

Pan/Personal Computer News  
Computer Library

**Ian Ritchie**

# **BBC Micro Music Masterclass**

Pan Books London and Sydney

---

## **Contents**

### **Da Capo**

1. **The SOUND Statement** 11
2. **The ENVELOPE Statement** 28
3. **The Sound of Noise** 43
4. **More on SOUND and ENVELOPE** 53
5. **Music Theory for Micros** 70
6. **Chords and Harmony** 85
7. **Music Graphics** 101
8. **Automatic Composition** 129
9. **Applications** 147
10. **Computers in Modern Music** 197
11. **Interfacing and Peripherals** 207
12. **Al Fine**

**Index** 237

**Front cover** **Back cover**

---

First published 1984 by Pan Books Ltd,

Cavaye Place, London SW10 9PG  
in association with Personal Computer News  
9 8 7 6 5 4 3 2 1  
© Ian Ritchie 1984  
ISBN 330 28673 0

Photoset by Parker Typesetting Service, Leicester  
Printed and bound in Great Britain by  
Richard Clay (The Chaucer Press) Ltd, Bungay, Suffolk

This book is sold subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out or otherwise circulated without the publisher's prior consent in any form of binding or cover other than that in which it is published and without a similar condition including this condition being imposed on the subsequent purchaser

---

Please note:

All of the programs in this book have been thoroughly tested on a BBC Model B microcomputer fitted with BASIC 2. Most programs will also work with the earlier BASIC 1. If you are in doubt about the version in your machine, please contact your local BBC Micro dealer.

---

# Da Capo

Over the past few years computers have increasingly come to feature in the creation of pop music, invading a new and exciting field of human creativity. Bands like Orchestral Manoeuvres In the Dark, The Human League and Depeche Mode have spearheaded this development through the use of synthesisers and drum machines. As a result, it has become acceptable for machines to play a major role in the creation of the modern record.

Take, for example, the popularity of the MC-4 Microcomposer, much favoured by Yazoo, which makes it unnecessary for a human being to lay even one finger on a musical instrument. This computer has completely taken over the mechanical business of playing an instrument, so that all the individual musician has to do is to provide the inspiration.

One of the most recent developments in the mastering and playing of instruments has been a trend away from the traditional specialised musical tool altogether. In their place, standard microcomputers are increasingly utilised both as composing aids and as sound producers in their own right. The Fairlight Computer Musical Instrument is a good example of the next generation of computers, which not only have a range of excellent musical software, but can also be put to work as word processors or calculators.

As a result of observing these trends and changes, I decided that it was about time someone wrote a book about the use of computers in music from a practical standpoint. It seemed only logical to base the book around a computer with good musical potential, sophisticated sound generation and reasonably straightforward interfacing capabilities. In addition, the computer needed to have a flexible and powerful version of the BASIC language and be in widespread use. Bearing all these prerequisites in mind, I came to the conclusion that the only possible candidate which satisfied the requirements was the BBC Microcomputer.

Once the best model was identified all that remained was for someone to put pen to paper (or fingers to word processor in my case). At this point it occurred to me that I was reasonably well qualified for the job, since I am musician by profession. Having started my playing career as a saxophonist, in recent years I have branched out into composing and keyboard programming. In the process of so doing, I found myself entrusted with the MC-4 programming on David Grant's album, as well as creating drum programming and supervising computer input on various records by Dee C. Lee, Steve Levine, SPK, Beggars and Co and Miro Miroe.

My musical interests are wide ranging, embracing pop, jazz and classical and, apart from music, my other endearing passion in life is computers, having started years ago on a Sinclair ZX81 and graduating to a BBC Model B. The BBC's versatile sound commands soon had me up to my ears in various

programs, many of which you will come across in the course of working your way through this book.

As I have said, one of the strengths of the BBC Microcomputer is the range and versatility of its sound commands. The SOUND command itself allows control over channel, pitch, volume and duration of note, while the complex ENVELOPE command has parameters too numerous to mention in a short introduction.

Nevertheless, there is a drawback to this very versatility since it is no simple matter to master the necessary techniques to get the best out of your BBC. Lesser computers go for the option of pre-defined scales and envelopes, which greatly facilitates mastering the superficial and limited uses of sound however, it is worth aiming higher, since none of these machines come close to emulating the breadth of application possible on the BBC.

This book sets out to bring that same breadth of application within the reach of every BBC owner, not only by supplying you with self-contained and entertaining programs, but also by clarifying the ins and outs of BBC BASIC'S SOUND and ENVELOPE statements, and indicating the multifarious uses to which they can be applied.

The first three chapters are therefore devoted exclusively to the language of sound on the BBC: Chapter One will explore the use of the SOUND command to play simple melodies and scales. Chapter Two deals with the somewhat complicated ENVELOPE statement. By breaking this rather confusing command into smaller, more manageable sections, its use and applications shine out much more clearly. Chapter Three delves into channel zero, the noise channel. This essential feature is used to create sound effects and percussive musical sounds.

Chapter Four contains more information about the SOUND and ENVELOPE statements and looks at how the remaining sound channels can be synchronised together.

At this point in the book, before going too deeply into crotchets, quavers and cadenzas, it seemed a good idea for us to go back to school in Chapter Five and learn a bit of music theory, so that all the potential Mozarts and Liszts out there can get off on the right foot! A rudimentary knowledge of music will prove tremendously valuable when we move on and look at auto-composition and harmonising melodies in later chapters.

Chapter Six deals with the problem of multi-part music. A computer is very good at dealing with sequential problems, however when a number of things happen simultaneously, as often happens in music, we must provide the computer with a sensible way of dealing with the information.

Chapter Seven will give you some ideas on the various methods of displaying music. Although music has a complex sign language of its own, microcomputers are ideally suited to embracing this language and presenting it in an exciting new way.



Chapter Eight touches on the interesting field of automatic composition, Can computers really create original, innovative music? You will be in a better position to judge the possibilities after RUNning the programs contained in this chapter.

Chapter Nine is intended to give you still further ideas about applications of music and sound. The programs in this chapter stand up by themselves, as well as illustrating methods and techniques which are explored elsewhere in the book.

Chapter Ten is intended as an inspiration to anyone interested in modern electronic music. Most, if not all, of the new modern instruments are built using the same microchip technology as that of the computer and hence every computer, especially the adaptable BBC, has the potential to be transformed into the equivalent of portions of the effective circuitries of instruments like the Fairlight CMI or the Casio VL-Tone.

Chapter Eleven has been designed to give you a start in the interesting area of interfacing. Another of the BBC's strong points is the breadth of interfacing possibilities. We will look at a number of useful but simple techniques, none of which require cartloads of additional hardware or, come to that, practical electronic skill.

In the final chapter I turn to contemplating what we can expect in the future. The microcomputer is coming to play a central part in the world of music just as it is infiltrating virtually every other field of human activity. I see this as an exciting development which promises a new era of music just around the corner...

# 1 The SOUND Statement

As I mentioned in the Introduction, the basic command used for producing any form of noise on the BBC is the SOUND command. In this chapter we will deal with the use of SOUND to produce music and, for simplicity's sake, we will temporarily ignore the fact that SOUND can also produce white noise and chords, and concentrate solely on the monophonic (one note at a time) case.

The format of the SOUND statement is:

```
SOUND C,A,P,D
```

where

C is the channel number.  
A is the amplitude (volume) of the note.  
P is the pitch (frequency) of the note.  
D is the duration (length) of the note.

The channel number c can take values from zero to three, but for the purposes of this chapter it will always be set to one. Channel one produces a square wave tone which can be heard by typing CTRL-G and holding both keys depressed.

The amplitude parameter A can take values between -15 and 15 but, again for the purposes of this chapter, we need only concentrate on the range zero to 15, which produces a range from silent to loud.

P can take values from zero to 255 and each single value is equivalent to 1/8th of a tone in musical terminology. A semitone (the basic musical building block) would therefore equal an increment of four.

The final parameter, D, can take any value between 0 and 255. In this case each increment is equal to 1/20th of a second, so a value of 20 for D would give a note length of one second.

These cold facts do not provide us with an accurate reflection of the vast possibilities inherent in the SOUND command. To see where we could use this command we must first explore the individual parameters and relate them in some way to everyday musical experience. Let us begin by looking at the effect of volume by trying the following short routine:

```
10 REM *** AMPLITUDE DEMO ***

20 FOR A=1 TO 15

30 SOUND 1, -A,53, 20

40 NEXT A
```

None of these volumes are exactly ear-splitting, but it is the range of dynamics available, rather than mere volume, that is important for most musical applications. We will, however, be looking at the potential for squeezing more decibels out of the sound chip in a later chapter.

The range of frequencies available can be heard by RUNning the following:

```
10 REM *** PITCH DEMO ***

20 FOR P=0 TO 255

30 SOUND 1, -10,P,3

40 NEXT P
```

At this juncture you might think that, as music goes, this program is not exhilaratingly exciting, so bear in mind that its function is to illustrate the range of P! The value of P is obviously one of the more important parameters for musical applications. Whoever designed the BBC Micro obviously knew this, because we can take the above program and simply change Line 20 to read:

```
20 FOR P= 0 TO 255 STEP 4
```

We can now hear intervals that are more familiar to most of us: these are known as semitones. Moving on, and using `RND(100)*4` to specify pitch, we can now persuade the BBC to compose rather erratic melodies:

```
10

20 REM *** RANDOM ***

40

50 REPEAT

60 P=RND (100)*4

90 SOUND 1, -10, P, 5

100 UNTIL P=999
```

Before we become totally involved in trying out and defining values for pitch, try RUNning the following routine which will give you an idea of duration:

```
10 REM *** DURATION DEMO ***

20 FOR D=1 TO 100

30 SOUND 1,-10,73,D

40 SOUND 1,0,0,D

50 NEXT D
```

Line 40 has been inserted to provide a period of silence between sounds. This is of equal duration to the sounds themselves. It should be noted at this point that setting the duration D equal to 255 will result in a note without end - presumably a device only of interest to admirers of John Cage!

The pitch parameter

Let us now return to pitch and relate values of P to musical notes by looking at the table below.

Note Name	Octave						
	1	2	3	4	5	6	7
B	1	49	97	145	193	241	
A#	0	45	93	141	189	237	
A		41	89	137	185	233	
G#		37	85	133	181	229	
G		33	81	129	177	225	
F#		29	77	125	173	221	
F		25	73	121	169	217	
E		21	69	117	165	213	
D#		17	65	113	161	209	
D		13	61	109	157	205	253
C#		9	57	105	153	201	249
C		5	53	101	149	197	245

The first column contains the notes listed as musical symbols. (Refer to Chapter Five for any problems you may have with musical terminology or notation.) The equivalent P values are listed in columns according to octave. These P

values are simply an arbitrary choice made by the BBC's programmer and therefore have no direct significance in themselves. Middle C has a value of 53 in this table and its actual measured frequency is 261.625Hz (cycles per second).

Another point to note before we go much further is the accuracy of P. Up to values of one hundred, P is basically equivalent to text book frequencies. Beyond this, however, the pitch has a tendency to drift, a vagary shared with most normal musical instruments. For most practical purposes this will not be a problem, so from now on let us give the tone generator the benefit of the doubt and treat it as if it has perfect pitch.

## Scales

One of the fundamental building blocks of music is the scale. Many of you will either remember, or still be experiencing, singing Doh, Re, Mes ad infinitum at school or in an organised music class - which is simply the traditional and rather crude method of instilling the Western harmonic scales into the brains of budding but reluctant Andre Previns. Using a computer is a much less painful method of learning scales and how they relate to Wham or the Eurythmics. (In this chapter we are dealing with how music in the real world relates to SOUND as it is played by the BBC, so it will be necessary for me to assume a certain knowledge of music theory. If any of the concepts or terminology are unfamiliar to you I strongly advise reading Chapter Five, Music Theory for Micros, before proceeding any further.)

The first scale we will look at is the scale of C major:

```

10 REM

20 REM    *** Scale at C Major ***

30 REM

40 FOR N=1 TO 4

50 REPEAT

60 READ A

70 SOUND 1, - 10, A, 10

80 UNTIL A= 101

90 RESTORE

100 NEXT

110 DATA 53,61,69,73,81,89,97, 101
```

This program uses a DATA statement to store the actual P values required for the scale. This method of remembering a sequence of notes will come in very handy when we want to program tunes at a later stage. Another method of playing the same scale is shown in the following program:

```

10 REM

20 REM *** SCALE-2 ***

30 REM

40 P=53

50 REPEAT

60 SOUND 1, -10, P, 10

70 READ increase

80 P=P+increase

90 UNTIL increase=0
```

```
100 DATA 8,8,4,8,8,8,4,0
```

In this example the DATA statement holds the increase in pitch. Using this method, you can change the key of the scale by the single expedient of changing P in Line 40. Thus for example:

If P=61 the scale would be D Major.

If P=77 the scale would be F# Major.

Let us now see this in practise and create a program which simulates violin practise in action:

```
10

20 REM *** Violin Practise ***

30

40 FOR p=53 TO 101 STEP 4

50 Pitch=P

60 REPEAT

70     SOUND 1, -10, Pitch, 10

80 READ increase

90     Pitch=Pitch+increase

100 UNTIL increase=0

110     RESTORE 130

120 NEXT P

130 DATA 8,8,4,8,8,8,4,0
```

This program has a major advantage over real violin practise in that the computer never makes a mistake. Anyone who has ever been subjected to listening to an inexperienced violinist will greatly appreciate this advantage.

In practise, you will discover that the scale program which uses actual pitch values is the universally used method, for the simple reason that, in practise, working out the increase or decrease in value for melodies more complex than steadily increasing scales becomes mind-bogglingly complex. Ease of use is a crucial criterion in all music programs since, ultimately, all the methods suggested in this book have been singled out to be used for creative ends. Bear in mind that spontaneity is essential to composition, so we should also aim to program with this in mind as far as practicably possible.

The DATA used in the scale of C Major could easily be used to convert the computer into a simple musical keyboard. The following program allows you to play the notes of the C Major scale using keys 1 to 8.

```
10

20 REM     *** KEYBOARD ***

30

40 DATA 53,61,69,73,81,89,97,101

50 DIM P(8)

60 FOR N=1 TO 8

70     READ P(N)

80     NEXT N
```

```

90 P$=INKEY$(0)

100 IF P$=" " THEN GOTO 90

110 SOUND 1,-15,P(VAL(P$)),2

120 GOTO 90

```

The DATA is first READ into the array P(s). The keyboard is then scanned for a key press using the INKEY\$ statement in Line 90. Since the bracketed number is zero, the computer does not wait at Line 90 but immediately returns to Line 100, which in turn sends the computer back to 90 if no key has been pressed. If a key press is detected the program jumps to Line 110 and a note that is equivalent to the value of the number pressed is played. For example; If 1 is pressed then P(1)=53, therefore C is SOUNDED, and if 5 is pressed then P(5)=81, therefore G is SOUNDED. This program could be extended to produce the full chromatic scale demonstrated earlier:

```

10

20 REM   *** Chromatic Keyboard ***

30

40 K$="ZSXCFVGBNJMK,L./"

50 *FX11,1

60 *FX12,1

70 POTE$=GET$

80 REPEAT

90 P=INSTR(K$,POTE$)*4+37

100 SOUND 1,-10,P,-1

110 REPEAT

120 note$=INKEY$(2)

130 UNTIL note$<>POTE$

140 SOUND &11,0,0,0

150 IF note$=" " THEN POTE$=GET$ ELSE POTE$=note$

160 UNTIL POTE$=" "

170 *FX12,0

180 END

```

This program uses the array K\$ to store its key names. Line 90 then returns a value for P (pitch) by using the INSTR command to obtain the position of POTE\$ in K\$. This is then returned as a number from 1 to 16 (sixteen is the number of elements in K\$). P is then derived by multiplying by 4, and adding 37. Thus if C is pressed POTE\$=C, the INSTR expression will return 4 (i.e. 4th in the array), and P will have the value of (4\*4)+37=53 (Middle C).

The \*FX commands are used to speed up the auto repeat. \*FX 11,1 sets the delay before repeats start to 1/100 of a second. \*FX 12,1 sets the repeat period to 1/100th of a second as well. Note that the space bar must be pressed to END the program. If you ESCAPE from the program before Line 170 the new auto repeat values will stay in operation. This will make it very difficult to type any further program information. In Line 170 the statement \*FX12,0 resets both the auto repeat and the auto repeat delay values to normal.

The chromatic scale program serves to illustrate the point that a typewriter keyboard is not the easiest musical device to play. Segovia is unlikely to ever trade in his guitar for a BBC and a copy of the above program! This is not to

deny the fact that the computer does have a useful role to play in creating music. This is due to the computer's unique memory and its capacity to repeat series of events accurately and consistently without feeling the human emotion of boredom or exhibiting human fallibility. One obvious use for this talent is in controlling a sequencer.

The sequencer is a fairly recently developed phenomenon which was originally created for controlling analogue synthesisers. The first sequencers were clumsy modular machines. Each module was set to generate a particular voltage and each voltage was triggered in sequence using a low frequency oscillator (LFO). These voltages were then fed into the synthesiser, which played notes related to the voltages at a speed controlled by the LFO.

Any system which consists of a number of identical modules which work in sequence is obviously very inefficient. It soon struck the designers that there should be some way of remembering the list of voltages so that only one voltage generator would be required. After a period of tinkering and experimentation computer controlled sequencers were born. The logical extension of solely storing note values is to remember timings as well. Once both pitch and note length are recorded the sequencer becomes a very flexible musical tool.

At present sequencers are enjoying great popularity amongst musicians of all persuasions. Depeche Mode, the Human League and Yazoo, to name but a few of the better known bands committed to sequences, would be completely helpless in the studio without the assistance of such machines as the Roland MC-4 Microcomposer and the Fairlight CMI.

The above mentioned machines are not musical instruments in the standard sense of the word, but are in fact complete music compositional system. (For a fuller description of modern electronic instruments see Chapter Ten.) Both are basically computers that have been programmed to control external synthesisers (in the case of the MC-4) or control and generate sampled sounds (in case of the Fairlight CMI). At present, then, these instruments represent the state of the art of professional machines. Although it is beyond the scope of this book to break down and dissect their wide-ranging facilities, we will go a long way towards indicating both how this type of system works and the type of software that makes it tick. The following program serves to illustrate the basic workings of a simple sequencer:

```

5  REM

6  REM      *** SEQUENCE ***

7  REM

10 MODE6: VDU 19,0,4,0,0,0

20 DIM

30 CLS

35 REM      Enter Pitch" A(N)

36 REM

40 FOR N=1 TO 8

50 INPUT "Enter Pitch" A(N)

60 SOUND 1,-10,A(N), 10

70 NEXT

75 REM      Play Sequence

76 REM

60 FOR N=1 TO 8

90 SOUND 1,-10,A(N),10

100 NEXT N
```

The program first sets up the array A(8) to hold the pitch information. An INPUT statement then asks for pitches to be entered. When all eight pitches have been entered the sequence plays four times then stops.

"Sequence" has a number of obvious limitations. Pitches must be entered as P values (i.e. adding four for every semitone) so the note/pitch table has to be referred to constantly. In addition, exactly eight notes must be entered and no variation in the tempo or the number of times the sequence is played is possible.

These drawbacks make "Sequence" less than a powerful musical tool, but don't lose heart or believe I've led you up a blind alley. By constantly refining this program as we learn more about the SOUND statement, we will eventually end up with a considerably more flexible program.

Before going any further with the sequencer program let us take a further look at DATA statements as a way of storing musical information. The following examples take the ideas explored in the scale programs and apply them to that well known Scandinavian melody 'Hall of the Mountain King'.

```

10 REM

20 REM *** HALL ***

30 REM

40 REPEAT

50 READ P

60 IF P=0 THEN END

70 SOUND 1,-10,P,5

80 UNTIL FALSE

90

100 DATA 61,69,73,81,89,73,89,89,85,69,

85,85,81,65,81,81,61,69,73,81,89,73,89,

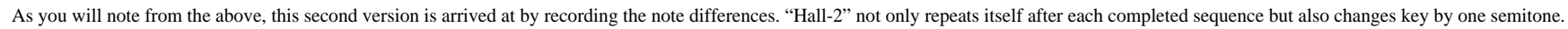
109,101,89,73,89,101,101,101,101,0

```

“Hall” uses the form of our first scale program but also incorporates a small amount of rhythmical variation by repeating DATA to produce a longer note. Using this method, one DATA entry must equal the shortest note in the place. In this case one entry equals a quaver (1/8 note). The following diagram shows how the numbers in the DATA statement relate to conventional musical notation:

## HALL OF THE MOUNTAIN KING





file:///C:/Documents%20and%20Settings/Chris%20Richardson/Desktop/New%20Folder/bbc\_mm\_01.htm (8 of 23)20/06/2012 08:46:03

```
160 UNTIL FALSE
```

```
170 DATA 0,8,4,8,8,-16, 16,0,-4,-16,16,
```

```
0,-4,-16, 16,0,-20,8,4,8,8,-16,16,20,-8, -12, -16, 16,12,0,0,0,1
```

The value of P, in Line 40, gives the starting note which, in this case, is D. Two REPEAT... UNTIL loops are used to cycle the program. On completion of each cycle a semitone increment of four is added to P (Line 70) which results in a semitone shift. This program still uses multiple entry of P values to give rhythmical variation. There are more satisfactory ways of achieving this, one of which would be to record values for the duration parameter D as separate items of DATA.

## The duration parameter

It is obviously time to say that, in many forms of music, rhythm is as important a factor to consider as pitch. The duration parameter D is the key to providing our control of rhythm on the BBC. In reality, D actually dictates two separate musical quantities, tempo and timing.

The tempo of a piece of music is defined as its speed in beats per minute. The length of one crotchet is equal to the length of one beat, so for a tempo of 120bpm we would hear 120 crotchets go by. Since o is given in 1/20ths of a second the value for D for a crotchet at 120bpm would be given by:

120bpm=120/60 beats per second=2 bps.

As calculated earlier, D=20 gives a duration of 1 second, and therefore the value of D for one beat would equal  $20/2=10$ . The general formula is  $D=20/\text{Tempo}/60$ , or  $D=1200/\text{Tempo}$  where Tempo is given in beats per minute.

At this juncture let me introduce a program called “Metronome”, an electronic metronome program with a difference. Instead of entering values for the tempo in beats per minute, you are asked to supply a value for D. Once the metronome is running the tempo can be changed using the up- and down- cursor keys. This program illustrates the important point that, since only integer values of D can be entered and a value of 1 for D = 1/20th of a second at fast tempos, only certain values for TEMPO are available.

This point can be further illustrated by looking at an example. If TEMPO is 120 the value of D is 10 (1200/120), whereas if TEMPO were 110, then  $D=1200/110=10.9$  which is not an integer value and would be interpreted by the SOUND statement as 10, rounding down.

```
10 MODE7
```

```
20 VDU23:8202:0:0:0;
```

```
30 PRINT TAB(5,4);CHR$(129);CHR$(141);" * * * ";CHR$(132);"METRONOME";CHR$(129); " * * * "
```

```
40 PRINT TAB(5,5);CHR$(129);CHR$(141);" * * * ";CHR$(132);"METRONOME";CHR$(129); " * * * "
```

```
50 PRINT TAB(7,13);CHR$(131);"TEMPO"; TAB(23,13);"Duration"
```

```
60 PRINT TAB(2,20);CHR$(134);"Press cursor up to increase TEMPO"
```

```
70 PRINT TAB(2,21);CHR$(134);"Press cursor down to decrease TEMPO"
```

```
80 PRINT TAB(2,22);CHR$(134);"Press cursor left to restart"
```

```
90 PRINT TAB(2,23);CHR$(134);"Press cursor right to END"
```

```
100 *FX4,1
```

```
110 ENVELOPE 1,1,0,0,0,0,0,0,126,-6,0,-4,120,0
```

```
120 REPEAT
```

```
130 INPUT TAB(8,8)"STARTING D VALUE",D
```

```
140 correct=D>0 AND D<255
```

```
150 IF NOT correct THEN PRINT TAB(25,8);"?? try again"
```

```
160 UNTIL correct
170 PRINT TAB(8,8)"
"
180 REPEAT
190 TEMPO=INT(1200/D)
200 PRINTTAB(9,15);TEMPO;"
";D;"
"
210 SOUND 1,1,100,D
220 K=INKEY(0)
```




---







## 22 The SOUND Statement

```
230 IF K=139 THEN D=D-1:IFD=0 LET D=1
240 IF K=138 THEN D=D+1
250 IF K=136 THEN SOUND&11,0,1,1:RUN
260 UNTIL K=137
270 DEF PROCend
280 *FX4,0
290 VDU23;8202;0;0;0;
300 CLS
310 PRINT TAB(10,10)"THAT'S ALL"
320 END
330 ENDPROC
```

The program above uses a number of aspects of BBC sound programming which we have not covered as yet. These will be explored in a later chapter where I'll also incorporate a more flexible version of the “Metronome” program.

The choice of a basic value for D is similar to choosing a ‘Time Base’ in programs such as that utilised by the MC-4 Microcomposer. A Time Base is defined as the number of subdivisions in one crotchet of music. A commonly used value is 48, i.e. at a TEMPO of 120 each subdivision would be 1/96th of a second. The following table lists the permitted TEMPOs, along with the rhythms that can be used at that TEMPO.

Note Value	Tempo						
	92	100	109	120	133	150	171
1 	52	48	44	40	36	32	28
$\frac{1}{2}$ 	26	24	22	20	18	16	14
$\frac{3}{4}$ 	—	18	—	15	—	12	—

8		13	12	11	10	9	8	7
$\frac{1}{4}$		—	6	—	5	—	4	—
$\frac{1}{8}$		—	4	—	—	—	—	—
$\frac{1}{12}$		—	3	—	—	—	2	—
$\frac{1}{16}$		—	—	—	—	—	1	—
$\frac{1}{32}$		—	—	—	—	—	—	—

Once a basic value of D for one crotchet has been determined it becomes possible to calculate the timing of a particular piece of music. For example, if TEMPO=120:

D	Musical Symbol
5	quaver
10	crotchet

10	crocnet
20	minim
40	semibreve

Note that the fact that the smallest increment of D which can be defined is 1/20th of a second seriously undermines the musical utility of the Duration parameter. As a result another method of introducing rhythm into music must be found for any truly flexible musical purposes. The best ways of doing this will be explored in a later chapter, so for the time being let us make the most of D and accept the TEMPO that the SOUND statement dictates for a tune, rather than the other way round.

The program “Amazing Grace” which follows uses DATA statements to store pitch and duration. In this example various values for D were experimented with until the tune ‘felt right’. Although this hardly appears to be a scientific method of programming unless you are incorporating a Tempo PROCedure into your program it is probably the easiest way of arriving at the correct speed for the music. Once a basic value of D has been chosen the rest of the timings can be determined relative to this same basic value.

```
20 REM          *** AMAZING GRACE ***
30
40 REPEAT
50 REPEAT
60 READ F%,D%
70 SOUND 1,-10,F%,D%
80 UNTIL F%=0 AND D%=0
90 RESTORE
100 UNTIL FALSE
110
120 DATA 33,12,53,24,69,4,61,4,53,4,69
,24,61,12,53,24,41,12,33,36,53,24,69,4,6
1,4,53,4,69,24,61,12,81,60
130 DATA 69,12,81,24,69,4,61,4,53,4,69
,24,61,12,53,24,41,12,33,36,53,24,69,4,6
1,4,53,4,69,24,61,4,69,4,61,4,53,60,0,0
```

---

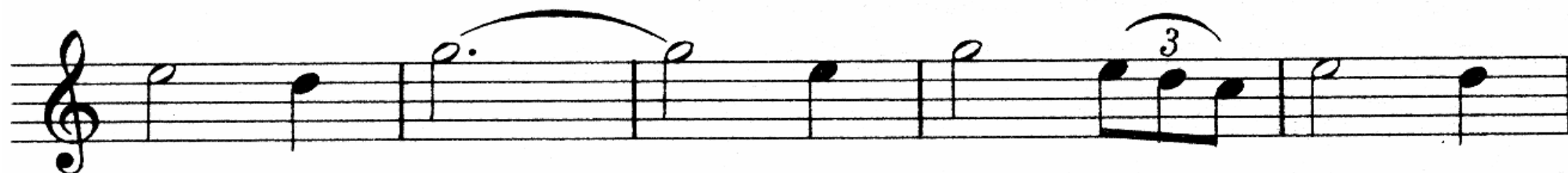
“Amazing Grace” is in 3/4 time, that is to say three beats to the bar. A value of D equal to 12 for a one crotchet duration was chosen both because it seemed a reasonable tempo and also because this number is easily divided by three. As a result, the crotchet triplets in the first bar have D values of 4.

## AMAZING GRACE

# AMAZING GRACE



Pitch: 33 53 69 61 53 69 61 53 41 33 33 53 69 61 53  
 Duration: 12 24 4 4 4 24 12 24 12 24 12 24 4 4 4



69 61 81 69 81 69 61 53 69 61  
 24 12 (60 ) 12 24 4 4 4 24 12



53 41 33 33 53 69 61 53 69 61 69 61 53  
 24 12 24 12 24 4 4 4 24 4 4 4 60

At this juncture we could also incorporate variable durations into our sequencer program:



T

```
10
20  REM      *** SEQUENCER-2" ***
30
35  REPEAT
40  CLS
50  INPUTTAB(10,10); "HOW MANY NOTES",

60  CLS
65  UNTIL T>0 AND T<2000
70  DIM A(T)
80  DIM D(T)
90  FOR N=1 TO T
94      spaces=N MOD 13
95      IF N MOD 13=0 THEN CLS
100      PRINTTAB(0,9+spaces);N
110      PRINTTAB(0,8)"N          Pitch
      Duration"
```

```
120     INPUTTAB(8,9+spaces),A(N)
130     SOUND 1,-1,A(N),10
140     INPUTTAB(24,9+spaces),D(N)
150     NEXT
160  CLS
170  INPUTTAB(10,10)"HOW MANY TIMES",X
180  FOR G=1 TO X
190    FOR N=1 TO T
200      SOUND 1,-1,A(N),D(N)
210    NEXT
220  NEXT
```

“Sequencer–2” is slightly more flexible than the mark one version in that, as well as having an array D(T) for storing D values, it also lets you vary the length of the sequence. Note that Line 170 also asks the user ‘How many times do you wish the sequence to be played?’. At this point in time we still have a long way to go before we arrive at a really useful sequencer program, but “Sequencer–2” contains all the fundamental components we will require later, bar one – a method of introducing rests!

## The amplitude parameter

The amplitude parameter can be used for introducing dynamics into a piece of music. As you will have noticed yourselves, the first beat of a bar is often emphasised in music in order to create a regular pulse or lilt. The following program uses an accented first note to create a waltz-time feel.

```

10
20  REM"****WALTZ****
30
40  REPEAT
50      READ Pitch%,Duration%,Amplitude
%
60      SOUND 1,-Amplitude%,Pitch%,Duration%
70  UNTIL Pitch%=0
80  END
90
100
110  DATA 69,10,15,69,20,13,81,10,15,8
1,10,13,89,10,13

```

```

120 DATA 89,10,15,89,50,13
130 DATA 73,10,15,73,20,13,89,10,15,8
9,10,13,97,10,13,97,10,15,97,50,13

```

---

## 26 The SOUND Statement

```

140 DATA 109,10,15,109,20,13,101,10,1
5,101,10,13,69,10,13
150 DATA 81,10,15,81,20,13,73,10,15,7
3,10,13,69,10,13
160 DATA 69,10,15,69,20,13,61,10,15,6
1,20,13
170 DATA 53,10,15,69,10,13,81,10,13,1
01,30,15
180 DATA 0,0,0,0,0,0,0,0,0,0,0

```

One further use for amplitude in the SOUND statement is, of course, the use of rests. A rest is a specified musical length where no note is heard. In the SOUND statement's terms this is easily done by making the amplitude A equal to zero. The following program demonstrates how you can introduce

rests into a program without having to create a separate array in which to store the amplitude values.

```
10
20 REM      *** TUNE ***
30
40 REPEAT
50 FOR N=1 TO 38
60 READ P,D
70 IF P=0 THEN A=0 ELSE A=-15
80 SOUND1,A,P,D
90 NEXT
100 RESTORE
110 UNTIL FALSE
120 DATA 33,16,41,4,53,4,0,6,69,18,0,2
,69,4,0,6,69,10,61,4,0,6,53,3,0,7
130 DATA 61,10,53,10,41,10,33,16,41,4,
53,4,0,6,69,18,0
140 DATA 2,69,4,0,6,69,10,61,4,0,6,53,
3,0,7,61,10,53,10,41,10
150 DATA 33,16,41,4,53,4,61,2,0,4,53,6
```



When a zero value for P is READ Line 70 specifies a value of zero for the Amplitude parameter, i.e. no audible sound. Note that as well as rests, this program also uses zero values to create varied phrasing of the notes. This technique will be developed upon and further explored in later programs.

Yet another use to which Amplitude can be put is demonstrated in the program “Ambulance”:

---

The SOUND Statement    27

```

10
20 REM      ***  AMBULANCE  ***
30
40 P=100
50 FOR A=0 TO 15 STEP 1
60 SOUND 1,-A,P+A,8
70 SOUND 1,-A,P-20+A,8
80 NEXT

```

```
90 FOR A=15 TO 0 STEP-1
100 SOUND 1,-A,F+A,8
110 SOUND 1,-A,F-20+A,8
120 NEXT
```

“Ambulance” is an effect program which has been devised to simulate an ambulance passing. Not only does it include a volume change which alters by one during each step of the loops, this program also simulates a Doppler frequency shift effect by changing pitch. As the ambulance approaches the siren both increase in volume and rises in pitch. Realistically enough, the sequence occurs in reverse when the vehicle ‘passes by’.

So far we have dealt with amplitudes in the range 0 to -15. Positive values for A call up ENVELOPES, which we will look at in detail in the following chapter. Fifteen such ENVELOPES are available to give the sound a rather more interesting ‘shape’ than the on-off tones we have used up to this point.

During the course of this chapter we have dealt with the major uses of the SOUND statement in isolation. The SOUND statement generates Pitch, Duration and Amplitude. In combination, these parameters provide most

of the INPUT required to create music and produce certain types of sound effects.

People of refined sensibilities might well be disappointed that the actual sound produced by SOUND appears to be a rather flat, square wave note. Do not despair, for this note can be made to sound *much* more interesting by utilising the second of the BBC's Sound of Music commands: ENVELOPE! Undaunted, then, let us proceed to the next chapter . . .

"Ambulance" is an effect program which has been devised to simulate an ambulance passing. Not only does it include a volume change which alters by one during each step of the loops, this program also simulates a Doppler frequency shift effect by changing pitch. As the ambulance approaches the siren both increase in volume and rises in pitch. Realistically enough, the sequence occurs in reverse when the vehicle 'passes by'.

So far we have dealt with amplitudes in the range 0 to -15. Positive values for A call up ENVELOPES, which we will look at in detail in the following chapter. Fifteen such ENVELOPES are available to give the sound a rather more interesting 'shape' than the on-off tones we have used up to this point.

During the course of this chapter we have dealt with the major uses of the SOUND statement in isolation. The SOUND statement generates Pitch, Duration and Amplitude. In combination, these parameters provide most of the INPUT required to create music and produce certain types of sound effects.

People of refined sensibilities might well be disappointed that the actual sound produced by SOUND appears to be a rather flat, square wave note. Do not despair, for this note can be made to sound *much* more interesting by utilising the second of the BBC's Sound of Music commands: ENVELOPE! Undaunted, then, let us proceed to the next chapter...

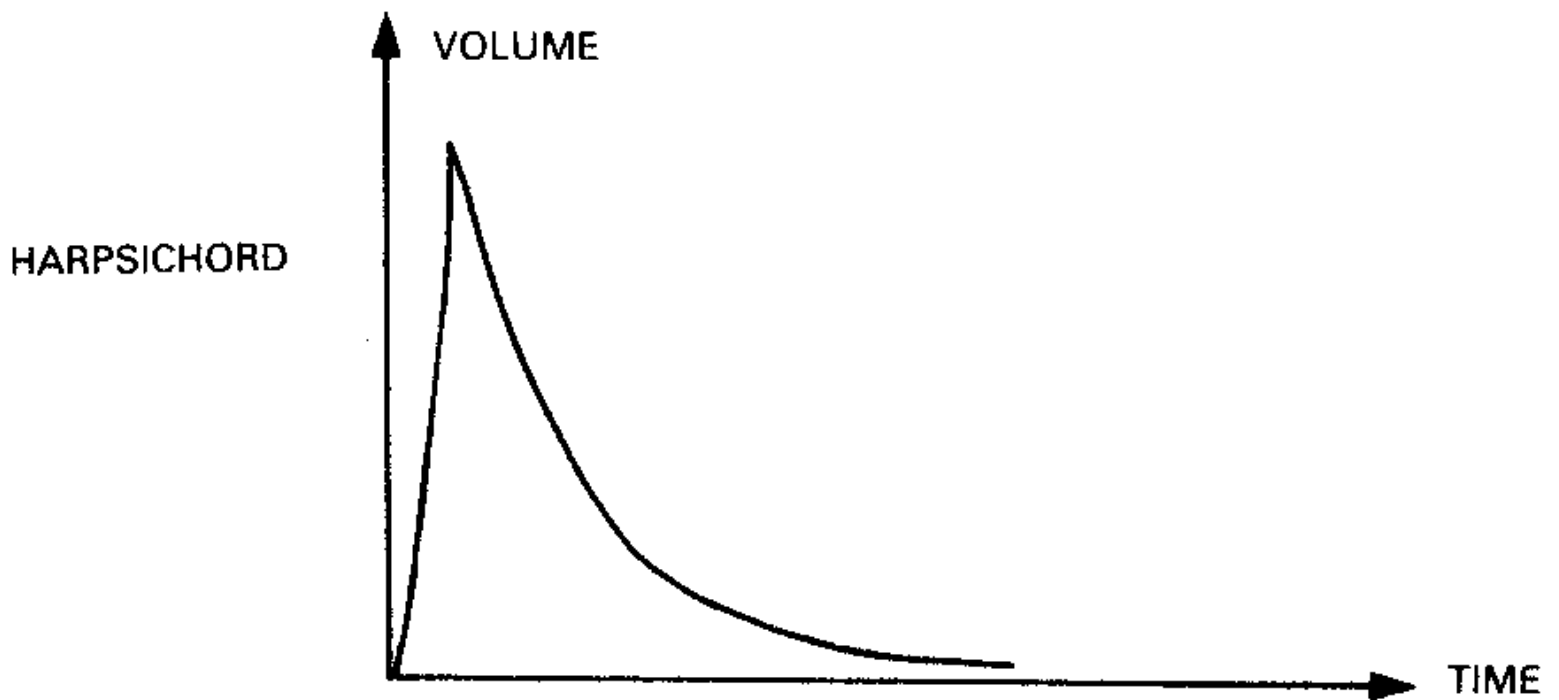
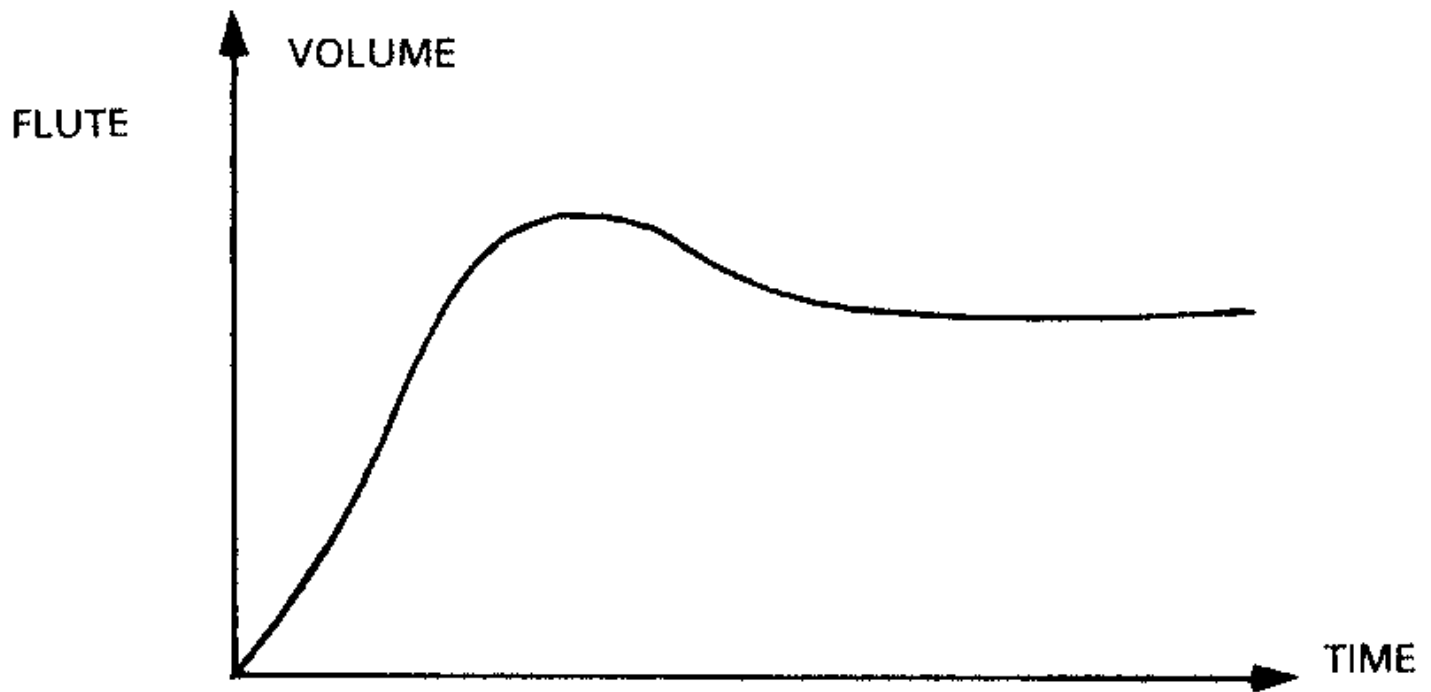


## 2 The ENVELOPE Statement

It is worth noting at this point that different musical instruments obviously have very different sound qualities. A flute has a pure, mellow sound and is generally played using continuous notes. A harpsichord on the other hand, is said to have a more abrasive quality of sound and, by the very nature of its construction and capabilities, must be played percussively.

Of course, the difference in sound mentioned above between these instruments is due not to one but to a number of factors - the most significant of which concern their volume envelopes and the actual sound source. In the case of the flute we are faced with what is almost a pure sine wave as a sound source, and an envelope which is in the main generated and controlled by the player. In the case of a harpsichord, by contrast, we are encountering a vibrating string capable of creating a rich and varied harmonic colour as a sound source, and a short envelope which 'rings' if the sustain pedal is pressed. The diagrammatic version of this explanation should help you appreciate what I'm talking about:

### VOLUME ENVELOPES

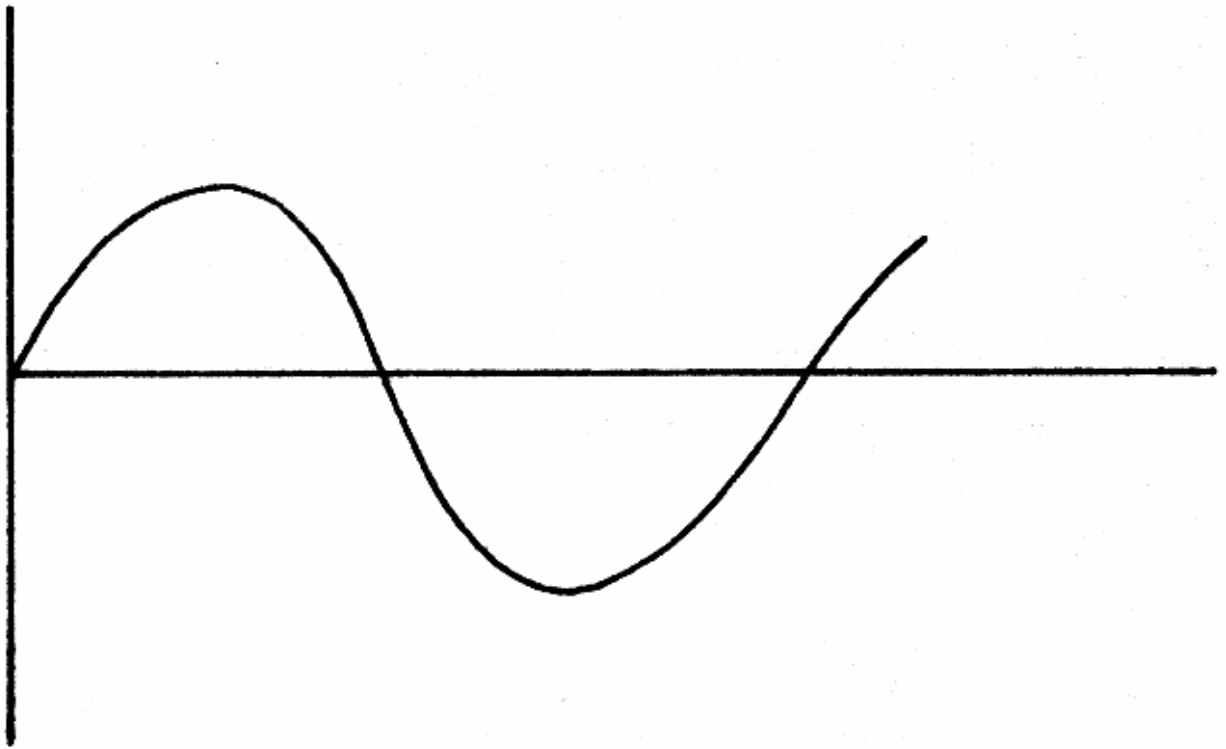


---

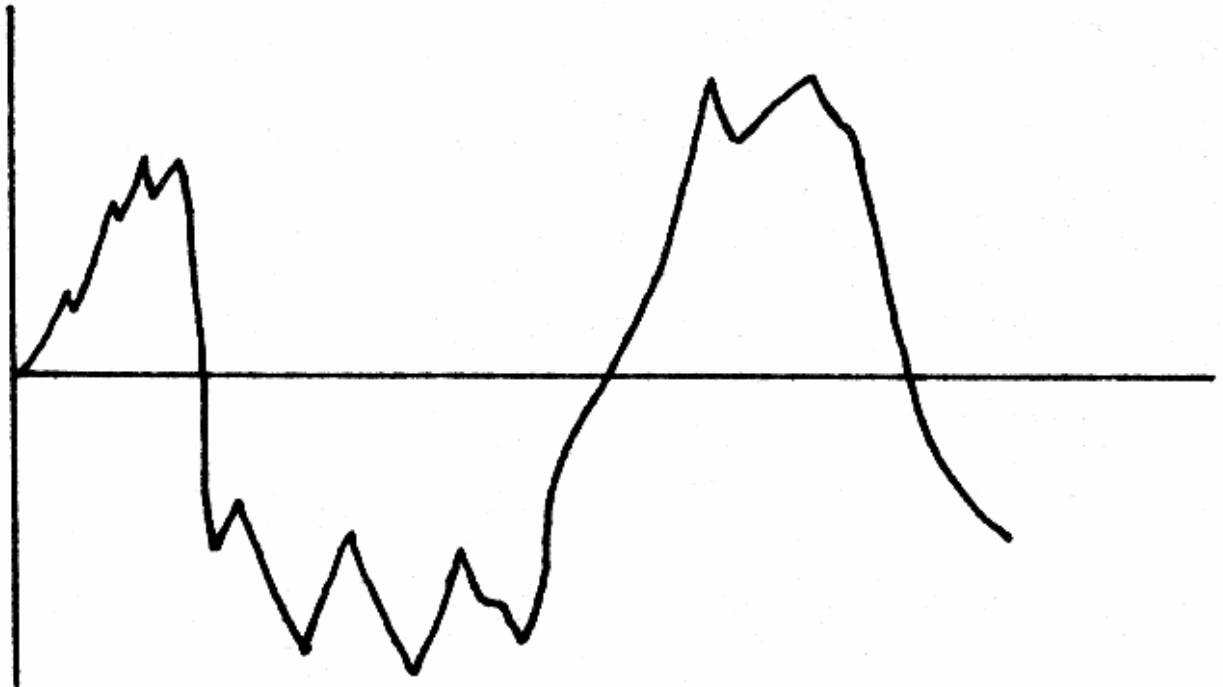
## SOUND WAVEFORMS

## SOUND WAVEFORMS

FLUTE



HARPSICHORD



The sound chip in the BBC Micro uses an electronically generated square wave as a sound source. Harmonics can be added to its basic sound by mixing together more than one channel, a topic with which we shall deal at a later point in this text. The basic sound, however, will be electronic. Do not despair! The most important factor that contributes to recognisability in most musical instruments is volume envelope. Thus, if we take the sound of a

clarinet and electronically give it a percussive envelope, the instrument's sound is immediately transformed into something which is akin to a marimba. By embracing this approach we can use the BBC's square wave and convert it into a range of different-sounding instruments. ENVELOPE is the tool we will use to create these new sounds.

It needs to be emphasised that ENVELOPE actually does more than merely control the volume (amplitude) envelope of a SOUND, for it also has a pitch envelope component. The sophistication of the ENVELOPE statement means that a large number of parameters are required. This chapter will therefore set about explaining these parameters and demonstrating their use in sound shaping.

The form of the ENVELOPE statement is as follows:

```
ENVELOPE n,s,Pi1,Pi2,Pi3,Pn1,Pn2,Pn3,AA,AD,AS,AR,ALA,ALD
```

where:

n is the envelope number

s is the length of each step in 1/100ths of a second

Pi1, Pi2 and Pi3 are the changes in pitch per step during sections 1, 2 and 3

Pn1, Pn2 and Pn3 are the number of steps in sections 1, 2 and 3 respectively

AA is the rate of change of amplitude during the attack phase

AD is the rate of change of amplitude during the decay phase

AS is the rate of change of amplitude during the sustain phase

AR is the rate of change of amplitude during the release phase

ALA is the target amplitude for the attack phase

ALD is the target amplitude for the decay phase

The ENVELOPE number n can take values from 1 to 16, and simply allows you to identify which ENVELOPE you have defined. The SOUND statement's A value then specifies which ENVELOPE is to be used, allowing you to predefine up to sixteen ENVELOPEs and call them up simply by changing A. It should be noted, however, that if the BASIC statement BPUT# is being used, the number of available ENVELOPEs is reduced to four.

Both the Amplitude and the Pitch ENVELOPEs are worked out in terms of steps, and s specifies the length of time assigned to each step. This parameter has one other function, which is to either enable or disable the ENVELOPE auto-repeat. For values of s up to 127, the ENVELOPE automatically repeats itself. Values of s over 127 disable this function. For example:

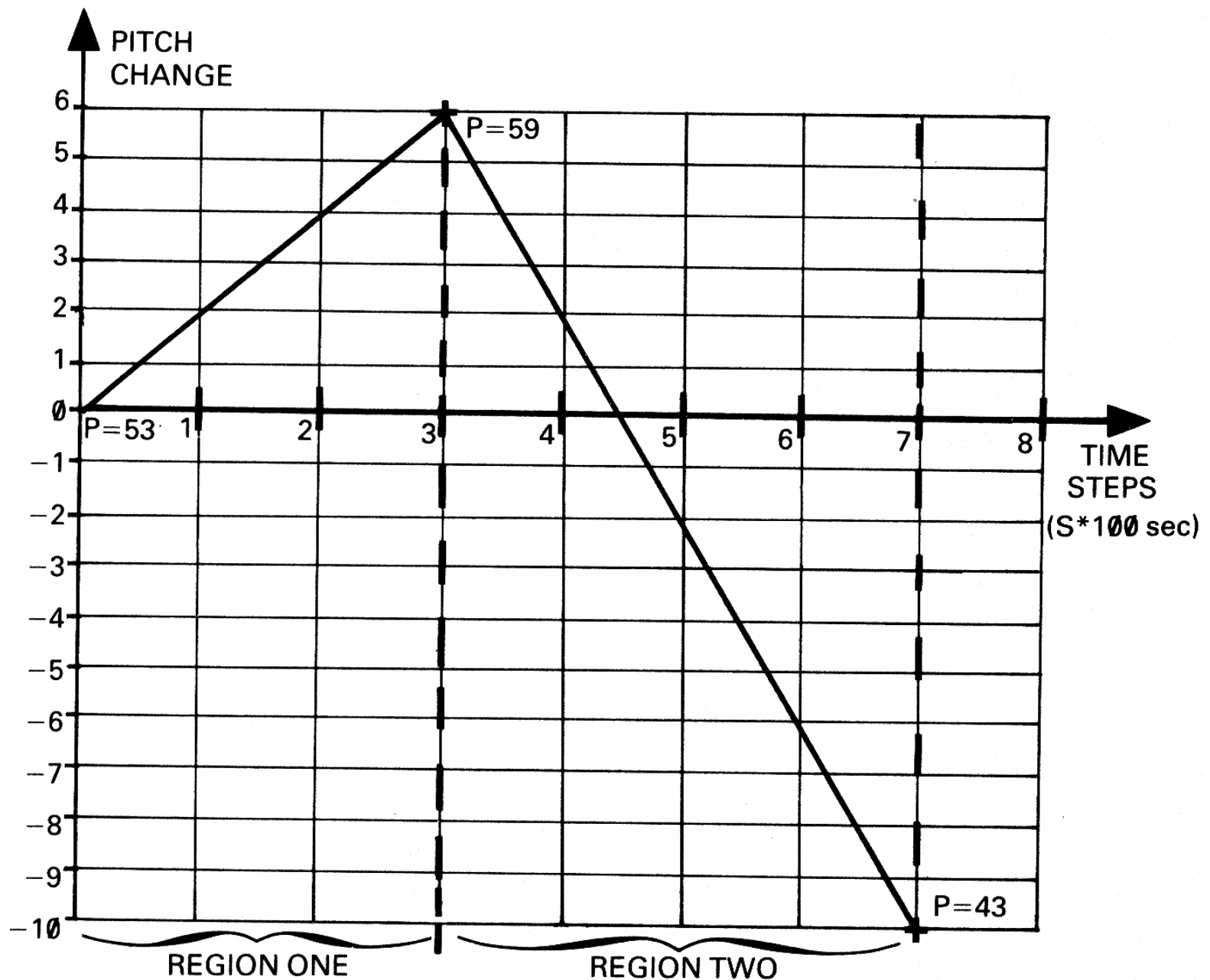
3=3/100ths of a second steps with repeat ENVELOPE

130= 127+ 3=3/100ths of a second steps with no repeats.

The six parameters following s define the pitch envelope part of the ENVELOPE statement. The length of the defined pitch envelope is completely independent of the amplitude envelope, however no sound will be heard unless a positive amplitude value is maintained.

Pi1, Pi2 and Pi3 can take values from -128 to 127. The ENVELOPE is divided into three regions and Pi1-3 control the degree of pitch change in each of these regions. Pitch can either rise or fall, and this accounts for the option of positive or negative values for Pi1-3. These values cannot be examined in isolation, but must also be related to the number of steps in each region.

Pn1, Pn2 and Pn3 can have values from 0 to 255. These parameters control the number of steps over which the specified pitch change will occur. This is best explained using the example below:



For the first region, let  $P_{i1}$  equal 2 and  $P_{n1}$  equal 3. this would mean that for every step in region one the value of  $P$  would change by 2. Since the length of region one, specified by  $P_{n1}$ , is three steps,  $P$  would change by a total of 6 over the complete section. For example: if  $v=53$  initially then at the end of the first section (region one)  $P$  will have the value 59. (See the diagram.)

If we decided that we wished the value of  $P$  to fall subsequent to this point, a negative value for  $P_{i2}$  would be used.

