The Amstrad stores the commands to play sounds in 'queues', each channel having its own queue. Within each queue there is room for up to five different commands, of which one is active (the command being played) whilst up to four more can be waiting. You can think of the queue rather like someone serving food onto a plate. In front of the person is a stack of five plates. The one at the top of the stack is at the front of the queue, and it is this one which is being dealt with. Once the food has been placed on the plate (processing the command), the next plate in

line is served, and so on. The whole point of this is to allow the Amstrad to continue with other processing tasks while carrying out the commands in the sound queue, only returning to pick up more sound commands when the queue is exhausted.

SOUNDS COMPLICATED?

Now all of the above may be completely straightforward to you, or it may be confusing. Whichever state you feel in after reading this chapter, you will find that you are well catered for in this book. We will be coming across everything I have mentioned as we work through these ideas in more detail to uncover a wealth of material that will transform your Amstrad into a machine that can perform wonders of entertaining sound.

Now let us take a look at the first BASIC command that allows us to program the Amstrad to produce a sound.

3 Sounds Abound!

Well, I've talked about all these wonderful facilities the Amstrad has, but how do we actually start producing the sounds? The first thing to do is to turn the volume dial, on the right-hand side of your keyboard, to maximum. Throughout the book leave the volume at this setting (unless it causes rifts in the happy family home!). Now type in the following line (not forgetting to press the ENTER key at the end):

```
SOUND 1,478
```

You will hear a quick, fairly soft tone (in fact this tone is middle C, lasting for a duration of one-fifth of a second). The numbers after the keyword SOUND indicate various parameters. In all, the SOUND command can have up to seven different parameters associated with its use. If you want to see what they are at this stage, take a look at Figure 3.1.

So far we have only used two of these parameters—the only two that you have to use. They are as follows:

```
SOUND C,P
```



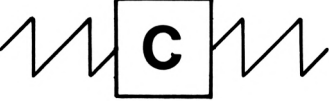
Let's look at each in more detail.

THE CHANNEL PARAMETER

The first parameter, C, indicates the channel(s) the sound(s) will be played through and its status. For the moment we will not concern ourselves with the status but with allocating a sound to a particular channel. In the case of the above example the channel number we have specified is 1 and this means that channel A will be used. In Chapter 2 I mentioned that three sound channels

existed. Now there are many different possible values that can be used for C (the range is in fact 1 to 255) but for the moment we will concentrate on accessing the three sound channels individually. Table 3.1 shows the values to be used for each sound channel.

Table 3.1 Sound channel values(1): Accessing the three sound channels.

<i>Channel</i>	<i>Value</i>
	1
	2
	3

Therefore the command `SOUND 2` would indicate that channel B should be used. Don't type in `SOUND 2`—at least two parameters are needed to produce a sound! I will come back to the other possible values later on in the chapter.

THE PITCH PARAMETER

The second parameter, P, is the tone or pitch of the sound. In the above example I used the number 478 which, as I indicated, is the pitch for middle C. How do I know that? Well, as you know, each note has its own frequency value, and in Appendix E you will find a table of pitch and frequency values and the musical notes associated with them. The parameter P accepts a number in the range 0 to 4095; the value you enter can be equivalent to a recognized musical note, such as 379 which would produce the musical note E—but it doesn't have to be! You can use any

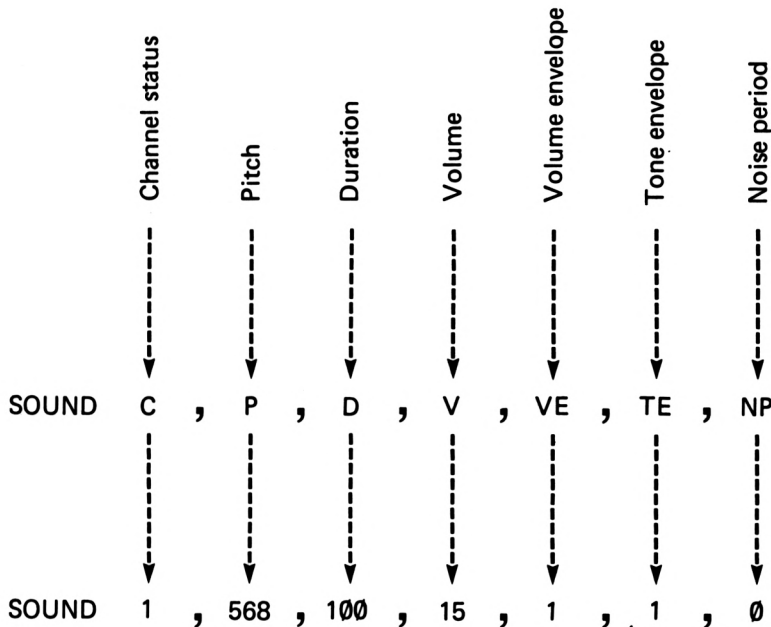


Figure 3.1 The sound parameter.

frequency within the range shown. Do note, however, that the value you enter must be the number given for the pitch and not the frequency. You can see that each frequency value given has an equivalent pitch number.

Try changing the value of P to see how the pitch changes. For example, type the following:

SOUND 1,956

SOUND 1,478

SOUND 1,60

All of these values are the note C played at different pitches. You will notice one thing which might seem odd. As the number decreases in value, the pitch increases (that is, the tone is higher). This is unexpected, but it is a feature of the Amstrad that you must remember, otherwise you may get very confused!

If you set the pitch parameter to zero ($P = 0$) then no frequency will be set. This can be of use when we create 'white noise' effects.

THE DURATION PARAMETER

Let's move on now and look at the next parameter, D, which is the duration of the sound or the length of time a note is played. Type the following line:

```
SOUND 1,568,100
```

This will play the note A on channel A for one second. The third parameter is measured in units of one-hundredth of a second, so in the example, 100 = one second. Any value that is greater than zero represents a duration of time. However, the duration parameter can also be used in two other ways.

The first is when D equals zero. This tells your Amstrad that the duration is specified and controlled by the volume envelope (ENV) associated with the sound channel. If this is not clear don't worry! I will explain this in detail when I come to the ENV command.

The other kind of value you can use for D is a negative number. This also refers to the ENV command and instructs the machine to play the relevant volume envelope a given number of times, the number being equal to the negative value specified by the programmer. For instance, the value -3 would repeat the volume envelope 3 times.

Experiment with the values in the D parameter to get an idea of how the length of a sound can be changed. Table 3.2 summarizes the values that can be used in the duration parameter.

Table 3.2 The effects of numeric value on the duration parameter.

<i>Value</i>	<i>Effect</i>
Default: 20	-
Range: -32768 to + 32767	-
> 0	Length of SOUND in 1/100ths second
= 0	Duration controlled by the volume envelope (ENV)
< 0	Number of times volume envelope is repeated

If no duration is specified, as in the first example of this chapter, then a default value of 20 is used by the Amstrad, causing the sound to last for one-twentieth of a second.

THE VOLUME PARAMETER

On to the fourth parameter, which is the volume, V. Type in Program 3.1 and run it.

Program 3.1

```
10 FOR volume = 0 TO 7
20 SOUND 1,478,25,volume
30 NEXT
```

The sound level increases each time around the loop, so that when 'volume' equals 7 the tone is at its loudest. The range of values can be between 0 and 7, where 0 equals no volume and 7 is the loudest.

If the volume envelope (ENV) is specified, then the value can be in the range 0 to 15 where the positive integer number (i.e. greater than zero) refers to the volume envelope being used (i.e. ENVELOPE number 1 etc).

These first four parameters form the heart of the SOUND statement, and even with them alone we can create some good music. In Chapter 5 we will see how we can generate music using just these parameters. However, there are still three further parameters we can use if we wish. The next two are involved with the envelope commands ENV and ENT, which are explained in Chapters 6 and 7 respectively. For now I will explain how they fit into the SOUND command, but a detailed account will be given in those two chapters.

USER-DEFINED ENVELOPES: ENV AND ENT

The fifth parameter, VE, is entered when a volume envelope (ENV command) has been defined. The volume envelope allows the programmer to define how the volume varies within a given sound.

The value that can be entered here can be in the range 0 to 15, though if 0 is used then no volume envelope is set. The positive

values refer to the envelope to be used. For example, if VE = 4 then ENvelope number 4 would be used.

The sixth parameter, TE, the tone envelope (ENT command), works in a similar fashion except it refers to the variation of tone within a sound. The range of values is 0-15, the same as for VE.

These options, when used, add new dimensions to the sound statement and—as we shall see—are the core of the different sound effects we can generate.

And finally ...

THE NOISE PARAMETER

I have mentioned previously that we can use 'white noise' effects on the Amstrad. The final parameter, NP, the noise period, specifies whether noise is to be used. If this parameter is not used or the value is zero then no noise is created. But if the value is in the range 1 to 31 then noise is added to the specified sound channel. To see the effect of this parameter, type in Program 3.2.

Program 3.2

```
10 ENV 15,43,-68,3
20 FOR volume = 0 TO 7
30 SOUND 1,478,25,volume,15,0,0
40 NEXT
```

When you run the program you will hear short on and off tones. Note that I have used all seven parameters. The VE parameter has been set to 15 to use ENvelope 15 which is defined in line 10. As before, you can pass over this command for the time being. The sixth parameter, TE, is set to zero as I haven't defined a tone envelope. The final parameter is set to zero, which means that no noise is being added to the sound channel. So at the moment this parameter is having no effect on the sound being produced. However, we can change this by altering that final value. Change line 30 to read:

```
30 SOUND 1,478,25,volume,15,0,10
```

Hear the effect? If you want to hear the difference this makes between a tone and a noise type the two following lines:

```
SOUND 1,478,100,15
```

```
SOUND 1,478,100,15,0,0,10
```

And that concludes the basic parameters of sound generation. You are now well equipped to start playing around with the SOUND command. It might seem a little complicated at first but the best way to master the sound commands is to type in a variety of sound statements and alter the parameters to see their effects. To help you sound out the Amstrad use Program 3.3 which allows you to enter the parameters P,D,V and NP. Run this program, and when you are satisfied that you thoroughly understand the effects of these parameters move on to the final part of this chapter.






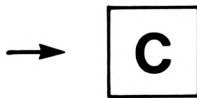
Program 3.3

```
10 CLS
20 PRINT"Enter pitch value (0 to 4095)"
30 INPUT p
40 PRINT"Enter duration value (1 to 32767)"
50 INPUT d
60 PRINT"Enter volume (0 to 7)"
70 INPUT v
80 PRINT"Enter noise, if any (0 to 31)"
90 INPUT np
100 SOUND 1,p,d,v,0,0,np
110 PRINT"Press a key to continue"
120 x$ = INKEY$: IF x$ = "" THEN 120 ELSE 10
```

SOUNDS UNIFIED: THE CHANNEL PARAMETER REVISITED

At the start of the chapter I described the channel parameter of the SOUND command. However, as I indicated, there is more to that parameter than just the three values that indicate which sound channel is to be used. We can now consider the other values you can input into this variable. Let us first update Table 3.1 where the sound channel values were shown. Table 3.3 shows a more complete set of values, though these are still not all the possible values, as I will shortly explain!

