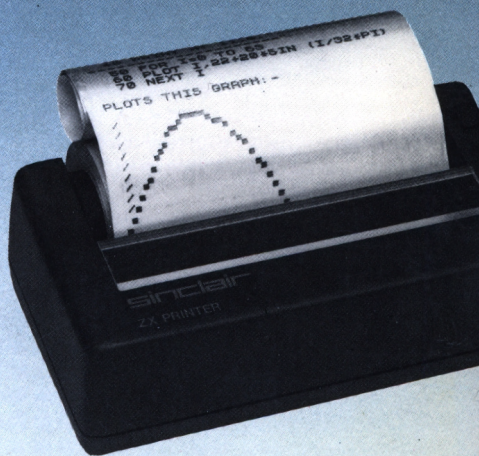


**MORE REAL
APPLICATIONS
FOR THE**

ZX 81 and ZX Spectrum



Randle Hurley

*More Real Applications
for the ZX 81
and ZX Spectrum*

Macmillan Computing Books

Advanced Graphics with the BBC Microcomputer Ian O. Angell and Brian J. Jones

Advanced Graphics with the Sinclair ZX Spectrum Ian O. Angell and Brian J. Jones

Assembly Language Programming for the BBC Microcomputer Ian Birnbaum

Advanced Programming for the 16K ZX81 Mike Costello

Beginning BASIC Peter Gosling

Continuing BASIC Peter Gosling

Practical BASIC Programming Peter Gosling

Program Your Microcomputer in BASIC Peter Gosling

Codes for Computers and Microprocessors P. Gosling and Q. Laarhoven

Microprocessors and Microcomputers — their use and programming
Eric Huggins

More Real Applications for the Spectrum and ZX81 Randle Hurley

The Sinclair ZX81 — Programming for Real Applications Randle Hurley

Z80 Assembly Language Programming for Students Roger Hutton

Digital Techniques Noel Morris

Microprocessor and Microcomputer Technology Noel Morris

The Alien, Numbereater, and Other Programs for Personal Computers — with notes on how they were written John Race

Understanding Microprocessors B. S. Walker

Assembly Language Assembled — for the Sinclair ZX81 Tony Woods

*More Real Applications
for the
ZX81 and
ZX Spectrum*

Randle Hurley

M

© Randle Hurley 1982

All rights reserved. No part of this publication may be reproduced or transmitted, in any form or by any means, without permission.

First published 1982

Reprinted (with corrections) 1983

Published by

THE MACMILLAN PRESS LTD

London and Basingstoke

Company and representatives

throughout the world

Printed in Great Britain by Unwin Brothers Limited,
The Gresham Press, Old Woking, Surrey.

ISBN 0 333 34543 6

Contents

	<i>Page</i>
1 Introduction	1
2 Basic Differences	4
3 At Least As Good	7
4 Till	20
5 A File Handling Survival Kit	36
6 Super Storage and Statistics	47
7 Cricket	69
8 Draw and Move	93
9 Effective 1K Programming	114
10 Building Block Programming	129
11 Music	150
Appendix ZX81 Hardware Modifications	171

The idea for this book must be the result of my contacts with other ZX enthusiasts. It is difficult to trace the material back to source, but I am very grateful to all who helped, consciously or otherwise.

Particular thanks must go to two people. John Watson supported the venture from the start and contributed many helpful suggestions. It was he who provided the final solution to the ZX Merge problem in the form of a ROM address which allowed the computer to re-initialise itself. Linda, my wife, provided the valuable criticism which has allowed me to keep the book as free from jargon and as readable for the beginner as it is. Without her support in the bleak periods and her constant attention to detail the final result would have been much less good.

However you helped, thank you.

Cover photographs of Sinclair equipment kindly supplied by Sinclair Research Ltd.

1 Introduction

There are two ways of approaching this set of programs for the Sinclair ZX81 and Spectrum. Firstly, you will find a collection of software which will do real work in a variety of situations. Secondly, programming is much more enjoyable when you understand what is going on and can see how to solve computing problems for yourself. This requires a lot more than a set of listings. Between the sets of listings you will find several chapters which explain the thinking behind the software and the way it is put together. If you follow the second approach to the material you will gain an appreciation of how the ideas and techniques presented here can be built into your own programming and allow you to produce software to a professional standard for your own use.

These are all working programs which do a wide range of complicated jobs. Often such large, complex programs are presented in one large block. It is all too easy to omit the odd line, to lose the thread of the underlying idea and to become very tired while keying such programs into the computer. Here the programs are broken down into short sections each of which does a single small job. These sections are seldom more than ten lines long and each is fully explained before it is presented. This approach reduces the fatigue experienced when entering programs and as a result there is less likelihood of mistakes being made.

Many of the individual lines of the programs have explanatory notes printed alongside. Each GO SUB is supported with a note on the action of the subroutine and each GO TO is accompanied by a reminder of what is to be found at the destination. The reader is not expected to have a perfect memory and if a similar line crops up a little later in the listing then the explanation is repeated. The programs are supplied with operating instructions because it is so easy to forget how to drive the packages and so difficult to find the necessary information in the body of the chapter. The hope is that readers will feel fully supported as they work towards an understanding of the programs and the ideas on which they are based.

The packages were written to run on both the ZX81 and the Spectrum as far as seemed appropriate. It seemed unnecessary to make Spectrum translations of the 1K programs for the ZX81. These were produced to explore the limitations of the 1K computer and to overcome them as far as possible. *Draw and Move* is basically a ZX81 idea while the demands of *Music* is beyond the capabilities of

the ZX81 without a lot of plug-in extras, so it has been written for the Spectrum alone. Apart from the set of programs written specifically for the 1K ZX81, all the software needs the larger, 16K computer. ZX81 users will not be hampered by lack of memory beyond 16K. Spectrum users, on the other hand, may be surprised by the small amount of memory which remains to them after the computer has booked its own needs. Only a little more than half remains. For this reason the rather ambitious Statistics program really needs a 48K Spectrum to show off its power even though a 16K ZX81 is quite large enough. The 16K Spectrum will run the program but will have room for only about 1½K of data.

There is no reason why ZX programs should not be at least as good as software written for other computers, and the chapter *At Least As Good* looks at a variety of ways of making your programs as professional as possible. The techniques described are to be found throughout the rest of the book and many have been used to build TILL. TILL is a program which will turn your ZX81 or Spectrum into a sophisticated cash register which prints till slips, daily accounts and, on the Spectrum, produces a range of the bleeps made by commercial tills.

Handling files of data is central to many of the jobs which computers do. *A File Handling Survival Kit* shows how data files can be created, up-dated and processed easily on either of the computers. The basis of an idea on making surprisingly efficient use of the available memory is given in this chapter and then developed thoroughly in the next two. *Super Storage And Statistics* really makes the computer work for its living. The storage technique used allows the ZX81 to store in its 16K of memory as many numbers as a 64K computer could store using conventional storage methods. The Statistics program allows some quite complicated mathematical treatment of the stored numbers, while CRICKET treats its data a little more lightly and produces all the batting and bowling statistics for a cricket club during a season, putting all the players in order of merit.

Draw and Move looks at a completely different aspect of computing, the production of a series of frames for an animated sequence. The program concentrates on making the production and editing of the frames as simple as possible. *Effective 1K programming* shows another aspect of animation. Almost all of the 1K programs in the set have some degree of animation. Some of them produce quite startling effects on the screen which might have been thought impossible in 1K. The chapter concentrates on the memory saving techniques which made the programs possible.

Merge is a new function, available on the Spectrum, which allows the user to add a program on tape to a program already stored in the computer. This function is essential to the implementation of the programming strategy presented in the rest of the book. If ZX81 users are to implement the strategy then they have to have Merge and this is one of the service programs presented in *Building Block Programming*. Merge 81 and some of the other routines are written in machine code for extra speed and

flexibility. This need not worry readers who are new to machine code because the entry of the programs has been made very straightforward. A couple of games programs were written for the chapter, to demonstrate the effects of the service programs and these have turned out to be worth including for their own sakes.

The final chapter *MUSIC* deals with a set of programs which allow the Spectrum to have its head. The programs make use of the sound and the high resolution graphics on the new machine to produce musical ideas on the computer. DATA, READ and RESTORE are all used extensively in these programs and this will be of interest to ZX81 users who are new to these statements. The programs can be used separately or merged into one large program if a 48K Spectrum is available. The ZX81 is catered for. The ZX81 program which the others developed from is included.

2 *Basic Differences*

Some of the software intended for one of the ZX computers was not rewritten for the other machine because it seemed inappropriate to do so. The little 1K programs in Chapter 9 are designed to explore the limitations of the 1K computer and to overcome these as far as possible. The MUSIC programs presented in Chapter 11 make impossible demands on the ZX81. Both these collections are available for the one machine only but the majority of the software which follows will run happily on either computer.

Where long programs have to run on both the ZX81 and the Spectrum then the style of programming has been adapted to suit the ZX81. To have adopted the multi-statement line approach more suited to the later machine would have meant two complete sets of listings. While the one statement per line approach does not make for the most efficient use of memory on the Spectrum, the ZX81 style software will run with only minor modifications. The few lines which are machine specific are given in the ZX81 form in the main listings and the Spectrum versions are shown alongside.

There are differences in the way in which the BASIC statements are interpreted by the two computers and some of these are not immediately obvious. The rest of this short chapter will deal with one or two of these differences which caused most trouble during the translation and de-bugging of the programs written first on the ZX81. First, though, a quick look at some of the decisions which had to be taken before a single Spectrum key was pressed in earnest.

When the Spectrum is switched on the keys produce lower case letters. To print in upper case the CAPS LOCK key has to be used. Should we use upper or lower case when writing the programs? The upper case is clearer in the listings but there is a danger that users of the programs will not realise that the computer has to be switched into upper case before the packages can be used. Variable names will be recognised in either upper or lower case but as soon as the computer has to compare strings of characters then trouble is possible. If the Spectrum is expecting an "A" it will reject an "a". The risk of such trouble seems to outweigh the slight advantage of producing the listings in the clearer upper case and so lower case has been used in all of these Spectrum programs.

To ZX81 users, a 16K computer is a big machine. The 16K ZX81 will hold all the program lines in the STATISTICS program given later and still have room for 9000 numbers. The same program, fed into the 16K Spectrum will only allow the storage of about 1500 numbers. The memory which was available for data storage in the ZX81 has been pinched by the Spectrum. The machine needs all the extra space, about 8K, to hold the information it requires to run a high resolution, colour screen. All of the software which follows will run in the 16K Spectrum but some packages need some extra space to really do themselves justice. Compared with the two ZX computers the smaller Spectrum is nowhere near as cramped as the 1K ZX81 but for the real applications which this collection of software is designed for, a little extra space would be useful from time to time. The 40K left to us out of the 48K machine's memory is five times the 8K available to us on the basic model and the cost of the memory upgrade seems well worth spending.

Now to the differences in the BASIC. They are few and small but worth spending a little time on to avoid wasting a lot of time in de-bugging programs translated from the ZX81. First a difference in the way that TAB is interpreted. The ZX81 would accept the following lines and produce the first display:

```
10 PRINT AT 10,10;"HELLO"
20 PRINT AT 9,10;"HELLO THERE"
;TAB 16;"THERE"
```

ZX81

Spectrum

```
HELLO THERE
HELLO THERE
```

```
HELLO THERE
THERE
```

The Spectrum seems to have lost part of line 10 which was printed on the screen first. This is a small change but not a little inconvenient at times. It is better to stick to the PRINT AT statement when changing a screen display or you may find yourself taking away more than you are adding.

The Spectrum will take multi-statement lines. Those of us who are used to the ZX81's single statement lines may not realise the pitfalls associated with the more complicated line structure. The main difficulty is associated with the use of the IF statement. In a line such as:

```
1000 IF a$ = "abc" THEN LET b$ = "def" : GO TO 90
```

the computer will ignore the GO TO instruction if a\$ is anything but "abc". If the condition is false then no more statements in the current line are read. The computer looks at the next line for its next instruction. Line 1000 will have to be split if the GO TO is to be read every time, and not just when the contents of a\$ are "abc".

```
1000 IF a$ = " abc " THEN LET b$ = " def "  
1005 GO TO 90
```

The general rule is: if you wish the whole of the line to be read and acted on no matter what happens then the IF should be in the last statement of the line.

The last of the differences looks very innocent but can be a source of trouble which is difficult to trace. The trouble is that the Spectrum is a little more thorough than the ZX81 in the way that it checks through lines. The ZX81 would be quite happy to work with a line like this one:

```
1000 IF A$ = "" OR VAL A$ = 20 THEN GO TO 1500
```

If the VAL instruction will not turn the contents of a\$ into a valid number then the Spectrum will give an error message and stop the run. Splitting the line into two is the answer.

```
1000 IF a$ = "" THEN GO TO 5000  
1005 IF VAL a$ = 20 THEN GO TO 5000
```

These are the only differences in the interpretation of the BASIC commands shared by the two computers which emerged. Doubtless there are others and these will come to light eventually but, generally, the skills that are built up on the ZX81 will transfer very smoothly to the Spectrum and the extra facilities available on the later machine will allow the user to develop further computing skills. This is far from saying that the power of the ZX81 has been exhausted by the programs in this collection. There seems to be a lot more that can be squeezed out of the little black box no matter how far you push it.

3 *At Least as Good*

That's what ZX programs should be, at *least* as good as programs produced for other computers: this is not an impossible target for the ZX enthusiast (especially as the quality of some professionally produced software leaves quite a lot to be desired!). As always, it is difficult to pass an objective opinion on your own work, especially if it has worked perfectly since it was finished, doing all that is required of it. As soon as you produce a piece of software which saves you time and effort, your friends and colleagues will want to use it to do the same for themselves. You hand over a program which has run faultlessly for months and it crashes within minutes. Friends and colleagues don't have the same understanding of the program that you do. They don't understand that a black blob at the bottom of the screen means that a number is needed, if there are no quotes on either side. They make a few tentative but inappropriate keystrokes and produce the full horror of the program listing on the screen. Not a sight to inspire confidence in a neophyte.

This chapter starts with a series of routines which will prevent the majority of the crashes to which inexperienced users are prone, and then goes on to examine the problem of producing screen displays of professional quality. This is intended not as a cherry on the top of the computing cake, a set of unnecessary frills, but as a series of standards which will build confidence in the new user and make programs more comfortable to use by the experienced user as well. The ideas put forward in this chapter are used in the programs throughout the rest of the book, and readers will be directed to examples of the techniques, so that they can see the benefits directly.

The menu which drives the program through its repertoire of activities is on the screen for a good proportion of the running time. The layout of this 'shop window' part of the program needs particular attention. When the menu has been put together it must not crash, no matter what keystrokes are made in response to it, and so a routine which looks after this need is given. The routine, in turn, dictates the structure of the rest of the program, making it both neater and easier to read. This looks like a good place to start.

Here are two menu pages, together with the program lines which produced them. The effort needed to improve on the first display is small compared with the increase in quality.

SIMPLE MENU

```
10 PRINT "TO ENTER TIME SHEET  
DETAILS KEY 1"  
20 PRINT "TO PRINT WAGE SLIPS  
KEY 2"  
30 PRINT "TAX ACCOUNTS KEY 3"  
40 INPUT A  
50 IF A=1 THEN GO TO 500  
60 IF A=2 THEN GO TO 900  
70 IF A=3 THEN GO TO 1350
```

```
TO ENTER TIME SHEET DETAILS KEY  
1  
TO PRINT WAGE SLIPS KEY 2  
TAX ACCOUNTS KEY 3
```

Most of the extra code needed to produce the improved menu, brings other benefits besides the better display. The screen is kept tidy during the whole of the program, and crashes due to inappropriate keystrokes are prevented. Let's have a look at the method used to avoid trouble first.

The important bit, from the point of view of preventing crashes, is the section between lines 935 and 945. The information is fed into a string variable, A\$, because any keystroke will be

BETTER MENU

```
900 CLS
905 PRINT TAB 8;"ENGINEERING I
910 PRINT AT 2,7;"TAX AND WAGES
SYSTEM"
915 PRINT ,,"1 ENTER TIME SHEE
T DETAILS"
920 PRINT ,,"2 PRINT WAGE SLIP
S"
925 PRINT ,,"3 PRINT TAX ACCOU
NTS"
930 PRINT ,,,, "SELECT OPTION (1
, 2 OR 3)", "THEN PRESS NEWLINE"
935 INPUT A$
940 CLS
945 IF LEN A$<>1 OR A$<"1" OR A
$>"3" THEN GO TO 905
950 GO TO VAL A$*1000
```

ENGINEERING INC

TAX AND WAGES SYSTEM

- 1 ENTER TIME SHEET DETAILS
- 2 PRINT WAGE SLIPS
- 3 PRINT TAX ACCOUNTS

SELECT OPTION (1, 2 OR 3)
THEN PRESS NEWLINE

■

accepted as part of a string, whereas letters fed into number variables often cause trouble. Once the information is inside the machine it can be checked and, if necessary, dumped before it can do any harm. The only legitimate input in response to this menu is a single character and so this is the first check that is made. If LEN A\$<>1 then the machine will be sent back for another go by line 945. Similarly, if the code of the input is outside the range of codes of the valid options, the information will have to be dumped and re-entered.

No matter which key is pressed, the screen will be cleared and this is an indication to the user that something has happened, as a result of the keystroke. The inference is that if the menu is re-printed, the entry must have been unsuitable. If an appropriate key is pressed, the character will be turned into a value by line 950, and this value will be used to direct the computer to the correct section of the program. Because of this line, the program is moulded into neat, 1000 line sections, which make it easy to read and find your way round in. Few programs have more than nine sections but if necessary line 950 can be amended to:

```
950 GOTO VAL A$*500 or even GOTO VAL A$*250
```

to cope with nineteen or thirty-nine sections.

It may seem strange that the line numbers of the menu start at 900 and are in steps of five not the usual ten. One implication of the pattern of program design which is emerging is that the sub-routines will have to be located at the start of the program, between lines 2 and 899. Line 1 will always have to be:

```
1 GOTO 900
```

to direct the machine to the menu. This leaves plenty of room for the sub-routines and suggests the possibility of adopting the same pattern of programming for these. If the same routines are to be needed in several programs, it seems reasonable to use the same line numbers in each program. This is an important point and will be discussed more fully a little later on. So far we have accounted for the menu starting at line 900, but not for the CLS at this line. The clear screen command is used here, and at the end of the menu routine, in order to avoid having to clear away text so often in the rest of the program. Any section of program should terminate by returning to the menu. If the screen is cleared automatically at this stage, then there will be no chance of text being inadvertently left behind to overfill the screen, giving an 'out of screen' error code, 5 on the ZX81 or forcing the Spectrum to SCROLL.

The next topic to be discussed enjoys the jargon label of 'mug trapping'. Mug traps prevent the inappropriate keystrokes made by 'mugs' from getting through to the important parts of the program, where they could cause crashes. The traps are routines which filter out and reject the input which does not fit the computer's requirements at the time. The checking line in the menu reading section is a mug trap, which only deals with single characters. We now need to consider more elaborate traps. I'm not sure that I approve of the label because I often feel the benefit of traps like these in my own programs. We seem to be stuck with the jargon so I had better put up with the indignity. More seriously, mug traps are useful, even in programs which you intend to keep entirely for your own use, and especially if a lot of data has to be fed in. The longer you have to spend doing repetitive tasks, the

more tired you will become, and the more mistakes you will make. Routines like these will improve the quality of your temper at the end of lengthy sessions, feeding the computer with data.

The following routines prevent every crash that I have had the misfortune to witness apart from those caused by the use of the BREAK key and by moving the ZX81's RAM pack on its contacts. These two major problems will have to be covered by a short education job on all new users of the programs. The problems which need to be covered are:

- a) keying a response to a menu which is not on offer, (this has already been covered);
- b) keying letters when numbers are expected;
- c) mixing letters in with numbers e.g. 1.03 using the letter "O" instead of zero, "0";
- d) entering two decimal points in the same number;
- e) feeding in values which cannot be used by the computer e.g. PRINT AT co-ordinates of 30, 10; instead of 10, 30;
- f) generating a value which exceeds the capacity of the computer e.g. trying to divide by zero.

The menu hardly ever needs more than one character to drive it so the check used in the last section was applied after all the multi-character input had been rejected. More often than not more than one character has to be checked and the simple means of checking the CODE will not work because the code returned by the computer is that of the first character only. If we adopt the habit of feeding all input into string variables then the machine will be happy to accept any character without complaint. Longer strings will have to be checked, character by character. The system used here allows all the characters to be fed in before checking. If it is required to check the input, a character at a time, then there is a system which does just this in the TILL program on page 24 lines 200 to 290. For the purposes of this section let us assume that the input has been fed into A\$. The length of the string can be used to tailor the checking routine to the exact requirements of the job and so it is used as the limit on a FOR/NEXT loop in the following piece of BASIC.

```
1      GOTO 1000      by-pass the routine
10     LET X = 0      a flag, X=0 if the input is OK
11     LET Y = 0      another flag. Y=0 until a decimal point
                    is found
12     IF A$ = "" THEN LET X = 1
13     FOR J = 1 TO LEN A$
14     IF A$(J) < "0" OR A$(J) > "9" THEN LET X = 1
15     NEXT J
16     RETURN
```

The variable X is set to zero at line 10 to act as a flag. While the flag remains unset the string is perfectly acceptable but as soon as an error is detected and the flag is set to one then the input has to be dumped. If the NEWLINE key (ENTER on the Spectrum) is pressed as the sole response to the input request then this has to be ignored. Line 12 sets the error flag if the string is empty. As it stands the loop will only accept integers and rejects any decimal points. Often integers are what the machine needs and then this is the input check routine to be preferred, but when fully blown numbers with lots of decimal places are needed, then a more extensive check is needed. The routine above has to be extended to allow the input of a single decimal point. The ZX81 will happily turn the string "." into the value "zero" but asking the Spectrum the VAL of "." will make it give an error message and stop the run. Line 12 has to be extended a little to allow for this and make the routine work just as well on both computers.

```
12     IF A$ = "" OR A$ = "." THEN LET X = 1
```

Line 14 now has to allow for the extra character, the decimal point. This means the addition of a pair of brackets to make the logic of the line correct.

```
14     IF (A$(J) < "0" OR A$(J) > "9") AND A$(J) <> "." THEN  
      LET X = 1
```

If the decimal point flag has already been set and a new "." is found then the error flag has to be set so:

```
15     IF A$(J) = "." AND Y = 1 THEN LET X = 1
```

Just before the computer loops back for another character the decimal point flag will have to be set if this is appropriate:

```
16     IF A$(J) = "." THEN LET Y = 1
```

The routine will not cope with numbers in scientific notation but seems happy to accept any other valid number. The only way to crash the routine seems to be to press the BREAK key while the machine is checking the characters in the string. Most of the items in the check list on page 11 have now been covered but the problem of screen co-ordinates remains. A simple routine like:

```
100    INPUT A  
110    INPUT B  
120    PRINT AT A, B; "*"
```

needs a line like 115 to prevent problems.

```
115   IF A < 0 OR A > 20 OR B < 0 OR B > 31 THEN GOTO 10
```

This would prevent the machine from giving "out of range" error reports but might confuse the user who would have little idea of why the computer did not respond to the input. A more helpful solution involves the calling of a sub-routine:

```
115   GO SUB 30
116   IF X = 1 THEN GOTO 100
```

And at line 30:

```
30   LET X = 0
31   IF A < 0 OR A > 20 OR B < 0 OR B > 31 THEN LET X = 1
32   IF X = 0 THEN RETURN
33   PRINT AT 21, 0; "CO-ORDINATES WRONG, TRY AGAIN."
34   PAUSE 250
35   PRINT AT 21, 0; " enough spaces to over-print the
    last message"
36   RETURN
```

When PLOT is used on the ZX81 the above routine needs the limits to be multiplied by two but the PLOT command on the Spectrum has screen co-ordinates in the range: 0 to 255 and 0 to 175.

Things are a lot simpler when the strings do not have to hold numbers. The only problem which might arise, is that a string of characters fed in from the keyboard, may be longer than the space set aside for it, and the system of procrustean assignment helps here. If the ZX81 is told the number of characters to be allocated to a string, by means of a DIMensioning command, then extra long input is trimmed to fit and short strings of characters are padded out with spaces. Procrustes was an inn keeper who boasted that his beds always fitted his guests. He made sure that this was the case, by chopping off the feet of the tall ones and stretching the shorter visitors on a rack until they were long enough. DIMensioning strings makes the ZX81 perform the same savage tasks.

When we were discussing the structure of the menu, the design of the rest of the program being dependent on the menu was mentioned, and it was suggested that this would have some far reaching implications. The possibility of sections of code being used in more than one program is just one of the possible benefits which might arise. There seems little point in re-designing program segments which do the same, or very similar jobs, as segments from earlier programs. If the program is in the form of neat, 1000 line sections, then the job of loading a standard section into the ZX81, before the rest of the program is added, is fairly straightforward. Later, in Chapter 10, there is a piece of code which will allow programs to be merged together in the ZX81 without loss of

one or the other. This will allow the ideas put forward in this chapter to be implemented on the ZX81 as well as on the Spectrum which has a merge facility built into its BASIC. This approach to programming should allow the programmer to develop a style of coding which remains fairly constant from one program to the next. The problem of debugging programs should ease as a segment which works well in one package should do just as well in another. If an improvement to one of the segments is discovered then it should be an easy task to upgrade the segment in all the programs which include it.

If, as seems likely, all the sub-routines are to be between line 1 and line 900, then it seems reasonable to make as many of these as possible into a library of sub-routines, with the same line numbers in all of the programs in which they occur. This not only cuts down on the effort needed when writing a program, but makes it highly likely that the portable bits of the main program could call sub-routines at the same line numbers, in many different programs, and find the code in place.

Now to the problem of keeping the screen display professional at all times. One thing that need not concern us, is the amount of memory that the display is using. The ZX81, with the RAM pack in place, keeps the whole screen in its memory, even if there is nothing printed on it. The 1K computer's screen file can consist of nothing but a string of NEWLINE characters, to allow the maximum memory for holding the program. In the programs being considered here, the RAM pack will be in place for the majority, and so we can use the screen to its fullest extent, without worrying about the memory cost. There is no point, however, in using memory up unnecessarily, so the following could be adopted as a habit:

```
10 PRINT " THIS IS THE FIRST LINE "  
20 PRINT , , , , " AND THIS IS THE FOURTH "
```

There is no need to write empty PRINT lines at a cost of six bytes each and extra typing when a couple of commas in the PRINT lines do the same job more quickly and easily for less memory. The Spectrum will accept the two commas but an apostrophe does the same job for one byte less. Similarly, there is no extra cost for the use of inverse characters. These are easily available from the keyboard and can be used to advantage to highlight parts of the display. Inverse graphics can be very effective when used in moderation but can be rather messy if they are overdone. The ZX printer will cope with inverse graphics but the quality of the reproduction is not up to the standard that the printer manages for normal print. It might be as well to avoid inverse graphics when using the printer. When you decide to use inverse graphics consider the two blocks shown. The one set in a rectangular block with leading and trailing inverse spaces is a lot more attractive.