If values are chosen such that  $P_{i2} = -4$  and  $P_{n2} = 4$  then the total pitch change over region two would be  $(-4 * 4)$  giving us a change of -16. So at the end of the second section (region two)  $P$  will equal  $59 - 16 = 43$ .

The length of time each section will take is derived by taking the  $P_n$  value and multiplying by the value of  $s$ . The length of the complete ENVELOPE would then be given by the formula:  $(P_{n1}+P_{n2}+P_{n3})*s$ . Thus, if in the above example  $s=100$  then the length of each step would be one second. ( $100 * 1/100\text{secs.}$ ) Since  $P_{n1}=3$  and  $P_{n2}=4$ , the total envelope would hence take seven seconds to complete.

We should note at this point, however, that the duration of SOUND is also controlled by the  $o$  parameter in the SOUND statement. If  $D$  is less than the total pitch envelope time then the envelope will be cut off. If  $D$  is

## 32 The ENVELOPE Statement

greater than the total envelope time, the envelope will either repeat (if  $s < 128$ ) or, after one complete ENVELOPE cycle,  $P$  will continue till the end of  $D$ , persisting at the final value of the  $P$  parameter when the end of section 3 of the envelope is reached.

Various interesting effects can be produced using the pitch envelope with no change in volume. For instance, the following examples can be used to produce sound effects when incorporated within the body of larger programs:

```

10
20 REM"                ROAD WORKS
30
40 ENVELOPE 1,1,16,12,200,3,1,5,126,0
,0,-126,126,126
50 SOUND 1,1,1,-1
```

This program uses the noise channel to create a pneumatic drill effect. The last six parameters have been chosen to produce a constant maximum volume which is precisely equivalent to using no ENVELOPE and setting the

last six parameters have been chosen to produce a constant maximum volume which is precisely equivalent to using no ENVELOPE and setting the amplitude parameter A in the SOUND command to  $-15$ .

This is a useful technique if you are writing programs which are designed to be compatible with the Electron Microcomputer. The Electron uses a limited version of BBC BASIC, which still includes the ENVELOPE command. However the Electron command only controls the Pitch envelope parameters, and therefore although all fourteen parameters must still be typed, only the first eight have any effect on the sound produced. In addition, note that the Electron's SOUND statement amplitude parameter can only be set to zero or  $-15$  (off or on). Any negative values apart from  $-15$  are assumed to be on, so no dynamic control is possible.

"Road Works" and the two programs which follow have been written to be compatible with the Electron and will sound pretty similar on both that machine and the BBC.

```

60
70 REM"                ALARM CLOCK
80
90 ENVELOPE 2,1,70,16,2,2,0,0,126,0,0
,-126,126,126
100 SOUND 1,2,100,-1

```

"Alarm Clock" oscillates rapidly between a number of pitches in order to reproduce that familiar, nerve-jangling bell-like sound which shatters our early morning tranquility so regularly.

```

110
120 REM"                GUN
130
140 ENVELOPE 3,6,-1,-2,-3,5,5,5,126,-2
,-1,-20,126,70

```

```
, -1, -20, 126, 70
150 SOUND 1, 3, 200, 5
155 END
```

The shot effect produced in this program is obtained by rapidly sweeping through adjacent pitches. The pitch envelope can also be used to create vibrato effects which obviously greatly enhance the realism of computer synthesised instruments:

```
160
170 REM"                VIBRATO
180
190 ENVELOPE 4, 4, -1, 1, 0, 2, 2, 0, 126, 0, 0,
-126, 126, 126
200 SOUND 1, 4, 58, -1
```

The final six parameters are used to define the volume envelope. AA, AD, AS and AR are the basic control parameters, with ALA and ALD defining the amplitudes during various phases of the envelope. The amplitude envelope can be thought of as the 'shape' of a particular sound, and the judicious and imaginative use of a recognisable amplitude envelope can efficiently transform the BBC's simple square wave tone into a passable imitation of a clarinet, piano or any other instrument that the user might choose to simulate. Try it, but first let us look at the process more closely.

AA is defined as the change of amplitude per step in the attack phase of the sound. The range of allowable values for AA is  $-128$  to  $127$ . Since the starting amplitude is always zero, AA is nearly always positive. In fact, I have not been able to think of any occasion when a negative value would be useful! If, after reading this book, any of *you* can think of a use for a negative AA, please do write to me care of Pan Books (Pan/PCN, 62 Oxford Street, London) and I will include any relevant suggestion in the next edition.

The various phases of the Amplitude envelope are completely independent from the phases of the Pitch envelope. The length of each phase can be calculated by relating AA, AD, etc., to the target amplitudes ALA and



be calculated by relating AA, AD, etc., to the target amplitudes ALA and ALD. For example: if ALA=60 and AA=10 and s=5, then the attack time (in seconds) would be given by evaluating the formula  $(ALA/AA)*s/100$ , i.e.  $attack\ time = (60/10)*5/100 = 0.3$  seconds.

The larger the attack time, the more gradual the increase in amplitude at the beginning of the note. Conversely, the larger the AA value, the faster

## 34 The ENVELOPE Statement

the increase. The maximum amount of attack can be obtained by making AA equal ALA.

The amount of attack (or lack of it) is a very important factor in sound recognition. For example if you consider why a drum has a distinctive sound you will realise that this is largely a result of its very rapid attack time. Similarly many brass instruments become unrecognisable if the attack portion of their sound is removed.

AD is the change in Amplitude per step between the ALA value and ALD (the decay segment target value). In addition AD can also be positive or negative, -128 to 127. If the ENVELOPE is being used to imitate a musical instrument AD will always be negative. A decay time can be calculated using a formula similar to the one previously used for attack time:  $decay\ time = ((ALA - ALD)/AD)*s/100$ . For musical purposes, a small negative AD value is common.

AS acts like the previous values except that it can only take negative values. Note, too, that there is no target value in this case. The sustain phase continues until the duration defined in the SOUND statement expires. If the amplitude is still positive at this juncture the envelope proceeds to the release phase. Unlike the pitch envelope, the amplitude envelope has no auto-repeat, so if the amplitude reaches zero during any phase after the decay phase no sound will be heard thereafter.

Instruments such as organs and saxophones have sustain phases during which little or no change in amplitude takes place. This is such a commonly encountered situation that most synthesisers tend to set a single sustain level rather than bothering with another control designed to vary

sustain level rather than bothering with another control designed to vary sustain. Thus you will find that the synth will sustain its note while a key is depressed and, when it is released, the ENVELOPE will in turn proceed to the release phase.

Instruments which are either struck or plucked, such as pianos, drums and guitars, have no sustain phase and therefore simply jump from decay to release phase.

Finally, let us look at the release parameter, AR. This can have values ranging from zero to -127. The release phase does not start until the duration defined in the SOUND statement has elapsed. Once again, the AR value is defined in terms of change of amplitude per step, the length of steps having already been defined (in increments of 1/100ths of a second) by s.

The release phase will only be initiated if no other note is waiting to be played. In the following example the first four notes are terminated at the end of the sustain phase, but the final note is allowed to proceed to the release phase since no other note is waiting to be played:

```

10
20 REM"                CLOSE
30
40 ENVELOPE 1,4,0,0,0,0,0,0,70,-1,0,-
1,126,100
50 FOR P%=1 TO 5
60 READ Pitch%
70 SOUND 1,1,Pitch%,10
80 NEXT
90 END
100 DATA 61,69,53,5,33

```

Of course, it has to be pointed out that all this talk of attack parameters, target amplitudes and pitch envelope will mean much more to you after

Of course, it has to be pointed out that all this talk of attack parameters, target amplitudes and pitch envelope will mean much more to you after you have had a chance to experiment for a while on your own. In order to help you try this I have included a program which allows you to fiddle with the ENVELOPE parameters without having to repeatedly type in new ENVELOPE commands.

The up and down cursor keys allow you to move up and down the ENVELOPE command parameters, which are listed in a column on the screen. The left and right cursor keys then increase or decrease the parameter indicated by the text cursor. Pressing the space bar plays the new ENVELOPE you have created. The pitch and duration parameters are also included from the SOUND statement, as these are relevant to the sound produced by a given ENVELOPE.

```

10
20
30  REM  *****
40  REM  ****  ENVELOPE SHAPER  ****
50  REM  *****
60
70
80
90  REM This program allows you to
vary the ENVELOPE statement parameters
using the cursor keys.
100
110
120  *TV255,0
130 @%=5: CLEAR
140 MODE 7

```

```

150 DIM E%(14): DIM Max%(14): DIM Min%(1

```

```
150 DIM E%(14):DIM Max%(14):DIM Min%(1
4)
160 PROCmenu
170 PROCsetup
180 REPEAT:Z%=GET
190     IF Z%=32     PROCsound
200     IF Z%=136    PROCdec
210     IF Z%=137    PROCinc
220     IF Z%=138    PROCdown
230     IF Z%=139    PROCup
240     UNTIL Z%=13
250 GOTO130
260
270
280 REM These are the cursor control
procedures.
290
300
310 DEFFROCup:IF A%> 0 A%=A%-1:VDU11:
ENDPROC
320 DEF PROCdown:IF A% <14 A%=A%+1:VD
U10:ENDPROC
330 DEF PROCinc:IF E%(A%)<Max%(A%) E
%(A%)=E%(A%)+1:PROCdisp:ENDPROC
340 DEF PROCdec:IF E%(A%)>Min%(A%) E
%(A%)=E%(A%)-1:PROCdisp
350 ENDPROC
360
370
380 REM This PROC plays the newly
defined ENVELOPE.
390
400
410 DEF PROCplay
```

```

400
410 DEF PROCsound
420 ENVELOPE 1,E%(2),E%(3),E%(4),E%(5)
,E%(6),E%(7),E%(8),E%(9),E%(10),E%(11),E
%(12),E%(13),E%(14)
430 SOUND 1,1,E%(0),E%(1)
440 *FX 15,1
450 ENDPROC
460
470
480 REM This PROCEDURE displays the
changing ENVELOPE parameters.

```

---

The ENVELOPE Statement 37

```

490
500
510 DEFPROCdisp:PRINTTAB(33,(A%+2))E%(
A%);:PRINTTAB(37,(A%+2));:ENDPROC
520 PRINTTAB(33,(A%+2))E%(A%);:PRINTT
AB(37,(A%+2));
530 ENDPROC
540
550
560 REM This PROC prints the
parameter titles and controls the colo
ur of each line of text.
570
580
590 DEFPROCData
600 CLS:PRINT" *** ENVELOPE SHAP
ER ***"
610 PRINT
620 PRINT" Pitch (SOUND).....

```

```

620 PRINT" Pitch (SOUND).....
....."
630 PRINT" Duration (SOUND).....
....."
640 PRINT"s (step length).....
....."
650 PRINT"Pi1. (change in Pitch)....
....."
660 PRINT"Pi2. (change in Pitch)....
....."
670 PRINT"Pi3. (change in Pitch)....
....."
680 PRINT"Pn1. (number of steps)....
....."
690 PRINT"Pn2. (number of steps)....
....."
700 PRINT"Pn3. (number of steps)....
....."
710 PRINT"AA.....
....."
720 PRINT"AD.....
....."
730 PRINT"AS.....
....."
740 PRINT"AR.....
....."

```

---

## 38 The ENVELOPE Statement

```

750 PRINT"ALA: attack target.....
....."
760 PRINT"ALD: decay target.....
....."
770 PRINT:PRINT:PRINT:PRINT:PRINTTAB(3

```

```

770 PRINT:PRINT:PRINT:PRINT:PRINT:PRINTTAB(3
)"Press    <RETURN>    for Menu"
780 PRINTTAB(3)"Press    <SPACE BAR>
for SOUND"
790 ENDPROC
800
810
820 REM Displays Menu page.
830
840
850 DEF PROCmenu
860 CLS
870 PRINTTAB(8,2)CHR$141"    *** MENU *
**"
880 PRINTTAB(8,3)CHR$141"    *** MENU *
**"
890 PRINTTAB(2,6)"Use cursor keys as
controls:-"
900 PRINTTAB(0,12)"UP.....
.....UP COLUMN"
910 PRINTTAB(0,13)"DOWN.....
.....DOWN COLUMN"
920 PRINTTAB(0,15)"LEFT.....
...DECREASE PARAMETER"
930 PRINTTAB(0,14)"RIGHT.....
...INCREASE PARAMETER"
940 PRINTTAB(0,20)" Press <SPACE
BAR> to RUN"
950 REPEAT:J%=GET:UNTIL J%
960 ENDPROC
970
980
990 REM This PROCEDURE sets the sta

```

```

990  REM  This PROCeedure sets the sta
rting ENVELOPE parameters and defines a
maximum and minimum value for each
parameter.

```

```
1000
```

```
1010
```

```
1020  DEF PROCsetup
```

```
1030  RESTORE 1160
```

---

The ENVELOPE Statement 39

```

1040  FOR I%=0 TO 14:READ E%(I%):NEXT
1050  RESTORE 1170:FOR X%=0TO14:READ Ma
x%(X%):NEXT

```

```

1060  RESTORE 1180:FOR X%=0TO14:READ Mi
n%(X%):NEXT

```

```
1070  VDU23;29194;0;0;0;:CLS:PROCData
```

```
1080  *FX4,1
```

```
1090  *FX12,4
```

```
1100  *FX11,40
```

```

1110  FORA%=14 TO 0 STEP-1:PROCdisp:NEX
T:A%=0

```

```
1120  ENDPROC
```

```
1130
```

```
1140
```

```
1150
```

```

1160 DATA 53,20,2,0,0,0,0,0,0,126,-1,0,
-1,126,100

```

```

1170 DATA 255,254,255,127,127,127,255,2
55,255,126,127,0,0,126,126

```

```

1180 DATA 0,0,0,-128,-128,-128,0,0,0,0,
-127,-127,-127,0,0

```

Three arrays E%(14), Max%(14) and Min%(14) are set up to contain the



Three arrays E%(14), Max%(14) and Min%(14) are set up to contain the ENVELOPE parameters and their maximum and minimum values. The E%(14) parameters are then varied in PROCinc and PROCdec. The PROCinc value is compared to Max% and the PROCdec value with Min% to make sure they are not out of range. PROCdisp then controls the display of the updated value. PROCup and PROCdown move the text cursor by using the commands VDU11 for up and VDU10 for down.

Lines 180 to 240 form the control part of the program. Contained inside the REPEAT . . . UNTIL loop is the routing to the various PROCedures. This is accomplished by using a GET statement which waits until it has recognised a value of Z% and then sends the program off to the relevant PROCedure.

PROCsetup uses the \*FX command \*FX4,1 to disable cursor editing and allow the cursor keys to be used as above. \*FX11 and \*FX12 change the default auto-repeat values for the keyboard: \*FX11,40 causes the key to wait 40 centiseconds (0.4 seconds) before the repeat starts and \*FX12,4 sets the repeat period to 4 centiseconds (0.04 seconds).

Also in this PROCedure the values for Max%, Min% and the starting values of E% are READ into their respective arrays.

The bulk of what remains of the program is concerned with display – apart from PROCsound which plays the ENVELOPE when the space bar is pressed.

Setting Pi1-Pi3 and Pi1-Pn3 to zero allows you to experiment with the

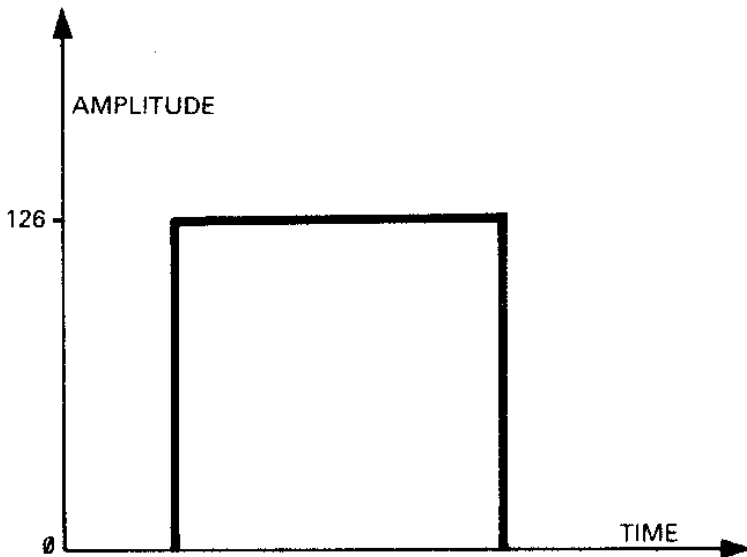
## 40 The ENVELOPE Statement

Amplitude ENVELOPE in isolation. It should be noted that the values for s and the Duration parameter have a large effect on the envelope heard. Overall volume can also be controlled using ALA and ALD, a technique we will look at later when we come to introduce dynamics into our micro music.

The workings of the Pitch envelope can be heard more clearly if a constant amplitude is set. This can be accomplished as described earlier on, by using the following values for the last six parameters:

126,0,0,-126,126,126

126 gives the maximum possible volume. A graph of this amplitude envelope would look like this:



When used in combination, the pitch and amplitude envelopes can produce vivid sound pictures in spite of the fact that they are using only a few program lines. In the following program, “Bomb”, you will see that many more lines are devoted to graphics than to sound.

```

5
6 REM"                BOMB
7
8 ENVELOPE 1,4,118,33,77,174,25,194,
126,0,0,-1,126,100
10 ENVELOPE 2,3,0,0,0,0,0,0,126,-1,0,
-1,125,100
20 SOUND 1,1,178,151
30 MODE 1 : VDU 23;8202;0;0;0;
40 VDU 19,2,2,0,0,0
50 PROCshape
55 PRINTTAB(18,24);CHR$(227)
60 x=17
70 PROCshape

```

```
60 X=1/  
70 PROCground
```

```
80 COLOUR 3  
90 FOR y=7 TO 25  
100 PRINT TAB(x,y-1) " "  
110 PRINT TAB(x,y)bomb$  
120 TIME=0 : REPEAT:UNTIL TIME>35  
130 NEXT y  
140 PROCexplode  
150 PRINT TAB(x,22) " "  
160 VDU28,15,31,27,20  
170 PRINT TAB(RND(7),RND(5));CHR$224;  
180 PRINT TAB(RND(7),RND(5));CHR$225;  
190 PRINT TAB(RND(7),RND(5));CHR$226;  
200 SOUND 0,2,6,20  
210 END  
220 DEF PROCshape  
230     VDU 23,224,240,120,255,255,255,2  
55,120,240  
240     VDU 23,225,0,0,255,255,255,255,0  
,0  
250     VDU 23,226,0,0,252,255,252,240,0  
,0  
260     bomb$=CHR$224+CHR$225+CHR$226  
265     VDU23,227,24,24,16,124,84,16,56  
,40  
270 ENDPROC  
280 DEF PROCexplode  
290 GCOL 0,1  
300 MOVE 489,200  
310 DRAW 530,400
```

```

310 DRAW 530,400
320 PLOT 85,560,200
330 DRAW 600,310
340 PLOT 85,620,200
350 DRAW 660,270
360 PLOT 85,680,200
370 DRAW 720,350
380 PLOT 85,720,200
390 ENDPROC
400 DEF PROCground
405 GCOL0,2
410 MOVE 0,212
420 PLOT85,1320,212
430 PLOT85,0,0
440 PLOT85,1320,0
450 ENDPROC

```

---

## 42 The ENVELOPE Statement

An interesting effect can be obtained by setting the following ENVELOPE:

```
ENVELOPE 1,10,8,4,-8,20,20,20,126,0,0,-1,126,100
```

With Pi values equal to tone and semitone intervals and an s value in the region of 1/10th of a second the ENVELOPE can be made to play scales. In addition, if you give much larger values to Pi, a variety of random music is generated. For example try:

```
ENVELOPE 1,15,120,-48,96,15,30,5,126,0,0,-1,126,100
```

When the pitch values go out of the 0 to 255 range they wrap around and are subsequently forced back into range by the computer. Thus  $P=300$  is equal to  $P=(300-255)=45$  and  $P=-69=(255-69)=186$ .

There are many programs available which illustrate ENVELOPES as two

There are many programs available which illustrate ENVELOPES as two sets of graphs, one of Amplitude against time and the other as Pitch against time. However, in my opinion I think that they tend to fail in their objective of providing a useful visual representation of ENVELOPE. The reason for this is the great variance in the lengths of time with which we are dealing. A percussive SOUND can be over in under 1/10th of a second, while that of a bowed violin can carry on indefinitely. For this reason I have not included an ENVELOPE graph drawing program. Nevertheless I do urge you to experiment extensively with the “Envelope Shaper” program, and either keep a note of useful ENVELOPES or, better still, write an additional routine which defines all sixteen ENVELOPES ready for use in a program.

That is all we can usefully say on ENVELOPES for the moment. Do bear in mind that, in the course of the text that follows, you will come across many useful examples of ENVELOPES which should be saved and used when you are writing your own programs. We will now press on with the next chapter – a rather raucous chapter entitled: Noise.

---

## 3 The Sound of Noise

Up until now we have been dealing exclusively with channel one, which produces what should by now be a familiar square wave tone. Both channel two and channel three produce identical sounds, and we will soon turn to look at how chords and tunes with accompaniment can be produced. In this chapter, however, we shall concentrate on the functions of channel zero, the noise channel.

Unlike the other channels, channel zero is incapable of producing melodies by varying the pitch parameter. In the case of channel zero the pitch parameter has an entirely different effect, being used to select one of eight possible noises. These can be described as in the table which follows:

0	High frequency sawtooth type tone
1	Medium frequency sawtooth type tone
2	Low frequency sawtooth type tone
3	Periodic noise of variable frequency
4	High frequency 'white' noise
5	Medium frequency 'white' noise
6	Low frequency 'white' noise
7	Random noise of variable frequency

Noises 0, 1 and 2 can be viewed as one group of noises. These are, in fact, strictly speaking tones rather than noises and have musical pitches of C in three octaves. It is not possible to vary these pitches, however, so you will find that the application of this group is to a large extent limited.

Noises 4, 5 and 6 make up another separate group, comprising three frequencies of white noise. The best way to describe white noise is to think of the sound a transistor radio gives out when it is not tuned to a station. White noise is versatile and has a whole range of applications, mostly in the sound effect area, as we shall see later on.

The final group is made up of pitch values 3 and 7. These can be thought of as 'pitched noises'. The frequency of both these noises is directly related to the current frequency specified for channel one.

Thus, for example, if pitch of channel one equals 0 both 3 and 7 give out clicking noises. The clicks from 3 appear regular, whereas 7 gives out random clicks which somewhat resemble the sound of a geiger counter.

If the pitch of channel one equals 200, noise three gives out a definite pitched note, whereas 7 gives a high-pitched white noise.

As practise clarifies matters much better than simply absorbing theory, of course, you ought to experience these sounds for yourself by running the following program. The program has been derived to play each noise in turn, as well as display its pitch value on the screen.

---

## 44 The Sound of Noise

If the pitch of channel one equals 200, noise three gives out a definite pitched note, whereas 7 gives a high-pitched white noise.

As practise clarifies matters much better than simply absorbing theory, of course, you ought to experience these sounds for yourself by running the following program. The program has been derived to play each noise in turn, as well as display its pitch value on the screen.

```

10
20 REM      *** NOISE DEMONSTRATION ***
30
40 MODE 7
50  FOR P%=0 TO 255
60      SOUND&0010,-10,P%,-1
70      PRINTTAB(4,10);CHR$(141);CHR$(13
4);"Pitch value:";CHR$(135);P%;"      "
80      PRINTTAB(4,11);CHR$(141);CHR$(13
4);"Pitch value:";CHR$(135);P%;"      "
90      TIME=0:REPEAT:UNTIL TIME=100
100  NEXT

```

Note that an extended form of the SOUND statement is used in this program, in order to synchronize the screen display with the sound being played. This technique is explained in the following chapter. The rest of

played. This technique is explained in the following chapter. The rest of the program is similar in design and construction to those used to demonstrate Pitch and Amplitude in chapter One.

You will probably have noticed that the noise demo program uses values of P greater than seven. The sound chip is programmed to return one of the eight sounds for any value of P above zero and below 255. For example, a value of eight for P would give the same sound as a value of zero. In fact, P is reduced MOD 8 for any value over 7. Look at the table below if this is not quite clear to you!

0	8	16	24	32	40	48 etc.
1	9	17	25	33	41	49 etc.
2	10	18	26	34	42	50 etc.
3	11	19	27	35	43	51 etc.
4	12	20	28	36	44	52 etc.
5	13	21	29	37	45	53 etc.
6	14	22	30	38	46	54 etc.
7	15	23	31	39	47	55 etc.

The following program allows you to play the eight possible noises using keys Z, X, C,V,B,N and M, anyone who is interested in producing what can best be describe as 'scratching' records will find the following program

The Sound of Noise 4

almost as effective as the real thing. Of course, it has the additional advantage of being much less wearing on your record collection!

```

10
20 REM      *** NOISY KEYS ***
30
40 MODE7
50 PRINTTAB(4,10); "Press <SPACE BAR>
to END"
60 PRINTTAB(2,18); CHR$(134); "Z      X
C      V      B      N      M      "
```



```

10 PRINT TAB(2,18);CHR$(134);"Z" X
C V B N M , "
70 PRINTTAB(2,20);CHR$(129);"O" 1
2 3 4 5 6 7 "
80 noise$="ZXCVBNM,"
90 *FX11,1
100 *FX12,1
110 note$=GET$
120 REPEAT
130 Pitch=INSTR(noise$,note$)-1
140 SOUND 0,-10,Pitch,-1
150 REPEAT
160 N$=INKEY$(2)
170 UNTIL N$<>note$
180 SOUND &10,0,0,0
190 IF N$="" THEN note$=GET$ ELSE note
$=N$
200 UNTIL note$=""
210 *FX12,0
220 END

```

Lines 40 to 70 set up the text display. The string `noise$` holds the symbols for each of the keys used. Lines 90 and 100 set the key auto-repeat to its fastest value. The program then waits at Line 110 for a key to be pressed. When any key is pressed the program enters the REPEAT... UNTIL loop and Line 130 compares the entered value of `note$` with the key symbol string `noise$`. The function `INSTR` tests for the occurrence of one string inside another and returns a numerical value for its position inside the string. For example, if C was pressed, `note$="C"` and "C" is contained within `noise$` at the position third from the left. `INSTR(noise$,note$)` will return the value 3. Pitch would therefore equal 2, the value of `(INSTR(noise$,note$)-1)`.

Pitch is first fed into the `SOUND` statement at Line 140 and then played. The program then enters the second loop at Line 150. The `INKEY$(2)` function tests for a keypress and the loop REPEATS UNTIL a key is no longer being depressed. Line 180 causes the channel zero sound to abruptly fall

## 46 The Sound of Noise

silent, using an extended SOUND command which will be discussed in the next chapter. Line 190 temporarily holds up the program if no key is currently being pressed and waits for a further keypress. If a key is depressed the REPEAT loop continues, with the same value for N\$. Pressing the space bar will exit the program and cause the keyboard auto-repeat to be reset to the default values (in Line 210).

Note that in the above examples both programs set an unchanging, fixed value for the frequencies of the pitched noises (pitches of three and seven respectively). This was, of course, the lowest frequency available, since no value for channel one pitch was given. In this case the sound chip assumes a zero value. The following program demonstrates all the possible frequencies of the pitched noises. The program asks for a value of Pitch to be inputted. First of all, try 3 and 7, and then any of the other possible noises. You will discover that the varying channel one pitch only affects the pitched noises.

```

10
20 REM    *** Pitch=3 and 7 NOISE ***
30
40 MODE 7
41 INPUT TAB(6,11); "WHICH NOISE? INPU
T Pitch: "Pitch
42 PRINT TAB(6,11); "
    "
50 FOR F%=0 TO 255
60     SOUND&0011,-10,F%,0
65     SOUND&0010,-10,Pitch,-1
70     PRINTTAB(4,10);CHR$(141);CHR$(13
4); "Pitch value: ";CHR$(135);F%; "    "
80     PRINTTAB(4,11);CHR$(141);CHR$(13
4); "Pitch value: ";CHR$(135);F%; "    "
90     TIME-A-REPEAT-UNTIL TIME-EA

```

```

4); "Pitch value: "; CHR$(135); P%; "    "
90    TIME=0: REPEAT: UNTIL TIME=50
100  NEXT

```

The next program allows you to experiment further with these extremely useful pitched noises. The program first asks you which noise you require, and then makes it possible for you to vary the pitch of your selected noise by using the up- and down-cursor keys.

```

10
20 REM    **** PITCHED NOISE TEST ****
30
40 MODE7
50 *FX4,1

60 PRINTTAB(2,4);CHR$(131);"Use the u
p and down cursor keys to"
70 PRINTTAB(2,5);CHR$(131);"control t
he frequency of the noise."
80 PRINTTAB(6,18);"Press <SPACE BAR>
to END."
90 PRINTTAB(2,7);CHR$(129);"Which noi
se do you want? 3 or 4?"
100 Pitch=GET
110 REPEAT
120 PRINTTAB(10,10);"PITCH: ";P%; "    "
130 K=GET
140 IF K=138 THEN P%=P%-1
150 IF K=139 THEN P%=P%+1
160 SOUND1,0,P%,0
170 SOUND&10,-10,Pitch,-1
180 UNTIL K=32
190 *FX4,0

```

```

190 *FX4,0
200 SOUND&10,0,0,0
210 END

```

While tinkering with all the possibilities, you may have noticed that noise 3 creates discernable pitch jumps which are rather like the tone channels. The “Noise 3 Scale Demo” program illustrates that a roughly chromatic scale can be wrung out of this particular noise:

```

10
20 REM      *** NOISE 3 SCALE DEMO ***
30
40 FOR P%=255 TO 0 STEP -4
50 SOUND1,0,P%,0
60 SOUND&10,-1,3,-1
70 TIME=0:REPEAT:UNTIL TIME=50
80 NEXT

```

“Noise 3” could therefore be used to add extra harmonics to a sound made up of various channels added together, or that extra note which will enable you to create four-note chords.

Noise 7 on the other hand, is more useful since it gives us many options, particularly in the creation of sound effects. Look at the program below to see what I mean:

---

#### 48 The Sound of Noise

```

10
20 REM      ***** THUNDER *****
30
40 REPEAT
50   T%=RND(10)*30
60   FOR P%=255 TO 0 STEP -1
70     SOUND1,0,P%,0
80     SOUND&10,-1,7,-1

```

```

70      SOUND&10,0,0,0
80      SOUND&10,-1,7,-1
90      NEXT
100     SOUND&10,0,0,0
110     TIME=0:REPEAT:UNTIL TIME=T%
120     UNTIL FALSE

```

You will realise that the noise channel is best exploited and used for the creation of sound effects which enhance games. As we have already seen with the tone channels, the ENVELOPE command can be employed to produce a wider range of sounds. The following examples will show you what can be created in the way of laser zaps and gunshots, using both the ENVELOPE statement and the noise channel:

```

1
2
4 REM"   VARIOUS SOUND EFFECTS
5
6
10 MODE7
20 PRINTTAB(3,10)"M for Motor Cycle"
30 PRINTTAB(3,12)"R for Rocket"
40 PRINTTAB(3,14)"T for Train"
45 PRINTTAB(3,16)"D for Dive"
50 PRINTTAB(3,6)"PRESS ANY KEY TO ST
OF SOUND"
60 PRINTTAB(9,8)OR"
70 REPEAT
80 A$=GET$
90 IF A$="M" PROC MOTORCYCLE
100 IF A$="R" PROC ROCKET
110 IF A$="T" PROC TRAIN
112 IF A$="D" PROC DIVE
120 UNTIL FALSE
130
140

```

130

140

150 REM"            MOTOR CYCLE

The Sound of Noise 49

160 DEF PROCMOTORCYCLE

170 P%=0

180 REPEAT

190 SOUND 0,-10,3,1

200 SOUND 1,0,P%,1

210 IF P%&lt;100 P%=P%+1 ELSE P%=100

220 B\$=INKEY\$(1)

230 UNTIL B\$&lt;&gt;" "

240 ENDPROC

250

260

270 REM"            ROCKET

280 DEF PROCROCKET

290 P%=160

300 REPEAT

310 SOUND 0,-10,7,4

320 SOUND 1,0,P%,4

330 IF P%&lt;254 P%=P%+1 ELSE P%=254

340 P%=P%+1

350 B\$=INKEY\$(1)

360 UNTIL B\$&lt;&gt;" "

370 ENDPROC

380

390

400 REM"            TRAIN

410 DEF PROCTRAIN

420 ENVELOPE1,1,3,-2,8,8,10,6,3,-1,0,-  
1,0,0

430 SOUND1,1,200,-1

440 SOUND0,-15,7,-1

```
560 NEXT
570 SOUND0,2,4,200
580 ENDFROC
```

[illegible]

[illegible]



```

-24,-32,-1,126,126
  100 ENVELOPE 4,1,0,0,0,0,0,0,126,-1,0,
-9,126,62
  110 REPEAT
  120 A$=GET$
  130 IF A$="Z" THEN SOUND&11,1,5,1
  140 IF A$="X" THEN SOUND&12,2,36,3
  150 IF A$="C" THEN SOUND&13,3,8,2
  160 IF A$="V" THEN SOUND&10,4,4,2
  170 UNTIL A$="Q"

```

If you are mesmerised by the sheer cunning of this technique, turn to the application chapter where these drum sounds are incorporated in a drum machine program which allows you to program a number of one-bar drum patterns and chain them together to form a complete drum part or solo!

It goes without saying that more interesting effects can be created by using more than one channel at a time. A snare drum, for instance, has a pitch component as well as a rattle contained in its sound. As the drum is hit, so the snare rattles and the drum changes in pitch. The pitch change part of the sound can be created using the routine which follows. If we add this to the snare rattle sound that we created earlier, a much more interesting snare sound will be derived.

```

10
20 REM"*****IMPROVED SNARE*****
30
40 ENVELOPE1,1,0,0,0,0,0,0,126,-6,-2,
-3,126,5
50 ENVELOPE2,1,-1,0,0,40,0,0,126,-6,-
1,-3,126,40
60 ENVELOPE3,1,-2,0,0,40,0,0,126,-6,-
1,-3,126,40
70 ENVELOPE4,1,-10,0,0,100,0,0,126,-4
0,-10,-120,126,0

```

```
0,-10,-120,126,0
80
90
100 REPEAT
110 A$=GET$
120 SOUND0,1,6,1
130 SOUND1,2,40,1
```

---

```
140 SOUND 2,3,80,2
```

```
150 SOUND 3, 4, 200, 1
```

```
160 UNTIL FALSE
```

In the previous example two sounds were played simultaneously simply by writing two SOUND statements for different channels. This method of layering sounds is perfectly acceptable for a large number of applications. Chords can be built up using this method, and providing only one chord is to be played all the notes will be sounded at approximately the same time. If, however, a string of chords is to be played or the sound is to be synchronised with graphics, another method must be found.

The BBC Micro comes to our aid at this juncture and supplies just such a method, which we shall look at in the chapter which follows.

## 4. More on SOUND and ENVELOPE

In this chapter we will expand our range of activities to include the remaining SOUND channels two and three and then carry on to take a look at some further applications of ENVELOPE. It should come as no surprise to you to discover that the use of two or more channels simultaneously brings a number of problems in its stead. How do we go about synchronizing channels? Why is it that SOUND commands apparently act out of sync with screen displays? Over the next few pages we will find an answer to these questions and apply our new-found knowledge to programs such as a polyphonic organ, screen sync and polyphonic melody playing. One of the first new concepts we need to examine at this juncture is the SOUND queue.

### THE SOUND Queue

Each SOUND channel has a small portion of memory set aside to store up to five SOUND statements. This memory stores SOUND statements in a queue. the queue is a useful device which allows short strings of SoUND state-ments to be executed without holding up a program for the duration of the routine. To clarify this point let us look at the following program:

```
10 REM QUEUE

20 PRINT"C"

30 SOUND 1,-10,53,50

40 PRINT"E"

50 SOUND 1,-10,69,50

60 PRINT"6"

70 SOUND 1,-10,81,50
```

At first glance, you might think that when this program is RUN, C will be printed and simultaneously played. Following this you might imagine that E will be printed and accompanied by E SOUNDing, and finally G will appear on the screen at the same time as the sound G is heard.

In fact, this is not the way it happens at all, for C, E and G are PRINTed on the screen before the first note has finished playing. Has the BBC gone loopy? Has it decided to read Lines 20, 40 and 60 before

going back for the SOUND commands? This apparent leaping around the program can be simply explained with reference to the SOUND queue.

The machine in fact read and executed the program lines as normal. This is what happens: Line 20 is read and executed immediately. C is PRINTed on the screen. Line 30 is read and executed immediately. The note of pitch C is heard. At this point things do not go quite as expected. Instead of waiting for the note to finish before proceeding, the computer now reads and executes line 40. As a result, E is printed while c is still being heard. Line 50 is then read and the SOUND information for this line is stored in the buffer memory for channel one. This allows the computer to proceed to Line 60, PRINT G and carry on to the final Line, 70. Line 70's SOUND information also goes into the buffer memory, coming after E in the queue. At this point C E G is seen on the screen, while c is still being SOUNDED. Control of the computer is returned to the user even though E and o are still to come. Since the BBC operates so quickly the initial c notes has not yet ended and the computer continues to play all three notes in the channel one SOUND queue, even though it is possible to type new information into the computer. Only by pressing ESCAPE or BREAK will you find it possible to terminate the sound.

The SOUND queue comes in very handy when you want to play a quick fanfare, of the kind found in a computer game, for example. Providing the number of notes in the fanfare does not exceed five, the program will not be held up. When you come to read the 'Applications' chapter, you will see that a number of examples of music which may be used in games are explored.

The program below "Queue 2", will give you a clearer idea of how the queue works. As it is written "Queue 2" PRINTs READING P% INTO Queue for values of P% from 1 to 6. Since there are then six SOUND commands, one is played straight away and the remaining five are held in the queue, to be played in turn. FINISHED is PRINTed immediately.

```
10 REM QUEUE

20 CLS

30 FOR P%=1 TO 6

40 SOUND 1,-10, P%*24, 20

50 PRINT "READING "; P%;" INTO QUEUE"

60 NEXT

70 PRINT "FINISHED!!!"
```

If we try to add just one more SOUND statement to the queue the program as a whole is held up. Try changing Line 30 to read:

```
FOR P%=1 TO 7
```

In this case we find that the 'READING' message for each value of P% up to 6

---

More on SOUND and ENVELOPE 55

is PRINTED while the first note in the queue is being played. Not until the second note starts, however, do we see 'READING 7 INTO QUEUE' and 'FINISHED'. If we make Line 30:

```
FOR P%=1 TO 20
```

then 'READING 7' will be PRINTED when note 2 is being played, 'READING 8' will be PRINTED when note 3 is being played – and so on. Control of the computer will only revert to the user when there are less than six notes left.

let us turn to another consideration at this point and note that if synchronization with a screen display is required the buffer creates a problem. How do we get the SOUND part of our program to move in step with our graphics? One solution would be to carefully time our music and effects to last as long as the visual action. We would still encounter problems, however, if a number of winning fanfares accumulated in the buffer when in reality we had been shot down ages ago.

As usual, the BBC provides a way round this problem but before looking at the solution in detail I must first introduce a second new and crucial notion – the extended SOUND statement.

## THE EXTENDED SOUND STATEMENT

At this point in the text we have dealt exclusively with a simplified form of the SOUND statement. In its extended form the SOUND command acquires three extra values. Up until now these additional parameters have invariably been set to zero and have therefore been inoperative. They are most easily written as a 4 digit hexadecimal number which is substituted in place of the channel number. This number takes the form:

most easily written as a 4 digit hexadecimal number which is substituted in place of the channel number. This number takes the form:

**&HSFC**

The complete extended SOUND statement will then have the form:

**SOUND &HSFC,A,P,D**

The use of the ampersand (&) sign specifies that what follows is a hexadecimal number. The number could equally well be written in decimal form but at this stage this would obscure the workings of the various new parameters. These parameters take the following values:

---

## 56 More on SOUND and ENVELOPE

Parameter	Range	Effect
H	0 or 1	continuation control
S	0 to 3	synchronization control
F	0 or 1	flush control
C	0 to 3	channel number

### C for Channel

The channel number C can take any of the values 0, 1, 2 and 3. 1, 2 and 3 are identical square wave tone channels and 0 is the noise channel. In the preceding pages we have dealt exclusively with channel one so let us now note that channels two and three act in precisely similar ways. In addition, we can of course use more than one channel simultaneously. For example to produce a C Major chord we simply write a SOUND statement for each note:

```
10 SOUND 1,-10,53,100
20 SOUND 2,-10,69,100
30 SOUND 3,-10,85,100
```

```
20 SOUND 2, -10, 53, 100
30 SOUND 3, -10, 85, 100
```

Note that the above SOUND commands could equally well have been written in the extended form: SOUND &0001,-10,53,100 etc. This is, in fact, unnecessary when the remaining parameters are set at zero.

The following example shows a simple but effective way of using our newly found SOUND channels two and three. If we simply duplicate the channel one pitch and duration information we can hear the three channels played in unison. This produces a far richer and more interesting sound than that derived from using the rather austere single channel.

```
10 REM   *** EXAMPLE ***
20
30 FOR R%= 1 TO 12
40   READ P%,D%
50   FOR C%=1 TO 3
60     SOUND C%,-10,P%,D%*8
70   NEXT
80 NEXT
90
100 DATA 117,9,109,1,101,1,97,1,101,1,
109,1,117,1,101,1
110 DATA 121,6,117,1,109,1,101,16
```

The above program uses a FOR . . . NEXT loop to play each channel in quick succession. Since BBC BASIC is fast, the end result might lead you to think

that you are hearing three notes starting simultaneously. The interesting vibrato effect is produced because the three tone generators are very close to being completely in phase, but not exactly so.

The following program uses the same technique to play a song in three part harmony. Note that the pitch information in this example is altered for each channel.

part harmony. Note that the pitch information in this example is altered for each channel:

```

10    REM    *** HARMONY ***
20
30    DIM P%(3)
35    REPEAT
40        FOR R%= 1 TO 9
50            FOR C%=1 TO 3
60                READ P%(C%)
70            NEXT
75            READ DUR%
80
90            FOR C%=1 TO 3
100                SOUND C%,-10,P%(C%),DUR%
110            NEXT
112        NEXT
113    RESTORE 130
115    UNTIL FALSE
120
130    DATA 33,53,69,16,41,53,73,16,49,6
1,81,16,41,53,73,16
140    DATA 33,53,69,8,41,53,73,8,49,61,
81,8,41,53,73,8,33,53,69,32

```

All is not as straightforward as we might think, however, for this simple method of producing harmonization would fall down if, for example, we want to have a bass part following a separate rhythm. Such a program would be impossible to write with the SOUND queue in operation because as soon as one channel got more than five notes ahead of the other the synchronization would break down. To convince yourself of the truth of this statement RUN the following program:

```

10    REM QUEUE 3

```



```

10  REM QUEUE 3
20  FOR I=1 TO 24
30  READ F1%,D1%,F2%,D2%
40  SOUND 1,-10,F1%,D1%
50  SOUND 2,-10,F2%,D2%
60  NEXT I

```

---

## 58 More on SOUND and ENVELOPE

```

70  DATA 53,5,5,10,69,5,33,10,61,5,5,
10,81,5,33,10
80  DATA 53,5,5,10,69,5,33,10,61,5,5,
10,81,5,33,10
90  DATA 53,5,5,10,69,5,33,10,61,5,5,
10,81,5,33,10
100 DATA 53,5,5,10,69,5,33,10,61,5,5,
10,81,5,33,10
110 DATA 53,5,5,10,69,5,33,10,61,5,5,
10,81,5,33,10
120 DATA 53,5,5,10,69,5,33,10,61,5,5,
10,81,5,33,10

```

Please note that each DATA statement in this program is identical and therefore it can be copied using the key provided for that purpose. You will notice that the DATA for channel two has Duration values that are longer than those for channel one's notes. This is reflected in the sounds produced by the program until channel one gets more than five notes ahead. At this point channel one is full and synchronization breaks down.

We have therefore seen that there is a need for some special commands which would help us to relate one SOUND channel to another, or disable the SOUND queue. These are precisely the functions provided by the other extended SOUND parameters.

## F FOR FLUSH:

A value of zero for F gives exactly the same result as the simple SOUND statement. A value of one, however, gives that SOUND statement priority over all the SOUNDS that have already come in the channel queue. As a result the queue is flushed of all current pitch, duration and amplitude information and the new SOUND command is played immediately. For example, if we change Line 70 in the “Queue 1” program (page 53) to incorporate a value of one for the F parameter, the result is not the three note chord you might expect. Insert the following to see what I mean:

```
70 SOUND&0011,-10,81,50
```

In this case the program appears to only play Line 70. In fact, the order of events is as follows: the program proceeds just as before through each line of the program. When 70 is reached C E G has been PRINTed and C has just started to play. The F parameter in the SOUND statement of Line 70 is set to 1, which causes the channel one buffer to be flushed of all previous SOUNDS and G is played. Once again, the speed of the BBC makes the

---

More on SOUND and ENVELOPE 59

whole thing happen so quickly that you are unlikely to hear the first note play!

The program which follows illustrates how the flush parameter can be used to synchronize SOUNDS with a screen display.

```
10
20
30 REM      *** FLUSH DEMO ***
40
50
60 MODE 2
70 T=0
```

```

60 MODE 2
70 T=0
80 VDU23;8202;0;0;0;
90 REPEAT
100 FOR X=1 TO 7
110 T=T+1
120 COLOUR X
130 PRINTTAB(0,X*4);"00000000000000000000
000"
140 SOUND&11,-10,T*4,-1
150 TIME=0:REPEAT:UNTIL TIME=60-T
160 CLS
170 NEXT
180 UNTIL T=56
190 MODE 7:VDU23;8202;0;0;0;:SOUND&11,
0,0,0
200 END

```

In the SOUND statement, Duration is set to be infinite by making D equal to -1. As each loop of the program is executed the SOUND statement is kept in perfect synchronization with the display by flushing the buffer and playing the updated value of P immediately. Note that since the other parameters in the hexadecimal number are zero the number can be written as &11, instead of &0011, in Line 190.

One frequently used application of the flush parameter is to keep the BBC quiet. If abrupt and total silence is required, say at the end of a program, the statement:

SOUND &11,0,0,0

will ensure peace and quite. For example:

```

10  REM *** THE END ***
15  CLS
20  TIME=0
30  REPEAT
40      SOUND1,-15,RND(30)*4,8
50      UNTIL TIME>300
60
70  SOUND&11,0,0,0
80  PRINTTAB(10,10);CHR$(141);"THE EN
D";TAB(10,11);CHR$(141);"THE END"

```

As written, the program stops rather abruptly as soon as the message reaches the screen. Try RUNNING the program again with Line 70 deleted and you will find that the sound continues for several notes after the message has been displayed.

## S FOR SYNCHRONIZATION

The s parameter allows a number of SOUND statements first to be queued and then to be played simultaneously. s can take a value of 0,1,2 or 3. This value corresponds to the number of channels that must be queued before a final SOUND will make it possible for the group to play together. To illustrate the s parameter in action first run the following routine:

```

10  SOUND 1,-10,53,100
20  TIME=0:REPEAT:UNTIL TIME>100
30  SOUND 2,-10,69,100

```

Just as you no doubt anticipated, the SOUND on channel one plays first, to be followed by a pause and then the channel two pitch plays. Now RUN the following routine:

```

10  SOUND &0101,-10,53,100
20  TIME=0:REPEAT:UNTIL TIME>100

```

```

10 SOUND &0101, -10, 53, 200
20 TIME=0:REPEAT:UNTIL TIME>100
30 SOUND &0102, -10, 69, 100

```

In this case there first appears a pause and then both notes start simultaneously. The program reads Line 10 and sees a value of one for the synchronization parameter. It therefore waits until it has found one other note on a different channel before it SOUNDS channels one and two at the same time.

The usefulness of the synchronization parameter is perhaps less immediately apparent than that of the flush parameter. If we want to play

---

More on SOUND and ENVELOPE 61

a single three-note chord, for instance, there is little perceptible difference between writing:

```

10 SOUND&201, -10, 53, 200
20 SOUND&202, -10, 69, 200
30 SOUND&203, -10, 85, 200

```

and:

```

10 SOUND 1, -10, 53, 200
20 SOUND 2, -10, 69, 200
30 SOUND 3, -10, 85, 200

```

In more complex programs, however, where lines are inserted between SOUND statements, the S parameter becomes very useful. For an example of this in action you could turn to the “Drum Machine” program which follows in the later chapter on Applications.

## H FOR CONTINUATIONS?

The strangely named H parameter allows a dummy SOUND command to be inserted into the SOUND queue. To do this H is set at one. A value of zero has no effect. The previous note in the queue then continues during the

inserted into the SOUND queue. To do this H is set at one. A value of zero has no effect. The previous note in the queue then continues during the dummy command's duration. Logically enough, any values for A and P parameters therefore have no effect.

It is not immediately apparent why anyone should want to insert a dummy value into the SOUND queue. The following illustration should serve to clarify the situation somewhat. First type and RUN the following programs:

```
10 ENVELOPE1,2,0,0,0,0,0,0,126,-1,0,-1,126,100
20 REPEAT
30 SOUND 1,1,53,100
40 SOUND &1001,0,0,10
50 UNTIL FALSE
```

```
5      *** PROG TWO ***
10 ENVELOPE1,2,0,0,0,0,0,0,126,-1,0,-1,126,100
20 REPEAT
30 SOUND 1,1,53,10
40 SOUND &1001,0,0,100
50 UNTIL FALSE
```

---

## 62 More on SOUND and ENVELOPE

The only differences between the two programs are the Duration values of the two SOUND statements. When "PROG ONE" is RUN we hear the note of C played at the sustain amplitude for a duration of 100, followed by a truncated release phase lasting for 10 Duration units. The overall note lasts for 110, which is exactly the same length of time as the note in "PROG TWO". In the second program, however, the sustain Duration is only 10 and the release phase is allowed to continue for 100 units. The F value therefore allows us to hear the release phase of the SOUND immediately ahead in the queue.

One possible use for this phenomenon is the introduction of rests into a music program. As we have seen, it is possible to force a rest or a period of

One possible use for this phenomenon is the introduction of rests into a music program. As we have seen, it is possible to force a rest or a period of silence by using the command `SOUND &0011,0,0,0`. This uses the F parameter to flush the queue and cause silence from the tone generator for channel one. One disadvantage implicit in using this technique is the abruptness of the note's ending. By using the H parameter we can allow the release phase of the previous note to continue. For example:

```

10 ENVELOPE 1,2,0,0,0,0,0,0,126,-1,0,
-1,126,100
20 SOUND 1,1,101,-1
30 FOR F=1 TO 3000:NEXT F
40 SOUND&1011,0,0,100

```

The program uses the line `SOUND &101,0,0,100` to force the previous note into its release phase. This technique can be used to give a more 'natural' sounding finish to a note, rather than the abrupt and somewhat jawing silence normally produced. The following program uses the H parameter in just such a way:

```

10 CLS
20
30 REM    STAR AND SPANGLED BANNER
40
50 ENVELOPE1,1,0,0,0,0,0,0,126,-1,0,-
1,126,100
60 REPEAT
70 FOR N=1 TO 36
80 READ F,D
90 IF F=0 THEN SOUND&1001,0,F,D ELSE
SOUND1,1,F,D
100 NEXT
110 RESTORE
120 UNTIL FALSE

```

```

130 DATA 81,12,69,4,53,16,69,4,0,12,81
,4,0,12,101,16,0,16
140 DATA 117,12,109,4,101,16,69,4,0,12
,77,4,0,12,81,16,0,16
150 DATA 81,16,117,16,0,8,109,2,0,6,10
1,4,0,12,97,32
160 DATA 89,4,0,8,97,4,101,4,0,12,101,
16,81,16,69,4,0,12,53,16

```

'The Star and Spangled Banner' uses DATA statements to supply rest information, as well as that for Duration and Pitch. In this respect it is identical to the 'Tune' program in Chapter One. A rest is called up if a Pitch value of zero is READ. The statement SOUND &1001,0,0,D is then put in the channel one queue. This dummy entry, instead of abruptly stopping the note each time a rest appears, allows the release phase of the previous note to be entered and continued for the duration of the rest. Sometimes this results in a period of silence when the release phase is completed. The overall effect is reminiscent of the sound produced by an instrument playing in a large room, where the room reverberation can be heard when the instrument has stopped playing.

We have now covered all the main commands which are used in creating sound on the BBC. For the remainder of this chapter we shall look at a couple of examples of the extended SOUND statement at work, and also investigate a useful alternative method for producing sound from the Beeb.

One way in which we could use our new found knowledge about channels two and three and the extended SOUND command would be to write a polyphonic organ program. (As you will doubtlessly realise, we are restricted to three note polyphony with three channels). That is quite straightforward, I hear you say, for we just take our monophonic organ and add a couple of channels. Do not be fooled, however, for as is so often the case in computing, things are not quite as simple as they may at first appear.



the case in computing, things are not quite as simple as they may at first appear.

For a start, if we use the technique of looking for a keypress to start a note (and then looking for a key release) we must devise a way of telling the computer which note has stopped being depressed. This very quickly becomes extremely confusing. An easier method is shown in the following program, "Three Note Organ":

```

10
20 REM"      *****
30 REM"      ***Three Note Organ***
40 REM"      *****

45 ENVELOPE 1,1,0,0,0,0,0,0,126,0,0,-
1,126,126
50
60 REPEAT
70   Channel%=1
80   RESTORE170
90   FORp%=1TO26
100    READkey%
110    IFINKEY(-key%) SOUND&10+Channel%,
1,p%*4+40,2:Channel%=Channel%+1
120    IFChannel%>3 p%=26
130  NEXT
140 UNTIL FALSE
150 END
160
170 DATA 65,97,2,66,34,82,35,51,68,36,
84,69,85,54,70,71,55,87,56,88,73,57,89,4
1,74,58

```

"Three Note Organ" uses the negative INKEY function to test if a particular key has been pressed. Page 275 of the *BBC User Guide* gives a list

“Three Note Organ” uses the negative INKEY function to test if a particular key has been pressed. Page 275 of the *BBC User Guide* gives a list of the numbers used to test for specific keys. In the case under discussion, I have used the middle two rows of keys, including <CAPS LOCK>, <CTRL> and <RETURN>. A selection of the INKEY values are as follows:

KEY	NUMBER
<CAPS>	-65
<TAB>	-97
<CTRL>	-2
A	-66
F	-68
@	-72

The DATA is constantly being scanned to see if a particular key has been pressed. If a key has been pressed during the remaining part of that loop the channel variable is increased by one. The duration in the SOUND statement is set to be longer than the time taken to complete the loop and scan all the keys. On the second loop after a keypress, if the key is still being pressed, the SOUND statement will flush the channel and continue playing the same note. If the key is no longer being depressed the note will stop after 3/20th of a second (the total Duration).

In this program the F parameter has been set to one. Note the form of the extended command. C% is added to the total hexadecimal number rather than simply being tagged onto the end.

If more than three notes are played the new note will play on channel one by using Line 120 to flip out of the loop. Line 70 then sets Channel% to one and the loop starts from 1. The action of the organ is rather sluggish, due to the necessity of using 3/20ths of a second for the SOUND duration. In musical terms, of course, this is quite a long time. If a shorter duration is used, however, glitches can be heard – this is due to the fact that the time the loop takes to complete is longer than the duration. The program

is used, however, glitches can be heard – this is due to the fact that the time the loop takes to complete is longer than the duration. The program could be speeded up, however, by writing Lines 60 to 110 on one line, removing all extraneous spaces and reducing the variable names to one letter. If you follow the above procedure, you should be able to reduce the Duration to 2 and in so doing speed up the keyboard action considerably.