Table 3.3 Sound channel values(2): Channel status.

<i>Value</i>	<i>Channel(s)</i>	<i>Effect</i>
1		SOUND channel A
2		SOUND channel B
4		SOUND channel C
8		Rendezvous with channel A
16		Rendezvous with channel B
32		Rendezvous with channel C
64		Hold
128		Flush

The state of the channels (or channel status) indicates to the machine which sound channel is to be used, whether the channel is played in unison with one or more other sound channels, if the sound is held (i.e. continues until RELEASEd) or if the sound channel is flushed (i.e. the sound queue is cleared).

The values shown in Table 3.3 send the relevant command to

the sound generator. By adding these values together, we can combine several commands. For example, if we want to send a sound to channels A and B at the same time, we could write it as follows:

SOUND 3,478,50,7

where 1 (channel A) + 2 (channel B) = 3.

To give you a better idea of this process, type in Program 3.4, which sets up a few keys on the keyboard.

Program 3.4

```
10 KEY 128,"SOUND 1,478,50,7" + CHR$(13)
20 KEY 138,"SOUND 3,478,50,7" + CHR$(13)
30 KEY 140,"SOUND 5,478,50,7" + CHR$(13)
40 KEY 131,"SOUND 7,478,50,7" + CHR$(13)
```

As you can see, the KEY command is used to define four keys. First run the program. You should just get back the prompt 'Ready'. To play the commands assigned to these keys you will need to hold the CTRL key down while pressing one of the keys on the numeric key pad. Try it in the following order:

```
CTRL + 0
CTRL + .
CTRL + ENTER
CTRL + 3
```

You will have to listen very carefully, but should be able to hear differences in the sounds produced. In a similar way by different permutations of the other values, different statuses of the channels can be created. For example, to send a sound to channel A, rendezvous with channel B and hold, the channel status value would be 81 because:

1 (channel A) + 16 (rendezvous with B) + 64 (hold) = 81

These facilities are very important, as they offer the programmer the ability to synchronize sound channels, and to clear the sound channels when necessary.

SQ AND RELEASE

And finally two further BASIC commands which will be of use. The first is the SQ command which indicates the number of free entries left in a sound queue for the specified sound channel. This is also useful in determining if a sound channel is still active. Program 3.5 shows an example of its use.

Program 3.5

```
10 PRINT"Channel A is sounding"  
20 SOUND 1,261,250,7  
30 PRINT"The End"
```

If you run this program you will see that the message 'The End' appears straight away, even though the sound continues. So in order to test the sound channel to see whether the channel is still active we have to use the SQ command. Add line 25 as follows:

```
25 WHILE SQ(1) > 127:WEND
```

Now if you run the program the final message does not appear until the sound has terminated. This is because a value of greater than 127 will be returned while the sound channel is active. Line 25 continues in a loop until that condition is broken. The number in brackets after the SQ refers to the sound channel number.

Secondly the RELEASE command. This releases a sound on a specified channel if the sound is in a 'hold' state. The values are in the range 1 to 7. For example RELEASE 1 would release a hold state, if it existed, on channel A.

And that is the SOUND command. The best way to understand everything discussed in this chapter is, as I suggested earlier, to play around with the SOUND command. This in itself is good fun and may yield some unexpected results. I will return to the SOUND command in Chapter 5.

4 A Musical Interlude

This chapter is an introduction to musical terms and notation. If you are acquainted with musical notation, if perhaps you play an instrument, you can quickly skim through this chapter. However, as this book is intended to introduce the sound possibilities of the Amstrad to every user, it would be incomplete without helping those not familiar with musical jargon and notation. It is by no means a complete course, but if carefully read it will be of great assistance in understanding later chapters and when you convert any piece of music to the Amstrad.

STAVES AND NOTES

First of all, don't be put off by the sight of musical notation. To those who have never read music it can seem a very daunting task. All those lines, dots and strange symbols ... Well, it's not as hard as it may look. Let's acquaint ourselves firstly with the way music is set out on paper. Just as words are often written on lined paper, music is written on lines. These are grouped together in a *stave*, a set of five horizontal lines. Figure 4.1 shows a *stave* with nothing written on it, rather like a blank piece of paper.

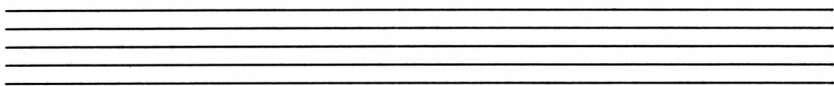


Figure 4.1 A blank stave.

Musical tones are represented on a *stave* by characters called *notes*. The vertical position of the note on the *stave* represents its pitch. Notes are written on or between the lines. The higher the pitch, the higher the note is placed on the *stave*. The line on

which the note lies, or the lines it lies between, indicate the letter name of the note. Figure 4.2 shows a typical set of notes.

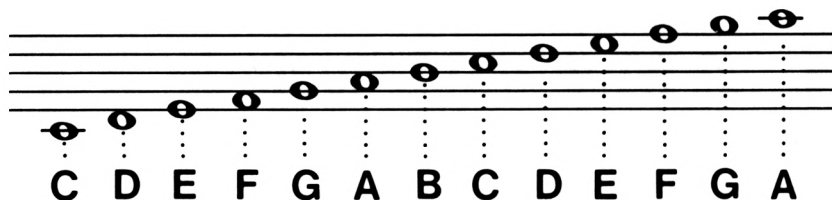


Figure 4.2 Set of notes on a staff with ledger lines.

Notice that the notes are repeated. The notes are given letter names from A to G and then—an octave higher—from A again. Notes too high or low to be placed on the staff can be shown by adding short lines called ledger lines above or below the staff. These ledger lines therefore allow the musician to extend the range of a staff. However, it is conventional not to use too many ledger lines as this could make the music virtually unreadable.







DURATION

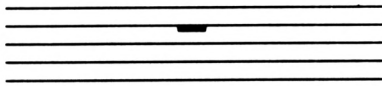
Leaving the pitch of the note for the moment, we can look at how notes of different duration are represented. A different symbol is used for each note, indicating its timing value. At this stage a point to be remembered is that these timing values are not an indication of how fast or slow a piece of music should be played. That is something I will cover shortly. The different timing values accorded to notes are an indication of the length of each note in relation to the others. Table 4.1 shows these timing values with the names associated with each.

The semibreve is (normally) the longest sounding note and the demisemiquaver the shortest. If a note has a dot (.) after it the duration of the note is increased by one-half. It is also possible to indicate periods of silence in the music by inserting rests into the music. Figure 4.3 shows the symbols for rests: the time values are equal to the notes of the same name.

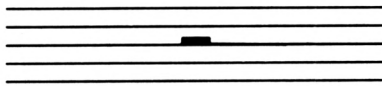
If the rest is for a long time, then a number is placed above the rest. For example, if the number 12 were placed above a rest, this would indicate 12 measures of that rest.

Table 4.1 Timing values.

<i>Note</i>	<i>Name</i>
	Semibreve or whole note
	Minim or half note
	Crotchet or quarter note
	Quaver or 8th note
	Semiquaver or 16th note
	Demisemiquaver or 32nd note



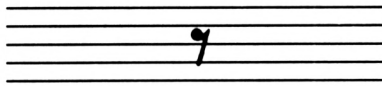
Semibreve rest



Minim rest



Crotchet rest



Quaver rest



Semiquaver rest



Demisemiquaver rest

Figure 4.3 Rests.

CLEFS AND SCALES

I mentioned earlier that the range of a staff could be extended by ledger lines. To make the music more legible we can add another staff below the first one. These are then two separate staves whose pitch range is shown by a clef sign. A clef sign that appears at the start of the staff fixes the pitch or letter name of a particular note. The two most common clefs used, in particular for piano music, are the treble and bass clefs. The treble clef has the higher pitch of the two and is therefore placed on the top staff. Figure 4.4 shows an example of this.

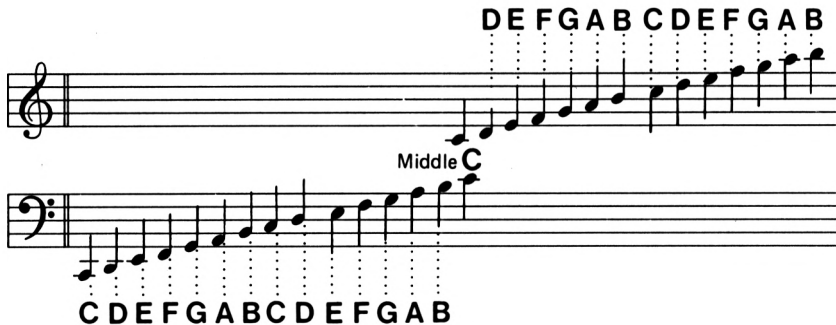


Figure 4.4 Treble and bass clefs.

You can see that the positions of the notes on the staves are different according to the clef, and that middle C can be shown on either staff. A series of eight notes is called a scale. The simplest scale is that of C major. This can be shown on a piano keyboard as in Figure 4.5.

You will note that C major uses only white notes, and that there is no black note between E and F or between B and C. These pairs of notes are closer together in pitch: they are only one semitone apart. All other pairs of notes have a black note in between them, and are two semitones or one tone apart. All major scales follow this same pattern: tone, tone, semitone, tone, tone, tone, semitone. If you move up the keyboard in a single sequence, playing all the black notes as well as the white notes, you will move up a semitone each time, and after you have played twelve notes you will be back at the note name you started with.

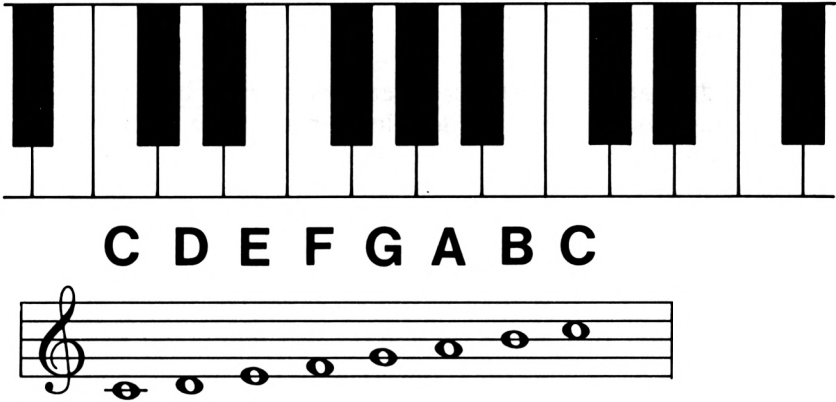


Figure 4.5 How the notes on the staff relate to the piano keyboard.

This twelve-note sequence is called a chromatic scale.