AN ILLUSTRATION OF THE
BLOCK PRODUCTION OF
INVERSE GRAPHICS

AN ILLUSTRATION OF THE
BLOCK PRODUCTION OF
INVERSE GRAPHICS

Printing data onto the screen is quite predictable, as long as numbers are not used. Even numbers are not too much bother, as long as they are integers, but as soon as the computer starts to produce decimal places, the display gets quite messy. Division produces up to eight decimal places, most of which are never needed. Have a look at this piece of coding which gives a nice neat line of print at first:

```
10    LET A = 8
20    PRINT " THE ANSWER IS "; A ; " POTS OF PAINT "
```

Try editing the first line to:

```
10    LET A = 8 / 3
```

and the line becomes spread over two screen lines, with a break in a word caused by the extra, unwanted decimal places. The following amendment produces just two decimal places. Substituting 1000 for 100 will make the computer print three places.

```
10    LET A = INT ( 8 / 3 * 100 ) / 100
```

But the value is nearer to 2.67 than it is to 2.66 and so the added refinement in the next version of line 10 is necessary to print to the nearest 0.01:

```
10    LET A = INT ( 8 / 3 * 100 + .5 ) / 100
```

This prints the more correct value of 2.67. This simple method allows you to dictate to the computer exactly how many decimal places you need at any time, but a line like this may cost forty bytes. The line is an obvious candidate for promotion to a sub-routine, perhaps to be included in your repertoire of standard routines. The factor which dictates the number of decimal places can be fed into a suitable variable, say DP. Once the value to be printed is safely in the variable A, then the following routine can be called.

If the value to be printed is held in the variable A, then all that is needed to mould it to the needs of the screen display, are

these few lines:

```
1000 LET DP = 1000      for three decimal places
1010 GOSUB 20           the next convenient site for the
                       routine
```

And then at line 20:

```
20 LET A = INT ( A * DP + .5 ) / DP
21 RETURN
```

The hallmarks of a good display of numbers is a column of figures, with their decimal points all underneath the ones above, and with a constant number of decimal places. Such lists are not just the demands of excessively houseproud data processors, they are easier to read and so more likely to be read accurately. See how the relative size of the values in the following lists become more readily apparent as you go from left to right. All the lists contain the same values.

<i>LEFT HAND JUSTIFIED</i>	<i>JUSTIFIED TO THE POINT</i>	<i>TRIMMED TO SIZE</i>
14	14	14.00
0.17	.17	0.17
0.00649	.00649	0.01
1006.49	1006.49	1006.49
100594	100594	100594.00
1000.796	1000.796	1000.80
1.000805	1.000805	1.00

The two routines that we have put together so far, will do everything but organise the lining up of the decimal point, to give the right-hand justification of the third column in the above table. In order to do this final part of the job, we need one more sub-routine to find the decimal point in numbers:

```
22 LET A$ = STR$ A
23 LET P = -1      P will hold the position of the
                  decimal point.
```

Next a loop to search through the string for the decimal point:

```
24 FOR J = 1 TO LEN A$
25 IF A$(J) = "." THEN LET P = LEN A$ - J
26 NEXT J
27 RETURN
```

The LEN A\$ - J bit is needed to decide where on the screen the number is to be printed. If there is no decimal point in the number, then the minus one in P does the same job. The main program should have a line like:

```
1000 PRINT TAB 20-LEN A$ + P ; A$
```

The piece of code which follows merges all these routines into a number handling routine. This is a little slow to operate in the ZX81's SLOW mode and may well be used to build a screenful of print in FAST mode. This method is used in the statistics program in Chapter 6, and allows a professional screen to build up in the shortest possible time.

NUMBER HANDLING ROUTINE

```

1 GO TO 1000
10 LET X=0
11 LET Y=0
12 IF A$="" OR A$="." THEN LET
X=1
13 FOR J=1 TO LEN A$
14 IF (A$(J)<"0" OR A$(J)>"9")
AND A$(J)<> "." THEN LET X=1
15 IF A$(J)="." AND Y=1 THEN L
ET X=1
16 IF A$(J)="." THEN LET Y=1
17 NEXT J
18 RETURN
1900 LET A=INT (A*DP+.50001)/DP
201 RETURN
202 LET A$=STR$ A
203 LET P=-1
204 FOR J=1 TO LEN A$
205 IF A$(J)="." THEN LET P=LEN
A$-J
206 NEXT J
207 RETURN
1000 PRINT "KEY THE NUMBER "
1010 INPUT A$
1020 GO SUB 10
1030 IF X=1 THEN GO TO 1100
1040 LET A=VAL A$
1050 LET DP=100
1060 GO SUB 20
1070 GO SUB 22
1080 PRINT TAB 20-LEN A$+P;A$
1090 GO TO 1010
1100 PRINT AT 20,0;"INVALID NUM
BER"
1110 FOR J=1 TO 200
1120 NEXT J
1130 PRINT AT 20,0;"
1140 GO TO 1010

```

The earlier list of numbers is printed in FAST mode by this routine in four seconds. A different approach to the problem of printing columns of numbers is built into the TILL program, and this may well suit the programming need a little better, so have a look at that one as well.

Many programs which are written to process a lot of data, build the information into the stores inside the computer as soon as they are fed in. The problem with this style of program is that it is often difficult to correct mistakes or, as in the extreme example which follows, to notice that there has been a mistake.

```
10 PRINT " INVOICES FOR 1/1/82 "  
20 FOR J = 1 TO X X would be set to the  
number of invoices  
30 PRINT " KEY AMOUNT OF INVOICE NO. " ; J  
40 INPUT A  
50 LET TOTAL = TOTAL + A  
60 NEXT J
```

The data could have been fed into a temporary array and printed on the screen. When all the information had been fed in, the user could be asked to check the figures, before they were all added to the TOTAL. This improvement is fine, but not the best possible system for ensuring that mistakes are kept to a minimum. It is easy to become mesmerised by a display and press the 'all OK' button without thinking, realising as you do so that the last figure was wrong. Too late, the information is in, and you will probably have lost the display as the computer makes room for the next set of figures. You cannot even be sure that there was a mistake, let alone remember what it was. The only solution is to dump the program, re-load from the tape, and start all over again.

Another disadvantage of a system such as described in the last paragraph is that all the information will have to be fed in again, even if only the last figure was wrong. A solution to the typing mistake problem, which may well be more appropriate in many programs, is to hold all the data in a file as it is fed in, and do all the necessary work on it in a separate part of the program. The advantage of this system is that a mistake noticed, even after all the processing is complete, will still be recorded in the files, and can be corrected before the computer is sent off to do all the calculations again. An approach like this will only be feasible if there is a finite amount of data to be keyed in, and if there is room for this amount of information. Numbers are expensive to store.

There are two programs in the chapters which follow, which use both approaches to the problem. STATISTICS shows how the problem of storing numbers in a small space can be reduced considerably; enough to give the 16K ZX81 the storage capacity of a 64K computer, using conventional storage techniques. This program stores all the information in as many separate storage sessions as the user finds convenient, and allows the processing of the data to be carried out at any time, and as often as required. CRICKET adopts the alternative approach, of giving the user frequent opportunities to check the input before it is built into the machine's memory irretrievably. The number of matches in a cricket season, played

by a club which runs many sides, cannot be predicted as extra fixtures are booked during the season, so the data structure cannot be planned at the start of the year in the same detail. If you don't know how many matches will be played, then you can't be sure how many spaces to leave.

Both these programs include features designed to make the feeding in of large amounts of data as painless as possible. If mistakes are noticed in either program then the previous, correct data, need not be dumped along with the mistake. A cursor shows the position of the piece of data being considered by the computer, and this can be moved to any place in the list without altering the contents. When the cursor points to the mistake, the user may change the record with ease.

All the features mentioned in this chapter are attempts to make the programs written for the ZX computers at least as good as programs written for other computers and better than many commercial programs which are often disappointingly 'bug-ridden'. I hope that you will find it helpful, and that you will enjoy the satisfaction that I have felt in raising the standard of my own programs. I feel quite relaxed about asking my colleagues to do their own data processing, now that I know that they are much less likely to experience the horror of a crash. I feel particularly relaxed in the knowledge that, if my colleagues can safely be left to do their own data processing, then I won't have to do it all for them.

4 *Till*

If you have an electric drill, the likelihood is that you will have an attachment or two to go with it. Most do-it-yourselfers bolt sanders or saws onto their drills and extend the usefulness of the basic tool considerably. Purpose designed sanders and power saws are preferred by some professionals but, for the handyman, they are less useful because they lack flexibility. The ZX computers seem to be the computing equivalent of the power drill. They are inexpensive but the range of work that they can do is wide. Good "bolt on" programs allow good quality results to be produced. TILL is a program which will transform your ZX range into a flexible cash register which can be tailored to any business and outperform many commercial tills.

The aim of this book, and of its predecessor, is to explore the ZX range and produce programming ideas. It is important not to leave the ideas in mid-air but to follow them through to a conclusion and produce a finished piece of work which will do a useful job. Readers who don't need a till may be tempted to skip the rest of this chapter and go on to others which have more immediate benefits to offer. Read the next few paragraphs before you go. They contain information on those aspects of the till program which will be of much wider interest.

The main problem to be overcome before a professional presentation could be achieved was printing columns of figures neatly. If the computer has to print these figures: 2, 106, 100.2 and 0.003 in a column, the result will be something like this:	2 106 100.2 .003
---	---------------------------

But a commercial till will produce a column which is neater and easier to read which will look like this:	2.00 106.00 100.20 0.003
---	-----------------------------------

Some versions of BASIC include the PRINT USING facility which organises numerical data into a format which the programmer can specify. The format is often referred to as a mould. This facility is made available on the ZX computers by eight lines of quite

simple code. The lines are organised into a sub-routine which starts at line 50 in the program.

Another idea which will be interesting to the collector of programming techniques is a new method of selecting an option from a menu. If the menu is encountered continuously during a program run, this can become rather tedious and slow up the operation. In this program the most usual method of paying for goods, cash, is printed on the screen when the total payable has been calculated. If this method is appropriate then the user has only to start entering the amount of cash tendered to make this choice from the menu. If some other method of payment is to be made, cheque or one of the credit cards, then the NEWLINE or ENTER key is pressed until the appropriate message is printed. The printing is all done in the same space on the screen to avoid cluttering up the display. If the desired option is missed the NEWLINE key will make the whole series cycle through again. When the correct message is displayed then the next entry of information will automatically select the right section of code for the accounting.

At the end of business, the machine will print full details of the day's transactions, the number of each type of transaction and the takings in cash, credit card and cheque. The float is allowed for and a report is given of the amount of cash which should be in the till.

To avoid crashing the system, the code has been written so that only legitimate numbers can be entered. Mixtures of letters and numbers are rejected, as are numbers which contain more than one decimal point. If only real figures can be entered, then the crashes caused by inappropriate entry are avoided. The crashproofing makes use of the ZX string handling and has been written to make the keying in of the numbers as fast as possible without the use of machine code routines.

This is the first of the major programs and is written for both the ZX81 and the Spectrum. A few differences are to be found in the listings for the two computers. The ZX81 list is given first and any additions or amendments for the Spectrum are printed inside a box to draw attention to them. All the Spectrum variables should be in lower case and upper case should be used only where it would look more attractive on the screen. It is wise to use the SHIFT key and resist the temptation to use CAPS LOCK.

Now down to the code itself. The first line sends the computer to the correct part of the program if the RUN command is used, and then follows a routine which reads the keyboard and feeds the key-stroke into B\$.

```

1 GOTO 2000
2 REM READ THE KE BOARD

```

loop until key is released

```

3 IF INKEY$<>"" THEN GOTO 3

```

loop until new keystroke

```

4 IF INKEY$="" THEN GOTO 4

```

transfer data to B\$

```

5 LET B$=INKEY$
6 RETURN

```

Line 3 makes the computer wait until the last keystroke is released, before allowing it to go on. Line 4 makes it wait until a new keystroke is made. The last line transfers the information into B\$ before returning to the main program.

An extra line is needed to overcome the Spectrum's awkward decimal point, obtained by using SHIFT and the "m" key. Line 6 does the trick but displaces the RETURN.

```

6 IF b$="m" THEN LET b$="."
7 RETURN

```

The second routine does a simple clearing up job, tidying up the variables and moving unwanted text. Line 20 prints six spaces over an old number in the display. Y holds the position of the decimal point in a number and has to be set back to zero for each new number. The numbers are all fed into A\$ for treatment by the routine which gets them into the right format. Old numbers have to be cleared out of A\$ and line 10 does this job.

remove old number

```

10 LET A$=""

```

and from the screen too

```

20 PRINT AT 20,1: " "

```

reset the decimal point flag

```

30 LET Y=0
40 RETURN

```

Next is the important routine, which allows the printing of numbers in neat columns, with their decimal points all above each other. The routine is in two parts, the first finds the position of the decimal point and the second makes sure that there are two and no more than two decimal places in each number. The printing of the numbers in the columns is carried out in the main program. A\$ always holds the number being considered and these three lines search through A\$ to locate the point.

reset the decimal point flag

```
50 LET Y=0
```

set loop to length of number

```
60 FOR J=1 TO LEN A$
```

set Y to position of point

```
70 IF A$(J) = "." THEN LET Y=J  
80 NEXT J
```

If Y is still set to zero after these three lines, then ".00" has to be added onto A\$ and Y has to be set to two less than the length of the number. A\$(Y) will then always be ".". If there are no digits after the decimal point then "00" needs to be added. If there is one digit after the point then only "0" needs to be tacked on. Lines 90 to 120 do these jobs, and line 130 strips off unwanted decimal places.

add .00 if no point

```
90 IF Y=0 THEN LET A$=A$+".00"
```

sets flag to point position

```
100 IF Y=0 THEN LET Y=LEN A$-2
```

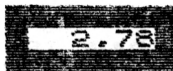
add correct number of 0s

```
110 IF Y=LEN A$ THEN LET A$=A$+  
"00"  
120 IF Y=LEN A$-1 THEN LET A$=A$+  
"0"
```

remove unwanted decimal places

```
130 LET A$=A$( TO Y+2)  
140 RETURN
```

All the figures are fed into the screen by way of a small window or box. The figures enter from the right of the box which is drawn by these two lines:



8 inverse spaces

```
200 LET A$="██████████"  
210 PRINT AT 19,0;A$,"█";TAB 7  
;"█",A$
```

Key GOTO 200 and then NEWLINE to make sure that your box is correct before continuing.

The ZX81 holds all its characters, including the digits 0 to 9, in code. All the digits have codes between 28 and 37. Any key-

stroke with a code outside these limits needs checking. Too many checks in the main loop would make the program slow to respond, and so as many checks as possible are placed outside the loop, at lines 150 and 160. If the keystroke is acceptable, then line 250 adds it to the growing number inside the box. Line 270 dumps numbers which grow too long or too big. The routine at line 10 simply resets the variables A\$ and Y, and then empties the box of the old number. Line 280 prints the latest version of the number in the box.

clear variables

```
220 GOSUB 10
```

read the keyboard

```
230 GOSUB 3
```

check anything but digits

```
240 IF CODE B$<28 OR CODE B$>37  
THEN GOTO 150
```

add to A\$

```
250 LET A$=A$+B$
```

set decimal point flag

```
260 IF B$="." THEN LET Y=1
```

dump if too big

```
270 IF LEN A$>6 OR VAL A$>999.9  
9 THEN GOSUB 10
```

print in box

```
280 PRINT AT 20,7-LEN A$;A$  
290 GOTO 230
```

Spectrum codes are different and the following line should be used for the later machine:

```
240 IF CODE b$<46 OR CODE b$>57  
OR CODE b$=47 THEN GO TO 150
```

This last routine has no RETURN statement. Keying "T" for total or "Z" for enter should make the computer return to the main program, and line 150 accomplishes this. Line 160 allows the first decimal point, but rejects the subsequent ones, because the decimal point flag will have been set to 1. Nothing special has to be done to dump incorrect entries. The machine falls through the program after line 160 and, by the time it gets back to building A\$ again at line 230, all the variables will have been cleared and reset.

```
150 IF B$="Z" OR B$="T" THEN RE  
TURN  
160 IF B$="." AND Y=0 THEN GOTO  
250
```

Tills make noises so the Spectrum's sound can be brought into play here.

```
150 IF b$="z" OR b$="t" THEN BE  
EP .1,10: RETURN
```

The Spectrum is less tolerant than the '81. The earlier machine gives an answer of zero if asked the "VAL" of a decimal point but the Spectrum gives an error message and stops the run. Here is the Spectrum version of line 260:

```
260 IF b$="." THEN LET y=1: IF  
LEN a$=1 THEN GO TO 280
```

The next sub-routine organises the printing of the column of figures. Keying "T" for total in the last routine will cause the machine to jump to this segment of code. Line 420 prints the price of the item onto the screen, so that its last digit will be in column thirty. If all the numbers have been organised to have two decimal places, then all the points will line up neatly underneath each other in column twenty-eight. The GOTO rather than GOSUB at line 440 saves a little processing time. There is a RETURN statement at line 40 which will finish off this sub-routine. When the line counter, Z, reaches a value of sixteen, there will not be enough room on the screen for all the text that will be generated later, and so the individual prices are removed and replaced by a sub-total. Line 435 sends the computer to a sub-routine which does this job.

return to work out total

```
400 IF B$="T" THEN RETURN
```

build a new number

```
410 GOSUB 50
```

print it in the column

```
420 PRINT AT Z,30-LEN A$;A$
```

add 1 to the line number

```
430 LET Z=Z+1
```

print sub-total if needed

```
435 IF Z=16 THEN GOSUB 300
```

clear variables and return

```
440 GOTO 20
```

And here is the code at line 300 which clears the necessary space. The variable A holds the running total of the bill, and this is loaded into A\$ by the line 330. The sub-routine at 50 ensures the correct format so that line 350 can print the figure in the correct position. The variables Z and A\$ are tidied up before the computer returns to the main program.

```

300 FOR J=1 TO 15
print 6 spaces
310 PRINT AT J,24;"
320 NEXT J

load sub-total into A$
330 LET A$=STR$ A

correct format
340 GOSUB 50

print at top of column

350 PRINT AT 1,16;"ELETTAL";TA
B 30-LEN A$;A$

tidy variables
360 LET Z=2
370 LET A$=""
380 RETURN

```

The segment of the program at line 500 controls the sub-routines we have met so far. The variable Z holds the line number for the next printing and D\$ holds the date. Line 520 allows a number to build and line 530 makes the computer print it on the next line. If "T" for total is keyed, then the computer by-passes the arithmetic and doesn't go back for another figure. The total, A, is loaded into A\$, formatted and then printed under the addition line. first line

```

500 LET Z=1

heading and date
510 PRINT "EXPI TILL",D$

build a number
520 GOSUB 200

print the number
530 GOSUB 400

print the total
540 IF B$="T" THEN GOTO 570

up-date running total
550 LET A=A+VAL A$
560 GOTO 520

load total into A$
570 LET A$=STR$ A

format total
580 GOSUB 50

print addition line and then the total
590 PRINT AT Z+1,24;"-----";TA
B 30-LEN A$;A$,"-----"

```

The differences in the interpretation of the BASIC mentioned in Chapter 2 make it necessary to give two versions of lines 570 to 750. Spectrum users will find that printer commands and a little more sound have been added to their code. ZX81 users will find a section at the end of the chapter which allows the use of the printer.

Next follows the segment which selects the method of payment. The routine at line 3 reads the keyboard and feeds the result into B\$. If a mistake has been made, the information may be dumped by keying "M" for mistake. The code for NEWLINE is 118 so any key-stroke but NEWLINE will cause the computer to set X equal to 1 and send the control to line 800. Later the machine will recognise X=1 as a signal that cash has been selected from the menu. The rest of the code in this section does a similar job for credit card and cheque payments. Line 750 sends the machine back to the start of the menu in case the required option was missed.

read the keyboard

```
600 GOSUB 3
```

dump error

```
605 IF B$="M" THEN PRINT ,"MISS"
610 IF B$="M" THEN GOTO 930
```

select CASH if NEWLINE is not pressed

```
615 IF CODE B$<>118 THEN LET X=
1
620 IF CODE B$<>118 THEN GOTO 8
20
```

select ACCESS if NEWLINE is not pressed

```
630 PRINT AT Z+3,16; "ACCESS"
640 GOSUB 3
650 IF CODE B$<>118 THEN LET X=
2
660 IF CODE B$<>118 THEN GOTO 8
20
```

ditto for BARCLAYCARD

```
670 PRINT AT Z+3,16; "BARCLAYCARD"
680 GOSUB 3
690 IF CODE B$<>118 THEN LET X=
3
700 IF CODE B$<>118 THEN GOTO 8
20
```

ditto for CHEQUES

```
710 PRINT AT Z+3,16; "CHEQUES"
720 GOSUB 3
730 IF CODE B$<>118 THEN LET X=
4
740 IF CODE B$<>118 THEN GOTO 8
20
```

loop if option was missed

```
750 GOTO 590
```

Here is the Spectrum version of these lines.

```
570 BEEP .1,12: LET a$=STR$ a
580 GO SUB 50
```

```

590 PRINT AT z+1,24;"██████████";TA
B 30-LEN a$;a$;"CASH": LPRINT
TAB 9;"██████████";TAB 15-LEN a$;a$
600 GO SUB 3
605 IF b$="M" THEN BEEP .5,-12:
PRINT "MISTAKE": LPRINT TAB
10;"MISTAKE": LPRINT : LPRINT
: LPRINT
610 IF b$="M" THEN GO TO 930
615 IF CODE b$<>13 THEN LET X=1
620 IF CODE b$<>13 THEN GO TO 8
00
630 PRINT AT z+3,16;"ACCESS"
640 GO SUB 3
650 IF CODE b$<>13 THEN LET X=2
660 IF CODE b$<>13 THEN GO TO 8
00
670 PRINT AT z+3,16;"B/CARD"
680 GO SUB 3
690 IF CODE b$<>13 THEN LET X=3
700 IF CODE b$<>13 THEN GO TO 8
00
710 PRINT AT z+3,16;"CHEQUE"
720 GO SUB 3

```

Line 800 causes the computer to take in a number if X=1. If X is set to any other value there is no need to read in the amount tendered, because a credit card slip or a cheque will be made out for the exact amount of the transaction. For cash, A\$ is set to the empty string and the routine at line 200 is called, but a little further on than usual. The result of this is that the figure that is keyed in is added to the empty string, A\$, as the first of the characters in the new number. This is a little difficult to appreciate but does increase the speed of the routine significantly. The figure is formatted at line 820 and then printed. The amount of change now has to be calculated and line 840 accomplishes this. The addition of .001 overcomes the slight inaccuracies of the computer's arithmetic, which loses a penny from time to time. Line 850 formats the change and this is then printed in place.

The variable CA holds the amount of cash each day. CA1 holds the number of cash transactions. Lines 900 to 925 load amounts of money into the ACCESS CARD, BARCLAYCARD and CHEQUE accounts. Finally, A is re-set to zero, the computer waits for the signal to clear the screen and get ready for a new customer.

by-pass the cash section

```
800 IF X<>1 THEN GOTO 900
```

start building a new number

```
805 LET A$=""
810 GOSUB 240
```

format it and print in place

```
820 GOSUB 50
830 PRINT AT Z+3,30-LEN A$;A$
```

calculate the change, format it and print in place

```
840 LET A$=STR$ (VAL A$-A+.001)
```

```

850 GOSUB 50
860 PRINT AT Z+4,16;"CHANGE";TA
B 30-LEN A$;A$

```

up-date cash balance and number of cash customers

```

870 LET CA=CA+A
875 LET CA1=CA1+1

```

update other accounts according to value of X

```

900 IF X=2 THEN LET AC=AC+A
905 IF X=2 THEN LET AC1=AC1+1
910 IF X=3 THEN LET BC=BC+A
915 IF X=3 THEN LET BC1=BC1+1
920 IF X=4 THEN LET CC=CC+A
925 IF X=4 THEN LET CC1=CC1+1

```

clear total, wait for signal (NEWLINE) to restart

```

930 LET A=0
940 INPUT A$
950 CLS
960 GOTO 500

```

This Spectrum specific code is not due to any difficulty with the language but simply shows the code needed to make the Spectrum drive the printer.

```

900 IF X=2 THEN LET AC=AC+A: LP
PRINT TAB 16;"Access Card"
905 IF X=2 THEN LET AC1=AC1+1
910 IF X=3 THEN LET BC=BC+A: LP
PRINT TAB 16;"Barclays Card"
915 IF X=3 THEN LET BC1=BC1+1
920 IF X=4 THEN LET CC=CC+A: LP
PRINT TAB 16;"Cheque"
925 IF X=4 THEN LET CC1=CC1+1
930 LET A=0
940 INPUT A$
950 CLS
960 GO TO 500
999 REM change
1000 CLS: PRINT "
M TILL PROGRAM"

```

The next section of the program would never be reached if the break key were not pressed. It was felt that something special was needed to cause the computer to move over to printing out the daily report and so a direct command was chosen as the signal. The only variable that we have not discussed is FL which holds the float. This section prints a heading, the date and then the number of each type of transaction and the amount of money involved. The float is printed and then the total is calculated by line 1230. The amount of cash in the till is calculated by line 1260 and then printed before the program stops running because of line 1290.

print heading and date, print no. of ACCESS slips

```

1000 PRINT "
1010 PRINT ,,D$
1020 PRINT ,,AC1;TAB 5;"ACCESS";

```

load ACCESS total into A\$ format and print value

```

1030 LET A$=STR$ AC
1040 GOSUB 50
1050 PRINT TAB 30-LEN A$;A$

```

repeat for BARCLAYCARD

```
1060 PRINT BC1;TAB 5;"B/CARD";
1070 LET A$=STR$ BC
1080 GOSUB 50
1090 PRINT TAB 30-LEN A$;A$
```

repeat for CHEQUES.

```
1100 PRINT C01;TAB 5;"CHEQUES";
1110 LET A$=STR$ C0
1120 GOSUB 50
1130 PRINT TAB 30-LEN A$;A$
```

repeat for CASH

```
1140 PRINT CA1;TAB 5;"CASH";
1150 LET A$=STR$ CA
1160 GOSUB 50
1170 PRINT TAB 30-LEN A$;A$
```

repeat for the FLOAT

```
1180 PRINT TAB 5;"FLOAT";
1190 LET A$=STR$ FL
1200 GOSUB 50
1210 PRINT TAB 30-LEN A$;A$;TAB
24;"          "
```

load total into A\$ and print it

```
1220 PRINT TAB 5;"TOTAL";
1230 LET A$=STR$ (FL+CA+AC+BC+C0
)
1240 GOSUB 50
1250 PRINT TAB 30-LEN A$;A$;,"T
ILL SHOULD CONTAIN £";
```

load cash total into A\$ and print it

```
1260 LET A$=STR$ (FL+CA)
1270 GOSUB 50
1280 PRINT A$
1290 GOTO 4000
```

The auto-start routine is used to print the program onto tape, and this is coded so that the first line that the computer goes to, when it is loaded, is line 2000.

wait for signal (NEWLINE)

```
3000 INPUT A$
```

print onto tape

```
3010 SAVE "TILL"
```

start when loaded

```
3020 GOTO 2000
```

The Spectrum does this job automatically and so needs only one line to replace the ZX81's three:

```
3000 SAVE "till" LINE 2000
```


The job which the code at line 2000 has to do is a simple but important one, setting all the variables. The date and float are set here, and lines 2130 to 2160 allow them to be dumped if a mistake has been made. As soon as "C" is entered at line 2140, work can start.