The following program shows how we can use the synchronization parameter S to ensure that the well-known melody remains in correct harmony from start to finish:

```

10
20 REM"          ***HAPPY BIRTHDAY***
30
40 MODE 7
41 VDU23;8202;0;0;0;
50 PRINTTAB(4,10);CHR$(131);CHR$(141)
;"PRESS ANY KEY TO PLAY";TAB(4,11);CHR$(
130);CHR$(141);"PRESS ANY KEY TO PLAY"
60 REPEAT
70 A$=GET$
80 RESTORE 520
90 PROCinit
100 PROCplay
110 UNTIL FALSE
120
130 DEFPROCinit
140 K$="BC*D*EF*G*A*"
150 TEMPO%=4:S%=0:B$="0"
160 ENVELOPE1,2,0,0,0,0,0,0,126,-1,-1,
-1,89,80
170 ENVELOPE2,2,0,0,0,0,0,0,110,-1,-1,
-1,77,72
180 ENVELOPE3,2,0,0,0,0,0,0,126,-1,-1,
-2,80,71
190 ENDPROC

```

```

-,-,-,-
190 ENDPROC
200
210
220
230 REM"          ***PLAY TUNE***

```

---

## 66 More on SOUND and ENVELOPE

```

240
250 DEFPROCplay
260 REPEAT
270   FOR C%=1 TO 3
280     READ Nn$,Duration%
290     IF Nn$="END" GOTO 320
300     SOUND 512+C%,C%,FNnote(Nn$),(Dur
ation%*TEMPO%)
310   NEXT C%
320 UNTIL Nn$="END"
330 ENDPROC
340
350 REM"          ***CONVERT FITCH***
360
370 DEF FNnote(N$)
380 S%=0
390 Symbol$=LEFT$(N$,1)
400 Octave%=VAL(RIGHT$(N$,1))
410 IF LEN(N$)=3 THEN B$=MID$(N$,2,1)
ELSE B$="0"
420 IF B$="f" S%=1
430 IF B$="b" S%=-1
440 Value%=INSTR(K$,Symbol$)+S%-1
450 note%=Value%*4+(Octave%*48)

```

```

450 note%=Value%*4+(Octave%*48)
460 =note%
470
480
490 REM"          ***THE NOTES***
500
510
520 DATA G1,2,D1,2,B1,2,G1,1,D1,1,B1,1
530 DATA A1,3,E1,3,C1,3,G1,3,E1,3,C1,3
,C2,3,G1,3,E1,3
540 DATA B2,6,G1,6,D1,6,G1,2,D1,2,B1,2
,G1,1,D1,1,B1,1
550 DATA A1,3,F1,3,C1,3,G1,3,D1,3,B1,3
,D2,3,B2,3,G1,3
560 DATA C2,6,G1,6,E1,6,G1,2,D1,2,B1,2
,G1,1,B1,1,D1,1
570 DATA G2,3,E2,3,C2,3,E2,3,C2,3,G1,3
,C2,3,G1,3,E1,3
580 DATA B2,3,G1,3,D1,3,A1,6,F1,6,C1,6
590 DATA F2,2,C2,2,A1,2,F2,1,C2,1,A1,1

```

---

More on SOUND and ENVELOPE 67

```

600 DATA E2,3,C2,3,G1,3,C2,3,G1,3,E1,3
,D2,3,B2,3,G1,3
610 DATA C2,12,E1,12,C1,12
620 DATA END,0

```

This program plays 'Happy Birthday' in three-part harmony if you simply press any key on the keyboard. In this case I have chosen to write the extended SOUND command in decimal notation, both to show that it can be done this way and also to demonstrate why hexadecimal notation is preferable. The value of 512 is calculated from the hexadecimal equivalent, which is 0200.

preferable. The value of 512 is calculated from the hexadecimal equivalent, which in this case is &200:

&	2	0	0	:200
	(16*16)*2	0	0	:512 decimal

Since s equals 2, two statements on differing channels must therefore be collected before all three channels are SOUNDED on execution of the third statement. &200 is the general extended value for ensuring three-part harmony.

Lines 10 to 70 make the program wait for a keypress before continuing VDU23;8202;0;0;0; switches off the cursor. PROCinit sets up the program, defines ENVELOPE and K\$, which will be used in FNnote to convert the DATA written in music symbols to BBC pitch information.

At this point I should point out something which most of you have probably realised, namely that at various points in the book I have used differing methods of storing pitch information. I hope, rather than being confusing, that this makes the relationship between the BBC's musical language and the music much clearer. The following technique has as an advantage the fact that anyone with a measure of knowledge of music theory will find it readily understandable.

In Line 390 Symbol\$ first strips the leftmost part of the entry. This would be the musical symbol. Octave% does the same for the rightmost part, giving the octave number. Line 410 then determines whether the entry is three spaces long (ie: whether it contains a sharp or a flat). If it is, S% is given a value of 1 for a sharp or -1 for a flat. Value% is then calculated by comparing Symbol\$ with K\$, using the INSTR function. On the basis of this calculation we arrive at a Value% between 0 and 11. The actual Pitch which will be used in the SOUND statement is then determined by combining all this information in Line 450:

$$\text{note\%} = \text{Value\%} * 4 + (\text{Octave\%} * 48)$$

The rest of the program is a basically straightforward play PROCEDURE which uses a loop and the variable C% to select the relevant channel and

which uses a loop and the variable C% to select the relevant channel and

---

## 68 More on SOUND and ENVELOPE

ENVELOPE. You may have noted that in this program's current set of DATA the sharp and flat routine is, in fact, redundant. You might like to try writing in DATA for some alternative melodies. The order of variables in the DATA statement is as follows:

Channel one Note symbol/Octave, Channel one Duration, Channel two Note symbol/Octave, Channel two Duration, Channel three Note symbol/Octave, Channel three Duration, Channel one Note . . . etc.

It is equally useful to realise that there is another method of making noises that can sometimes be overlooked by those who have not been introduced to it. If you possess a Sinclair Spectrum instead of a BBC then your only chance of making a racket via your computer is by using its beep command. The BBC also has a beep, as can be demonstrated by pressing CTRL-G.

## THE BELL TOLLS

There is also one simple way of using sound in a program which does not involve the SOUND or ENVELOPE statements at all. Typing VDU7 <RETURN> will give the same result as typing CTRL-G! This device is used in many programs in order to create a warning bleep or to confirm that an entry of some sort has been made. The pitch, duration and even the ENVELOPE of this bell sound can be altered by the simple expedient of using a few \*FX commands.

### \*FX 214: Bell Duration

The default duration is set at 7. To change it to, say, 50 we simply type \*FX 214,50.

### \*FX 213: Bell Pitch

### **\*FX 213: Bell Pitch**

The default pitch is set at 101 and could be altered to, say 200 by typing  
\*FX 213,200.

### **\*FX 212: Bell Amplitude**

The Amplitude value in this case is calculated by using the following formula:

$$\text{Value} = 256 + (\text{Amplitude} - 1) * 8$$

To set an amplitude of -4, for example, we would therefore calculate the value as:

$$\text{Value} = 256 + (-4 - 1) * 8 = 256 - 40 = 216$$

---



after doing this we would have to type `*FX 212,216`. An ENVELOPE can also be chosen using this value. In this case the value to be used is calculated by the formula:

$$\text{Value} = (\text{ENVELOPE No.} - 1) * 8$$

Therefore, to select ENVELOPE four, since  $(4 - 1) * 8 = 24$ , we would type `*FX 212,24`

### **\*FX 211: Bell channel**

The default channel used is channel 3. If we wished to select the noise channel we would have to type `*FX 211,0`. As is the case with standard sound statements, the Pitch value controls which noise is selected.

### **\*FX 210: Sound On/Off**

This `*FX` command has the useful facility of being able to disable the sound output entirely. Any value other than zero following the command will stop all sound: `*FX210,1 . . . Sound Off`

`*FX210,0 . . . Sound On`

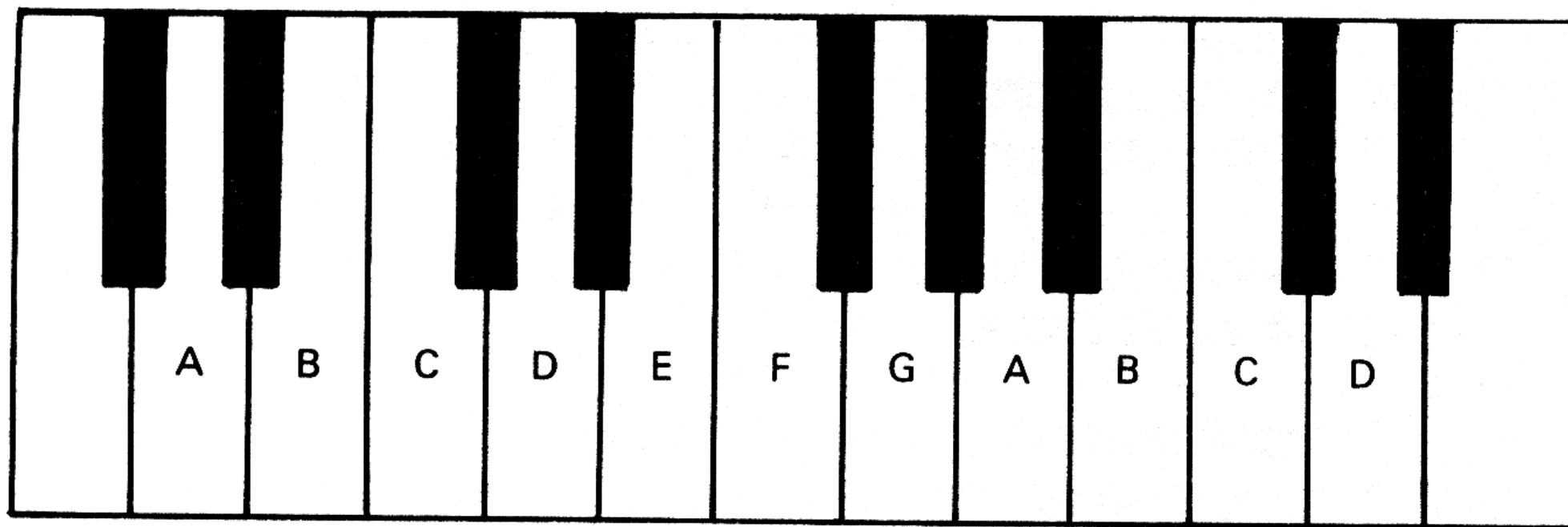
## 5. Music Theory for Micros

If you stop to consider the matter you will realise that literally every subject of any complexity inevitably acquires its own specialist vocabulary. Music is no exception to this general rule for just as the pioneers discovered that it was necessary to develop languages like BASIC and FORTH to facilitate the use and development of computer technology, so the world of music also has a special language all of its own. Since the scope of music only encompasses pitched notes and rhythm, however, its structure is considerably simpler than that of any computer language. For this reason I believe that most newcomers to music theory who have had some general computer experience will pick up the rudiments of music theory quite easily.

I have felt it necessary to digress somewhat and emphasize this point because of the experience of music teaching many of you will doubtlessly have had at school. For me, school music lessons comprised of note learning, primarily concerned with absorbing key signatures, composers' names and other pieces of musical gobbledygook. In this chapter I do not intend to bombard you with meaningless mnemonics designed to make it easier for you to remember the notes of the treble clef. This type of knowledge can be acquired quite naturally through growing familiarity with music itself. Instead, I intend to provide you with a vocabulary which will allow you to understand the more musically technical sections of this book and, show you how easy it is to use sheet music as reference material.

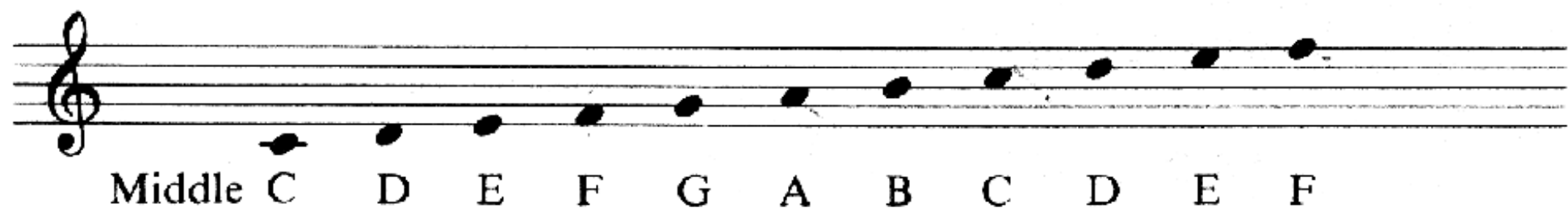
### PITCH NOTATION

In order to make it possible for us to speak about individual note pitches we must first start off by giving each note a name. The accepted method for doing this is to call them after the letters in the alphabet between A and G. If you look at a piano keyboard, you will see that the notes are found at the positions shown below:



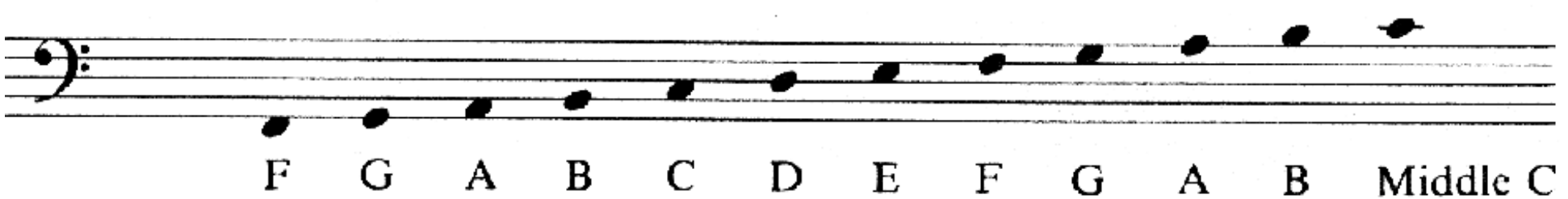
On the BBC Micro we have pitch values such that the A note above Middle C=89, B=97, C=101, D=109, E=117, F=121, G=129 and, for the A note one octave above the first, A=137. As you can see, this method of notation has its drawbacks. All the As, in whatever octave, have the same name. Bear in mind that a note which is one octave above another has double that note's frequency.

A better method for illustrating pitch is to abandon numbers and letters altogether and draw them on a musical stave, as is illustrated below:



In this case there is no ambiguity in discriminating between notes of different octaves. In addition, the concept of higher notes being situated higher up the staff is both extremely helpful and logical. A staff is, of course, the name given to the standard arrangement of five lines!

The symbol at the beginning of the staff is called a Treble Clef. This clef is used for the treble register of a piano and all instruments that play in that range, such as flutes, oboes, guitars etc. Bass instruments have their own clef, unsurprisingly called the Bass Clef. This is used by instruments such as trombones, bassoons and double basses. It looks like this:



A piano covers such a large range that it uses both clefs, and we can correlate piano keyboard and notation as follows, with each white note having its own line or space. BBC Micro pitch values are shown also.

PIANO KEYBOARD

C#	D#	F#	G#	A#	C#	D#	F#	G#	A#	C#	D#	F#	G#	A#	C#	D#	F#	G#	A#	C#																	
9	17	29	37	45	57	65	77	85	93	105	113	125	133	141	153	161	173	181	189	201	209	221	229	237	249												
A	B	C	D	E	F	G	A	B	C	D	E	F	G	A	B	C	D	E	F	G	A	B	C	D	E	F	G	A									
1	5	13	21	25	33	41	49	53	61	69	73	81	89	97	101	109	117	121	129	137	145	149	157	165	169	177	185	193	197	205	213	217	225	233	241	245	253



The above diagram includes extra notes which are either a semitone higher (sharp) or lower (flat) to the white notes. The sharp sign is # and the flat sign  $\flat$ . These are the black notes on the piano and make up the full complement of twelve notes used in Western music.

It is worth pointing out at this juncture that the rules and notations described in this chapter apply only to Western music. If we were playing Indian music, for instance, a completely different set of rules would apply. This is analogous to the use of different languages in computers for while the vocabulary itself might seem to be different, the general principles which apply are, of course, the same.

The only other area you need to know about before you are ready to read and write musical pitches concerns key signatures and keys. Different keys arise because of the particular tone intervals used in Western scales. An interval is the pitch difference between any two notes. Turning back to the piano keyboard, you, will see that if you play two adjacent notes, ie.: C to C or E to F, the interval would be one semitone. This is the smallest interval possible in Western music.

There are twelve semitones in every octave, but as most of you will know the scales normally used only use eight of these notes. For example, the scale of C Major comprises of the notes:

C D E F G A B C

This is because the intervals between the notes of a Major scale always conform to a particular pattern:



Where a 2 indicates 2 semitones or one full tone difference and 1 shows a 1 semitone change between the notes. As you can probably guess, if you start your scale on a note other than C, say F#, it would be necessary to play some black notes on the piano and include some sharps or flats in the notation of the scale.

Let us look at the scale of F#, which would be:

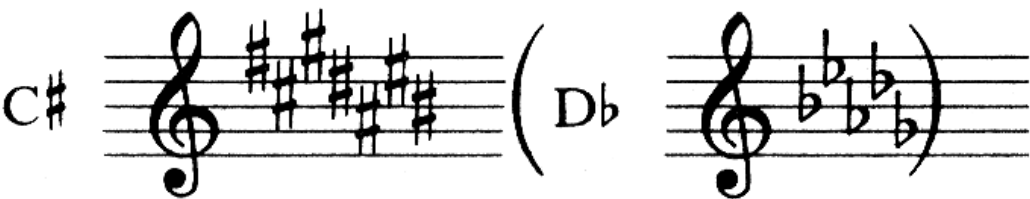
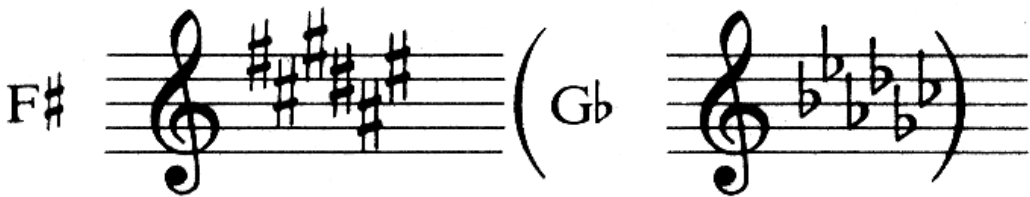
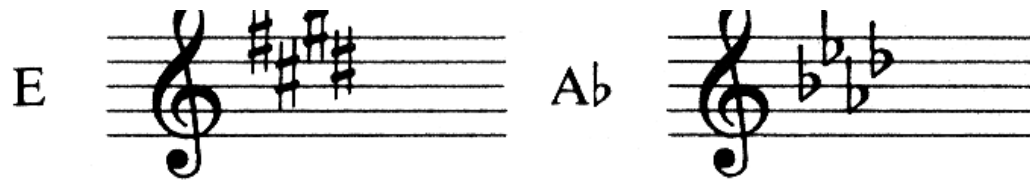


You have probably guessed that it would prove rather tedious writing sharps every time they occurred in F # Major so they are written on the staff just once, at the beginning of the piece of music to be played in that key, as follows:



This is called the key signature and each of the twelve possible scales (count them!) has its own unique signature.





Should we need a note which is *not* to be played sharp or flat as required by the Key signature it has a natural sign,  $\natural$ , placed before it. This effect lasts for a bar, and if the same note reverts to the key it requires a sharp or flat sign to signify this.

You will no doubt be gratified to learn that you are now equipped with enough information to take a piece of sheet music and convert the musical notation into pitch information for the computer - and vice versa! Alternatively, you could write a program to do the job for you, but you would be well advised to wait until you've availed yourself of the techniques still to come!

#### DURATION NOTATION

The use of the staff form of musical notation really comes into its own when dealing with durations. Various symbols are used to indicate the length of a note. These are as follows:

**1 Semibreve (Whole note)**

**equals**

**2 Minims**

**or**

**4 Crotchets**

**or**

**8 Quavers**

**or**

**16 Semi-quavers**

**or**

**32 Demisemi-quavers**

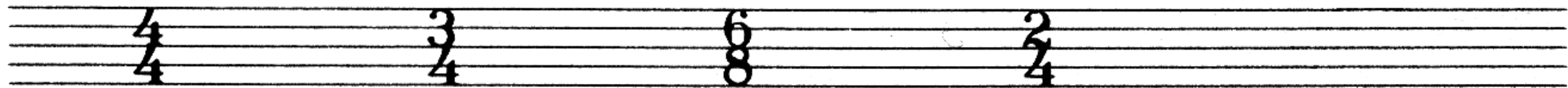
A dot after a note increases its value by half, so that is equivalent to

These note lengths are defined relative to each other, and are completely independent from the tempo or speed of the music, which defines the length of a bar (see below). Each note duration symbol has its equivalent note rest symbol, which is used to mark the periods of silence which fall between notes:

		or			

It would of course be perfectly possible simply to string crotchets and quavers in a line across the page, with the only breaks in the line consequently being the ends of pages. This would, however, prove rather confusing, don't you agree! One better way of dealing with this problem would be to group them in easy-to-cope-with bundles, rather like stringing words together within a sentence. The musical name for one of these bundles is a Bar and the length of each bar is

given by the time signature. The time signature is written at the beginning of a piece of music, directly after the key signature:



The upper figure is the number of beats per bar length and the lower figure gives the length of each beat. Hence 3/4 indicates three beats per bar with each beat a quarter note, i.e. a crotchet. 6/8 would indicate six beats per bar with each beat an eighth note, i.e. six quavers per bar. The most common time signatures you will encounter are 4/4 (four crotchets per bar) and 3/4, waltz time.

All that now remains to be added to this body of information is the final piece which is required to define that rhythm of a tune. And that is the tempo. Tempo is normally given in terms of crotchets per minute. So ♩=120 would mean 120 beats per minutes. If you look at classical music, you will notice that various Italian words are used to indicate the overall tempo of a piece. A selection of the commonly encountered terms are listed below.

Classical Tempo Terms

Adagio	Slow, leisurely
Allegro	Lively, reasonably fast
Andante	Moderate pace
Grave	Very slow
Largo	Slow and stately
Lento	Slowly
Presto	Very quick
Prestissimo	As fast as possible
Vivace	Quick

When programming music on the BBC tempos can be converted into time values by using the updated Metronome program which follows below. You will notice that I use the words ‘time values’ and not ‘duration’ in this case. As I mentioned earlier, this is because the duration parameter’s 1/20th of a second resolution is not great enough to provide for the full range of possible tempos. For instance, if a tempo of 120bpm is required, then there is no problem. That is to say 120bpm=2bps or 10 (20/10) duration values for a crotchet. However, if, say, we wanted a tempo of 119bpm this would give 1.95bps or 20/1.95=10.256... duration values.

The BBC would round this non-integer value down to ten, which would produce a tempo of 120bpm just as before. The next slower usable tempo is 100, for a crotchet value of 12. Not only would this be true, but even at 120bpm quavers are the shortest notes allowed. Semiquavers would give a D value of 2.5, which is also unacceptable. Refer to the table given on page 22.

Let us now get back to Metronome, shall we? The first thing we notice is that this means an alternative method for inputting timing must be found. One possible available method is to use the TIME variable. By controlling the length of a REPEAT... UNTIL loop of the type:

```
TIME=0: REPEAT:UNTIL TIME < length%
```

we could resolve our timing down to around 1/100th of a second. It must be noted that for certain applications this is still not fine enough, however.

Another method would be to resort to a machine code routine. The simplest and most effective BASIC method available is to use a FOR... NEXT loop, however, as is done here:

```
10 *KEYO MODE6:M VDU19,0,4,0,0,0:M VD
U14:M LIST:M
20*KEY1 RUN:M
```



```

30
40 REM      ***** METRONOME-2 *****
50
60 MODE7
70 VDU23;8202;0;0;0;
80 PRINT TAB(5,4);CHR$(129);CHR$(141)
; " * * * ";CHR$(132); "METRONOME";CHR$(129
); " * * * "
90 PRINT TAB(5,5);CHR$(129);CHR$(141)
; " * * * ";CHR$(132); "METRONOME";CHR$(129
); " * * * "
95 REPEAT
97 PRINT TAB(26,13) "
"

100 INPUT TAB(14,13); "Enter TEMPO",d%
102 IF d%>1000 OR d%<1 THEN PRINT TAB(
26,13);CHR$(134); "NO GOOD":GOTO100
105 UNTIL TRUE:PRINT TAB(26,13) "
"

```

```

110 D%=3000000/d%
120 PRINT TAB(14,13); "
"

```

---

## 78 Music Theory for Micros

```

130 PRINT TAB(14,13);CHR$(131);"TEMPO"
140 PRINT TAB(2,20);CHR$(134);"Tap cur
sor up to increase TEMPO"
150 PRINT TAB(2,21);CHR$(134);"Tap cur
sor down to decrease TEMPO"
160 PRINT TAB(2,22);CHR$(134);"Press c
ursor left to restart"
170 PRINT TAB(2,23);CHR$(134);"Press c
ursor right to END"
180 *FX4,1
190 *FX11,0
200 ENVELOPE 1,1,0,0,0,0,0,0,126,-54,0
,-10,126,4
210 REPEAT
220 FOR A%=0 TO D%:NEXT

```

```
230 d%=300000/D%
240 PRINT TAB(15,15);CHR$(141);d%;"
"
250 PRINT TAB(15,16);CHR$(141);d%;"
"
260 SOUND &10,1,4,-1
270 K=INKEY(0)
280 IF K=139 THEN D%=D%-20: IF D%=0 LET
D%=1
290 IF K=138 THEN D%=D%+20
300 IF K=136 THEN SOUND&11,0,1,1:RUN
310 UNTIL K=137
320 DEF PROCend
330 *FX4,0
340 VDU23;8202;0;0;0;
350 *FX11,11
360 CLS
370 PRINT TAB(10,10)"THAT'S ALL"
380 END
390 ENDPROC
```

For a full explanation of how “Metronome—2” uses an extended SOUND statement to produce a more sophisticated duration control see the previous chapter.

One comprehensive and inexpensive source for the novice micro-musician of musical reference material is called the *Rudiments and Theory of Music*, and is produced by the Associated Board of the Royal Schools of Music. However, for our purposes, that’s enough of the nuts and bolts of music theory! This, however, is only one small part of the story. During

---

the rest of this chapter I should like to turn to the consideration of some of the more useful musical ideas that have been developed over the past few hundred years. Let us start this quest by seeing how musicians gradually came to assemble notes into scales and arpeggios.

#### SCALES AND ARPEGGIOS

As Western music gradually developed over the centuries it was found that of the twelve possible notes (including sharps and flats) some were used more than others. Initially this was because crumhorns, bagpipes and the like were only capable of playing certain notes of the scales. It was left to Bach to devise a system whereby the scales of all the instruments played in his day were incorporated into one ‘even-tempered’ scale called the chromatic scales.



With the arrival of the chromatic scale any number of scales (sequences of notes) could henceforth be derived. The most common of these is the Major scale, as illustrated previously. The importance of the scale lies in establishing a note priority. In the key of C Major any note from the scale will sound more ‘in tune’ than any note outside it. Older forms of music such as that popularized by Bach, as well as fledgling jazz, seldom if ever strayed to notes outside the scale in use. More modern forms, such as bebop or music of composers like Bartok, on the other hand, positively revel in the use of ‘outside’ notes. Either way, it goes without saying that the scale is still an invaluable melodic framework.

Other scales commonly used in various forms of music are the Harmonic and Melodic Minor, the Natural Minor, Blues and the primitive Pentatonic. In addition to scales, a second level of note priority is provided by arpeggios, so let us now turn to survey precisely what is meant by the term.

An arpeggio, in its basic form, is a broken chord made up of the 1st, 3rd and 5th notes of whatever scale happens to be in use. For instance, this sequence would make the arpeggio of C Major:



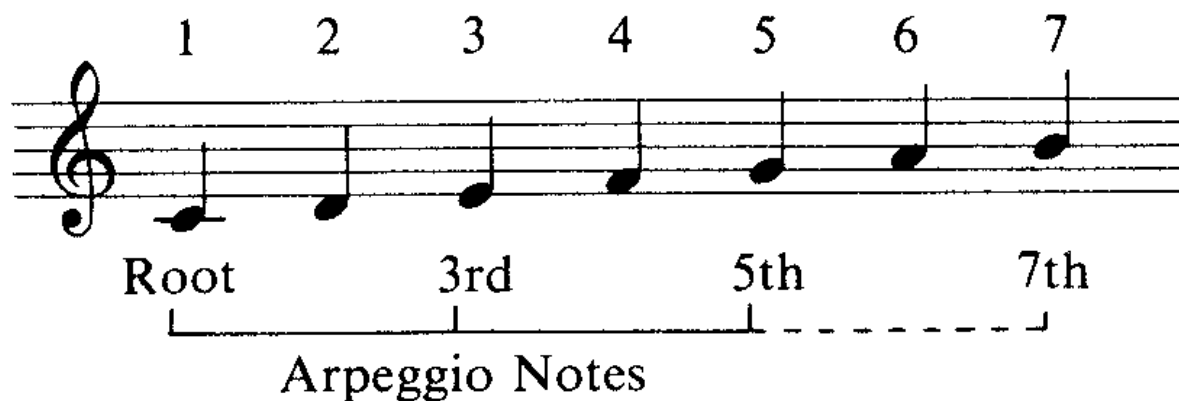
## 80 Music Theory for Micros

Minor arpeggios are based on any of the Minor scales, such as E Minor:



If we now go back to the C Major scale and look at its relative note priorities in conjunction with the C Major arpeggio, we get the following order:

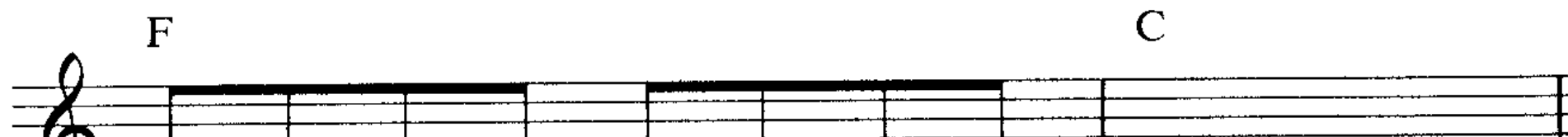
C	Root (1)	Perfect harmony by definition
E	3	} Harmonious
G	5	
B	7	Less in tune
D,F,A	2,4,6,	Least in tune

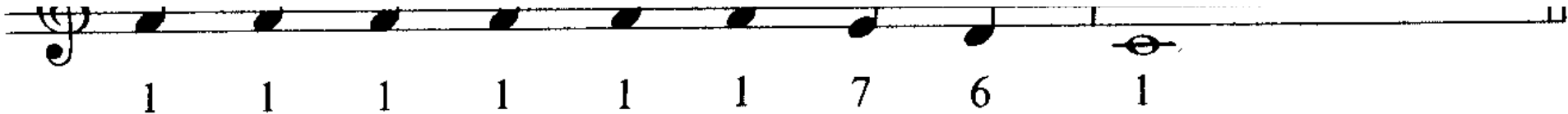


When you look at the average, popular piece of music, you will notice that a single scale is used throughout the whole piece. It is usual to have sequences of chords, however. A chord is the name given to a group of notes making up an arpeggio. Chords can be used in two distinct ways.

First of all, they dictate the note priority in use at any point in the tune. For example:

Chord: C





The majority of notes used are from the predominating chord. Other notes tend to be passing notes and are placed in less important places in the bar, that is to say not on the first or third crotchets in the bar.

Secondly, you must remember that chords also supply a method of

generating accompaniment by supplying notes which can be played simul-taneously with the melody notes. For an example, see the “Happy Birthday” program in the previous chapter. A chord sequence is the name given to the order in which the chords are played underneath any given melody. Because of the importance of the chord sequence in defining a harmonic accompaniment a method of notating chords had to be developed over the centuries. The basic chord symbol defines the note triad (three note chord) to be used. Look at the table below to see what I mean:

C...	C,E,G	1st, 3rd and 5th notes of the C Major scale
A...	A,C,E	1st, 3rd and 5th notes of the A Major scale
Dm...	D,F,A	1st, 3rd and 5th notes of the D Minor scale

In these cases the lack of a subscript indicates a Major chord. A small m indicates a Minor chord. There are various other symbols which you also might come across, including Dim, which indicates the 1st, 3rd and 5th of a diminished scale, and Aug, which indicates a Major chord with an augmented 5th note, i.e. the 5th is made a semitone sharper than it would be otherwise.

A number of additional notes can be added to the basic three-note chord if you wish to make 4,5 or even 6 note chords. Using the key of C, here are some examples:

Name	Added Notes	Notes	Chord
MAJ7	7th note of Major scale	CEGB	CMAJ7
7	Minor 7th note	CEGB	C7
6	6th note of Major scale	CEGA	C6
9	9th+Minor 7th notes	CEGB D	C9
11	11th+9th+7	CEGB DF	C11

These symbols are then written above the part of the melody to which they apply. This can be seen if you look at most sheet music, as well as examples scattered throughout the pages of this book.

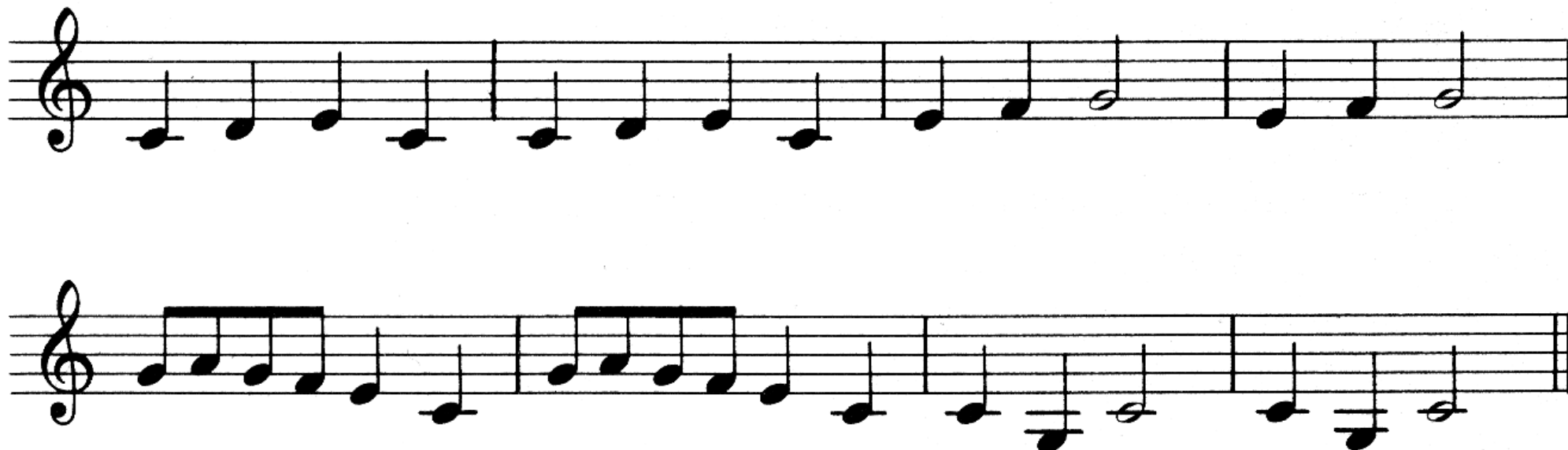
You will doubtlessly remember me saying earlier on in the text that melodies suggest particular chord sequences; it is equally true to say that chord sequences also suggest certain melodies. Melodies could be generated by describing a chord sequence and letting the computer generate rhythms and melodies. This can be accomplished using a constrained random numbers generator, the constraints, in their most basic form, being the harmonic considerations mentioned above. Further constraints which would also need to be considered would include song or piece structure and rhythm.

MUSICAL STYLES

Rhythm, or lack of it, is very important in determining the impact of a particular piece of music. Most of the popular styles of music, such as pop, folk and jazz, embrace and are distinguished by their strong and distinctive rhythms. More esoteric forms of classical music, however, are slightly less simple to pin down. We will therefore limit our discussion to general points of style and structure and go into more detail about specific areas in the chapter which concentrates on pinning down and therefore explaining automatic composition.

Virtually every form of music you can think of relies on and uses a certain amount of repetition as a structural device. If you think of the popular children’s song, ‘Frere Jacques’, for example, you will see how this device makes it possible to reinforce and emphasise themes by creating memorable refrains.

FRERE JACQUES



You will notice that this melody has a total of eight bars. Each bar contains a distinct phrase, and these phrases appear in the pattern AA BB CC DD, i.e. the first bar is played, and is then repeated. a new phrase is introduced in the third bar, then this is repeated - and so on.

While this may not be a particularly common type of structure, it does provide us with a very clear example, since each phrase is repeated exactly. It is, however, more common to have slightly different endings to each phrase.

The structures A A B A or sometimes A A B C are frequently encountered. The latter is a frequently used structure both in classical pieces and in popular songs. In this case, first the main theme is stated (this could be of any length and is usually eight bars in the case of songs) and then the theme is repeated. Sometimes it incorporates a variant ending which leads the song into the new theme (or middle eight, in the case of song structure). At the end of the middle eight either of two possibilities is open to the composer: In classical music, the most common option is the recapitulation of the original theme; in the case of the song either a recapitulation or the introduction of an ending-theme are equally common.

One device commonly used in composing both folk and pop music is to include a verse of eight or sixteen bars followed by a chorus. The purpose of the verse is to relate the song's narrative, and the chorus usually contains a simple, memorable 'hook' line which the audience can easily remember. This type of structure can still be used even when no words are being sung. The following original piece employs the verse/chorus format:

Music Theory for Micros 83

One device commonly used in composing both folk and pop music is to include a verse of eight or sixteen bars followed by a chorus. The purpose of the verse is to relate the song's narrative, and the chorus usually contains a simple, memorable 'hook' line which the audience can easily remember.



This type of structure can still be used even when no words are being sung. The following original piece employs the verse/chorus format:

```

10    REM    *** POP SONG ***
20
30    ENVELOPE1,1,0,0,0,0,0,0,122,-1,0,
-10,123,100
40    REPEAT
50        FOR R%= 1 TO 60
60            READ P%,D%
70            FOR C%=1 TO 3
80                SOUND C%,1,P%,D%
90            NEXT
100        NEXT
110        RESTORE 140
120        UNTIL FALSE
130
140    DATA 53,8,53,8,73,8,89,8
150    DATA 81,8,93,8,89,8,73,8
160    DATA 53,8,53,8,73,8,89,8
170    DATA 81,8,69,8,73,16

```

```

180 DATA 53,8,53,8,73,8,89,8
190 DATA 81,8,93,8,89,8,73,8
200 DATA 53,8,53,8,73,8,89,8
210 DATA 81,8,89,8,93,8,101,8
220 DATA 121,8,101,8,89,8,121,8
230 DATA 117,8,101,8,81,8,117,8
240 DATA 109,8,121,8,129,8,117,8
250 DATA 121,8,101,8,89,8,129,8
260 DATA 121,8,101,8,89,8,121,8
270 DATA 117,8,101,8,81,8,117,8
280 DATA 109,8,121,8,129,8,117,8
290 DATA 121,32

```

Another common musical form is known as the twelve-bar blues. This form employs what could be described as an A A B structure. The main theme is played, then repeated, after which a concluding theme is played. In this example the chords which are played are changed in a traditional sequence, as is illustrated below for the key of C:

Another common musical form is known as the twelve-bar blues. This form employs what could be described as an A A B structure. The main theme is played, then repeated, after which a concluding theme is played. In this example the chords which are played are changed in a traditional sequence, as is illustrated below for the key of C:

C/// C/// C/// C///

F/// F/// C/// C///  
G/// F/// C/// C///

Let us leave this topic for the moment, since blues and various other types and styles of music can be explored in greater depth when we come to apply some of our new-found musical knowledge to programming and BBC Micro. In the chapters which follow we shall also attempt to draw musical symbols and staves, generate random music in particular styles, interface the BBC with synthesisers and program from sheet music. Hopefully, on the basis of the expertise we have so effortlessly accumulated in this chapter, none of these tasks will prove insurmountable, and all will enhance our musical enjoyment and output.

## 6 Chords and Harmony

As we saw when we tried to create a three-note organ on the BBC, polyphony presents special problems for the programmer and the musician. The techniques we use for single note situations do not always transpose happily or smoothly into the multi-note case. Modifications are therefore essential. In this chapter we shall look at the problems which need to be summoned when creating polyphonic pieces of music.

If you think back to Chapter Four, you will remember that we dealt with a special case of polyphonic playback. In the piece 'Happy Birthday' the entire tune was played in three-part harmony; this was only possible because, rhythmically, each harmony part was identical. As soon as the harmony parts become rhythmically independent we are immediately faced with a problem. This is best illustrated by looking at the simplest possible example, a two-part melody.

### TWO PART TUNE SYNCHRONISATION

If The BBC was endowed with infinitely long SOUND queues, synchronisation would be straightforward. We would simply transfer our SOUND information into the channel one and channel two buffers and, providing we had INPUT the durations for each part correctly, the two channels would play back in perfect sync. However, since these queues are limited to five requests, if one channel is made up of short duration statements and the other of long duration statements, a breakdown in sync will eventually occur.

The program "Synchronisation Demo" illustrates this effect quite clearly. The program uses a pair of two-dimensional arrays to store the Pitch and Duration DATA for channels one and two. This information then feeds the SOUND statements in Line 120. Channel one is made up of long duration notes and channel two of short duration notes. RUN the program and see what happens...

```
10
20 REM"SYNCHRONISATION DEMO
30
40 DIM Pitch%(2,20),Duration%(2,20)
```

## 86 Chords and Harmony

```

50 FOR Channel%=1 TO 2
60 FOR A%=1 TO 20
70 READ Pitch%(Channel%,A%),Duration%
(Channel%,A%)
80 NEXT
90 NEXT
100
110 FOR A%=1 TO 20
120 SOUND1,-10,Pitch%(1,A%),Duration%(
1,A%):PRINTTAB(6);A%:SOUND2,-10,Pitch%(2
,A%),Duration%(2,A%):PRINTTAB(12);A%
140 NEXT
150
160
170 REM"CHANNEL ONE
180
190 DATA 53,20,61,20,69,20,73,20,53,20
,61,20,69,20,73,20,53,20,61,20
200 DATA 53,20,61,20,69,20,73,20,53,20
,61,20,69,20,73,20,53,20,61,20
210
220 REM"CHANNEL TWO
230
240 DATA 81,5,89,5,81,5,89,5,81,5,89,5
,81,5,89,5,81,5,89,5
250 DATA 81,5,89,5,81,5,89,5,81,5,89,5
,81,5,89,5,81,5,89,5

```

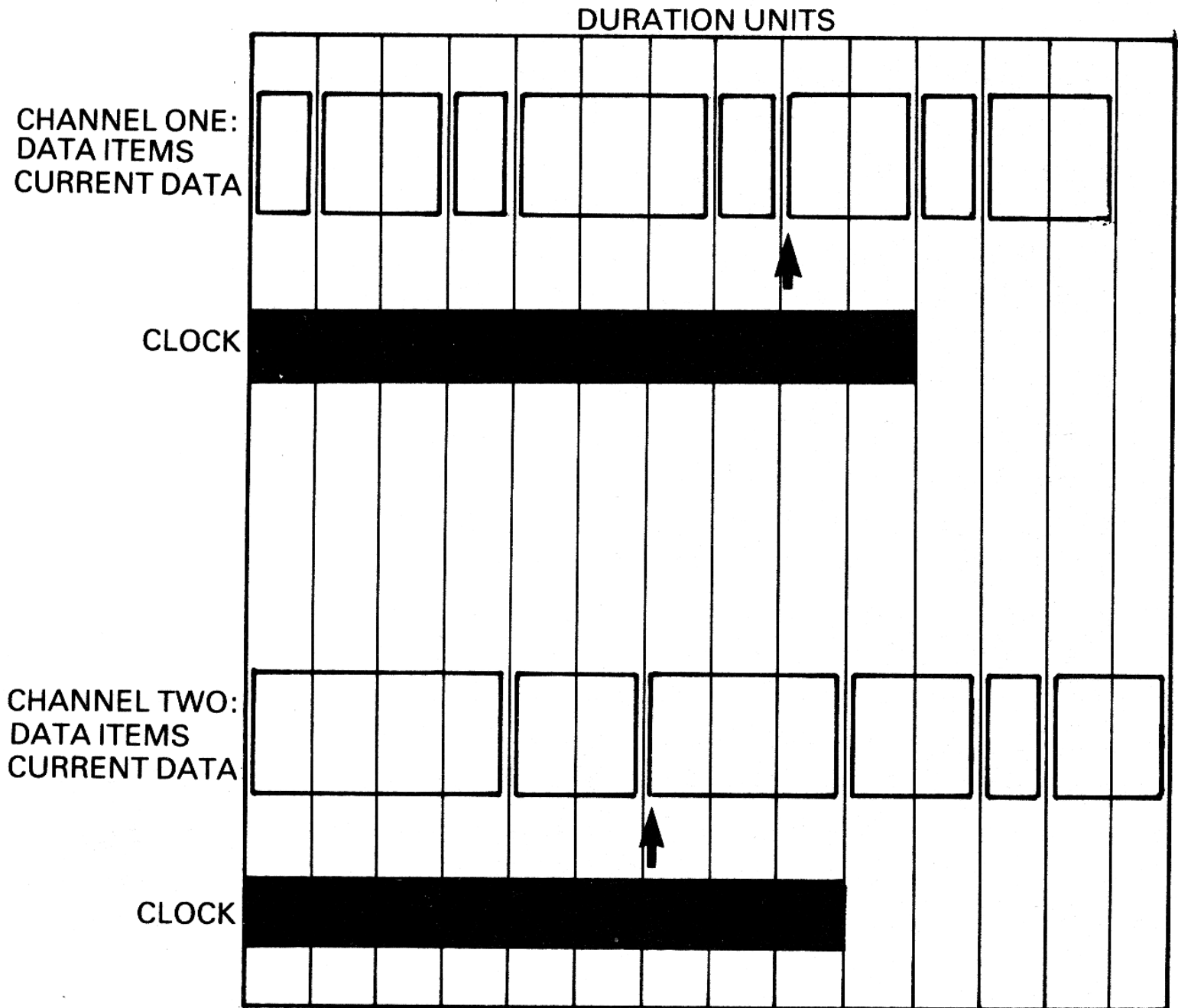
This program starts, as expected, with a long note on channel one synchronised with short notes on

channel two. Four notes on channel two should be played for every note played on one. When the second long note is played, however, channel two ties up and plays notes at the same time as each new notes on one. The display will show each channel starting a note and putting five notes in each queue. Since channel two is playing notes more frequently than one, by the end of the first note on one, four SOUNDS have been played on two.

At this point one more SOUND is added to each queue, but note that channel two is continuing to use up notes quicker than one. The SOUND statement in Line 120 controls how quickly new SOUND statements are put on both queues. Since this is filling the buffers at the rate of the slowest duration, by the end of the second note on channel one the channel two buffer is exhausted. From this point on, a new SOUND is only added to the channel two queue at the same rate as to the queue on channel one. As a result, channel one's queue will always be full, and two's will always be empty. This is why we get the effect of both parts SOUNDing together, even though their durations are quite different.

One solution to this problem is to arrange for the channel two buffer to be topped up more frequently than the channel one buffer. We could accomplish this in a number of ways.

Our first option is instead of having two separate DATA sources, to combine them into one. By recording the channel number as well as its duration and pitch, only one SOUND statement would be required. If you look back at the "Synchronisation Demo" example we met with earlier on, you will note that we could then arrange for each channel one DATA item to be followed by four items of channel two DATA. If we did this, the channel two buffer would never become exhausted, so synchronisation would necessarily have to occur. This technique makes program writing simple, but arranging the music DATA for a complicated piece tends to present horrendous difficulties. As a result the musical problems involved make this method impractical.



The second option is to retain our separate DATA statements and combine the musical information inside the program. One way of doing this would be to keep a record of the total duration played on each channel. In the case of the “Synchronisation Demo”, if the total duration on channel one exceeds that of channel two, channel two needs to be topped up. This technique can be thought of as something akin to keeping a clock for each channel.

As each channel is SOUNDED the duration for that statement is added to the clock for that channel. At any given time the channel with the clock reading the smallest value (channel two in the diagram) will be SOUNDED. This would give a statement which looks like the following formula:

```
IF clock1 > clock2 SOUND 2 ELSE SOUND 1
```

or equally well...

```
IF clock2 > clock1 SOUND 1 ELSE SOUND 2
```

We could incorporate this type of statement into our synchronisation demonstration as follows:

---

```

10
20  REM"SYNCHRONISATION DEMO TWO
30
40  REM"SET UP ARRAYS
50
60  DIM Pitch%(2,20),Duration%(2,20)
70  FOR Channel%=1 TO 2
80  FOR A%=1 TO 20
90  READ Pitch%(Channel%,A%),Duration%
(Channel%,A%)
100 NEXT
110 NEXT
120
130 A%=1:a%=1
140 clock1=0:clock2=0
150
160 REM"THE PROGRAM
170
180 REPEAT
190 IF clock2>clock1 SOUND1,-10,Pitch%
(1,A%),Duration%(1,A%):clock1=clock1+Dur
ation%(1,A%):A%=A%+1:ELSE SOUND2,-10,Pit
ch%(2,a%),Duration%(2,a%):clock2=clock2+
Duration%(2,a%):a%=a%+1
200 UNTIL a%=20

```



```
200 UNTIL a%=20
210
220 REM"THE DATA
```

---

## Chords and Harmony

```
230
240 REM"CHANNEL ONE
250
260 DATA 53,20,61,20,69,20,73,20,53,20
,61,20,69,20,73,20,53,20,61,20
270 DATA 53,20,61,20,69,20,73,20,53,20
,61,20,69,20,73,20,53,20,61,20
280
290 REM"CHANNEL TWO
300
310 DATA 81,5,89,5,81,5,89,5,81,5,89,5
,81,5,89,5,81,5,89,5
320 DATA 81,5,89,5,81,5,89,5,81,5,89,5
,81,5,89,5,81,5,89,5
```

This second demonstration RUNs in perfect sync throughout. A REPEAT... UNTIL loop is used in this case because the two channels must be allowed to run independently of each other. Two step counting variables, respectively A% and a%, are used to keep track of where we are at any given point in each of the two channel arrays. These are set to one at Line 13 and are increased by one every time a SOUND is inserted in a channel queue.

In this case, when the program first reaches Line 190 clock1 equals clock2. Channel two will therefore be SOUNDED. We then have clocks equalling 5 and a% equal to 2. The program will then go round the loop and the next visit to Line 190 will see clock2 > clock1 as a result of which channel one will be SOUNDED. Since this loop is executed in a fraction of a second both channels one and two will appear to be SOUNDing simultaneously. At this point clock1=20 and A%=2. On the third visit to line 190 clock2=5 and clock1=20, so clock2 < clock1 and therefore channel two will add another SOUND to its queue. Now clock2=10 and a%=3. On a fourth visit clock2=15 and a%=4. As you can see channel two is filling its queue far more rapidly than channel one. Since it is also emptying its queue faster, the end

result balances out at precisely what we want.

This approach will work for any set of two-part musical DATA, not merely for the example I have chosen. As you can see, therefore, we do have a general method for writing two-part tunes which allows the musical DATA for both parts to be listed separately. Over the rest of the chapter we will develop this method further, in order to make it easy for us to cope with more complex musical examples. The first of these is a melody with bass accompaniment, which includes rests and repetitions.

“Blood and Sand” is an original piece of music which has been written especially for this book by the author. The two parts consist of a melody line and a bass part.

---

## 90 Chords and Harmony

```

10
20
30 REM"BLOODANDSAND
40
50
60 REM"SET UP VARIABLES etc....
61
62 MODE7:VDU23;B202;0;0;0;
71 PRINTTAB(7,10);CHR$(141);CHR$(129)
;"Blood";CHR$(134);"and";CHR$(131);"Sand
";TAB(7,11);CHR$(141);CHR$(129);"Blood";
CHR$(134);"and";CHR$(131);"Sand"
80 Clock1%=0:Clock2%=0
90 ENVELOPE 1,1,0,0,0,0,0,0,67,-1,-1,
-1,100,90
100 ENVELOPE 2,1,0,0,0,0,0,0,126,-4,-1
,-2,116,70
110 ENVELOPE 3,1,0,0,0,0,0,0,126,-1,0,
-1,0,0
120
130 REM"SET UP ARRAYS

```

```

130 REM"SET UP ARRAYS
140
150 DIM Pitch%(500,2),Duration%(500,2)
160 E2%=2
170 C%=1
180 REPEAT
190 P%=1:p%=1
200 REPEAT
210 READ Pitch%(P%,C%),Duration%(P%,C%
)
220 P%=P%+1
230 UNTIL Pitch%(P%-1,C%)=0
240 C%=C%+1
250 RESTORE 680
260 UNTIL C%=3
270
280 REM"THE PROGRAM
290
300 p%=1:P%=1
310 REPEAT
320 IF Pitch%(P%,1)=-1 E1%=3 ELSE E1%=
1

```

---

Chords and Harmony 91

```

330 IF Pitch%(p%,2)=-1 E2%=3 ELSE E2%=
2
340 IF Clock1%=Clock2% SOUND&101,E1%,P
itch%(P%,1),Duration%(P%,1):SOUND&102,E2
%,Pitch%(p%,2),Duration%(p%,2):Clock2%=C
lock2%+Duration%(p%,2):Clock1%=Clock1%+D
uration%(P%,1):p%=p%+1:P%=P%+1

```

```
uration%(P%,1):p%=p%+1:P%=P%+1
```

```
350 IF Clock2%>Clock1% SOUND1,E1%,Pitch%(P%,1),Duration%(P%,1):Clock1%=Clock1%+Duration%(P%,1):P%=P%+1
```

```
360 IF Clock1%>Clock2% SOUND2,E2%,Pitch%(p%,2),Duration%(p%,2):Clock2%=Clock2%+Duration%(p%,2):p%=p%+1
```

```
370 UNTIL Pitch%(P%,1)=0
```

```
380 GOTO 300
```

```
390
```

```
400 REM"THE DATA: Pitch,Duration...
```

```
410
```

```
420 REM"CHANNEL ONE:THE BASS
```

```
430
```

```
440 DATA 41,5,89,5,41,5,89,5,53,5,101,5,53,5,101,5,49,5,97,5,49,5,97,5,61,5,109,5,61,5,109,5
```

```
450 DATA 41,5,89,5,41,5,89,5,53,5,101,5,53,5,101,5,49,5,97,5,49,5,97,5,61,5,109,5,61,5,109,5
```

```
460 DATA 41,5,89,5,41,5,89,5,53,5,101,5,53,5,101,5,49,5,97,5,49,5,97,5,61,5,109,5,61,5,109,5
```

```
470 DATA 41,5,89,5,41,5,89,5,53,5,101,5,53,5,101,5,49,5,97,5,49,5,97,5,61,5,109,5,61,5,109,5
```

```
480 DATA 33,5,81,5,33,5,81,5,33,5,81,5,33,5,81,5,33,5,81,5
```

```
490 DATA 49,5,97,5,49,5,97,5,49,5,97,5,49,5,97,5,49,5,97,5
```

```
500 DATA 33,5,81,5,33,5,81,5,33,5,81,5,33,5,81,5,33,5,81,5
```

,33,5,81,5  
 510 DATA 49,5,97,5,53,5,101,5,61,5,109  
 ,5,49,5,97,5  
 520 DATA 33,5,81,5,33,5,81,5,33,5,81,5  
 ,33,5,81,5  
 530 DATA 49,5,97,5,49,5,97,5,49,5,97,5  
 ,49,5,97,5

---

## 92 Chords and Harmony

540 DATA 33,5,81,5,33,5,81,5,33,5,81,5  
 ,33,5,81,5  
 550 DATA 49,5,97,5,53,5,101,5,61,20  
 560 DATA 41,5,89,5,41,5,89,5,53,5,101,  
 5,53,5,101,5,49,5,97,5,49,5,97,5,61,5,10  
 9,5,61,5,109,5  
 570 DATA 41,5,89,5,41,5,89,5,53,5,101,  
 5,53,5,101,5,49,5,97,5,49,5,97,5,61,5,10  
 9,5,61,5,109,5  
 580 DATA 41,5,89,5,41,5,89,5,53,5,101,  
 5,53,5,101,5,49,5,97,5,49,5,97,5,61,5,10  
 9,5,61,5,109,5  
 590 DATA 41,5,89,5,41,5,89,5,53,5,101,  
 5,53,5,101,5,49,5,97,5,49,5,97,5,61,5,10  
 9,5,61,5,109,5  
 600 DATA 41,5,89,5,41,5,89,5,53,5,101,  
 5,53,5,101,5,49,5,97,5,49,5,97,5,61,5,10  
 9,5,61,5,109,5  
 610 DATA 41,5,89,5,41,5,89,5,53,5,101,  
 5,53,5,101,5,49,5,97,5,49,5,97,5,61,5,10  
 9,5,61,5,109,5  
 620 DATA 0,0,0,0,0,0,0,0,0,

```
620 DATA 0,0,0,0,0,0,0,0,0,
630
640
650 REM"CHANNEL TWO:THE TUNE
660
670
680 DATA -1,5,137,5,145,5,149,5,145,5,
137,10,137,10,137,5,145,5,149,5,145,5,13
7,20
690 DATA 137,5,145,5,149,5,145,5,137,1
0,137,10,137,10,137,10,133,10,137,10
700 DATA 137,5,145,5,149,5,145,5,137,1
0,137,10,137,5,145,5,149,5,145,5,137,20
710 DATA 137,5,145,5,149,5,145,5,137,1
0,137,10,137,10,137,10,145,10
720 DATA 129,45,145,40,129,40,145,40
730 DATA 129,40,145,40,129,40,145,10,1
49,10,157,25
740 DATA 137,5,145,5,149,5,165,10,149,
5,145,10
750 DATA 185,5,185,10,185,5,185,5,177,
5,185,10
```

---

```

760 DATA 137,5,145,5,149,5,165,10,149,
5,145,10
770 DATA 185,5,185,10,185,5,185,5,177,
5,185,10
780 DATA 137,5,145,5,149,5,165,10,149,
5,145,10
790 DATA 185,5,185,10,185,5,185,5,177,
5,185,10
800 DATA 137,5,145,5,149,5,165,10,149,
5,145,10
810 DATA 185,5,185,10,185,5,185,5,177,
5,185,165
820 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

```

When the program is RUN you will notice a slight pause which occurs as the arrays load up before the actual melody starts. In this piece we are being presented with an example which is similar to our Sync Demo. Thus, you will notice that the bass part consistently plays more notes than the melody part. It should be obvious to you by now that any attempt to relate the two parts when writing the DATA statements would be next to impossible. The program falls into three parts; as follows.