For major scales other than C you will need to use the black notes. Black notes are named according to the white note immediately above or below them in pitch. For example, the black note which is one semitone above D is called D sharp (#) and the black note one semitone below D is called D flat (b). If these signs are included at the beginning of the music they are known as the key signature. For instance, the key signature of G major has one sharp (#), that being F#. This means that when the scale is played the note F is always raised by a semitone. Figure 4.6 shows you a few sample key signatures.



Figure 4.6 Some key signatures.

A note that has been sharpened or flattened can be restored to its original pitch by the sign \natural (natural) which cancels the previous sharp or flat.

BARS AND RHYTHM

Music is divided into portions called measures or bars by vertical lines called bar lines. A double vertical line indicates the conclusion of a passage or piece. At the beginning of the music the length of the bars is indicated by the time signature. This consists of an upper and lower number, where the upper number indicates the number of counts (beats) within each bar and the lower number shows the time value of each count. Figure 4.7 shows this.



Figure 4.7 Bars and timing values.

If the time signature is $2/4$, for example, this indicates to the musician that there are two beats in a bar and that each beat is a crotchet: each count equals a quarter note or crotchet, represented by the lower number being 4. It is by these means that rhythm can be established.

More useful points to note are the different ways notes can be played. There are three aspects especially worth looking out for. The first is if a dot appears over or under a note. If it does, then the note is played staccato: this means to cut the note short, thereby leaving a slight gap between the note and the next note.

The opposite of this is a slur, which is a curved line connecting a series of two or more notes. These notes should be played in a smooth and connected way. This is called legato.

Finally, if a curved line appears between notes of the same pitch it is called a tie. This has the effect of adding the time value of the tied notes together, which will in effect produce a single, longer note.

Figure 4.8 shows examples of the above three notations.



Figure 4.8 Slurs, ties and staccato.

Now all of this may have confused you. If it has, don't despair. You need not know this section by heart, but you may find it useful later to refer to some of the points made. In a book of this nature it is not possible to give a thorough knowledge of even the basics of music. If you wish to go further into the theory of music there are many books which can teach you the rudiments of music and its notation.

As we progress, you will see that the lessons learned in this chapter will help you cope with understanding the musical programs and how to convert musical notation into a form that the Amstrad will understand. So once you feel happy (well, almost!) with the principles in this chapter, move on to Chapter 5, and we will start writing music on our Amstrad.

5 The Sound of Music

Converting music into a form that the Amstrad will understand is a relatively simple task. In Chapter 3 I said that the pitch parameter of the sound statement can produce a recognizable pitch (see Appendix E). We can now use those values to enable the sound chip to play musical notes.

CHROMATIC SCALES

A good place to start understanding the relationship between the Amstrad's pitch parameters and musical notes is to consider the problem of constructing a chromatic octave from any note, that is, the series of 12 semitones that makes up an octave. Writing such a program is not that easy! The intervals between each semitone on the pitch parameter table (Appendix E) are not uniform, and nor are the frequency intervals. What we therefore need to do is to obtain the correct frequency for each note and convert this to its pitch parameter.

The conversion of a given note into a frequency number can be carried out by an equation. Appendix 7 of the Amstrad user guide gives a formula to do this—but it is incorrect! If you use the equation given you will be about 18 semitones away from the correct note! The correct formula reads as follows:

$$\text{frequency} = 440 * (2 \uparrow (\text{octave} + ((n - 10) / 12)))$$

where 'octave' is the octave number over the eight-octave range (see Appendix E) and 'n' is the note (C = 1, C# = 2 and so on for each semitone).

Having obtained the correct frequency, this is then converted into the pitch parameter by the following formula:

$$P = \text{ROUND}(125000/\text{frequency})$$

The program we want is therefore Program 5.1.

Program 5.1

```
10 FOR num = 1 TO 12
20 READ a(y)
30 freq = 440 * (2↑(0 + ((a(y) - 10) / 12)))
40 pitch = ROUND(125000/freq)
50 SOUND 1,pitch,35,15
60 NEXT
70 DATA 1,2,3,4,5,6,7,8,9,10,11,12
```

Line 70 contains numbers which represent the different musical notes starting at 1 (C) and working upwards a semitone at a time to play all 12 semitones in the octave. Line 30 converts the note number to a frequency and line 40 converts that figure to a pitch value.

SCALES

From that basis it is not difficult to go one step further and to create a scale, say C major. In the last chapter I gave you the rule for producing major scales, the progression being: tone, tone, semitone, tone, tone, tone, semitone.

All we then need to do is to change the data in line 70 to:

```
70 DATA 1,3,5,6,8,10,12,13
```

and change line 10 to:

```
10 FOR num = 1 TO 8
```

Run the program and you will now hear the scale of C major starting at middle C and finishing at C above middle C. The data in line 70 is easy to comprehend. The value 1 is middle C, and the other seven values represent the progression of tones and semitones given above. A semitone is represented by an increment of one and a tone by an increment of two.

TUNES!

We can take this idea much further and convert familiar tunes to play on the Amstrad. Type in Program 5.2 and run it. You should hear a familiar ragtime tune, but I'm not telling you the name. You're going to have to type it in to find out!

Program 5.2

```
10 tempo = 2.5
20 RESTORE 90
30 FOR x = 1 TO 37
40 READ pitch,duration
50 freq = 440 * (2↑(0 + ((pitch - 10) / 12)))
60 pitchnum = ROUND(125000/freq)
70 SOUND 1,pitchnum,duration * tempo,15
80 NEXT
90 DATA 27,10,29,10,25,10,22,20,24,10,20,
20
100 DATA 15,10,17,10,13,10,10,20,12,10,8,
20
110 DATA 3,10,5,10,1,10,-2,20,0,10,-2,10,
-3,10,-4,40
120 DATA 3,10,4,10,5,10,13,20,5,10,13,20,
5,10,13,40
130 DATA 13,10,15,10,17,10,13,10,15,10,17,
20,12,10,15,20,13,40
```

Recognize it?! Now you're probably wondering how it all works. The tempo figure in line 10 can be altered: increasing the value will slow down the tune, decreasing it will speed it up. Line 70 is the sound statement. You will notice that the third parameter, the duration number, is a combination of the tempo setting and a duration figure. The duration figure is read in from the data statement where the notes have been entered in pairs, the first value being the note to be played, as I have shown previously, and the second value the duration of the note. The duration figures are equivalent to a musical timing—we look at these in

Chapter 10. For the moment you can ignore these values and just look at the pitch numbers.

The core of the program, lines 50 to 70, is the same as Program 5.1, but I have translated the piece of music by taking each note and entering its pitch parameter. In order to help you do this, Appendix E has these numbers set out, assuming that you work from middle C and that that octave is given the value zero (see line 50). Once you get the hang of this method it can be quite quick to convert a set of musical notes to sound on the Amstrad.

To practise this, take an easy piece of music, one where there is a line of single notes, not chords, and convert these notes into a data statement. Disregard the duration parameter by inserting an average time in the duration part in line 70. Without the duration it will sound a bit odd but this will help you to get the feel of converting music.

Now let's leave music for a while, and look at the two remaining sound commands, ENV and ENT.

6 The Volume Envelope

The volume envelope command (ENV) allows the user to manipulate the volume within a given sound. When used it can alter the sound generated by a SOUND command. In common with the SOUND command it has a number of parameters, and may appear complicated at first. But there is far less to explain than you might think, as most of these parameters follow the same rules.

To see what I mean take a look at the command in all its glory:

```
ENV n,P1,Q1,R1,P2,Q2,R2,P3,Q3,R3,P4,  
Q4,R4,P5,Q5,R5
```

Don't take fright! This is obviously what is known as a user-friendly micro! Actually the only parameters I need explain are the first four. But first let's take a closer look at the shape of a sound.

SHAPING THE SOUND

When using the ENvelope commands a great deal of time can be wasted entering a variety of values. Because of the numerous permutations available, creating the sound you desire cannot be left to chance. It is obviously worth your while to invest a few minutes in thinking about the sound you wish to create. When an ENvelope is defined by the user, the control of the amplitude of that sound is handled by the ENV command.

Every sound you hear has a distinctive shape and it is this shape that we are defining when we write the ENV command. Type in Program 6.1 and run it.

Program 6.1

```
10 ENV 1,10,4,3,5,-3,20,1,0,20,5,3,  
10,10,-3,30  
20 SOUND 1,478,0,0,1,0,0
```

The sound you hear is being totally controlled by the ENV statement. Line 20 defines the pitch, but the duration is set to 0 and this tells the machine that the sound should last until the end of the volume envelope. Also the volume parameter is set to 0 as the amplitude is under the control of the volume envelope. The 'VE' parameter indicates that volume envelope number 1 should be used, which is the envelope we have defined in line 10. To see what is happening in that line look at Figure 6.1.

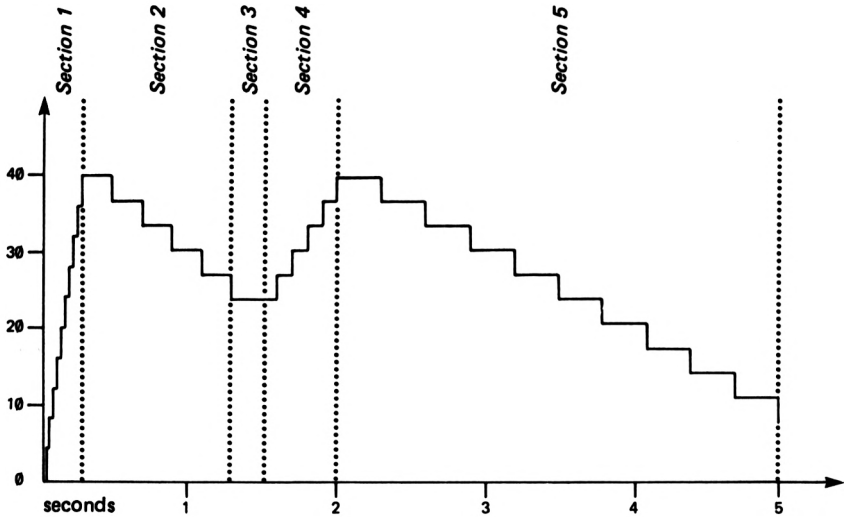


Figure 6.1 Example shape of volume envelope.

The shape of the note is divided into five sections, each section having three parameters. Figure 6.2 shows how the five sections relate to the envelope parameters.

Within each section there are three parameters, the step count (P_n), (where n equals the section number), step size (Q_n) and the pause time (R_n). At least one complete section must be defined,

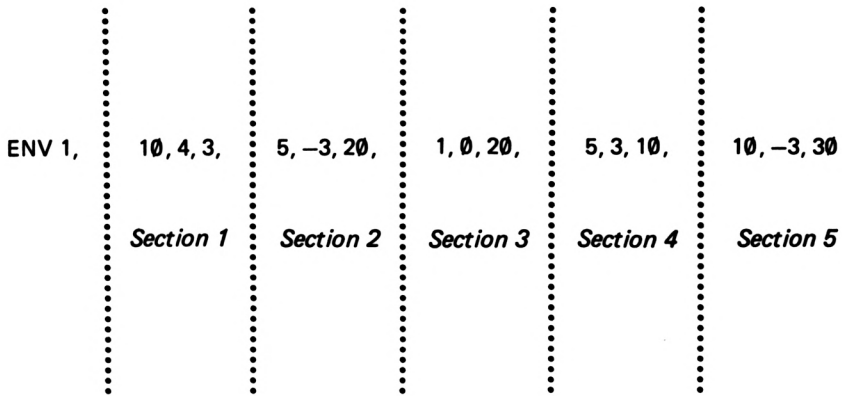


Figure 6.2 The five sections of the volume envelope.

and up to five can be used.