D\$ = date, CASH, ACCESS, B/Card and FLOat set

```

2000 PRINT "KEY THE DATE",
2010 INPUT D$
2015 PRINT D$
2020 LET CA=0
2030 LET CA1=CA
2040 LET AC=CA
2050 LET AC1=CA
2060 LET BC=CA
2070 LET BC1=CA
2080 LET C0=CA
2090 LET C01=CA
2100 PRINT , "KEY THE FLOAT ";
2110 INPUT FL
2120 PRINT "£";FL

```

dump if wrong

```

2130 PRINT , "KEY C IF CORRECT"
2140 INPUT A$
2150 CLS
2160 IF A$ <> "C" THEN GOTO 2000
2170 LET A=0

```

start work

```

2180 GOTO 500

```

When the program is loaded into the computer, the first thing that happens is that the date is requested. Any form of date will do but something like "1.1.82" is simple to enter. When the date has been logged, the computer asks for the amount of float, the cash used to start off the day's business. Key "C" when the details are as they should be. Key NEWLINE alone if there is a mistake.

The computer will now understand only the use of the number keys, the decimal point, the newline key and the keys Z, T and M. Key in a number and then key Z to log it. The number will appear top right of the screen. Enter another number and key Z to print it under the first. Key T to calculate the total and print it in place. CASH will appear on the screen. Key NEWLINE (ENTER on the Spectrum) and the CASH will be replaced with ACCESS. Keying Z at this point will log the amount under the ACCESS account. NEWLINE can be used to alter the ACCESS to B/CARD or back to CASH again if necessary. If cash is to be entered start keying in the amount of cash that has been tendered and then key Z to log it. The amount of change that has to be given is printed. Whichever option is selected, the next thing that will appear is the prompt to enter a character at the bottom of the screen. Key NEWLINE and the screen will clear and then set itself for a new customer.

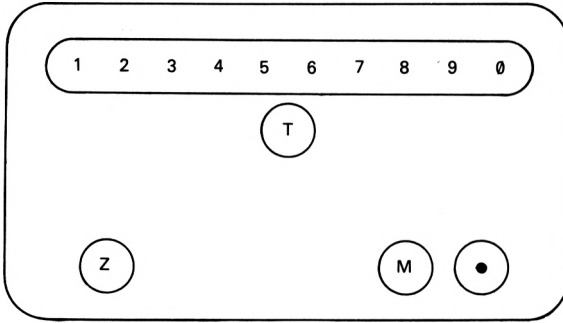
When the total has been calculated there is an opportunity to indicate a mistake. Key M at this stage and the screen will clear and set itself again. Spectrum users should use the P key.

Try entering twenty items and you will find that the list will

clear after item fourteen and be replaced with a sub-total. There is no limit to the number of items that can be logged, but there is a limit on the amount of cash that can be dealt with. The maximum that the program will accept is 999.99. It is unlikely that any more will be needed for the small businesses which will use the till but, if it causes problems, line 270 will have to be changed:

```
270 IF LEN A$ > 7 OR VAL A$ > 9999.99 THEN GOSUB 10
```

Now try entering numbers incorrectly. Figures with too many characters disappear from the box, as do those which include an I instead of a 1, a 0 instead of a 0 or two decimal points. The only key which can mess up the smooth flow of the program is the BREAK key. The Spectrum's BREAK is no problem because it only works with



the SHIFT key but the ZX81 BREAK key is more exposed and will have to be disabled in some way. The simplest method is to cover it with a coin held in place with blu-tack. A more elegant solution to the problem is to make a keyboard mask from a sheet of plastic. Holes should be punched in the plastic to allow only those keys which the user needs to show through.

When the day's business is done, the program can be made to print a report on the trading. To avoid the accidental generation of the report a special procedure is needed to generate it. Press the BREAK key and key GOTO 1000 followed by ENTER or NEWLINE as necessary. This will result in the production of the report. The program will stop running with a report code of 0/1290.

If a ZX printer is available, then only a few extra lines will make the production of a daily trade report and customers' till slips possible. The first extra line is:

```
515 LPRINT " ALF'S STORES " , D$
```

It is best to avoid the inverse graphics. The printer will produce the white on black print, but the quality is not all that it could be, especially if the printer has not been used for a short while and the paper has set into a distorted shape due to the pressure rollers. Here are the rest of the lines needed to produce a till slip for each customer:

```

425  LPRINT TAB 30 - LEN A$ ; A$
595  LPRINT TAB 24 ; " ██████████ " ; TAB 30 - LEN A$ ; A$
803  LPRINT , " CASH " ;
835  LPRINT TAB 30 - LEN A$ ; A$
865  LPRINT, " CHANGE " ; TAB 30 - LEN A$ ; A$
941  IF X = 2 THEN LPRINT " ACCESS "
942  IF X = 3 THEN LPRINT " B/CARD "
943  IF X = 4 THEN LPRINT " CHEQUE "
944  LPRINT " THANKYOU FOR YOUR CUSTOM "

```

And now the single line to produce the daily record of trading for filing:

1285 COPY

The use of the printer raises the question of reliability of the computer when made to run for long periods of time and, if the TILL is to be used all day and every day, then some precautions should be taken. The two big causes of trouble are overheating and loss of memory when the RAM pack moves on its contacts. If the Sinclair power supply is used the computer will get hot. The power pack described in the appendix will allow the computer to run much cooler for long periods of time. It costs only a few pounds to make and is well worth the investment, whatever use you make of the computer. The second problem needs more careful treatment. The obvious solution is to strip the computer out of its case, do the same with the ram pack, and then solder the two together. An extra benefit of this is that the machine can lose heat more quickly with its box removed. There is a less drastic and still quite effective solution. Buy a box of cottonwool buds and a bottle of surgical spirit (alcohol). Every day, clean the edge connector of both the computer and the printer with a bud which has been dampened with the spirit. Place the computer in such a position that it is protected from knocks and vibrations and the likelihood of loss of information is much less.

My own ZX printer has run all day at exhibitions and has given no trouble but, there are reports of reduced performance after the printer has been running for some time. Experiment with your own equipment before trusting it to work all day.

VARIABLES USED IN THE TILL PROGRAM

```

A$    characters of the number being worked on
B$    last key pressed
D$    date
A     running total of each transaction
CA    running total of cash
CA1   number of cash transactions
BC    running total of Barclaycard

```

BC1 Barclaycard transactions
AC running total of Access
AC1 Access transactions
CQ running total of cheques
CQ1 cheque transactions
FL daily float
J loop counter
X selection from the menu
Y position of decimal point in A\$
Z line number of next print position

TILL OPERATING INSTRUCTIONS

As soon as the program is loaded in from tape, it will start working by asking for a date. There is no fixed format for the date and any input will be accepted. The information fed in at this stage will be used as a label for all transactions during the run. After the date the machine will expect to be given a number when it asks for the float, the amount of money in the till at the start of the day. As soon as the information has been entered, the computer will start work on the first transaction of the day.

RECOVERING FROM CRASHES

With the keyboard mask, page 31, in place on the ZX81, these should be a rare occurrence. If a crash happens then the program can be started again by GOTO 1. This will preserve the information which has been entered already.

TRANSACTIONS

Enter a number into the machine and it will grow into the box which is printed on the screen. If the number grows too long, if it gets too big, or if it contains characters which make it invalid as a number, then it will be rejected and will disappear from the box. When the number is complete, press the Z key to enter the information. The number will be moved from the box to the head of the column of figures at the right of the display. Enter some more figures until the bill is complete. Use the Z key to enter the data each time. When the last figure is to be entered, key T for total. After a short wait the computer will draw a line under the list of numbers, print the total and then print the word CASH in inverse video on the next line.

SELECTING THE METHOD OF PAYMENT

Press the NEWLINE key several times and watch how the word in the block of inverse characters changes. The machine will cycle through the methods of payment available to it. As soon as any other key is pressed before NEWLINE, the computer will adopt the method of payment shown on the screen. Select CASH by pressing the NEWLINE key until CASH is displayed. Next start keying in a number. The machine will start building up a number in the box

again. This will be the amount of cash offered by the customer, and will be used to calculate the change to be given. Press the T key when the number is complete and the computer will go on to the final part of the cycle. The amount of change is calculated, and the book-keeping is done before the input prompt appears on the screen, bottom left. Press NEWLINE and the screen will be cleared in preparation for the next customer.

OTHER METHODS OF PAYMENT

When payment is made by cheque or by credit card, no change has to be given, so the computer assumes that the amount paid is exactly the amount of the bill whenever any of the alternative methods of payment are selected. Select the method of payment by pressing NEWLINE until the display is correct, and then press any other key to indicate your choice. The computer will now require you to press NEWLINE to finish the cycle and allow it to prepare the screen for the next customer.

MISTAKES

The M key on the ZX81 or the P key on the Spectrum can be pressed when the computer is asking for the method of payment. The effect of this is to clear out all the information on the current transaction but leave the rest of the data inside the machine intact. If a mistake has been made while preparing a bill, key T to produce the total and then key M or P to indicate a mistake.

LONG BILLS

If more than eighteen items have to be totalled then the machine will clear the first set of items, produce a sub-total, and then carry on making the list. This happens automatically and takes a little longer than normal, so expect a slight delay when the list reaches almost to the bottom of the screen.

THE DAILY REPORT

The keyboard mask will have to be removed and the BREAK key pressed in order to produce this information. After pressing BREAK, key GOTO 1000 and the machine will print a full report of the days transactions. You will be told the number of each type of transaction, cash, credit card, cheque, and the amount taken on each type of transaction. The total will be printed and the amount of cash that should be in the till will be shown. The information may be copied by the ZX printer simply by pressing the COPY key and then NEWLINE. There is no need to record the program on tape. Just keep the daily record slips and load the program into the computer afresh each day.

5 *A File Handling Survival Kit*

At meetings and courses centring around the use of the ZX range I like to discuss with people just what it is that they want to be able to do with their computers. The variety of uses mentioned is very wide but over and over again the following problem lies at the heart of the projects.

A mass of information has to be stored and the machine has to search through it and identify those items which fit a set of conditions. Because this description fits so many projects it might be useful to look at the way that such programs could be put together from simple blocks. The following jobs need to be done:

Storage space has to be set up.

The space has to be filled with information.

Allowance has to be made for updating the information.

The conditions of the searches have to be specified.

Some arithmetic may have to be done on the data.

The results of the data processing have to be printed out.

The data has to be stored on tape.

It is often the organisation of the storage space which holds up these projects. When this job is done the rest of the program often falls into shape quite quickly. The information is of two types, characters and numbers and it would seem reasonable to store these two types of data in number arrays and in character arrays respectively, but it is often better to store everything as characters. The ZX computer uses only *one* memory byte to hold a character but it takes *five* bytes to hold a single number, even 1 or 0. The 16K ZX81 will hold only a little more than three thousand numbers stored normally and this is sometimes a serious limitation on its usefulness. Numbers can be stored much more cheaply in the form of character codes as long as they are whole numbers in the range 0 to 255. It is surprising how often numbers can be made to fit this pattern quite easily. A little later on there is a method of storing heights in the range zero metres to 255 metres and 99 centimetres using only two bytes per number.

Imagine, for the purposes of this chapter, that you wish to log the name, address, age, height, phone number and marital status of

a number of people. A 16K ZX81 will hold all the data for over 150 people and still have room for a sophisticated program to handle the data smoothly. All the number information can be handled as character codes. Years and months are whole numbers within the 0 to 255 range, metres and centimetres fit the same pattern and the 'phone number can be stored as characters. The marital status can be coded: 1 for single, 2 for married, 3 for divorced, 4 for widowed and so on for as many states as seem necessary for the job. If more than 9 descriptions need to be coded then the letters A to Z can be used. Only one character will be needed for the marital status if this system is used. A few decisions need to be made before the data structure is designed; how many characters to book for the name and how many for the address and the 'phone number. The decisions will depend on the type of information that you have on your own program subjects. The following have been allocated arbitrarily for the purposes of the chapter:

Name	15 characters	characters 1 to 15
Address	40 characters	characters 16 to 55
Age (years)	1 character	character 56
Age (months)	1 character	character 57
Height (metres)	1 character	character 58
Height (cm)	1 character	character 59
'Phone No	15 characters	characters 60 to 74
Marital status	1 character	character 75

So each of the people to be described will need seventy-five bytes set aside in the machine for their particulars.

If you have 100 people in the set then 7500 characters will be needed to hold all the information, well within the scope of the ZX81 but only just inside the limit for the 16K Spectrum. The smaller version of the Spectrum uses the same amount of memory to hold its screen information as the 48K machine and this amounts to almost half of the 16K of RAM. The array to hold the data could be simply 7500 bytes long but there are advantages in making it into a block which is 100 items long and 75 characters deep. The statement which sets up this block is:

```
DIM A$( 100, 75 )
```

and this is best given as a direct command followed by a press on the NEWLINE or ENTER key. The statement could be given a line number and then it would appear in the program list, but there is always a chance that the computer would find this line again by mistake and scrub all the data that has been fed into the array ready for a new set. A method of protecting the data from the use of the RUN and CLEAR keys is given later on. If all these precautions are taken then the data will be quite safe.

We have accomplished the first of the tasks in the original list, setting up the storage space, by means of the single, direct command shown above. When you come to write your own data handling programs, and have to design the storage space, you will find a list like the one shown on page 37 helpful. Remember to make a permanent record of the information because it is not always easy to decide which bit is which just from the listing. After a program has been in use for a while, it is amazing how difficult it is to remember what the lines are for without a few notes. The array is in two dimensions to make the job of isolating the same piece of information from each person's file simple. To find the marital status of any person, all that is necessary is to specify the person number and the character in the file which holds the piece of information.

```
PRINT A$( J , 75 )
```

will print the information for person J because the marital status is stored in character number 75. The name is found by keying:

```
PRINT A$( J , TO 15 )
```

because the name occupies the first fifteen characters of each string of characters.

The task of filling the newly set-up data structure can be coded as a series of questions in PRINT statements, interspersed with INPUT statements. Here is a routine which would do the job adequately. A more sophisticated routine will be offered later as an alternative.

CONVENTIONAL STORAGE OF DATA

the array to hold the data

```
1 DIM A$(100,75)
```

counting through the list

```
10 FOR J=1 TO 100
```

using the loop counter to label the person

```
20 PRINT "KEY THE NAME OF PERSON NO. "; J
30 INPUT Z$
40 LET A$(J, TO 15) = Z$
50 PRINT Z$, "KEY THE ADDRESS"
60 INPUT Z$
```

the name in the first 15

```
70 LET A$(J, 15 TO 55) = Z$
80 PRINT Z$, "KEY THE AGE IN YEARS"
```


the same string variable is used for all input

```
90 INPUT Z$
```

the age is fed into the 56th and 57th characters as character codes

```
100 LET A$(J,56)=CHR$ VAL Z$
110 PRINT Z$,"AND MONTHS?"
120 INPUT Z$
130 LET A$(J,57)=CHR$ VAL Z$
140 PRINT Z$,"KEY THE HEIGHT I
N METRES"
150 INPUT Z$
160 LET A$(J,58)=CHR$ VAL Z$
170 PRINT Z$,"AND IN CMS?"
180 INPUT Z$
190 LET A$(J,59)=CHR$ VAL Z$
200 PRINT Z$,"KEY THE PHONE NU
MBER"
210 INPUT Z$
```

the phone number is held as characters and not numbers

```
220 LET A$(J,60 TO 74)=Z$
230 PRINT Z$,"KEY THE MARITAL
STATUS"
240 INPUT Z$
250 LET A$(J,75)=Z$
260 PRINT Z$
```

a signal to indicate readiness for the next person

```
270 INPUT Z$
280 CLS
290 NEXT J
```

Now that the data is in, it has to be checked for typing errors and, if necessary, dumped and re-loaded. If a lot of data is to be typed in at once, the likelihood of mistakes being made is high, and the process of re-loading the data is likely to become very trying. The more sophisticated routine mentioned earlier eases the problem but here is a typing error check routine to add to the above, straightforward version.

by-pass the subroutine

```
2 GO TO 10
```

mistake signal; anything but C

```
3 PRINT "KEY""C""IF CORRECT"
4 INPUT Z$
```

keeping the screen tidy

```
5 CLS
```

back to the main program

```
6 RETURN
```

This routine needs to be called and the signal generated by it needs to be acted upon. These few lines need to be added to the basic storage routine.

```
260 PRINT Z$,,
```

call the routine

```
270 GO SUB 3
```

re-load if unsatisfactory

```
280 IF Z$(">"C" THEN GO TO 20
```

otherwise next person

```
290 NEXT J
```

This next version of the loading routine is quite a lot longer than the more straightforward one and more complex, but the results are much more professional, and the section which prints out the data stored previously can be called by other parts of the program. This section is a sub-routine and lies between lines 100 and 180. The first line by-passes this routine until it is needed.

By-pass until called

```
1 GO TO 200
100 PRINT "PERSON ";J,A$(J, TO
15)
110 PRINT "ADDRESS",A$(J,16 TO
31),A$(J,32 TO 47),A$(J,48 TO 55
)
120 PRINT AT 4,0;"AGE IN YEARS"
, CODE A$(J,56)
130 PRINT "AND MONTHS",CODE A$(
J,57)
140 PRINT "METRES TALL",CODE A$(
J,58)
150 PRINT "CMS",CODE A$(J,59)
160 PRINT "PHONE NUMBER",A$(J,6
0 TO 74)
170 PRINT "MARITAL STATUS",A$(J
,75)
180 RETURN
```

The headings and the contents of the files are printed in two columns, making use of the comma to lay out the display. At first the files are empty but later, if the files need to be up-dated, the existing information is presented. A simple mechanism allows the correct information to be left and only the out-of-date items to be changed. In lines 120 to 150 the information needs to be converted from the stored characters into the CODE of each character. The rest of the lines which print the data print the characters directly.

Now follow a series of four line routines which lie within a loop. The loop counts through the files from 1 to 100 but there is a line at the end of the FOR/NEXT loop which allows the user to jump around in the files to make editing easier. At first a cursor is printed in place by line 220. This is removed and replaced on a lower line on the screen by the same line of BASIC which prints the data in place. The input is checked to see if it is an empty string. If this is the case, the user has simply pressed NEWLINE or ENTER and this is taken as the signal that the existing data is still correct. There is no need to change the file so the line which would normally change A\$ is hopped over. If new data is fed in then the appropriate characters of A\$ are changed by means of a LET statement.

counts through the files

```
200 FOR J=1 TO 100
```

print the headings

```
210 GO SUB 100
```

print the cursor

```
220 PRINT AT 0,15;"█"
```

read in the data

```
230 INPUT B$
```

print in place, change the cursor

```
240 PRINT AT 0,15;" ";B$;AT 1,1
5;"█"
```

skip if no change

```
250 IF B$="" THEN GO TO 270
```

up-date the file

```
260 LET A$(J, TO 15)=B$
```

Now follow nine such four line segments which fill the files. The address is filled in three goes to allow the conventional layout of the address on the paper.

```
270 INPUT B$
280 PRINT AT 1,15;" ";B$;AT 2,1
5;"█"
290 IF B$="" THEN GO TO 310
300 LET A$(J,16 TO 31)=B$
310 INPUT B$
320 PRINT AT 2,15;" ";B$;AT 3,1
5;"█"
330 IF B$="" THEN GO TO 350
340 LET A$(J,32 TO 47)=B$
350 INPUT B$
360 PRINT AT 3,15;" ";B$;AT 4,1
5;"█"
370 IF B$="" THEN GO TO 390
380 LET A$(J,48 TO 55)=B$
390 INPUT B$
400 PRINT AT 4,15;" ";B$;AT 5,1
5;"█"
```

```

410 IF B$="" THEN GO TO 430
420 LET A$(J,56)=CHR$ VAL B$
430 INPUT B$
440 PRINT AT 5,15;" ";B$;AT 6,1
5;"■"
450 IF B$="" THEN GO TO 470
460 LET A$(J,57)=CHR$ VAL B$
470 INPUT B$
480 PRINT AT 6,15;" ";B$;AT 7,1
5;"■"
490 IF B$="" THEN GO TO 510
500 LET A$(J,58)=CHR$ VAL B$
510 INPUT B$
520 PRINT AT 7,15;" ";B$;AT 8,1
5;"■"
530 IF B$="" THEN GO TO 550
540 LET A$(J,59)=CHR$ VAL B$
550 INPUT B$
560 PRINT AT 8,15;" ";B$;AT 9,1
5;"■"
570 IF B$="" THEN GO TO 590
580 LET A$(J,60 TO 74)=B$
590 INPUT B$
600 PRINT AT 9,15;" ";B$;AT 10,
15;"■"
610 IF B$="" THEN GO TO 630
620 LET A$(J,75)=B$
625 NEXT J

```

The final part is the few lines which allow the user to jump around in the files. This makes the files "random access" rather than serial access. Random access files are a lot quicker to use than the serial type but this version defaults to serial if no signal is given. If you simply key NEWLINE in response to lines 630 and 640, the machine automatically asks for the information on the next person.

J is set back by one so that, when NEXT J is encountered the correct file is found.

```

630 PRINT "KEY""Z""TO SELECT
ANOTHER PERSON"
640 INPUT Z$
650 IF Z$="Z" THEN INPUT J
660 IF Z$="Z" THEN LET J=J-1
670 CLS
680 NEXT J

```

Try out both these data entry systems. The question "is the extra effort worth making for the more sophisticated system?" is yours to answer, but remember that the display of the data allowed by this version may be useful in other parts of the program. To correct a mistake in the data, key "Z" at the end of the section, key the number of the person again and the file will be re-presented for amendment. Key NEWLINE (ENTER on the Spectrum) to skip over the correct data and, when the cursor is opposite the mistake, enter the correct version. If this system is of interest you may like to have a look at the data entry system used in the STATISTICS package described in Chapter 6. The method used there is a development of this system of data entry. A related method of

making a selection from a menu of options is given in the TILL program in Chapter 4.

Having decided on the system that suits you best and having fed the data into the files, the next job to attack is searching through the information. It might be necessary to identify all the people who are between the ages of fifty and sixty and who are widowed. The following few lines would make the computer search through the files and print out a list of all the persons who fit the description. Replace the PRINT statement with a LPRINT and the ZX printer will give you a permanent record of the information.

age 50 to 60, code for "widowed" = 3

```
1000 FOR J=1 TO 100
1010 IF CODE A$(J,56) >=50 AND CODE
DE A$(J,56) <=60 AND A$(J,75) = "3"
THEN PRINT A$(J, TO 15),
1020 NEXT J
```

This seems so simple and it is usually convenient to write the search specifications into the program in this way. Occasionally it is necessary to allow the setting up of search criteria while the program is running. If this is the case, then the limits for the search will have to be fed into variables by input statements and then used in lines such as line 1010 in place of the values 50, 60 and 3.

If it is important to know the decimal places in a number, or if numbers which fall outside the range 0 to 255 have to be stored, then character arrays cannot be used; the more expensive number arrays will have to be used. It is often convenient to use the two types of array together, and to use the same letter to refer to both. The items can be described in both of the files at the same time, with the characters in one and the numbers in the other. The loop counter can be used to feed the data into the same elements in the different arrays. If all of the people described in the last list were members of a savings club, then it would be necessary to store their balances in an array which could be set up by the statement:

```
DIM A ( 100 )
```

These lines could be tacked onto the end of the simple loading routine.

```
625 PRINT Z$, "KEY THE ACCOUNT
BALANCE"
626 INPUT A(J)
627 PRINT A(J)
```

Obviously the numbers stored in the number array can be used in arithmetic and, at first sight, it would seem that the same is not

true for the numbers stored in code form in character arrays, but this is not the case. With a little effort these numbers can be used in calculations as well. The following code will find the height of all the people in the list that we have been working on.

T is the store for the total

```
2000 LET T=0
```

N holds the number of people

```
2005 LET N=0
2010 FOR J=1 TO 100
2020 LET Z=CODE A$(J,58)*100+CODE
A$(J,59)
```

If Z = 0 then no height is stored so the next two lines are skipped.

```
2030 IF Z=0 THEN GO TO 2060
2040 LET N=N+1
2050 LET T=T+Z
2060 NEXT J
2070 PRINT "THE AVERAGE HEIGHT O
F THESE ";N," PEOPLE IS ";T/N/100
;" METRES"
```

There is no guarantee that information has been filed in all the possible records, so a line like 2030 is needed to avoid counting in a lot of zeros, and producing a ridiculously low result.

Another job which often needs to be done when dealing with a set of files is the sorting of the information into some sort of order. Alphabetical order is the most usual requirement but the method shown here is easy to adapt to produce a list in many other different formats. Line 3040 is the one which allows the machine to take the decisions. The line shown allows the computer to swap the files round in the list if the earlier one is greater than the later one. By "greater than" the computer understands "higher up in its list of character codes". If the first characters are the same in both records, then the computer checks the subsequent characters until a difference has been found, or the end of the strings are reached.

If you thought that there was a bit of a delay when you tried out the last section of code, averaging the heights, then you will understand the reason for the FAST command at line 3000. Spectrum users can ignore this line and also line 3095. Without this switch to high speed processing the computer would take nearly twenty minutes to re-organise a list of one hundred items. The job is done in a little more than four minutes in FAST mode. The machine can be excused the delay, it takes ten thousand decisions in this time.