1. Initialisation: 'Blood and Sand' is PRINTed on the screen using MODE 7 double height text. Both clocks are set to zero and three ENVELOPE are defined. The first and second are for channels one and two, while the third is a zero volume ENVELOPE called up for rests. When the pitch and duration information is READ into the arrays, Pitch%(P%-1,C%)=0 is tested to indicate the end of DATA for that channel when the condition is satisfied. This allows this program to be used for any set of two-part DATA providing zeroes are placed at the end of each set of channel information. C% is the channel number and Line 250 RESTOREs the program to the channel two set of DATA.

2. The Program: Line 300 sets the step counters to one. Lines 320 and 330 allow for rests in the music by INPUTting -1 as the pitch DATA. If Pitch= -1 then the silent ENVELOPE 3 is called. The main part of the program is in a slightly different form from that which we have already seen in the demo. Line

340 synchronises the channels if their SOUND statements fall on the same beat, i.e. when clock1 =clock2. This move makes sure that the two parts start exactly at the same time and are periodically pulled back into time. Even though the duration values of the sound generator are highly accurate, the relative slowness of BASIC can sometimes cause a time delay to occur between the two parts. This line ensures any delay which might be generated as a result of this flaw has no effect. Because of the addition of this line it becomes necessary to split the clock comparison statement into two lines, 350 and 360. Overall, these three lines have the same effect as we have noted earlier on. The channel with the smallest clock reading always SOUNDS. Line 370 tests for zeros in the DATA statements and, if found, sends the program back to Line 300 where the step counters are reset, causing the tune to play from the beginning. This program will REPEAT indefinitely and stay in perfect synchronisation.

3. The Music DATA: The DATA is written as BBC pitch and duration values, pitch followed by duration followed by pitch and so on. There is no reason why any of the other available forms for recording musical information, such as music nomenclature or symbols, MC-4 synthesiser code or a form of your own devising, should not be used. Providing the translation of DATA into BBC BASIC is carried out before being stored in the pitch and duration arrays, no slowing down of the program will occur apart from the unavoidable pause as the arrays load, which we have already noted and considered. It is a good idea to be systematic about the amount of information you put in each DATA statement. In this example, two bars per statement seemed reasonable. A systematic approach also has the additional benefit of making it fairly easy to trace back and eradicate the inevitable typing errors when you become horrified to hear that the playback sounds like some form of avant garde jazz!

Now that we have successfully tackled two-part tune synchronisation we can try a similar technique on three- and four-part works.

## MULTIPLE PART TUNE SYNCHRONISATION

I have written another original piece as a demonstration of multiple part synchronisation. “Vermillion Sands” has been composed as an electronic piece of music suitable for four voices. It comprises lead part on channel one, bass on two, counter melody on three and snare drum on channel zero, the noise channel.

```

10
20
30 REM"    ***FOUR VOICE SYNC***
31
32 REM  "*****VERMILLION SANDS*****
```



41

42 REM" SET UP

Chords and Harmony 95

```

43
45  ENVELOPE 1,1,0,0,0,0,0,0,126,-4,-
1,-3,110,50
50  ENVELOPE 2,1,0,0,0,0,0,0,30,-1,-1,
-1,125,100
60  ENVELOPE 3,1,0,0,0,0,0,0,126,-4,-2
,-126,126,0
65  ENVELOPE 4,1,0,0,0,0,0,0,126,-4,-
3,-1,110,0
70  DATA 178,5,194,5,206,5,228,15,178,
5,194,5,206,5,228,10
80  DIM Pitch%(3,300),Duration%(3,300)
,clock%(3),N%(3)
81  MODE 7:VDU23;8202;0;0;0;
82  PRINTTAB(7,10);CHR$(141);CHR$(129)
;"Vermillion";CHR$(131);"Sands";TAB(7,11)
;CHR$(141);CHR$(129);"Vermillion";CHR$(
131);"Sands"
83
84  REM"THE PROGRAM
85
88  PROCinit(0)
90  PROCinit(1)
100  PROCinit(2)
110  PROCinit(3)

```

```
110 PROCinit(3)
120 PROCsync(3)
123
130 REM"SYNCRONIZE VOICES
131
140 DEF PROCsync(C%)
150     sync = (C%-1)*%100
160     FOR Channel%=0 TO C%
170         SOUND sync+Channel%,0,0,5
180         clock%(Channel%)=0 : N%(Channel
1%)=0
190     NEXT Channel%
200     REPEAT
210         S%=1
220         FOR Channel%=0 TO C%
230             IF clock%(Channel%)<clock%(S
%) THEN S%=Channel%
240         NEXT Channel%
250         PROCplay(S%)
```

---

96 Chords and Harmony

```
260 UNTIL Fitch%(1,N%(1)-1)=0 AND cloc
k%(1)>110:FOR Channel%=0 TO 3:clock%(Cha
nnel%)=0:N%(Channel%)=0:NEXT:GOTO200
270 ENDPROC
271
272 REM"PLAY CHANNEL
273
280 DEF PROCplay(Channel%)
290     N%(Channel%)=N%(Channel%)+1
300     A%=N%(Channel%)
```

```
300   A%=N%(Channel%)
310   clock%(Channel%)=clock%(Channel%
)+Duration%(Channel%,A%)
320   IF Pitch%(Channel%,A%)=-1 THEN E
%=0 ELSE E%=Channel%+1
330   SOUND Channel%,E%,Pitch%(Channel
%,A%),Duration%(Channel%,A%)
340   ENDPROC
342
343   REM"LOAD ARRAYS
344
350   DEF PROCinit(Channel%)
360   IF Channel%=0 RESTORE 4000 ELSE RE
STORE Channel%*1000
370   A%=1
380   REPEAT
390   READ Pitch%(Channel%,A%),Duration%
(Channel%,A%)
400   A%=A%+1
410   UNTIL Pitch%(Channel%,A%-1)=0
420   ENDPROC
500
600   REM"THE FITCH AND DURATIONS
700
800
900   REM"THE MELODY
908
1000  DATA 109,20,117,10,121,10,97,20,10
1,10,109,10
1010  DATA 81,20,89,10,97,10,69,40
1020  DATA 109,20,117,10,121,10,97,20,10
1,10,109,10
1030  DATA 81,20,89,10,97,10,69,40
```

1030 DATA 81,20,89,10,97,10,69,40  
1040 DATA 97,20,89,20,85,20,69,20,61,20  
,89,20,85,40

---

1050 DATA 89,10,137,10,109,10,117,10,89  
,40  
1060 DATA 89,10,137,10,109,10,117,10,89  
,40  
1070 DATA 89,10,137,10,109,10,117,10,89  
,40  
1080 DATA 89,10,137,10,109,10,117,10,89  
,20,97,20,101,160  
1900 DATA 0,0,0,0,0,0,0,0,0,0  
1991  
1992 REM"THE BASS  
1995  
2000 DATA 13,5,13,10,13,5,13,20,13,5,13  
,10,13,5,13,20  
2010 DATA 13,5,13,10,13,5,13,20,13,5,13  
,10,13,5,13,20  
2020 DATA 13,5,13,10,13,5,13,20,13,5,13  
,10,13,5,13,20  
2030 DATA 13,5,13,10,13,5,13,20,13,5,13  
,10,13,5,13,20  
2040 DATA 21,5,21,10,21,5,21,20,21,5,21  
,10,21,5,21,20  
2050 DATA 21,5,21,10,21,5,21,20,21,5,21  
,10,21,5,21,20  
2060 DATA 25,5,25,10,25,5,25,20,25,5,25  
,10,25,5,25,20

```
,10,25,5,25,20
2080 DATA 33,5,33,10,33,5,33,20,33,5,33
,10,33,5,33,20
2090 DATA 25,5,25,10,25,5,25,20,25,5,25
,10,25,5,25,20
2100 DATA 33,5,33,10,33,5,33,20,33,5,33
,10,33,5,33,20
2110 DATA 13,5,13,10,13,5,13,20,13,5,13
,10,13,5,13,20
2120 DATA 13,5,13,10,13,5,13,20,13,5,13
,10,13,5,13,20
2900 DATA 0,0,0,0,0,0,0,0
2910
2920 REM"THE COUNTER MELODY
2930
3000 DATA -1,25,109,1,170,1,178,1,185,1
,198,1,206,50
3001 DATA -1,25,109,1,170,1,178,1,185,1
,198,1,206,50
```

---

## 98 Chords and Harmony

```
3002 DATA -1,25,109,1,170,1,178,1,185,1
,198,1,206,50
3003 DATA -1,25,109,1,170,1,178,1,185,1
,198,1,206,50
3010 DATA -1,5,166,10,166,5,158,10,158,
5,166,10
3020 DATA 166,10,166,5,170,10,170,5,166
,10
3030 DATA 166,10,166,5,158,10,158,5,166,
10
```

10

3040 DATA 166,10,166,5,170,5,166,5,170,  
5,178,5

3050 DATA 150,5,170,5,186,5,198,15,150,  
5,170,5,186,5,198,10

3060 DATA 150,5,170,5,186,5,198,10

3070 DATA 158,5,178,5,194,5,206,15,158,  
5,178,5,194,5,206,10

3080 DATA 158,5,178,5,194,5,206,10

3090 DATA 150,5,170,5,186,5,198,15,150,  
5,170,5,186,5,198,10

3100 DATA 150,5,170,5,186,5,198,10

3110 DATA 158,5,178,5,194,5,206,15,158,  
5,178,5,194,5,206,10

3120 DATA 158,5,178,5,194,5,206,15

3130 DATA 158,10,158,5,150,10,150,5,158  
,10

3140 DATA 158,10,158,5,166,10,166,5,158  
,10

3150 DATA 158,10,158,5,150,10,150,5,158  
,10

3160 DATA 158,10,158,5,166,10,166,5,158  
,5

3900 DATA 0,0,0,0,0,0,0,0,0

3910

3920 REM"THE SNARE DRUM

3930

4000 DATA -1,10,6,20,6,20,6,20,6,20,6,2  
0,6,20,6,20,6,20,6,20,6,20,6,20,6,2  
0,6,20,6,15,6,5,6,5,6,15

4010 DATA 5,20,5,20,5,20,5,20,5,20,5,20  
,5,10,4,5,5,5,4,5,4,15,4,20,4,20,4,20,4,  
20,4,20,4,20,4,20,4,20

20,4,20,4,20,4,20,4,20

4020 DATA 5,20,5,20,5,20,5,20,5,20,5,20,

5,20,5,20,4,20,4,20,4,20,4,20,4,20,4,20,

4,15,5,5,4,5,5,5

5000 DATA 0,0,0,0,0,0,0,0

The program will RUN, after a short pause, with a regular bass pattern and off-beat snare. The melody is divided into verse, bridge and chorus, with the counter melody changing pattern during each section. The entire piece REPEATS UNTIL the <ESCAPE> key is pressed. It works as follows.

1. Initialisation: This is done as we mentioned earlier on in the chapter but with a four-DIMensional array and four ENVELOPE statements. The array loading section of the program has to be tackled in a specific PROCedure which we look at in the following paragraph.
2. Load Arrays: First PROCinit is called from the main program and is fed a value from zero to four for channel%. Line 360 RESTOREs the program to the correct set of DATA. The pitch and duration information is then READ into the array and a zero value for pitch is tested for, as before, to indicate the end of DATA.
3. The Program: In this example the program consists of PROCinit called four times, once for each of the four sets of DATA, and PROCsync, which in turn calls PROCplay.
4. Synchronise Voices: PROCsync can logically be divided into two distinct parts. The first part uses the extended SOUND statement's s parameter to synchronise the start of the piece. If this part of the program is omitted you will hear timing discrepancies between the four parts. Line 180 zeroes the channel clocks and step numbers, both of which are held in arrays. This allows a general play procedure to be created. The second part of PROCsync controls which channel will play at any given time. You will notice that this step replaces the previously noted technique, the three lines we included for this purpose in 'Blood and Sand'. The channel clocks are then compared and the channel with the slowest clock is passed to PROCplay. Line 260 allows the loop to REPEAT... UNTIL a value of zero is detected for the expression Pitch%(1,N%(1) - 1). The piece is restarted when this is TRUE.
5. Play Channel: Line 290 increments the step number N% of the current channel by one and for the rest of the PROCedure this is stored in the variable A%. Line 310 updates the channel clock. Line 320 selects the ENVELOPE to be used (ENVELOPE I for channel zero, ENVELOPE 2 for channel 1, etc.)

or zero amplitude if a rest is detected (pitch= -1). Line 330 SOUNDS the channel.

6. The Pitch and Durations: These are BBC values which are arranged in two bar segments just as before. Each set of DATA starts at a particular line number (1000 2000 3000 4000) in order to make the RESTORE statement in Line 360 as general as possible. Note that using the RENUMBER command will require Line 360 to be rewritten.

This program can be used for any set of four part DATA, though the note arrays may have to be increased in size for particularly long pieces. (They can accept up to three hundred notes as the program stands). Three or two part tunes can be INPUT by filling the channel three and four DATA statements with rests or zeros. It should be noted that because the program only synchronises the voices using the s parameter at the very start of the program, and not during REPEATs, the parts will eventually slip out of time. This is a result of the time BASIC takes to process the various synchronisation and play commands. This phenomenon should only become noticeable after four or five REPEATS.

There is yet another available alternative to the single DATA stream method of multi-channel sync. This would involve dividing durations up into small increments which, at any given point in a tune, could either be considered notes or rests. The channel synchronisation parameter could then be used to keep each increment on each channel in step:

## Notes:

Channel 1

Channel 2

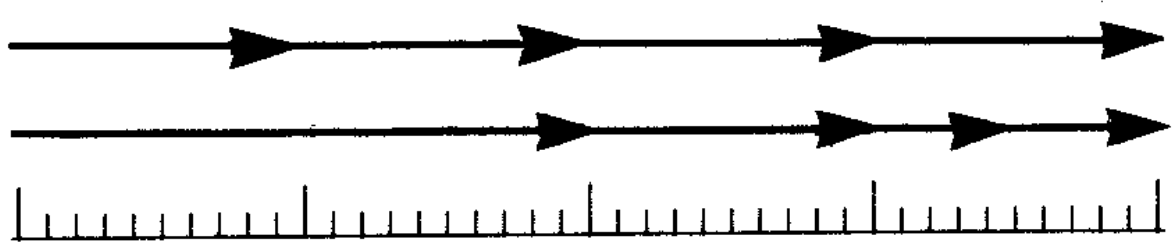


## Durations:

Channel 1

Channel 2

Duration steps



Using this method we could ensure absolute synchronisation of each channel at all times. This technique is particularly useful when dealing with real time playing. If a melody is played in from the keyboard the timing is automatically corrected to the nearest increment and stored in an array. This is equivalent to the MC-4 type of programming. One example of this type of multi-channel syncing can be seen in the drum machine program presented in Chapter Nine: Applications. One or more of the drum channels could be programmed to accept note information, as well as timing information, direct from the keyboard. I will leave it up to you to experiment with the possibilities made available by this technique.



## 7. Music Graphics

Music incorporates a diverse range of signs and symbols and, unhelpfully enough, you will see that the way in which they are used is often inconsistent with theoretically immutable rules. Using a computer to display these sharps, flats, clefs and quavers brings with it a variety of problems which we will have to learn to overcome. In this chapter we shall meet these problems head-on and develop the necessary techniques required to dismantle and solve them.

As is the case with many graphics applications there are two basic approaches to displaying music. The first is to use the PLOT, MOVE and DRAW commands to draw the symbols, etc., onto the graphics screen. This method has the advantage of flexibility in terms of object size and the ease of drawing straight lines. The disadvantage of this method is that it is slow. The second approach lies in the use of user-defined characters PRINTed on the graphics screen. This method is faster than the graphics technique and is very detailed; its main disadvantage centres on the size restrictions which are implicit in its use.

In practice I have found that it is best to use a mixture of the two techniques, with DRAW to display the stave (and possibly the clefs), and employing user-defined characters to PRINT notes, sharps and flats. Let us embark on this topic by taking a look at graphics techniques utilising MOVE and DRAW.

### GRAPHICS METHODS

Before attempting to draw notes or clefs it is necessary to display the traditional group of five lines known as the stave. To do this we need to use the two graphics commands MOVE and DRAW.

MOVE moves the graphics cursor from the initial position to co-ordinates X, Y without drawing on the screen. The command takes the form: MOVE X, Y.

DRAW draws a line from the previous graphics cursor position to a new position given by the co-ordinates A, B. The command takes the form: DRAW A,B.

---

## 102 Music Graphics

The following short routine illustrates the use of these commands by drawing a stave:

```
10 REM *** STAVE ***
20
30 MODE 1
40 FOR Lines%=0 TO 4
50 MOVE 0,896-Lines%*32
```

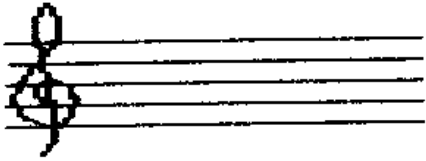
```
60 DRAW 1280,896-Lines%*32
70 NEXT
```

We will use this method of drawing a stave throughout, as it is by far the most straightforward method available. When dealing with more complicated shapes, such as the treble clef, we will have to resort to graph paper to plan our shape. If you turn to page 495 of the *BBC User Guide* you will see a graphics planning sheet. This gives you an idea of how the screen is divided up and where a MOVE or DRAW statement will place a particular line or symbol. In order to ensure that symbols fit the stave we have already drawn (128 from top to bottom) we require a larger scale grid.

First you must find some graph paper with at least 150 squares in both the X and Y directions. Then you have to draw two lines across the page, 128 divisions apart. These represent the top and bottom lines of the stave. A treble clef can now be drawn, (preferably in pencil!) and the necessary DRAW information can be worked out, once you've got a recognisable clef, by redrawing the result and shape using straight lines, and recording the grid positions. The shorter the lines used the more accurate the final result of course, and it doesn't really matter where you start. When dumped to a

image, the design looks like this:

printer, my design looks like this:



Rather than use screenfuls of separate DRAW statements it's better to contain the relative co-ordinate information in a DATA statement. We can then reconstruct a treble clef using the following program:

```

10
20 REM"      DRAW A TREBLE CLEF
30
40 MODE5
50 MOVE 500,500
60 FOR Z%=1 TO 39
70   READX%,Y%

```

```

      80   DRAW 500+X%,500+Y%
      90   NEXT
     100

```

```

110 DATA -10,5,-20,10,-20,30,-10,35,0,
40,10,38,20,32,30,24,28,10,40,0,38,10,30
,-25,20,-32,10,-38,0,-40

```

```

120 DATA -10,-38,-20,-34,-30,-30,-38,-
20,-42,-10,-45,0,-45,10,-42,20,-40,30,10
,100,15,120,17,130,15,140

```

```

130 DATA 10,155,0,170,-10,160,-14,140,
-14,110,-12,80,5,-20,10,-40,8,-60,0,-80,
-10,-90

```

```

140

```

The stave and clef programs could then be arranged in procedures, to be called up whenever needed.

```

10
20 REM"      STAVE/CLEF
30
40 MODE5
50
60 PROCstave
70 PROCclef

```

```
80 END
90
100 DEF PROCclef
110 RESTORE 290
120 MOVE 100,500
130 FOR Z%=1 TO 39
140   READX%,Y%
150   DRAW 100+X%,500+Y%
160 NEXT
170 ENDFPROC
180
190
200
210 DEF PROCstave
220 FOR Lines%=0 TO 4
230 MOVE 0,596-Lines%*32
240 DRAW 1280,596-Lines%*32
250 NEXT
260 ENDFPROC
270
```

## 104 Music Graphics

280

```
290 DATA -10,5,-20,10,-20,30,-10,35,0,  
40,10,38,20,32,30,24,28,10,40,0,38,10,30  
, -25,20,-32,10,-38,0,-40
```

```
300 DATA -10,-38,-20,-34,-30,-30,-38,-  
20,-42,-10,-45,0,-45,10,-42,20,-40,30,10  
, 100,15,120,17,130,15,140
```

```
310 DATA 10,155,0,170,-10,160,-14,140,  
-14,110,-12,80,5,-20,10,-40,8,-60,0,-80,  
-10,-90
```

The introduction of a size factor provides a range of clef sizes.

10

```
20 REM"      DRAW DIFFERENT SIZES
```

```
30 MODE 7
```

40

```
50 INPUTTAB(10,10)"SIZE FACTOR:",F
```

```
60 MODE5
```

```
70 MOVE 500,500
80 FOR Z%=1 TO 39
90 READ X,Y
100 DRAW 500+X*F,500+Y*F
110 NEXT
120
130 DATA -10,5,-20,10,-20,30,-10,35,0,
40,10,38,20,32,30,24,28,10,40,0,38,10,30
,-25,20,-32,10,-38,0,-40
140 DATA -10,-38,-20,-34,-30,-30,-38,-
20,-42,-10,-45,0,-45,10,-42,20,-40,30,10
,100,15,120,17,130,15,140
150 DATA 10,155,0,170,-10,160,-14,140,
-14,110,-12,80,5,-20,10,-40,8,-60,0,-80,
-10,-90
```

If we apply this technique to staves the result will be the creation of an alignment problem between the stave and clef. If different sizes of clef and stave are required, the stave can be drawn with five separate MOVE/DRAW commands. This will make the introduction of a scale factor straight-

forward. This DRAWing technique can also be applied to the bass clef, or any other music symbols. The alternative technique, however, has advantages for our purposes.

## USER-DEFINED CHARACTERS METHOD

By using the VDU 5 command to PRINT text at the graphics cursor we can place any text character on the graphics screen, simply by using MOVE followed by PRINT. To PRINT the letter 'E' in the middle of the screen we would use:

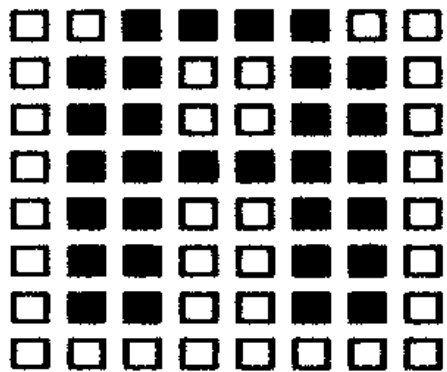
```
MODE 4:MOVE640,512:PRINT"E"
```

Combine this with the fact that any of the character shapes which have ASCII codes from 224 to 255 can be redefined by the user and we have a very powerful graphics tool! The first step which needs to be taken toward using this tool is redefining a character shape.

Each character the computer PRINTs on the screen is drawn within an 8 by 8 grid. For example the letter "A" is defined as follows:



By using a 7 column by 8 row array of squares, we can define an 8x7 array.



The computer stores this information in binary form as follows:

BINARY	DECIMAL
00111100	60
01100110	102
01100110	102
01111110	126
01100110	102
01100110	102
01100110	102
00000000	0

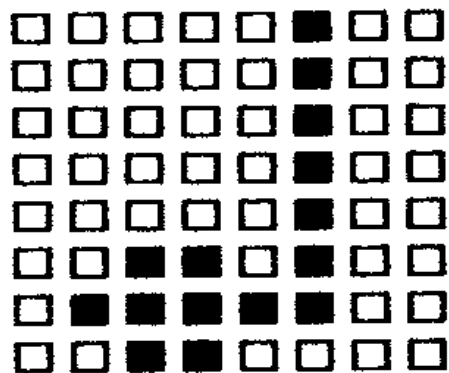
We can change this information using the VDU 23 statement. In this case if

we wished, for some reason, to change the character with the ASCII code 224 to “A” we would enter the following:

VDU 23, 224, 60, 102, 102, 126, 102, 102, 102, 0

If PRINT CHR\$(224) <RETURN> were now typed the letter “A” would be PRINTed on the screen. To take a more useful example, we could define the ASCII 225 character as a crotchet. First we need to draw the shape in the 8 by 8 grid; we can then transfer this to a binary pattern:

## 106 Music Graphics



BINARY

DECIMAL

00000100

4

00000100

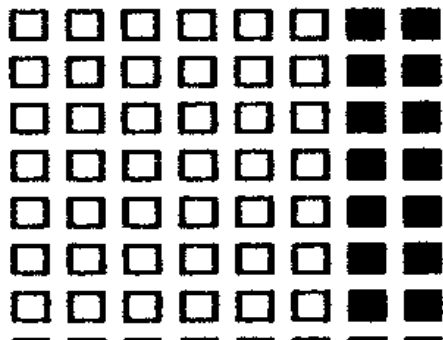
4

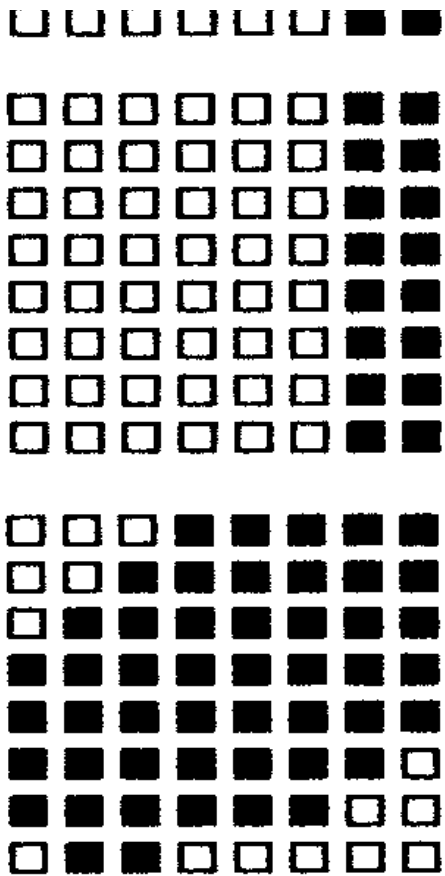
00000100	4
00000100	4
00000100	4
00110100	52
01111100	124
00110000	48

Now we have to work out the various decimal equivalents, as above, and finally enter the VDU command as follows:

VDU 23, 225, 4, 4, 4, 4, 4, 52, 124, 48

Note that we can use groups of these redefined characters to create larger symbols. A more easily read crotchet could be drawn using a mere three characters:





The VDU statements would be as follows:

```
VDU 23, 230, 31, 63, 127, 255, 255, 254, 254, 252, 96
VDU 23, 224, 3, 3, 3, 3, 3, 3, 3, 3
```

The completed crotchets could then be printed using the following routine:

The completed crotchet could then be printed using the following routine.

```

10  REM    PRINT CROTCHET
20
30  MODE1
40  PRINTTAB(15,15);CHR$224
50  PRINTTAB(15,16);CHR$224
60  PRINTTAB(15,17);CHR$230
70  END

```

VDU 5 is used to turn PRINTing at the graphics cursor on. VDU 4 is used to turn it off. All this working out of VDU statements is rather tedious so let us look at the following program which lets us design our new character directly on the screen, while the VDU 23 statement itself is worked out by the computer.

```

10  *FX12,0
20
30
40  REM"  ***USER DEFINE KEY PROG***
50

```

```

60  DIM HEX%(8,8)
70  DIM A(8)
80  MODE1
90  *FX4,1
100 X=12:Y=12
110 VDU23;8202;0;0;0;
120 COLOUR2
130 PRINTTAB(5,3)"PRESS R TO RESET"
140 PRINTTAB(5,29)"CURSOR KEYS TO MOV
E"
150 PRINTTAB(5,7)"PRESS Q FOR VDU 23"
160 COLOUR3
170 PRINTTAB(5,5)"PRESS E TO END"
180 PRINTTAB(5,25)"1....SQUARE ON"
190 PRINTTAB(5,27)"2....SQUARE OFF"
200 COLOUR 1
210 VDU23,241,0,0,0,0,0,0,0,0

```

```

220  VDU23,270,230,230,230,230,230,230
,255,255

```

```

230  FOR A%=11 TO 20:FOR B%=11 TO 20 S
TEP 9:PRINTTAB(A%,B%);CHR$(240):NEXT:NEX
T

```

```

240  FOR A%=11 TO 20:FOR B%=11 TO 20 S
TEP 9:PRINTTAB(B%,A%);CHR$(240):NEXT:NEX
T

```

```

250  COLOUR2

```

```

260  REPEAT

```

```

270  K=GET

```

```

280  IF K=136 X=X-1:IF X<12 X=12

```

```

290  IF K=137 X=X+1:IF X>19 X=19

```

```

300  IF K=138 Y=Y+1:IF Y>19 Y=19

```

```

310  IF K=139 Y=Y-1:IF Y<12 Y=12

```

```

320  IF K=49 PRINTTAB(X,Y);CHR$(240):H
EX%(X-11,Y-11)=1

```

```

330  IF K=50 PRINTTAB(X,Y);CHR$(241):H
EX%(X-11,Y-11)=0

```

```

340  IF K=81 PROChex

```

```

350  IF K=69 GOTO 380

```

```

360  IF K=88 GOTO 380

```

```
360 IF K=82 RUN
370 UNTIL FALSE
380 *FX4,0
390 MODE7
400 PRINTTAB(10,10)"THAT'S ALL"
410 END
420 DEF PROChex
430 FOR Y=1 TO 8
440 FOR X=1 TO 8
450 IF HEX%(X,Y)=1 A(Y)=2^(8-X)+A(Y)
460 NEXT
470 NEXT
480 PRINTTAB(0,23);"VDU 23,#,";A(1);"
,";A(2);",";A(3);",";A(4);",";A(5);",";A
(6);",";A(7);",";A(8)
490 X=12:Y=12
500 FOR T=1 TO 8:A(T)=0:NEXT
510 ENDPROC
```

This program is very straightforward to use. When RUN, a red square which defines the 8 by 8 character is drawn. Squares within the character are filled in with yellow by pressing '1' Pressing '2' blanks a square. The



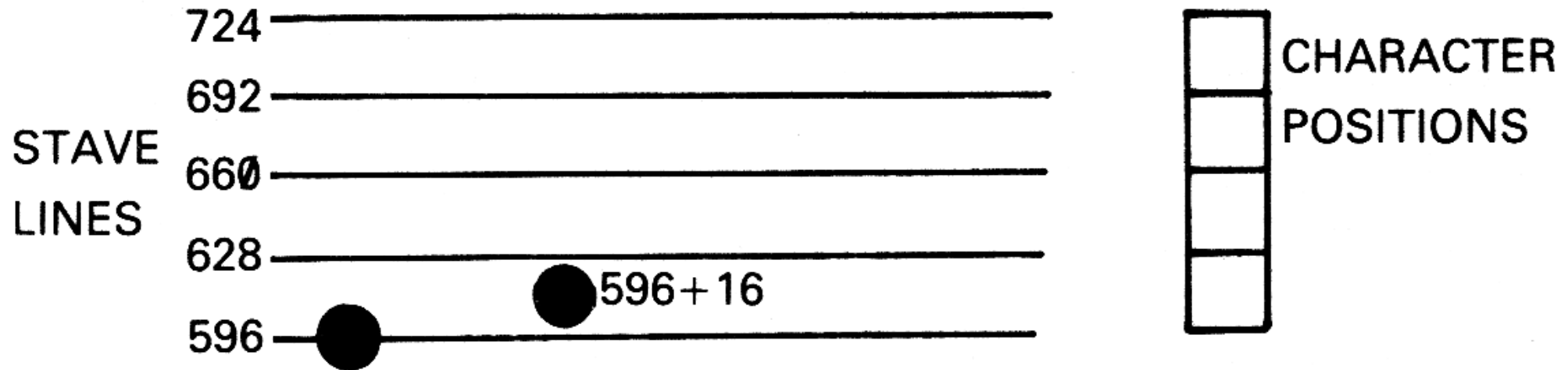
are filled in with yellow by pressing 1. Pressing 2 draws a square. The cursor keys are used to change position within the 8 by 8 grid. When the

design is completed pressing 'Q' causes the VDU 23 statement to be PRINTed. At this stage you may continue adding to the design, reset to a blank square (press 'R') or end the program (press 'E'). It is important to exit the program using 'E' rather than so that cursor editing can promptly be, restored using \*FX 4,0. VDU 23 values and PRINTs them on the screen.

The program itself uses two VDU 23 statements. One is necessary to produce a blank square (all zeros) and one produces a completely filled square (all 255). The call \*FX 4,1 in Line 80 turns off cursor editing and allows the cursor keys to be used to control horizontal and vertical movement within the square. Lines 220 to 230 draw the red square. Lines 270 to 300 control position within the square and create a boundary. Lines 310 and 320 cause either a filled or blank square to be PRINTed at position X,Y. This fact is then recorded as a 1 or zero in the array HEX%(I,S). Finally, PROChex works out the VDU 23 values and PRINTs them on the screen.

Once a character has been created the next problem which presents itself is how to control its position on the screen. One useful aid to this end is the alternative graphics planning sheet explained on page 494 of the User Guide. This grid relates character positions to the graphics dimensions. A sensible arrangement for positioning the lines of the stave would be on five of the horizontal planning sheet lines. This would correspond to one text character in height, or 32 graphics divisions apart, as we indeed did to draw staves earlier.

Once text characters for notes have been created their position on the stave can be controlled by adding 16 for the next note position



The following example uses the above techniques to draw a piano-type double stave on the screen. Notes can then be selected using the notes' alphabetical name. (We could have equally well used a keyboard arrange-ment for this purpose.) The octave of the note is controlled by pressing 'O'. The sequence of notes can then be played back by pressing 'P'.

```
10
20
30 REM"****WRITE****
40
```

50

60 VDU23,224,3,3,3,3,3,3,3

---

## 110 Music Graphics

```
70VDU23,225,1,1,1,1,1,1,3,3
80VDU23,226,5,9,17,17,33,33,65,65
90VDU23,227,67,67,65,33,17,9,7,1
100VDU23,228,128,64,32,32,32,32,32,32
110VDU23,229,64,64,128,128,0,0,0,0
120VDU23,230,31,63,127,255,255,254,252
,96
130VDU23,232,0,0,56,68,130,1,1,1
140VDU23,233,129,129,3,2,12,24,224,0
150VDU23,234,0,7,24,32,32,64,64,112
160VDU23,235,112,112,0,0,0,0,0,0
170VDU23,236,0,0,0,0,1,2,12,112
180VDU23,237,0,128,96,22,22,16,8,8
190VDU23,253,8,8,16,22,22,16,32,32
200VDU23,254,64,64,128,128,0,0,0,0
```

```
210  
220  
230  
240 REM"***SET UP PROGRAM***  
250  
260  
270 MODE1  
280 VDU23;8202;0;0;0;  
290 VDU28,1,31,39,20  
300 ENVELOPE1,1,0,0,0,0,0,0,89,-1,0,-1  
,45,23  
310 DIM F%(7)  
320 DIM Fote%(100)  
330 X%=100  
340 O%=2  
350 T%=1  
360 Tempo%=10  
370 Note$="FGABCDE"  
380 FOR D%=1 TO 7:READ F%(D%):NEXT
```

```
390
400
410
420 PROCnewline:REM"DRAW CLEF/STAVE
430
440
450
460 REPEAT
470 REPEAT:REM"THE PROGRAM LOOP
480 A$=GET$
```

---

Music Graphics 111

```
490 IF A$="O" O%=O%+1:COLOUR1:PRINTTAB(8,1);O%:IF O%=4 O%=0:PRINTTAB(8,1);O%
500 IF A$="T" Tempo%=Tempo%+1:COLOUR1:PRINTTAB(8,3);Tempo%; " ":IF Tempo%=16
Tempo%=1:PRINTTAB(8,3);Tempo%; " "
510 IF A$="F" AND T%>1 PROCplay
520 Note%=INSTR(Note$,A$)
530 UNTIL Note%>0
```

```
540 PROCNote
550 X%=X%+100
560 IF X%>1200 PROCnewline:X%=100
570 UNTIL FALSE
580
590
600
610
620DEF PROClines:REM"DRAW LINES
630 VDU5
640 GCOL0,3
650 FOR I%=0 TO 192 STEP192
660 FOR L%=0 TO 4
670 MOVE0,896-L%*32-I%:DRAW1300,896-L%
*32-I%
680 NEXT
690 NEXT
700 VDU4
710ENDPROC
720
730
```

```
740
750DEF PROCclef:REM"DRAW CLEFS
760 VDU5
770 GCOL0,2
780MOVE0,896:PRINT CHR$224;CHR$228
790MOVE0,864:PRINT CHR$225;CHR$229
800MOVE0,832:PRINT CHR$226;CHR$232
810MOVE0,800:PRINT CHR$227;CHR$233
820MOVE0,768:PRINT CHR$234;CHR$237
830MOVE0,736:PRINT CHR$235;CHR$253
840MOVE0,704:PRINT CHR$236;CHR$254
850MOVE4,768:PRINT CHR$224
860 VDU4
870ENDPROC
880DEF PROCnewline
```

---

112 Music Graphics

```
890CLG:PROClines:PROCclef:PROCoct
900 ENDPROC
```

```
910
920
930
940 DEF PROCNote:REM"DRAWNOTESPLAY
950 PROCrite
960 Pote%(T%)=P%(Note%)+0%*48
970 SOUND1,1,Pote%(T%),Tempo%
980 T%=T%+1
990 VDU4
1000 ENDPROC
1010
1020
1030
1040 DEF PROCOct:REM"RESET DISPLAY
1050 COLOUR1
1060 PRINTTAB(0,1)"Octave: ";0%
1070 PRINTTAB(0,3)"Tempo : ";Tempo%
1080 PRINTTAB(0,5)"Press 'P' to Play"
1090 ENDPROC
1100
```

```
1100
1110
1120 DEF PROCplay
1130 FOR E%=1 TO T%-1
1140 SOUND1,1,Fote%(E%),Tempo%
1150 NEXT
1160 ENDPROC
1170 DATA 25,33,41,49,53,61,69
1180
1190
1200
1210 DEF PROCrite:REM"DRAWNOTES
1220 VDU5
1230 GCOL0,2
1240 MOVE X%,Note%*16+560+0%*112:PRINT
CHR$(230)
1250 MOVE X%,Note%*16+592+0%*112:PRINT
CHR$(224)
1260 MOVE X%,Note%*16+624+0%*112:PRINT
CHR$(224)
```



```

1270 VDU4
1280 ENDPROC

```

Lines 60 to 200 are devoted to defining the user-programmable characters. The two clefs and the crotchet shape are constructed in this way. In the set-up part of the program the VDU 28 command sets up a text window to contain the octave and tempo information. The array P% contains Pitch DATA which is READ into the array at Line 380.

The main part of the program consists of a REPEAT... UNTIL loop containing a GET\$ statement. The GET\$ line waits for various INPUTS on which to act. “O” causes the O% variable to increase and so changes the octave of the INPUT Pitch. “T” acts in the same way on tempo. The smaller the Tempo% variable the faster the tempo. “P” causes the program to go to PROCplay and play the notes INPUT so far. If the INPUT does not consist of “O”, “T” or “P” the program compares the contents of A\$ with the components of Note\$, the alphabetical note names. If A\$ is contained in Note\$, a value of Note% >0 will be returned and the program will exit the inside loop and go to PROCnote. This draws the note as a crotchet on the screen and records the pitch value in Pote%(100). The x co-ordinate screen position is increased by 100 at Line 550 and end of screen is tested for in Line 560.

The PROClines routine draws the stave lines just as before, but incorporates a modification which makes it possible to draw a double stave. The lines then appear on the screen at Y co-ordinates 896, 864, 832, 800, 768 and 704, 672, 640, 608, 576. VDU 5 turns on PRINTing at the graphics cursor and VDU 4 turns it off. These commands have been included at the beginning and end of each PROCEDURE to facilitate the use of the entire PROC in your own programs. GCOL 0,3 selects white as the colour of the lines.

PROCclef uses the user-defined character method of PRINTing the treble and bass clef symbols. The top left-hand point of the character is positioned at the cursor. The character therefore appears below the line it is MOVED to. GCOL 0,2 selects yellow.

PROCnewline is called up whenever the current stave is full. It simply erases the current screen and displays an empty stave in its place.

PROCNote converts the P% pitch and O% octave information into a note value Pote% which is stored in a 100 note array. T% is the number of pitches currently in the array.

PROCoct sets up the text display in the text window. COLOUR 1 selects red.

PROCplay plays all the notes currently stored in array Pote% at a tempo determined by Tempo%.

When PROCrite is called, this procedure draws a crotchet in yellow on the correct line or space of the stave. The lowest note possible in this program is F, on the space below the bass clef. This would require the top left-hand corner of the crotchet ball to be placed on the bottom line, 576. The Note% value for F is 1. The octave O% value would be zero, so the Y co-ordinate value can be calculated as:

## 114 Music Graphics

$$Y=(1*16)+560+(0*112)=576$$

The calculation for A on the treble clef would be as follows:

$$Y=(3*16)+560+(2*112)=832$$

The stem of the note is simply drawn in two parts, 32 and 64 divisions above the ball.

‘Write’ is not by any stretch of the imagination meant to be the definitive music-writing program. Too many possible facilities have been left out for this to be the case. Instead, it is intended to provide a starting point from which and on the basis of which you can learn to develop your own ideal program in this area. I say ideal because no one’s requirements of a music writing program tend to be identical to anyone else’s. Some people might prefer a piano keyboard method (as in the sequencer program in Chapter 9) of INPUTting notes, perhaps even using an external keyboard rather than the computer’s. Others might only require one stave but prefer having more than one line on the screen at one time. Specific application and, of course, the individuals demands and store of musical knowledge can affect the requirements he or she makes of the program.

These considerations apart, it has to be emphasised that if we are looking for an all purpose music writing program there are certain other additional facilities which we need to incorporate. These are as follows:

1. The ability to write sharps or flats. This could be added as a PROCEDURE called up from the main program.  $Pote\%(T\%-1)$  would be increased by 4 for a sharp or decreased by the same amount for a flat. User-defined accidentals would then be PRINTED on the screen.
2. A delete note PROCEDURE could be accomplished by overPRINTING a character with black as the foreground colour.
3. Different durations could be selected using a duration variable similar to the tempo and octave variables. I.e.  $D\%$  equal to 1 would select a user defined minim, 2 a crotchet, 4 a quaver etc. An array which could be used to recall duration values would also be required.
4. Selection of playback starting points could be controlled using  $T\%$ .

Less fundamental additions could include: editing commands that would allow copying, transposition, insertion and bulk deletions. Key selection, polyphonic entry either as chords or on a multiple stave system and note display on playback, perhaps with a colour change on the note currently playing. These features can all be found in the Music Maker program from

Island Logic for which I have been writing tunes.

My own solution to the expansion of ‘Write’ into a program of greater flexibility is given below. Digest, absorb and work with it, and then transpose it into your own ideal.

ADVANCED MUSIC WRITER

The Advanced Music Writer lets the user write melodies on the screen using standard musical nomenclature. Letter keys select the notes displayed and if an error is made it is a simple matter to delete the mistake and try again. The filing system allows you to save your finished works on tape or disc and makes it possible to build a library of your favourite tunes or compositions.

On RUNning the program the screen display is as follows:

```
Octave: 0 : 'O'      Press 'S' for Sharp
Tempo  : 4 : 'T'      Press 'D' to Delete
Duration: 4  'L'      Press 'P' to Play
'Z'..Transpose      Press  to Stop
To Reset            'R'...Read File
Press <BREAK>       'W' .. Write File

      Current File:  IRISH
```

Most of the extra facilities can be seen, displayed beneath the stave. Octave and Tempo values are changed in an identical fashion to the “Write” program. If ‘0’ is pressed the Octave value changes, in the range 0 to 4, with zero being the lowest octave and 4 the highest.

Tempo has a larger range of possible values with 1 being the fastest tempo and larger values progressively slower. Duration is a new addition. This value allows you to change the length of note displayed on the stave.

116 Music Graphics

The following table shows the note lengths available:

Duration	Note length
1	Quaver

2	Crotchet
3	Dotted crotchet
4	Minim

This facility greatly adds to the musicality of the program. Accidentals are also allowed in the program. The current entry can be sharpened after entry simply by pressing 'S'.

Using all these facilities, if we wished to write the note of G  $\sharp$  as a dotted crotchet on the bottom stave we would follow this procedure:

1. Change to lowest octave by pressing 'O' to select a value of zero.
2. Press 'L' to select 3, a dotted crotchet, for duration.
3. Press G. At this stage a dotted crotchet would be drawn on the bottom line of the lower stave and G would be SOUNDED.
4. Press 'S' to select 'Sharpen the last entry', i.e. G to G  $\sharp$ . A sharp sign will be added on the screen in front of G and G  $\sharp$  would be SOUNDED.

This may seem a somewhat long winded process but in practise it will not be necessary to change all the parameters to select a new note.

If a mistake is made, pressing the <DELETE> key will erase the last entry. Repeated pressing of delete will erase the entire tune. To play your work at any stage you simply press 'P'. To repeat the tune a number of times simply press 'P' the required number of repeats. Pressing the

times simply press 1 the required number of repeats. Pressing the <ESCAPE> key will stop the repeat sequence.

The <BREAK> key is programmed to erase the current sequence and reset the octave, tempo, etc., parameters to their default values. To escape entirely from the program it is therefore necessary to press <CTRL> <BREAK> or switch off the machine. The 'Z' key allows transposition of the current tune. You are requested to INPUT the transposition (up or down) as a number of semitones, so that one tone down transposition would be entered as -2, a fourth up would be 5, and so on.

The final facility allows loading and saving of tune files. To save a file press 'W' for Write. You are then prompted to enter a name for your piece and pressing <RETURN> commits the file to tape or disc. Pressing 'R' for Read also prompts for a file name. On pressing <RETURN> the file is loaded. If it is not found you are returned to the Menu.