The step count P_n can have any integer value from 0 to 127. It represents the number of steps in each section in Figure 6.1. Each of these steps has a fixed duration, and this is the pause time (R_n), measured in hundredths of a second—a value in the range 0 to 255 can be entered. In between these two parameters is the step size (Q_n). This indicates the amount by which the amplitude increases or decreases at each step, a negative number indicating a decrease. The range of possible values is from -128 to +127. The effect on the shape of the note can be seen in Figure 6.1. Each step in section 1 increases in intervals of 4 (the step size).

The graph indicates that the sound will last for 5 seconds. We can check this by multiplying the pause time figure by the step count figure in each section and adding these totals together:

$$(3 * 10) + (20 * 5) + (20 * 1) + (10 * 5) + (30 * 10) = 500$$

As we are counting in hundredths of a second, 500 is equal to 5 seconds.

Once an envelope has been set these parameters are stored by the sound chip and used every time the envelope is called. If you wish to cancel the effect of an envelope on a SOUND command this can be done by specifying the appropriate envelope by name but without allocating any sections. So in order to cancel the effect of ENVelope 1 you would type:

ENV 1

The envelope would then be inactive.

PLAYING AROUND!

Having said earlier that you should think about the shape of the sound you intend to imitate before writing the envelope, I am going to make an exception to that point right now. To give you an idea of the effects of altering the different parameters, Program 6.2 allows you to edit the three parameters. Only one section has been used.

Program 6.2

```
10 CLS
20 PRINT"Enter step count (0 to 127)"
30 INPUT stepcount
40 PRINT"Enter step size (-128 to +127)"
50 INPUT stepsize
60 PRINT"Enter pause time (0 to 255)"
70 INPUT pausetime
80 ENV 1,stepcount,stepsize,pausetime
90 SOUND 1,240,15,15,1,1,0
100 ENV 1
110 PRINT"Press a key to continue"
120 x$ = INKEY$: IF x$ = "" THEN 120
130 GOTO 10
```

See what you can devise just from messing about with a few numbers. You may come across some very interesting sounds!

INSTRUMENTS

It is likely that you will want to try to create the sounds of musical instruments. This is possible, within reason, though do not forget that we are dealing with an electronic sound chip which does have its restrictions. Whenever anyone produces an envelope that they believe sounds like a violin or an organ, you will surely find someone else who thinks they sound like a clarinet or a saxophone! With this in mind I won't give you a list of pre-defined instrument envelopes but let you discover the right

ones for yourself. To help you on your way, here are a few pointers as to how some instrument envelope shapes look. This is not a definitive list, and you must decide which sounds you like for yourself.

Figure 6.3 shows approximate sound shapes produced by various kinds of musical instrument.

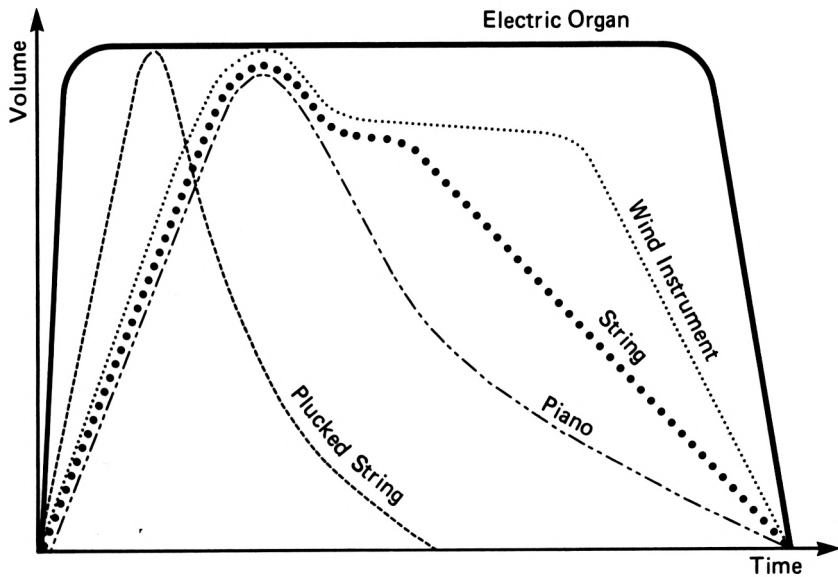


Figure 6.3

Aim to match the shape of the curve, and then you can attempt to 'tune' the sound more finely to your taste.

For a more subtle adjustment of the sound you can add changes to the frequency of the note by using the tone envelope command, which is explained in the next chapter.

7 The Tone Envelope

If you have understood how the volume envelope works then this chapter should be a nice easy read. The tone envelope (ENT) is formed in virtually the same manner as the ENV command except that its effect is not on the amplitude of the note but on the tone.

The ENT command causes small variations in the pitch of a note. In thus varying the tone by creating a pulsating effect we are producing a form of vibrato within the sound. This affects the harmonics of the sound wave, and many musical instruments rely on these variations for their unique sounds.

SHAPING TONES

The parameters of the ENT command should not be difficult to follow: they are arranged in groupings similar to the ENV command. The parameters are as follows:

```
ENT n,T1,V1,W1,T2,V2,W2,T3,V3,W3,T4,  
V4,W4,T5,V5,W5
```

where n is the envelope number. As in the previous chapter, I have followed the conventional letter descriptions used by the Amstrad user guide, though they seem to have little in common with the parameters being described.

As with the volume envelope, the tone envelope can be set out in the form of a visual graph, and it is best in the cases of both envelopes to sketch out the shape of the envelope. The ENT command controls the tone of a note and when this envelope has been defined, the control of the note in the SOUND statement is handled by the relevant ENT command.

Program 7.1 defines a sample envelope using four of the five

possible sections allowed. When you first run the program you will hear the tone as it sounds without the ENT statement. Remove line 15, which like its counterpart ENV, simply cancels the definition of the tone envelope.

Program 7.1

```

10 ENT 1,65,5,1,10,-2,10,10,2,5,30,-5,1
15 ENT 1
20 SOUND 1,478,50,15,0,1,0

```

The sound you now hear is the effect of the tone envelope on the note. Figure 7.1 describes what is happening in both the SOUND and ENT commands.

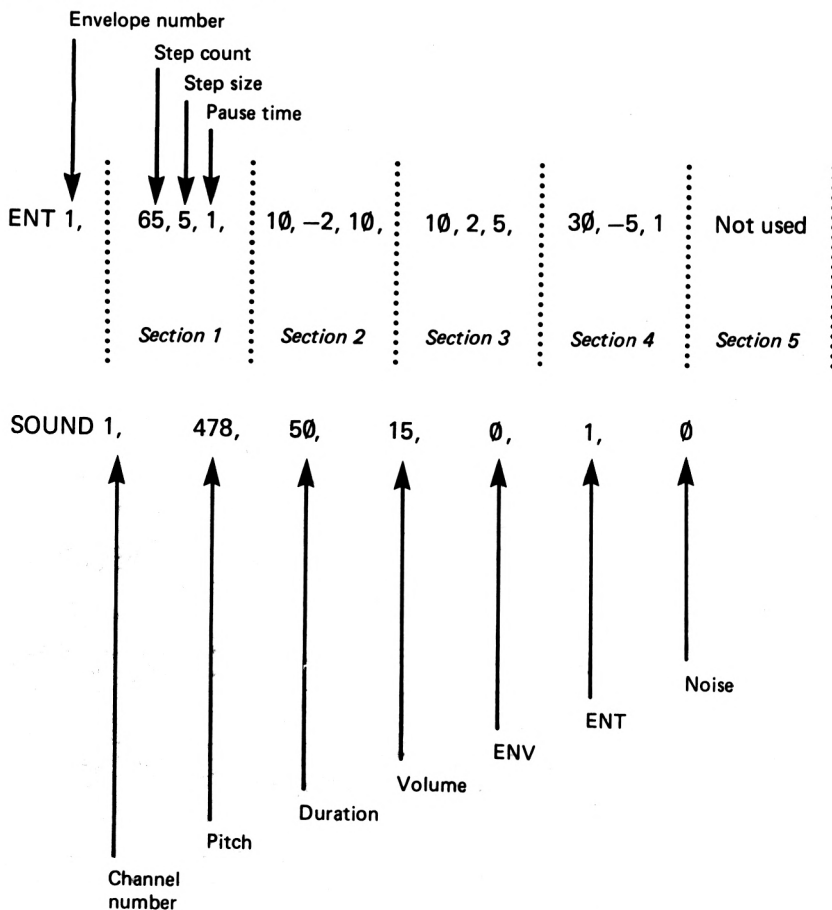


Figure 7.1 The relationship between SOUND and ENT statements and sections of ENT commands.

Another example of the shape of the ENT envelope is shown in Figure 7.2.

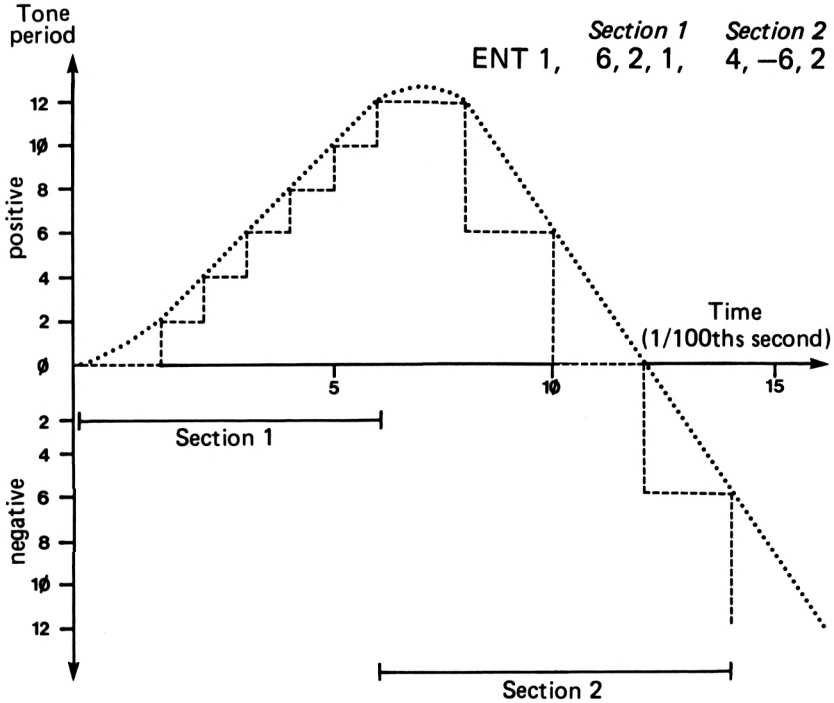


Figure 7.2 Example shape of tone envelope.