3000 FAST

C\$ will hold the file which needs to be moved.

```
3010 DIM C$(75)
3020 FOR J=1 TO 100
3030 FOR K=1 TO 99
```

Only the name (1st 15 characters) are compared but ...

```
3040 IF A$(K+1, TO 15) <= A$(K, TO
15) THEN GOTO 3060
```

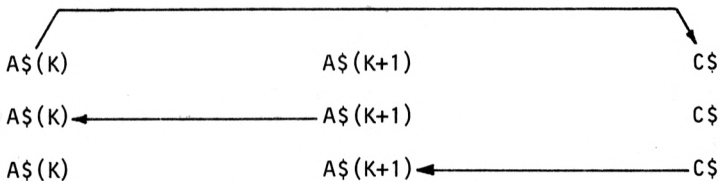
the whole file is swapped

```
3050 LET C#=A$(K)
3060 LET A$(K)=A$(K+1)
3070 LET A$(K+1)=C$
3080 NEXT K
3090 NEXT J
```

Finally, the new list is printed

```
3091 FOR J=1 TO 100
3092 IF A$(J, TO 15) = "
" THEN GOTO 3094
3093 PRINT A$(J, TO 15)
3095 SLOW
```

There are faster sorting routines but this one has a big advantage, it is easy to see how it works. Other, faster routines are difficult to follow, and one of the aims of this book is that all the code should be straightforward if at all possible. Finding mistakes is a slow process and slower still if you don't understand how the program works. Here is the way that the files are swapped around if they are in the wrong order.



It takes quite a time to load all the data into the files and it can be a tiring job. If the commands RUN or CLEAR were to be used after all this work then all the files would be emptied instantly. This possibility must be guarded against at all costs, but, fortunately the costs are very low. It is a simple job and there is an additional advantage. The answer is to keep the computer running the program and never allow it to stop and this doesn't mean that you can never turn the computer off. The computer can be fooled into thinking that it was running when the program and files were stored on tape. Here is the code which does the job for the ZX81:

```

4000  CLS
4010  PRINT "START THE TAPE AND PRESS NEWLINE"
4020  INPUT Z$
4030  LET Z$ = " DATA FILES "
4040  SAVE Z$
4050  GOTO 1

```

The Spectrum BASIC covers this automatically and all that is needed for the later machine is this one line:

```

4000  CLS : SAVE "DATA FILES" LINE 1

```

At line 4030 in the ZX81 program the computer will save the program and all its variables including the information on where to go for the next instruction. The next instruction is GOTO 1 and so, as soon as the computer is loaded with the program from the tape next time it starts work where it left off and GOes TO line 1. As far as the ZX81 is concerned it has never been switched off. The added advantage? You never have to start off a program run or set up the save procedure, these are automatically taken care of by the six lines of BASIC. You may notice that, after the auto-start routine, the last character in the name of the program is changed to the inverse video version of itself if a line like:

```

4030  SAVE " DATA FILES "

```

is used. This only causes problems if the program is saved back onto tape after it has been updated in some way. The data file type of program is used in this way and the two lines, 4030 and 4040 are used to avoid the problem. The name is effectively re-defined each time the program is saved.

For once you will not end up with a working program after working through the lines of BASIC in this chapter. True, the program will work, but the job that it does is a very specialised one. This departure from the general pattern followed in the rest of the book is inevitable, because the aim of this chapter has been to give a set of guidelines which will help in the construction of any number of programs, with the same general need at their centres. The next chapter extends this idea and offers a set of guidelines which will help in the planning and writing of *any* program.

6 *Super Storage and Statistics*

It is possible to obtain memory packs for the ZX81 which extend it far beyond the 16K which was originally thought to be its limit. Extra memory, however, is not the best answer to lack of space in many cases. There is a hidden cost over the above the very reasonable cost of the memory and it is reckoned in time not money. At the rate that the ZX81 reads data from tape, it takes about seven minutes to fill the 16K RAM pack completely. To fill a 128K memory at the same rate would take almost an hour, and the up-dated program and its files would take another hour to record on tape. This puts the idea of making safety copies of every program out of the question. The very large memories only become viable when a disc system is available to fill the space at a rate of several kilo-bytes a second. For many uses the cost of the extra memory and the disc system is not worth spending because other solutions to the space problem are available.

In the first book in this series I showed how it was possible to hold up to eighteen thousand single digit numbers in the 16K RAM pack and still have room for the program which loads them in, processes them and then prints out the results. This chapter is devoted to a program technique which extends even this quite acceptable storage efficiency but, as usual, this is only half the story. Large data banks need filling and this job has to be made as comfortable as possible. Several methods of easing the task of data logging are shown in detail for you to consider for use in your own programs, and the quality of the screen presentation has been kept very much in mind. All the points made in the chapter on making ZX programs at least as good as commercial programs, have been built into this package.

The Spectrum situation is a little strange. The idea of saving memory on the 48K version might be dismissed out of hand because there is so much space, running out is out of the question. A version of this program has been written on a 16K Spectrum and there was room for only 1½K of data because the screen and the system variables take so much memory. Saving memory or using it efficiently is a priority on the smaller computer and important on the 48K model as well. No matter how much space is available there never seems to be quite enough for the next program that is planned and, as the techniques used here are suitable for both computers, a Spectrum version has been prepared. As before, keep the Spectrum in lower case and key in the ZX81 program, over-writing the lines with the code in boxes. The boxed sections contain the lines which allow for the slight differences in the way the two versions of BASIC are interpreted and additional code to bring the Spectrum's special facilities into play.

The program stores a mixture of numbers and letters which makes it possible to store a lot of numerical information on a group of *named* individuals. The range of numbers that can be stored is the usual computing range of 0 to 255. The ZX81, running the program listed, stores 8370 numbers which would normally need 41850 bytes, 1890 characters of names and has room for the program which occupies about 6.5K. This adds up to more than a 48K computer could hold conventionally.

There has to be a vehicle for all these ideas and the one chosen does a statistical job. Such a job demands that a lot of number data is stored in association with a set of names and that the data gets a lot of mathematical treatment. I wanted to demonstrate that numbers stored unconventionally could be treated as easily as numbers stored in number variables. As with all the other programs, this package will do a big job of work as it is listed, or it can be stripped down and the components used in other programs which the readers have in mind for development. Parts of the program which lend themselves particularly to blending in with other programs are the sections which deal with the loading of names and numbers into a set of files, the method used to store numbers and the ways of treating the stored numbers mathematically.

When students follow courses of study which are assessed throughout the course and not just at a final examination, all the marks earned have to be treated very carefully. Adding up all the marks and then taking an average will not allow for the fact that some assessments are easier than others. If some students follow a route through the course which includes a lot of the more difficult assessments, they will be at a disadvantage and will be placed lower in the list of candidates as a result. This is a method of treating the results statistically which irons out all these difficulties and makes any mark directly comparable with any other gained by the same set of students. To illustrate how necessary this job is, think back to your own school career. Few of your fellow students gained more than 70% for English or Languages, but to gain 100% or very nearly 100% in Maths was not uncommon. Without careful treatment, the maths student would have an unfair advantage if the 'raw scores' for these two subjects were compared.

The statistical treatment requires the calculation of the standard deviation and the use of this to re-grade each piece of work. The maths is only covered in outline, you will have to consult a text on statistics for a fuller account. The program will collect the scores on thirty-six assessments from up to two hundred and ten students, re-grade them and then place the students in order of their achievement during the course.

The program starts with the usual sub-routine by-pass to send the user straight to the menu. In all these programs a user should be able to recover from a crash by keying GOTO 1. And talking of crashes, the next lines are a sub-routine which should prevent the vast majority of them. It is the routine which checks that a number is a real integer. The number to be checked is in A\$. If X

still equals zero after the routine, then the string will safely translate into a number when the command: LET A = VAL A\$ is met.

the menu

```
1 GOTO 900
```

mistake flag

```
10 LET X=0
```

rejects empty string

```
11 IF A$="" THEN LET X=1
```

rejects all characters except numbers

```
12 FOR J=1 TO LEN A$
13 IF A$(J) <"0" OR A$(J) >"9" T
HEN LET X=1
14 NEXT J
15 RETURN
```

Next the menu as described in earlier chapters. All input except the suggested keystrokes is rejected by line 950 and then line 970 sends the computer to the address of the different sections. All the sections are laid out in neat, 1000 line blocks. The screen is cleared, both before and after the menu, to keep the rest of the program both tidy and as short as possible.

```
900 CLS
```

the commas space out the display

```
910 PRINT TAB 9; "STATISTICS"
920 PRINT "1 ENTER NAMES"
930 PRINT "2 ENTER MARKS", "3 STANDARDI
SE", "4 STUDENTS RECORD", "5 FI
LE"
930 PRINT "SELECT OPTION (1 T
O 5)"
```

reject multi-character input and anything but "1" to "5"

```
940 INPUT A$
950 IF LEN A$ <> 1 OR A$ <"1" OR A
$ > "5" THEN GOTO 940
960 CLS
```

send to appropriate block

```
970 GOTO VAL A$*1000
```

All that was quite standard. Next comes the section of the program which allows the user to enter all of the names into the files. There is room for nine characters of text. Anything beyond nine characters is ignored by the computer. If more space is needed then all that is necessary is to increase the second dimension of the file array which is set up at line 6060, but this will mean less space will be available for complete files. In

practice, nine characters is enough to hold an initial and enough of a long name to identify it. The choice will depend on the number of students that have to be allowed for. More students means less space for their names.

The names are fed into the files in groups of thirty. The user is first asked which of the groups is to be loaded, the reply is loaded into A\$. This string is then checked to make sure that it is a number. If it passes this check then it must still be eliminated if it is too high or too low. The variable E will eventually contain the total number of students, so E/30 equals the group number of the last group. At the end of this section, B will hold the group number and G will hold the number of the first student in that group.

```

1000 PRINT "          ENTERING NAM
ES"
1010 PRINT ",,"KEY THE GROUP NUMB
ER"
1020 INPUT A$

```

is it a number?

```

1030 GOSUB 10
1040 IF X=1 THEN GOTO 1020

```

is it in range?

```

1045 IF A$="0" OR VAL A$>E/30 TH
EN GOTO 1020
1050 LET B=VAL A$

```

1st student in group

```

1060 LET G=1+(B-1)*30
1070 CLS

```

In SLOW mode it takes rather too long for the ZX81 to look up all the information in the files even if these are empty to start with. The first thing to do before asking the computer to build the list is to set it into FAST mode. The Spectrum works in FAST mode all the time so Spectrum users can ignore lines 1080 and 1130. The list is printed in two columns and line 1110 prints both of these so the counter only counts up to the fifteenth member of the group.

speedy printing

```

1080 FAST
1090 PRINT "GROUP ";B,,,
1100 FOR J=G TO G+14

```

2 column format with 2nd 15 in 2nd column

```
1110 PRINT " ";J;" ";S$(J,40 TO
),J+15;S$(J+15,40 TO )
1120 NEXT J
```

for smooth presentation

```
1130 SLOW
```

Now that the list is in place (it will be a blank screen at first!) the user will need to know where he or she is in the list, and so a cursor is printed at the first name to be entered. The co-ordinates of the cursor on the screen have to be set to '2 down and 0 across' and lines 1140/50 do this. Setting D to 15 allows the cursor to be printed for the second column; this may explain the seemingly over-complicated print lines which follow. The name, or one of several signals, can be fed into A\$ at line 1170. If a name (other than "1" or "Z") is entered, then line 1200 prints it in place and line 1210 feeds it into the correct file, characters 40 to the end.

screen co-ordinates

```
1140 LET A=2
1150 LET D=0
```

cursor

```
1160 PRINT AT A-D,D;"█"
```

enter name or signal

```
1170 INPUT A$
1175 PRINT AT A-D,D;" "
```

"Z" allows escape to menu

```
1180 IF A$="Z" THEN GOTO 900
```

"1" or "" dealt with later

```
1190 IF A$="1" OR A$="" THEN GOT
0 1220
```

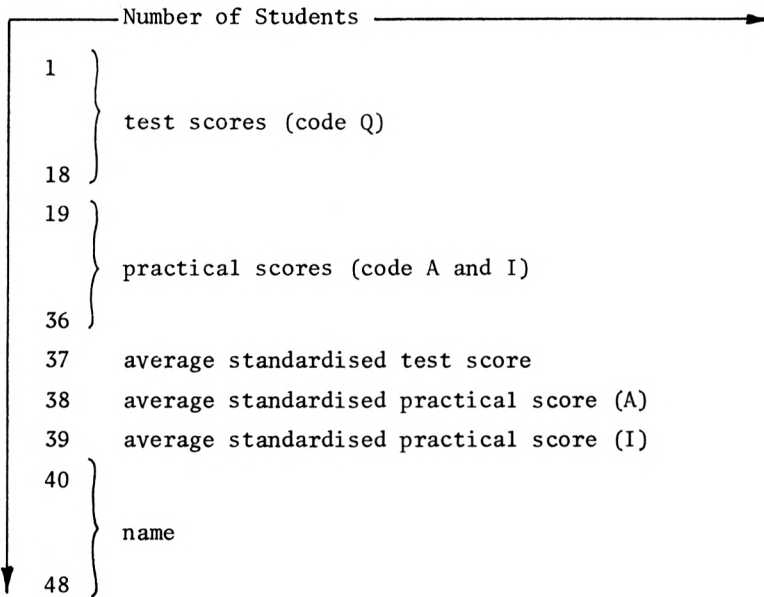
print name

```
1200 PRINT AT A-D,D+3;A$
```

enter in file

```
1210 LET S$(G+A-2,40 TO )=A$
```

Most of the information is held in an array which is as long as there are students and as deep as there is information to store on them. For the purposes of this program, the depth of the array is 48 characters and the length is set up the first time the program is used. Here is the detail of the storage in the main array, S\$.



This array is set up later by a command: DIM S\$(number of students, 48). A number array has to be set up as well to hold the detailed number information, but this will be dealt with a little later, in the section which makes use of it.

The computer will respond to a number of signals at this stage and the next section deals with these. "1" allows the user to 'back-space' in the list by deducting two from the line number of the cursor. The line number of the cursor is incremented by the next line, line 1230, and then it is checked to make sure the user doesn't go outside the limits of the list on the screen. The print position is set to the start of the list if this happens. Lines 1240 and 1250 make sure that the column being treated is the correct one. They do this by setting D to zero for the first column and to fifteen for the second.

back-space

```
1220 IF A$="1" THEN LET A=A-2
```

next in list

```
1230 LET A=A+1
```

keep within limits

```
1235 IF A=1 OR A=32 THEN LET A=2
```

correct column

```
1240 IF A<17 THEN LET D=0  
1250 IF A>16 THEN LET D=15
```

next item in the list

```
1260 GOTO 1160
```

You may have noticed that the variable A not only kept tabs on the correct print position in the list, but was used in line 1210 to select the correct file in the array.

The facility of keying either a name or a signal in the last section of the program, was an added convenience when keying a lot of names and, perhaps, getting tired and making mistakes. To be able to back-space or skip over existing information helps make correcting mistakes easier. The next segment of the program is the one which will be used most often, and so the bulk of the convenience 'extras' have been concentrated here. The method adopted does have a drawback. Pressing the BREAK key at any time will stop the program run and frighten the unwary by printing the program listing. This is because the INKEY\$ function is in use for some of the time. Keying BREAK during these times will make the ZX81 stop running the program. Although this is mentioned as a possible problem, it has not caused any trouble while my colleagues have been using the program. I have found the extra convenience very worthwhile and well worth incurring the slight risk. There is less of a problem on the Spectrum because the BREAK key has to be used with the SHIFT key.

First we have to select the correct group. This is a repeat of the task covered earlier, but the second piece of information needed, the assessment code, needs checking a little differently. The code of the assessments consists of a letter and a number. The number is checked separately by feeding the whole code into B\$ and then transferring the number part into A\$. A\$ is checked in the normal way by the sub-routine at line 10.

```
2000 PRINT "GROUP?", "GROUP?",  
2005 INPUT A$
```

is it a number?

```
2010 GOSUB 10  
2013 IF X=1 THEN GOTO 2005
```

is it in range?

```
2015 IF A$="0" OR VAL A$>E/30 TH  
EN GOTO 2005
```

B holds the group number

```
2020 LET B=VAL A$
```

G holds the 1st student in the group

```
2025 LET G=1+(B-1)*30  
2030 PRINT A$  
2040 PRINT "TEST?"
```

code into B\$

```
2045 INPUT B$
```

number part into A\$

```
2050 LET A$=B$(2 TO )
```

is A\$ a number?

```
2055 GOSUB 10  
2060 IF X=1 THEN GOTO 2045
```

The program is designed to respond to only a few codes, those pre-fixed by the letters "Q", "A" and "I". Q is the label for a test and all these are stored in bytes 1 to 18 in each element of the array. It might be wise to look back to page 52 and remind yourself of the way the information is stored. "A" labels the first batch of practical assessments and these results are stored in bytes 19 to 27. Bytes 28 to 37 will hold the results of the practical assignments labelled "I". Read through the next few lines of code and see how the variable T will end up with the correct byte number if the code is legitimate. If an error has been made, T will still be set to zero and so the machine loops back for another code if this is the case.

```
2065 LET T=0  
2070 IF CODE B$=54 THEN LET T=VAL  
L B$(2 TO )  
2080 IF CODE B$=38 THEN LET T=VAL  
L B$(2 TO )+18  
2090 IF CODE B$=46 THEN LET T=VAL  
L B$(2 TO )+27  
2100 IF T=0 THEN GOTO 2045
```

The Spectrum's character codes are different and so lines 2070 to 2090 have been rewritten. As a result the Spectrum will respond to either upper or lower case keystrokes.

```
2070 IF b$(1)="q" OR b$(1)="Q" T  
HEN LET t=VAL b$(2 TO )  
2080 IF b$(1)="a" OR b$(1)="A" T  
HEN LET t=VAL b$(2 TO )+28  
2090 IF b$(1)="i" OR b$(1)="I" T  
HEN LET t=VAL b$(2 TO )+27
```

The computer now knows the batch of students that you are interested in, and it knows the byte number in each of the thirty files in which it has to store the information which you are about to feed in. The next job is to make it easy for you to do your part, feed in the data. The computer prints a two column list for you as before, but this time with some scores. At first these will all be zero because nothing has been stored. The computer finds these scores in the files in the byte that the variable T points to. The score is stored as the code of one of the computer's character codes 0 to 255. Again, it is better to switch into FAST and then out again at the end of the printing routine to avoid an annoying wait. Spectrum users should always ignore the FAST and SLOW commands.


```

2110 CLS
2115 FAST
2120 PRINT "SCORE ";B,"TEST ";B$
...

```

two columns with marks

```

2130 FOR J=G TO G+14
2140 PRINT S$(J,40 TO );" ";COD
2150 S$(J,T),S$(J+15,40 TO );" ";CO
2160 DE S$(J+15,T)
2150 NEXT J
2155 SLOW

```

This next section is only slightly different from the one which pushed the cursor round in the list of names. This time it is necessary to check that the input is a real number and lines 2215 and 2220 see to this. The program would crash if a number greater than 255 were attempted, hence line 2230. "Z" is the code which allowed you to escape to the menu again but there is another signal, "A" to replace "1". This makes the system even more flexible.

screen co-ordinates

```

2160 LET A=2
2170 LET D=0

```

cursor

```

2180 PRINT AT A-D,D+10;"■"
2190 INPUT A$

```

escape to menu

```

2200 IF A$="Z" THEN GOTO 900

```

new signal

```

2205 IF A$="A" THEN GOTO 2300

```

remove cursor

```

2210 PRINT AT A-D,D+10;" "

```

is score a number?

```

2215 GOSUB 10
2220 IF X=1 THEN GOTO 2160

```

is it in range?

```

2230 IF VAL A$>255 THEN GOTO 210
0

```

print score in list

```

2240 PRINT AT A-D,D+10;" ";A$;"

```

The spaces in the last line allow corrections to be made neatly. If, for instance, a score of 100 were entered and only 1 should have been, the new score has to remove all traces of the previous record which occupied three screen spaces not one.

Now for the storing of the score. It is entered as a number and is now stored in A\$. If the VAL (value) of A\$ is taken, and then the character which has this value as its code is found, then the character can be stored in the correct place in the file. All this is done by the end of line 2250. A and D are re-set as before to make the cursor point to the next name in the list.

stores the score

```
2250 LET S$(G+A-2,T)=CHR$ VAL A$
```

increments the cursor

```
2260 LET A=A+1
```

finds the right column

```
2270 IF A<17 THEN LET D=0
2280 IF A>16 THEN LET D=15
```

next score

```
2290 GOTO 2180
```

The first line in the next section uses up a little time, slowing down the pace of the operation a little to make it comfortable. The line after says "if there is not a finger on the keyboard then wait until there is". The next line is a little more involved, this one says "if the keystroke except A or Z are used then go back to the normal section at line 2180". Line 2300 uses up a little time because the computers are a little too fast at this point.

Spectrum users should replace this line with:

```
2300 PAUSE 20
```

The ZX81 screen display is disturbed by a PAUSE statement so a little time-wasting arithmetic is used instead.

waste time

```
2300 LET C=RND*RND
```

wait for a keystroke

```
2310 IF INKEY$="" THEN GOTO 2310
```

go back if it's not A or Z

```
2320 IF INKEY$<>" " AND INKEY$<>"
A" AND INKEY$<>"Z" THEN GOTO 2180
```

The idea is that the two signals A and Z will move the cursor up and down the list all the time that you have the key pressed. This is much more convenient when entering a late result in the middle of the list than keying NEWLINE or ENTER repeatedly and makes back-spacing very much easier.

remove cursor

```
2325 PRINT AT A-D,D+10;" "
```

up?

```
2330 IF INKEY$="A" THEN LET A=A-  
1
```

down?

```
2340 IF INKEY$="Z" THEN LET A=A+  
1
```

keep inside the limits

```
2345 IF A>31 OR A<2 THEN LET A=2  
2350 IF A>16 THEN LET D=15
```

set the correct column

```
2360 IF A<17 THEN LET D=0
```

replace cursor

```
2370 PRINT AT A-D,D+10;"█"
```

cycle

```
2380 GOTO 2310
```

The code which follows will make the computers perform many thousands of calculations and the ZX81 will have to be switched into FAST mode until the arithmetic is over.

Spectrum users can replace line 3000 with:

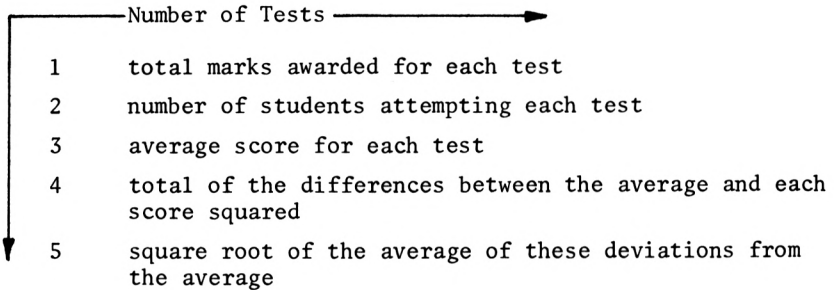
```
3000 PRINT AT 10, 10; "CALCULATING"
```

The job takes about ten minutes even in FAST so be prepared for a wait if you key "3" at the menu. If you do this by mistake then don't worry. Press the break key and then key GOTO 1 followed by NEWLINE and you will be back at the menu.

All the scores have to be added up to give 36 totals. And all the scores that have been entered have to be counted. These two pieces of information will enable the computer to calculate the average score. The code which does this first part of the calculation is in the form of a pair of loops, nested one inside the other. The loop which counts through the students is controlled by the counter J and set to the value of E as its limit. E, you will recall, holds the number of students.

The loop which counts through the tests is controlled by the counter K, and the limit of this loop is 36, the number of tests. If the machine has to add together over two hundred scores for each test, then numbers in excess of the maximum allowed, 256, will soon be generated. These totals and other numbers will have to be stored in number variables but, fortunately, only a few of

these will be needed compared with the number of stores that have to be set for the bulk of the data. An array is organised to hold all these numbers and is dimensioned in line 6070. The array is thirty-six numbers long and five deep. The numbers stored in it are as follows:



This last figure is known as the STANDARD DEVIATION and is a measure of the way the marks are spread. 67% of the marks will be within one standard deviation of the average.

```

3000 FAST
count through the students
3010 FOR J=1 TO E
count through the tests
3020 FOR K=1 TO 36

ignore record if no score is stored
3030 IF S$(J,K)="" THEN GOTO 3060
60

add score to total
3040 LET A(K,1)=A(K,1)+CODE(S$(J,K))
add 1 to number of attempts
3050 LET A(K,2)=A(K,2)+1
3060 NEXT K
3070 NEXT J

```

The loop at line 3080 works out the averages.

counts through the tests

```
3080 FOR K=1 TO 36
```

ignores tests no one has taken

```
3085 IF A(K,2)=0 THEN GOTO 3100
```

stores average in 3rd byte

```
3090 LET A(K,3)=A(K,1)/A(K,2)
3100 NEXT K
```

The rest of the section which does the standardising is made up of very similar loops. This next one finds the difference between the score and the average, squares the result and adds it to a total held in the fourth element of the array. The average deviation or standard deviation is found by taking the square root. The result is stored in the fifth element.

counts through students

```
3110 FOR J=1 TO E
```

counts through tests

```
3120 FOR K=1 TO 36
```

ignore empty records

```
3130 IF S$(J,K)=" " THEN GOTO 3150
```

difference between score and average

```
3140 LET B=A(K,3)-CODE (S$(J,K))
```

squared and stored

```
3145 LET A(K,4)=A(K,4)+B*B
3150 NEXT K
3160 NEXT J
```

counts through tests

```
3170 FOR K=1 TO 36
```

ignores tests no one has taken

```
3175 IF A(K,3)=0 THEN GOTO 3190
```

standard deviation stored

```
3180 LET A(K,5)=SQRT (A(K,4)/A(K,3))
3190 NEXT K
```

You may miss the NEXT J at the end of the next section. It's not there because two other loops run inside the loop controlled by J, and the NEXT J line is right at the end of this part of the program, just before the line which returns the user to the menu

after all the frantic calculation. The first of the set of three loops checks through each student's tests, finding how far from the average the mark is, and comparing this with the standard deviation. The maths at line 3300 is complicated but the result is just what we need - a new mark which gives an indication of how well the student has done on that test compared with all the other students who sat the same assessment. This is called the STANDARDISED SCORE. Standardised scores can safely be compared with scores from other tests and the results can be averaged to give a meaningful result. This seems like a lot of work for nothing more than a set of different marks, but the results are very much fairer to all the students and this is important when so much hangs on examination result.

counts students

```
3250 FOR J=1 TO E
```

total standardised scores

```
3260 LET B=0
```

number of tests sat

```
3270 LET C=0
3280 FOR K=1 TO 18
```

ignore empty files

```
3290 IF S$(J,K)="" OR A(K,S)=0
THEN GOTO 3320
```

calculate standardised scores

```
3300 LET B=B+30/A(K,S)*(CODE S$(
J,K)-A(K,3))+100
```

increment tests sat

```
3310 LET C=C+1
3320 NEXT K
```

by-pass if no results

```
3325 IF C=0 THEN GOTO 3340
```

store average standardised test scores

```
3330 LET S$(J,37)=CHR$(INT (B/C
))
```

The next two loops do a very similar job on the other two sets of results from the two types of practical assignment:

```
3340 LET B=0
3350 LET C=0
3360 FOR K=19 TO 27
3370 IF A(K,S)=0 OR S$(J,K)=""
THEN GOTO 3400
3380 LET B=B+30/A(K,S)*(CODE S$(
J,K)-A(K,3))+100
```

```

3390 LET C=C+1
3400 NEXT K
3405 IF C=0 THEN GOTO 3420
3410 LET S$(J,38)=CHR$(INT (B/C
))
3420 LET B=0
3430 LET C=B
3440 FOR K=20 TO 36
3450 IF A(K,5)=0 OR S$(J,K)=" "
THEN GOTO 3480
3460 LET B=B+30/A(K,5)*(CODE S$(
J,K)-A(K,3))+100
3470 LET C=C+1
3480 NEXT K
3490 IF C=0 THEN GOTO 3510
3500 LET S$(J,39)=CHR$(INT (B/C
))
3510 NEXT J
3520 SLOW
3530 GOTO 900

```

Spectrum users will now have to make a few minor alterations to the code between lines 3000 and 4000. An empty ZX81 or Spectrum string array contains spaces. The ZX81 code for a space is zero but the Spectrum codes it as 32. Spectrum strings are specially treated after they are set up later in the program, and will contain CHR\$ 0 in place of many of the spaces. In lines 3030, 3130, 3290, 3370 and 3450 the spaces shown as " " should be replaced by CHR\$ 0. Finally, the SLOW command at line 3520 is omitted.