The display of notes on the stave only operates when you are actually writing a particular line of music. Once a fresh stave is drawn (when the end of an on screen line is reached) it is not possible to redraw, so checks should be made as to the accuracy of your writing before 'turning a page.'

```

20
30 REM"##ADVANCED MUSIC WRITER##
40
50 REM"Copyright.....Ian Ritchie
60
70 VDU23,224,3,3,3,3,3,3,3,3
80 VDU23,225,1,1,1,1,1,1,3,3
90 VDU23,226,5,9,17,17,33,33,65,65
100 VDU23,227,67,67,65,33,17,9,7,1
110 VDU23,228,128,64,32,32,32,32,32,32
120 VDU23,229,64,64,128,128,0,0,0,0
130 VDU23,230,31,63,127,255,255,254,25
2,96
140 VDU23,232,0,0,56,68,130,1,1,1
150 VDU23,233,129,129,3,2,12,24,224,0
160 VDU23,234,0,7,24,32,32,64,64,112
170 VDU23,235,112,112,0,0,0,0,0,0
180 VDU23,236,0,0,0,0,1,2,12,112
190 VDU23,237,0,128,96,22,22,16,8,8

```

```
200 VDU23,253,8,8,16,22,22,16,32,32
210 VDU23,254,64,64,128,128,0,0,0,0
220 VDU23,238,37,38,36,44,52,36,100,16
4
230 VDU23,239,255,255,255,255,255,255,
255,255
240 VDU23,240,7,9,49,65,129,130,132,12
0
250 VDU23,241,248,206,199,99,49,25,12,
6
260 VDU23,242,0,0,0,24,24,0,0,0
270
280
290
300 REM"***SET UP PROGRAM***
310
320
330 ON ERROR PROCoct:GOTO 600
340 *KEY10 OLD:M RUN:M
350 MODE1
```



```
360   VDU23;8202;0;0;0;
370   VDU28,1,31,39,16
380   ENVELOPE1,1,0,0,0,0,0,0,89,-1,0,-
1,126,100
```

---

## 118 Music Graphics

```
390   DIM P%(7)
400   DIM PITCH%(150)
410   DIM DUR%(150)
420   X%=150
430   O%=2
440   T%=1
450   S%=0
460   L%=2
470   FILE$=""
480   Note%=0
490   Tempo%=4
500   Note$="FGABCDE"
510   FOR D%=1 TO 7:READ P%(D%):NEXT
```

```
520
530 DATA 25,33,41,49,53,61,69
540
550
560 PROCnewline:REM"DRAW CLEF/STAVE
570
580
590
600 REPEAT
610 REPEAT:REM"THE PROGRAM LOOP
620 A$=GET$
630 IF A$="O" O%=O%+1:COLOUR1:PRINT
TAB(8,1);O%:IF O%>3 O%=0:PRINTTAB(8,1);O
%
640 IF A$="T" Tempo%=Tempo%+1:COLOUR1:PRINTTAB(8,3);Tempo%:" ":IF Tempo%>8
Tempo%=1:PRINTTAB(8,3);Tempo%:" "
650 IF A$="L" L%=L%+1:COLOUR1:PRINTTAB(10,5);L%:" ":IF L%>4 L%=1:PRINTTAB(10,5);L%:" "
```

```
660      IF A$="P" AND T%>1 PROCplay
670      IF A$="S" PROCsharp
680      IF A$="W" PROCsave
690      IF A$="R" PROCload
700      IF A$="Z" PROCTRANSPOSE
710      IF A$=CHR$(127) PROCdel
720      Note%=INSTR(Note$,A$)
730      UNTIL Note%>0
740      PROCNote
750      X%=X%+100
```

---

Music Graphics 119

```
760      IF X%>1200 CLG:PROClines:PROCoct
: X%=50
770      UNTIL FALSE
780
790
800
810
```

```
810
820 DEF PROClines:REM"DRAW LINES
830   VDU5
840   GCOL0,3
850   FOR I%=0 TO 192 STEP192
860   FOR LEG%=0 TO 4
870     MOVE0,896-LEG%*32-I%:DRAW1300,896
-LEG%*32-I%
880   NEXT
890   NEXT
900   VDU4
910 ENDPROC
920
930
940
950 DEF PROCclef:REM"DRAW CLEFS
960   VDU5
970   GCOL0,2
980   MOVE0,896:PRINT  CHR$224;CHR$228
990   MOVE0,864:PRINT  CHR$225;CHR$229
```

```
1000 MOVE0,832:PRINT CHR$226;CHR$232
1010 MOVE0,800:PRINT CHR$227;CHR$233
1020 MOVE0,704:PRINT CHR$234;CHR$237
1030 MOVE0,672:PRINT CHR$235;CHR$253
1040 MOVE0,640:PRINT CHR$236;CHR$254
1050 MOVE4,768:PRINT CHR$224
1060   VDU4
1070 ENDPROC
1080 DEF PROCnewline
1090 CLG:PROClines:PROCclef:PROCoct
1100   ENDPROC
1110
1120
1130
1140   DEF PROCNote:REM"DRAWNOTESPLAY
1150   IF L%=4 SYM%=240 ELSE SYM%=230
1160   IF L%=1 SYS%=241 ELSE SYS%=224
1170   PROCrite
```

```
1180   DUR%(T%)=L%
1190   PITCH%(T%)=P%(Note%)+O%*48+S%
1200   SOUND1,1,PITCH%(T%),Tempo%
1210   T%=T%+1
1220   VDU4
1230   ENDPROC
1240
1250
1260
1270   DEF PROC Oct:REM"RESET DISPLAY
1280   CLS
1290   COLOUR1
1300   PRINTTAB(0,1)"Octave: ";O%;" : 'O'
";TAB(18,1);"Press 'S' for Sharp"
1310   PRINTTAB(0,3)"Tempo : ";Tempo%;"
: 'T'";TAB(18,3);"Press <DEL> to Delete"
1320   PRINTTAB(0,5)"Duration: ";L%;" : '
L'";COLOUR3:PRINTTAB(18,5)"Press 'P' to
Play"
```

```
1330  PRINTTAB(0,7)" 'Z'..Transpose";TAB
(18,7)"Press <ESC> to Stop"
1340  COLOUR2
1350  PRINTTAB(0,11)"Press <BREAK>";TAB
(18,11);" 'W'..Write File"
1360  PRINTTAB(0,9)"To Reset";TAB(18,9)
;" 'R'...Read File"
1370  IF FILE$<>"" : COLOUR2:PRINTTAB(4,1
3);"Current File: ":COLOUR3:PRINTTAB(19,
13);FILE$
1380  ENDPROC
1390
1400
1410  DEF PROCplay:REM"PLAY TUNE
1420  E%=1
1430  REPEAT
1440  FOR Channel%=1 TO 3:SOUND&200+Chan
nel%,1,PITCH%(E%),Tempo%*DUR%(E%):NEXT
1450  E%=E%+1
1460  UNTIL E%=T%
```

```
1470  ENDFPROC
1480
1490
1500
```

---

Music Graphics 121

```
1510  DEF PROCrite:REM"DRAWNOTES
1520  VDU5
1530  GCOL0,2
1540  MOVE X%,Note%*16+560+0%*112:PRINT
CHR$(SYM%)
1550  MOVE X%,Note%*16+592+0%*112:PRINT
CHR$(224)
1560  MOVE X%,Note%*16+624+0%*112:PRINT
CHR$(SYS%)
1570  IF L%=3 MOVE X%+30,Note%*16+560+0
%*112:PRINT CHR$242
1580  VDU4
1590  ENDFPROC
1600
```



```
1600
1610
1620 DEF PROCsharp:REM"SHARPENNOTES
1630 S%=4
1640 VDU5
1650 T%=T%-1
1660 X%=X%-100
1670 MOVE X%-40,Note%*16+570+0%*112:PR
INT CHR$(238)
1680 MOVE X%-40,Note%*16+542+0%*112:PR
INT CHR$(238)
1690 PROCNote
1700 VDU4
1710 S%=0
1720 X%=X%+100
1730 ENDPROC
1740
1750
1760 DEF PROCdel:REM"DELETE NOTES
1770 VDU5
1780 T%=T%-1
```

```
1790   X%=X%-100
1800   GCOL0,0
1810   FORgit%=0TO16: MOVE X%-40,git%*32+
528:PRINT CHR$(239);CHR$(239);CHR$(239):
NEXT
1820   PROClines
1830   VDU4
1840   ENDPROC
1850
```

---

## 122 Music Graphics

```
1860
1870   DEF PROCsave:REM"SAVE FILE
1880   CLS
1890   INPUT"FILE NAME: ",A$
1900   FILE$=A$
1910   V=OPENOUT A$
1920   V%=1
1930   REPEAT
```

```
1940 PRINT#V,PITCH%(V%),DUR%(V%),1%
1950 V%=V%+1
1960 UNTIL PITCH%(V%)=0
1970 CLOSE#V
1980 CLS:PROCoct
1990 ENDPROC

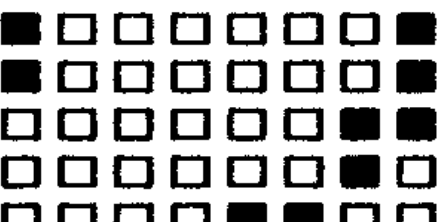
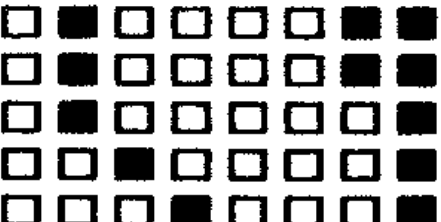
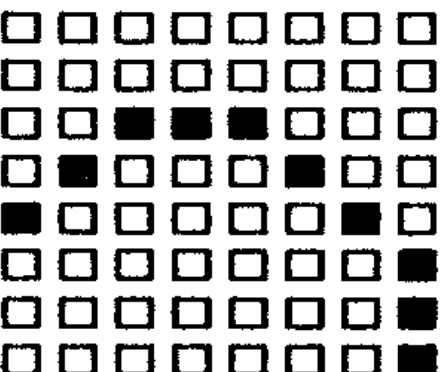
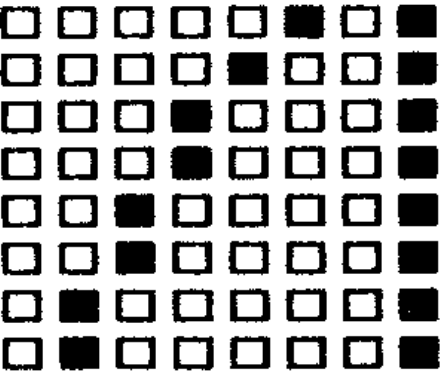
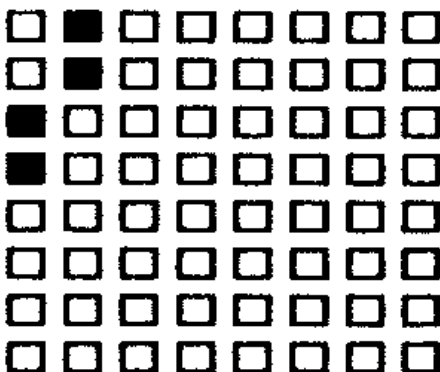
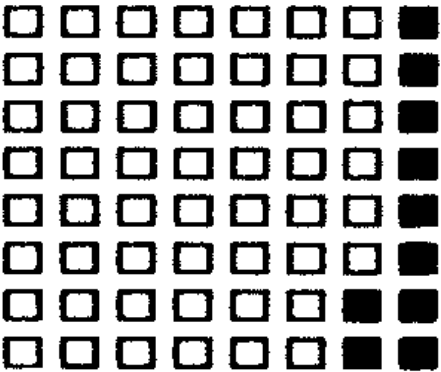
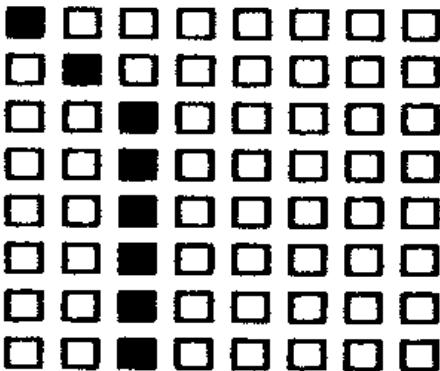
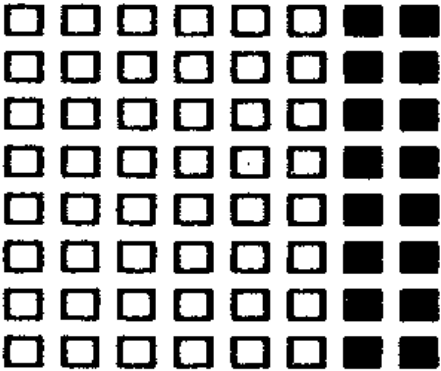
2000
2010
2020 DEF PROCload:REM"LOAD FILE
2030 CLS
2040 INPUT"FILE NAME: ",A$
2050 FILE$=A$
2060 Y=OPENIN A$
2070 V%=1
2080 REPEAT
2090 INPUT#Y,PITCH%(V%),DUR%(V%),T%
2100 V%=V%+1
2110 UNTIL EOF#Y
2120 CLOSE#Y
2130 CLS:PROCload
```

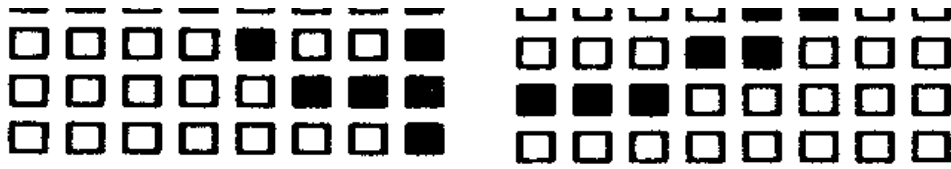
```

2130   CLS:PROC OCT
2140   ENDPROC
2150
2160   DEF PROCTRANSPOSE
2170   CLS
2180   INPUT "HOW MANY SEMITONES DO YOU W
ISH" ' ' "TO GO UP OR DOWN(-): ",M%
2190   Z%=1
2200   REPEAT
2210   PITCH%(Z%)=PITCH%(Z%)+M%*4
2220   Z%=Z%+1
2230   UNTIL PITCH%(Z%)=0
2240   PROC Oct
2250   ENDPROC

```

Lines 70 to 260 are devoted to defining the characters in the program. The following diagram illustrates the way part of the treble clef is built up:





User-programmed characters are used for the various note shapes, clefs and sharps. Line 330 ensures that if an error occurs the program carries on RUNNING. This means that we can use the <ESCAPE> key as a means of stopping the program playing. Also, if an incorrect or unfound file name is used when loading or saving files the program does not crash. Line 340 allows the <BREAK> key to be used to reset the program.

The VDU 23 statement switches off the cursor while the VDU 28 at line 370 defines a text window at the bottom of the screen. The remainder of the set up program is concerned with initialising variables and setting up arrays. The first routine called is PROCnewline, which in turn calls PROClines and PROCclef, but the main program is contained within two loops from Lines 600 to 770.

**The Main Program:** If the INPUT at Line 620 is a musical note, i.e. contained in Note\$ (which is tested for using INSTR(Note\$,A\$)) a note is SOUNDED and drawn on the screen, otherwise the program stays within the inside REPEAT . . . UNTIL loop. If 'O' is typed the octave number O% is

## 124 Music Graphics

incremented by one. The same applies to T for tempo and L for duration (since D is a musical note). Each of these variables has a maximum value above which the variable is reset to zero. Other letters call various PROCedures, some of which you will have seen before, such as PROCplay, and others that are entirely new such as PROCTRANSPOSE and PROCload.

**PROClines:** This is identical to PROClines in the earlier “Write” program. It simply DRAWS the two sets of five lines for the treble and bass staves.

**PROCclef:** This PROCedure is also identical to its earlier counterpart in “Write” and PRINTs the user-defined characters making up the treble and bass clefs on the screen at the beginning of each stave.

**PROCnote:** This differs from its predecessor in that it has a number of additions to take account of sharps and different note lengths. The variable L% is first tested in lines 1150 and 1160 to determine which symbol should be PRINTed. At Line 1180 it is then recorded in the duration array DUR%(T%). PROCrite is called to draw the symbol on the stave and then the note is SOUNDED at Line 1200. The variable S% has the value 4 when the note is to be sharpened and zero otherwise. The variable T% keeps track of the number of steps in the sequence.

**PROCoct:** PROCoct is called whenever the text display has to be reset and displays the current parameter values and file name in various colours.

**PROCplay:** This PROCEDURE differs from its “Write” counterpart in that it takes account of the duration of a note when playing back the sequence. Also all three sound channels are synchronised together to give a more interest tone. As it stands the play PROCEDURE plays once and then returns to the main program. If ‘P’ has been pressed more than once PROCplay will be called again immediately and produce a repeat performance. To make the melody truly repeat you could simply insert another REPEAT . . . UNTIL loop around lines 1420 to 1460. It would then only be possible to exit PROCplay by pressing <ESCAPE>, which would return you to the main program.

**PROCrite:** PROCrite PRINTs the note symbols on the screen, taking account of the note duration via the variables SYS% for the stem and SYM% for the ball. If 1%=3 a dot is added after the note (line 1570).

**PROCsharp:** This is called when ‘S’ is pressed and PRINTs a sharp in front of the current note. The X% position is moved back one step, as is the step number, so that PROCnote can be called for a second time with S%=4. The sharp symbol is made up of the user-defined character CHR\$(238) PRINTed twice.

**PROCdel:** A note deletion is attained by overprinting the existing note in



**PROCdel:** A note deletion is attained by overprinting the existing note in black (GCOLOR,0) and decrementing T% one step. Line 1810 accomplishes this, and by calling PROClines redraws the stave which has been erased in the process.

**PROCsave:** This PROCEDURE is very similar to the one described later on in the book (refer to Chapter Eleven). FILE\$ remembers the current file name

Music Graphics 125

so that it can be displayed in PROCot. Three sets of variables are saved for pitch, duration and number of notes.

**PROCload:** This is simply the opposite of PROCsave.

**PROCTRANSPOSE:** A transposition or key change is attained simply by adding or subtracting a set amount from each value of PITCH%(Z%). M% equals the number of semitones up or down.

Using this program it is easy to build a library of interesting tunes and compositions. Once the files are on disc or cassette it is not necessary to use the melody writing part of the program to play a previously recorded time. For this reason I have included below a short program which will play back “Advanced Music Writer” files. This has the advantage of releasing a large amount of memory for such uses as adding graphic

reserving a large amount of memory for each user to adding graphic accompaniments to your tunes or storing a number of tunes in the computer memory. Such extra storage could be implemented by using two-dimensional pitch and duration arrays and adding a tune choice control to the Menu.

```

1
2
3  REM"          ***TUNE PLAYER***
4
5
6
10  MODE7:REM"SET UP
20  ENVELOPE1,1,0,0,0,0,0,0,89,-1,0,-
1,126,100
30  DIM PITCH%(150)
40  DIM DUR%(150)
50  FILE$="....."
60  Tempo%=4
70  ON ERROR GOTO 130
80
90

```

```

100
110 REPEAT:REM"THE PROGRAM
120 CLS
130 PRINTTAB(4,14);CHR$(129);CHR$(141
); "***";CHR$(130); "TUNE PLAYER";CHR$(129
); "***"TAB(4,15);CHR$(129);CHR$(141); "***
*";CHR$(130); "TUNE PLAYER";CHR$(129); "***
*"
140 PRINTTAB(1,20); "The current tune
is ";CHR$(131);FILE$

```

---

126 Music Graphics

```

150 PRINT TAB(15,2)"TEMPO: ";Tempo%;"
"
160 PRINTTAB(0,6)"Press 'R' to READ f
ile""Press 'P' to play tune""Press 'T'
to change TEMPO"
170 G$=GET$
180 IF G$="R" PROCload

```

```
190 IF G$="F" AND PITCH%(1)>0 PROCpla  
Y  
200 IF G$="T" Tempo%=Tempo%+1:IF Temp  
o%>8 Tempo%=1:PRINT TAB(15,2)"TEMPO: ";T  
empo%: " "  
210 UNTIL FALSE  
220  
221  
223  
230 DEF PROCplay:REM"PLAY TUNE  
240 REPEAT  
250 E%=1  
260 REPEAT  
270 FOR Channel%=1 TO 3:SOUND&200+Chan  
nel%,1,PITCH%(E%),Tempo%*DUR%(E%):NEXT  
280 E%=E%+1  
290 UNTIL E%=T%  
300 UNTILO  
310 ENDPROC  
320
```

```
330
340
350
360 DEF PROCload:REM"LOAD FILE
370 CLS
380 INPUT"FILE NAME: ",A$
390 FILE$=A$
400 Y=OPENIN A$
410 V%=1
420 REPEAT
430 INPUT#Y,PITCH%(V%),DUR%(V%),T%
440 V%=V%+1
450 UNTIL EOF#Y
460 CLOSE#Y
470 CLS
480 ENDPROC
490
```

---

The program uses PROCload and PROCplay from “Advanced Music Writer”. A small Menu program using a GETS statement is set up to allow the choice of reading a new file, playing the current file or changing tempo. The program repeats melodies indefinitely or until <ESCAPE> is pressed to return control to the menu.

On RUNning you are immediately displayed the only page of the pro-gram. Your first action must be to press ‘R’, for Read, to load a file. Once loaded you may play it back by typing ‘P’ and stop the tune

by pressing <ESCAPE>. The only other control is for varying the tempo, which is identical to that of the “Advanced Music Writer”.

Music graphics need not be entirely practical, of course. If you look forward to the interfacing chapter, you will see that I have included a program which draws graphs of sounds entering through the Beeb’s analogue interface. The following program PRINTs coloured bars on the screen at a position related to the pitch of notes played on the keyboard.

The program uses PROCload and PROCplay from “Advanced Music Writer”. A small Menu program using a GET\$ statement is set up to allow the choice of reading a new file, playing the current file or changing tempo. The program repeats melodies indefinitely or until <ESCAPE> is pressed to return control to the menu.

On RUNNING you are immediately displayed the only page of the program. Your first action must be to press ‘R’, for Read, to load a file. Once loaded you may play it back by typing ‘P’ and stop the tune by pressing <ESCAPE>. The only other control is for varying the tempo, which is identical to that of the “Advanced Music Writer”.

Music graphics need not be entirely practical, of course. If you look forward to the interfacing chapter, you will see that I have included a program which draws graphs of sounds entering through the Beeb’s analogue interface. The following program PRINTs coloured bars on the screen at a position related to the pitch of notes played on the keyboard.

```

180 IF N% > 0 IF N% < > " " GOTO , 1 + N% MOD 3 : MOVEX%
, 44 * N% - 16 : PLOT 1 , 0 , 40 : PROCNOTE (N%) ELSE P
ROCNOTE (0)

```

```

200IFX%=1280X%=0:VDU4,12,5
210UNTIL FALSE
220DEF PROCNOTE(N%) IF N%=0 FORI%=17 T
0 19:SOUND I%,0,0,0:NEXT:ENDPROC ELSE FO
R I%=17 TO 19:SOUND I%,-15,N%*4+28,-1:NE
XT:ENDPROC

```

---

```
230 *FX 12,0
```

```
240 VDU4
```

```
250 PRINT "THE END"
```

This array, which should be familiar to you by now, is used to hold the active key information. In this example, the top two lines of letters correspond to the white and black notes. VDU 5 makes it possible to PRINT at the text cursor and Lines 130 and 140 set the key auto repeat. Line 160 tests for a keypress using INKEY\$(4) and this value is compared with P\$ in Line 170. If a valid key has been pressed a coloured bar is PLOTted on the screen at a Y co-ordinate position decided by N% and PROCNOTE is called with a positive value for N%. If there is no valid keypress nothing is drawn on the screen and PROCNOTE is called with a zero value for N%.

PROCNOTE plays all three SOUND channels with a value of one for the flush parameter of the extended SOUND statement. This means that the queue for each channel is constantly being flushed. If N%=0 then SOUND statements with zero amplitudes are called, which causes immediate silencing of the tone generators. If N%>0 the three tone generators play notes of a pitch calculated as  $N\% * 4 + 28$ . To exit the program <ESCAPE> can be pressed. This causes an error 17, which sends the program to Line 230. The keyboard auto-repeat is then set to default, normal cursor control is returned and the program ends.



## 8: Automatic Composition

One of the most interesting areas being developed in the field of computer music is called autocomposition. As its name suggests, this technique involves allowing the computer to compose music without the direct influence of a human being.

The crudest method of achieving autocomposition is simply to feed random numbers into the SOUND statement as Pitch values. The following program is unlikely to enthrall anyone who is not completely tone deaf for longer than about fifteen seconds but ably illustrates the results of the above-mentioned method:

```
10 REPEAT:Pitch=RND(64): SOUND 1,-15,Pitch*4,5:UNTIL0
```

One of the reasons why this type of ‘music’ is extremely unsatisfying to listen to is because it is virtually structureless. Over the decades, human beings have come to expect certain musical guidelines to be adhered to and while these guidelines vary from culture to culture their absence jars on the ears and the brain. A Japanese audience, by contrast with their Western equivalent, can remain riveted to their futons for hours on end listening to the musical structures which their ears have been trained to appreciate. Outsiders, however, would find it difficult to stifle yawns before a few minutes were up! Of course, this does not make the Japanese better or worse musicians than Westerners, but simply emphasises the fact that their musical vocabulary is different to ours.

The BBC Microcomputer has no inherent musical vocabulary whatsoever. To persuade it to compose sensible music we must first identify the basic principles involved in making music, then find a way of inputting this information into the Beeb. It is not feasible to attempt to give the computer a general musical education, for the simple reason that we would come up against memory constraints. Luckily this is not necessary (or even desirable). Most human musicians are actually severely limited in the range of music they can play and these limitations are much more severe than just a matter of Norwegian musicians being unable to play Indian tunes and vice versa. Most Jazz musicians have little experience of classical music and few classical musicians spend their leisure hours playing jazz. Consequently, when we try to musically educate the BBC, we must try to follow the human example and stick to a narrow stylistic vocabulary.

As I indicated in the Music Theory chapter over the centuries music has developed from being a matter of simple modes and scales to the plethora of styles and techniques which we enjoy and appreciate today. If you look around, you will notice that, of late, both jazz and classical music seem to have removed all harmonic restrictions from the shoulders of the composer. For this reason, amongst others the computer does not find it at all daunting to emulate modern avant garde styles!

...

In this program the random number generator is only restricted to values for Pitch within the BBC's range and Durations which are reasonably short. Lovers of Shoenberg and the like should find this program extremely satisfying! In order to titillate the rest of us, I have also included a second avant garde program. Avant Garde 2 restricts the pitch interval from one note to the next. This is accomplished simply by introducing a variable P% which remembers the previous Pitch value. The REPEAT... UNTIL loop at Line 35 then holds back Pitches which would precipitate too big a leap:

...

OK, I admit it. This is not a fantastic improvement on our first attempt. Nevertheless, the technique of restricting intervals does make for more natural-sounding musical lines. You will doubtlessly be pleased to hear that further progress towards harmony are still possible. By far the biggest step towards aesthetically pleasing music is taken when we restrict the computer to scales. The next program sets up an array P%(22) which contains notes of the C Major scale. Line 150 then randomly picks values from the array.

---

```
10 REM"***MUSIC 1***
```

...

Automatic Composition 131

harmony are still possible. By far the biggest step towards aesthetically pleasing music is taken when we restrict the computer to scales. The next program sets up an array P%(22) which contains notes of the C Major scale. Line 150 then randomly picks values from the array.

```
10 REM"***MUSIC 1 ***
20
30 REM"Set up Array and Initialise
40
```

```

30 REM "Set up Array and initialise
40
50 CLS
60 ENVELOPE 1,1,0,0,0,0,0,0,126,-1,0,
-1,110,80
70 DIM P%(22)
80 FOR X%=1 TO 22:READ P%(X%):NEXT
90 P%=73
100
110 REM "The program
120
130 REPEAT
140   REPEAT
150     Pitch%=P%(RND(21)+1)*4+5
160   UNTIL Pitch%-P%<32 AND Pitch%-P%>
-32
170   Duration%=5
180   SOUND1,1,Pitch%,Duration%
190   P%=Pitch%
200 UNTIL FALSE
210
220 REM "The Note Source
230
240 DATA 0,2,4,5,7,9,11,12,14,16,17,19
,21,23,24,26,28,29,31,33,35,36

```

Note that in all the remaining programs in this chapter I have chosen a particular convention for recording Pitch values. Since at no point will we be dealing with intervals which are smaller than a semitone, I have made a semitone interval equal to 1 unit. Bearing in mind that this is the generally accepted convention for synthesisers, I have used the following formula to convert to BBC nomenclature from the absolute pitch values used by instruments such as the Roland MC-202 and the MC-4:

$$(\text{Pitch} \star 4) + 5 = \text{BBC Pitch value}$$

$$(\text{Pitch} \times 4) + 5 = \text{BBC Pitch value}$$

The table which follows accordingly gives the conversion from music

Note that in all the remaining programs in this chapter I have chosen a particular convention for recording Pitch values. Since at no point will we be dealing with intervals which are smaller than a semitone, I have made a semitone interval equal to 1 unit. Bearing in mind that this is the generally accepted convention for synthesisers, I have used the following formula to convert to BBC nomenclature from the absolute pitch values used by instruments such as the Roland MC -202 and the MC-4:

$$(\text{Pitch} \times 4) + 5 = \text{BBC Pitch value}$$

The table which follows accordingly gives the conversion from music symbols. You should quickly find that you can remember all the value of the notes which we use frequently.

For example, the C Major scale becomes:

---

## 132 Automatic Composition

symbols. You should quickly find that you can remember all the value of the notes which we use frequently.

For example, the C Major scale becomes:

<b>C</b>	<b>D</b>	<b>E</b>	<b>F</b>	<b>G</b>	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E . . .</b>
<b>0</b>	<b>2</b>	<b>4</b>	<b>5</b>	<b>7</b>	<b>9</b>	<b>11</b>	<b>12</b>	<b>14</b>	<b>16 . . .</b>

	Octave				
Note	1	2	3	4	5
C	0	12	24	37	48
D	2	14	26	38	50
E	4	16	28	40	52
F	5	17	29	41	53
G	7	19	31	43	55
A	9	21	33	45	57
B	11	23	35	47	59

The “Music 1” program contains all the main components required for auto composition:

1. A note source, usually a DATA statement containing, in this case, three octaves of the C Major scale.
2. An array to hold this note information while the program is operating (in this example P%(22)). Since each entry in the note source is numbered it is a simple matter to set up a random select routine which picks notes from the array.
3. A random choice statement. In “Music 1” this takes the form:

$$\text{Pitch\%} = \text{P\%}(\text{RND}(21) + 1) * 4 + 5$$

Random values of P%(22) between 1 and 22 are chosen and converted to BBC Pitch values using the formula as shown.

4. The pitch interval restriction is accomplished first by remembering the previous Pitch% in Line 190, with P%=Pitch%. The value of P% is then compared with the new Pitch% in Line 160. If the difference between the values (i.e. the size of interval) is greater than a set value (32 in this case, equalling 8 semitones) the loop is forced to REPEAT . . . UNTIL an acceptable value is generated.

Different intervals can be tried in order to hear their effect. Basically, the larger the interval the more erratic the music. The smaller the interval the more predictable the sound. As is so frequently the case in music, we

The “Music 1” program contains all the main components required for auto composition:

1. A note source, usually a DATA statement containing, in this case, three octaves of the C Major scale.
2. An array to hold this note information while the program is operating (in this example P%(22)). Since

each entry in the note source is numbered it is a simple matter to set up a random select routine which picks notes from the array.

3. A random choice statement. In “Music 1” this takes the form:

```
Pitch% = P%(RND(21)+ 1)*4+ 5
```

Random values of P%(22) between 1 and 22 are chosen and converted to BBC Pitch values using the formula as shown.

4. The pitch interval restriction is accomplished first by remembering the previous Pitch% in Line 190, with P% =Pitch%. The value of P% is then compared with the new Pitch% in Line 160. If the difference between the values (i.e. the size of interval) is greater than a set value (32 in this case, equalling 8 semitones) the loop is forced to REPEAT... UNTIL an acceptable value is generated.

Different intervals can be tried in order to hear their effect. Basically, the larger the interval the more erratic the music. The smaller the interval the more predictable the sound. As is so frequently the case in music, we are obliged to play off the twin evils of monotony and chaos...

The combination of restricted interval jumps and a scalar note source has given us harmonically acceptable music (for the moment we will neglect timing). Nevertheless merely listening to all the possible note combinations within a single Major scale eventually does get a little dull. The following program accordingly improves on “Music 1” in two areas. It first introduces a further note priority within the scale, and secondly a chord change is also introduced.

are obliged to play off the twin evils of monotony and chaos . . .

The combination of restricted interval jumps and a scalar note source has given us harmonically acceptable music (for the moment we will neglect timing). Nevertheless merely listening to all the possible note combinations within a single Major scale eventually does get a little dull. The following program accordingly improves on “Music 1” in two areas. It first introduces a further note priority within the scale, and secondly a chord change is also introduced.

```
10 REM"***MUSIC 2 ***
```

```
20
```

```
30 REM"Set up Array and Initialise
```

```
20
30 REM"Set up Array and Initialise
40
50 CLS
60 ENVELOPE 1,1,0,0,0,0,0,0,126,-1,0,
-1,110,80
70 DIM P%(42,2)
80 FOR Y%=1 TO 2:FOR X%=1 TO 42:READ
P%(X%,Y%):NEXT:NEXT
90 P%=73
100
110 REM"The program
120
130 REPEAT
140 FOR Y%=1 TO 2
150 FOR Notes%=1 TO 8
160 REPEAT
170 Pitch%=P%(RND(21)+1,Y%)*4+5
180 UNTIL Pitch%-P%<32 AND Pitch%-P%>
-32 AND Pitch%<>P%
190 Duration%=5
200 SOUND1,1,Pitch%,Duration%
210 P%=Pitch%
220 NEXT
230 NEXT
240 UNTIL FALSE
250
260 REM"The Note Source
270
280 REM"For this program we have
290 REM"tried aNote Sourceof
300 REM"Major Scale/Arpeggio.
310
320 REM"C Major
```

```

330 DATA 0,4,7,12,16,19,24,28,31,36,40
,43,48,0,2,4,5,7,9,11,12,14,16,17,19,21,
23,24,26,28,29,31,33,35,36,38,40,41,43,4
5,47,48
340 REM"G Major
360 DATA 2,7,11,14,19,23,26,31,35,38,4
3,47,50,2,4,5,7,9,11,12,14,16,17,19,21,2
3,24,26,28,29,31,33,35,36,38,39,41,43,45
,47,48,50

```

The note source in this program has been modified to include the Major chord in addition to the Major scale. This means that the notes C, E and G (0,4,7 . . .) for C Major and G, B and D (2,7,11 . . .) occur twice in their respective arrays. This makes it twice as likely that a chord note will be randomly chosen. This simple assumption is in fact, very effective in making the computer's musical doodling more reminiscent of a human-style effort at composition.

Note that one further interval restriction has also been added. If  $\text{Pitch}\% = \text{P}\%$  then a REPEAT is forced. This stops the computer repeating the same note.

A chord change is introduced simply by setting up a two dimensional pitch array,  $\text{P}\%(42,2)$  and alternating between the two dimensions every eight notes. It should be noted that since the key of this piece is C, I have used a mode of C Major as G scale information. As a general rule, the scale information should remain fixed no matter what chords are being played. The exception to this rule occurs when a key change occurs during a piece.

A key change was common to the 'standard' songs of the 40s and 50s but is less frequently encountered when you look at contemporary pop and folk music. For this reason the computers improvisations are more closely related to these simpler genres than the musically sophisticated



and folk music. For this reason the computers improvisations are more closely related to these simpler genres than the musically sophisticated standards of yesterday.

You might like to try substituting a G Major scale for the C mode as an experiment. As well as this, you should try different scales and arpeggios as note sources. The listing below gives the DATA for Pentatonic scales and Natural Minor scales, with arpeggios. Other possibilities include blues, diminished and whole tone scales, and these scales are given here:

The note source in this program has been modified to include the Major chord in addition to the Major scale. This means that the notes C, E and G (0,4,7...) for C Major and G, B and D (2,7,11...) occur twice in their respective arrays. This makes it twice as likely that a chord note will be randomly chosen. This simple assumption is in fact, very effective in making the computer's musical doodling more reminiscent of a human-style effort at composition.

Note that one further interval restriction has also been added. If Pitch%=P% then a REPEAT is forced. This stops the computer repeating the same note.

A chord change is introduced simply by setting up a two dimensional pitch array, P%(42,2) and alternating between the two dimensions every eight notes. It should be noted that since the key of this piece is C, I have used a mode of C Major as G scale information. As a general rule, the scale information should remain fixed no matter what chords are being played. The exception to this rule occurs when a key change occurs during a piece.

A key change was common to the 'standard' songs of the 40s and 50s but is less frequently encountered when you look at contemporary pop and folk music. For this reason the computers improvisations are more closely related to these simpler genres than the musically sophisticated standards of yesterday.

You might like to try substituting a G Major scale for the C mode as an experiment. As well as this, you should try different scales and arpeggios as note sources. The listing below gives the DATA for Pentatonic scales and Natural Minor scales, with arpeggios. Other possibilities include blues, diminished and whole tone scales, and these scales are given here:

Scale      Interval   sequence

Major                      8    8    4    8    8    8    4

Diminished	4	8	4	8	4	8	4	8
Blues	12	8	4	4	12	8		
Hindu	8	8	4	8	4	8	8	
Wholetone	8	8	8	8	8	8		
Dorian Minor	8	4	8	8	8	4	8	
Aeolian Minor	8	4	8	8	4	8	8	
Harmonic Minor	8	4	8	8	4	12	4	
Pentatonic	8	8	12	8	12			

Automatic Composition 135

Scale	Interval sequence							
Major	8	8	4	8	8	8	4	
Diminished	4	8	4	8	4	8	4	8
Blues	12	8	4	4	12	8		
Hindu	8	8	4	8	4	8	8	
Whole tone	8	8	8	8	8	8		
Dorian Minor	8	4	8	8	8	4	8	
Aeolian Minor	8	4	8	8	4	8	8	
Harmonic Minor	8	4	8	8	4	12	4	
Pentatonic	8	8	12	8	12			

100

200 REM" MUSIC PROGS DATA

250

300 REM"D minor Chord/scale

310 DATA 2,5,9,14,17,21,26,29,33,38,41  
,45,48,2,4,5,7,9,11,12,14,16,17,19,21,23  
,24,26,28,29,31,33,35,36,38,40,41,43,45,

,24,26,28,29,31,33,35,36,38,40,41,43,45,  
47,48

320 REM"G Major Chord/scale

330 DATA 2,7,10,14,19,22,26,31,34,38,4  
3,46,50,2,4,5,7,9,10,12,14,16,17,19,21,2  
2,24,26,28,29,31,33,34,36,38,40,41,43,45  
,46,48

340

350

360

370

380

390

400 REM"C Pentatonic

410 DATA 0,2,4,7,9,12,14,16,19,21,24,2  
6,28,31,33,36,38,40,43,45,48,50,52,55,57  
,60

420 REM"G Pentatonic

430 DATA 2,4,7,9,11,14,16,19,21,23,26,  
28,31,33,35,38,40,43,45,47,50,52,55,57,5  
9,62

The music the computer is now composing closely follows many of the basic musical rules laid down by J.S. Bach and his predecessors. It is solidly grounded in the notes of the relevant chord and also employs scale notes to a lesser degree. No accidentals (sharp or flat) are used at all. It is,

The music the computer is now composing closely follows many of the basic musical rules laid down by J. S. Bach and his predecessors. It is solidly grounded in the notes of the relevant chord and also employs scale notes to a lesser degree. No accidentals (sharp or flat) are used at all. It is, however, rather difficult to assess how well the computer is sticking to the chord changes. To make this easier why don't you try the program "Bach 1". This uses the same melody generation routine as "Music 2" but adds a two note chord accompaniment.

## 136 Automatic Composition

however, rather difficult to assess how well the computer is sticking to the chord changes. To make this easier why don't you try the program "Bach 1". This uses the same melody generation routine as "Music 2" but adds a two note chord accompaniment.

```

10 REM"***BACH 1 ***
20
30 REM"Set up Arrays and Initialise
40
50 CLS
60 ENVELOPE 1,1,0,0,0,0,0,0,126,-1,0,
-1,110,80
70 ENVELOPE 2,1,0,0,0,0,0,0,100,-3,-2
0,-40,90,0
80 ENVELOPE 3,1,0,0,0,0,0,0,0,0,0,0,0,0
,0
90 PRINTTAB(4,10);CHR$(133);"Press M
for Major Scale/Arpeggio"
100 PRINTTAB(4,12);CHR$(133);"Press P
for Pentatonic Scale"
110 REPEAT
120 A$=GET$
130 IF A$="M" RESTORE 490:S%=42:DIM P%
(S%,2)
140 IF A$="P" RESTORE 610:S%=26:DIM P%
(S%,2)
150 UNTIL A$="P" OR A$="M":CLS
160 FOR Y%=1 TO 2:FOR X%=1 TO S%:READ
P%(X%,Y%):NEXT:NEXT
170 P%=73
180 Chord%=1
190

```

```

180  LNOFO%=1
190
200  REM"The program
210
220  REPEAT
230    FOR Y%=1 TO 2
240      FOR Notes%=1 TO 4
250        FOR C%=1 TO 2
260          IF C%=1 E%=2 ELSE E%=3
270          REPEAT
280            Pitch%=P%(RND(21)+1,Y%)*4+5
290            UNTIL Pitch%-P%<32 AND Pitch%-
P%>-32 AND Pitch%<>P%
300            Duration%=5

```

Automatic Composition 137

```

310      IF Y%=1 SOUND&0202,E%,53,5:SOU
ND&0203,E%,69,5
320      IF Y%=2 SOUND&0202,E%,61,5:SOU
ND&0203,E%,81,5
330      SOUND&0201,1,Pitch%,Duration%
340      P%=Pitch%
350    NEXT
360  NEXT
370 NEXT
380 UNTIL FALSE
390
400 REM"The Note Source
410
420 REM"For this program we have
430 REM"Aa choice.....
440
450 REM"1. Major Scale/Arpeggio.

```

```
450 REM"1. Major Scale/Arpeggio.
460
470 REM"C Major
480
490 DATA 0,4,7,12,16,19,24,28,31,36,40
,43,48,0,2,4,5,7,9,11,12,14,16,17,19,21,
23,24,26,28,29,31,33,35,36,38,40,41,43,4
5,47,48
500
510 REM"G Major
520
530 DATA 2,7,11,14,19,23,26,31,35,38,4
3,47,50,2,4,5,7,9,11,12,14,16,17,19,21,2
3,24,26,28,29,31,33,35,36,38,39,41,43,45
,47,48,50
540
550 REM"Dr.....
560
570 REM"2. Pentatonic
580
590 REM"C Pentatonic
600
610 DATA 0,2,4,7,9,12,14,16,19,21,24,2
6,28,31,33,36,38,40,43,45,48,50,52,55,57
,60
620
630 REM"G Pentatonic
```

---

640

```
650 DATA 2,4,7,9, 11, 14, 16,
19,21,23,26,28,31,33,35,38,40,43,45,47,50,52,55,57,59,62
```

If we think of the melody line as a series of quavers (1/8th notes) then the chords only sound on the beat (every 1/4 note). In point of fact, a chord SOUND statement is put in the queue each time a melody note is played. This makes for easy SOUND synchronisation using the extended SOUND statement. The S value is set at 2 (two statements on other channels must be collected before all three can SOUND together), and the off beat rests are attained by using a silent volume ENVELOPE (ENVELOPE 3).

The loop Notes% is split in half and another loop C% is inserted. When C%=1 (an on beat) ENVELOPE 2 is selected and a chord is heard. When C%=2 (an off beat) ENVELOPE 3 is selected and no chord is heard.

“Bach 1” also lets you select either a Pentatonic or a Major chord/scale note source. Note that as it stands the program must be restarted to select the opposing set of DATA.

One possible extension of letting the computer compose melodies would be giving it control over chords. In practice, this move generates very unsatisfactory music. This is because even though there are a large number of possible sequences in theory, only a few are actually employed. A more constructive improvement to the program would be to let the user INPUT his or her own sequence and have the computer improvise variations over the changes. The next program allows you to do just that.

```

10
20
30 REM" ***BACH 2 ***
40
50
60 REM"Set up and READ's DATA
70 REM" into the Arrays....
80
90 CLS
100 ENVELOPE 1,1,0,0,0,0,0,0,126,-1,0,-1,110,80
110 ENVELOPE 2,1,0,0,0,0,0,0, 100,-3,-20, -40, 90, 0
120 ENVELOPE 3,1,0,0,0,0,0,0,0,0,0,0,0,0
```

```
130 S%=42:DIM P%(S%,7),Chord%(2,7),SEQ%(8)
```

Automatic Composition 139

```

140 FOR Y%=1 TO 7:FOR X%=1 TO S%:READ
P%(X%,Y%):NEXT:NEXT
150 FOR Y%=1 TO 7:FOR X%=1 TO 2:READ C
hord%(X%,Y%):NEXT:NEXT
160 F%=73:Y%=0
170
180 REM"Choose the Chords
190
200 FOR Q%=1 TO 8:PRINTTAB(6,8);CHR$(1
34);"Chord ";Q%:INPUTTAB(8,10),SEQ%(Q%):
CLS:NEXT
210 CLS:PRINTTAB(2,10);CHR$(134);"Bar
No. : 1 2 3 4 5 6 7 8"
220 PRINTTAB(3,12)"Sequence:"
230 FOR Q%=1 TO 8:PRINTTAB(Q%*3+10,12)
;SEQ%(Q%):NEXT
240
250
260 REM"The program
270
280 REPEAT
290   FOR Q%=1 TO 8
300     Y%=SEQ%(Q%)
310     FOR Notes%=1 TO 4
320       FOR C%=1 TO 2
330         IF C%=1 E%=2 ELSE E%=3
340         REPEAT
350           Pitch%=P%(RND(41)+1,Y%)*4+5
360           UNTIL Pitch%-P%<26 AND Pitch%-

```



```
360      UNTIL Pitch%-F%<26 AND Pitch%-  
P%>-26 AND Pitch%<>P%  
370      Duration%=5  
380      SOUND&0202,E%,Chord%(1,Y%)*4+5  
,5:SOUND&0203,E%,Chord%(2,Y%)*4+5,5  
390      IF RND(100)>20 SOUND&0201,1,Pi  
tch%,Duration% ELSE SOUND&1201,3,Pitch%,  
Duration%  
400      F%=Pitch%  
410      NEXT  
420      NEXT  
430      NEXT  
440 UNTIL FALSE  
450  
460 REM"The Note Source  
470
```

---

## 140 Automatic Composition

```
480 REM"This program uses chordal  
490 REM"notes + C Major scale...  
500  
510  
520  
530 REM"C Major  
540  
550 DATA 0,4,7,12,16,19,24,28,31,36,40  
,43,48,24,2,4,5,7,9,11,12,14,16,17,19,21  
,23,24,26,28,29,31,33,35,36,38,40,41,43,  
45,47,36  
560 REM"D minor  
570  
580 DATA 2,5,9,14,17,21,26,29,33,38,41  
,45,50,2,4,5,7,9,11,12,14,16,17,19,21,23
```

,45,50,2,4,5,7,9,11,12,14,16,17,19,21,23  
,24,26,28,29,31,33,35,36,38,40,41,43,45,  
47,48,50

590 REM"E minor

600

610 DATA 4,7,11,16,19,23,28,31,35,40,4  
3,47,52,4,5,7,9,11,12,14,16,17,19,21,23,  
24,26,28,29,31,33,35,36,38,40,41,43,45,4  
7,48,50,52

620 REM"F Major

630

640 DATA 0,5,9,12,17,21,24,29,33,36,41  
,45,48,0,2,4,5,7,9,11,12,14,16,17,19,21,  
23,24,26,28,29,31,33,35,36,38,40,41,43,4  
5,47,48

650 REM"G Major

660

670 DATA 2,7,11,14,19,23,26,31,35,38,4  
3,47,31,2,4,5,7,9,11,12,14,16,17,19,21,2  
3,24,26,28,29,31,33,35,36,38,40,41,43,45  
,47,48,43

680 REM"A minor

690

700 DATA 0,4,9,12,16,21,24,28,33,36,40  
,45,48,0,2,4,5,7,9,11,12,14,16,17,19,21,  
23,24,26,28,29,31,33,35,36,38,40,41,43,4  
5,47,48

710

720 REM"B diminished

730

```

740 DATA 2,7,11,14,19,23,26,31,35,38,4
3,47,35,2,4,5,7,9,11,12,14,16,17,19,21,2
3,24,26,28,29,31,33,35,36,38,40,41,43,45
,47,48,43

```

```
750
```

```
760 REM"Chord DATA
```

```
770
```

```

780 DATA 12,16,14,17,16,19,17,21,19,23
,21,24,23,26

```

The seven DATA statements contain chord/scale information for the seven possible triad chords generated from any Major scale. The following table shows how the triads are arrived at for C Major:

Note	Chord	Name
C	C E G	C Major
D	D F A	D Minor
E	E G B	E Minor
F	F A C	F Major
G	G B D	G Major
A	A C E	A Minor
B	B D F	B Diminished

These seven chords allow any melody in C Major to be harmonised and the technique itself can be applied to all the other keys to generate chords. In this way it becomes possible to harmonise melodies that change key. For the moment, however, let us walk before we run and accordingly stick with the simplified case of one key.

Each DATA statement contains the relevant chord notes over four octaves, plus the C Major scale notes. Before the improvisation starts you are asked to input an eight-bar sequence of chords. The American chord nomenclature is used for this information.

Chord:	C	Dm	Em	F	G	Am	Bdim
No. :	1	2	3	4	5	6	7

Chord:	C	Dm	Em	F	G	Am	Bdim
No.:	1	2	3	4	5	6	7

This numerical notation has the advantage over the British chord symbol nomenclature in that it is independent of key. The root note of the scale is always labelled 1 – and so on. If we wished to enter the sequence:

C      G      C      G      Am      Dm      G      C

we would type:

1      5      1      5      6      2      5      1

142 Automatic Composition

The most frequently used chords are 1, 4 and 5 (C,F and G in the key of C). These make up the infamous three chord trick so frequently used in pop music. The following sequence is pleasingly familiar:

1      4      1      5      1      4      5      1

The blues sequence also uses these notes, but since it employs a twelve-bar cycle the program would have to be adapted to exploit the sequence. Other useful changes include the 2 5 1 sequence:

1      2      5      1      1      2      5      1

You could also try the following:

6      2      5      1      6      2      5      1

1      2      1      5      7      6      5      4

“Bach 2” also contains a straightforward method of introducing rests into the improvisation. Line 390 allows a note to SOUND as usual if a random number between 0 and 100 is over 20. If this is not the case a rest is

the improvisation. Line 390 allows a note to SOUND as usual if a random number between 0 and 100 is over 20. If this is not the case a rest is introduced using:

SOUND &1201, 3, Pitch%, Duration%

A value of one for the extended SOUND statements H parameter puts a silent note of length defined by the value of Duration% into the channel one queue and allows the release phase of the previous note to continue.

A greater percentage of rests can be created by changing the 20 in line 390 to a higher number. 100 would give all rests and zero would give no rests. You are probably starting to realise that the law of diminishing returns very definitely applies to autocomposition, since we are rapidly approaching the point where it would be accurate to say that only a large increase in the number of program lines and DATA statements will give us a substantial increase in program musicality.

The style of the improvisation can easily be changed, however, and you do this quite simply by changing the note source. One possibility is to try including a blues scale as DATA. The exact composition of each statement would have to be arrived at by a process of trial and error, but you could start by experimenting with increasing the size of arrays and adding to the DATA. The C blues scale comprises the following notes:

C	E	F	F	G	B	C
0	3	5	6	7	10	12

The final program in this series once again restricts the random choice of notes. Within a bar, the on-beat notes are more important harmonically than the off-beats and thus if the random choice is restricted to chord notes at these points within the bar a more positive chord sound will be heard. The remaining scale notes can then act as passing notes. It is a question of personal taste to decide whether the decrease in melodic variation is a fair price to pay for this result. The manoeuvre is accomp-

variation is a fair price to pay for this result. The manoeuvre is accomplished by Line 350. When C%=1 we have an on-beat and the random choice is restricted to the first thirteen notes of the array. (i.e. the chordal notes). When C%=2 the choice is made from the entire array.

Another restriction which it would be possible to make along these lines is to make the restricted choice occur only on beats 1 and 3, and this is done specifying Notes%=1.

Other updates to be found in this program include a tempo control (by INPUTting a value for Duration% in Line 171), a chord sequence choice of anything up to 100 entries (lines 191 to 230), a drum effect (Line 375), and first beat in the bar emphasis (using ENVELOPE 5 as the accent chosen in line 390).

```

10
20
30  REM"***BACH 3 ***
40
50
60  REM"Set up and READ's DATA
70  REM"into the Arrays....
80
90  CLS
100  ENVELOPE 1,1,0,0,0,0,0,0,126,0,0,
-1,110,110
110  ENVELOPE 2,1,0,0,0,0,0,0,100,-3,-
20,-40,100,0
120  ENVELOPE 3,1,0,0,0,0,0,0,0,0,0,0,
0,0
125  ENVELOPE 4,1,0,0,0,0,0,0,126,-4,-
8,-20,80,10
126  ENVELOPE 5,1,0,0,0,0,0,0,126,0,0,
-1,126,126
130  S%=42: DIM P%(S%,7),Chord%(2,7),SE
Q%(100)
140  FOR Y%=1 TO 7:FOR X%=1 TO S%:READ

```

```

140  FOR Y%=1 TO 7:FOR X%=1 TO 5:READ
P%(X%,Y%):NEXT:NEXT
150  FOR Y%=1 TO 7:FOR X%=1 TO 2:READ
Chord%(X%,Y%):NEXT:NEXT

```

---

## 144 Automatic Composition

```

160  P%=73:Y%=0:I%=1
170
171  INPUTTAB(6,4);"What duration% do
you want",Duration%
172  CLS
180  REM"Choose the Chords
190
191  INPUTTAB(0,4);"How many chords ar
e in your sequence",number%
192  IF number%>12 OR number%<1 THEN C
LS:GOTO 191
200  FOR Q%=1 TO number%:PRINTTAB(6,8)
;CHR$(134);"Chord ";Q%:INPUTTAB(8,10),SE
Q%(Q%):CLS:NEXT
205  FOR Q%=1 TO number%:IF SEQ%(Q%)<1
OR SEQ%(Q%)>7 THEN PRINT "Input out of
range. Please re-enter.":TIME=0:REPEAT:U
NTIL TIME>300:CLS:GOTO 200
206  NEXT
210  CLS: PRINTTAB(0,10);CHR$(134);
230  FOR Q%=1 TO number%:PRINTTAB(Q%*3
,12);SEQ%(Q%);TAB(Q%*3,10);Q%:NEXT
240
250
260  REM"The program

```

```

260  REM"The program
270
280  REPEAT
290    FOR Q%=1 TO number%
300      Y%=SEQ%(Q%)
310      FOR Notes%=1 TO 4
320        FOR C%=1 TO 2
330          IF C%=1 E%=2 ELSE E%=3
335          IF C%=1 AND Notes%=1 THEN L%=
5 ELSE L%=1
340          REPEAT
350            IF C%=1 Pitch%=P%(RND(12)+1,
Y%)*4+5 ELSE Pitch%=P%(RND(41)+1,Y%)*4+5
360            UNTIL Pitch%-P%<26 AND Pitch%
-P%>-26
375            IF C%=2 SOUND&0300,4,4,Durati
on% ELSE SOUND&0300,4,6,Duration%
380            SOUND&0301,E%,Chord%(1,Y%)*4+
5,Duration%:SOUND&0303,E%,Chord%(2,Y%)*4
+5,Duration%

```

Automatic Composition 145

```

390      IF RND(100)>20 SOUND&0302,L%,
Pitch%,Duration% ELSE SOUND&1302,3,Pitch
%,Duration%
400      P%=Pitch%
410    NEXT
420  NEXT
430  NEXT
440  UNTIL FALSE
450
460  REM"The Note Source

```



```
460  REM"The Note Source
470
480  REM"This program uses chordal
490  REM"notes + C Major scale...
500
510
520
530  REM"C Major
540
550  DATA 0,4,7,12,16,19,24,28,31,36,4
0,43,48,24,2,4,5,7,9,11,12,14,16,17,19,2
1,23,24,26,28,29,31,33,35,36,38,40,41,43
,45,47,36
560  REM"D minor
570
580  DATA 2,5,9,14,17,21,26,29,33,38,4
1,45,50,2,4,5,7,9,11,12,14,16,17,19,21,2
3,24,26,28,29,31,33,35,36,38,40,41,43,45
,47,48,50
590  REM"E minor
600
610  DATA 4,7,11,16,19,23,28,31,35,40,
43,47,52,4,5,7,9,11,12,14,16,17,19,21,23
,24,26,28,29,31,33,35,36,38,40,41,43,45,
47,48,50,52
620  REM"F Major
630
640  DATA 0,5,9,12,17,21,24,29,33,36,4
1,45,48,0,2,4,5,7,9,11,12,14,16,17,19,21
,23,24,26,28,29,31,33,35,36,38,40,41,43,
45,47,48
650  REM"G Major
660
```

## 146 Automatic Composition

```

670 DATA 2,7,11,14,19,23,26,31,35,38,
43,47,31,2,4,5,7,9,11,12,14,16,17,19,21,
23,24,26,28,29,31,33,35,36,38,40,41,43,4
5,47,48,43
680 REM"A minor
690
700 DATA 0,4,9,12,16,21,24,28,33,36,4
0,45,48,0,2,4,5,7,9,11,12,14,16,17,19,21
,23,24,26,28,29,31,33,35,36,38,40,41,43,
45,47,48
710
720 REM"B diminished
730
740 DATA 2,7,11,14,19,23,26,31,35,38,
43,47,35,2,4,5,7,9,11,12,14,16,17,19,21,
23,24,26,28,29,31,33,35,36,38,40,41,43,4
5,47,48,43
750
760 REM"Chord DATA
770
780 DATA 12,16,14,17,16,19,17,21,19,2
3,21,24,23,26

```

You will quickly discover that values of around five for Duration% give the most realistic results. The extended sequence modification allows you to be more flexible with your changes. Try entering the blues sequence: 1 1 1 1 4 4 1 1 5 4 1 1 ... 12 entries)

or the following possibility:

1 4 2 5 4 6 7 5 (8 entries)

1 4 2 5 4 6 7 5 . . . (8 entries)

**These programs are not intended to illustrate the definitive method of creating computer-generated music. Instead, they are meant to serve as an indication of the types of considerations which must be borne in mind when attempting to set the BBC off on the right musical foot!**

## 9 Applications

In this chapter we shall explore some of the possible applications of our expertise with SOUND and ENVELOPE. Many uses have already been covered in earlier chapters so in the following pages I shall restrict myself to what can best be described as ‘more developed programs’ than those listed to date.

Apart from a few exceptions, the programs we have looked at up till now have been stripped down to their basic components. For example, our use of graphics has been purely functional, and many routines which would certainly have to be included in any program intended for serious applications have been omitted. The main category of omissions in this area come under the general heading of ‘Idiot Trapping’ routines, routines which are intended to prevent a program crash when a wrong key is hit accidentally.

The programs which follow now are immune to all but the most determined ‘Idiot’ who is going to press <BREAK> or <ESCAPE> come hell or high water! I could, of course, have disabled these keys too. Programming the <BREAK> key to list the program is accomplished by the simple expedient:

```
*KEY 10 OLD M LIST M <RETURN>
```

We could equally well program the key to re-enter the program at the <BREAK> point. To disable the <ESCAPE> key we use the \*FX command \*FX 229,1. To re-enable this key you only need to type \*FX 229,0.

Nevertheless, all misgivings pushed aside, idiot trapping of this type has been left out - as a result the programs are more easily keyed-in and do not become over-long.

The programs in this section are not arranged in any particular order so I would suggest that you look through, and then start with whichever one most takes your fancy. The ‘Patriotic Program’ plays a stirring three-part version of ‘Rule Britannia’ while hauling a Union Jack up and down a flag pole. ‘Clypsos 2’ is an autocomposition program with, appropriately enough, a distinctly Caribbean feel. The ‘Chord Organ’ program is a short routine which lets you play three-note chords by pressing keys on the keyboard. ‘Multi 1’ (courtesy of Igor Thomas), on the other hand, is a neat

**real-time sequencer which can be used with either an external keyboard or the Beeb’s alphanumeric. The ‘Drum Synth’ program turns the BBC into a Linn Drum (well, almost!) which has the facility to chain your rhythm**

a Linn Drum (well, almost!) which has the facility to chain your rhythm patterns into songs. Finally, you will come across an intriguing little routine which plays a scale which seems to carry on indefinitely . . .

## A PATRIOTIC PROGRAM

There is no need for me to supply instructions to accompany this program, as you simply type it in, save a copy for safety and RUN. When you look at the screen, you should see a Union Jack appear at the bottom of a flagpole. This will be accompanied by a three-part harmony version of 'Rule Britannia'. Deciding whether or not you wish to stand up while the program is playing is optional!

On completion of one chorus of the melody the flag will move up the pole and the tune will modulate up a semitone. This scenario will be repeated a number of times before the flag reaches the top of the pole. At this point (surprise, surprise!) the flag starts to travel down the pole and the melody modulates down in semitones. The program is, in fact, endless – and would make excellent after dinner entertainment for a British Legion gathering.

```

10
20
30
40 REM" *****
50 REM" ***A PATRIOTIC PROGRAM***
60 REM" *****
70
80
90
100 REM"      ***THE PROGRAM***
110
130 PROCinit
140 REPEAT
150  RESTORE 680

```

```

150  RESTORE 680
160  MODE 2
170  VDU 23;8202;0;0;0;
180  PROCpole
190  PROCflag(200+W%*10)
200  PROCplay
210  W%=W%+L%
220  IF W%>52 OR W%<2 L%=-L%
230  UNTIL FALSE

```

Applications 149

```

240  END
250
260
270  REM"      ***INITIALISATION***
280
290  DEFPROCinit
300  VDU 23,79,24,24,24,24,24,24,24,24
310  VDU 23,80,0,0,0,0,0,0,255,255
320  K$="BC*D*EF*G*A*"
330  Z%=4:S%=0:B$="0":W%=0:L%=4
340  ENVELOPE1,2,0,0,0,0,0,0,126,-1,-1,
-1,89,80
350  ENVELOPE2,2,0,0,0,0,0,0,110,-1,-1,
-1,77,72
360  ENVELOPE3,2,0,0,0,0,0,0,126,-1,-1,
-2,80,71
370  ENDPROC
380
390
400
410  REM"      ***PLAY TUNE***
420
430  DEFPROC-1...