Within each section there are three parts, the step count (T_n), the step size (V_n) and the pause time (W_n). As with the volume envelope, at least one section must be defined, unless you are cancelling an envelope, where you can just use the form ENV n. If you begin the definition of a section you must complete it, as the envelope will not work if an operand is missing.

The range of values allowed is identical to that for the ENV statement for the step size and pause time parameters, but the step count has a range of 0 to 239. The first parameter n, which specifies the envelope to be used, can also indicate whether a tone envelope is to be repeated. By using a negative number the tone envelope will repeat for the duration of the sound. This is of use when you wish to create tremolo effects.

If you want to experiment with the properties of the ENT command, type in Program 7.2, which works in an identical

manner to Program 6.2 except that it has been set up for use with the ENT envelopes.

Program 7.2

```
10 CLS
20 PRINT"Enter step count (0 to 239)"
30 INPUT stepcount
40 PRINT"Enter step size (-128 to +127)"
50 INPUT stepsize
60 PRINT"Enter pause time (0 to 255)"
70 INPUT pausetime
80 ENT 1,stepcount,stepsize,pausetime
90 SOUND 1,240,100,15,0,1,0
100 ENT 1
110 PRINT"Press a key to continue"
120 x$ = INKEY$:IF x$ = "" THEN 120
130 GOTO 10
```

ATTACK, SUSTAIN AND DECAY

And with that we have almost concluded our look at envelopes. The inclusion of envelopes in the BASIC language gives a powerful facility to the programmer wishing to control the sounds generated by the sound chip. Defining an envelope allows you to control the attack, sustain and decay rates of a sound. If these terms aren't familiar to you, do not fear. You have actually been looking at these factors in these past two chapters. When we use the ENV and ENT statements we are either increasing, decreasing or keeping constant the amplitude or pitch. When you hear a sound the note increases in volume, usually sharply, this being the attack phase. When this has reached its peak the sound will be sustained while the note continues to sound, and then the decay phase comes into play as the note dies away. Therefore different sounds (perhaps instruments) will all have these three parts contained within the generated sound, but the rates within these parameters will differ according to the shape of the sound being created. Let's look at the factors of attack, sustain and decay in a

little more detail with reference to a volume envelope.

The attack phase is the first part of sound generation. It determines the amount of time a note will take to reach its peak or maximum level. You can hear examples of this in all the musical instruments we use. Pianos, for instance, have a very sharp attack rate because the instrument's sound is created by striking the strings with a hammer. Try and think of the sounds of other instruments and whether they build up to their maximum amplitude sharply or gradually.

The sustain phase keeps the volume of the note constant, and finally in the decay phase the volume dies away back to nothing. Figure 7.3 shows these three phases.

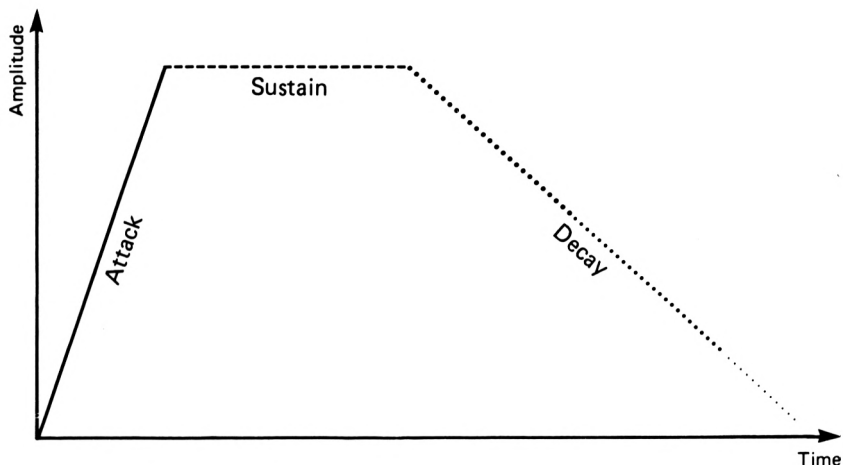


Figure 7.3 The three phases of sound.

It is difficult for the Amstrad sound chip to produce accurate representations of musical instruments, as the tone it produces is not a pure sine wave but a square wave which inherently includes its own harmonic pattern. However, all is not lost. Reasonable approximations can be synthesized by controlling the amplitude and tone envelopes, and with a little work you should be able to build a library of your own instrument sounds.

And that explains the main commands concerned with the generation of sound. The following chapters continue our exploration of the sound chip with everything from music to space invaders.

8 Whiz Bang Wallop!

This book's title *Bells and Whistles ...* is most appropriate in view of the noises that you can generate! Besides producing music the Amstrad has great facilities for providing sound effects for many programs, especially games, or just for creating sound for its own sake.

The Amstrad can be a noisy beast and we can make effective use of these sounds in different ways. The first place we can use sound effects, and probably the most popular use, is in games writing.

Computer games just wouldn't be the same without those ear-shattering sounds of crashes, explosions and laser blasts! Let your imagination run wild and you will soon start to produce sounds to match the visual images on your monitor screen.

ADVENTURE IN SOUND

A journey through the sound channels of space and time

You're drifting through space at hyperspeed 7.6 when on your intergalactic supersensitive monitor you notice an alien vessel approaching at ultrawarp speed. You reach for the general alert button and ...

... you type in Program 8.1!

Program 8.1

```
10 ENT 1,80,-4,1
20 SOUND 1,478,50,15,0,1,0
30 GOTO 20
```

Having sounded the alert, the crew of the Federation Star Amcruiser rush to their battle stations to await the first sign of hostile action. Suddenly, the alien ship disappears from your scanners and all is quiet. Time drifts by, but there is no sign of the aliens. You are beginning to wonder whether the spacecraft ever existed, when all of a sudden the ultra-matrix printer jerks into life and prints out ...

Program 8.2

```
10 FOR type = 1 TO 45
20 SOUND 1,901,7,15,0,0,1
30 FOR delay = 1 TO 100:NEXT
40 NEXT
```

The printer continues to print out a garbled collection of characters which means nothing in any human language. Why don't they use electromagnetic communication? As if someone has read your thoughts the communicator by your hand buzzes to life.

Program 8.3

```
10 FOR repeat = 1 TO 6
20 radio = INT(RND * 20)
30 SOUND 1,radio,100,15,0,0,1
40 NEXT
```

But your communicator isn't capable of tuning into the sending frequency of the aliens. Will you ever find out what is happening? Probably not, as this fantasy is reaching its conclusion. Your only chance is to call upon the ship's computer. You quickly tap in your request and the computer processes your problem.

Program 8.4

```
10 FOR repeat = 0 TO 140
20 compute = INT(RND * 150)
30 SOUND 1,compute,3,15,0,0,0
40 NEXT
```

You eagerly await the reply. Nail-biting seconds pass, and then on your screen comes the reply: "GAME OVER. PLEASE INSERT 10p FOR ANOTHER GAME!"

Well, perhaps my imagination leaves much to be desired, but you can see that it's not hard to fit a sound or noise to a storyline. You may like to join the above programs into one and play each sound at the appropriate stage in the tale.

By adding noise to a sound, using the last parameter of the SOUND command, the sounds in Program 8.4 can be easily converted to form the second half of a 'drop the bomb and explode' program. Type in Program 8.5.

Program 8.5

```
10 FOR drop = 50 TO 150
20 SOUND 1,drop,3,15,0,0,0
30 NEXT
40 FOR repeat = 0 TO 45
50 compute = 1
60 SOUND 1,compute,3,15,0,0,31
70 NEXT
```

All that is required is a little imagination and exploration. With these sound effects it is usually best to experiment with sound and envelope commands to produce the noise you need. The next program is an envelope generator program to help you construct volume envelope and sound commands. Type in Program 8.6 and run it.

ENVELOPE GENERATOR

Program 8.6

```
10 :
20 REM Initialize program
30 :
40 ZONE 40
50 ENV 1
```

```

60 CLS
70 :
80 REM Menu definition
90 :
100 PRINT"VOLUME ENVELOPE GENERATOR"
110 PRINT,,"1 Volume envelope"
120 PRINT,,,,"Enter your choice"
130 a$ = INKEY$
140 a = VAL(a$):IF a<>1 THEN 130
150 PRINT a
160 ON a GOTO 170
170 CLS
180 :
190 REM Choose number of sections
200 REM within specified envelope
210 :
220 PRINT"ENVELOPE NUMBER 1"
230 PRINT,,"Enter number of sections (max 5)"
240 INPUT sections
250 FOR sec = 1 TO sections
260 PRINT,,"Enter step count (0 to 127)"
270 INPUT sc(sec)
280 PRINT,,"Enter step size (-128 to +127)"
290 INPUT ss(sec)
300 PRINT,,"Enter pause time (0 to 255)"
310 INPUT pt(sec)
320 :
330 REM Write parameters to volume
340 REM envelope statement
350 :

```

```

360 ENV 1,sc(1),ss(1),pt(1),sc(2),ss(2),
    pt(2),sc(3),ss(3),pt(3),sc(4),ss(4),
    pt(4),sc(5),ss(5),pt(5)
370 :
380 REM Return to next section, if any
390 :
400 NEXT
410 :
420 REM Enter parameters for
430 REM sound statement
440 :
450 CLS
460 PRINT"S O U N D   P A R A M E T E R S"
470 PRINT,"Which channel?"
480 INPUT channel
490 PRINT"Enter duration"
500 INPUT duration
510 PRINT"Enter noise (0 to 31)"
520 INPUT noise
530 :
540 REM Play sound
550 :
560 SOUND channel,478,duration,1,1,0,noise
570 :
580 REM Print out the envelope
590 REM parameters
600 :
610 CLS
620 PRINT,, "The envelope parameters are as
    follows:"
630 PRINT"Envelope 1"

```

```

640 FOR x = 1 TO 5
650 PRINT"SC";x;" = ";sc(x)
660 PRINT"SS";x;" = ";ss(x)
670 PRINT"PT";x;" = ";pt(x)
680 NEXT
690 PRINT"Do you want to hear that again?"
700 a$ = INKEY$:IF a$ = "" THEN 700
710 IF a$ = "Y" OR a$ = "y" THEN 560
720 PRINT,, "Do you want to change the
      sound?"
730 PRINT"parameters?"
740 a$ = INKEY$:IF a$ = "" THEN 740
750 IF a$ = "Y" OR a$ = "y" THEN 450
760 GOTO 40

```

The program first presents you with a menu options page. In this version of the program there is only one option for you to choose, that being the volume envelope option. I suggest that when you have got used to the program you add a second option for the tone envelope, or indeed any improvement on the program which helps you with the development of noise or sound effects. Once entered into the main program you are given the choice of how many sections of the envelope command you wish to define. If you press the ENTER key at this stage no envelope will be defined. Entering a number between 1 and 5 will specify the number of sections to be given values.