Four small sections of the program remain. The first of these will print out a current 'state of play' for an individual student so the first thing is to identify the student. The usual checks are made to avoid crashes.

```

4000 PRINT "STUDENT? ";
4005 INPUT A$
4010 GOSUB 10

```

is the entry a number in the right range?

```

4015 IF X=1 OR VAL A$>E THEN GOT
O 4005
4020 LET B=VAL A$

```

Line 4040 prints the first half of each line of the display. The comma at the end means that line 4045 and 4050 print the second half of the information starting halfway across the same line. These two lines also ensure that the correct code is printed for all the practical assignments. Line 4055 prints the scores stored for the practicals.

```

4025 FAST

```

number and name

```

4030 PRINT B,S$(B,40 TO )
4035 FOR J=1 TO 18

```

test code and score

```
4040 PRINT "Q";J;" ";CODE S$(B,J)
;
```

practicals 1 to 9 codes

```
4045 IF J<10 THEN PRINT "AS";J;"
";
```

practicals 10 to 18 codes

```
4050 IF J>=10 THEN PRINT "IS";J-
9;" ";
```

practical score

```
4055 PRINT CODE S$(B,J+18)
4060 NEXT J
```

Finally the most important bit, the standardised average score on tests and both types of assessed practicals.

```
4065 PRINT
4070 PRINT CODE S$(B,37),CODE S$(
B,38),,CODE S$(B,39)
4080 SLOW
```

The user can now go on to the next student or back to the menu. Keying NEWLINE or ENTER sends the computer back for the next student unless this student is outside the permitted range. In this case you end up at the menu whether you want to or not. Throughout the programs in this book, Z is used as the input which allows the user to escape from the section of the program being used.

next student

```
4090 LET B=B+1
4100 INPUT A$
```

if number is wrong or "Z" is keyed, then go to menu

```
4110 IF B>E OR A$="Z" THEN GOTO
900
4120 CLS
```

otherwise next student

```
4130 GOTO 4025
```

The familiar ZX81 'file and automatic re-start' routine follows but, because this program will be repeatedly loaded onto tape the name is loaded into A\$ and the computer is told to save A\$. This makes for better loading, probably something to do with the way that the last letter of the name changes to inverse video after the save routine.


```

5000 PRINT AT 9,9;" START TAPE
";TAB 9;" KEY NEWLINE "
5010 INPUT A$
5020 LET A$="STATISTICS"
5030 SAVE A$

```

straight from tape to the menu

```
5040 GOTO 900
```

The Spectrum code is much simpler and line 5000 replaces these four.

```
5000 SAVE "statistics" LINE 900
```

When the program is loaded onto tape for the first time it should be by means of GOTO 6000. The section of program there is not obtainable from the menu and so will only be used once to set up the system. The number of students has to be set so that the main array can be dimensioned and the variable E adjusted to this number.

```

6000 SAVE "STATISTICS"
6010 PRINT "KEY THE NUMBER OF GR
OUPS", "OF 30 STUDENTS"
6020 INPUT A$

```

is it a number?

```
6030 GOSUB 10
```

in the right range?

```
6040 IF X=1 OR VAL A$>6 THEN GOT
O 6020
```

total students

```
6050 LET E=VAL A$*30
```

main array of files

```
6060 DIM S$(E,48)
```

number storage

```
6070 DIM A(36,5)
```

to the menu

```
6080 GOTO 900
```

The Spectrum BASIC makes it necessary to split line 6040 as shown.

```

6040 IF X=1 THEN GO TO 6020
6045 IF VAL A$>1 THEN GO TO 6020
: REM a 48K PROG.

```

The limit of 1 group of students is for the 16K machine only!
48K Spectrum users can allow twenty groups quite safely.

The Spectrum array has to be conditioned and line 6080 changes
all the spaces that the numbers are to occupy into CHR\$ 0.

```
6080 LET a$="": FOR J=1 TO 39: L
ET a$=a$+CHR$ 0: NEXT J: FOR J=1
TO e: LET s$(J, TO 39)=a$: NEXT
J: LET a$="": GO TO 900
```

It would be a long job, printing each student's record on the
screen and then making an ordered list of the standardised scores
The computer can do this for you and make a permanent record on
the ZX printer. If a printer is not available then change all the
LPRINT commands in the next section of the program to PRINT. This
will make the computer print the list on the screen for you to
copy by hand. To read each successive page of the display key CONT
and then NEWLINE or ENTER.

After the section at line 3000 has been set loose on the
results, the students will have standardised scores in bytes 37 to
39 of their files. The maths ensures that the average student will
have a score of 100, and that the other scores will be in the
range 60 to 140. Only exceptional cases will be otherwise. The
first score that the ZX81 looks for is the lowest to be expected,
60, and it continues its search until it reaches the upper limit
of 140, and then stops. The score being looked for is stored in A,
set to 60 at line 7010 and incremented after each search at line
7090. Line 7100 checks that the upper limit is not reached before
line 7120 sends the ZX81 round again, looking for the next score
in the series. 7050/60/70 check each student's records in turn and
print the name at the end of the file if the score stored matches
the value of A this time round.

```
7000 FAST
```

lower limit of search

```
7010 LET A=60
```

headings

```
7020 LPRINT "TESTS";TAB 10;"PRAC
TS A";TAB 20;"PRACTS I"
7030 LPRINT A;" MARKS",,
7040 FOR J=1 TO E
```

does test average match A?

```
7050 IF CODE S$(J,37)=A THEN LPR
INT TAB 0;S$(J,40 TO );
```

does practical A match?

```
7060 IF CODE S$(J,38)=A THEN LPR
INT TAB 10;S$(J,40 TO );
```

does practical B match?

```
7070 IF CODE S$(J,39)=A THEN LPA  
INT TAB 20;S$(J,40 TO );  
7080 NEXT J
```

increment mark to check

```
7090 LET A=A+1
```

upper limit of search

```
7100 IF A>140 THEN STOP
```

next mark in series

```
7120 GOTO 7030
```

This is a very slow, inefficient search by computing standards, but it produces the information at about the rate that the ZX printer can put it on paper, so why write a faster, more complex search?

These two sections, at 6000 and at 7000 are only used once and so are not on offer at the menu. The setting up procedure is brought into play automatically if the program is saved onto tape by GOTO 6000. The search is a confidential matter, not to be made generally available, and so it is made a little more difficult to produce. The method is given in detail in the operating instructions.

The ZX81 version of the program is set up to deal with six groups of thirty students and this should be sufficient for most courses. If more space is needed then this should be made available by the simple expedient of not including the last two sections in the program. The section at line 7000 can be added at the end of the year when it is needed and the following few direct commands will replace the section at line 6000. Seven groups of thirty students can then be fitted in. By trimming the program to the bare bones it is possible to set the program up for eight groups of thirty; this was done successfully this year. For most schools and colleges, though, a program to deal with up to 180 students is enough.

SETTING UP COMMANDS

```
LET E = (the number of students)* NEWLINE  
DIM S$(E, 48) NEWLINE  
DIM A(36, 5) NEWLINE  
GOTO 900 NEWLINE
```

* E must be set to a multiple of thirty and so only the following values can be fed in: 30, 60, 90, 120, 150, 180 and 210.

This is quite a complex program and some operating instructions are needed to help you remember how to use it to best advantage. It is surprising how quickly the available facilities are forgotten

about. In the next chapter the operating instructions are written for a complete novice. These instructions are not as detailed, to make them so would not be the best use of the available space.

OPERATING INSTRUCTIONS FOR "STATISTICS"

In these instructions, Spectrum users should read "NEWLINE or ENTER" wherever they see "NEWLINE" or "N/L".

After loading the program for the first time, you will be asked for the number of groups of thirty students which have to be dealt with. If the program has been used before then this will not happen. Key the number followed by NEWLINE. Only values between 1 and 6 will be accepted. Once the computer has accepted your input it will display the menu of options available to you.

ENTER NAMES

Key 1, N/L at the menu and you will be asked to indicate the group you wish to work on. From this point it will be assumed that N/L will be pressed after each keystroke, unless otherwise indicated. Key the group number and the screen will blank for five seconds as the display is built up. The list will be empty of names on the first run through but later, all previously entered names will be present.

Key N/L alone and the cursor will jump to the next position in the list, without altering the contents of the first. This will be useful later for altering spellings or changing the names.

Key 1 and the cursor will move back a space.

Key a name (anything but "1" or "Z") and the name will be printed in place and the cursor will jump to the next position.

Key Z and the computer will return you to the menu.

It is impossible to move out of the list. If you try you will end up at position 1 every time. The only escape from the routine is to key Z.

ENTER MARKS

Key 2 at the menu and you will be asked for the group number. Enter this and you will next be asked for the test code of the marks to be entered. The computer will accept codes prefixed by Q, A and I only. The files are set up for eighteen Q assessments, nine A and nine I assessments, making thirty-six in all.

The screen will blank again while the list is constructed.

Key a mark between 0 and 255 and it will be printed in the list.

Key Z and you will be returned to the menu.

Key A and the input prompt will disappear. Until the input prompt returns you will no longer have to key N/L after a key-stroke.

Hold Z down and the cursor will move down the list until you release the key.

Hold A down and the reverse will happen.

Use these keys to skip over scores that have been already logged and then press any other key, N/L is the best choice, and you will be returned to the input prompt and the earlier set of commands.

Again, the escape from this routine is the Z key.

STANDARDISE

Key 3 at the menu and the screen will blank for anything up to twelve minutes depending on the number of students being dealt with. At the end of this time the screen will clear and the menu will be presented. The computer will have performed many thousands of calculations and generated a set of standardised scores which will be filed under the students' names.

STUDENT'S RECORD

Key 4 at the menu and you will be asked for the student number. The screen will blank again and, after five seconds, the list of all the scores earned by the student whose name appears at the top of the screen will be displayed. At the bottom of the screen you will see the three standardised scores which correspond to the three types of test listed. These will be zero until the standardising routine has been used, and then they will be set to the most recently calculated values. There is no need to wait until the end of the course to standardise the stored scores and this job can be repeated as often as required.

FILE

Key 5 at the menu and you will be asked to start the tape and then press NEWLINE. Allow a few seconds for the tape speed to steady before starting the SAVE process by keying N/L.

RECOVERY FROM CRASHES

If the computer suddenly produces a screenful of program lines, don't press any keys until you have re-read this. Key SHIFT and hold it down while you key 1. Key NEWLINE. Key GOTO 1, N/L and you should be returned to the menu. This covers the majority of crashes and crashes should be very few and far between. Other crashes will probably be irrevocable and the computer should be switched off and then on again to allow the program to be re-loaded from the tape.

PRINTING THE MERIT ORDER

Set up the computer with the ZX printer attached and load in the program. Key 5 at the menu as if to file the program on tape, key N/L and then BREAK to stop the process. The command GOTO 7000 should start the printer off, producing the list. If all you get is a blank screen with a code o/o at the bottom left, then the code is not in place. Key in the lines from 7000 to 7120 on page 64 and repeat the above process.

If a printer is not available, follow the above procedure until you have pressed the BREAK key. Key LIST 7000 and check that lines 7020, 7030, 7050, 7060 and 7070 have PRINT statements and not LPRINT. Make any alterations necessary and then key GOTO 7000. Write down the information from the screen and then key CONT, N/L. Make a note of the information on this second page and then repeat the process until the machine stops.

VARIABLES USED IN THE STATISTICS PROGRAM

S\$ main storage files DIMensioned to (number of students, 48)
A() numerical store DIMensioned to (36, 5)
A\$ temporary store for characters
B\$ temporary store when A\$ is in use
A&D screen co-ordinates
J&K loop counters
B temporary store for numbers
C temporary store when B is in use
E number of students
G 1st student in current group
T test number
X mistake flag

7 *Cricket*

By way of relaxation, here is a program which uses the same ideas as were built into the last chapter, but applied to a lighter subject. Cricket enthusiasts may not consider the storage and handling of their scores and bowling figures to be less important than the information dealt with in STATISTICS, but I don't think they will quarrel with the description 'lighter'. Cricketers do seem to be pre-occupied, obsessed even, with these figures so there may be a big demand for a program like this one, which takes all the hard work out of the job and reduces the likelihood of errors creeping into the statistics. This is a big program and so the space left over for data would be insufficient if all the numbers were to be stored in number variables, and so the use of character arrays improves the efficiency of the program.

16K Spectrum users will be pleased to learn that, big though the program is, there is still space for up to sixty players' records. Most clubs should be able to manage with two or three copies of this program. A 48K Spectrum will cope with the biggest of clubs.

In this program it is highly likely that numbers which will exceed the storage available in a single byte will crop up. The number of runs that your best batsman makes in a season will probably be more than 255, as will the number of runs scored off your bowlers. The solution used last time, running a parallel number variable to hold the more difficult data, is one way but there is another possibility which is used here in order to give as many examples of solutions to computing problems as possible. The larger numbers are stored in two bytes using a system like the one used by the ZX80, the forerunner of the '81.

The probability is that cricket clubs will delegate this chore to a junior member and such a person will not necessarily have any experience of computers, so the input will have to be checked most carefully to avoid crashes. All the input lines are carefully checked and one of the checking routines mentioned earlier is in place at line 10 and is worked very hard. There is not much more that needs to be said at this stage, so the lines of code start in the next paragraph. Full documentation will be needed and this will be found at the end of the chapter. Your computer operator may well need an extract of this documentation at hand when the

program is being used. Several of the facilities which have been built in to help ease the task of inputting the data are not mentioned in any of the screen displays. To prompt the user about all the available features of the program would make the screen very cluttered.

The first section is a pair of sub-routines. The second has been encountered earlier - it checks that strings will translate into numbers without trouble. The first routine prints out the name of a player. This information is held in the first ten bytes of the player's file. The files are all held in a string array C\$, which is dimensioned to the number of players and the number of characters needed to hold all the information. Line 1 by-passes the routines and sends the computer to the menu at line 900.

```
1 GOTO 900
```

prints the name

```
2 PRINT ,C$(Z(J) , TO 10)
3 RETURN
```

mistake flag. X=0; no mistake

```
10 LET X=0
```

rejects an empty string

```
12 IF A$="" THEN LET X=1
```

rejects non-number characters

```
13 FOR K=1 TO LEN A$
14 IF CODE A$(K)<28 OR CODE A$(K)>37 THEN LET X=1
15 NEXT K
17 RETURN
```

The menu is the same standard style menu used throughout the book.

The commas organise the print onto separate lines

```
900 CLS
910 PRINT "          ZX81 USERS CRI
CKEY CLUB"
920 PRINT ,,"1  ENTER MATCH FI
GURES",,"2  PRINT CLUB FIGURES",
"3  PRINT FIGURES FOR INDIVIDUA
L4  ENTER NAMES",,"5  FILE REC
ORDS"
930 PRINT ,,,, "KEY OPTION (1 TO
5)"
940 INPUT A$
```


rejects all but the options on offer

```
950 IF LEN A$>1 OR CODE A$<29 OR  
R CODE A$>33 THEN GOTO 940  
960 CLS
```

VAL A\$*1000 = 1st line of option

```
970 GOTO VAL A$*1000
```

Spectrum codes are different and this line must be used:

```
950 IF LEN a$>1 OR a$<"1" OR a$  
>"5" THEN GO TO 940
```

So that the user will know where he stopped storing the figures, the name of the last opponents played and the date of the match are held in two string variables, M\$ and D\$. This pair of variables has to be updated for each game that is recorded.

M\$ holds name of last opponents

```
1000 PRINT "LAST MATCH ENTERED W  
AS AGAINST",M$
```

D\$ holds date of last match

```
1005 PRINT "ON ";D$  
1010 PRINT "KEY THE OPPONENTS  
IN THIS MATCH "
```

temporary store before checking

```
1020 INPUT A$  
1030 PRINT A$  
1040 PRINT "KEY THE MATCH DATE",
```

temporary store before checking

```
1050 INPUT B$  
1060 PRINT B$,"KEY C FOR CORREC  
T OR NEWLINE"  
1070 INPUT Z$  
1080 CLS
```

reject mistakes

```
1090 IF Z$<>"C" THEN GOTO 1000
```

load into final stores

```
1100 LET M$=A$  
1110 LET D$=B$
```

The team which played in the match in question has to be recorded and this is done by requiring the user to input the eleven players' reference numbers. The numbers are stored in Z, an array which is re-dimensioned for each game and consequently, emptied of

the previous records. This trick cannot be used in many versions of BASIC but is a useful feature of ZX BASIC.

```
1115 DIM Z(11)
1120 FOR J=1 TO 11
1130 PRINT "PLAYER "; J;
1140 INPUT A$
```

check that A\$ is a number

```
1142 GOSUB 10
```

reject A\$ if flag is set

```
1144 IF X<>0 THEN GOTO 1140
```

transfer A\$ to A

```
1146 LET A=VAL A$
```

reject if unsuitable

```
1148 IF A<1 OR A>P THEN GOTO 1140
```

record the player

```
1150 LET Z(J)=A
```

print his name

```
1160 GOSUB 2
1170 NEXT J
```

If the player number is less than one then it cannot be accepted. The variable P holds the number of players' records and so, if the player number is greater than P, it cannot be accepted. Line 1148 rejects these possibilities. This line was found to be necessary during the de-bugging of the program, and the line numbers are left un-tied as a reminder that programs need to evolve, and that the programmer need not feel concerned if the results are not perfect at first. In Chapter 10 you will find full details of several programs which will renumber programs automatically.

If the first set of player numbers that are fed into the computer are not entirely correct, there is no need to dump all of them and start again. This next segment allows the mistake to be remedied. Two pieces of information are needed, the team number at which the wrong player is logged, and the player number of the correct player for this slot. The first of these numbers is fed into A in line 1226. This segment, again, contains many input checks to avoid crashes.

```
1180 PRINT AT 12,0;"KEY C IF THE
SE ARE CORRECT "; "OTHERWISE KEY
1190 INPUT A$
```

by-pass if correct

```
1200 IF A$="C" THEN GOTO 1300
```

over-prints the previous message

```
1210 PRINT AT 12,0;"KEY THE INCO  
RRECT PLAYER, [ ]"  
1215 PRINT "THEN KEY THE CORRECT  
CODE, [ ]"  
1220 INPUT A$
```

checks that A\$ is a number

```
1222 GOSUB 10
```

dumps if necessary

```
1224 IF X<>0 THEN GOTO 1220  
1225 IF VAL A$>11 THEN GOTO 1220  
1226 LET A=VAL A$  
1228 IF A<1 OR A>P THEN GOTO 122  
0
```

The position of the mistake is now indicated by the cursor printed by line 1230. The player number is loaded into the array Z if it has satisfied all the check routines, and then the new name is taken from the array C\$ and printed over the old name. Finally, the computer does not assume that the new information is correct, or that there was only one mistake. It goes back to line 1180 and asks again if the information is correct. The message at line 1180 over-prints the message printed by lines 1210 and 1215. The effect of this over-printing is very professional and makes the corrections less tedious.

a cursor

```
1230 PRINT AT A-1,15;" [ ]"  
1240 INPUT A$
```

check that A\$ is a number

```
1242 GOSUB 10
```

reject if unsuitable

```
1244 IF X<>0 THEN GOTO 1240  
1246 IF VAL A$<1 OR VAL A$>P THE  
N GOTO 1240
```

store in team array

```
1248 LET Z(A)=VAL A$
```

print name

```
1250 PRINT AT A-1,15;" ";C$(Z(A)  
, TO 10)  
1260 GOTO 1160
```

Having established the make-up of the team, the next job is to organise the feeding in of the details of each man's performance. This is done in two stages because the bowling figures for all the players will not be required. It is important for working out the batsman's average, to know how many innings he played and how

often he was 'not out'. These lines again use a cursor to indicate the current information being fed in.

```
1300 CLS
1310 FOR J=1 TO 11
```

player's name

```
1320 PRINT "MATCH REPORT FOR ";
C$(Z(J), TO 10)
```

all text printed at once with cursor against first item

```
1330 PRINT " WAS HE OUT? (Y/N) ";
TAB 20; " ", " RUNS SCORED", " DID H
E BOWL? (Y/N) "
1340 INPUT F$
```

check and reject line

```
1350 IF F$<>"Y" AND F$<>"N" THEN
GOTO 1340
```

over-print old cursor with data and re-print cursor

```
1360 PRINT AT 2,20;F$;TAB 20;" "
1370 INPUT A$
```

check that A\$ is a number

```
1380 GOSUB 10
1390 IF X=1 THEN GOTO 1370
1400 LET A=VAL A$
```

prevents exceeding storage capacity later at line 1535

```
1410 IF A>255 THEN GOTO 1370
1420 PRINT AT 3,20;A;TAB 20;" "
1430 INPUT B$
1440 IF B$<>"Y" AND B$<>"N" THEN
GOTO 1430
```

The problem of using TAB on the Spectrum was mentioned in Chapter 2. Here is a case where PRINT AT has to be substituted. The lines which need changing are:

```
1360 PRINT AT 2,20;f$;AT 3,20;" "
1420 PRINT AT 3,20;a;AT 4,20;" "
```

Lines 1390 to 1440 repeat the above processes for the other information. Even if the information passes the checks it might still be incorrect and so the user has to check the data before it is built into the data structure. This program does some of the processing as it goes along, and we cannot change information that has been fed in in previous runs as we did in the STATISTICS program. The user has to take the responsibility for the correctness of his entry each time the program is loaded from the tape.

```
1450 PRINT AT 4,20;B$,"KEY C I  
F OK"
```

accept or reject

```
1460 INPUT A$
```

next section if OK

```
1470 IF A$="C" THEN GOTO 1500  
1480 CLS
```

repeat the record

```
1490 GOTO 1320
```

To specify the position of any particular record in any one file in the main array would need quite a lot of code. If the whole file was loaded into a temporary file, then this could be more easily up-dated, and then written into the records in the place of the earlier file. E\$ is used as such a file. In the diagram you will find details of the information stored in different positions in the two strings, C\$ and E\$.

<i>Information Stored</i>	<i>C\$ element</i>	<i>E\$ element</i>
NAME	1 to 10	-
NO. OF INNINGS	11	1
NOT OUT	12	2
NO. OF RUNS SCORED	13 & 14	3 & 4
BEST SCORE SO FAR	15	5
NO. OF 20s	16	6
NO. OF 50s	17	7
NO. OF 100s	18	8
OVERS BOWLED	19	9
MAIDEN OVERS	20	10
WICKETS	21	11
RUNS CONCEDED	22 & 23	12 & 13
BATTING AVERAGE	24	14
BOWLING AVERAGE	25	15

The averages are held as integers only and are used to put the club members into rough order of merit a little later on in the program. The averages are printed to eight places of decimals in the part of the program which prints out the players' individual figures.