```

```

420
430 DEFPROCplay
440 REPEAT
450   FOR C%=1 TO 3
460     READ Nn$,Dur%
470     IFNn$="END" Dur%=0
480     SOUND 512+C%,C%,FNnote(Nn$)+W%,(
Dur%*Z%)
490   NEXT C%
500 UNTIL Nn$="END"
510 ENDPROC
520
530 REM"      ***CONVERT PITCH***
540
550 DEF FNnote(N$)
560 Name$=LEFT$(N$,1):Oct%=VAL(RIGHT$(
N$,1))
570 IF LEN(N$)=3 THEN B$=MID$(N$,2,1)
ELSE B$="0"
580 S%=0:IF B$="f" S%=1
590 IF B$="b" S%=-1
600 Note%=INSTR(K$,Name$)+S%-1
610 note%=Note%*4+(Oct%*48)

```

---

## 150 Applications

```

620 =note%
630
640
650 REM"      ***THE NOTES***
660
670
680 DATA E1,6,C1,6,G0,6,E1,2,C1,2,G0,2
690 DATA F1,2,C1,2,A0,2,F1,4,C1,4,A0,4

```

```

690 DATA F1,2,C1,2,A0,2,F1,4,C1,4,A0,4
,E1,2,C1,2,G0,2
700 DATA F1,3,C1,3,A0,3,E1,1,C1,1,G0,1
,D1,3,A0,3,F0,3,C1,1,A0,1,F0,1
710 DATA B1,8,G0,8,D0,8
720 DATA G1,4,D1,4,B1,4,F1,4,C1,4,A0,4
730 DATA E1,1,C1,1,G0,1,C1,1,G0,1,E0,1
,F1,1,C1,1,A0,1,D1,1,A0,1,F0,1,G1,2,D1,2
,B0,2,F1,2,C1,2,A0,2
740 DATA E1,4,C1,4,G0,4,D1,4,B0,4,G0,4
750 DATA C1,8,G0,8,C0,8
760 DATA END,0,END,0,END,0,0,0
770
780
790 REM"      ***DRAW FLAG***
800
810 DEF PROCflag(Y%)
820 VDU24,360;Y%-120;760;Y%+120;
830 VDU29,560;Y%;
840 VDU19,2,4;0;
850 GCOL0,130
860 CLG
870 GCOL0,2
880 G%=5
890 REPEAT
900   y%=120+120 DIV G%;z%=120-120 DIV
G%
910   IF G%=5 THEN GCOL0,7 ELSE GCOL0,1
920   I%=0
930   REPEAT
940     MOVE-200,-y%;MOVE-200,-z%;PLOT85
,200,z%

```



,200,z%

950 MOVE200,y%:PLOT85,-200,-z%

960 y%=-y%:z%=-z%

970 I%=I%+1

980 UNTIL I%=2

990 G%=G%+3

---

Applications 151

1000 UNTIL G%=11

1010 G%=3

1020 REPEAT

1030 IF G%=3 THEN GCOL0,135 ELSE GCOL  
0,129

1040 VDU24,-200 DIV G%,-120;200 DIV G  
%;120;

1050 CLG

1060 VDU24,-200;-120 DIV G%;200;120 D  
IV G%;

1070 CLG

1080 G%=G%+2

1090 UNTIL G%=7

1110 ENDPROC

1120

1130

1140 REM" \*\*\*DRAW POLE\*\*\*

1150

1160 DEF PROCpole

1170 COLOUR 3

1180 Q%=3

1190 PRINTTAB(5,2);"P"

1200 REPEAT

```

1170 PRINTMB(0,2); F
1200 REPEAT
1210  PRINTTAB(5,Q%); "Q"
1220  Q%=Q%+1
1230  UNTIL Q%=31
1240  ENDPROC

```

Now that you know what to expect we will look at how it works.

**Initialisation:** The musical part of the program is similar to “Happy Birthday”, listed earlier. In the initialisation procedure a string K\$=“BC\*D\*EF\*G\*A\*” is used to hold the note symbol information. It is not necessary to include sharps or flats at this stage, since that is taken care of when the length of the note symbol is tested in PROCplay. The “\*” is required to make the total string length up to 12. When the position of, say, G is then tested, using INSTR, the value 9 is returned (counting left to right from B).

K\$:	B	C	*	D	*	E	F	*	G	*	A	*
INSTR:	1	2	3	4	5	6	7	8	9	10	11	12

The ENVELOPES to be used are also defined in the initialisation PROC (in this case, one for each voice). Note that they are all slightly different, as this necessarily produces a more interesting sound. The VDU 23 statements

---

## 152 Applications

define the components used to make up the flagpole. This method is similar to the one we used to draw musical notes.

**Play Tune:** This is identical to the Play Tune PROCEDURE in “Happy Birthday”.

**Convert Pitch:** This is identical to the note% function PROCEDURE in the above mentioned program.

**The Notes:** The DATA is entered as pitch followed by duration for each channel before moving to the next note. Pitches are held as the musical symbol followed by the octave number. The duration number is decided

channel before moving to the next note. These are held as the duration symbol followed by the octave number. The duration number is decided by finding the smallest note length in the piece. In this version of the song the note equivalent to the word 'the' is the shortest note and would be written as a quaver (1/8th note). This is then made equal to one duration increment. This makes Rule=6, Bri=2, tan=2, ia=4, etc.

### RULE BRITANNIA



One useful tip which should be borne in mind when assembling musical DATA is to make each DATA statement a separate bar. If you remember to do this, it will prove much easier to track down rogue harmonies and similar less than endearing components and effects.

Three sets of "END" signifying DATA are included because we must allow the program to complete the channel FOR . . . NEXT loop before restarting the program. We could have flipped out of the loop as soon as the first "END" was detected, but this would eventually have caused the program to crash.

**Draw Flag:** The flag-drawing part of the program makes use of the multicolour MODE 2 graphics and the PLOT 85 triangle drawing facility. VDU 24 is used to define a graphics window exactly the size of the flag. VDU 29 gives the X and Y co-ordinates of the graphics origin. In this case we will make the origin the centre of the flag. These two lines, 820 and 830, give us control over the size and position of the flag. VDU 19 changes the default colour green (2) to blue (4). GCOL0,130 then selects blue (4) as the background (128+4=130) colour. Once into the REPEAT . . . UNTIL loop, the

ground (128+4=130) colour. Once into the REPEAT . . . UNTIL loop, the foreground white is selected and the diagonal pattern is drawn. This is then REPEATED in red, using narrower bands, the size of the bands being

controlled by G%. The larger G% is the smaller the bands. Next, you will see that the horizontal and vertical bands are drawn. These simply over-draw the diagonals rather than attempt the actual asymmetrical pattern necessary to reproduce the markings of a real Union Jack.

**Draw Pole:** The letters O and P were re-defined in the initialisation procedure as the two parts that make up the flagpole. (VDU 23, 79, . . . =O : VDU 23, 80, . . . =P). This PROCEDURE simply PRINTs the pole characters in the text part of the screen next to the flag.

**The Program:** All that is left for the program to do is call up the various PROCEDURES and reposition the flag at the end of each cycle of the tune. The variables W% and L% are used both to modulate the melody and to raise (or lower) the flag. L% equals the increment of increase or decrease, W% equals the current total of L%'s. Line 220 determines whether the tune and flag go up or down. This is achieved by testing for the maximum and minimum allowable values for W%.

The program could easily be modified to perform the relevant patriotic paraphernalia for any country, although you must remember that some flags will prove much more easy to draw than others!

## DRUM MACHINE

This is a fairly sophisticated drum machine program which allows you to tap rhythms in real time and hear them played back by the computer immediately. Four drums are supplied and can be played simultaneously and up to fifty patterns can first be stored and then chained together to form complex rhythm parts.

This program is not really self-explanatory, so let us begin by considering some instructions on how to operate it. You could use the

sidering some instructions on how to operate it. You could use the following notes to assemble your own 'help' page in the program if required. The program is modelled on a Linn drum-type drum computer in that there are two basic modes of operation. These are known as pattern edit and song chain. A third page, the Menu, allows you to move freely from one mode to another. When the program is RUN the first page you will see on the screen is the Menu.

This page allows you to select the pattern number, the tempo, edit mode or song mode. The selection is made by pressing one of the keys 'P', 'E', 'S' and 'T'. Default values of one for pattern and 120 for tempo are already selected. If you wish to move straight onto playing then you must simply press key 'E'.

The pattern edit mode allows you to tap your rhythms using the four drum keys to create a one-bar length pattern. The drum keys are laid out as follows:

---

#### 154 Applications

KEY:	Z	X	C	V
DRUM:	HiHat	Tom-Tom	Bass Drum	Snare

The position in the bar is shown by a moving \* on the screen. Once you have completed your first pattern, return to the Menu is accomplished by pressing 'M'.

Once a number of patterns have been completed you might decide that you want to hear them linked together. The song chain mode allows you to chain together all the patterns you have previously created into a song or drum solo. When selected, the song mode first asks if a new song is to be entered. If the answer is yes, you simply type in the order of patterns required. The chain is ended by typing END. If no is selected, the computer will play any chain which has been previously entered. When running, the current pattern number will be displayed. That is basically how the program works, so why don't you now type it in, RUN it and have some fun!

some fun!

[illegible]

```

112 T%=120:K%=1
120
130 ENVELOPE 1,1,-10,0,0,1,0,0,126,-81
,0,-29,120,100
140 ENVELOPE 2,3,-1,0,0,7,0,0,126,-81,
0,-13,120,100
150 ENVELOPE 3,128,-73,0,0,11,0,0,126,
-24,-32,-1,120,100
160 ENVELOPE 4,1,0,0,0,0,0,0,126,-1,0,
-9,120,100
170
180
190
191 REM "the whole program
192
195 REPEAT
197 IF A$="M" PROCmenu
210 IF A$="E" PROCset:PROCedit
215 IF A$="S" PROCplay
220
230
239 UNTIL FALSE
240 REM"      ££££PATTERN EDIT££££
250
260
270
280 DEF PROCedit
290 PRINTTAB(0,1);CHR$(134);CHR$(157);
CHR$(129);CHR$(141);"          EDIT MOD
E":PRINTTAB(0,2);CHR$(134);CHR$(157);CHR
$(129);CHR$(141);"          EDIT MODE"
300PRINTTAB(4,15);CHR$(141);CHR$(133);
K%:" "":PRINTTAB(4,16);CHR$(141);CHR$(13

```

```

300PRINTTAB(25,15);CHR$(141);CHR$(132);
K%;" " :PRINTTAB(4,16);CHR$(141);CHR$(13
3);K%;" "
310PRINTTAB(25,15);CHR$(141);CHR$(132)
;T%;" " :PRINTTAB(25,16);CHR$(141);CHR$(1
32);T%;" "
320 REPEAT :REM Uno loop
330 N%=1:REPEAT :REM Duo loop
340
350
360 IF X%(N%,1,K%)=1 THEN SOUND&11,
1,5,1

```

---

156 Applications

```

370 IF X%(N%,2,K%)=1 THEN SOUND&12,
2,36,3
380 IF X%(N%,3,K%)=1 THEN SOUND&13,
3,8,2
390 IF X%(N%,4,K%)=1 THEN SOUND&10,
4,4,2
400
410 PRINTTAB(N%*2-2,10)" *"
420 A$=INKEY$(0)
430 IF N%=1PRINTTAB(30,10)" "
440
450 IF A$="Z" THEN SOUND&11,1,5,1:X
%(N%,1,K%)=1
460 IF A$="X" THEN SOUND&12,2,36,3:
X%(N%,2,K%)=1
470 IF A$="C" THEN SOUND&13,3,8,2:X
%(N%,3,K%)=1
480 IF A$="V" THEN SOUND&10,4,4,2:X

```



```

480      IF A$="V" THEN SOUND&10,4,4,2:X
%(N%,4,K%)=1
490      TIME=0:REPEAT:UNTIL TIME>1300/T
%
500      N%=N%+1
510      UNTIL N%=17 OR A$="M"
520      UNTIL A$="M"
530 ENDPROC
540
550
560
570 REM"      ££££SET DISPLAY££££
580
590
600
610 DEF PROCset
620 CLS
630 PRINTTAB(1,13);CHR$(134);" Pattern
      Tempo"
640 O%=1:REPEAT:PRINTTAB(O%*2,11)"^":O
%=O%+1:UNTIL O%=17
650 PRINTTAB(1,9);CHR$(131);"1          2
      3          4  "
660 ENDPROC
670
680
690

```

```

700
710 REM"      ££££MENU££££
720
730
740
750 DEF PROCmenu
760 CLS
770 PRINTTAB(0,1);CHR$(131);CHR$(157);
CHR$(132);CHR$(141);"      MENU":
PRINTTAB(0,2);CHR$(131);CHR$(157);CHR$(1
32);CHR$(141);"      MENU"
780 PRINTTAB(1,13);CHR$(134);" Pattern
Tempo"
790 PRINTTAB(1,11)"TEMPO .....
.....";CHR$(131);"T"
800 PRINTTAB(1,5)"PATTERN .....
.....";CHR$(131);"P"
810 PRINTTAB(1,9)"SONG MODE .....
.....";CHR$(131);"S"
820 PRINTTAB(1,7)"EDIT MODE .....
.....";CHR$(131);"E"
830PRINTTAB(25,15);CHR$(141);CHR$(132)
;T%;" ":PRINTTAB(25,16);CHR$(141);CHR$(1
32);T%;" "
840PRINTTAB(4,15);CHR$(141);CHR$(133);
K%;" ":PRINTTAB(4,16);CHR$(141);CHR$(133
);K%;" "
850 REPEAT
860 A$=GET$
870 IF A$="T" THEN INPUTTAB(20,18)"Ent
er Tempo:"T%:PRINTTAB(25,15);CHR$(141);C
HR$(132);T%;" ".PRINTTAB(25,16);CHR$(141)

```

```

      880PRINTTAB(20,18) "
      900 IF A$="P" THEN INPUTTAB(3,18) "Enter Pattern:"
      K%:PRINTTAB(4,15);CHR$(141);CHR$(133);K%:" "
      ::PRINTTAB(4,16);CHR$(141);CHR$(133);K%:" "
      910PRINTTAB(3,18) "
      "
      920 UNTIL A$="E" OR A$="S"
      940 ENDPROC
      950
      960

```

---

## 158 Applications

```

      970
      980 REM"      ££££PLAY SONG££££
      990
     1000
     1010
     1020 DEF PROCplay
     1030 CLS
     1040 PRINTTAB(0,1);CHR$(132);CHR$(157);
CHR$(135);CHR$(141);"      SONG MOD
E"
     1042 PRINTTAB(0,2);CHR$(132);CHR$(157);
CHR$(135);CHR$(141);"      SONG MOD
E"
     1050 PROCplay

```

E"

1050 PROCchain

1060 PROCset

```
1070 PRINTTAB(0,1);CHR$(132);CHR$(157);
CHR$(135);CHR$(141);"          SONG MOD
E"
```

```
1072 PRINTTAB(0,2);CHR$(132);CHR$(157);
CHR$(135);CHR$(141);"          SONG MOD
E"
```

```
1080 PRINTTAB(25,15);CHR$(141);CHR$(132)
);T%;PRINTTAB(25,16);CHR$(141);CHR$(132)
;T%
```

```
1090 PRINTTAB(4,15);CHR$(141);CHR$(133)
;K%;" " ::PRINTTAB(4,16);CHR$(141);CHR$(1
33);K%;" "
```

1100 U%=1

1125 REPEAT :REM charlie loop

1130 IF S%(U%)=0 THEN U%=1

1140 K%=S%(U%)

```
1150 PRINTTAB(4,15);CHR$(141);CHR$(1
33);K%;" "TAB(4,16);CHR$(141);CHR$(133);
K%;" "
```

1160 N%=1

1170 REPEAT :REM delta loop

```
1180 IF X%(N%,1,K%)=1 THEN SOUND&11
,1,5,1
```

```
1190 IF X%(N%,2,K%)=1 THEN SOUND&12
,2,36,3
```

```
1200 IF X%(N%,3,K%)=1 THEN SOUND&13
,3,8,2
```

```
1210 IF X%(N%,4,K%)=1 THEN SOUND&10
,4,4,2
```

```

1220      TIME=0:REPEAT:UNTIL TIME>1300/
T%
1230      PRINTTAB(N%*2-2,10)"*"
1240      IF N%=1 PRINTTAB(30,10)"      "
1250      A$=INKEY$(0)
1260      N%=N%+1
1270      UNTIL N%=17 OR A$="M"
1280      U%=U%+1
1290      UNTIL A$="M"
1310 ENDPROC
1320 PROCmenu
1330
1340
1350 REM"      ££££CHAIN SONG££££
1360
1370
1380
1390 DEF PROCchain
1400 PRINTTAB(0,10);CHR$(134);"Do you w
ish to start a new chain? Y/N"
1410 A$=GET$
1420 PRINTTAB(0,10)"
      "
1430 IF A$="Y" THEN GOTO 1440 ELSE ENDP
ROC
1440 U%=0:REPEAT
1450 U%=U%+1
1460 INPUT"CHAIN: ";S%(U%)
1470 UNTIL S%(U%)=0
1480 ENDPROC

```

Up to Line 170 the program above sets up all the parameters which are going to be used. The array S% stores the song chain information, while the three-dimensional array X% contains the pattern step information for all four drums and fifty patterns.

The REPEAT . . . UNTIL loop starting at Line 195 controls the order of the PROCedures which we have to follow. Since the initial value of A\$ is "M" we start by going to PROCmenu.

PROCmenu: The bulk of the menu PROCedure is MODE 7 text display. CHR\$ codes are used to produce different colours and double height effects. If you turn to page 154 of the *User Guide* you will find a list of these codes and their effects. The GET\$ statement in Line 860 waits for one of the letters 'P', 'E', 'S', or 'T'. 'E' or 'S' cause the program to exit the PROCedure, and the program control loop will then send the program to

## 160 Applications

Edit mode or Song mode respectively. Typing 'T' prompts a change of T%, the tempo parameter. Typing 'P' prompts a change in K%, the pattern parameter.

PROCedit: Lines 290 to 310 print the page heading and the current tempo and pattern values. The main part of this PROCedure is contained within two REPEAT . . . UNTIL loops. The inside loop increments the step variable N% from 1 to 16. The outside loop resets N% to 1. Lines 360 to 390 SOUND the various drums if X%(N%, ChannelNo., K%)=1. On entry into the Edit mode, these values will be zero and will remain thus until a drum has been played. Lines 450 to 480 both SOUND a drum when it has been played and set X%(N%, ChannelNo., K%) to 1. A drum hit is detected by the INKEY\$ statement in Line 420. The PRINT statements in Lines 410 and 430 PRINT and unPRINT the moving display and Line 490 defines the tempo using T%.

PROCchain: This first provides you with the possibility of making a

**PROCchain:** This first provides you with the possibility of making a choice between old or new chains, using the GET\$ statement in Line 1410. Any answer apart from "Y" sends the program to PROCplay. A reply of "Y" causes a new chain to be started and the pattern numbers stored in the array S%(100). If S%(U%)=0 the loop will end. The BBC gives any unassigned variable a default value of zero. This is why typing END <RETURN> causes an exit from the loop. PROCchain is only called up from within PROCplay, so on ENDPROC we go to PROCplay.

**PROCplay:** Lines 1020 to 1090 are taken up with text display and calling PROCchain and PROCset. The remaining lines contain two loops. The inner loop increments the pattern step N% and the outer loop selects which pattern is to be played and resets U% when the end of the chain is reached. This causes the chain to cycle round until it can escape from the loop. Line 1130 resets U% and Line 1140 recalls the pattern number from the S%(100) array. The remainder of the PROCedure works in an entirely similar way to PROCedit. The INKEY\$ statement in Line 1250 waits for an "M" keypress before exiting the PROCedure and returning to the Menu.

**PROCset:** Sets up the graphics that are common to PROCedit and PROCplay.

There is plenty of room for development within this program. For example, the inclusion of a facility to use twelve steps, in addition to sixteen, would allow 3/4 rhythms to be entered. Alternatively, go the whole hog and make the program based on 48 steps. This would immediately make it possible for much more subtle rhythms to be played.

A pattern and drum delete facility would be quite easy to accomplish as a separate PROCedure. Similarly, a pattern copy and error correct (necessary with a 48 step version) would change the program so that it behaved exactly like a Linn or Oberheim drum machine. In addition to these changes, a choice of drum sound could also be offered as part of the menu.

## CHORD ORGAN

This program is ideal for those of you who like to sing but have no access to an instrument such as the piano or guitar. The Chord Organ makes it

to an instrument such as the piano or guitar. The Chord Organ makes it possible for you to accompany yourself with simple three-note chords at the touch of a button. The chords, generated from a scale of C Major, are selected using the bottom row of keys on the alphanumeric keyboard. The screen display shows which key will play which chord.

```

10 PRINTTAB(8,10); "Value: "; CHR$(134);
V%
20
30 REM"   ***CHORD ORGAN***
40
50 REM"   .....Set Up
60 MODE 7
70 *FX11,1
80 *FX12,1
90 VDU23;B202;0;0;0;
100 *KEY9 *FX12,0IM
110 PRINTTAB(2,10); "Press function key
f9 on Escape"
120 PRINTTAB(0,15); CHR$(132); "Key   :
Z   X   C   V   B   N   M"
130 PRINTTAB(0,17); CHR$(129); "Chord:
C   Dm  Em  F   G   Am  Bdim"
140 BORED=FALSE
150 DIM CHORD%(3,7)
160 Note$="ZXCVBNM"
170
180 FOR P%=1 TO 7
190   FOR C%=1 TO 3
200     READ W%:CHORD%(C%,P%)=W%*4+5
210   NEXT
220 NEXT
230
240 REM".....The Program
250 REPEAT

```



```

240 REM".....The Program
250 REPEAT
260   REPEAT
270     A$=GET$
280     P%=INSTR(Note$,A$)
290     *FX21,0
300   UNTIL P%>0 AND P%<8
310   FOR C%=1 TO 3

```

## 162 Applications

```

320     SOUND&10+C%,-10,CHORD%(C%,P%),1
330   NEXT
340 UNTIL BORED
350
360 REM".....Chord Information
370
380 DATA 12,16,19,14,17,21,16,19,23,12
,17,21,14,19,23,12,16,21,14,19,23

```

The program itself is totally straightforward. Lines 70 and 80 change the keyboard auto-repeat to its fastest setting. Line 90 switches off the cursor and Line 100 sets the user-programmable key 9 to reset the default auto-repeat values. This is a device which makes it unnecessary for you to have to remember to press a particular key when you decide that you want to end the program. Simply press <ESCAPE> followed by \*KEY 9 to reset normal operation. The graphics are contained in Lines 120 and 130. CHORD%(3,7) holds the three note information for all seven chords. Note\$ holds the key name information.

Lines 180 to 220 READ the note information into the CHORD% array. In this program the DATA is stored in MC-4 nomenclature so the conversion formula  $W\%*4+5$  is used. The main part of the program is contained in Lines 250 to 340. A\$=GET\$ waits for a keypress, and if P%=INSTR (Note\$,A\$) returns a value within the range 1 to 7 (i.e. a chord note has been pressed) the three note chord is played. If the key is held down the cycle REPEATS

returns a value within the range 1 to 7 (i.e. a chord note has been pressed) the three note chord is played. If the key is held down the cycle REPEATS itself UNTIL the key is released, when the program will hold at Line 270 as A\$=GET\$ waits for a new keypress. \*FX 21,0 in Line 290 flushes the keyboard buffer and, as a result, accordingly speeds up the program.

## THE SHEPARD SCALE

This is an intriguing little routine which plays a scale which apparently never ends.

```

10 REM      *** SHEPARD SCALE ***
20
30 REPEAT
40   FOR X=0 TO 11
50     SOUND 1,-11,X*4+48,10
60     SOUND 2,-X,X*4,10
70     SOUND 3,X-11,X*4+96,10
80   NEXT
90 UNTIL FALSE

```

Applications 163

The three SOUND statements play a chromatic scale which REPEATS itself – with a difference. SOUND 2 is an octave below SOUND 1 and increases in volume to match SOUND 1 at step 11. SOUND 3 is an octave above SOUND 1 and decreases in volume from matching SOUND 1 to zero at step 11. The resultant effect is reminiscent of a scale which is continually chasing its own tail. I have no idea how this works, but the overall effect is not unlike an aural version of one of those curious optical illusion drawings produced by M. C. Escher.

## IGOR TUNE

‘Igor Tune’ is a real time sequencer which was written by the famous  
 file:///C:/Documents%20and%20Settings/Chris%20Richardson/Desktop/New%20Folder/bbc\_mm\_09.htm (23 of 72)20/06/2012 08:47:04

'Igor Tune' is a real time sequencer which was written by the famous computer wizard Igor Thomas. The program allows you to play melodies from the keyboard, and promptly hear them played back at the tempo of your choice. You are also given the facility to save your masterpiece on tape or disc.

The program is Menu based so the various options can be chosen at the press of a key. Pressing 'N' (for New tune) selects the record mode which allows you to play your tune into the sequencer using the following keys:

W E T Y U O P [ : ]  
A S D F G H J K L ; : ]

Pressing 'M' stops the recording and returns you to the Menu. Pressing 'A' while in Menu mode lets you add notes to a previously recorded sequence, and 'S' selects SAVE the tune. When SAVEing it is necessary to supply a filename for the tune. LOAD also requires the INPUT of a filename and, in addition, a value for the length of file. This is the third figure along in the extended file information which you will see displayed at this point.

'Igor Tune' was originally designed to operate with an external keyboard connected to the analogue input port. This allowed the user to enter melodies on a real keyboard and hear them played back by the computer. I have included an optional tune-write PROCEDURE at the end of the listing for anyone who has access to a one volt per octave synthesiser with control and gate voltage outputs. If an external keyboard is used the control voltage should be connected to ADVAL 1 input and the gate to ADVAL 2. (See Chapter Eleven: Interfacing for the details of the A/D connections.)

```
100 REM *** IGOR TUNE"
110 REM
120 REM Seat-o'th'-pants sequencer
130 REM Curtesey of - Igor Thomas Esq.
140
```

## 164 Applications

```
150 MODE 7
160
170 REM start of buffer is B%
180 B% = &2800:REM at least TOP + 100
190 IF B% < TOP+100 VDU7:STOP
200
210 REM end of buffer is C%
220 C% = HIMEM - 100
230
240 REM pointer will be P% (and moves
250 REM in steps of 2)
260 REM *** P%=0 :REM zero pointer
270
280 REM initial playback factor
290 F = 1
300
310 REM flush flush flush
320 *FX 15
330
340 REM just use first 2 channels
350 *FX16,2
360
370 REM enable length display
380 *OPT 1,2
390
400 REPEAT
410 CLS
420 PRINTTAB(3,3);  "NNEW TOONE" "A
ADDITIONAL TOONE" "PPLAH TOONE" "SSAVE
TOONE" "LOAD TOONE" "MMENU/STOP"
```

```

ADDITIONAL TUNE      FFLR TUNE      WRITE
TOON" "LOAD TOON" "MENU/STOP"
430 G$=GET$
440 IF G$="N" PRINT NOO  ":P%=0:PROC
WRITE
450 IF G$="A" PRINT ADD  ":PROCWRITE
460 IF G$="P" AND P% PRINT PLAH":PRO
CREAD
470 IF G$="S" AND P% INPUT "FILENAME:
" F$:OSCLI"SAVE "+F$+" "+STR$~B%+" "+ST
R$~P%
480 IF G$="L" INPUT "FILENAME: " F$:OS
CLI"LOAD "+F$+" "+STR$~B%:INPUT "Length
&" L$:P%=EVAL("&"+L$)
490 SOUND 0,-9,6,4
500 UNTIL 0

```

Applications 165

```

510
520
530
540 DEFPROCWRITE
550 *FX11,1
560 *FX12,1
570 KEY$="AWSEDFTGYHUIJKOLP;:[]"
580
590 REM *** entry here ***
600 REM wait for a gate or a key
610 A$=GET$
620
630 REM quit if a key pressed. or

```

```
630 REM quit if a key pressed, or
640 REM if end of buffer reached
650 IF A$="M" B%!P%=0:P%=P%+2:PROCRESSE
T:ENDPROC
660 IF B%+P% > C% PROCRESET:ENDPROC
670
680 REM skip ahead if first note
690 IF P%=0 GOTO 760
700
710 REM save TIME, or 255, whichever
720 REM is less
730 IF TIME<255 B%!P%=TIME ELSE B%!P%=
255
740 P%=P%+1
750
760 REM *** entry here *** force
770 REM ch one to start converting
780 *FX17,1
790
800 REM wait for end of conversion,
810 REM then save value in V%
820 REPEAT
830 A$=GET$
840 V%=INSTR(KEY$,A$)
850 UNTIL V%>0
860
870 REM save V% in buffer (B%), and
880 REM advance pointer (P%), and
890 REM reset TIME to 0
900 B%!P%=V% :REM same as ?(B%+P%)=V%
910 P%=P%+1:TIME=0
930
```

930

## 166 Applications

```
940 REM sound the note (V%), until
950 REM it changes, or until no gate
960 REPEAT:SOUND17,-9,V%*4,-1:UNTIL IN
KEY$(2)<>A$ OR INKEY$(2)=""
970
980 REM save TIME, or 255, whichever
990 REM is less
1000 IF TIME<255 B%?P%=TIME ELSE B%?P%=
255
1010 P%=P%+1
1020
1030 REM if still gate, back up for
1040 REM more notes
1050 REM IF ADVAL2>5000 GOTO 720
1060
1070 REM shut off sound, and
1080 REM save some silence
1090 SOUND 17,0,0,0
1100 B%?P%=0:P%=P%+1:TIME=0
1110
1120 GOTO 590 : REM repeat
1130
1140
1150
1160 DEFPROC READ
1170
1180 REM change factor
1190 PRINT "Factor now ":F": change to
```

```
1180 REM change factor
1190 PRINT "Factor now ";F"; change to
? ";:INPUT "" F$:IF F$<>"" F=EVALF$
1200
1210 REM play it all back
1220 FOR X% = B% TO B%+P% STEP 2
1240
1250 REM decide first
1260 IF ?X% GOTO 1340 :REM jmp if not 0
1270
1280 REM the sound of silence
1290 TIME=0:REPEAT
1300 SOUND 17,0,0,0
1310 UNTIL TIME*F > X%?1
1320 GOTO 1390
1330
1340 REM the sound of music
1350 TIME=0:REPEAT
```

---

Applications 167

```
1360 SOUND 17,-9,?X% *4,-1
1370 UNTIL TIME*F > X%?1
1380
1390 REM keep a'goin'
1400 NEXT
1410 ENDPROC
1420
1430
1470
1480
1490 DEF PROCRESET
1500 *FX 12,0
```



```
1500 *FX 12,0
1510 *FX15,0
1520 ENDPROC
```

### Optional Keyboard Routine:

```
5000
5010 DEFPROCWRITE
5020
5030 REM *** entry here ***
5040 REM wait for a gate or a key
5050 REPEAT:K%=INKEY0:UNTIL(30<ADVAL2DI
V1024) OR K%<>-1
5060
5070 REM quit if a key pressed, or
5080 REM if end of buffer reached
5090 IF K%<>-1 B%!P%=0:P%=P%+2:ENDPROC
5100 IF B%+P% > C% ENDPROC
5110
5120 REM skip ahead if first note
5130 IF P%=0 GOTO 5200
5140
5150 REM save TIME, or 255, whichever
5160 REM is less
5170 IF TIME<255 B%?P%=TIME ELSE B%?P%=
255
5180 P%=P%+1
5190
5200 REM *** entry here *** force
5210 REM ch one to start converting
5220 *FX17,1
5230
5240 REM wait for end of conversion,
```

## 168 Applications

```
5250 REM then save value in V%
5260 REPEATUNTILADVALODIV256=1:V%=ADVAL
1DIV1024
5270
5280 REM save V% in buffer (B%), and
5290 REM advance pointer (P%), and
5300 REM reset TIME to 0
5310 B%?P%=V% :REM same as ?(B%+P%)=V%
5320 P%=P%+1:TIME=0
5330 PROCDISPLAY(P%)
5340
5350 REM sound the note (V%), until
5360 REM it changes, or until no gate
5370 REPEAT:SOUND17,-9,V%*4,-1:UNTIL AD
VAL1DIV1024<>V% OR ADVAL2<5000
5380
5390 REM save TIME, or 255, whichever
5400 REM is less
5410 IF TIME<255 B%?P%=TIME ELSE B%?P%=
255
5420 P%=P%+1
5430
5440 REM if still gate, back up for
5450 REM more notes
5460 IF ADVAL2>5000 GOTO 5200
5470
5480 REM shut off sound, and
5490 REM save some silence
5500 SOUND 17,0,0,0
5510 B%?P%=0:P%=P%+1:TIME=0
5520
```

```

5520
5530 GOTO 5030 : REM repeat
5540
5550 ENDPROC
5560

```

This program uses the indirection operator ? to read and write direct to the computer memory. Lines 170 to 230 define the memory area to be used and P% acts as pointer to where we are within the memory at any given time. F is the tempo control factor and Lines 310 to 350 only apply to the analogue interface version of the program. \*OPT 1,2 specifies display of the extended file information.

The control part of 'Igor Tune' is contained in a REPEAT . . . UNTIL loop starting at Line 400. Pressing either 'N' or 'A' sends the program to

---

Applications 169

whichever version of PROCWRITE is being used at the time. Specifying P%=0 starts the recording at the beginning of the memory space. Not specifying P%=0 adds the new information to whatever was there previously. P selects PROCREAD, the tune play procedure. The entire SAVE and LOAD routines are stored in Lines 470 and 480. Line 490 causes a sound to be played on the completion of any routine included in the Menu.

PROCWRITE at Line 540 uses the computer keyboard to input musical information. Lines 550 and 560 set the auto-repeat to its fastest setting. KEY\$ is used to identify the active (valid) keys. The program waits at the GET\$ statement for a keypress before continuing.

The routine in Lines 820 to 850 tests the keypress for validity against the list of keys in KEY\$. If a valid entry is made the position of the entry within KEY\$ is recorded in V% and also recorded in the memory at Line 900. Line 910 advances the pointer P% one place in the memory, ready to store the duration information (measured by TIME which is zeroised at this point). Line 960 SOUNDS the note and also tests for either a different keypress or a key release <>V% or "".

keypress or a key release  $\langle \rangle V\%$  or  $\langle \rangle$ .

Once released from this loop the duration is recorded into the memory at Line 1000. This line also tests for excessively long durations. Any duration longer than 2.55 seconds is reduced to 2.55 seconds. Duration information can therefore always be held in 1 byte of memory. Line 1010 advances the pointer another place, and the remainder of PROCWRITE silences the SOUND statement if a key is not being pressed.

PROCREAD at Line 1190 requests INPUT for the tempo factor. Entering 2 would double the tempo, 4 would quadruple the tempo, etc. The FOR . . . NEXT loop in Line 1220 goes from the X% value of the start of the memory segment at B% to the current end value given by P%. Line 1260 tests for information at memory location X%. If there is no information the silence routine (Lines 1280 to 1320) is called. If there is information present, this routine is omitted and the sound is played. The pitch is taken from location X% using ?X% and the duration from location X%+1 with X%?1. At this point the loop repeats until all the values of pitch and duration up to location P% have been played.

PROCRESET simply resets the default key repeat settings.

PROCWRITE: The second PROCWRITE section is intended for use with an external keyboard, and is listed starting at Line 5000. If it is typed separately from the main program it can be added to the other code using the following merging procedure.

Type and save both programs. Load the larger of the two (in this case the main program) and type: PRINT~TOP-2 <RETURN>. This will produce a hexadecimal number, which is the top of the program address minus two (number). Load the remaining program using: \*LOAD programname number <RETURN>. List the resulting merged program. If there is a line number clash RENUMBER the program. (In our example this is not the case).

Line 5050 waits for a gate (ADVAL 2) to start the program. The program acts exactly as before up till Line 5260, where a pitch value is received from ADVAL1 instead of from the computer keyboard. This is saved in the

acts exactly as before up till Line 5260, where a pitch value is received from ADVAL 1 instead of from the computer keyboard. This is saved in the buffer just as before. Line 5370 tests for a new input through ADVAL 1 or no gate ( $\text{ADVAL } 2 < 5000$ ). In Line 5460 the program is sent back to Line 5200 if a gate is still being detected at ADVAL 2. If not, the sound is shut off as before by SOUND 17,0,0,0.

Various improvements could be made to 'Igor Tune', including the incorporation of edit commands and a more accurate duration measurement. Note, however, that the technique of reading and writing information directly to and from the memory is highly space saving and thus extremely useful. Long sequences can be saved in this way, and extra channels can also be included.

## CLYPSO 2

"Clypso 2" is based on the autocomposition programs listed earlier. As well as improvising a melody using a calypso-type style, it also accompanies itself with the appropriate backing. A reggae influence creeps in with the introduction of so-called 'dub' sections, where various combinations of instruments drop in and out for different periods of time. A multi-colour screen is also included.

No operating instructions are required, just type the program in, RUN and have a carnival!

```

10
20
30 REM      **** CLYPSO2 **** 26FEB
50
60
70 MODE7:VDU23;B202;0;0;0;
71 DIM DUB(3):T%=0
80 PROCPICTURE
90 ENVELOPE 1,1,0,48,0,0,0,0,60,-1,0,
-120,126,100

```

```

-120,126,100
  100 ENVELOPE2,1,0,0,0,0,0,0,32,-1,-1,-
4,0,0
  110 ENVELOPE3,1,0,0,0,0,0,0,126,-1,-2,
-1,100,40
  120 ENVELOPE4,1,0,0,0,0,0,0,120,-10,-2
,-9,110,5
  130 DIM c%(8,4),C%(3,4),B%(4),b%(4)
  140 FOR N=1 TO 4:FOR n=1 TO 8:READ c%(
n,N):NEXT:NEXT

```

Applications 171

```

  150 FORN=1 TO 4: FOR n=1 TO 3:READ C%(
n,N):NEXT:NEXT
  160 FOR N=1 TO 4:READ B%(N):NEXT
  170 FOR N=1 TO 4:READ b%(N):NEXT
  180 pitch2%=0:R%=0:r%=0
  190
  200
  220
  230 REPEAT
  232 T%=NOT T%:IF T% FOR L%=0TO3:DUB(L%
)=RND(6):NEXT
  240 FOR L%=1 TO 4
  250 FOR N%=1 TO 4
  260 G%=R%
  270 R%=RND(3)
  280 IF R%=G% THEN GOTO 270
  290 pitch1%=C%(R%,L%)
  300 IF pitch1%=pitch2% THEN GOTO 270
  310 S%=RND(6):IFS%=3 THEN E%=2 ELSE E%

```

```

310 S%=RND(6):IFS%=3 THEN E%=2 ELSE E%
=1
320 IF N%=1 OR N%=3 THEN V%=2 ELSE V%=
3
330 IF DUB(0)=DUB(1) SOUND&300,0,0,0 E
LSE SOUND&300,4,4,5
340 IF DUB(0)=DUB(2) SOUND&302,0,0,0:S
OUND&303,0,0,0 ELSE SOUND&302,V%,B%(L%),
5:SOUND&303,V%,b%(L%),5
360 IF DUB(0)=DUB(3) SOUND &301,0,0,0
ELSE SOUND&301,E%,pitch1%,5
370 q%=r%
380 r%=RND(8)
385 q%=RND(24)*40
386 ?(&7C00+q%)=128+RND(7):?(&7C01+q%)
=136+RND(15)
390 IF r%=q% THEN GOTO 380
400 pitch2%=c%(r%,L%)
410 IF pitch2%=pitch1% OR pitch2%-pitc
h1%>29 OR pitch2%-pitch1%<-29 THEN GOTO
380
420 S%=RND(6):IFS%=3 THEN E%=2 ELSE E%
=1
430 IF DUB(0)=DUB(1) SOUND&300,0,0,0 E
LSE SOUND&300,0,4,5

```

---

172 Applications

```

440 IF DUB(0)=DUB(2) SOUND&302,0,0,0:S
OUND&303,0,0,0 ELSE SOUND&302,3,B%(L%),5
:SOUND&303,3,b%(L%),5
460 IF DUB(0)=DUB(3) SOUND&301,0,0,0 E
LSE SOUND&301,E%,pitch2%,5
470 NEXT

```

```
470 NEXT
480 NEXT
490 UNTIL FALSE
500
510
530
540 REM The scales and chords.....
550
560 DATA 53,61,69,81,89,97,53,101
570 DATA 73,81,89,101,109,121,137,73
580 DATA 81,89,97,109,117,129,81,69
590 DATA 53,101,81,97,53,61,101,117
600 DATA 53,69,81
610 DATA 73,89,101
620 DATA 81,97,109
630 DATA 101,117,129
640 DATA 33,53,61,81
650 DATA 5,25,33,53
655
656
660 DEF PROCPICTURE
670 x=0
680 FOR Y=2 TO 20 STEP2
690 x=x+2
700 FOR X=0 TO x
710 PRINTTAB(0,Y)CHR$(132)TAB(X,Y)CHR$(
(141)" EVERYBODY DANCE "TAB(0,Y+1)CHR$(1
32)TAB(X,Y+1)CHR$(141)" EVERYBODY DANCE
"
730 NEXT:X=0
740 ENDPROC
```

Lines 70 to 190 set up the various parameters, DIM statements, ENVELOPES and variables. Array c%(8,4) contains scale information, C%(3,4) arpeggio information and n% and h% the backing notes. In the case of "Clunso 2"



and variables. Array `c%(8,4)` contains scale information, `C%(3,4)` arpeggio information and `B%` and `b%` the backing notes. In the case of “Clypsso 2” the various combinations of notes were chosen largely by a process of trial and error.

The main part of the program is contained within the `REPEAT . . . UNTIL` loop which starts at Line 230. A different set of DUB conditions is set up by

Applications 173

Line 232 every second cycle. The `L% FOR . . . NEXT` loop controls the four-bar chord sequence and `N%` steps the program forward in beat increments. The variable `R%` chooses the on-beat note information from the arpeggio array. Line 400 stops repeated notes. The various `SOUND` statements may be muted depending on the DUB conditions in Lines 330 to 360. The second half of the main program mirrors the first half but chooses `Pitch2%` from the scale arrays. Channel one contains the melody, two and three the chords and channel zero the rhythm.





Line 386 `POKEs` colour information directly onto the screen and this causes the ever changing display. `PROCPICTURE` sets up a simple text display.

## SEQUENCER

“Sequencer” is a sophisticated one channel compositional tool based on the Roland MC-4 and MC-202 type of numerical musical input. When `RUN` the program displays a Menu which allows you to select either pitch record, duration record, step select or play modes. In pitch mode, notes are entered in standard synthesiser nomenclature, where C=12, 24, 36 etc. Durations are also entered numerically. In this case 48 gives a crotchet, 96 a minim, 12 a semiquaver. The choice of 48 as a time base value means that subtle timings such as swing and shuffles are possible. Swing, for instance, would be written with duration values as follows:

Time Base = 48



Time Base = 48				
	24	24	24	24
	36	12	36	12
	32	16	32	16
Swing Time:				
	34	14	34	14

Default values of 24 are set for the durations at the start of the program. The default starting point for all modes is step one. Step selection allows you to play or record from any point in the piece. This is invaluable for editing purposes. Various tempos can also be selected. These are roughly equivalent to beats per minute. Q for Quit should be pressed to exit the record modes and return to the menu.

## 174 Applications

```

10MODE7
20
30
40 REM"*****
50 REM"*****SEQUENCER*****
60 REM"*****
70
80
90 MODE7
100 VDU23;8202;0;0;0;
110 DIM Pitch%(500).Dur%(500)

```

```
110 DIM Pitch%(500),Dur%(500)
120 FOR N%=1 TO 500:Dur%(N%)=24:NEXT
130 ENVELOPE 1,1,0,0,0,0,0,0,126,-1,0,
-1,126,110
140 N%=1
150 T%=100
160
170
180 PROCmenu
190
200
210 DEF PROCPitch
220 CLS
230 PRINTTAB(9,5);"RECORD PITCH"
240 REPEAT
250 PROCdisplay
260 INPUTTAB(14,15),Pitch%(N%)
270 SOUND1,-10,Pitch%(N%)*4,10
280 N%=N%+1
290 UNTIL Pitch%(N%-1)=0
300 ENDPROC
310
320
330
340 DEF PROCDuration
350 CLS
360 PRINTTAB(7,5);"RECORD DURATION"
370 REPEAT
380 PROCdisplay
390 INPUTTAB(26,15),D%
400 IF D%>0 Dur%(N%)=D%
```

```
400 IF D%>0 Dur%(N%)=D%
410 N%=N%+1
420 UNTIL Pitch%(N%-1)=0 AND N%>1
430 ENDFPROC
```

---

Applications 175

```
440
450
460
470 DEF PROCmenu
480 REPEAT
490 REPEAT
500 CLS
510 PRINTTAB(12,2); "*****MENU*****"
520 PRINTTAB(0,5); "R for RECORD"; ' ; "
P to PLAY"; ' ; "D for DURATIONS"; ' ; " S fo
r STEP"; ' ; "T for TEMPO"
530 PRINTTAB(0,12) "Enter Pitch as whol
e numbers where C=12 and D=14 etc. Octav
e C's would be 12, 24, 36, 48 etc..."
540 PRINTTAB(0,16) "Duration are entere
d where a crotchet=48, a quaver=24 etc..
."
550 PRINTTAB(0,19) "Press Q to Quit any
mode."
560 M$=GET$
570 IF M$="R" PROCPitch
580 IF M$="D" PROCDuration
590 IF M$="P" PROCPlay
600 IF M$="T" INPUTTAB(4,21); " INPUT
```

```
600 IF M$="T" INPUTTAB(4,21);" INPUT  
NEW TEMPO";T%  
610 N%=1  
620 UNTIL M$="S"  
630 PROCstep  
640 M$=""  
650 UNTIL FALSE  
660 ENDPROC  
670  
680  
690  
700 DEF PROCPlay  
710 CLS  
720 PRINTTAB(9,5);"***PLAY**"  
730 REPEAT  
740 IF Pitch%(N%)<>0 SOUND&11,1,Pitch%  
(N%)*4,-1:PROCdisplay ELSE PRINTTAB(14,1  
5)"END";TAB(27,15);"END"  
750 FOR W%=1 TO Dur%(N%)*T%/2:NEXT  
760 N%=N%+1  
770 UNTIL Pitch%(N%-1)=0
```

---

176 Applications

```
780 SOUND&11,0,0,0  
790 ENDPROC  
800  
810  
820 DEF PROCstep  
830 CLS  
840 PROCdisplay
```

```

840 PROCdisplay
850 INPUTTAB(2,15),N%
860 ENDPROC
870
880
890 DEF PROCdisplay
900 PRINTTAB(0,13); " Step           Pitch
    Duration"
910 PRINTTAB(2,15); " ";N%; " ";TAB(14,
15); " ";Pitch%(N%); " ";TAB(26,15); " ";D
ur%(N%); " "
920 ENDPROC

```

The program first initialises the two one-dimensional arrays to contain pitch and duration values. Line 120 loads the duration array with the default values of 24. N% is the step number and T% the tempo value.

PROCmenu: This displays instructions and waits at Line 560 for a valid keypress. Tempo is changed at Line 600 using a simple INPUT statement. Pressing 'S' sends the program to PROCstep, where the starting note can be chosen. If 'S' is not pressed the program defaults to a N% value of one.

PROCPitch: Pitches are entered using an INPUT statement. This is TABbed so that a ? appears in front of the current pitch value. If 'Q' is entered the BBC reads this as a Pitch% value of zero, so the program leaves the loop at Line 290.

PROCDuration: PROCDuration acts in the same way as PROCPitch. The variable D% is used in this case to allow <RETURN> to be pressed if a duration is not to be altered, so that if D%=0 or no value is entered Dur% remains the same. The program automatically returns to the Menu when it runs out of Pitch% values.

PROCplay: Line 740 tests for zero values in the Pitch% array. If Pitch%(N%)<>0 then a SOUND is produced. In this play procedure durations do not rely on the SOUND statement's D values. A FOR...NEXT loop is used to make possible a much finer timing resolution than the 1/20th of a second which is possible when using D. The REPEAT...UNTIL loop once

second which is possible when using D. The REPEAT . . . UNTIL loop once again tests for zero values of Pitch% before returning the program to the Menu

PROCstep sets the starting N% value and PROCdisplay displays the current step, pitch and duration values whenever it is called.

This program could well form the basis of a multi-channel composer, incorporating more sophisticated editing procedures. One useful addition to the program would be a bar counter, since this would make it possible for long sequences to be copied or deleted in bar length chunks. The synchronisation method explored in Chapter Seven could also be utilised to synchronisation channels on playback. All of the programs in this chapter can be developed or combined with other programs and techniques contained in the book. Your imagination is the only limit to the many possibilities available to you. The next program is a development of “Microcomposer” that suited my purposes.

## ADVANCED SEQUENCER

In its earlier incarnation the sequencer allowed numeric input of pitches and durations. This system is retained. What has been added is a more sophisticated editing system allowing deletion, insertion, rests and easy stepping through the sequence. These additions to the program are my particular preferences, and other enhancements are possible, depending on your own requirements. The increased sophistication of the sequencer makes a fairly lengthy explanation of its use inevitable. Before explaining how the Advanced Sequencer program works, therefore, I will describe its use in creating music.

As with the commercial microcomposers mentioned above the Advanced Sequencer has two main modes, Play and Edit. These functions are selected from a menu which is displayed on running the program:

SECRET

[illegible]

The options of Pitch Record, Play, Duration edit, Step and Tempo are each selected by pressing a single letter. Pitch Record and Duration will be explained below when we deal with editing. Tempo asks for INPUT of the playback tempo in beats per minute. A default value of 120 bpm is automatically selected on RUNNING the program.

The Step option has no function until some pitches have been entered. It then allows you to enter a particular step for editing either pitch or duration.

## PLAY MODE

**In Play mode the entire sequence is replayed and repeated until <SPACE> is pressed. No new information can be added in this mode and it hence can be thought of as safe storage. The display consists of the step, pitch and duration values, synchronised with the playback. The current playback tempo is also displayed, as beats per minute. To change this value it is necessary to return to the Menu page.**



BBC Micro Music Masterclass: Applications  
tempo is also displayed, as beats per minute. To change this value it is necessary to return to the Menu page.

EDIT MODE

The Edit mode consists of pitch recording and duration editing. The pitch record mode is the 'master' editing mode from which insertions and deletions can be made and rests entered. The screen display is like this:



Step	Pitch	Duration
1	?12	24

Press 'f0' to END  
Press 'f9' for Forward  
Press 'f8' for Backward  
Press 'f1' for a Rest  
Press 'f2' to Insert a note  
Press 'f3' to Delete a note

Pitch entry is indicated by a question mark in front of the pitch value. To enter a pitch, a value is typed and <RETURN> pressed. Pitch values related to note symbols can be found in the following table:

		OCTAVE			
NOTE	0	1	2	3	4
C	-	12	24	36	48
C #	1	13	25	37	49
D	2	14	26	38	50

C	1	13	25	31	49
D	2	14	26	38	50
D#	3	15	27	39	51
E	4	16	28	40	52
F	5	17	29	41	53
F#	6	18	30	42	54
G	7	19	31	43	55
G#	8	20	32	44	56
A	9	21	33	45	57
A#	10	22	34	46	58
B	11	23	35	47	59

The function keys are programmed for the following tasks:

f0 to return to the menu.

f9 to step forward through the sequence.

f8 to step backward through the sequence. You are automatically returned to the menu if you try to step forward past the last entry or backwards past the initial entry.

f1 enters a rest instead of a note. You are not required to type return.

f2 allows you to insert a note of pitch and duration equal to the previous note in the sequence. This default value can then be changed to your required value for pitch and duration simply by entering the new values as normal.

f3 lets you delete a chosen note, i.e. the currently displayed note arrived at using the forward and back keys.

In similar fashion duration entry is indicated by a question mark in front of the duration value in the display. Duration values are calculated using a time base of 48 (i.e. a crotchet=48 increments). As mentioned earlier, the use of 48 as a time base makes the entry of 'swing feel' quavers possible, thus:

Straight quavers: 24 24 24 24 . . .  
 Swing quavers: 28 20 28 20 . . .

Deletion, insertion and rest placement cannot be performed while in the

Deletion, insertion and rest placement cannot be performed while in the duration edit mode. These functions only work in the pitch mode. The remaining function keys for ending, editing and stepping backward and forward operate as normal, however.

The following data, when entered into the sequencer, will play an Irish Reel. A suitable value for Tempo would be around 150.

---

## 180 Applications

PITCH : 31 36 31 28 24 28 31 36 38 36 35 33 31 36 35 36 38 36 38 40  
DURATION: 16

PITCH : 38 36 35 33 31 36 31 28 24 28 31 36 38 36 35 33 31 36 35 36  
DURATION: 16

PITCH : 38 36 38 40 36 36 36 38 40 41 40 40 41 43 38 40 38 38 35 36  
DURATION: 16 16 16 16 16 16 32 16 16 16 16 16 16 16 16 16 32 16 16

PITCH : 35 36 33 38 36 35 31 31 31 31 33 29 33 33 35 36 31 33 31 31  
DURATION: 16 16 16 16 16 16 16 16 32 16 16 16 16 16 16 16 16 16 16

PITCH : 33 35 36 35 36 38 36 38 40 36 36 36  
DURATION: 16 16 16 16 16 16 16 16 16 16 16 32

Note that the Reel is in 6/8 time, so a quaver triplet would be 1/3 of a beat in length, i.e. a duration equal to 16.