The three parameters within each section will now be presented, with their respective range of values. Finally you will be presented with sound parameter options where you must supply the channel number, duration and whether there is any noise. The sound you have created will be played and the envelope parameters you chose displayed. Further options allow you to repeat the sound or change the sound parameters.

The program should be easy to follow, as the main sections are marked with REM statements. It is a time-saving utility, enabling you to create and test a variety of sounds until you are satisfied with the results.

To get you started I have prepared a few of my own sounds to enter into the sound generator program. These are shown in

Table 8.1. Do bear in mind, though, that some sound effects may require extra manipulation not within the range of the program.

Table 8.1 Envelope parameters (sections 1–3 only) and sound values.

SOUND description	Section 1			Section 2			Section 3			SOUND channel	Duration	Noise value
	SC1	SS1	PT1	SC2	SS2	PT2	SC3	SS3	PT3			
Guns firing	12	13	14	15	16	17	18	19	20	7	145	30
String instrument	12	13	14	15	16	17	18	19	20	7	145	0
Knock on door/drum	23	-68	3	-	-	-	-	-	-	7	120	12
Explosion	15	5	1	15	-1	30	-	-	-	7	120	20

PERCUSSION

Finally, whilst on the subject of noise effects, percussion sounds will be of obvious interest to the musician. The noise channel in conjunction with the tone and volume envelope commands will yield a number of pseudo-percussive effects from snare drums to bass drums. Program 8.7 is a short example of the kind of drum sounds possible.

Program 8.7

```

10 ENV 1,23,-68,3
20 FOR duration = 1 TO 50
30 SOUND 7,748,duration,15,1,1,1
40 NEXT

```

And that concludes this introduction to the world of noise effects. You can spend many hours sorting through the weird and wonderful noises the sound chip will produce, and it won't be long before you can apply these noises in your own programs for effect, amusement or simply to whiz bang wallop your Amstrad.

9 Music Maestro Please!

We have covered a lot of ground since Chapter 1, and I think it's about time that you turned into a concert pianist—almost! One of the aims of this book is to make life easier for you when trying to create music, but until now the only way you could play music was to enter all the notes in DATA statements. This can be time-consuming and difficult with long pieces of music.

What we really need is a way of letting you play music directly on the keyboard. The problem is that the Amstrad doesn't have a set of piano keys. So we must make do with the next best thing. Yes, you guessed it, the QWERTY keyboard. Although not ideal for a musical instrument, it is an ordered set of keys which we can use. But how do we set about turning a typewriter keyboard into a musical instrument? You may think that this would involve a long and complicated program, but you'd be wrong. We can in fact write the program in only 8 lines!

AM-SYNTH

The Amstrad Piano Keyboard

Take a look at the keys on a piano and then at your Amstrad keyboard. Not much in common, is there? The point they do share is that every key can be depressed independently of the others. Now that gives us a good basis to work from. Our next problem is a bit more tricky. A piano keyboard has its keys laid out in one long line covering a range of octaves, from the lowest pitch to the highest. We are not so fortunate with the QWERTY layout. But this will not stop us. Our immediate aim should be to have a keyboard that has at least an octave range. Later on I'll show you how to extend this.

Let's take the top row of alphabetic keys (QWERTY etc) and

make this our row of 'white' keys. The 'black' keys will be on the row above, the numeric keys. And now we can begin to write the program. Type in these lines as they appear and I will explain their functions as we proceed. Do not run the program until it is complete. The first line is line 30.

```
30 w$ = INKEY$:IF w$ = "" THEN 30
```

This loops endlessly back on itself, waiting until a key has been depressed, when it goes on to line 40. Next we need to tell the machine which keys we are using. This is defined in line 10. The characters comprising s\$ are the keys that will represent a series of notes at semitone intervals. Do take note at this stage that when you run the program you must not have the CAPS LOCK key set to produce upper-case characters, as this will prevent the program from working properly.

```
10 s$ = "q2w3er5t6y7ui9o0p"
```

The sound statement comes next. This is line 60, which will play the note:

```
70 SOUND 1,pitch,15,15
```

Notice that the pitch is not yet defined. To work out the correct pitch we use the equations (given in Chapter 5) for converting a frequency to the pitch parameter. This is carried out in lines 50 and 60.

```
50 frequency = 440 * (2↑(1 + ((note - 10) / 12)))
```

```
60 pitch = ROUND(125000/frequency)
```

If you remember, the value of 'note' in line 50 determines the note played (where C = 1, D = 3 etc). Since we have decided that the first key will be 'q', then this becomes the musical note 'C'. Why? Well, take a look at line 40.

```
40 note = INSTR(s$,w$)
```

The INSTR command has been used in line 40 to check whether the key pressed, which is w\$, matches with any of the characters in s\$. If it does, a value is returned by the INSTR command and given to the numeric variable 'note'. Therefore if the user presses the key 'q', 'note' is given the value 1, and this corresponds to the musical note 'C'.

All we now need to do is to keep the program running in a loop, and we can achieve this by using a WHILE – WEND loop in lines 20 and 80.

```

20 WHILE x = 0
80 WEND

```

And that completes the program! Run the program and you have at your fingertips a keyboard starting at C above middle C (q) and continuing for a full octave plus two further tones. I have included this extension to give you that little more to play with. If you just want a one octave range you can cut out the last four characters of the 's\$' definition, in other words finishing on the alphabetic key 'i'.

Figure 9.1 shows how the Amstrad keyboard relates to a conventional piano keyboard. Remember that Appendix E has the full pitch and frequency tables with the relevant numbers to enter into the 'note' position in the frequency equation.

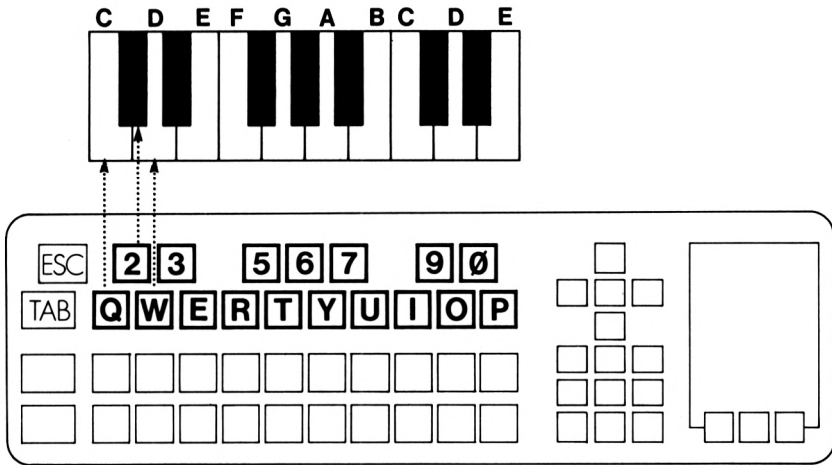


Figure 9.1 The relationship between the Amstrad keyboard and the piano keyboard.

Changing the octave being played is very easy. The octave position in line 50, which is defined in the above program as 1, can be changed to suit the pitch of your voice. If, for example, you replace the 1 in line 50 by 0, the range moves down an octave. All you need to do now is to give your first performance on the Am-Synth! Just in case you really aren't too musically orientated, Figure 9.2 shows a series of notes for you to play, but giving the

QWERTY keys to press rather than the musical notes. There is of course no indication as to the tempo, but I'm sure you will work that out as soon as you recognize the tune, which no doubt you will!

Q	W	E	Q	E	Q	E	W	E	R
R	E	W	R	E	R	T	E	T	E
T	R	T	Y	Y	T	R	Y	T	Q
W	E	R	T	Y	Y	W	E	R	T
Y	U	U	E	R	T	Y	U	I	I
U	Y	Y	R	U	T	I	Q	W	E
R	T	Y	U	I	Q				

Figure 9.2 *Mystery tune.*

MORE OCTAVES PLEASE!

Extending the range of the keyboard

I mentioned earlier in the chapter that the range of octaves could be extended. This may be done in a variety of ways, but the method I prefer is where the bottom two rows of keys on the Amstrad play a lower range, and the upper two rows the octave above. This would then give you a continuous run of at least two octaves.

To implement this range of two octaves plus, the following lines must be added to the Am-Synth program:

```

15 octave$ = "zscxdcvghbnjm,l.:/"
65 IF note = 0 THEN GOSUB 90

```

```

90 note = INSTR(octave$,w$)
100 frequency = 440 * (2↑(0 + ((note - 10) /
    12)))
110 pitch = ROUND(125000/frequency)
120 RETURN

```

You now have a keyboard with a range of over two octaves. Therefore to play the scale of C major over two octaves you would press the following keys: 'ZXCVBNM,WERTYUI'.

Both the upper and lower rows of keys themselves also extend a little over the octave range, so you have a choice of keys to play some mid-range notes.

The Am-Synth keyboard is a useful tool to play around with or to compose on. The program presented is intended to give you a start in using synthesizer keyboards and I hope you make use of it not just as it stands but as the basis of creating a synthesized keyboard with a range of effects. This would involve the use of the envelope statements and a little imagination on your part. Good luck and happy composing!

10 Dig that Beat!

Cast your mind back to Chapter 5 and you will recall that I showed you how to convert music into numbers that could be held in DATA statements in a program. I explained how to convert notes to numbers but the question of converting the duration of a note to a number was left unanswered. In this chapter we will see how to convert that musical notation into figures that the Amstrad can understand.










Timing in a tune is obviously very important. When was the last time you heard music that had each note played for the same length of time? It is the variation between the duration of the notes that gives music its characteristic beat and tempo.

But let's clarify that last statement. Remember that musical time notation, such as crotchets and quavers, represent differences in duration. These differences are relative to each other, and so these timing marks are not indications of tempo (the overall speed a piece of music is played at). Don't confuse these two concepts, because they are totally separate.

Returning to Chapter 5, you will have noticed when you played the mystery tune in Program 5.2 that not only did the tune play at an appropriate speed overall but each note had its own timing. When I translated the music into numbers I had to have a chart to refer to for the equivalents of the note timings (quavers, semiquavers etc). In constructing the timings it was important to ensure that all the timings were correct relative to the others. To see what I mean, look at Table 10.1 which gives the relative durations for note lengths.

You know that if a quaver has a duration of 20, the crotchet has a duration twice that of the quaver. By reference to the chart you can see that this is so. The crotchet has a duration of 40. These timings are those to be entered into the duration parameter of the SOUND statement. Our next problem then occurs. Every piece

Table 10.1 Amstrad durations in relation to musical note length.

<i>Note</i>									
<i>Duration</i>	10	15	20	30	40	60	80	120	160

of music plays at different speeds, so using these timings 'raw' will be of no use in most cases.