The meaning of the following part of the program will be a lot easier to grasp now that the storage details have been given. The information is stored as character codes and these would give a most peculiar display if they were to be printed directly. When

information is being loaded into the array the command is likely to be something like:

```
LET E$(10) = CHR$ 50
```

and the line which reads the information out again is going to look like:

```
PRINT CODE E$(10)
```

You may not remember that at this stage in the program the number of runs scored by the batsman under consideration is held in A. The next few lines add one to the innings played, add one to the 'not out' tally if necessary, add the runs scored to the running total and check for a 'best so far this season' record.

load record into E\$

```
1500 LET E$=C$(Z(J),11 TO )
```

add 1 to innings played

```
1510 LET E$(1)=CHR$(CODE E$(1)+1)
```

update 'not out' record

```
1520 IF F$="N" THEN LET E$(2)=CHR$(CODE E$(2)+1)
```

check for a record and change 'best score' if found

```
1530 IF A>=CODE E$(5) THEN PRINT  
AT 3,24;"XXXXXXXX"  
1535 IF A>CODE E$(5) THEN LET E$(5)=CHR$ A
```

The number of 20s, 50s and 100s scored must be updated. Here is the code to do this job:

```
1540 IF A>=20 THEN LET E$(6)=CHR$(CODE E$(6)+1)  
1550 IF A>=50 THEN LET E$(7)=CHR$(CODE E$(7)+1)  
1560 IF A>=100 THEN LET E$(8)=CHR$(CODE E$(8)+1)
```

In the introduction to the chapter it was mentioned that there were some numbers that might need two bytes to hold them. These numbers are stored by dividing them by 256, putting the integer value of the result into one byte and the 'remainder' in the second byte. Such a number can be regenerated by multiplying the contents of the first byte by 256 and adding the answer to the contents of the second byte. Each element of a character array consists of 1 byte and so this allows the numbers to be stored in two elements of an array. Line 1570 adds the total already stored

in the computer to the variable A and lines 1580/90 store the result in the temporary file E\$. The average has to be stored in E\$(14) and is found by dividing the total runs by the number of innings in which the batsman was dismissed. If the batsman never was dismissed then the computer would be asked to divide by zero and this would cause a crash. Line 1572 checks to make sure that a dismissal has been logged. The computer is made to by-pass the section if the batsman was never out. Line 1574 makes sure that the maximum average score held in the files is 255 runs. If the average is needed in greater detail, then the full value is obtained by generating the player's personal records.

regenerate the total runs, by-pass if never out

```

1570 LET A=A+CODE E$(3)+256*CODE
      E$(4)
1572 IF CODE E$(1)-CODE E$(2)<1
      THEN GOTO 1580
1574 IF A/(CODE E$(1)-CODE E$(2)
      )>255 THEN LET E$(14)=CHR$(255)
1576 IF A/(CODE E$(1)-CODE E$(2)
      )>255 THEN GOTO 1580
1578 LET E$(14)=CHR$(INT (A/(COD
      E E$(1)-CODE E$(2)))

```

store in two bytes

```

1580 LET E$(4)=CHR$(INT (A/256))
1590 LET E$(3)=CHR$(A-256*INT (
      A/256))

```

don't bother with bowling figures if player didn't bowl

```

1600 IF B$="N" THEN GOTO 1920

```

The lines which allow the machine to accept the bowling figures allow a very similar pattern but the values are fed into three variables before the checking and storage routines. The 'runs scored' value is left in the variable A but the overs bowled is held in OB, the maiden overs in MA, and the wickets taken in WI.

all printing done at once

```

1610 PRINT AT 6,0:" BOWLING FIGU
RES ", "OVERS BOWLED?"
1620 PRINT "MAIDENS?", "WICKETS?"
      " ", "RUNS SCORED?"
1630 INPUT A$

```

check that A\$ is a number

```

1640 GOSUB 10

```

reject if A\$ is not a number

```

1650 IF X=1 THEN GOTO 1630
1660 LET OB=VAL A$

```

over-print old cursor with data and print cursor on next line

```
1670 PRINT AT 7,20;A$;TAB 20;"█
1680 INPUT A$
1690 GOSUB 10
1700 IF X=1 THEN GOTO 1680
1710 LET MA=VAL A$
1720 PRINT AT 8,20;A$;TAB 20;"█
1730 INPUT A$
1740 GOSUB 10
1750 IF X=1 THEN GOTO 1730
1760 LET WI=VAL A$
1770 PRINT AT 9,20;A$;TAB 20;"█
1780 INPUT A$
1790 GOSUB 10
1800 IF X=1 THEN GOTO 1780
1810 LET A=VAL A$
1820 PRINT AT 10,20;A$
```

The TABs will upset the Spectrum again so substitute:

```
1670 PRINT AT 7,20;a$;AT 8,20;"█
1720 PRINT AT 8,20;a$;AT 9,20;"█
1770 PRINT AT 9,20;a$;AT 10,20;"█
```

The operator now gets another chance to approve or reject the information before the data base is irrevocably altered. As soon as he keys "C", the values are all added to the values already stored in E\$. At line 1894 there is another check, which makes sure that the number of runs per wicket does not exceed 255. This line gives an alternative solution to the problem, which you may find more suitable for a similar problem in your own programs.

```
1830 PRINT ,, "KEY C IF OK"
```

signal that data is correct

```
1840 INPUT A$
```

reject if not

```
1850 IF A$<>"C" THEN GOTO 1610
1860 LET E$(9)=CHR$(CODE E$(9)+
0B)
1870 LET E$(10)=CHR$(CODE E$(10)
)+MA)
1880 LET E$(11)=CHR$(CODE E$(11)
)+WI)
```

regenerate the two byte number, runs conceded

```
1890 LET A=A+CODE E$(12)+256*COD
E E$(13)
```

avoid dividing by zero

```
1892 IF CODE E$(11)<1 THEN GOTO
1900
```


avoid exceeding 255

```
1894 IF INT (A/(CODE E$(11))) > 25
5 THEN LET A=255*CODE E$(11)
1896 LET E$(15)=CHR$ INT (A/(COD
E E$(11)))
```

store in two bytes

```
1900 LET E$(13)=CHR$ (INT A/256)
1910 LET E$(12)=CHR$ (A-256*INT
(A/256))
```

All that remains is to store all the information in the main filing system; line 1950 sees to that. The NEXT J line keeps sending the computer back for more information until the last man in the team has been logged, and then the computer returns to the menu at line 900.

```
1920 PRINT AT 15,0;"KEY NEWLINE
■ TO CONTINUE"
1930 INPUT A$
1940 CLS
1950 LET C$(Z(J),11 TO )=E$
1960 NEXT J
1970 GOTO 900
```

Remember that ENTER should be used in place of N/L or NEWLINE in Spectrum programs.

After that mammoth effort the rest of the sections of the program are much shorter. The next one prints out the merit order of batsmen and bowlers. It is assumed that a ZX printer is not available and the lines are a little more involved than they would be if the printer was used. The main problem when printing a lot of data onto the screen is that the computer gives an error code when the screen fills up and stops running. The extra code is needed to check that there is enough room on the screen before results are printed there. At the end of this section you will find a short piece on which lines to delete and which to change, in order to drive the printer.

The variables A and B are loaded with the worst possible batting and bowling figures, and then the computer works through the files looking for better results. Each time a result is found that is better than the result in A or B, the better result is placed in the appropriate store. The result of this procedure is that A and B will eventually contain the best figures from the whole club. The computer then starts searching through the records again, looking for and printing the names of all the members with the current score held in A or B. When all these have been printed it starts looking through again, this time looking for the second best figures and so on. Eventually the machine will print all of the club members in order of their performance at batting or bowling.

Here is the code which searches the files to find the best figures. The section is preceded with a message, which warns the user that the figures may take a time to be produced. If an inexperienced user is in the driving seat, then switching into FAST mode and losing the screen display will be worrying, and the time penalty of working in SLOW mode seems the lesser problem. Line 2052 ignores all the players with no bowling figures, or the machine will end up with zero stored in B every time, and this would be a waste of the time spent on the search.

```
2000 PRINT " BATTING AND BOWLING
AVERAGES "
2010 PRINT " THE FIGURES WILL TA
KE A WHILE "
```

worst batting figure

```
2020 LET A=0
```

worst bowling figure

```
2030 LET B=255
```

P holds the number of players

```
2040 FOR J=1 TO P
2050 IF CODE C$(J,24) >A THEN LET
A=CODE C$(J,24)
```

ignores non-bowlers

```
2052 IF CODE C$(J,25)=0 THEN GOT
O 2060
2055 IF CODE C$(J,25) <B THEN LET
B=CODE C$(J,25)
2060 NEXT J
2070 CLS
```

Only eighteen results are printed on the screen to avoid overfilling the display. The variable X is used to count the names as they are printed. A is set to zero to start with, and incremented each time a name is printed. When X reaches eighteen, the user is asked to " KEY NEWLINE " to clear the screen for the next eighteen names. If less than eighteen players with the same score are found, then the screen will be cleared anyway, and the opportunity given to go onto the next section. It is not likely that the fine detail of the batting figures of the worst batsman will be needed all that often, so an escape is required.

Spectrum users should substitute ENTER wherever they encounter N/L or NEWLINE in the blocks of code which follow.

```
2080 PRINT " BATTING FIGURES " , "
NAME" , "AVERAGE"
```

name counter

```
2090 LET X=0
2100 FOR J=1 TO P
```

ignores all players with other batting averages

```
2110 IF CODE C$(J,24) <>A THEN GO
TO 2180
```

counts the names

```
2120 LET X=X+1
```

prints the names

```
2130 PRINT C$(J, TO 10), CODE C$(  
J, 24)
```

if more than eighteen names clears the screen and resets the counter

```
2140 IF X<18 THEN GOTO 2150  
2150 PRINT "KEY NEWLINE TO CONT  
INUE"  
2160 INPUT A$  
2165 LET X=0  
2170 CLS
```

decrements the current average

```
2180 NEXT J  
2190 LET A=A-1
```

stops at zero

```
2195 IF A=0 THEN GOTO 2300
```

allows user to go to bowling

```
2200 PRINT "KEY NEWLINE OR B FO  
R BOWLING"  
2210 INPUT A$  
2220 CLS  
2230 IF A$<>"B" THEN GOTO 2080
```

And now an almost identical section which does the same job for the bowling figures. The bowling averages start low and increase, so B is incremented until the value reaches 255. Again, there is an opportunity to leave the loop to return, this time to the menu at line 900.

```
2300 PRINT "BOWLING FIGURES", "  
NAME", "AVERAGE"  
2310 LET X=0
```

P holds the number of players

```
2320 FOR J=1 TO P
```

ignores all players with other batting averages

```
2330 IF CODE C$(J, 25) <> B THEN GO  
TO 2400  
2340 LET X=X+1  
2350 PRINT C$(J, TO 10), CODE C$(  
J, 25)  
2360 IF X<18 THEN GOTO 2400  
2370 PRINT "KEY N/L TO CONTINUE"  
2380 INPUT A$
```

keeping the screen clear

```
2390 CLS
```

```

2395 LET X=0
2400 NEXT J

```

increments current average

```

2410 LET B=B+1
2420 PRINT "KEY N/L OR 3 TO 3TO
P
2430 INPUT A$
2435 CLS

```

allows user to go back to the menu

```

2440 IF A$="5" OR B>255 THEN GOT
D 900
2450 GOTO 2300

```

Earlier, it was mentioned that the listing for this section could be a lot shorter if the ZX printer was available. Here is a version of the program segment with all the unnecessary lines removed and the few alterations that need making in place. Lines 2075 and 2420 won't be needed for the Spectrum.

LPRINT replaces all the print statements, otherwise the lines are very similar in action to the earlier version.

```

2075 FAST
2080 LPRINT " BATTING FIGURES ",
"NAME", "AVERAGE"
2100 FOR J=1 TO P
2110 IF CODE C$(J,24) <> A THEN GO
TO 2180
2130 LPRINT C$(J, TO 10), CODE C$
(J, 24)
2180 NEXT J
2190 LET A=A-1
2195 IF A=0 THEN GOTO 2300
2200 GOTO 2100
2300 LPRINT
2310 LPRINT " BOWLING FIGURES ",
"NAME", "AVERAGE"
2320 FOR J=1 TO P
2330 IF CODE C$(J,25) <> B THEN GO
TO 2400
2350 LPRINT C$(J, TO 10), CODE C$
(J, 25)
2400 NEXT J
2410 LET B=B+1
2420 IF B=256 THEN SLOW
2430 IF B=256 THEN GOTO 900
2450 GOTO 2320

```

These lines are given as a Spectrum alternative but there is no reason why ZX81 users should not use them. The result of the change will be that the computer will present a screenful of information at a time and less use of the NEWLINE key will be needed. Add LPRINT statements or substitute them for PRINT. There is a lot

of scope for tailoring this section to the exact needs of your club.

```
2190 LET a=a-1
2200 IF a>0 THEN GO TO 2100
2210 PRINT "Key ENTER or B for b
cetting"
2220 INPUT a$
2230 CLS
2240 IF a$="b" THEN GO TO 2300
2250 GO TO 900
2300 PRINT "BOWLING FIGURES" "Na
ne"."Average"
2310 LET x=0
2320 FOR j=1 TO P
2330 IF CODE c$(j,25) <> b THEN GO
TO 2400
2340 LET x=x+1
2350 PRINT c$(j, TO 10),CODE c$(
j, 25)
2360 IF x<18 THEN GO TO 2400
2370 PRINT "Key ENTER to continu
e"
2380 INPUT a$
2390 CLS : LET x=0
```

Line 3040 will have to be split for the Spectrum.

```
3040 IF x=1 THEN GO TO 3020
3045 IF VAL a$>P THEN GO TO 3020
```

The program segment starting at line 3000 prints out the individual players' personal records. There is no need for an alternative routine to drive the printer because it is a simple matter to include the COPY command at a convenient place in the lines. The user is first asked for the player's code. The screen is cleared after the input is thoroughly checked, as before, and the report is presented.

```
3000 PRINT " INDIVIDUAL FI
GURES"
3010 PRINT ",,"KEY THE PLAYERS CO
DE"
3020 INPUT A$
```

check that it is a number

```
3030 GOSUB 10
```

reject if necessary

```
3040 IF X=1 OR VAL A$>P THEN GOT
O 3020
3050 CLS
```

Once A\$ has been checked, it is used to find the correct pieces of information in the main file, C\$. It is possible to present these figures to eight places of decimals and the bowling figures are printed to this level of detail. To remind the reader of the method of trimming off excess decimal places, the batting figures are organised to two decimal places only by line 3110.

VAL A\$ points to the correct string in the array C\$. The element is selected by the second number

```
3060 PRINT "PLAYER NO.":A$,C$(VAL
L A$, TO 10),,, "INNINGS",CODE C$(
VAL A$,11)
3070 PRINT "NOT OUT",CODE C$(VAL
A$,12)
```

two byte number

```
3080 LET A=CODE C$(VAL A$,13)+25
6*CODE C$(VAL A$,14)
3085 PRINT "BEST SCORE",CODE C$(
VAL A$,15)
```

B holds number of times the player was out

```
3090 LET B=CODE C$(VAL A$,11)-CO
DE C$(VAL A$,12)
```

avoids dividing by zero

```
3100 IF B=0 THEN GOTO 3120
```

prints to two decimal places

```
3110 PRINT "TOTAL RUNS",A,"AVERA
GE",INT (A/B*100+.5)/100
3120 PRINT "TWENTIES",CODE C$(VA
L A$,16)
3130 PRINT "FIFTIES",CODE C$(VAL
A$,17)
3140 PRINT "HUNDREDS",CODE C$(VA
L A$,18)
```

The bowling figures are found by some very similar lines.

```
3150 PRINT ,,"BOWLING FIGURES"
,"OVERS",CODE C$(VAL A$,19)
3160 PRINT "MAIDENS",CODE C$(VAL
A$,20)
3170 PRINT "WICKETS",CODE C$(VAL
A$,21)
```

two byte number

```
3180 PRINT "RUNS CONCEDED",CODE
C$(VAL A$,22)+256*CODE C$(VAL A$,
,23)
```

avoids dividing by zero

```
3184 IF CODE C$(VAL A$,21)=0 THE
N GOTO 3190
3186 PRINT "AVERAGE", (CODE C$(VA
L A$,22)+256*CODE C$(VAL A$,23)
)/CODE C$(VAL A$,21)
```

allows the user to escape to the menu

```
3190 PRINT ,,"KEY N/L FOR A NEW"
,"PLAYER OR S TO STOP"
3200 INPUT A$
3210 CLS
3220 IF A$="S" THEN GOTO 900
3230 GOTO 3000
```

There has to be a section which allows the names of the players to be entered into the computer. It is possible to add new names at any time during the season, but not to change the identity of a player once the code number has been allocated and figures have been fed into his file. The best thing to do if a member leaves the club, and another joins, is to keep both in the records. For this reason, the program should be set up to hold a few more than the number of players in the club, at the start of the season. To start off this part of the program, the user is given some information on the organisation of the storage and is asked to specify the particular group of club members. The input checks make sure that the response is a real number and that it is within a suitable range.

the lack of a space in line 4010 is deliberate

```

4000 PRINT "          ENTERING PLAYER
S NAMES"
4010 PRINT "THE PLAYERS ARE LI
STED IN GROUPS OF TWENTY. KEY THE
GROUP NUMBER AND THEN NEWLINE"
4020 PRINT "KEY THE NAME
CORRESPONDING TO", "THE NUMBER Q
UOTED IN THE RECORDS"
4030 INPUT A$
4040 GOSUB 10

```

P holds the number of players

```

4050 IF X=1 OR VAL A$>P/20 THEN
GOTO 4030

```

N is set to the first member of the specified group of twenty

```

4060 LET N=VAL A$*20-19

```

If some of the names have previously been entered, then the next part of the program will print these in place in the list which will appear on the screen. The ZX81 is put into FAST mode to reduce the delay, but this can be omitted, if it is likely to worry inexperienced users.

```

4070 CLS
4100 FAST

```

print twenty numbers

```

4105 FOR J=N TO N+19

```

print names if held in the file

```

4110 PRINT AT J-N,0;"PLAYER NO."
;J,C$(J, TO 10)
4120 NEXT J

```

The computer will respond to several signals during the next part of the program. First, if a name is entered followed by NEWLINE or ENTER then the name will be printed in place and logged in the files. If NEWLINE is keyed on its own, the computer will leave the existing name in place and go on to the next number in

the list. If "Z" is keyed followed by NEWLINE, then the machine returns to the menu at line 900. Finally, if "1" is keyed the computer will move back one place in the list. The user is made aware of the latest position in the list by way of a cursor which moves up and down, printed by line 4140 and removed by line 4200.

```
4125 SLOW
4130 FOR J=N TO N+19
```

cursor

```
4140 PRINT AT J-N,15;"█";
4150 INPUT A$
```

names or signals are fed into A\$

```
4155 IF A$="Z" THEN GOTO 900
4160 IF A$="1" OR A$="" THEN GOT
O 4200
4170 PRINT A$
4180 LET C$(J, TO 10)=A$
```

remove the cursor

```
4200 PRINT AT J-N,15;" "
```

move back up the list if "1" is keyed

```
4210 IF A$="1" THEN LET J=J-2
4220 NEXT J
```

If "N" is not keyed in response to the next input request, the machine returns automatically to the menu. When "N" is signalled, the user is asked for the number of the next group to be entered.

back to the menu if N is not keyed

next 20 players, menu if last player in

```
4230 PRINT "KEY N FOR NEXT GROU
P"
4240 INPUT A$
4250 IF A$(<)"N" THEN GOTO 900
4260 LET N=N+20
4270 IF N>P THEN GOTO 900
4280 CLS
4290 GOTO 4100
```

There are two parts of the program which allow the program to be saved onto tape in such a way that it starts running in the correct place when it is loaded back into the computer. The first one sends the machine straight to the menu and is the SAVE routine used when the user selects option 5 to file the records on tape.

```
5000 PRINT AT 9,9;" START TAPE
";TAB 9;"KEY NEWLINE"
5010 INPUT A$
5020 CLS
```

safer for programs which are loaded and saved many times

```
5030 LET A$="CRICKET"
5035 SAVE A$
5040 GOTO 900
```


This one Spectrum line replaces these six ZX81 lines:

```
5000 SAVE "cricket" LINE 900
```

The second SAVE routine is used only once a year to set up the system for the new membership. The program can be CLEARED before the computer is sent to this section. All the necessary variable initialisation is organised in this short section, which also asks the user to decide how many groups of twenty players will be needed to hold all the information. Some of the loops assume that there will be twenty players in each group, and if there are not, an error message results. If the storage is set up in units of twenty players, then no problems of this nature will cause the program to stop running.

```
6000 SAVE "CRICKET"  
6010 PRINT "HOW MANY GROUPS OF 2  
0 PLAYERS", "WILL YOU NEED ROOM F  
OR?"
```

the usual checks. 17 lots of 20 is the most that the program will deal with

```
6020 INPUT A$  
6030 GOSUB 10  
6040 IF X<>0 OR VAL A$>17 THEN G  
OTO 6010
```

P holds the number of players

```
6050 LET P=VAL A$*20
```

main storage

```
6060 DIM C$(P,25)
```

last team and date

```
6070 LET M$=""  
6080 LET D$=""
```

menu

```
6090 GOTO 900
```

Spectrum users should use line 6000 below:

```
6000 SAVE "cricket" LINE 6010
```

VARIABLES USED IN THE CRICKET PROGRAM

C\$	Main storage file (number of players, 25)
D\$	Date of last match recorded
M\$	Name of opponents in last match recorded
E\$	File used during up-dating
A\$	Temporary storage for information of temporary importance
B\$	Temporary storage used when the ones above need to be preserved
F\$	
Z\$	
Z	Team array (11)
P	Number of player files set up
J & K	Loop counters
OB	Overs bowled
MA	Maiden overs
WI	Wickets
A	Temporary number store
N	Temporary store while A is in use
X	Flag set to 1 if A\$ cannot be translated to a number by VAL

This program may well be passed to a club member who is completely unfamiliar with computers, so full user's notes are given in the following pages. If notes of this type are to be useful they have to be quite extensive, and have to be written in language that is completely non-technical. This is the reason why the style of the text changes so dramatically over the next pages.

CRICKET OPERATING INSTRUCTIONS

It you are unfamiliar with computers, here are some instructions which will help you to use the ZX computer when it has been programmed with the CRICKET program. This program allows the user to store all the figures on all the matches played by your club during a season. When the program is loaded into the computer for the first time, it will ask a few questions, and the answers that you give will allow it to set aside all the space it needs to hold all the information that you will be giving it. This part of the program will not appear again, it only needs to be used once. In order to use the program, you will need a fresh cassette of tape with at least ten minutes playing time on each side. You will also need a list of the players, numbered from 1 upwards.

The computer is programmed to check on all information as it is fed in, and not to accept information which is obviously wrong or

which will exceed its capacity. Until you get used to the program, you will find that you make keystrokes which seem not to have any effect. This is the computer rejecting keystrokes which might cause trouble. Try again, this time making sure that you read exactly what is printed on the screen. If, for any reason, the screen suddenly becomes covered with strange text: 1) don't panic, 2) don't press any keys until you have re-read these notes.

The recipe to get you out of trouble is:

- a) Press the key marked SHIFT and hold it down while keying "1" with the other hand. This is not as difficult as it sounds!
- b) Press the NEWLINE or ENTER key.
- c) Press the G key and then the 1 key. The bottom line of the screen should now read GOTO 1. If it doesn't, go back to a).
- d) Press NEWLINE or ENTER again and the menu should appear.

The likelihood of this happening is very small but it is as well to be prepared. Load the program using the instructions in the manual. The name of the program is CRICKET. When using the ZX81 the screen will eventually clear and you will be asked some questions. If the screen changes from horizontal stripes to a fuzzy pattern, the load has not been successful and another attempt should be made at a different volume setting.

You will be asked for the number of groups of twenty players that have to be dealt with. Make sure that you allow enough room for the players who might join the club later in the season. As soon as you key in the number of groups and then key NEWLINE or ENTER the computer will present you with a selection of several options. This list is known as the menu, and to choose from it all that is necessary is to key the number opposite the option. Option 4, Entering Names, is a good one to start with. The ZX81 NEWLINE key is called ENTER on the Spectrum computer. If you are using a Spectrum press the ENTER key whenever NEWLINE or N/L appear in these instructions.

ENTERING NAMES

The computer will tell you how the records are stored and ask for a group number. The number that you key will have to be followed by NEWLINE. This key will have to be pressed after each piece of information that is fed in, and from now on will be abbreviated to N/L. The screen will go blank for a while and then a list will be presented. Obviously, there are no names in the list at this stage, but later, when some have been fed in, they will appear in the list. A cursor (a black square) is printed opposite the first member of the list, and will later move up and down the list indicating your position.

Either Enter a name and then key N/L.

The name will appear in the list and the cursor will move down.

- Or Key N/L alone.
The cursor will move down. This is useful later when a name has to be added to the list because the other names can be jumped over.
- Or Key 1 and then N/L.
The cursor will move back one place in the list.
- Or Key Z and then N/L.
You will be returned to the menu.

When the list is finished you will be asked to key either N, N/L or N/L alone. N, N/L allows you to go onto the next group and N/L alone sends you back to the menu. If you key a group number that exceeds the number of groups that you specified earlier, then you will be sent back to the menu automatically.

MATCH FIGURES

Key 1 at the menu and then key in the name of the opponents, followed by the date of the match. Don't forget to key N/L after each. When asked if the information is correct, Key C, N/L unless there is something wrong, when you should key N/L alone. If you don't key C, the information will be asked for again.

When PLAYER 1 appears on the screen, key in the code number of the first member of the team, N/L. Carry on keying in the code numbers and building up the team list. Correct information is signalled in the usual way. If there is a mistake, the message at the bottom of the screen will alter. Key the team number at which the mistake is noticed, N/L. Key the correct player code for this position in the team, N/L. At first the cursor will move to the position of the mistake, and then the new name will appear in the list and you will be asked to check for mistakes again. You may make as many alterations as you like before entering C for correct.

A series of questions will appear on the screen next, and the cursor will indicate the question that has to be answered next. Key Y for Yes, N for No or a number as appropriate. For technical reasons the machine will not accept more than 255 runs in one innings. If a player has a score in excess of this number, then key 255 and make a note of the number of runs that will have to be added to his next score, in the next match that he plays. Mistakes are corrected by retyping the data. If you have answered N to the question "DID HE BOWL", then you will not be asked to enter any bowling figures. The computer will continue to ask for match figures until the last man has been dealt with, and then it will return you to the menu.

INDIVIDUAL RECORDS

Key 3 at the menu and you will be asked to supply a player code. The machine responds to this by printing out the records to date

on the player you have specified. The machine will continue to ask for player codes until you key S, N/L, when it will return you to the menu.

CLUB FIGURES

Key 2 for this option and the machine will explain that the figures will take a while to compile. After a short pause, the screen clears and a moment or two later the first name should be printed along with the players batting average. This is the best average in the club but only to the nearest whole number. Several members may be printed in the same section and their averages may be slightly different. To sort out the fine detail, look at the individual records after this routine has finished.

Eventually, you will be asked to key NEWLINE and before long you will be given the chance to indicate that you don't need any more batting figures by keying B, N/L. This will take you on to the bowling figures which are presented in exactly the same way. Remember to make notes of the screen display before keying N/L.

If the ZX printer is attached to the machine, and the program you are using is the version designed to drive the printer, then you will have nothing to do but watch as the computer and the printer together make a permanent record of the club members' performances to date.

FILE RECORDS

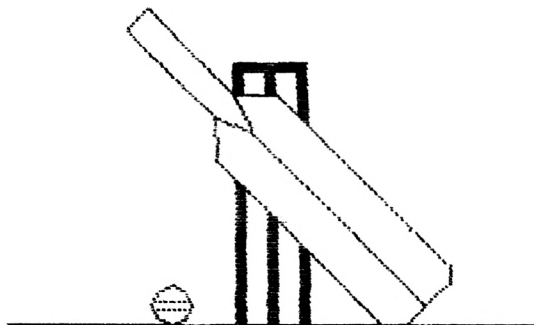
Fit the tape cassette into the recorder and wind the tape back to the start. Connect the MIC socket of the tape recorder to the MIC socket on the ZX81 by means of the lead supplied but, DO NOT CONNECT THE EAR SOCKETS. Key 5 and the computer will ask you to start the tape and key N/L. Push both the PLAY and RECORD buttons on the tape player and allow a few seconds before keying N/L to allow the tape speed to settle. When using the ZX81 the screen will blank for five seconds and then change to a pattern of dancing, horizontal stripes for several minutes. Eventually the screen should clear and the menu should be seen on the TV. Turn the tape over and run it back to the start, and repeat the above process to make a second safety copy of the records.

Spectrum users will see the dancing lines in the screen border only and will be told "OK" when the job is over. Look up the use of the VERIFY command in the Spectrum handbook. This will allow you to check that the records have been recorded accurately.