That about covers the use of the Advanced Sequencer program. Below, we'll take a look at how the program actually works. First the program itself:

```
10
20
30 REM"*****
40 REM"****ADVANCED ****
50 REM"****SEQUENCER****
```

```

40  REM *****ADVANCED *****
50  REM"*****SEQUENCER*****
60  REM"*****
70
80  REM"Copyright.....Ian Ritchie"
90
100  ON ERROR GOTO 280
110  *KEY10 OLDIM RUNIM
120  *KEY0 EIM
130  *KEY9 FIM
140  *KEY8 BIM
150  *KEY1 101IM
160  *KEY2 IIM
170  *KEY3 DIM
180  MODE7
190  VDU23;8202;0;0;0;
200  DIM Pitch%(500),Dur%(500)
210  FOR N%=1 TO 500:Dur%(N%)=24:NEXT
220  ENVELOPE 1,1,0,0,0,0,0,0,126,-1,0
,-1,120,110

```

Applications 181

```

230  ENVELOPE 2,1,0,0,0,0,0,0,0,0,0,0,0,0,
0,0
240  N%=1
250  T%=37:TEMPO%=4440/T%
260
270
280  PROCmenu
290
300
310  DEF PROCPitch
320  CLS
330  PRINTTAB(0,5);CHR$(157);CHR$(135)

```

```

330 PRINTTAB(0,5);CHR$(157);CHR$(135)
;CHR$(129);"          RECORD PITCH"
340 PRINTTAB(4,18);CHR$(133)"Press 'f
0' to END";TAB(4,19);CHR$(133);"Press 'f
9' for Forward";TAB(4,20);CHR$(133);"Pre
ss 'f8' for Backward"
350 PRINTTAB(4,21);CHR$(133);"Press 'f
1' for a Rest";TAB(4,22);CHR$(133);"Pres
s 'f2' to Insert a note";TAB(4,23);CHR$(
133);"Press 'f3' to Delete a note"
360 REPEAT
370   PROCdisplay
380   INPUTTAB(14,15),Pitch$;"      "
390   IF VAL Pitch$>0 Pitch%(N%)=VAL
Pitch$
400   IF Pitch$="B" N%=N%-2
410   IF Pitch$="I" PROCINSERT
420   IF Pitch$="D" PROCDELETE
430   IF N%<0 N%=0
440   SOUND1,-10,Pitch%(N%)*4,6
450   N%=N%+1
460   UNTIL Pitch%(N%-1)=0 OR Pitch$=
"E"
470 ENDPROC
480
490
500
510 DEF PROCDuration
520 CLS
530 PRINTTAB(0,5);CHR$(129);CHR$(157)
;CHR$(134);"          RECORD DURATION"

```

```

540  PRINTTAB(4,20);;CHR$(133)"Press '
f0' to END";TAB(4,21);CHR$(133);"Press '
f9' for Forward";TAB(4,22);CHR$(133);"Pr
ess 'f8' for Backward"
550  REPEAT
560      PROCdisplay
570      INPUTTAB(26,15),D$;"  "
580      IF VAL D$>0 Dur%(N%)=VAL D$
590      IF D$="B" N%=N%-2
600      SOUND1,-10,Pitch%(N%)*4,6
610      N%=N%+1
620      UNTIL Pitch%(N%-1)=0 OR D$="E"
630  ENDPROC
640
650
660
670  DEF PROCmenu
680  REPEAT
690      REPEAT
700          CLS
710          PRINTTAB(8,2);CHR$(130);"ADVA
NCED SEQUENCER"
720          PRINTTAB(10,4);CHR$(141);"***
*";CHR$(130);"MENU"CHR$(135);"*****"TAB(1
0,5);CHR$(141);"*****";CHR$(130);"MENU"CH
R$(135);"*****"
730          PRINTTAB(0,10);CHR$(129)"R...
.....PITCH RECORD";';CHR$(131)
;"F.....PLAY";';CHR
$(134);"D.....DURATIONS"
-'"C.....STEP":':

```

```

$(134); "D.....";
; ; " S.....STEP"; ;
CHR$(133); "T.....TEM
PO"

```

```

740      K=INKEY20
750      M$=GET$
760      IF M$="R" PROCpitch
770      IF M$="D" PROCDuration
780      IF M$="P" PROCPlay
790      IF M$="T" INPUTTAB(4,21); "
INPUT NEW TEMPO"; TEMPO%; IF TEMPO%>0 T%=4
440/TEMPO%
800      N%=1
810      UNTIL M$="S"
820      PROCstep

```

Applications 183

```

830      M$=" "
840      UNTIL FALSE
850      ENDPROC
860
870
880
890      DEF PROCPlay
900      CLS
910      PRINTTAB(30,2); CHR$(133); "TEMPO";
TAB(32,3); TEMPO%; " "
920      PRINTTAB(9,5); CHR$(141); CHR$(132)
; "****"; CHR$(131); "PLAY"; CHR$(132); "****"; TA
B(9,6); CHR$(141); ; CHR$(132); "****"; CHR$(13
1); "PLAY"; CHR$(132); "****"
930      PRINTTAB(4,20); CHR$(132); "Press":

```

```
930 PRINTTAB(4,20);CHR$(132);"Press";
CHR$(135);" <space>";CHR$(132);" to STOP
"

940 REPEAT
950     N%=1
960     REPEAT
970         IF Pitch%(N%)=101 E%=2:C%=&10
11 ELSE E%=1:C%=&11
980         IF Pitch%(N%)<>0 SOUND C%,E%,P
itch%(N%)*4,-1:PROCdisplay
990         FOR W%=1 TO Dur%(N%)*T%:NEXT
1000         N%=N%+1
1010         UNTIL Pitch%(N%)=0 OR INKEY(-
99)<>0
1020     UNTIL INKEY(-99)<>0
1030     *FX15,0
1040     SOUND&11,0,0,0
1050     ENDPROC
1060
1070
1080 DEF PROCstep
1090     CLS
1100     PROCdisplay
1110     INPUTTAB(2,15),N%
1120     ENDPROC
1130
1140
1150 DEF PROCdisplay
1160     PRINTTAB(0,13);" Step           ";CHR
$(129);"Pitch           ";CHR$(134);"Duration"
```



## 184 Applications

```
1170 IF Pitch%(N%)<>101 PRINTTAB(2,15)
; " ";N%; " ";CHR$(129);TAB(14,15); " ";Pit
ch%(N%); " ";CHR$(134);TAB(26,15); " ";
Dur%(N%); " " ELSE PRINTTAB(2,15); " ";N%;
" ";CHR$(129);TAB(13,15); " REST";TAB(2
6,15); " ";Dur%(N%); " "
1180 ENDFPROC
1190
1200
1210 DEF PROCINSERT
1220 FOR IN%=499 TO N% STEP -1
1230 Pitch%(IN%+1)=Pitch%(IN%)
1240 Dur%(IN%+1)=Dur%(IN%)
1250 NEXT
1260 ENDFPROC
1270
1280
1290 DEF PROCDELETE
1300 FOR IN%=N% TO 499
1310 Pitch%(IN%)=Pitch%(IN%+1)
1320 Dur%(IN%)=Dur%(IN%+1)
1330 NEXT
1340 ENDFPROC
```

In the program Lines 100 to 170 are devoted to assigning values to the various function keys to be used in edit mode. Line 190 switches off the text cursor. Pitch and duration arrays are dimensioned in Line 200 and Line 210 sets a default value of 24 for each element of the duration array. Finally two ENVELOPES are defined, one on (Line 220) and one off (Line 230). Line 250 sets a default tempo of about 120 bpm.

The main section of the program is accessed by Line 280, which calls

The main section of the program is accessed by Line 280, which calls PROCmenu, from which all the other PROCedures are called as required.

**PROCmenu:** The first part of the menu PROCedure (Lines 700–730) takes care of the menu graphics. The different colours used make for a pleasing and easy to use display. The remainder of PROCmenu uses the string variable M\$ to determine which PROCedure is to be called – R for PROCPitch, D for PROCDuration, etc.

**PROCPitch:** The main part of PROCPitch is held within a REPEAT . . . UNTIL loop. PROCdisplay is called to update the step, pitch and duration values, and Line 380 prompts a pitch entry with a question mark. The pitch value is entered as a string rather than a numeric variable in order that letters can be used to call PROCedures for functions such as insert and delete. The function keys have previously been programmed to input these letters, and so will call a new PROCedure with a single key entry. Line 390

---

Applications 185

determines whether the input is a letter (VAL Pitch\$=0) or a number (VAL Pitch\$>0x. If the entry is a number it is written into the pitch values array, Pitch%(N%).

Line 440 plays the entered pitch. A factor of 4 is required to convert to BBC pitch nomenclature. The loop will REPEAT UNTIL the current pitch value is zero or “E” is entered for Pitch\$ (function key f0).

**PROCDuration:** This PROCedure is in essence the same as PROCPitch. No editing functions can be called, but line 590 allows back stepping and forward stepping can be accomplished by pressing any key other than the numerals, ‘B’ and ‘E’. (It is convenient, however, to continue to use f9 for forward and f8 for backward, as with PROCPitch.)

**PROCPlay:** PROCPlay is contained within two loops, so that the sequence automatically repeats. Line 970 chooses which ENVELOPE is to be used in the SOUND statement. If Pitch%(N%)=101 a rest is selected by choosing ENVELOPE 2. The extended channel parameter is made equal to &1011 allowing the previous note’s release phase to complete. In all other cases ENVELOPE 1 is selected and the note is SOUNDED. Line 980 ensures that a

ENVELOPE 1 is selected and the note is SOUNDED. Line 980 ensures that a note is only played if a note has been entered, i.e. when  $\text{Pitch\%(N\%)} < > 0$ .

A FOR . . . NEXT loop is used to control duration in Line 990. PROCPlay is exited by pressing the space bar. This is tested, using INKEY (-99), in lines 1010 and 1020.

PROCstep simply sets N%, the step number variable, and so allows editing of a particular note in the sequence without stepping through from the beginning.

PROCdisplay controls the variable display.

PROCINSERT looks after insertions by copying  $\text{Pitch\%(N\%)}$  into  $\text{Pitch\%(N\%+1)}$  for each value of N% from the current value up to 499. The same is done for duration, and so an extra entry at position N% is created. This can then be changed to the required inserted value.

PROCDELETE handles deletion by using exactly the opposite technique to PROCINSERT. For each step the N%+1 values are copied into the N% array positions, thus removing one step.

That completes the description of my Advanced Sequencer program. Many further modifications could obviously be made but I find the above program sufficiently comprehensive for most uses. I hope you find the same! Our final program introduces a further level of complexity – Assembly language.

In dealing with all the aspects of music production on the Beeb thus far, only BASIC has been used, since this makes the principles of music programming accessible to all users. Any committed assembly language programmers will no doubt be able to put what they've learned through the BASIC examples to good use. However, there is one area of music programming where machine code has to be used, which is when we need background music that will continue to play no matter what else is going

on in a program, which in this context is almost certainly a game! This is made possible by using an interrupt-driven routine such as our next program provides. It is useable as it stands, but the description will be of

program provides. It is useable as it stands, but the description will be of little use to anyone entirely ignorant of assembly language. The BASIC programmer should look at the BASIC part of the program, since some interesting techniques are used, and resolve to get to grips with assembler someday!

All computers 'interrupt' whatever foreground process (running a program, listing it to a printer, etc.) is in hand at intervals to scan the keyboard and perform other housekeeping tasks. The current state of affairs is stored, and the computer jumps to the start of the housekeeping routine. This jump can be made to a machine language program entered by the user, which is then executed. This program sends control back to the normal interrupt sequence on completion, and the computer continues with its housekeeping, returning to the foreground process when this is done. As a general description, this is true of all home micros, but the wonderful Beeb has a facility which allows the user to easily create timed 'Events' which are ideal for our purposes. Using the interval timer, which is a five byte value incrementing every one-hundredth of a second, and setting it to appropriate values, we can make an 'Event' occur at a programmed interval.

When the interval timer value crosses 0 (i.e. passes from 0 to 1) a system event occurs, if enabled by \*FX14,5, or the OSBYTE equivalent, as in the program. The program sets the timer to -10, so that it produces this event every tenth of a second, when control passes to the vector address stored at locations &220,&221, which points to the start of the music routine. This works by continuously updating the three sound channel buffers, entering a new note if the buffer queue is not full (using OSBYTE with A=128). The channels are synchronised at the start of the tune with an extended SOUND command, using a parameter block (OSWORD with A=7). Since the accumulator, on entry to the event handling routine, contains an event number value indicating the cause of the event, this can be used to differentiate between a programmed interval timer event and one caused by the pressing of the <ESCAPE> key, provided the <ESCAPE> event is enabled. This is done in the program, and on detection of an ESCAPE event, the tune is re-started, since <ESCAPE> flushes the SOUND buffers. The

the tune is re-started, since <ESCAPE> flushes the SOUND buffers. The same thing is done when the tune has finished, the synchronisation notes being placed in the buffers and the tune started again. The program may be assembled to any convenient location in memory, and as given here is installed below a reset PAGE boundary, starting at &1200. Non-disc users should change the PAGE values in Line 10 to &1100, setting the program variable equal to &E00 in Line 60.

Applications 187

```

10
20 ENVELOPE 1,1,0,0,0,0,0,0,6,0,-10,-
5,110,0
30 ENVELOPE 2,1,0,0,0,0,0,0,121,-1,-2
,-2,100,80
40 ENVELOPE 3,9,0,0,0,0,0,0,121,-10,-
5,-2,120,80
50
60 DIM program 256+768
70 pitch=&70
80 duration=&72
90 index=&74
100 channel=&77
110 par_block=&78
120
130
140 OSBYTE=&FFF4
150 OSWORD=&FFF1
160
170 FOR pass=0 TO 2 STEP 2
180   P%=program
190   [ OPT pass
200
210 \ *****

```

```

200
210 \ *****
*****
220 \ *
*
230 \ * INITIALISE MUSIC ROUTINE
*
240 \ *
*
250 \ *****
*****
260
270 .initialise \ Initialise
music routine and events.
280
290 lda #event MOD 256 \ Set up even
t vector to point to music
300 sta &220 \ routine.
310 lda #event DIV 256
320 sta &221
330

```

---

188 Applications

```

340 lda #14 \ Enable inte
rval timer crossing
350 ldx #5 \ zero event.
360 jsr OSBYTE
370
380 lda #14 \ Enable ESCA
PE pressed event.
390 ldx #6
400 jsr OSBYTE
410

```

```

400      jsr OSBYTE
410
420      .init0                                \ Initialise
variables and buffers.
430      lda #128
440      sta index+0                            \ Index into
channel 1's notes.
450      sta index+1                            \ Index into
channel 2's notes.
460      sta index+2                            \ Index into
channel 3's notes.
470
480      ldx #4                                \ Flush sound
buffers.
490      lda #21
500      .flush
510      jsr OSBYTE
520      inx
530      cpx #8
540      bne flush
550
560      jsr init_clock                        \ Set interval
1 timer.
570
580      rts                                  \ End of initialisation.
590
600      \ *****
*****
610      \ *
*
620      \ *      INTERRUPT DRIVE MUSIC ROUT
TMC

```

```

620 \ * INTERRUPT DRIVE MUSIC ROUT
INE *
630 \ * Execute this routine every t
ime *

```

Applications 189

```

640 \ * interval timer crosses zero.
    *
650 \ *
    *
660 \ *****
*****
670
680 .event \ Come here w
hen interval interval
690 php \ timer cross
es 0 or ESCAPE key pressed.
700 pha \ Save regist
ers on stack.
710 tya
720 pha
730 txa
740 pha
750
760 tsx
770 lda &103,X \ Find event
type.
780 cmp #5 \ Check for i

```



```

780    cmp #5                \ Check for i
interval timer event.
790    beq e0                \ Branch if t
imer event occurred,
800    jsr init0             \ otherwise,
ESCAPE must have been.
810    lda #125              \ Therefore,
clear sound buffers etc,
820    jsr OSBYTE           \ and set ESC
APE condition.
830    jmp return           \ Return.
840
850    .e0                  \ Come here i
f interval timer caused event.
860    jsr init_clock       \ Reset inter
val timer.
870
880    lda #data MOD 256    \ Point to ch
annel 1 data.
890    sta pitch
900    lda #data DIV 256
910    sta pitch+1
920    lda #(data+128) MOD 256

```

---

## 190 Applications

```

930    sta duration
940    lda #(data+128) DIV 256
950    sta duration+1
960
970    .s0
980    lda #1                \ Set sound c

```

980	lda #1	\ Set sound c
channel	number.	
990	sta channel	
1000		
1010	.loop	\ For sound c
channels	1,2 and 3.	
1020	sec	\ See if soun
d	buffer for channel	
1030	lda #251	\ is full by
performing	an OSBYTE	
1040	sbc channel	\ call A=128,
X	determines chanel.	
1050	tax	
1060	lda #128	
1070	jsr OSBYTE	
1080	txa	\ X contains
number	of free spaces	
1090	bne insert	\ in buffer.
Branch	if space.	
1100		
1110	.next_channel	\ Update soun
d	queue for next channel.	
1120	inc pitch+1	\ Point to no
te	data for next channel.	
1130	inc duration+1	
1140	inc channel	\ Next channe
l.		
1150	lda channel	\ Check chann
el	number.	
1160	cmp #4	\ Continue if
more	channels.	

```

more channels.
1170    bne loop
1180
1190    .return                                \ End of rout
ine.
1200    pla                                    \ Restore reg
isters.
1210    tax
1220    pla

```

---

Applications 191

```

1230    tay
1240    pla
1250    plp
1260    rts                                \ Return from
interrupt routine.
1270
1280    .insert                                \ Insert note
into buffer.
1290    lda #0                                \ Clear param
eter block.
1300    sta par_block+1                        \ Clear msb o
f channel.
1310    sta par_block+3                        \ Clear msb o
f amplitude.
1320    sta par_block+5                        \ Clear msb o
f pitch.
1330    sta par_block+7                        \ Clear msb o
f duration.
1340
1350    ldy channel                            \ Get current

```

```

1350    ldx channel          \ Get current
channel.
1360    stx par_block+0     \ Store chann
el number.
1370    stx par_block+2     \ Store envel
ope number.
1380
1390    dex
1400    ldy index,X        \ Get index i
nto note list.
1410    bmi end_list       \ Force start
of tune?
1420    lda (duration),Y   \ Get duratio
n of note.
1430    beq end_list       \ Branch if n
o more notes.
1440    sta par_block+6     \ Store durat
ion.
1450    lda (pitch),Y      \ Get pitch o
f note.
1460    sta par_block+4    \ Store pitch
.
1470    bne okay          \ If pitch=0,
then note is a rest,

```

```

1480    sta par_block+2    \ set volume
of channel to 0.

```

of channel to 0.

1490

1500 .okay

\ Parameter b

lock set up now.

1510 clc

\ Increment c

channel index to point

1520 lda index,X

\ to next note

e.

1530 adc #1

1540 sta index,X

1550

1560 .do\_sound

\ Insert

t note by calling OSWORD

1570 ldx #par\_block MOD 256

\ with

A = 7.

1580 ldy #par\_block DIV 256

1590 lda #7

1600 jsr OSWORD

1610 jmp next\_channel

\ Go onto next

t channel.

1620

1630 .end\_list

\ Start tune

again for that channel.

1640 lda #2

\ Synchronize

to the other two channels by

1650 sta par\_block+1

\ inserting a

dummy note with sync bits set.

1660 lda #0

\ Set index b

ack to first note.

1670 sta index,X

1680 sta par\_block+2

\ Set volume

of sync note to 0.

1690 lda #100

\ Set duration

of sync note to 0.

```

1690    lda #100                \ Set duration
n of silent sync note to 5 sec.
1700    sta par_block+6
1710    jmp do_sound           \ Insert sync
note into channel's buffer.
1720
1730
1740    .init_clock             \ Set
interval timer to interrupt
1750    lda #4                 \ afte
r one tenth of a second.

```

Applications 193

```

1760    idx #timer_value MOD 256
1770    ldy #timer_value DIV 256
1780    jsr OSWORD
1790    rts
1800
1810    .timer_value           \ Table conta
ining the 5 byte value
1820    EQUB &F6              \ for setting
the interval timer to
1830    EQUB &FF              \ reach 0 in
one tenth of a second.
1840    EQUB &FF
1850    EQUB &FF
1860    EQUB &FF
1870
1880    .data                  \ Data for
tune goes here.
1890
1900    ]

```

```
1900    J
1910    NEXT
1920
1930    tempo%=1
1940    PITCH%=data
1950    DURATION%=data+128
1960    PROC_read_voice(1)
1970    PROC_crab(2,length%)
1980    PROC_crab(3,length%)
1990    PRINT "PLAY"
2000    CALL initialise
2010    END
2020
2030    DEF PROC_read_voice(voice%)
2040    offset%=256*(voice%-1)
2050    REM RESTORE 30000+(voice%-1)*1000
2060    READ length%
2070    FOR note%=0 TO length%-1
2080        READ pitch$, duration$
2090        PITCH%?(offset%+note%)=FN_pitch(
pitch$)
2100        DURATION%?(offset%+note%)=FN_dur
ation(duration$)
2110    NEXT
2120    DURATION%?(offset%+length%)=0
2130    ENDPROC
```

```
2140
2150    DEF PROC_crab(voice%,length%)
2160    offset%=256*(voice%-1)
```

```
2160 offset%=256*(voice%-1)
2170 FOR note%=0 TO length%-1
2180   PITCH%?(offset%+length%-1-note%)
=PITCH%?note%
2190   DURATION%?(offset%+length%-1-note%)=DURATION%?note%
2200   NEXT
2210 DURATION%?(offset%+length%)=0
2220 ENDPROC
2230
2240 DEF PROC_silent_voice(voice%)
2250 offset%=256*(voice%-1)
2260 DURATION%?offset%=0
2270 ENDPROC
2280
2290 DEF FN_pitch(note$)
2300 octave%=0
2310 octave$="C D EF G A B"
2320 pitch%=0
2330 IF INSTR(note$,"#") pitch%=4
2340 IF INSTR(note$,"b") pitch%=-4
2350 FOR p%=2 TO LEN(note$)
2360   IF MID$(note$,p%,1)="*" octave%=
octave%+1
2370   NEXT
2380 pitch%=pitch%+octave%*48
2390 p%=INSTR(octave$,LEFT$(note$,1))
2400 IF p%=0 pitch%=0 ELSE pitch%=pitch
%+1+4*p%
2410 =pitch%
2420
2430 DEF FN_duration(note$)
2440 range$="tseqhw"
```



```

2440 range$="tseqhw"
2450 duration%=INSTR(range$,RIGHT$(note
$,1))
2460 duration%=2^(duration%-1)*tempo%
2470 IF INSTR(note$,":") duration%=dura
tion%*1.5
2480 =duration%
2490
2500 DATA 90
2510

```

Applications 195

```

2520 DATA C*,h,E*b,h,G*,h,A*b,h,B,h,R,q
,G*,h,F*#,h,F*,h,E*,h,E*b,h,D*,q,D*b,q
2530 DATA C*,q,B,q,G,q,C*,q,F*,q,E*b,h,
D*,h,C*,h,E*b,h,G*,e,F*,e,G*,e,C**,e,G*,
e,E*b,e,D*,e,E*b,e,F*,e,G*,e,A*,e,B*,e,C
**,e
2540 DATA E*b,e,F*,e,G*,e,A*b,e,D*,e,E*
b,e,F*,e,G*,e,F*,e,E*b,e,D*,e,E*b,e,F*,e
2550 DATA G*,e,A*b,e,B*b,e,A*b,e,G*,e,F
*,e,G*,e,A*b,e,B*b,e,C**,e,D**b,e,B*b,e,
A*b,e,G*,e,A*,e,B*,e,C**,e,D**,e,E**b,e,
C**,e
2560 DATA B*b,e,A*b,e,B*,e,C**,e,D**,e,
E**b,e,F**,e,D**,e,G*,e,D**,e,C**,e,D**,
e
2570 DATA E**b,e,F**,e,E**b,e,D**,e,C**
,e,B*,e,C**,q,G*,q,E*b,q,C*,q

```

The program sets ENVELOPES from BASIC, and defines the variables for zero page locations used. The initialisation section of the assembler sets

THE PROGRAM SETS ENVELOPES FROM BASIC, and defines the variables for zero page locations used. The initialisation section of the assembler sets the event vector to point to the music routine, and enables the interval timer and ESCAPE events. The index counters for each channels' notes are set up, and the buffers flushed. The main program is a loop, checking each channel in turn to see if the buffer is full. If not, the next note is inserted. We have seen this sequence executed in BASIC earlier in the book.

The DATA for the tune, which in this case is Bach's Crab Canon, is stored in string form, and decoded by FN\_pitch and FN\_duration. These use the INSTR function to derive the numeric values needed for the pitch and duration values. Asterisks give the octaves, b and # are used for flats and sharps, note\$ has the note values, whilst R, signifying a rest, will not be found in octave\$, so p% will be 0 in Line 2390, and this will set pitch% to 0 in Line 2400. Duration is set according to the position in range\$ of the letter defining the note value (w for whole note, h for half note, etc.).

The functions are used by PROC\_read\_voice, and the decoded values are placed, using the indirection operator ?, in the music routines' data store. 256 bytes are used for each channel, and a maximum of 128 notes can be stored for each channel, as one byte holds the pitch value (0 being taken as a rest), and one the duration (0 marking the end of the list of notes for that channel). The data is held like this:

## Channel 1

INDEX	PITCH	:DURATION	
	0	127:128	255

## DATA

## Channel 2

INDEX+1	PITCH	:DURATION	
	256	.	511

**INDEX+1****PITCH**  
256**:DURATION**  
:

511

**Channel 3****INDEX+2****PITCH**  
512**:DURATION**  
:

767

The index values hold the position of the next note within the data block.

If inserting new tunes into the data statements, note that Line 2500 must hold the number of notes for each voice. The peculiarity of the Crab Cannon, which is a sort of musical palindrome, reading the same back-to-front, is the reason for the inclusion of PROC\_crab, which takes the Voice 1 data directly from memory and re-inserts it. For a normal tune, two more sets of DATA, each starting with the number of notes in the section, are required, and PROC\_read\_voice must be called for each voice. PROC\_silent\_voice is not used by the program as it stands, but will allow you to program a tune for only two voices, calling this procedure for the silent third voice.

That completes my selection of musical applications. You should have learnt enough by now to be able to execute your own musical desires, or at least have a good idea of how to expand (and improve!) some of the programs presented here. Next, we'll have a look at the dedicated processors taking over the music world . . .

# 10 Computers in Modern Music

Until fairly recently, if you hoped to play music of reasonable standard, you had to resign yourself to years of arduous practise on a saxophone, guitar or, worse still, violin. Ah, you may argue, surely this wasn't true of the new wave bands? Yes, even during the punk phase of pop music bands still had to get together for hours on end, even if they were simply deciding in what order to play the songs at their next gig. Computers have completely changed this state of affairs. Modern musicians now find themselves able to create technically sophisticated music in their own bedrooms by using a bewildering array of drum machines, synthesisers and sequencers that would put the BBC Radiophonic Workshop to shame.

All the above mentioned machines are basically microcomputers which are dedicated to the particular task in hand, such as producing drum beats or digitally synthesising sounds. We can learn a great deal about the best ways of programming sound on the BBC Micro from examining these devices, so in this chapter I would like to describe how some of them operate.

## Drum Machines

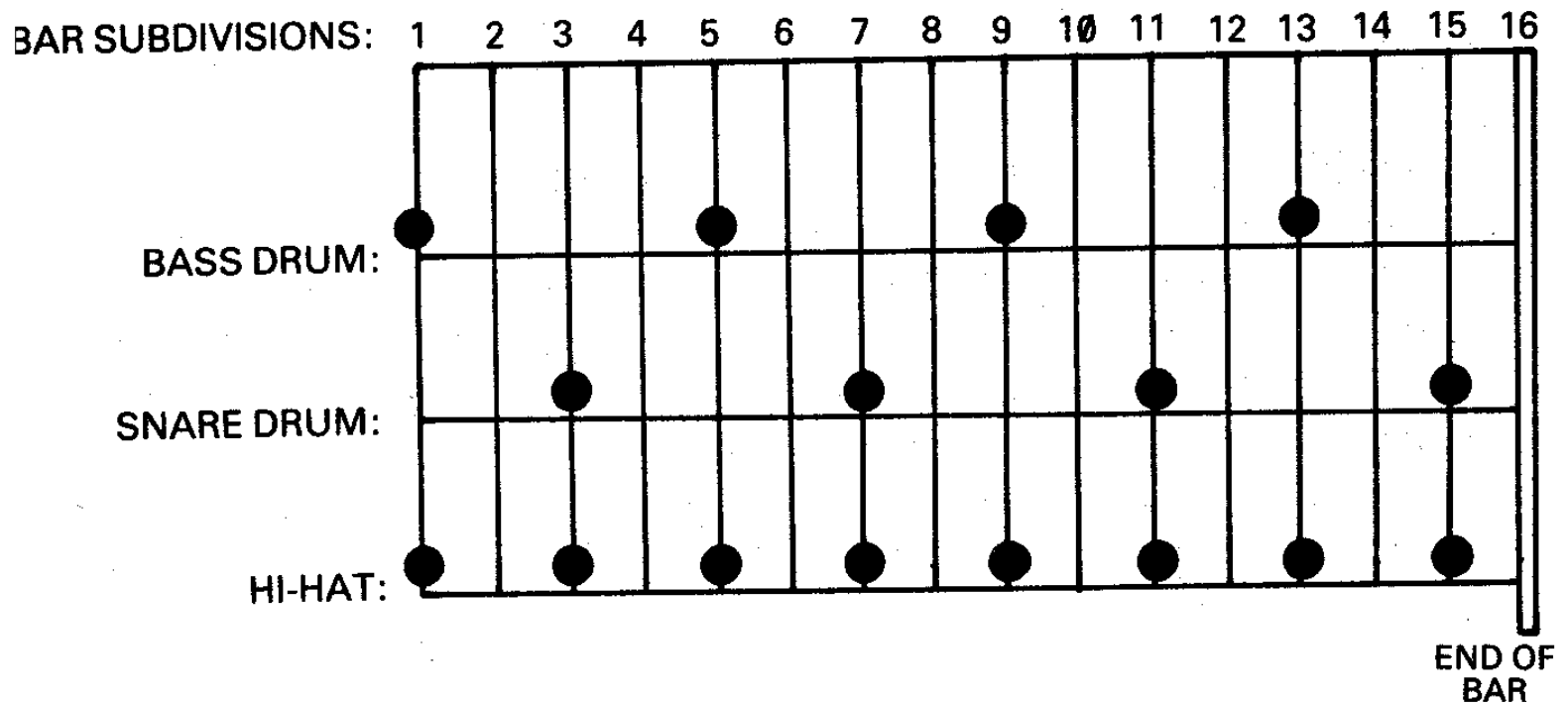
Computers are ideally suited to act as substitute drummers. They are accurate, unimaginative, stupid and do not mind undertaking boring and repetitive tasks for hours on end. It is not surprising then, that when computers began to infiltrate music one of the first tasks they were set was to act as a drum machine.

The original drum boxes that came attached to home organs contained a small selection of fairly standard and uninteresting pre-programmed rhythms and sounds. This was obviously an unsatisfactory state of affairs and everyone soon realised that it would be much better if a rhythm was programmed by the musician to fit the particular tune being played at any given time. The problem with this scenario was to make the programming simple enough for machine-shy musicians, a not inconsiderable percentage of the music community both then and now. One of the first drum boxes devised to solve this problem was called the Roland Dr Rhythm.

This machine was cheap and straightforward enough for even the most recalcitrant musician to feel at home with and approach optimistically.

The sounds were produced by pre-set noise producing circuitry, but the small on-board computer allowed almost limitless variations of rhythm to be tapped in by the user. On the Dr Rhythm each bar of music is simply divided into sixteen subdivisions for 4/4 time or twelve subdivisions for 3/4 time. The rhythm for each drum (bass drum, snare, sidestick and accent) is tapped in rather like sending morse code, using one button for a drum hit and another for a rest. Once a bar of rhythm has been completed for a particular drum it can then be checked against the inbuilt eighths or sixteenths hi-hat before

proceeding to another drum. The diagram below shows how we would produce a pattern of on beat brass drum and off beat snare drum against hi-hat eighths:



This method of breaking down a bar into subdivisions is fundamental to all drum machine programming. After the introduction and success of Dr Rhythm, drum boxes which incorporated larger memories and more facilities soon became generally available. The Roland Drumatix and TR808 machines were amongst the most popular and had a wider selection of drum sounds to their credit, as well as the ability to chain together bar length patterns. This chaining technique allows complex drum parts to be programmed, complete with fills, tempo and time changes.

However, looked at objectively, one sees that these machines are only superficially more sophisticated than the Dr Rhythm. To gain a substantial improvement the number of subdivisions per bar must increase. In the Dr Rhythm and its Soundmaster counterpart semiquavers are the shortest playable notes. In 4/4 time it would thus not be possible to play triplets (16/3). This limitation could be overcome by increasing the number of subdivisions to forty-eight per bar. A machine which could subdivide bars this finely could not only play semiquavers (since a semiquaver is 1/16 of a bar, and therefore would equal  $48/16=3$  divisions) but also quaver triplets (quaver triplets occupy 1/12 of a bar and therefore would occupy  $48/12=4$  divisions). The first machine to reach and make available this level of sophistication was Roger Linn's LM-1.

## Digital Drum Computers

To call a Linn Drum (or the Oberheim, MXR, etc., counterpart) a drum box would be rather like referring to a Stradivarius violin as a fiddle. We are in a completely new ball game here, for these machines are very far removed from the more primitive devices we have been looking at to date. They

differ in two main areas. Firstly let us look at the sounds. Linn sounds are digital recordings of real drums which are captured in Rows (Read Only Memory). D/A (Digital to Analogue) converters are then used to turn this digital information back into noises.

Secondly, in the programming area, all these machines are capable of far more subdivisions per bar than, say, the Tp808. In Linn terminology we talk about correcting to the nearest 1/16, 1/32 or 1/48th of a bar or recording in real time. This is because all these computers have abandoned the morse code method of the Dr Rhythm, for tap buttons which allow you to play the drum rhythms into the machine. Needless to say, this gives a much more spontaneous feel to the music produced by this new and exciting generation of machines.

In addition to pattern entry we are also able to choose a song mode which allows easy construction of complex arrangements. As the sophistication and size of memory increases, so editing facilities also have to be refined accordingly. All Digital Drum computers allow the deletion, insertion and copying of both overall and individual drum patterns. At least half of the pop records which are currently being aired on the radio are made by using computer drums of one kind or another. For a simple application of some of these principles, see the 'Drum Machine' program in Chapter Nine.

Drum machines are by no means the only musical area into which computers are insinuating their technology, merits and effects. In ten short years, for example synthesisers have changed from being gangling giants which took up half a room to the sleek, keyboard size instruments which we now find in virtually every well-equipped studio.

## Monophonic Synthesisers

The early synthesiser developed by Robert Moog and used by Walter Carlos to produce his famous Bach recordings looks quite unlike the instruments we see Depeche Mode and other contemporary bands play today. This is because the original Moogs were built as a series of modules, connected together in different ways to produce the various effects. These modules had the following functions:

**VCO:** the Voltage Controlled Oscillator produced the actual sound in the form of a sine, square (like the BBC) or sawtooth wave.

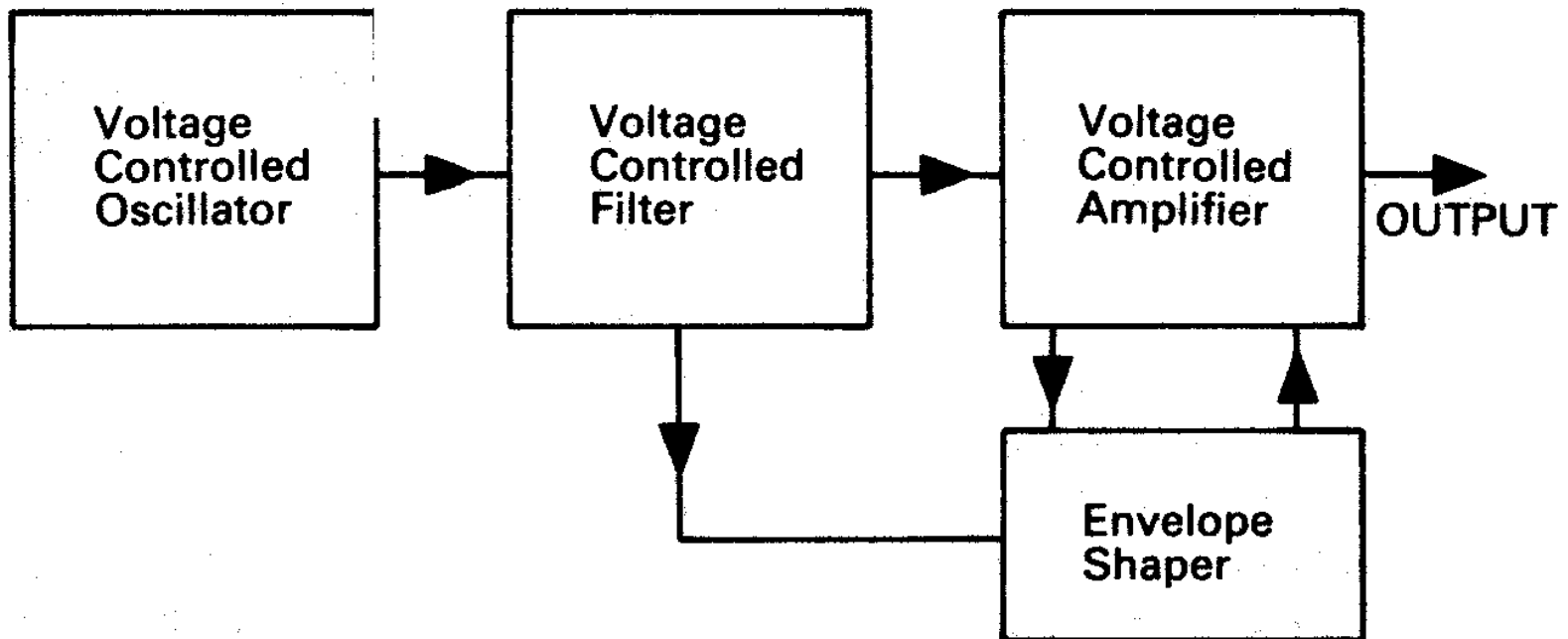
**VCF:** the Voltage Controlled Filter alters the nature of the basic sound by filtering out certain frequencies.

**ENVELOPE:** which acts in a similar way to the BBC's volume envelope.

**KEYBOARD:** the keyboard in these early instrument had to provide two sets of voltages. The control voltage (CV) controls the pitch of the note played by supplying set voltages to the VCO. A standard of one volt per octave is now widely used such that a CV of one volt gives a pitch of C. The gate voltage controls the duration for which the note is played. This voltage was normally zero for no note, and 2.5

volts or more for a keypress.

Of course, various other modules were often included, for instance to introduce vibrato into the sound using a low frequency oscillator (LFO) output or to generate white noise, but the above VCO, VCF and ENVELOPE generator are all that is needed, when amplified, to produce the familiar synthesiser sound we have come to know and love.

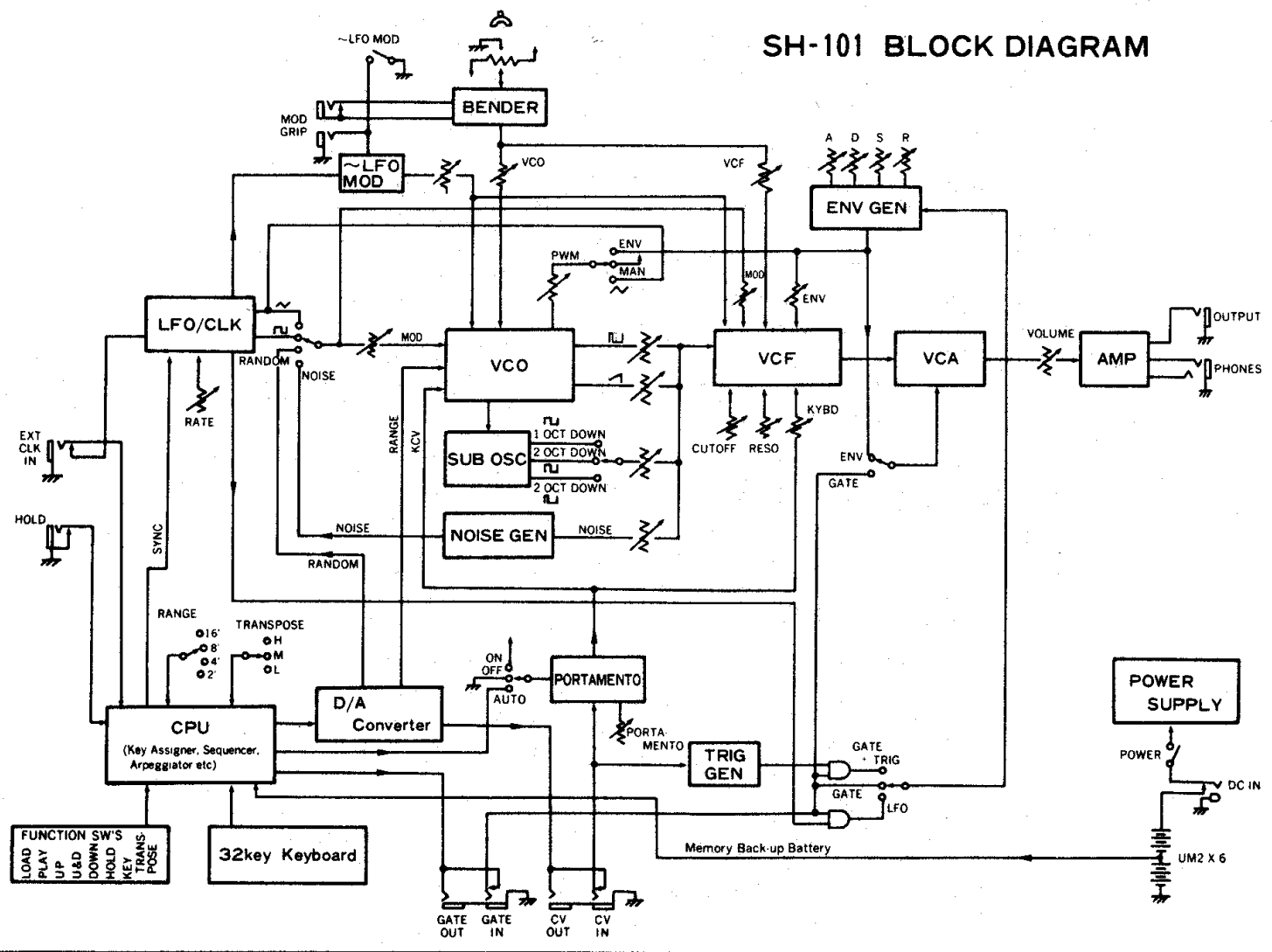


Voltage Voltage Voltage Controlled Controlled Controlled Oscillator Filter Amplifier OUTPUT

## Envelope Shaper

A modern synthesiser still includes the traditional components but instead of connecting together these elements with patch leads, a microprocessor is used to achieve the same effect. This computer's function varies according to the machine. Thus, it can merely be used to connect the keyboard to the various modules, as in the case of monophonic synths, such as the Sequential Circuits PRO ONE or the Roland SH-101, or it can be utilised to memorise entire sets of control settings, as with the polyphonic Oberheim OB8 and Roland Jupiter 8 machines.

In Britain, a cheap but impressive-sounding synth called the Wasp was the first computer-controlled instrument to become widely used in the music business. Like the more modern SH-101 and the Pro One, this synthesiser employs a microprocessor to interface the piano keyboard with the analogue oscillators and filters. Opposite, courtesy of the manufacturers, is a block diagram of the Roland SH – 101 elements to illustrate the current state of the art.



## POLYPHONIC SYNTHESISERS

A polyphonic synth is simply the amalgam of a number of monophonic synths which are put in the same box and controlled by the same keyboard (five in the case of Prophet, eight in the case of Jupiter). These added voices create extra work for the computer, however, in more ways than one!

First of all, the computer has to convert up to eight keypresses into control codes in order to be able to define pitches for the VCOs. Then it has to decide which VCO will generate which note. Finally, just as before, it has to send gate information to the relevant envelope generator. This can prove quite a problem when the player could be anyone from Semprini to my tone-deaf cat running up and down the keys! If you turn to look at the organ program in the previous chapter you will find a scaled down solution to this problem.

In the process of developing better and faster synthesisers it soon occurred to the manufacturers that since a computer was needed on board anyway, it might as well be used to its full capacity. Soon afterwards the programmable polysynth was born. One immediate benefit was that instead of having to keep drawings of control knob settings to hand to reproduce your favourite sound, it now became



available at the flick of a switch. This is because, after a sound is set up on a programmable synth, the knob settings are stored in the computer memory. The sound is also assigned a number, and whenever the number is typed in, the sound is called up.

This development changed the face of music. Previously, in order to fully exploit the possibilities introduced by the synthesiser musicians also had to have a boffin like mentality. The patch leads and bewildering array of knobs and switches put many a potential synthesist off the whole idea of tackling this promising but daunting hardware. Now that sounds are available simply by pressing a button, all that twiddling and tweaking can be done in the privacy of your own home, thus, saving hours of valuable studio time (and a lot of red faces at concerts).

## SEQUENCERS

One logical extension of using a computer to create rhythms is to get it to play melodies. A machine that does precisely this job is known as a sequencer. A sequencer, therefore, is any device which is used for the automatic control of an external sound synthesiser.

Sequencers originally functioned entirely without the use of computer technology, for they were simply a series of modules which could be set to particular voltages and were then triggered sequentially. These voltages caused the external synthesiser to play a sequence of notes. Since such a repetitive task is ideally suited to computers, this old-fashioned analogue

method of sequencing was quickly abandoned in favour of digital technology.

Simple sequencers capable of recalling up to one hundred steps are now built into modern synthesisers as a standard facility. The Roland SH 101 and the PRO ONE can both be interfaced with a drum box, which makes it possible for the sequence stepping to be synchronised with the drum machine's tempo. If you hunt out some of their records and listen to them bearing this in mind you will soon realise that this type of mechanical sequencing is used extensively by Giorgio Moroder and Kraftwerk.

More sophisticated sequencers such as the Roland MC-4 and the Oberheim DSX allow many channels of music to be recorded along with phrasing and dynamics. These machines have developed from the single channel one hundred note sequencers in the same way that the Linn Drum developed out of the early rhythm boxes. By recording the sequence using increasingly smaller and smaller time intervals, you arrive at an effect which precisely mirrors real time playing.

The MC-4 Microcomposer, for example, is a music-dedicated 48k microcomputer. Musical information is entered into the computer using a combination of the function keys and a numeric keypad. To input a melody three types of information must be entered:

1. Pitch: A 125 note chromatic scale is numbered as follows:

12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 C C# D D# E F F# G G# A A# B C C# D D#

To enter pitches the edit mode is selected and the chromatic pitch values (CV's) typed in sequence. A bar structure can be set up at this stage by pressing the <<>MEASURE END<>> key whenever a barline is required.

2.Step: The step time is defined as the number of steps from the start of one note to the start of the next in the sequence. Suppose, for example, we decided to define a bar of 4/4 as 192 steps:

Note	Value	Step	Time
semibreve	1	192	
minim	1/2	96	
crotchet		1/4	48
quaver	1/8	24	
semiquaver	1/16	12	

This high number of subdivisions per bar is chosen because the larger this figure the more subtle the phrasing that results when we input the final set of information.

3. Gate: The gate value defines the phrasing of the notes and can be any number from zero to the step time. For example if the gate value equals the step then legato phrasing results. If the gate value is less than the step then staccato phrasing results. A zero value for gate produces a rest. The following musical example should make the functions of Pitch, Step and Gate clearer. CV's for Pitch are entered as above. The Step time is the entire length from the start of one note to the start of the next, and the two quavers hence have a quaver rest added (24+24=48). The Gate time input then gives the phrasing of the music:

Time Base = 48

Pitch (CV): 21 23 24 26 21

Step: 72 24 48 48 192

Gate: 70 20 24 24 144

Measure End

The MC-4 has four channels each with two pitches, one step and one gate. As a result, it becomes possible for the musician to compose harmonically complex music. In practise, this would be next to impossible, were it not for the sophisticated editing commands. Any part of a piece can be deleted, copied, transposed or inserted by means of a fairly straight-forward command syntax. For instance, if we wished to copy bars 1 to 16 at the end of a 32 bar tune the following syntax would be used:

```
COPY 1,16,1 <>>;ENTER<>>;
```

The cursor is set at bar 33 (the end of bar 32), the COPY key is pressed followed by 1(bar 1 start), 16 (bar16 finish), 1 (one time), and .

This refined form of sequencing is a long way from the haphazard analogue approach and would be totally impossible to achieve without a micro - and a very clever piece of software.

## DIGITAL SYNTHESISERS

This latest giant leap in synth technology has seen the computer take over the production of certain forms of musical composition almost entirely. Not content with just controlling and storing sounds, some ambitious microchip decided it should get in on the act and generate the sounds as well! Instruments such as the Yamaha DX7 and the PPG Wave 2.2 are simply microcomputers disguised as keyboards.

Wave shapes are generated digitally rather than using simple sine or square wave oscillators. These waves are processed and shaped inside the computer and finally appear as analogue sounds, courtesy of a D/A converter. The most sophisticated electronic instrument available to date belongs to this class of digital synthesisers and is called the Fairlight CMI. Unlike the instruments described so far, the Fairlight makes no pretense of being anything other than a computer. The alphanumeric keyboard, twin disc drives and VDU are enough to send the average musician screaming from the studio. Music makers such as Vince from the Assembly (ne Yazoo), Depeche Mode and the world famous producer Steve Levine, on the other hand, have taken to the revolutionary new instrument like the proverbial duck to water.

The Fairlight can best be described as a complete music production system in one box. It can either generate sounds itself, from digitally produced sine and square waves, or sample noises from the real world, create a scale of similar sounds and play them back. Once in the computer, the sound waveforms can also be displayed on the VDU and directly modified to create startling new effects. Once the sound has been chosen it can be played manually on a piano keyboard (with eight note polyphony) or programmed, using a variety of compositional languages. Using any one of these languages, the Fairlight can play up to eight different sounds at the same time to create complex arrangements which were previously only made possible by using a large group of highly trained musicians.

Because the Fairlight is a software-based instrument it is much less prone to obsolescence than its more traditional counterpart such as the Wave 2.2. Since its inception many modifications have been made in order to improve its sampling and refine the programming languages.

For those of you who are as yet unfamiliar with the concept of sampling I should insert a short explanation of the term at this point in the text. Sound sampling involves digitally recording a natural sound into a computer memory. Basically, the amplitude of the wave shape of the sound is measured against time and then this information is recorded (as a series of numbers) by the computer.

AMPLITUDE	AMPLITUDE	VALUES
SOUNDWAVE	TIME SAMPLING	TIME SEQUENCE

The quality of the sample is dependent on the frequency with which the sampling takes place. The Fairlight samples at a maximum frequency of 30000Hz (30000 samples per second). Once in the computer, the sound can be displayed and changed using a light pen or numerical input.

Vibrato, filtering and tuning changes can all be made simply by using the correct commands. Since the facilities of the CMI are so wide ranging each area is designated as a separate 'page'. These are as follows:

Page No.	Title	Functions
1	INDEX	Self explanatory
2	DISK CONTROL	Loading/saving files and programs
3	KEYBOARD CONTROL	Tuning, polyphony, scaling, instrument files
4	HARMONIC	Graphs, looping
	ENVELOPES	
5	WAVEFORM	Additive synthesis
	GENERATION	

6	WAVEFORM	Light pen, merging, display
	DRAWING	
7	CONTROL	Key velocity, portamento, vibrato
	PARAMETERS	
8	SAMPLING	Rates, filtering
9	SEQUENCER	Basic sequencing
D	WAVEFORM DISPLAY	Display
L	LIBRARY	Disk library, files
C	MCL	Music Composition language
R	PAGER	Rhythm sequencer

From a brief survey of the above; you have doubtlessly realised that it is beyond the scope of this book to go into detail about the Fairlight or any of the other instruments mentioned in this chapter. However they have provided the inspiration for many of the programs contained in this book and, hopefully, knowing something about them will provide you with a few ideas of your own for further experimentation.

# 11 Interfacing and Peripherals

The musical interfacing potential of the BBC is so great that it well deserves a book devoted entirely to the topic. The user port, the one megahertz bus and the printer port can all accept and transmit digital information which can in turn be used to control synthesisers, Digital to Analogue and Analogue to Digital converters, relays and any number of similar devices. The on-board Analogue to Digital port can also accept analogue voltage information, such as that generated by a synthesiser or a tape recorder. It is beyond the scope of this book to go into the details of the design and construction of the sort of interfaces which would, for instance, allow the BBC to generate the voltages required to drive a synthesiser, or sample frequencies via a fast A/D converter. Other books and magazines can supply this information, a partial list of which can be found in the appendix. There are, however, a number of straightforward add-ons we can use or construct without embarking on a university electronics course. So let us begin by taking a look at the first of these, namely the common or garden joystick.

## Joysticks

The humble joystick can be put to a number of other uses apart from the traditional and well documented one of controlling laser cannons or manoeuvring Pacmen. Before we come to use the stick to control elements of a program, however, we should start by RUNning the following test program.

```

10
20 REM *** Stick Test ***
30
40 MODE 7
50 VDU23;8202;0;0;0;
60
70 REPEAT
80   X%=ADVAL1DIV1024
90   Y%=ADVAL2DIV1024

```

```

100  PRINTTAB(5,10);CHR$(129);"ADVAL(1
): ";CHR$(134);X%;"  "
110  PRINTTAB(5,12);CHR$(129);"ADVAL(2
): ";CHR$(134);Y%;"  "
120  SOUND&11,-10,X%,-1
130  SOUND&12,-10,Y%,-1
140  UNTIL FALSE

```

On my joystick this program returns the following ADVAL values:

	ADVAL 1	ADVAL 2
Top left position	63	63
Top right position	0	63
Bottom left position	63	0
Bottom right position	0	0

If your stick does not give similar readings, one adjustment you can make is to try turning it upside down! If this does not work, you will need to change parts of the program in order to return the correct values. Note, however, that the *absolute* value returned is not as important as the *relative* increase and decrease resulting from stick movement.

This program uses the ADVAL function to test for voltages appearing at two of the four Analogue to Digital converters in the computer. The values of ADVALs 1 and 2 (normally returning 0 to 65520) are divided by 1024, in this case, to bring their values into a usable range. These values are then fed into the display and two SOUND statements.

Once we have standardised our sticks we can RUN the following program, which allows us to draw abstract images on the screen while being accompanied by even more abstract music.

```

10
20 REM    *** Stick Fun ***
30

```

```

30
40 MODE 2
50 VDU23;8202;0;0;0;
60
70 PROCADVAL
80 MOVEX%*10,Y%*8
90
100 REPEAT
110   PROCADVAL
120   GCOL0,RND(7)

```

Interfacing and Peripherals 209

```

130   PLOT5,X%*10,Y%*8
140   SOUND&11,-10,X%,-1
150   SOUND&12,-10,Y%,-1
160 UNTIL FALSE
170
180 DEF PROCADVAL
190 X%=127-ADVAL1DIV512
200 Y%=ADVAL2DIV512
210 ENDPROC

```

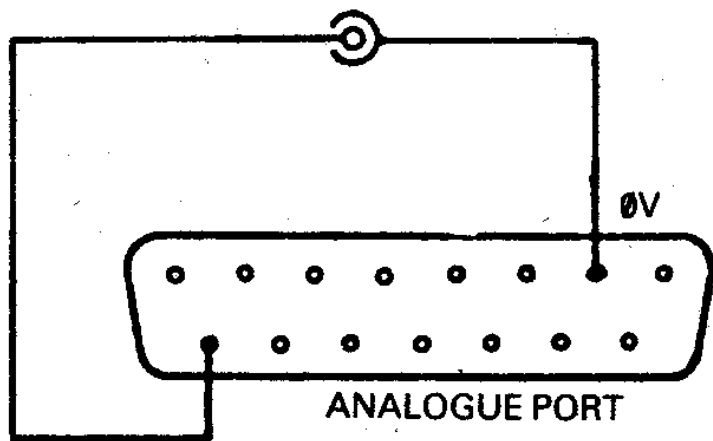
“Stick Fun” uses the ADVAL1 and ADVAL2 values to PLOT randomly coloured lines on the screen (Lines 120 and 130), and feed two SOUND statements in Lines 140 and 150.

This example is perhaps not the most practical application to which you might decide to put your joystick. Let us now turn to another option and try using a stick to position music on a stave. The “Write” program in the Chapter Seven would make a good starting point for this project, since the fire button could be used to deposit the note onto the stave. The fire-button itself is controlled via ADVAL0. (See Page 202 of the *User Guide* for further details).



## TAPE RECORDER

The headphone output of a tape recorder or amplifier gives voltages of a suitable range for direct connection to one of the Beeb's built-in A/D converters. To accomplish this you will have to go out first and buy an analogue port connector and lead (preferably already made up) as well as a plug which is suitable for the headphone output you are using. Once ready and fully equipped the headphone plug should, to use ADVAL1, be connected across the two analogue port pins, as shown in the diagram below:



Most connectors have numbered plugs, so choosing the correct wires should not present you with any difficulty. Plug one end into the computer and the other into a tape recorder which allows you to listen to music while the headphone plug is connected.