Well, we certainly don't want to keep different sets of data for all possible speeds! This is why in Program 5.2 I have a line that sets the tempo. This value is then multiplied by the duration figure to produce the overall length for the SOUND duration parameter. By altering just one line, line 10 in the case of Program 5.2, we can easily change the tempo of the tune.

From this basis you can then start to think about the different rhythms and beats possible for music. Experiment with rock and roll, or waltzes, or whatever—it doesn't matter. Rhythm is a very important factor in music, and changes the nature of a tune. There are almost infinite variations in rhythm, and you can try to generate tunes that play in different styles.

You now have all the information you need to convert music to play on the Amstrad. Now it's your job to get into the swing of things and start to liven up your synthesized tunes with a beat. So boogie on down and dig that beat!

11 Bells and Whistles

The spectrum of sound on the Amstrad is vast. It is virtually limited only by your imagination and the time you invest in seeking out new sounds, noises and effects. All the basic information you need to explore the sound channels of your machine is in this book. With that information you should now be in a position to create both noise and music effects.

Half the fun in writing a program that uses sound is to discover those beeps and noises for yourself. If you still get confused by the multitude of parameters or cannot understand the way a command or parameter is working, take your time. This book was intended for those who like to experiment. Play around with the commands as much as you like. It is one way of discovering how a command works.

I have also introduced the notion of music on the Amstrad to you, and with programs like Am-Synth you have a firm foundation from which to build more complicated and impressive musical performances. The subject of music has not been exhausted by any means and hopefully you will feel inspired and enthused enough to continue from where I've left off, creating instrument sounds, three-part harmony, drum kits and much more.

At the end of this book you will find a number of appendices which will act as a useful reference point for those moments of doubt, so don't hesitate to consult them.

Do remember that the sounds you devise are personal interpretations and that different people may hear the same sound somewhat differently. You are making no more than an approximation to a certain sound. Some approximations will be much better than others. This will be in many cases due to the limitations of the sound chip. Your job is to stretch the sound generator to its limits—and what those limits are is for you to find out.

I can't finish a book on sound without a few final quick sound

effects. And what better way to end than to use the noises that confront me through my working day? The annoying engaged telephone ...

```
10 SOUND 1,100,40,15
20 FOR pause = 0 TO 900:NEXT
30 GOTO 10
```

or if it's not engaged it rings, and rings, and rings!

```
10 ENV 1,100,122,1
20 SOUND 1,239,0,15,1,0,0
30 FOR x = 0 TO 2000:NEXT
40 GOTO 20
```

And finally, this book wouldn't be complete without at least one whistle effect: a train whistle.

```
10 FOR blow = 1 TO 3
20 SOUND 1,239,80,15,1,2,1
30 FOR delay = 0 TO 900:NEXT
40 NEXT
```

And that is the end of the book. But for you it should just be the beginning of a journey of discovery, uncovering the hidden talents of your Amstrad and the vast range of sounds it can produce.

Appendix A Amstrad BASIC Keyword Summary

This appendix contains a summary of the Amstrad BASIC commands. It isn't a replacement for the user guide nor a detailed description, but sufficient to jog your memory on any command you may use in your programming. For further information on any of these, refer to the user guide.

ABS	Gives an absolute value
AFTER	Used with internal timers to GOTO a subroutine after a specified time period
AND	Logical 'and' operator
ASC	Converts a character into an ASCII code
ATN	Arc tangent
AUTO	Provides an automatic line numbering facility for typing in listings
BIN\$	Converts decimal number to binary
BORDER	Changes the colour of the screen border
CALL	Calls a machine-code subroutine
CAT	Gives a catalogue of files on cassette
CHAIN	Command to load and run a program CHAIN- "filename"

CHAIN MERGE	As above but merges one program into another
CHR\$	Converts an ASCII code into a character
CINT	Rounds up a figure to the nearest integer
CLEAR	Clears the memory of all variables previously used
CLG	Clears the graphics screen
CLOSEIN	Closes cassette input file
CLOSEOUT	Closes cassette output file
CLS	Clears the screen
CONT	Continue program
COS	Cosine
CREAL	Converts value to real number
DATA	The store of information which is taken by the READ statement
DEF FN	Defines a function
DEFINT	Defines integer variable
DEFREAL	Defines real variable
DEFSTR	Defines string variable
DEG	Degrees
DELETE	Used to delete lines from a program, e.g. DELETE 10-60
DI	Disable interrupts
DIM	Dimensions the size of an array
DRAW	Draws a line between specified points
DRAWR	Draws a line to a position relative to the graphics cursor position

EDIT	Edit a line number
EI	Enable interrupts
ELSE	Used in conjunction with IF-THEN to branch to another action if the condition is not fulfilled, e.g. IF counter > 20 THEN GOTO 50 ELSE GOTO 100
END	Tells the computer to terminate running a program
ENT	Tone Envelope
ENV	Volume Envelope
EOF	End of file
ERASE	Clears specified arrays
ERL	Gives the line number where the last error occurred
ERR	Returns the error number of the last error
ERROR	Used in conjunction with error number
EVERY	Used with internal timers to GOTO a subroutine at EVERY given interval
EXP	Calculates e to a given power
FIX	Removes numbers to right of decimal point
FOR	The start element of the FOR-NEXT loop
FRE	Provides information on amount of free memory
GOSUB	Sends program to a subroutine at a specified line number
GOTO	Sends the computer to a specified line in the program
HEX\$	Converts decimal to hexadecimal

HIMEM	Returns value of highest byte in BASIC memory
IF	Start of the IF-THEN statement
INK	Changes ink to a specified colour
INKEY	Checks keyboard for depression of key(s)
INKEY\$	Takes the input of a key from the keyboard
INP	Gives value from I/O port
INPUT	Issues a request for either numbers or strings of characters to be entered from the keyboard
INT	Converts a number to the nearest smaller integer
INSTR	Searches for the occurrence of a string within another string
KEY	Defines new function key
KEY DEF	Defines the value of a key when pressed
JOY	For use with joysticks
LEFT\$	String manipulation command for taking a number of given characters from the start of a string
LEN	Returns a number giving the length of a string
LINE INPUT	As for INPUT but ensures everything typed in is held in the variable
LIST	Lists to the screen all or part of a program
LOAD	Loads a program from cassette or disk into the computer's memory
LOCATE	Moves the cursor to a specified position on screen, e.g. LOCATE 1,5
LOG	Logarithm
LOWER\$	Converts upper-case characters (capitals) to lower case

MAX	Finds largest value
MEMORY	Resets BASIC memory
MERGE	To merge a program
MID\$	String manipulation command for taking a number of given characters from a specified position in a string
MIN	Finds smallest value
MODE	Sets the screen display mode
MOVE	Moves graphics cursor
NEW	Clears the memory of the computer
NEXT	Specifies the end of a FOR-NEXT loop
ON	Enables re-direction of program by altering the order of execution e.g. ON a GOTO 25,45 ON ERROR GOTO 10
ON SQ GOSUB	To enable an interrupt when there is a free space in the sound queue
OR	Logical 'or' operator
OPENIN	Opens a cassette input file
OPENOUT	Opens a cassette output file
ORIGIN	Sets the starting point for the graphics cursor
PAPER	Sets the colour of the paper (background)
PEEK	Enables user to look at contents of RAM
PEN	Sets the colour for the characters (foreground)
PI	The value of pi (3.14159...)

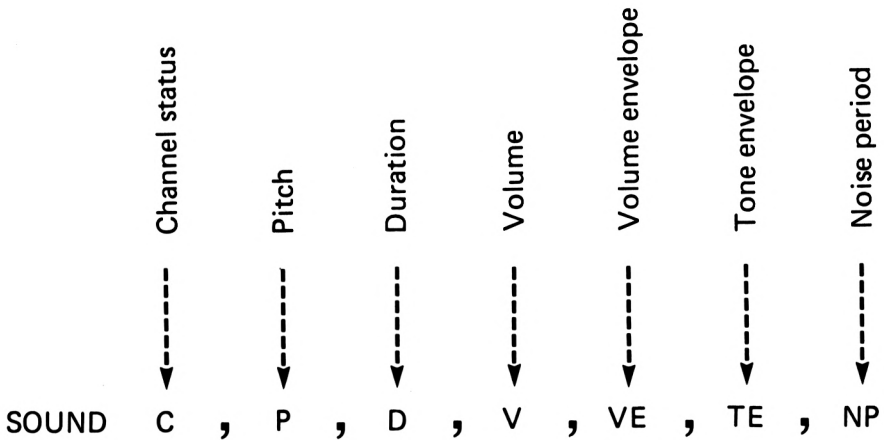
PLOT	Plots pixel points
POKE	Pokes values into memory addresses
PRINT	Prints given items on the screen
RAD	Radians
RANDOMIZE	Sets the initial value for RND generator
READ	Reads the information contained in the DATA statements
RELEASE	Releases 'hold' on sound channels
REMAIN	Disables an internal delay timer if set
RENUM	Rennumbers the lines of a program listing
RESTORE	Sets pointer to read data from a specified position
RESUME	Used after ON ERROR
RETURN	Marks the end of a subroutine
RIGHT\$	String manipulation command that takes a number of characters from the end of a string, working from right to left
RND	Chooses a random number
ROUND	Rounds number to a specified number of decimal places
RUN	Runs the program
SAVE	Saves a program to cassette or disk
SGN	Returns a value indicating the sign of a numeric value (positive or negative)
SOUND	Covered in this book!
SPACE\$	Places a string of spaces
SPEED INK	Sets time for first and second inks

SPEED KEY	Sets auto-repeat rate of keys
SPEED WRITE	Determines the speed at which a cassette file is SAVED
SPC	Places any number of specified spaces on the screen
SQ	Checks the free space in the sound queue
SQR	Square root
STEP	Specifies a step within the FOR-NEXT statement
STOP	Stops a program and displays the line number
STR\$	Converts a number into a character string, e.g. 1 becomes '1'
SYMBOL	Redefines character set
TAB	Used with PRINT to move the screen cursor to a specified position
TAG	Allows text to be written at graphics cursor point
TAGOFF	Turns TAG command off
TAN	Tangent
TESTR	Returns a value indicating the ink being used at a specified screen position
THEN	Used in conjunction with the IF statement
TIME	Gives the amount of time passed since the machine was switched on
TO	Used in conjunction with the FOR-NEXT loop to specify a numeric range
TRON	Trace facility
TROFF	Turns trace facility off
UPPER\$	Converts lower-case characters to upper case

VAL	Converts a number in a character string into numeric form, e.g. '8' becomes 8
VPOS	Gives the vertical position of the text cursor
WEND	Terminates the WHILE loop
WHILE	Creates a loop while a certain condition is true
WINDOW	Specifies the text window
XPOS	Gives horizontal position of graphics cursor
YPOS	Gives vertical position of graphics cursor
ZONE	Sets the width of the print zone

Appendix B

SOUND Parameters



SOUND CHANNEL VALUES

<u>Value</u>	<u>Channel(s) / Effect</u>
1	Sound Channel A
2	Sound Channel B
4	Sound Channel C
8	Rendezvous with A
16	Rendezvous with B
32	Rendezvous with C
64	Hold
128	Flush

DURATION VALUES

Default: 20
 Range: -32768 to +32767

<u>Value</u>	<u>Effect</u>
> 0	Length of Sound
= 0	Duration controlled by Volume Envelope
< 0	No. of times Volume Envelope Repeated

Appendix C Volume Envelope Parameters

ENV n,P1,Q1,R1,P2,Q2,R2,P3,Q3,R3,P4,Q4,R4,P5,Q5,R5

n = envelope number

Pn = step count (0 to 127)

Qn = step size (-128 to 127)

Rn = pause time (0 to 255)

Minimum = one section

Maximum = five sections

If a section is defined it must contain all three categories.