NEXT YEAR

Load the program from the master tape again and you will be

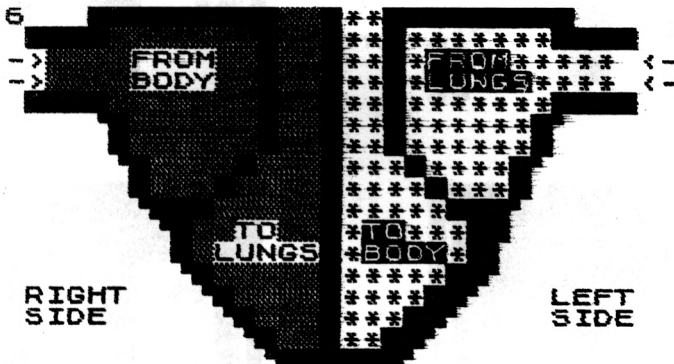
asked the question about the number of groups of twenty players.
The names will have to be re-loaded for the next season's
records.



8 *Draw and Move*

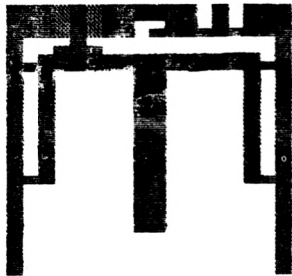
One person's moving graphics is no more than another's static display with a simple figure moving across it. If ZX81 generated displays are to compete seriously with other media, then the quality will have to be a lot more sophisticated than this. This package will allow you to produce animations in which the figures move across the background which, itself, can be moving. The only limit is the amount of skill and imagination that you can bring to the computer, and it would seem that ZX enthusiasts are not short of either of these.

The animation is generated as a series of frames, which are then presented in sequence on the screen. If one of the excellent, alternative character set, extension boards is available, then the results can rival the output of much larger computers with high resolution boards. Even using the Sinclair 'chunky' graphics the results can be very effective. Here are a few of the frames which have been produced on the package. The first is from the tape designed to accompany this book. It is one of the frames from a beating heart animation. The second is a set of stills from another program on the tape, the four-stroke engine cycle. The last example is from a set produced using the alternative character generating board.



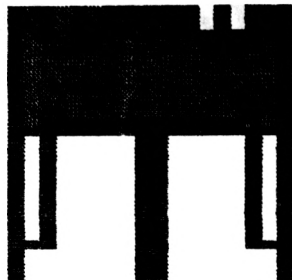
The alternative menu presentation style has been adopted here, because it is often important to change to another option without losing the screen display, and so CLS followed by a new menu page is out of the question. The menu is in the form of a block of inverse characters which changes without anything other than a press of the NEWLINE key. The option to be selected from the menu is indicated by keying any other character when the correct option

1



INDUCTION

5



INDUCTION

2



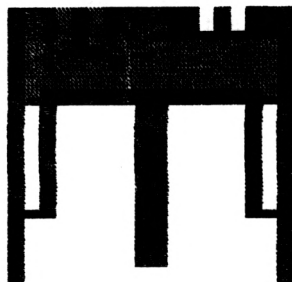
INDUCTION

6



COMPRESS-
ION

3



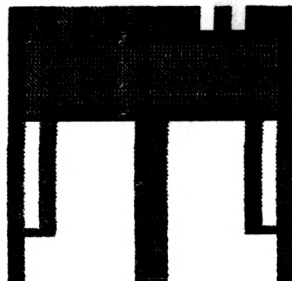
INDUCTION

7



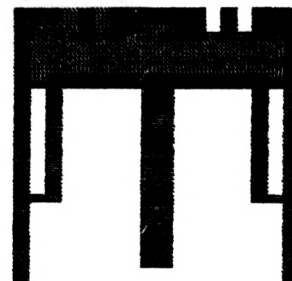
COMPRESS-
ION

4



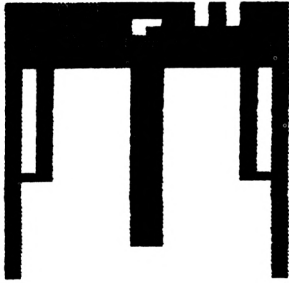
INDUCTION

8



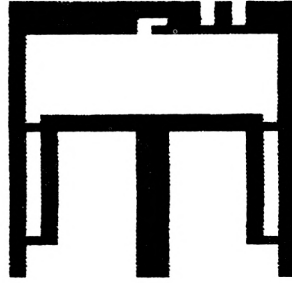
COMPRESS-
ION

9



IGNITION

13



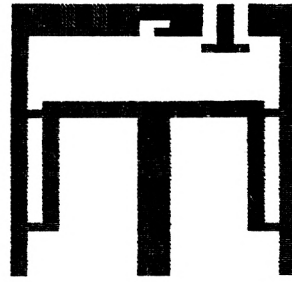
POWER

10



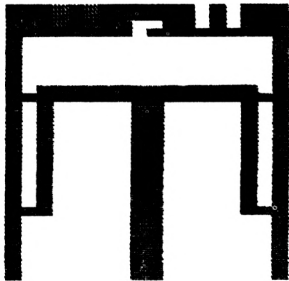
POWER

14



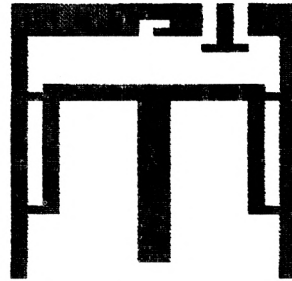
EXHAUST

11



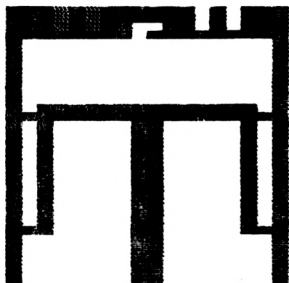
POWER

15



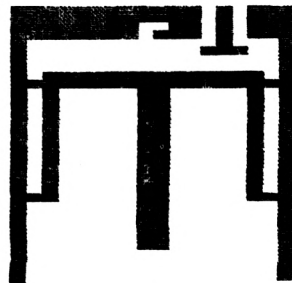
EXHAUST

12

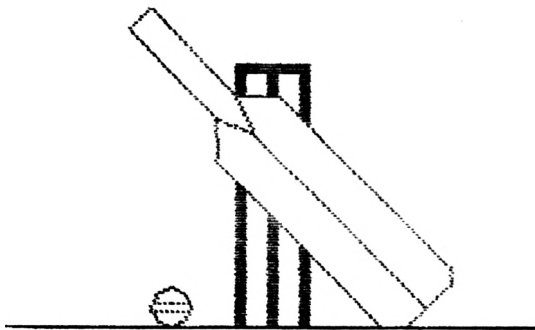


POWER

16



EXHAUST



is in the box. "L" is the most obvious one to use because it is right next door to the NEWLINE key.

There are seven choices from which the user can select an option. The first of these is called INFILL. Infill allows the user to select a broad brush and paint in the basic colour of the background as an artist might use a colour wash. The 'colours' that can be used are the different characters from the available character set. The whole screen may be painted in quickly, or selected areas can be filled with different blocks of characters.

DRAW is the second option and allows the user to select a colour to draw with. This time only single characters are used to produce a line drawing effect, but any other character may be selected at any time during the construction of the drawing. While in this option, it is also possible to move over previously drawn sections without altering them, to move to any of the other options, and to save the finished drawing in the memory.

CHANGE, the third option, allows the user to specify any one character, and this can be simply a space, and then indicate another character to replace all the examples of the first character in that frame. In this way the spaces which make up the background can, for example, be changed to black squares to invert the image.

REWORK is a simple option to use and to program. All that happens is that the user specifies one of the frames that was filed and then reworks it, possibly after watching the sequence, making use of the next option.

DISPLAY starts the ZX81 cycling through the frames that have been drawn. This allows the user to see the effectiveness or otherwise of an idea. Display and rework together allow the fine tuning of the animation for best effect.

RESET causes the ZX81 to scrap any existing frames, ask for the number of frames to be used in the next sequence, and then allocate the space.

FILE, the last option, allows the computer to save all your handiwork onto tape.

Two versions of the program are available. The differences are small but far reaching in their effects. It is the work of a few minutes to change from one version to the other, so the choice is not important at this stage. When the package is complete, both approaches can be tried out and the one best suited to the user's needs selected. The difference between the versions is the method used to store the finished frames. In the first version they are held in an array called P\$, which is dimensioned to the number of frames and to the number of characters in each. This approach is convenient because the existing frames can be cleared very quickly from the variables store by re-dimensioning the array. The other advantage of storing the pictures this way is that the ZX81 only has to set aside the minimum of space for each animation. The problem with this method is that the computer has a lot of re-organisation to do after filing one of the frames, and this makes for quite a wait after using the save routine. This is a particular nuisance when writing animations which make full use of the available space.

Method two is borrowed from the word processor in the first book in this series. A set of two line sub-routines is organised. Each routine sets A\$ to 576 spaces and then returns to the main program. The different frames are stored by POKEing the display into the different sub-routines. This sounds more complicated than it is, and a few minutes looking at the program will make things a lot clearer. This method is much faster to use, but the space will have to be set up each time the program is used for a new animation, by means of the EDIT function. The first frame takes a long time to type in, a space at a time, but this job has to be done once only, no matter which method of storage you finally opt for. The array method will be given in detail at first and then, at the end of the chapter, the alterations needed to change to the alternative style will be explained.

The basic plan of the program follows the, now familiar, pattern of all the others in the book. The sections are in the same neat, 1000 line blocks and the menu is still at line 900 in spite of the different menu method employed this time. The auto-start routine is built into the file routine to avoid accidental loss of the frames when the RUN or CLEAR commands are used. The second method of storing the frames, however, makes them immune from this danger. The first line by-passes the sub-routines and sends the ZX81 off to the menu, and the first sub-routine is common to both versions of the program.

The next sub-routine allows the user to move a cursor over a frame without altering the work that has been done already. It is necessary to store the position into which the cursor is about to move and the character which is stored there *before* the cursor moves. Lines 201 and 202 store these values and then line 203 pokes the cursor, an inverse F for FAST, into place. Line 204 reads the keyboard and sets B\$ to the result. If no key is pressed, then B\$ is equal to the empty string. Lines 205 to 208 then set the new cursor position according to the key which was pressed in this cycle, and then lines 209 and 210 make sure that you don't wander out of the frame and possibly overwrite important characters. B, you will remember, holds the address of the start of the display file, and B+594 is the address of the end of the frame allowing for the NEWLINE characters at the ends of the lines on the screen. Hence NEWLINE characters must be left untouched or the ZX81 will crash. Line 211 gently moves the print position one further on if a NEWLINE character, code 118, is PEEKed at the new position. The original character is poked back in its correct place before the next cycle by line 212. The signal to return to the slow drawing mode is a press on the SLOW key, which is D on the keyboard. Line 213 makes this final check before the computer is sent back to the start of the routine.

```
200 REM FAST
```

store this position

```
201 LET A1=A
```

store the character

```
202 LET A2=PEEK A
```

print a cursor

```
203 POKE A,171
```

read the keyboard into B\$

```
204 LET B$=INKEY$
```

backspace

```
205 IF B$="5" THEN LET A=A-1
```

down a line

```
206 IF B$="6" THEN LET A=A+33
```

up a line

```
207 IF B$="7" THEN LET A=A-33
```

forward a space

```
208 IF B$="8" THEN LET A=A+1
```

keep inside the frame

```
209 IF A<B THEN LET A=A+33
```

```
210 IF A>B+594 THEN LET A=A-33
```

avoid NEWLINE

```
211 IF PEEK A=118 THEN LET A=A+
1
```

replace character

```
212 POKE A1,A2
```

re-cycle if signal not given

```
213 IF B$<>"D" THEN GOTO 200  
214 RETURN
```

The menu comes next. The first thing that the menu does is to find the first frame and then print it on the screen. This is because the user will need to see the frame before deciding what has to be done next. Lines 900 and 910 find and then print the frame. Line 920 finds the display file starting address and so automatically sets the print position to the middle of the screen, and sets B to the first byte in the file. Lines 930 to 937 define the menu display in a way that makes it convenient to print in place. Line 938 sets E\$ to a line of spaces which will be convenient to use later to remove unwanted text on the screen. Notice the way the start of the words in the inverse blocks are staggered. This is to make it obvious that the display in the black box is changing.

find and print 1st frame

```
900 GOSUB 2  
910 PRINT AT 0,0;A$
```

find display file and set B to start and A to half way through available menu options

```
920 GOSUB 100  
930 DIM D$(7,9)  
931 LET D$(1)=" INFILL ""  
932 LET D$(2)=" DRAW ""  
933 LET D$(3)=" CHANGE ""  
934 LET D$(4)=" REWORK ""  
935 LET D$(5)=" DISPLAY ""  
936 LET D$(6)=" RESET ""  
937 LET D$(7)=" FILE ""
```

empty line for clearing unwanted text

```
938 LET E$=""
```

"

The first option is set to "INFILL" by setting the variable C to 1. Line 940 prints option 1 and then line 950 waits until a key-stroke is made. If this is just a press on the NEWLINE key, the computer adds one to C, in effect, moving to option 2. If the option number reaches eight by repeated use of the NEWLINE key, then the ZX81 re-sets the option to 1 at line 970. The usual method of sending the computer to the line number at which the selected routine starts is used in a form which is only slightly modified. Line 980 selects the correct piece of the program if any other key is pressed before NEWLINE. If this signal is not given, then 990 returns the machine to the start of the section.

option = 1

```
939 LET C=1
```

print option in place

```
940 PRINT AT 20,10;D$(C)
```

select or reject

```
950 INPUT B$
```

next option if reject

```
960 IF B$="" THEN LET C=C+1
```

reset to 1 after option 7

```
970 IF C=8 THEN LET C=1
```

GOTO section if selected

```
980 IF B$<>"" THEN GOTO C*1000
```

recycle if not

```
990 GOTO 940
```

The first option, INFILL, allows the user to fill specified areas of the screen quickly and avoids having to paint in a background with single characters. Line 1000 allows the user to specify the width of the infill and the characters which go to make it up. The infill could be made up of different characters and this would give a striped background, or it could be made up of all the same character. The starting position of the infill is set by lines 1020 to 1050. Each of the messages is printed on the same line, line 20. The menu options were printed on this line as well and all of the text will share this line. The new message is designed to over-print the earlier ones, so watch out for a few extra spaces at the end of some of the print lines. When the line has to be cleared this is done by print E\$, the empty line, on line 20.

specify the infill

```
1000 PRINT AT 20,0;"KEY INFILL S  
TRING "  
1010 INPUT C$
```

screen co-ordinates

```
1020 PRINT AT 20,0;"DOWN HOW MAN  
Y?  
1030 INPUT M  
1040 PRINT AT 20,0;"ACROSS HOW M  
ANY?"  
1050 INPUT N
```

remove unwanted text

```
1060 PRINT AT 20,0;E$
```

To keep the infill confined to the frame being worked on, the values of M and N have to be kept to the limits defined in lines 1070 and 1080. When the ZX81 is sure that the co-ordinates are within bounds, the string can be printed by line 1090. The next five lines do a similar job to the lines between 205 and 208. The escape signal is the one which is used throughout the programs in this book, Z, and using this signal returns the user to the main drawing sequence.

keep within bounds

```
1070 IF N<2 OR N>17 THEN LET M=2
1080 IF N<0 OR N>31 THEN LET N=1
```

print in specified position

```
1090 PRINT AT M,N;C$
```

read keyboard into B\$

```
1100 LET B$=INKEY$
```

move the print position

```
1110 IF B$="5" THEN LET N=N-1
1120 IF B$="8" THEN LET N=N+1
1130 IF B$="6" THEN LET M=M+1
1140 IF B$="7" THEN LET M=M-1
```

escape to draw sequence or recycle

```
1150 IF B$="Z" THEN GOTO 2005
1160 GOTO 1070
```

Now comes the main drawing sequence and this time there are a few more commands which the ZX81 will respond to. On page 104 is a diagram which shows the keys which give these commands. The up, down, left and right signals have been supplemented by some diagonal movement commands, which are made by the keys 9, 0, O and P. The other available commands have been given keys which will make the job of remembering them a lot easier. The A key is associated with the NEW command and is used to select a new character. The D key is associated with the SLOW command and the F key with FAST, so these keys switch the ZX81 between the slow drawing sequence and the fast cursor moving routine at line 200. The S key is the SAVE key in BASIC and this key has been allocated to the 'save frame' routine.

Line 2000 removes any text left behind by another section and line 2005 allows the user to specify the character with which to draw. The full range of single characters can be used, including the graphics characters, but don't use the multiple characters such as <= or RUN because these will disrupt the screen.

clear away text

```
2000 PRINT AT 20,10;E$
```


new character

```
2005 INPUT A$
```

print on screen at A

```
2010 POKE A, CODE A$
```

read the keyboard into B\$

```
2020 LET B$=INKEY$
```

go to FAST mode

```
2030 IF B$="F" THEN GOSUB 200
```

down/left

```
2040 IF B$="O" THEN LET A=A+32
```

select a new character

```
2050 IF B$="A" THEN GOTO 2000
```

down/right

```
2060 IF B$="P" THEN LET A=A+34
```

backspace

```
2070 IF B$="5" THEN LET A=A-1
```

up/left

```
2080 IF B$="9" THEN LET A=A-34
```

forward

```
2090 IF B$="6" THEN LET A=A+1
```

up/right

```
2100 IF B$="0" THEN LET A=A-32
```

down a line

```
2110 IF B$="8" THEN LET A=A+33
```

up a line

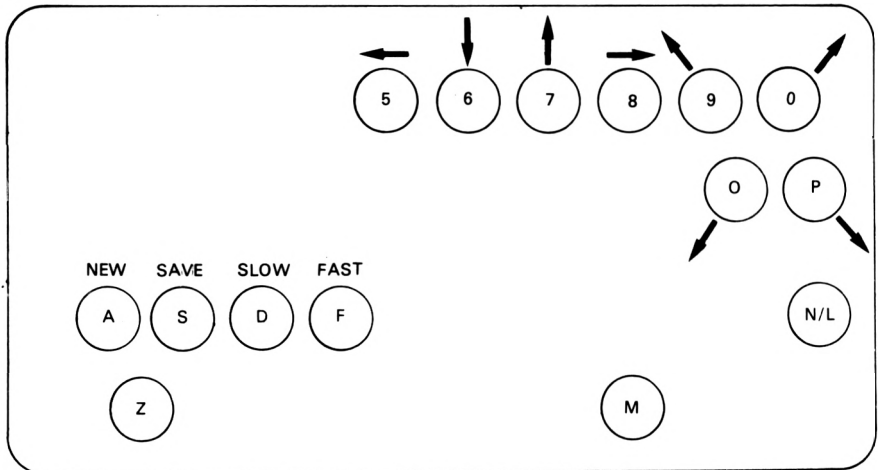
```
2120 IF B$="7" THEN LET A=A-33
```

go into SAVE sequence

```
2130 IF B$="5" THEN GOTO 2190
```

If the character to be printed over is a NEWLINE character, then the display would crash and take the rest of the program with it. The possibility is avoided by line 2140 which checks for NEWLINE characters, code 118, and adds one to the print position if it finds one. Lines 2150 and 2160 keep the print position inside the frame. The first line is safe 99% of the time but, in spite of elaborate checks, I have occasionally had a crash when trying to push the print position up from line one on the screen. For this reason I have played safe and set the minimum value of A to thirty-three bytes on from the start of the display file. The computer is told to POKE A with 184 at line 2170. The effect of this is to print an inverse S as a cursor in the print position. The S is used to indicate the slow drawing mode.

Command keys used to drive DRAW.



The keys 5 to 0, O and P move the print position around the screen in the direction indicated by the arrows. The diagonal movements are only available in DRAW mode.

While drawing, the keys A to F have special functions which are related to the key words printed on the keyboard. A new character is selected after the A(NEW) key is pressed. Pressing F(FAST) allows the cursor to be moved quickly over the screen without altering the existing drawing. D(SLOW) sets the user back into normal drawing. S(SAVE) allows the user to save the picture in the memory by keying Z, or gives the user access to the menu by keying M. Keying NEWLINE at this stage makes the ZX81 assume that the SAVE key was pressed by accident and returns the user to the drawing sequence.

As well as its normal function of confirming entry, the NEWLINE key is used to skip over choices from the menu until the correct one is displayed.

avoid the NEWLINES

```
2140 IF PEEK (A) = 118 THEN LET A = A+1
```

keep within the frame

```
2150 IF A < B+33 THEN LET A = A+33
2160 IF A > B+594 THEN LET A = A-33
```

cursor

```
2170 POKE A, 184
```

recycle

```
2180 GOTO 2010
```

If the user keys S then there are three options open. First keying Z confirms that the SAVE routine is wanted. Secondly, keying M returns the user to the menu, and thirdly NEWLINE alone indicates that the S was a mistake. Line 2190 indicates the options and the response is fed into B\$. The text is removed at 2210 and then the lines 2220 and 2230 send the ZX81 off to the correct section.

remind user of options

```
2190 PRINT AT 20,0;"KEY Z TO FILE,M FOR MENU"  
2200 INPUT B$
```

remove text

```
2210 PRINT AT 20,0;E$
```

save routine

```
2220 IF B$="Z" THEN GOTO 2240
```

menu routine

```
2225 IF B$="M" THEN GOTO 940
```

back to draw

```
2230 GOTO 2140
```

The saving is done in FAST mode to keep the delay to a minimum. The screen is located at line 2250 and then the variable K is set to the byte number in memory, which is occupied by the first character of the string, set up in the sub-routine at line 2. L holds the line number of the screen so that the loop at 2280 can poke a line at a time from the screen into the sub-routine. The screen is scrolled each time the loop finishes, so that the next line of the display can be poked in place. L is incremented each time the screen scrolls, and when L equals 19 the job is finished and the computer can break out of the loop. Each time a new frame is added to an animation, the new one is likely to be very similar to the first, so the next job is to look at the sub-routine at line 2 and then print this new version of A\$ onto the screen. The frame is stored in memory by line 2360. P\$ is set up a little later on and holds all the frames of the display as empty spaces at first. If it is at all difficult to understand how this section works, remove the FAST command at line 2240 and then save a frame. The changing display will support the explanation, but don't forget to replace the FAST line or you will be in for a slow transfer of information from one part of the memory to another.

fast transfer

```
2240 FAST
```

find start of screen

```
2250 GOSUB 100
```

1st byte of A\$

```
2260 LET K=16532
```

line no. 1

```
2270 LET L=1
```

transfer a line at a time

```
2280 FOR J=1 TO 32
2290 POKE K+J,PEEK (B+J)
2300 NEXT J
```

move display up a line

```
2305 SCROLL
```

next 32 bytes

```
2310 LET K=K+32
```

next line

```
2320 LET L=L+1
```

stop if last line

```
2330 IF L=19 THEN GOTO 2350
```

recycle

```
2340 GOTO 2280
```

set A\$ to new frame

```
2350 GOSUB 2
```

copy frame into file

```
2360 LET P$(F)=A$
2370 SLOW
```

You will have noticed that the current frame number F was used to allocate the frame to the correct element of the file array. If F is equal to the last frame in the sequence, then the animation is finished and the computer goes off to display your handiwork. If this is not the last frame, then the frame counter is incremented and the computer returns to the menu for further instructions. The part of the menu chosen as the return target is the part which prints A\$ onto the screen. This will be the last frame which was saved, and is ready for amendment to make the next frame in the sequence.

last frame so display

```
2380 IF F=G THEN GOTO 5000
```

remember last frame

```
2390 GOSUB 2
```

increment frame counter

```
2400 LET F=F+1
```

menu

```
2410 GOTO 910
```

The next facility makes for some interesting results. You are invited to indicate a character used in the display which you would like changed. The invitation comes on line 20 as before. When you indicate a character, you are asked for a new character with which to replace the old one. Things have to be speeded up for the exchange, so the computer is switched to FAST mode and then each character in the display is checked, and all the ones which are unwanted are replaced with the preferred character. To finish off, the message is removed by line 3080, the ZX81 is switched back to SLOW mode and then returns to the menu.

unwanted character

```
3000 PRINT AT 20,0;"CHARACTER TO  
CHANGE?"  
3010 INPUT B$
```

replacement character

```
3020 PRINT AT 20,0;"KEY THE NEW  
ONE"  
3030 INPUT C$  
3040 FAST
```

search from start of frame to end

```
3050 FOR J=B+33 TO B+594
```

exchange if old character found

```
3060 IF PEEK J=CODE B$ THEN POKE  
J, CODE C$  
3070 NEXT J
```

remove old text

```
3080 PRINT AT 20,0;E$  
3090 SLOW
```

menu

```
3100 GOTO 940
```

If that one was a little short the next section is shorter still. The job that has to be done is to reset the frame number to one which requires a little work doing on it. The only check that needs making is the one at line 4015, which rejects frame numbers beyond the maximum in the animation.

new frame number

```
4000 PRINT AT 20,0;"KEY THE FRAM  
E NUMBER"  
4010 INPUT F
```

reject those beyond limit

```
4015 IF F>G THEN GOTO 4010
```

print new frame and remove old text

```
4020 PRINT AT 0,0;P$(F);AT 20,0;
E$
```

back to drawing

```
4030 GOTO 2005
```

The display routine is no longer than the last. The computer counts through the frames which make up the animation, printing each of them in turn over the last one. Line 5020 checks that the keyboard has not been touched once every cycle, and sends the computer back to the menu if a keystroke has been made.

counts up to the last frame

```
5000 FOR J=1 TO G
```

prints the frame in place

```
5010 PRINT AT 0,0;P$(J)
```

checks that no signal has been given to return to the menu

```
5020 IF INKEY$("<") THEN GOTO 940
5030 NEXT J
```

re-cycles

```
5040 GOTO 5000
```

The routines at lines 6000 and 7000 are a little more substantial but only just so. The RESET routine is at line 6000. It asks how many frames to set aside memory for, rejecting any number above eighteen, the maximum number for which there is space. Throughout the section, E\$ is used to remove old, unwanted text.