The following program will give you a graphical display of any music which is played on the recorder.

---

210 Interfacing and Peripherals

```

10
20 REM"***AMPLITUDE SAMPLING***
30
40 *KEY9 RUN:M
50 *KEY0 MODE7:M VDU14:M LIST:M
60 REM"*****
70 REM"*PRESS F9 TO START SAMPLE*
80 REM"*****

```

```

80 REM"*****
90 REPEAT
100 REPEAT:S%=ADVAL2:UNTILS%>800
110 TIME=0
120
130 MODE0
140 MOVE0,100
150 FOR X%=0 TO 250
160 Y%=ADVAL2 DIV 30
170 PLOT 5,X%*5,Y%+100
180 TIME=0: REPEAT:UNTIL TIME=1
190 NEXT
200 UNTIL FALSE

```

Line 100 waits for a reasonable volume of music to be produced before the program starts, avoiding a response to tape noise. The display is drawn in MODE 0 and consists of an X axis left to right scan, with the Y axis height controlled by the volume of the music. The Y axis information is provided by the scaled ADVAL2 output.

If your recorder is of the type that gives a headphone output when a microphone is connected, this program is excellent for examining the volume profiles of words which are spoken into the microphone. As you will doubtlessly realise, this kind of activity is the starting point for what develops into word recognition systems. The following program uses similar techniques (but with Y% values from ADVAL3) to control bars of colour on the screen:

```

10
20 REM"***BAR CHART***
30
40 MODE2
50 VDU 23,224,255,255,255,255,255,25
5,255,255
60 X%=0
70 T%=0

```

```

60  A%-0
70  T%=0
80  VDU5
90  REPEAT

```

```

100  VDU23;8202;0;0;0;
110  REPEAT
120      *FX17,3
130      Y%=ADVAL3 DIV 100
140      GCOL0,T%
150      T%=T%+1:IF T%>6 T%=1
160      FOR L%= 0 TO Y% STEP50
170          MOVEX%,L%
180          PRINTCHR$(224)
190      NEXT
200      FOR R%= L% TO 800 STEP50
210          GCOL0,0
220          MOVEX%,R%
230          PRINTCHR$(224)
240      NEXT
250      X%=X%+100
260      IF X%=1200 X%=0
270      UNTIL X%=0
280  UNTIL FALSE

```

Another application of music input through the Beeb's A/D port could be that of controlling screen graphics or external lights through relays. If all four converters are used, each input could be selectively filtered to produce distinct frequency bands, (bass, middle, treble). Each band could then control a different light or graphics device to produce what is known as a 'Disco lights' effect.

## SYNTHESIZER

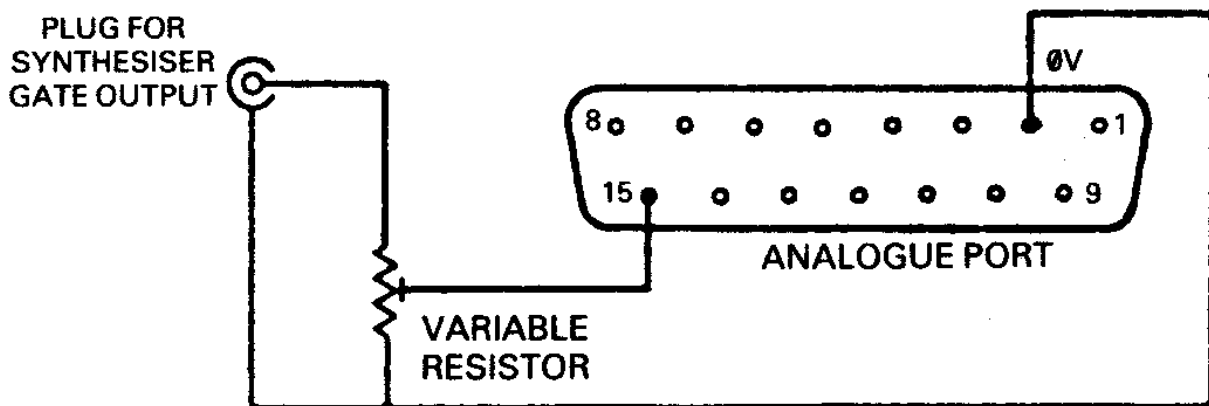
## SYNTHESISER

To take part in the next batch of interfacing, it is necessary for you to have a one volt per octave synthesiser or keyboard. In addition you will require two 4.7Kohm variable resistors, the analogue input port connector which we used previously and two plugs. The synthesiser should be of the type that has control and gate voltage outputs. Most Roland, Sequential Circuits and Moog synths fall into this category.

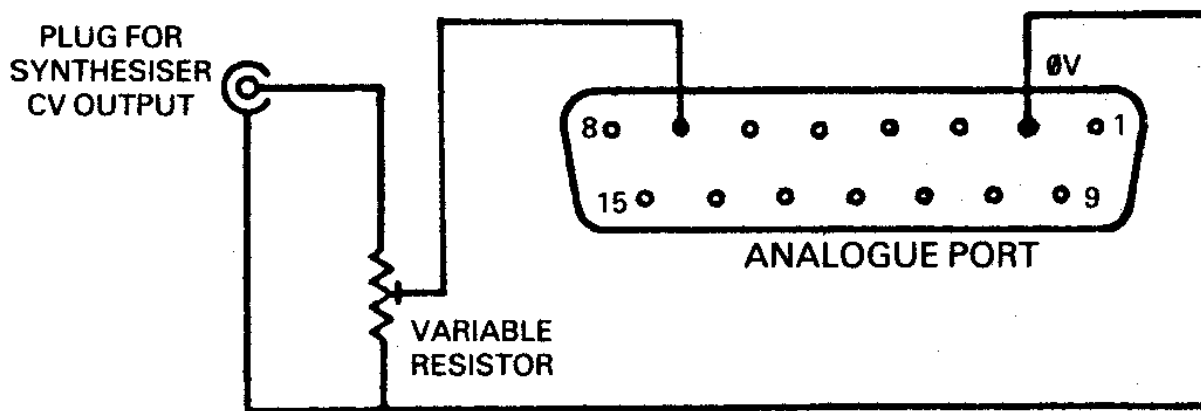
Korg and Yamaha synths, however, operate on a 1.5 volt per octave standard. If you are using one of these keyboards, adjustments will have to be made to the software conversion factors.

The connector and resistors should be connected as shown in the following diagram:

### 212 Interfacing and Peripherals



ADVAL 1: GATE OUTPUT



ADVAL 2: CV OUTPUT

The plugs required for connection at the keyboard end will depend on the

The plugs required for connection at the keyboard end will depend on the type of synth you are using. ADVAL2 (pins 2 and 7) should be connected to the gate output and ADVAL1 (pins 2 and 15) to the CV (control voltage) output of the synth. Once this is accomplished, type and RUN the following calibration program:

```

10
20
30
40
50  REM"*** P.TEST ***
60
70
80
90  CLS
100  VDU23;8202;0;0;0;
110  PRINTTAB(10,8);"  X%  Y% "
120  REPEAT
130    Y%=ADVAL(2)/1024
140    PRINTTAB(10,10);"    ";X%;"    ";
Y%;"  "
150    PRINTTAB(7,19);"ADVAL(1)/16:"
;ADVAL(1)/16;"  "
160    IF Y%>0 THEN X%=INT(ADVAL(1)/10
24):SOUND&111,-10,X%*4,-1:SOUND&12,-10,X
%*4,-1 ELSE SOUND&11,0,0,-1:SOUND&12,0,0
,-1
170    UNTIL FALSE

```

The variable attenuator (resistor) controlling ADVAL2 should be adjusted to give a zero reading when no key is pressed, and a reading greater than 10 when a key is being pressed. The attenuator for ADVAL1 should be adjusted to read multiples of 12 when the key 'C' is pressed. Test this over the complete extent of the range (including using octave controls if any). The lowest C on the keyboard should read 12. ADVAL1 is now reading control voltages and ADVAL2 gates from the synth. It should be possible to play the keyboard and hear the computer produce the correct notes.

The main part of the program lies in Line 160. If a positive gate (Y% value from ADVAL2) is detected it will SOUND a note and PRINT a value proportional to the control voltage (ADVAL1). If no gate is detected it will shut off the SOUND.

Assuming everything appears to be working hunky dory with your system so far, we can now all move on to the next program, "Keyboard Recognition". This program could be used as the starting point for a series of music tuition programs revolving around the piano keyboard. With this version the computer displays a note on a treble clef stave and then waits until the correct piano key is pressed. The number of tries is then displayed, accompanied by a suitable comment and the note SOUNDing. The process is then repeated until, hopefully, the pupil gains a thorough knowledge of the keyboard. Brief instructions are displayed as an option when the program is RUN.

The variable attenuator (resistor) controlling ADVAL2 should be adjusted to give a zero reading when no key is pressed, and a reading greater than 10 when a key is being pressed. The attenuator for ADVAL1 should be adjusted to read multiples of 12 when the key 'C' is pressed. Test this over the complete extent of the range (including using octave controls if any). The lowest C on the keyboard should read 12. ADVAL1 is now reading control voltages and ADVAL2 gates from the synth. It should be possible to play the keyboard and hear the computer produce the correct notes.

The main part of the program lies in Line 160. If a positive gate (Y% value from ADVAL2) is detected it will SOUND a note and PRINT a value proportional to the control voltage (ADVAL1). If no gate is detected it will shut off the SOUND.

Assuming everything appears to be working hunky dory with your system so far, we can now all move on to the next program, "Keyboard Recognition". This program could be used as the starting point for a series of music tuition programs revolving around the piano keyboard. With this version the computer displays a note on a treble clef stave and then waits until the correct piano key is pressed. The number of tries is then displayed, accompanied by a suitable comment and the note SOUNDing. The process is then repeated until, hopefully, the pupil gains a thorough knowledge of the keyboard. Brief instructions are displayed as an option when the program is RUN.

```

50
60  REM" *****
70  REM" ***KEYBOARD RECOGNITION***
80  REM" *****
90
100
110  REM"          SET UP
120
130  MODE7
140  PRINTTAB(2,10); "Do You Want Instr
uctions?   Y/N  "
150  A$=GET$
160  IF A$="N" THEN GOTO 170 ELSE PROC
instr
170  R%=0
180  DIM b%(21,3)
190  VDU23,225,30,63,127,255,255,254,2
52,96
200  VDU23,224,36,37,38,36,44,52,100,1
64
210  VDU23;8202;0;0;0;

```

---

## 214 Interfacing and Peripherals

```

220  MODE5
230  VDU19,1,6,0,0,0
240  FOR I%=0 TO 18
250      FOR O%=1 TO 2
260          READ b%(I%,O%)
270      NEXT
280  NEXT
290  DATA 0,0,0,1,1,0,2,0,2,1,3,0,3,1,

```

```
290 DATA 0,0,0,1,1,0,2,0,2,1,3,0,3,1,
4,0,4,1,5,0,6,0,6,1,7,0,7,1,8,0,9,0,9,1,
10,0,10,1,11,0,11,1
```

```
300
```

```
310
```

```
320 REM"          THE PROGRAM
```

```
330
```

```
340 REPEAT
```

```
350     CLG
```

```
360     L%=100:Q%=-1
```

```
370     PROC CLEF
```

```
380     PROC choose
```

```
390     PROC Note
```

```
400     VDU23;8202;0;0;0;
```

```
410     PROC adval
```

```
420     UNTIL FALSE
```

```
430 END
```

```
440
```

```
450
```

```
460 REM"          THE PROCEDURES
```

```
470
```

```
480 DEFPROC CLEF
```

```
490 GCOL0,2
```

```
500 FOR Line=0 TO 144 STEP 36
```

```
510     MOVE0,450+Line: DRAW1280,450+Line
```

```
520     NEXT
```

```
530 MOVE60,490
```

```
540 RESTORE600
```

```
550 FOR Z%=1 TO 39
```

```
560     READ X%,Y%
```

```
570     DRAW60+X%,490+Y%
```

```
580     GCOL0,3
```

```
590 NEXT
```



Interfacing and Peripherals 215

0



## 0

file:///C:/Documents%20and%20Settings/Chris%20Richardson/Desktop/New%20Folder/bbc\_mm\_11.htm (13 of 36)20/06/2012 08:47:09



ON A PIANO KEYBOARD"

```
1220  PRINTTAB(0,12); "A ONE VOLT PER  
OCTAVE SYNTHESISER MUST "
```

Interfacing and Peripherals 217

```

1230 PRINTTAB(0,13); "BE CONNECTED TO  
THE ANALOG INPUT PORT "  
1240 PRINTTAB(0,14); "ON THE BACK OF  
THE COMPUTER "  
1250 PRINTTAB(0,17); "NOTES OF THE T  
REBLE CLEF WILL BE"  
1260 PRINTTAB(0,18); "DISPLAYED AND  
YOUR KNOWLEDGE OF THE "  
1270 PRINTTAB(0,19); "KEYBOARD WILL  
BE ASSESSED BY YOUR "  
1280 PRINTTAB(0,20); "TUTOR "  
1290 PRINTTAB(4,23); "PRESS ANY KEY TO  
CONTINUE "  
1300 A$=GET$  
1310 A$=""  
1320 ENDPROC  
1330  
1340 REM"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
O  
1350 DEF PROCadval  
1360 TIME=0  
1370 REPEAT  
1380 IF ADVAL(2)/1024>20 THEN FORG=1  
TO 40:NEXT:N%=(ADVAL(1) DIV 1024)  
1390 IFN%<>L% THEN Q%=Q%+1  
1400 IFN%=L% THEN Q%=Q%  
1410 L%=N%

```

```

1410      L%=N%
1420      UNTIL N%=P% AND Q%>0
1430      SOUND1,-15,P%*4,40
1440      COLOUR1
1450      PRINTTAB(3,24); "You took ";Q%; " g
oes"
1460      PRINTTAB(3,27); "Time: ";TIME/100;
"secs"
1470      IF Q%<2 PRINTTAB(5,8); "WELL DONE!"
"
1480      IF Q%=2 PRINTTAB(2,8); "NOT SO GOOD
D...."
1490      IF Q%=3 PRINTTAB(2,8); "TRY HARDER
!!"
1500      IF Q%=4 PRINTTAB(2,8); "THIS IS JUST NOT
GOOD ENOUGH .."

```

## 218 Interfacing and Peripherals

```
1510 IF Q%=5 PRINTTAB(0,6);"WHAT DO YO  
U THINK":PRINTTAB(0,8);"YOU ARE PLAYING  
AT ?":PRINTTAB(0,10);"ARE YOU TONE DEAF?  
?"
```

```
1520 IF Q%=6 PRINTTAB(0,8); "I CAN'T BE  
LEIVE THIS"
```

```
1530 IF 0%>6 PROCabuse
```

```
1540  FORT=1  TO 4000:NEXT T
```

```
1550 UNTIL FALSE
```

```
1560  ENDPROC
```

1570

```
1580 REM"00000000000000000000000000000000
```

Q

```
1590 DEF PROC0
```

---

```

1590 DEF PROCQ
1600 Q%=Q%+1
1610 L%=N%
1620 G%=1
1630 ENDPROC
1640
1650 REM"oooooooooooooooooooooooooooooooooooo

```

The program is made up of a number of PROCedures, most of which you will notice are devoted to graphics. Many of these will be familiar to you by this point from the graphics discussion of Chapter Seven.

1. Set Up: The instruction option is offered in Lines 130 to 160. Array b%(21,3) is used to relate the stave position with note numbers and symbols. CHR\$(225) is defined as the note ball and CHR\$(224) as half of a sharp symbol.
2. The Program: The main part of the program consists of calling up the various PROCedures inside a REPEAT . . . UNTIL loop.
3. PROCLEF: As you may have come to expect from what we have discussed so far in this book, PROCLEF DRAWs a treble clef and five lines using MOVE and DRAW commands, rather than defined text characters. This technique is described fully in the earlier chapter which deals with graphics.
4. PROCchoose: Once the clef and stave have been drawn a note is chosen randomly by PROCchoose. Line 940 makes sure that the new note chosen is different to the previous note. Line 950 gives the vertical position of the note (Z%) by relating the random number R% to the b%() array. This rather complicated arrangement is required because the stave is related to Major scale notes, while, in contrast, SOUND statements are based on the chro-

The program is made up of a number of PROCedures, most of which you will notice are devoted to graphics.

Many of these will be familiar to you by this point from the graphics discussion of Chapter Seven.

1. Set Up: The instruction option is offered in Lines 130 to 160. Array `b%(21,3)` is used to relate the stave position with note numbers and symbols. `CHR$(225)` is defined as the note ball and `CHR$(224)` as half of a sharp symbol.
2. The Program: The main part of the program consists of calling up the various PROCedures inside a REPEAT... UNTIL loop.
3. PROCLEF: As you may have come to expect from what we have discussed so far in this book, PROCLEF DRAW s a treble clef and five lines using MOVE and DRAW commands, rather than defined text characters. This technique is described fully in the earlier chapter which deals with graphics.
4. PROCchoose. Once the clef and stave have been drawn a note is chosen randomly by PROCchoose. Line 940 makes sure that the new note chosen is different to the previous note. Line 950 gives the vertical position of the note (`Z%`) by relating the random number `R%` to the `b%()` array. This rather complicated arrangement is required because the stave is related to Major scale notes, while, in contrast, SOUND statements are based on the chromatic scale. Some way must therefore be found to equate an eight note scale with a twelve note system. The `b%` array holds vertical step positions for every chromatic note. Line 960 then decides whether the chosen note is sharp or natural. It does this by examining the second dimension of the array. If we have `b%(R%,2)=1` the program calls PROCsharp and a sharp is drawn at the correct position on the stave. The complete sharp is the two halves of the sharp stacked one on top of the other.
5. PROCNote. PROCNote not only PRINTs the note ball on the correct line or space, using VDU 5 to PRINT at the graphics cursor, but also DRAWs the stem of the note, either up or down depending on its position on the stave. Notes that are above the middle line are sent to PROCdown, the rest are sent to PROCup. These DRAW the stem relative to the note position variable `Z%`, using graphics techniques. Once the note has been displayed the analogue port input is used to test if the correct note is being pressed. ADVAL 2 tests for a keypress. If a key is pressed ADVAL 1 is read and its value compared with the previous value `L%` and the current note (`P%=R%+14`) derived from PROCchoose. `Q%` is used to count the number of attempts which have been made at guessing the correct note. When the correct note is chosen, so that `N%=P%`, it is SOUNDED in Line 1430, the number of attempts is PRINTed in Line 1450, the time in seconds is displayed and a comment, dependent on the number of goes, is PRINTed. If the number of goes is greater than six, the computer becomes frustrated and gives up!

Many changes and additions could be made to this program, depending on the use to which you wanted to put it. For example 'Tutor' could be modified to play a note and make a comment after every attempt. In addition an overall score could be kept, and a time limit introduced, to add an element of competition. Different clefs and keys could also be tested. If a piano type keyboard was not available the standard alphanumeric keyboard could be used either to type in symbols or to be labelled as white and black notes.

One logical extension of the parameters of this program is to display groups of notes in phrases of differing timings. A horizontal scroll technique could even be used to test sight reading of fairly complex pieces of music. A sideways scroll can be accomplished by re-programming register 13 of the 6845 video controller chip. VDU 23 is used to bring this about. See the *Advanced User Guide* for more information on this subject. The inclusion of the next program was requested by my editor. It is a revised version of the program given above. It was pointed out to me that people not possessing a musical keyboard capable of interfacing with the Beeb might be a

bit miffed at missing out on this whizzo program. Thus the revised program that follows, using many of the same PROCedures as the aforementioned program, has had a keyboard display included and the ability to accept input from the alphanumeric keypad.

---

## KEYBOARD RECOGNITION VERSION 2

On RUNning the program you are first asked if you wish to see the instruction page, revised from that of the original program. Once into the test the screen displays a treble clef with a note on the stave. Underneath a piano keyboard is displayed with each key labelled with a letter from the “qwerty...” and “12345...” key sequences of the computer keyboard.

The idea is to identify the note on the screen and press the corresponding key in as short a time as possible. If an incorrect key is pressed that note will be SOUNDED quietly and the note will continue to be displayed. If the correct note is pressed the note will SOUND loudly and a message will be displayed on the screen along with the time you took. If you got the note right first time the message will be congratulatory, otherwise...

This process will continue until either you fail to get the correct note so many times that the computer gives up, or you press . On escaping your overall score will be given as the number of successful first time attempts, out of the total number of notes.



```

10 REM" *****
20 REM" ***KEYBOARD RECOGNITION ***
30 REM" *** VERSION 2 ***
40 REM" *****
50
60
70 REM"          SET UP
80
90 MODE7
91 SCORE=0:NOS=0
95 ON ERROR MODE7:PRINTTAB(7,12);"YOUR SCORE IS ";SCORE;"/";NOS:END
100 PRINTTAB(1,2);CHR$(134);"MUSICAL RECOGNITION TEST";CHR$(134);" USING THE ALPHANUMERIC KEYBOARD"
110 PRINTTAB(2,10);"Do You Want Instructions? Y/N "
120 A$=GET$
130 IF A$<>"N" PROCinstr
140 MODE5
150 R%=0
160 DIM b%(21,3)
170 KEY$="Q2W3ER5T6Y7UI900P@^[\_ "
180 VDU23,225,30,63,127,255,255,254,252,96
190 VDU23,224,36,37,38,36,44,52,100,164

```

```
200  VDU23;8202;0;0;0;
210  PRINTTAB(2,18);CHR$(129);"and ass
essed by the computer."
220  VDU19,1,6,0,0,0
230  FOR I%=0 TO 18
240      FOR O%=1 TO 2
250          READ b%(I%,O%)
260      NEXT
270  NEXT
280  DATA 0,0,0,1,1,0,2,0,2,1,3,0,3,1,
4,0,4,1,5,0,6,0,6,1,7,0,7,1,8,0,9,0,9,1,
10,0,10,1,11,0,11,1
290
300
310  REM"          THE PROGRAM
320
330  REPEAT
340      GCOL0,128
350      CLG
360      L%=100:O%=0
370      PROC CLEF
380      PROC KEYBOARD
390      PROC choose
400      PROC Note
410      VDU23;8202;0;0;0;
420      PROC recog
430      UNTIL FALSE
440  END
450
460
470  REM"          THE PROCEDURES
480
490  DEFPROC CLEF
```





0

file:///C:/Documents%20and%20Settings/Chris%20Richardson/Desktop/New%20Folder/bbc mm 11.htm (24 of 36)20/06/2012 08:47:09

```

1160
1170  REM"000000000000000000000000000000000000
0
1180  DEF PROCinstr
1190  CLS
1200  PRINTTAB(10,3);CHR$(141);CHR$(134
);"INSTRUCTIONS";TAB(10,4);CHR$(141);CHR
$(134);"INSTRUCTIONS"
1210  PRINTTAB(2,6);" This program test
s your knowledge of"

```

## 224 Interfacing and Peripherals

```

1220 PRINTTAB(2,8);" the piano keyboard. "
1230 PRINTTAB(2,10);CHR$(131);"A note
will be displayed on the"
1240 PRINTTAB(2,12);CHR$(131);"the Tre
ble Clef. The keyboard diagram"
1250 PRINTTAB(2,14);CHR$(131);"will in
dicate which key you must"
1260 PRINTTAB(2,16);CHR$(131);"press."
;CHR$(129);"Your performance will be tim
ed"
1270 PRINTTAB(2,18);CHR$(129);"and ass
essed by the computer."
1280 PRINTTAB(2,20);CHR$(135);"Good Lu
ck!"
1290 PRINTTAB(4,23);"PRESS ANY KEY TO
CONTINUE      "
1300 A$=GET$
1310 A$=""
1320 ENDPROC
1330

```

```

1330
1340 REM"000000000000000000000000000000000000
0
1350 DEF PROCrecog
1360 TIME=0
1370 REPEAT
1380     PITCH$=GET$
1390     N%=INSTR(KEY$,PITCH$)+11
1400     IFN%<>P% AND N%>11 SOUND1,-15,N
%*4,15
1410     IFN%<>L% THEN Q%=Q%+1
1420     IFN%=L% THEN Q%=Q%
1430     L%=N%
1440     UNTIL N%=P% AND Q%>0
1450     FOR Channel%=1 TO 3:SOUND&200+Cha
nnel%,-15,P%*4,40:NEXT
1460     COLOUR1
1470     PRINTTAB(3,24);"You took ";Q%;" g
oes"
1480     PRINTTAB(3,27);"Time: ";TIME/100;
"secs"
1487     NOS=NOS+1
1490     IF Q%<2 PRINTTAB(5,8);"WELL DONE!
":SCORE=SCORE+1

```

```

1500 IF Q%=2 PRINTTAB(2,8);"NOT SO GOOD...."
1510 IF Q%=3 PRINTTAB(2,8);"TRY HARDER!!"
1520 IF Q%=4 PRINTTAB(2,8);"THIS IS JUST NOT GOOD ENOUGH"

```

GOOD ENOU

```
1530 IF Q%=5 PRINTTAB(0,6);"WHAT DO YO  
U THINK":PRINTTAB(0,8);"YOU ARE PLAYING  
AT ?":PRINTTAB(0,10);"ARE YOU TONE DEAF?  
?"
```

```
1540 IF Q%=6 PRINTTAB(0,8);"I CAN'T BE  
LEIVE THIS"
```

```

1550 IF Q%>6 PROCabuse
1560 FORT=1 TO 4000:NEXT T
1570 UNTIL FALSE
1580 ENDPROC

```

```
1590  
1600 REM "XXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

```

0
1610 DEF PROC

```

```

1620  Q%=Q%+1
1630  L%=N%
1640  G%=1
1650  ENDFROC

```

```
1670 REM"0000000000000000000000000000000000
```

```

0
1680
1690 REM" *****KEYBOARD DRAW*****

```

1700 DEF PROCKEYBOARD

1710 GCOLO.129

1720

1730 VDU5

```
1740      XPOS%=20:YPOS%=50:B%=1250:A%=300
```

```
1750 PROCBOX(1,XPOS%,YPOS%,B%,A%)
```

1760 XPOS%=45: YPOS%=50

1770 FORD%=1T016

```
1780      MOVEXPOS%,YPOS%:PLOT6,XPOS%,3
```



```

1780      MOVEXPOS%,YPOS%:PLOT6,XPOS%,3
50
1790      XPOS%=XPOS%+80
1800      NEXT
1810      YPOS%=170:A%=180:B%=45
1820      XPOS%=20

```

---

## 226 Interfacing and Peripherals

```

1830      FORD%=1 TO 15
1840      IF D%=2 OR D%=5 OR D%=9 OR D%=12
GOTO1860
1850      PROCBOX(2,XPOS%,YPOS%,B%,A%)
1860      XPOS%=XPOS%+80
1870      NEXT
1880      PROCNOTES
1890      ENDPROC
1900
1910      DEFPROCBOX(Colour%,XPOS%,YPOS%,W,
A%)
1920      GCOLO,Colour%
1930      MOVEXPOS%,YPOS%:MOVE(XPOS%+B%),YPO
S%:PLOT85,XPOS%,(YPOS%+A%):PLOT85,(XPOS%
+B%),(YPOS%+A%)
1940      ENDPROC
1950
1960      DEF PROCNOTES
1970      VDU5
1980      GCOLO,3
1990      WHITE$="QWERTYUIOP@[_"
2000      BLACK$="23 567 90 ^\"
2010      FOR X%=1 TO 13
2020          MOVE 55+80*X%,120
2030          PRINT MID$(WHITE$,X%,1)
2040          NEXT

```

```

2030 PRINT MID$(WHITE$,X%,1)
2040 NEXT
2050 GOTO,0
2060 FOR X%=1 TO 12
2070 MOVE 95+80*X%,250
2080 PRINT MID$(BLACK$,X%,1)
2090 NEXT
2100 VDU4
2110 ENDPROC

```

The program initialisation is similar to that of the original with the addition of two variables, SCORE and NOS, which keep track of the score and number of turns respectively. After the instructions have been called, the computer keyboard keys that are to be used are defined (as KEY\$) in Line 170. The array b%(21,3) is then loaded from a DATA statement and is used to relate the position of a note on the musical stave with its SOUND statement number and symbol.

The main program is contained within a loop and follows the following sequence:

1. Draws the clef and stave (PROCCLEF)

---

The program initialisation is similar to that of the original with the addition of two variables, SCORE and NOS, which keep track of the score and number of turns respectively. After the instructions have been called, the computer keyboard keys that are to be used are defined (as KEY\$) in Line 170. The array b%(21,3) is then loaded from a DATA statement and is used to relate the position of a note on the musical stave with its SOUND statement number and symbol.

The main program is contained within a loop and follows the following sequence:

1. Draws the clef and stave (PROCCLEF)
2. Draws the keyboard (PROCKEYBOARD)
3. Randomly chooses a note (PROCchoose)
4. Draws the note on the stave (PROCNote)
5. Waits for input then reacts to answer (PROCrecog)

PROCCLEF draws the treble clef using graphics techniques described earlier. PROCNote positions and draws the note on the stave and calls either PROCup for an upward stem or PROCdown for a downward stem. PROCchoose randomly chooses the next note, making sure that it is different to the previous value. PROCsharp

draws a sharp symbol in front of the chosen note if appropriate.

PROCabuse is called when the computer decides there is no hope for you as a piano player. In addition to berating you over your musical ability it also displays your overall score. PROCinstr contains your initial instructions.

PROCrecog differs from the PROCedure in the earlier version of the program in that instead of using the ADVAL function to test for a keypress on a keyboard attached to the analogue port, in this case GET\$ is used to give us a value for PITCH\$, which is converted to a numerical value, using INSTR, in Line 1390. N% is then compared with the computer's chosen value P%. The computer's reaction to your attempts depend on the value of Q%, the number of goes.

PROCKEYBOARD is concerned with drawing and labelling the piano diagram. It uses a series of different sized boxes to make up the keyboard. Since we are in a four colour mode the choice of colours for this purpose is somewhat limited. Due to the inconsistent pattern of piano keys PROCBOX must be called a number of times. PROCNOTES labels the keys using two strings for the white and black keys. Spaces are included in BLACKS to take account of the missing notes between B and C and between E and F.

## PERIPHERALS

Common peripheral devices for the BBC microcomputer include cassette recorders, disc systems, monitors and printers. The currently available models of printers and monitors are so varied that any comments I could possibly venture would only backfire immediately since chances are that they would only apply to a small percentage of users. For this reason I will leave the interesting subject of printer graphics to others! Many commercially available printer drivers will give hard copies of our music graphics.

Cassette recorders and disc systems can be used for recording DATA RS well as complete programs. We can therefore save our musical compositions in files separate from the program which will play them back. The BBC is so designed that the same set of commands will allow us to create both cassette and disc files. The program "Files" indicates how this might be done in the simple case of a note sequence.

---

### 228 Interfacing and Peripherals

```

10
20    REM      ***  FILES  ***
30
40    MODE 7
50    VDU23;8202;0;0;0;
60    DIM P%(500),D%(500)
70
80

```

```
80
90 REPEAT
100 CLS
110 PRINTTAB(0,6) "P.....PLAY"; ' ;
"R.....READ"; ' ; "S.....SAVE"; ' ;
"L.....LOAD"
120 B$=GET$
130 IF B$="P" PROCplay
140 IF B$="S" PROCsave
150 IF B$="L" PROCload
160 IF B$="R" PROCread
170 UNTIL FALSE
180
190
200 DEF PROCplay
210 X%=1
220 REPEAT
230 SOUND1,-10,P%(X%),5
240 X%=X%+1
250 UNTIL P%(X%)=0
260 ENDPROC
270
280
290 DEF PROCsave
300 INPUT"FILE NAME",A$
310 X=OPENOUT A$
320 X%=1
330 REPEAT
340 PRINT#X,P%(X%)
350 X%=X%+1
360 UNTIL P%(X%)=0
```

```
360  UNTIL P%(X%)=0
370  CLOSE#X
380  ENDPROC
390
400
410  DEF PROCload
420  INPUT"FILE NAME",A$
```

---

Interfacing and Peripherals 229

```
430  Y=OPENIN A$
440  X%=1
450  REPEAT
460  INPUT#Y,P%(X%)
470  X%=X%+1
480  UNTIL EOF#Y
490  CLOSE#Y
500  ENDPROC
510
520
530  DEF PROCread
540  X%=1
550  REPEAT
560  READ P%(X%)
570  X%=X%+1
580  UNTIL P%(X%-1)=0
590  ENDPROC
600
610
620  DATA 53,61,69,73,81,89,97,101,89,
56,34,65,109,0,0,0,0,0,0
```

Firstly a sequence of notes must be written into DATA statements. An example sequence of a scale is given. When the program is RUN this sequence can be READ into an array by pressing 'D'. Pressing 'D' will then

example sequence or a scale is given. When the program is RUN this sequence can be READ into an array by pressing 'R'. Pressing 'P' will then cause the sequence to be played. 'L' and 'S' allow you to LOAD and SAVE files on tape or disc. A file name must be typed in for each new sequence.

1. **The Program:** The Menu is contained in a REPEAT . . . UNTIL loop which allows you to select the various options by pressing 'P', 'S', 'L' or 'R'.
2. **PROCplay:** This reads the current notes from the pitch array P%. A value of P%(X%)=0 causes the PROCEDURE to come to an end.
3. **PROCsave:** Line 300 asks for a file name. This can be up to ten characters long. Line 310 opens the file and PRINT#X,P%(XC%) reads pitch data to the file for all X% values UNTIL P%(X24)=0. The file is then closed in Line 370.
4. **PROCload:** This procedure has precisely the opposite function to that of PROCsave. Once again a file name is requested and the file, if found, is opened. Information is then read from the file. Line 480 tests for the end of the file and closes the file when this is detected.
5. **PROCread:** This procedure READS note DATA into the pitch array.

This is a very basic example of file handling. If you were interested in adapting it for practical use, obviously a number of refinements would have to be added. For instance, as it stands, file names larger than ten letters would cause the system to crash, as would a file name which could not be found, for whatever reason, when loading.

There is one other external device that can be connected to the BBC. Note, however, that one disadvantage to embracing the device lies in the fact that this does involve opening up the case of the computer and having a quick root around inside.

## EXTERNAL AMPLIFIERS AND RECORDERS

The BBC's sound chip drives the internal speaker directly, without utilising any external form of amplification. This means that the volume levels which are attainable from the computer are not exactly ear-splitting. If this situation suits you down to the ground then you need read no further. If you have ever felt the need to play your latest computer composition to the whole street, however, then this is how you go about doing it. Do not attempt this unless you are confident of your mechanical and soldering skills, however, as it means opening up your BBC

Micro.

Since most of you probably possess some form of amplifier and speaker arrangement, be it a Quad Hifi or a 'Ghetto Blaster', I will not describe the construction of an amplifier! I will simply indicate how your BBC can be connected to one of these devices, either for live performance or direct recording into a tape recorder (i.e. not via a microphone).

The parts which you will require for this modification include a length of two-core wire (about a foot) and a socket (the exact type is entirely dependent on what you wish to hook it up to).

Solder the two cores at one end of the wire length to the socket in preparation for connection to the computer. The computer case can be removed by unfastening four Philips screws, the two underneath nearest the front and the two on the back of the BBC. It is also necessary to remove the keyboard by unfastening two additional screws underneath the computer. This done, the keyboard can now be removed entirely by carefully unplugging the multi-pin connecting cable and speaker lead. Once you have done this, the speaker can be clearly seen on the left of the keyboard.

Unfasten the speaker from the board by removing the restraining screw and solder the free ends of the two-core wire to the speaker connection. The socket end of the cable can be fed out through the Econet port or the hole in the computer casing which is to be found next to it. Now reassemble the computer so that you can connect the new sound output to the tape or tuner input of your amplifier.

---

A more sophisticated arrangement would include an internal/external sound switch as well as a volume control, but I will leave you to either work this out for yourself, or find someone who can! This type of connection will also allow you to make good quality recordings of your own computer compositions.

## EXTERNAL SYNCHRONISATION

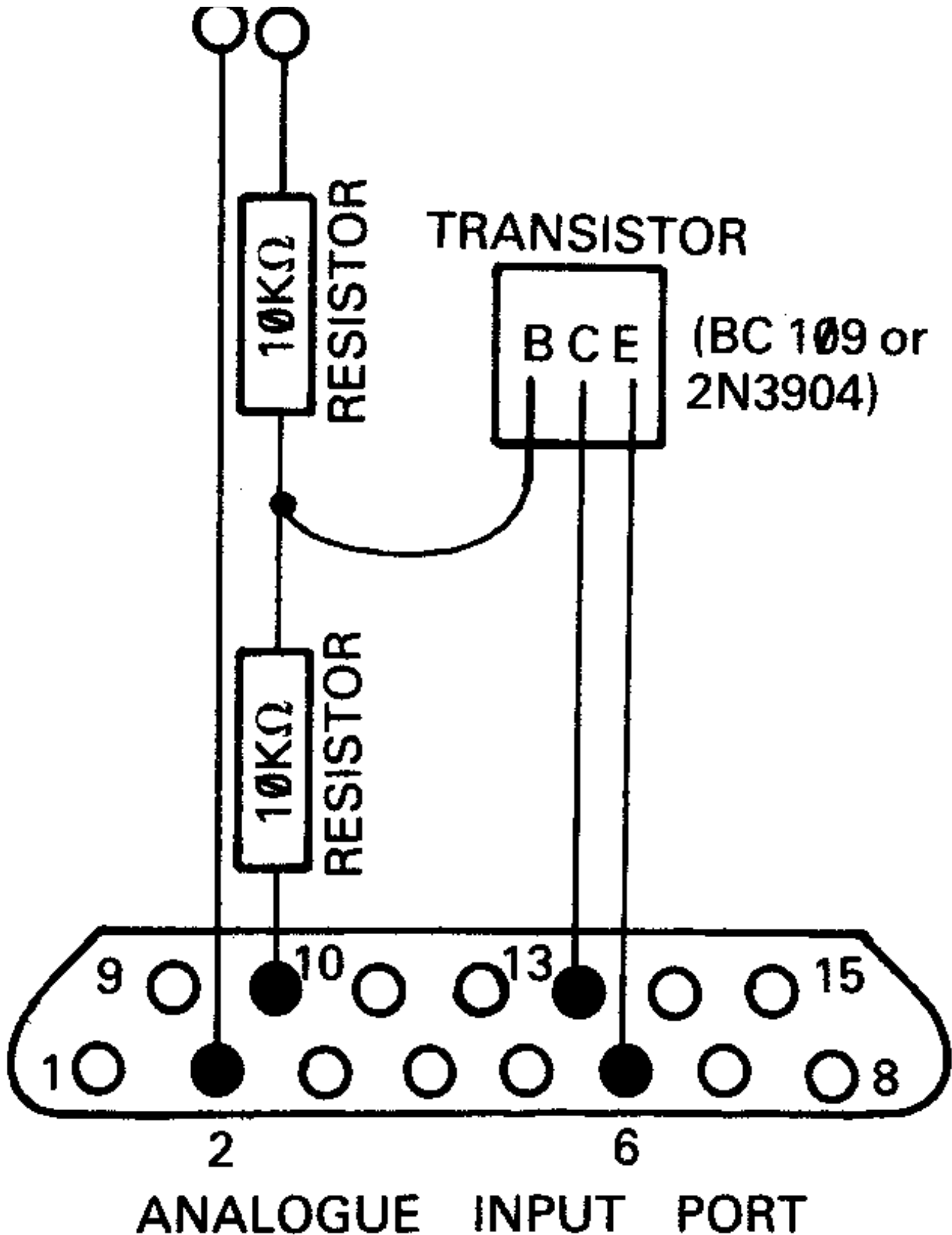
It is possible, using the analogue input port, to synchronise music programs to an external clock such as that provided by a drum machine or sequencer. This external clock would replace whatever timing loops were contained in the program and allow, say, the Drum Program to be run from a synthesiser/sequencer such as the Roland SH – 101 or Bassline or another drum machine.

To accomplish this we must first connect a few simple electronic components to the analogue input as shown in the following diagram:

### TRIGGER INPUT

**TRIGGER  
INPUT**







# ANALOGUE INPUT PORT

## ANALOGUE INPUT PORT

An NPN type transistor is used, such as the BC109 or the 2N3904. Other components include two 10Kohm resistors and a connector of your choice. The transistor should be labelled with the letters B C E (though not necessarily in that order) and must be connected in the fashion shown. The following PROCedure (PROCsync) can be then used to sense the incoming trigger pulse:

```
100 REM SINGLE TRANSISTOR SYNCHRONISER
110
120 REPEAT:PROCsync:VDU7:UNTIL FALSE
130
140 DEF PROCsync:REPEAT:UNTIL(16 AND ?&FE40)=0:ENDPROC
```

The above program allows you to test if your circuit is working correctly. The computer should ring its bell every time it senses an input. Once you have determined that your interface is operational, PROCsync can be incorporated into a program. For instance in the drum machine program PROCsync could be used to move through the sixteen steps rather than being controlled by the computer via tempo.

The circuit should be capable of sensing any trigger whose voltage exceeds 2 volts. Other factors that affect how well the circuit performs include the length of the trigger pulse and the polarity. It is up to you to experiment with the suggested set up to make it work under your particular conditions. At this point we must leave our final topic, as it becomes more a matter for the circuit-builder than the budding micro-musician.

## 12: All Fine

We have now come to the end of our voyage of discovery into ways of making music on the BBC Microcomputer. Of course, we have by no means covered all the possible and useful methods and techniques which can make our quest more enjoyable. Instead, I have imparted to you what I consider to be a good starting point for further explorations of your own. My choice of subjects and programs has been made from the standpoint of a professional musician - and computer enthusiast. All I can hope is that the material contained in these pages will have inspired you to experiment further in this totally fascinating field of pleasure and endeavour.

It should now be possible for you to combine and expand various programs in order to create practical musical tools in whatever area grabs your interest most firmly. Some of the graphics techniques used in Chapter Seven could greatly enhance the sequencer programs contained in the Applications chapter. Autocompositional techniques could well provide bass accompaniment to the three-note organ program. The drum synth program could be combined with the microcomposer program, in order to create rhythm and melody parts synchronised together. As you can see, the possibilities are endless!

Of course, there are also various other techniques which I have not had the time or space to explore in depth which should be tried at your leisure. It should be possible not only to step a synthesiser program externally, either in the fashion I indicated via the analogue interface or else employing the user port, to allow interconnection and synchronisation between synths, drum machines or other Micros (BBC or otherwise). The opposite could also be tried, i.e. triggering a drum machine from pulses generated at the user port or one megahertz bus. The possibilities are within your grasp now, so forge ahead and expand the musical cosmos.

I also need to mention again that one of the major limitations of the BBC SOUND command is the poor resolution possible for the timing of duration. As stated earlier on, for most practical musical purposes 1/20th of a second is, generally speaking an insufficiently small timing increment, but in many of the programs included in the text I have made do with this limitation, for the sake of additional simplicity and clarity. All the programs that fall into this category are intended as a demonstration of a particular programming technique and should therefore be altered to incorporate one of the other timing methods when it comes to practical use.

My autocompositional method is based on a thorough practical knowledge of musical composition and improvisation. This knowledge was used to provide the computer with a set of rules to constrain its random number generator and choose musical-sounding note patterns. A more mathematical, although still artistically manipulated, technique has been explored by Jim McGregor and Alan Watt in their excellent book, *Advanced Programming Techniques for the BBC Micro*. The authors employ first,

second and third order probability tables of various melodies. These tables tell the computer the most likely preceding and subsequent notes and thereby constrain the computer's random choice of notes in a different way. Since I saw no point in duplicating their already published work, I have presented an alternative method which relies on a more directly pragmatic and practical approach to the technique.

Autocomposition just about touches on a currently fashionable area of computer programming, namely artificial intelligence and expert systems. Might it be possible, rather than teaching the computer your own set of musical rules, to let it develop its own? You would then only have to act as a computer critic or A and R man giving the BBC the thumbs up or thumbs down for each new work it produced. An exciting and daunting thought, and, copping out somewhat, I will leave entirely up to you as to precisely *how* you might go about tackling this mammoth task!

One of the important subjects I have barely touched on in this study is that of interfacing. I make no excuses for this omission, as it is a large and complicated subject in its own right. There is, however, one type of interface which I should mention before concluding - and that is the MIDI interface.

The reason why this is worth mentioning at this late point in the book is because, by the time you read this, interfacing devices using this new standard are likely to be available for the BBC Micro. This will make it possible to drive keyboards such as the Yamaha DX7 and the Sequential Circuits Prophet T8, as well as various drum machines, direct from the computer, using software based on the microcomposer program and similar techniques. Since easy access to these interface devices is imminent at the time of writing, and they should be readily available at a reasonable price, I came to the conclusion that it would be redundant exploring into the electronic area of chips, transistors, bits and bytes. After all, once commercially available, it will be possible for you to adapt your existing musical software to run these new interface devices. For now, you can sit back and flex your new-found musical muscles using the internal sound chip!

So let me bid you farewell for the moment and leave you in the hope that you will enjoy RUNning and experimenting with the programs in the book just as much as I enjoyed writing them! Computers are the future of music, welcome to the future...

## Useful Reading

### Magazines:

*Acorn User* - May 83: 'Auto composition'

*Acorn User* - Dec 83: 'Envelope Shaper program'

*Acorn User* - April 83: 'Auto composition'

*Electronic Soundmaker*: Often has interesting articles relating computing to music.

*The Micro User* - Jan 84: Note Play program plus general Sound advice

*The Micro User* - Dec 83: Sound advice (an extended series of articles with many helpful hints and tips)

*Electronics and Computing* - July/Aug 83: A two part series on how to build a digital to analogue synthesiser interface for the BBC. An excellent constructional project for the more technical amongst you.

Books:

*The BBC Micro Book* by Jim McGregor and Alan Watt

This book has a clear introductory chapter on SOUND and ENVELOPE Advance Programming Techniques for the BBC Micro. It deals with multi-channel synchronisation, plus a mathematical approach to automatic composition using probability tables.

*Computers in Music* by John Hammond

A comprehensive coverage of computer-based musical instruments from the Casio to the Fairlight.

*The Advanced User Guide* by Bray, Dickens and Holmes

A vital tool for further exploration into interfacing techniques. This book is filled with useful technical information about the BBC.

If you have any problems finding information about computer music and the like, there is a company who keep a library of current books, leaflets, technical manuals, etc. They are called ESSP. (Electronic Synthesiser Projects) and can be contacted at the following address: ESSP, The Sound House, PO Box 37b, East Molesey, Surrey, KT8 9JB  
Telephone: (01) 979 9997.

## Index

---

# Index

## advanced

- music writer program 115–125

- sequencer programs 177–196

alarm clock (program) 32

Amazing Grace (program) 23–24

American chord nomenclature 141

ambulance (program) 27

ampersand 55

amplifiers 230–231

## amplitude

- demonstration program 12

- envelope graph 40

- parameters (ENVELOPE) 33

- parameter (SOUND) 25–27

- sampling program 210

applications 147–196

arpeggios 79–81, 134

ASCII codes 105

zero 43–52

## characters

- on-screen position 109

- redefining character shapes 105

- user defined 105–128

chords 80–81, 85–100

- in auto composition 133, 136, 141, 143

- American chord nomenclature 141

- Chord organ program 161–162

- sound layering 52

chorus 83

## chromatic

- keyboard (program) 16–17

- scale 47, 79

crochets, drawing 106–115

clef (programs) 102–104

clock diagram 87

close encounters (program) 35

- assembly language 185–196
- attack parameter 33–34
- automatic composition 129–146, 170
- auto repeat 17
- auto repeat delay 17
- avant garde music (programs) 130

- Bach (programs) 136–146
- bar chart (program) 210–211
- bars 76, 143
- bass clef diagram 71
- beats
  - first beat in the bar emphasis 143
  - per minute 20
- beep 68
- bell 32, 68
- Blood and Sand (program) 89–94
- blues sequence 142, 146
- bomb (program) 40–42

- calibration (program) 212–213
- changing key 15
- channel
  - parameter 56–58

- Clypso 2 (program) 170–173
- C major
  - arpeggio diagram 79
  - chord programs 56
  - on stave 73
  - scale programs 14–15
  - triads 141
- constant amplitude 32, 40
- continuation parameter & programs 61–63
- Crab Cannon (program) 195

- DATA statement 14, 15
- decay parameter 34
- digital
  - drum computers 199
  - synthesisers 204–206
- disco lights effect 211
- displaying music 101–128
- Doppler frequency shift simulation 27
- Dr. Rhythm programming 198
- drawing musical notation 101–128
- drum
  - in auto composition 143
  - machines 197–199

## 238 Index

- program 50–52
- sounds 50
- synth program 153–160
- dummy sound command 61
- duration
  - demonstration program 13
  - notation 75–79
  - of note program 13
  - parameter (SOUND) 20–25
- edit mode, advanced sequencer program 178–185
- Electron Microcomputer 32
- E minor arpeggio 80
- envelope
  - command 28–42
  - envelope shaper program 35–39
  - graph 28
  - number 30
  - parameters 30–42
- extended SOUND statement 55–63
- external amplifiers and recorders 230–231
- joysticks 207–209
- keyboard
  - musical keyboard program 16–17
  - recognition program 220–227
- keys 73
  - changing key 15
  - key signatures 74
- layering sounds 52
- length *see* duration
- light device 211
- Linn drum 199
  - synth program 153–160
- memory 53
- menu, advanced sequencer program 177
- metronome (programs) 20–22, 77–78
- middle C frequency 13
- modern music 197–206
- monophonic synthesisers 199–201

- external clock 231
- external synchronisation 231–232
- fanfare 54
- files (program) 228–230
- flush parameter 58–60
- format
  - ENVELOPE statement 29–30
  - SOUND statement 11
- four note chords 47
- frequency *see* pitch
- games 48, 54
- graphics 101–128, 211
- gun (program) 33
- Hall of the Mountain King (programs) 19–20
- Happy Birthday (program) 65–68
- harmonics 47
- harmony 65, 85–100
  - program 57
- hexadecimal notation 55, 67
- horizontal scroll 219
- monophonic by musicians 177–181
- multi-part music 85–100
- multi-channel composer 177
- multiple part tune synchronisation 94–100
- music
  - autocomposition programs 131–134
  - graphics 101–128
  - modern music 197–206
  - music writer programs 109–128
  - styles 82–84
  - theory 70–84
  - tuition program 213
- musical keyboard (programs) 16–17
- natural minor scales 134
- noise
  - channel zero 43–52
  - demonstration program 44
  - noise effects (program) 48–50
  - noise 3 scale demonstration (program) 47
  - noisy keys (program) 45–46
- note
  - in autocomposition 133, 142
  - duration program 13
  - duration symbols 75



Igor tune (program) 164–170  
 improvisation 129–146  
 increment step diagram 100  
 interface devices 207–232  
 interval 73  
 Irish reel (program) 179–180

## parameters

amplitude 11, 25–27  
 channel zero 43  
 duration 20–25  
 ENVELOPE 30–42  
 pitch 13–14  
 SOUND 11–27, 55–63  
 patriotic program 148–153  
 pentatonic scales 134  
 percussive musical sounds 43–52  
 peripherals 207–232  
 piano  
   diagram 71  
   keyboard diagram 72  
 pitch

lengths 116  
 positions 109  
 rest symbols 75

octave 73, 113  
 organ (programs) 63–65

Index 239

saving compositions 227–230  
 scales 14–20, 42, 79–81  
   in autocomposition 131, 134–135  
   Shepard scale program 163  
 screen 59, 109  
 scrolling 219  
 semitones 12, 72–73  
 sequencers 17, 202–204  
   programs 18, 24–25, 173–177  
 Shepard scale program 163  
 shot effect 33  
 shuffles 173  
 sideways scroll 219  
 silence (program) 59, 62  
 simultaneous sounds 52

- autocomposition 130
- changes diagram 31
- demonstration program 12
- notation 70–75
- parameters (channel zero) 43
- parameters (ENVELOPE) 30–31
- parameters (SOUND) 13–14
- ‘pitched noises’ 43, 46–47
- storing pitch information 67–68
- values 13, 71, 131
- play mode, advanced sequencer program 178
- pneumatic drill effect 32
- polyphonic
  - organ program 63–65
  - synthesisers 202
- pop song (program) 83
- position of screen characters 109
- programmable characters (diagram) 123
- queue, SOUND statements 53–55
- random music generation 12, 42

- single transistor synch (program) 232
- sizes of clefs 104
- snare drum 198
  - programs 50–52
- SOUND
  - chip 29, 44, 230
  - command 11–27, 53–69
  - effects 43–52
  - graph 29
  - layering 52
  - on/off 69
  - parameters 11–27, 55–63
  - queue 53–55
  - sampling, synthesisers 205
  - synchronised with screen display 59
- Star and Spangled Banner (program) 62–63
- stave
  - programs 102, 103–104
- step length parameters 30
- stick fun (program) 209
- storing
  - musical information 19
  - pitch information 67–68
- styles of music 82–84

- redefining character shapes 105
- recorders 230–231
- release parameter 34
- repetition 82
- rests 26
  - autocomposition 142
  - demonstration program 26
  - programs 62
  - total silence (program) 59–60
- rhythm
  - control 20
  - variation 19
- road works (program) 32
- Rule Britannia (program) 148–153

- sustain parameter 34
- swing 173
- synchronisation
  - external 231–232
  - multi-part music 85–100
  - parameter 60–61, 65
  - screen display with SOUNDSs 59
- synthesisers 199–201, 204–206, 211–219
- tape recorders 209–211
- tempo 20, 76, 113, 115, 173
  - autocomposition 143
  - tempo terms defined 76
  - tempo/time duration table 22

## 240 Index

- text window 113
- thunder (program) 48
- three note organ (program) 63–65
- three-part harmony (program) 67
- time
  - base 22
  - signatures 76

- tempo/time duration table 22
- timing 23
- tones 43
- treble clef 71, 102–103
- triad chords 141
- tune player (program) 125–126
- two clef diagram 72
- two part tune synchronisation 85–94

- user-defined characters 105–128

- values

  - ENVELOPE parameters 30–42

  - SOUND parameters 11–27, 55–63

- varying sequence length 25

- Vermillion Sands (program) 94–99

- vibrato 57

  - program 33

- violin practise (program) 15

- volume *see* amplitude

- waltz (program) 25–26

- warning bleep 68

**white noise 43**

**writing music (programs) 109–128**

---



**P E R S O N A L**  
**COMPUTER**  
COMPUTER **NEWS** LIBRARY

**IAN RITCHIE**

**BBC MICRO:**

**MUSIC  
MASTERCLASS**





# **YOUR MICRO AS MUSIC MACHINE!**

The world of computer music is explored through the medium of the micro in this, the first book produced by a professional musician for the Beeb. Starting with the essentials of programming sound and music, Ian Ritchie harnesses bytes and beat to show you the way to the harmony of clefs and chips that is modern electronic music.

At your command the versatile BBC Micro can be drum machine or synthesizer, instrument or interface, component or composer - **Music Masterclass** gives you the tools and techniques you need to turn your micro into a music machine.

Ian Ritchie introduces musical notation and the theory of chords and harmony and investigates each of the complex programming parameters of BBC BASIC's sound commands, before developing the Micro's music applications. These provide auto-composition, sequencer, synthesiser and drum machine facilities, and the potential of the BBC is compared to the performance and characteristics of commercial machines.

**Music Masterclass** goes on to describe and explore the simple interfacing techniques which illustrate how the BBC's computing power can be incorporated into more complex systems as well as the potential of graphics displays.

ISBN 0-330-28673-0

U.K. £5.95

9 780330 286732