If no section is defined (e.g. ENV 1) then the envelope will be cancelled.

Appendix D Tone Envelope Parameters

ENT n,T1,V1,W1,T2,V2,W2,T3,V3,W3,T4,
V4,W4,T5,V5,W5

n = envelope number

Tn = step count (0 to 239)

Vn = step size (-128 to +127)

Wn = pause time (0 to 255)

Minimum = one section

Maximum = five sections

If a section is defined it must contain all three categories.

If no section is defined (e.g. ENT 1) then the envelope will be cancelled.

Appendix E

Note, Frequency and Pitch Parameter Table

Middle C = 1 (octave 0)

n	Note	Frequency	Pitch	Octave parameter (P)
-35	C	32.703	3822	-3
-34	C#	34.648	3608	-3
-33	D	36.708	3405	-3
-32	D#	38.891	3214	-3
-31	E	41.203	3034	-3
-30	F	43.654	2863	-3
-29	F#	46.249	2703	-3
-28	G	48.999	2551	-3
-27	G#	51.913	2408	-3
-26	A	55.000	2273	-3
-25	A#	58.270	2145	-3
-24	B	61.735	2025	-3
-23	C	65.406	1911	-2
-22	C#	69.296	1804	-2
-21	D	73.416	1703	-2
-20	D#	77.782	1607	-2
-19	E	82.407	1517	-2
-18	F	87.307	1432	-2
-17	F#	92.499	1351	-2
-16	G	97.999	1276	-2
-15	G#	103.826	1204	-2
-14	A	110.000	1136	-2

n	Note	Frequency	Pitch	Octave parameter (P)
-13	A#	116.541	1073	-2
-12	B	123.471	1012	-2
-11	C	130.813	956	-1
-10	C#	138.591	902	-1
-9	D	146.832	851	-1
-8	D#	155.564	804	-1
-7	E	164.814	758	-1
-6	F	174.614	716	-1
-5	F#	184.997	676	-1
-4	G	195.998	638	-1
-3	G#	207.652	602	-1
-2	A	220.000	568	-1
-1	A#	233.082	536	-1
0	B	246.942	506	-1
+1	C	261.626	478	0
+2	C#	277.183	451	0
+3	D	293.665	426	0
+4	D#	311.127	402	0
+5	E	329.628	379	0
+6	F	349.228	358	0
+7	F#	369.994	338	0
+8	G	391.995	319	0
+9	G#	415.305	301	0
+10	A	440.000	284	0
+11	A#	466.164	268	0
+12	B	493.883	253	0
+13	C	523.251	239	1
+14	C#	554.365	225	1
+15	D	587.330	213	1
+16	D#	622.254	201	1
+17	E	659.255	190	1
+18	F	698.457	179	1
+19	F#	739.989	169	1
+20	G	783.991	159	1
+21	G#	830.609	150	1
+22	A	880.000	142	1
+23	A#	932.328	134	1
+24	B	987.767	127	1
+25	C	1046.502	119	2
+26	C#	1108.731	113	2

n	Note	Frequency	Pitch	Octave parameter (P)
+27	D	1174.659	106	2
+28	D#	1244.508	100	2
+29	E	1318.510	95	2
+30	F	1396.913	89	2
+31	F#	1479.978	84	2
+32	G	1567.982	80	2
+33	G#	1661.219	75	2
+34	A	1760.000	71	2
+35	A#	1864.655	67	2
+36	B	1975.533	63	2
+37	C	2093.004	60	3
+38	C#	2217.461	56	3
+39	D	2349.318	53	3
+40	D#	2489.016	50	3
+41	E	2637.021	47	3
+42	F	2793.826	45	3
+43	F#	2959.955	42	3
+44	G	3135.963	40	3
+45	G#	3322.438	38	3
+46	A	3520.000	36	3
+47	A#	3729.310	34	3
+48	B	3951.066	32	3
+49	C	4186.009	30	4
+50	C#	4434.922	28	4
+51	D	4698.636	27	4
+52	D#	4978.032	25	4
+53	E	5274.041	24	4
+54	F	5587.652	22	4
+55	F#	5919.911	21	4
+56	G	6271.927	20	4
+57	G#	6644.875	19	4
+58	A	7040.000	18	4
+59	A#	7458.621	17	4
+60	B	7902.133	16	4

Appendix F The Sound Chip—Technical notes

These notes are intended for the more technically inclined and those who will delve into the operating system of the Amstrad. They are not a complete rundown on the Amstrad firmware, but will give those who are interested a head start in disassembling the complexities of the sound chip.

The sound generator chip supplied within the Amstrad is a General Instruments AY-3-8912 which generates square wave-form sounds. The pitch generated is given by the period of the note and this period is incremented in steps of 8 microseconds.

The volume envelope can also have its sections defined either by hardware or by software selection. The software definitions we have covered in the main part of the book. If hardware definition is used then values can be written into registers 11, 12 and 13 of the sound chip in order to create a hardware envelope. The three sound channels are set up by amplitude control registers (registers 8 to 10). If a hardware envelope is going to be used in the defined channel, Bit 4 of the corresponding amplitude control register is set. If this bit has not been set then the volume is controlled by bits 0 to 3 of this register.

Register 13 (bits 0 to 3) control the shape of the envelope. Eight hardware envelopes are possible. These are:

- 8: Repeated sharp rise and sloping fall.
- 9: Sharp rise followed by a sloping fall and then sustained at zero amplitude.
- 10: Sharp rise followed by repeated sections of sloping fall and sloping rise.
- 11: Sharp rise, sloping fall, sharp rise and then sustained at maximum amplitude.
- 12: Repeated sloped rise and sharp fall.

- 13: Sloped rise and then sustained at maximum amplitude.
- 14: Repeated sloped rise and fall.
- 15: Sloped rise followed by a sharp fall and then sustained at zero amplitude.

An envelope period determines the length of the slopes. The period is a 16-bit number with the LSB being held in register 11 and the MSB in register 12. These periods are the time intervals between slope steps and are in units of 128 microseconds.

Register 7 determines whether noise is to be used or tone. Bits 0 to 2 disable the tone in channels A to C respectively, and bits 3 to 5 disable noise on channels A to C.

The noise generator produces a pseudo-random noise, and the data register for this is register 6. Registers 0 to 5 are the tone generators; each sound channel (channels A to C) has two tone period registers, one being a fine-tune register and the other a coarse-tune register.

Listed below you will find a list of routines and their addresses that can be called from the operating system by the user.

BCA7	Resets sound chip and clears all sound queues
BCAA	Adds a sound to sound queue(s) of channel(s)
BCAD	Checks to see if there are free spaces in the sound queue
BCB0	When queue is not full an event is set up
BCB3	Releases sounds on channels
BCB6	Halts sound
BCB9	Continues sounds after BCB6 has been invoked
BCBC	Sets up a volume envelope
BCBF	Sets up a tone envelope
BCC2	Interrogates for information as to where the address of a volume envelope is
BCC5	As above but for a tone envelope

Other titles of interest

- On the Road to Artificial Intelligence: Amstrad CPC 464** £5.95
Jeremy Vine
- Gateway to Computing with the Amstrad CPC 464 (each)** £4.95
Ian Stewart
Two books covering the fundamentals of computing for children.
- The Complete Introduction to the Amstrad CPC 464** TBA
Eric Deeson
- Computers in a Nutshell** £4.95
Ian Stewart
The layman's introduction to computing.
- Brainteasers for BASIC Computers** £4.95
Gordon Lee
'A book I would warmly recommend'—*Computer & Video Games*
- Microchip Mathematics: Number Theory for Computer Users** £12.95
Keith Devlin
- Programming for REAL Beginners: Stages 1 & 2 (each)** £3.95
Philip Crookall

ORDER FORM

I should like to order the following Shiva titles:

Qty	Title	ISBN	Price
___	ON THE ROAD TO ARTIFICIAL INTELLIGENCE: AMSTRAD CPC 464	1 85014 064 2	£5.95
	GATEWAY TO COMPUTING WITH THE AMSTRAD CPC 464		
___	BOOK ONE	1 85014 016 2	£4.95
___	BOOK TWO	1 85014 023 5	£4.95
___	THE COMPLETE INTRODUCTION TO THE AMSTRAD CPC 464	1 85014 002 2	TBA
___	COMPUTERS IN A NUTSHELL	1 85014 018 9	£4.95
___	BRAINTEASERS FOR BASIC COMPUTERS	0 906812 36 4	£4.95
___	MICROCHIP MATHEMATICS	1 85014 047 2	£12.95
___	PROGRAMMING FOR REAL BEGINNERS: STAGE 1	0 906812 37 2	£3.95
___	PROGRAMMING FOR REAL BEGINNERS: STAGE 2	0 906812 59 3	£3.95
___
___
___

Please send me a full catalogue of computer books and software:

Name

Address

.....

This form should be taken to your local bookshop or computer store. In case of difficulty, write to Shiva Publishing Ltd, Freepost, 64 Welsh Row, Nantwich, Cheshire CW5 5BR, enclosing a cheque for £

For payment by credit card: Access/Barclaycard/Visa/American Express

Card No Signature

Want to be a music composer, conductor, player and audience with your Amstrad?

With the help of Jeremy Vine you can join in the banging, hooting and music-making on this powerful and versatile computer. In a lively and cheerful style Jeremy encourages you to master the sound and envelope commands by practical experiment – neighbours be warned!

No knowledge of musical theory is assumed and not much of BASIC either – but as you whistle your way through the pages of this book you will find out about:

- composing and making music
- converting your Amstrad into a synthesizer
- the inner workings of the Amstrad sound chip

You will learn how to make all kinds of special sound effects:

- ringing telephones
- explosions
- alarms and sirens
- musical scales

and, of course,

- bells and whistles

Sounds OK!



Shiva Publishing Limited

UK price £4.95 net

GB £ NET +004.95

ISBN 1-85014-063-4



9 781850 140634

Shiva Publishing Limited



AMSTRAD

CPC



MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA



<https://acpc.me/>

[FRA] Ce document a été préservé numériquement à des fins éducatives et d'études, et non commerciales.

[ENG] This document has been digitally preserved for educational and study purposes, not for commercial purposes.

[ESP] Este documento se ha conservado digitalmente con fines educativos y de estudio, no con fines comerciales.