E\$ to remove old text

```
6000 PRINT AT 20,0;"HOW MANY FRA
MES?";E$
```

maximum number of frames

```
6010 INPUT G
```

reject if not enough room

```
6020 IF G>18 THEN GOTO 6010
```

remove text

```
6030 PRINT AT 20,0;E$
```

clear old frames and set aside space for new

```
6040 DIM P$(G,576)
```

set frame number to 1

```
6050 LET F=1
```

menu

```
6060 GOTO 900
```

Because the program may be saved several times on tape before the animation is completed, it is wise to include the extra line in the auto-save routine to avoid problems due to the inversion of the final character in the program name. If the name is placed into a string before the program is recorded, then the main source of the name will not be corrupted by the SAVE routine.

```
7000 PRINT AT 20.0; "START TAPE  
KEY NEWLINE"
```

signal when ready

```
7010 INPUT B$
```

name in string for safety

```
7020 LET B$="DRAW"  
7030 SAVE B$
```

ready for 1st frame

```
7040 CLS
```

menu

```
7050 GOTO 900
```

And that short section finishes the first version of the program. Here are the changes needed to convert your package to the second method of storing the frames. The first job is to decide how many frames are needed for your animation. Once this decision has been taken, key LIST 2, NEWLINE and then SHIFT/1 to get the first sub-routine at the bottom of the screen for editing. Rub out the line number and replace it with 4. Key NEWLINE again to place the new line in the list, and then key SHIFT/1 again, and repeat the process until the right number of sub-routines have been made. Finish all of these off with RETURN lines. The 'LET A\$=' lines should have even numbers and the 'RETURN' lines odd.

Line 900 is the next that needs attention and should be edited to:

```
900 GOSUB 2*F
```

and then everything remains the same until line 2260 which should read:

```
2260 LET K = 16532 + 593 * (F - 1)
```

Line 2360 is deleted and line 2350 becomes:

```
2350 GOSUB 2 * F
```

Lines 4020 to 4040 are different and need changing to:

```
4020 GO SUB F*2
4030 PRINT AT 0,0;A$;AT 20,0;E$
4040 GO TO 2005
```

And there is a new line after the revised version of line 5010

```
5010 GO SUB J*2
5015 PRINT AT 0,0;A$
```

The reset routine is not really viable in this version of the program.

To avoid a lot of alteration. this is the simplest solution to the problem:

```
5000 PRINT AT 20,0;"NOT AVAILABL
E IN THIS VERSION"
5010 INPUT B$
5020 PRINT AT 20,0;E$
5030 GO TO 940
```

And to finish the new program off, a few lines need to be added to the auto-restart routine.

G is the number of frames in the sequence.

```
7050 LET F=1
7060 LET G=5
7070 GO TO 900
```

The details of how to use this and the previous version of the DRAW package are to be found in the operating notes. In the meantime here are the variables used in the programs.

VARIABLES USED IN DRAW

A\$	current frame, 576 characters long
B\$	temporary storage
C\$	characters used in the INFILL routine
D\$	the menu
E\$	an empty line
P	main file dimensioned to the number of frames and to the length of a frame. (1st version of the program only)
A	current print position
B	start of the display file
A1/2	current character and position stores

C current menu option
F current frame
G last frame in use
J loop counter
K first byte of the storage file
L line number
M&N screen co-ordinates

OPERATING INSTRUCTIONS FOR THE DRAW PROGRAM

When the program is complete, key CLEAR/NEWLINE to get rid of any clutter and then give the command GOTO 7000. Start your tape recorder and allow the tape speed to settle before pressing NEWLINE. From now on the program will behave as if it were just loaded from tape. On the screen is the first frame which will probably be blank. On the tape which accompanies the book, I have printed a border round the screen to show the extent of the frames. Option 1 is automatically on offer; don't select this option at first, but keep pressing NEWLINE until the RESET option is indicated. Key L and then NEWLINE. Any key will do but L is convenient. The computer will ask for the number of frames that you wish to book. Give it the answer and you will be returned to the menu at option 1, so we might as well start work at this section.

INFILL. Each time the menu option is selected by keying any key before using the NEWLINE key. You will be asked to enter the string with which you wish to fill part or all of the screen. Key a short row of dots and then NEWLINE, and then key 5 for down and 5 for across. The string of dots will be placed on the screen, five lines down and five columns in from the left hand side. Use the cursor control keys to move the string around on the screen and fill as much space as you need, before using the Z key to indicate that you wish to escape from the routine. The escape is to the main drawing sequence.

DRAW. The input prompt is waiting for a character with which the ZX81 will help you draw your picture. Key SHIFT/9 to select graphic characters and then SHIFT/H to give a grey square. Key NEWLINE twice to place the square in the middle of the frame. The cursor flashes gently to indicate your exact position in the frame. This will become more important as more of the grey squares are printed; it would be difficult to know which one was the last without the cursor. Move the square about on the screen, using the cursor control keys and the additional ones which are indicated in the diagram on page 104. Key A to select a new character for the drawing and then NEWLINE to insert it at the last print position. Try using the F key to move into fast mode. The keys 5 to 8 are the only ones in use in this section. When the print position, indicated at all times by the cursor, is set to the desired point, key D for slow and continue drawing. Use the S key to save the frame when it is finished. There are three options open to you

when you key S. You can opt to save the frame by keying Z, or you can move over to the menu by keying M. The last alternative is to key NEWLINE alone to indicate that S was a mistake and that you wish to carry on drawing. Choose Z to file the frame, and the screen will blank for about ten seconds, and then gradually fill with the frame that you have just filed. You will now be back at the menu and can opt to move back into draw to produce the next frame, or move on to any of the other options. Select CHANGE.

CHANGE. The ZX81 first asks which of the characters on the screen you would like replaced, and then asks for the character with which to replace them. As soon as you indicate your choice, the screen will blank again and then clear after a short while to show the new version of the frame. You will be at the menu again at this stage and will have to select DRAW if you wish to save the new frame.

DISPLAY. This option will not be of much use until the animation is complete. When the last of the frames is filed, the computer automatically sends you to this section. To escape, simply press any key except the break key, and you will return to the menu.

RESET. This destroys all the frames except the one in the sub-routine at line 2. You will need to tell the computer exactly how many frames it will need for your next animation. The computer returns to the menu and displays the frame held in the sub-routine at line 2. If this is not suitable for the new sequence, then use the infill option to fill the frame with spaces.

REWORK. This is the simplest of the options to use and to program, but the effect is quite spectacular. The existing frame is replaced instantly with the frame that you have specified. The current frame number is reset at the same time, so the reworked frame is filed in its old position. This option does not return the user to the menu but directly to the drawing routine, because that is the most likely routine to be needed if a frame needs alteration.

FILE. This last option is the means by which the finished display may be saved onto tape. Allow the tape speed to settle before keying NEWLINE to start the ZX81's SAVE routine.

The second version of the program is used in exactly the same way but the RESET option will not be available. The program will have to be prepared for use by using the EDIT function, as described on page 109, to prepare empty frames. Don't forget to write a new version of line 7060, better still, key LET G = (the number of frames that you have set up) as a direct command. Start the program running by keying GOTO 7040.

When you have produced a sequence which you would like to run regularly, it would cut down on the LOAD time if you were to delete all the program, except those sections concerned with the

delete all the program except those sections concerned with the presentation of the frames. A full eighteen frames takes about six minutes to load in, so the two minutes saved by deleting the bulk of the program would give a very useful time saving.

9 *Effective 1k Programming*

Schools and colleges could find the ZX81 very useful indeed. The small machine does not suffer from the inconveniences which so often put teachers off using other computers in their lessons. The ZX81 is very portable, easy to set up and use. The price means that schools can afford a lot of them. The more computers that are available, the less difficult it is to find one when you need it. As soon as the RAM pack is plugged in, however, the convenience of the ZX81 drops considerably. If the RAM pack is needed for a program then it will take longer than the fifteen seconds that 1K programs take to load. When dropped on the floor, a 1K ZX81 will be still running its program, in most cases, whereas eager students knocking into the table may be enough to crash a 16K computer. The RAM pack almost doubles the cost of the computer and so reduces the number that will be available. The less convenient an educational resource is to use, the more likely it is that it will turn into a dust-gathering white elephant. There is a need for effective programs which will fit into the 1K ZX81.

This chapter shows how the 1K ZX81 can be programmed to run science simulations but, as usual, the working programs are only half of the story. The techniques which make the squeezing of the programs into the small space are at least as important. The science simulations will be of interest to some readers, but the methods used to produce them will enable many other teachers to produce 1K programs in their own areas. The first few programs described are simply small programs, written conventionally and no difficulty should be experienced in reading the listings. The later programs would not fit if they were to be written conventionally, and the tricks used to squeeze them in make the listings difficult to follow, so the amount of documentation of the programs increases as the chapter progresses.

These programs are mainly the ones which I have written to help my own students in their science courses. My students much prefer displays which move to those which consist of static text and numbers. The only one which does not include movement of any kind is included for another reason; it is the sort of program which does its job well but no better than other, more convenient techniques such as a blackboard and chalk. The carbon dating program is really a 'one line' program which does no more than a calculator. Any novelty will soon fade as computers are used more extensively. Where other resources are just as good, it seems silly to spend the effort in setting up the computer. The other programs are

designed to relieve the students of the need to hold all of the ideas that make up a concept in their heads at once. If the computer can present one of the more difficult ideas on the screen, then more mental effort can be applied to the other ideas and to the relationships between them.

CARBON DATING

Not another computer dating service, but a way of calculating the age of an article from the number of counts per second on a geiger counter, and the amount of carbon in the sample.

```
10 PRINT "THERE IS ENOUGH C14
IN AIR TO GIVE 12.5 COUNTS PER
MINUTE PER GRAM OF CARBON"
20 PRINT "HOW MANY COUNTS PE
R MINUTE FROM EACH GRAM OF CARBO
N IN THE SAMPLE?"
30 INPUT A
40 LET T=LN (12.5/A) / .00012115
50 PRINT "THE SAMPLE IS ABOU
T ";INT T,"YEARS OLD"
```

This is a small program and it fits with no problems. It could be smaller still because only lines 30 to 50 are really necessary. It might be thought worthwhile keying these three lines into the ZX81 and using it as a programmable calculator.

Compare the marginal benefits of this program with the real educational value of the next program, Half Life.

HALF LIFE

If you start with a very large number of radio-active atoms, the laws of probability predict that the time taken for half of the atoms to decay is the same for any sample of the same element. This time is called the half life of the element. The idea is difficult for school students to understand, and a computer simulation would help them to hold the ideas in their heads while they struggle to build up the concept. The ZX81 deals with a sample of 256 atoms (stars) and the half life that it produces varies between 25 and 35 seconds, but is usually quite close to 30 seconds. This inaccuracy, due to the small sample of atoms, can be put to good use in emphasising the random basis of the radio-active decay process.

Having printed the set of atoms on the screen, the ZX81 generates the position of the atoms to be obliterated as a location in the part of the memory which holds the display file. The location is checked to see if an atom exists and, if so, a counter is incremented. At first the computer cannot miss because there is an atom in every location, but as the program progresses, the likelihood of printing a space over an already obliterated atom becomes

larger and larger. The rate of decay falls but the half life stays roughly constant because less have to decay in each half life.

These first lines print the atoms in place:

```
1 FOR J=1 TO 256
2 PRINT "*";
3 NEXT J
```

Lines 4 and 5 set up the screen for later results and set the decayed atom counter to zero:

```
4 PRINT AT 9,0;"1/2 LIFE="
5 LET X=0
```

Next, the two bytes which hold the number of frames sent to the TV, have to be set to the maximum possible value (they count backwards to zero). It is these bytes which will be PEEKed later to find the number of TV frames that have been printed since the last setting. This number divided by fifty is equal to the number of seconds that have passed, because fifty frames are sent to the TV each second.

```
6 POKE 16436,255
7 POKE 16437,255
```

It is possible that the byte in the display file selected for obliteration could contain a newline character, code 118. Poking this would disrupt the display and crash the program. Line 8 finds the start of the display file and then adds a random number (0 to 265) to it. Line 9 checks for newline characters and sends the computer back for a new byte if one is found. Line 10 adds one to the value found in the decayed atom counter X if a hit has been made and the byte contains the code for a star, 23. Finally, the value of the byte is changed to zero, resulting in a space being printed in the screen.

```
8 LET A=PEEK 16396+256*PEEK 1
6397+AND*265
9 IF PEEK A=118 THEN GOTO 8
10 IF PEEK A=23 THEN LET X=X+1
11 POKE A,0
```

Line 12 checks that a half life has not elapsed before the ZX81 is sent back to generate another target atom. Half lives will have elapsed when 128, 192 and 224 atoms have been removed. A programming snag means that one has to be added to the counter X at each half life, so the numbers to be checked for are 128, 193 and 226.

```
12 IF X=128 OR X=193 OR X=226
THEN GOTO 20
13 GOTO 8
```


Lines 20 and 40 make the machine count up to a number that varies between zero and two hundred and fifty. This gives a random delay of up to ten seconds, during which time the machine checks that the keyboard is not touched. If a key is pressed, then INKEY\$ will not equal the empty string mentioned at line 30, and the computer will go off to line 200 and respond to the attempted dishonesty.

```

50 POKE 16436,255
60 POKE 16437,255
70 PRINT AT RND*15,RND*25; "███"
80 INPUT A$

```

Lines 50 and 60 set the TV frame counter to the maximum value, and line 70 produces the signal at random co-ordinates on the screen. The machine now waits for the NEWLINE key to be pressed, and as soon as it is, it calculates the delay from the value held in the TV frame counter, which drops by one each fiftieth of a second.

```

90 PRINT (65535-PEEK 16436-PEEK
K (16437)*256)/50-0.19; " SECONDS
"

```

The 0.19 of a second is the time that the machine takes to do all of its part of the job, and so this has to be deducted from the user's delay. The next three lines wait for a signal, clear the screen and go back for another attempt:

```

100 INPUT A$
110 CLS
120 GOTO 10

```

And, finally, the cheat's just desserts:

```

200 FOR J=1 TO 17
210 PRINT AT J,RND*25; "███"
220 NEXT J
230 GOTO 100

```

The "cheat" message can be printed in inverse graphics and they will appear at random positions along the top seventeen lines of the screen.

To use the program, key RUN and NEWLINE and then wait for the signal. Don't keep your finger on the NEWLINE key for too long or you will get a cheat report. The machine will print out the reaction time for each attempt and keying NEWLINE again, after the time has been printed, will allow the user another go. If the CHEAT messages appear, the computer will wait for a press on the NEWLINE key before returning to the main program.

BOYLE AND CHARLES

These two found how the volume of a gas changed as the temperature and the pressure changed. This program is to inject a little lightness into a section of the science syllabus at a point where it becomes rather bogged down in endless calculations. The students have to calculate the volume of the gas after changes in the temperature and the pressure. Finally, they have to find the conditions which will just make the gas expand enough to blow up the apparatus.

The temperature is first set to 273° Absolute, the pressure to one atmosphere and the volume to five units.

```
1 LET T=273
2 LET P=1
3 LET A=5
```

And then the sides and end of a gas syringe are printed, using the same FOR/NEXT loop for all the jobs. Black squares are printed onto the screen at co-ordinates which are related to the loop counter, J. This multiple use of the same loop can often be used to save space in a program, and line 6 is particularly interesting because it extends the usefulness of the loop beyond the normal limit.

```
4 FOR J=0 TO 30
5 PRINT AT 0,J;"■";AT 5,J;"■"
6 IF J<=5 THEN PRINT AT J,0;"■"
7 NEXT J
```

The code to fill up the syringe with gas is in the form of two loops, one nested inside the other. The first loop counts up to the value of A which holds the volume of the gas. The second counter, K, is used to fill the syringe in vertical columns.

```
8 FOR J=1 TO A
9 FOR K=1 TO 4
10 PRINT AT K,J;"■"
11 NEXT K
12 NEXT J
```

The next loop draws the plunger of the syringe in place:

```
13 FOR J=1 TO 4
14 PRINT AT J,A;"■";AT 2,J+A;"■"
15 NEXT J
```

You will have noticed that PRINT AT can be used more than once in the same line which saves quite a lot of space. Only the word AT need be used to produce the second and subsequent pieces of print on the screen.

```
16 PRINT AT 6,0;A*10;" CC OF G
AS AT ";T;" ABS","AND ";P;" ATM"
```

The comma after the " ABS " is to start the next part of the text on the next line. Commas can be used to save memory when laying out the text on the screen because they cost only 1 byte each, and empty PRINT lines cost 6 bytes.

```
17 PRINT "CHANGE T OR P?"
18 INPUT A$
19 PRINT "NEW ";A$
20 IF A$="T" THEN INPUT T
21 IF A$="P" THEN INPUT P
22 LET A=50/2730*T/P
```

As with the carbon dating program, the arithmetic at the heart of the package is the simplest bit. The volume of the gas is divided by ten before it is stored in A, to keep the drawing of the gas within the dimensions of the syringe over a reasonable range. All that remains is to clear the screen and go back to the drawing routine if the gas volume is less than the volume of the syringe, or show what happens if it is too big for the container.

```
23 CLS
24 IF A<=27 THEN GOTO 4
25 PRINT "BANG"
```

DOPPLER

As you walk down the High Street, the ambulance comes towards you, making its usual alarm. As the sound source passes you, the pitch of the sound drops and you have to explain why to your inquisitive off-spring. This program shows how the sound, radiating from the source, must change in pitch as far as the stationary listener is concerned because of the way that the sound pulses reach him or her in different circumstances. This simple demonstration can be used for a variety of jobs. It shows the difference between a longitudinal wave like sound, and the more usual waves on the sea. It helps in the teaching of the doppler effect at a simple level, and has even been used to calculate the speed of stars retreating from us.

The space saving method used here is to keep the whole of the moving display in one string of characters, and to print this over and over again. The bulk of the program is used to change the contents of this string. At first the sound source is standing still at the left hand side of the screen, and a couple of variables need to be set up:

```

5 PRINT " STANDING STILL "
10 LET A=1
15 LET Z=0
20 DIM A$(31)

```

For the time being, A will be set to 1, and this is the position of the sound source on the screen, so this is where the pulses should start. The pulses are shown by printing a black square on the screen and then over-printing it with a grey square immediately. The grey square is part of the string and is flanked by two symbols which will, eventually, radiate from the source in two directions. At first, however, the left hand one gets lost immediately.

```

50 PRINT AT 9,A;"■"
60 LET A$(A TO A+2)=" ■■ "

```

And now the string has to be adjusted to give the impression that the pulses of sound are radiating from the source. Spaces are added into the string on either side of the source, and the rest of the string is shifted to either the left or the right. This happens four times between each pulse, so the whole thing is organised inside a loop which counts up to four:

```

70 FOR J=1 TO 4
.. 80 LET A$(2 TO A)=A$(2 TO A)+"
90 LET A$(A+2 TO )=" "+A$(A+2
TO )
100 PRINT AT 9,0;A$
110 NEXT J

```

All of which is difficult to follow unless you key it into the computer and watch it all happen before your eyes.

The next two lines allow the ZX81 to either continue with this static sound source, or change to a display in which the sound source moves:

```

120 IF INKEY$("<>") THEN LET Z=1
130 IF Z=1 THEN GOSUB 200
140 GOTO 50

```

"Z" acts as a flag which indicates that the source has to move and, once set, it remains so until the next RUN. All that happens at the sub-routine is that the value of A is increased by one. This has the effect of winking the sound source one position to the right. The lines 60, 80 and 90 work just as well under these circumstances, and so the source moves, pulses four times, and then moves again:

```
200 LET A=A+1
210 PRINT AT 0,0;"          MOVING
220 IF A=16 THEN GOTO 300
230 RETURN
```

Line 220 allows the user to measure the new wavelengths and so at 300:

```
300 PRINT ,,"MEASURE THE WAVELE
NGTHS","THEN PRESS NEWLINE"
310 INPUT B$
320 CLS
330 RUN
```

Run the program and you will see the pulses start on the left of the screen, and then radiate towards the right. Press the BREAK key after the program has been running for a while and then measure the wavelength of the sound. Key RUN/NEWLINE again to restart the display. Next you will need the time taken for the pulses to move across the screen, and the distance from the source to the point at which the pulses disappear. From all this information the frequency of the sound can be calculated. These measurements can be made while the program is running, and as soon as they have been made, press a key (any except the BREAK key) and the display will change. Half way across the screen the sound source will stop, and you will be invited to measure the wavelengths. They will be different on either side of the source. Where the source is catching up the sound pulses it has made, the wavelength will be shorter, and where it is moving away from its pulses, the wavelength will have increased. From here it is a short step to understanding why the pitch of the sound drops as the ambulance speeds past you. The waves are closer together as the ambulance approaches, and sound with a short wavelength has a higher pitch than the long wavelength sound you hear as the sound source goes away from you.

So far the programs have been fitted into the computer by a combination of making the same loop do a lot of work in different places, keeping the display to a minimum and making the action all take place along the same line. The 1K ZX81 has a display file which consists of NEWLINE characters only at first. The full screen of 704 characters occupies more space than anything else, and so it is not set up unless absolutely necessary. To prevent

the computer giving you "out of screen space" error messages and stopping the program, the amount of screen that you ask the ZX81 to use must be limited. Printing too far over to the right, as well as too far down, will cause problems. This next program solves right-angled triangles and draws them for good measure. The drawing is done by means of the PLOT command and starts at the bottom of the screen. The program is included here because it illustrates the difficulty of printing too far over on the right. Most of the time the program will run quite well, but a triangle with a long base and vertical side will give problems. This program is best used in the same way as the Boyles' and Charles' law program, as a source of the right answers to liven up lessons which would be a little dull otherwise.

TRIANGLE

```

10 PRINT "KEY SHORT SIDE ";
20 INPUT A
30 PRINT A
40 PRINT "KEY LONG SIDE ";
50 INPUT B
60 PRINT B

```

A is set to the length of the short side of the triangle, which will be printed up the left hand side of the screen. The long side will be printed along the bottom, and the length is held in B. This is a very short program. All that remains of it is a set of loops which draw the lines, one line of arithmetic and a very long print line, which presents all the results in some free space on the screen:

```

70 FOR J=1 TO A
80 PLOT 0,J
90 NEXT J

```

This loop draws the first side:

```

100 FOR J=1 TO B
110 PLOT J,0
120 NEXT J

```

This one draws the longer side along the bottom of the screen:

```

130 LET D=180/PI*ATN (A/B)

```

That is half of the mathematics out of the way. The other half is calculated in the lines which also print the display. This is a simple way to save on scarce memory; do the maths as it is needed and use it right away. If the maths is done separately, then at least one extra line is needed. Line 150 below calculates each part of the hypotenuse and uses the result to decide where to plot the point:

```

140 FOR J=1 TO B
150 PLOT J,A/B*(B-J)
160 NEXT J

```

And, finally, the line which prints everything else. It is so long because there is no need for more than one print line. The ZX81 will accept a line of basic which will fill the whole screen:

```
170 PRINT AT 21,0;"■";AT 3,12;"
AREA ";A*B/2,TAB 12;"HYP ";SQR
(A**2+B**2),TAB 12;"ANGLES",TAB
12;"TOP ";90-D,TAB 12;"BTM ";D
```

PERISTALSIS

Each time a string is set up by a line such as:

```
10 LET A$=" LOTS OF CHARACTERS "
```

the information is held in the machine twice, once in the program area and again in the variables store. Once the line has been used, there is no reason to occupy the space in the program area and so the line can be deleted. The obvious disadvantage of this method of getting larger programs into 1K is that the listings are difficult to follow. The programs are often no more than a series of PRINT statements. Peristalsis is just such a program. This is another idea which needs repeated demonstration and explanation before the idea sinks in. The computer is made to cycle through a series of seven strings, to produce an animated display of the way that food is pumped through the digestive tract by the rhythmical contraction and relaxation of muscles around the tubes.

To centralise the display, the strings are printed at character nine along the line and so the value nine is needed many times over. The variable B is set to nine in the first line of the program, and the TAB positions are given as B in the lines which print all the strings. The strings are printed in sequence on the first seven lines of the screen, and then the sequence has to be repeated a line lower for a further twelve times:

```
10 LET B=9
20 FOR J=1 TO 13
30 PRINT AT J,B;A$;TAB B;B$;TA
B B;C$;TAB B;D$;TAB B;E$;TAB B;F
$;TAB B;G$
40 NEXT J
```

which is most of the program. To tidy up the display before the next sequence is used, the following line is needed:

```
50 PRINT AT 14,B;A$;TAB B;A$;T
AB B;A$;TAB B;A$;TAB B;A$;TAB B;
A$
60 GOTO 1
70 SAVE "PERISTALSIS"
80 GOTO 1
```


cycling through the process as often as is necessary, so that a single piece of the whole can be concentrated upon each time. The machine doesn't get tired of telling the story and so it can be left running all during the lesson if that helps.

The problem with moving things round on the screen is that the previous display has to be removed before the next version is printed. Two methods are used here: printing spaces over the earlier display to remove the unwanted characters, and printing the required characters with a trailing space. The space clears up the only bit of the display which is left after the characters have moved one space along the line. This last method can only be used when moving horizontally and by one character each time. When the electrons have to move up and down the plates, the electron has to be replaced with a piece of plate and not a space. This is obvious when you key this part of the program into the computer, but what is not obvious at first is that the plates do not have to be printed in the first place. This was the breakthrough in writing this program down to 1K. Not enough of a breakthrough to do the whole job, but enough to make the idea of getting this complex display onto the basic ZX81 realistic. After the breakthrough the program was still quite a bit too big, but it was clear that the number 6 cropped up a lot in the lines, so a variable was set to this value and the variable used in the lines as often as possible. To pare the last bit of fat off the package, this variable was used to set up a pair of PRINT AT co-ordinates in the second line. A number in the listing of a program costs a byte of memory for each character and five more for the value. Line 20 looks as if it should cost more memory than:

```
PRINT AT 0,0;
```

but it is cheaper in memory by just enough to get the program into the available space. There are other opportunities to use this technique, but it seemed unnecessary to make the listing more difficult to read than it needed to be. For the sake of example though, line 30 could have been written as:

```
FOR J = A/A TO A+A
```

and a few more bytes would have been saved. Here is a version of the listing which works and is as readable as possible:

```
10 LET A=6  
20 PRINT AT A-A,A-A;" +  
   BATTERY _"
```

To make sure that this line is not mis-read it consists of three spaces, a plus sign, nine spaces, the word battery in inverse graphics, ten more spaces and a minus sign.


```

30 FOR J=1 TO 12
40 PRINT AT A,16-J; "    "
50 PRINT AT A,16+J; "    "
60 PRINT AT A,16-J; "-CL"
70 PRINT AT A,16+J; "NA+"
80 LET A=3*2
90 NEXT J

```

The little bit of arithmetic at 80 makes the ZX81 re-calculate the value of A and takes up some time. The display is slowed down a little and is just that bit smoother as a result. Finely graded delays can be programmed by fiddling with the maths functions in this way.

These lines have moved the chloride ion (-CL) and the sodium ion (NA+) from the middle of the screen to the places where the plates will be drawn by the next part of the program:

```

100 FOR J=1 TO A
110 PRINT AT 7-J,3; "■"
120 PRINT AT J-1,30; "■"
130 PRINT AT A-J,3; "- "
140 PRINT AT J,30; "- "
150 LET A=3*2
160 NEXT J

```

All of this printing could have been done in a single print line to save even more memory, but the listing is difficult enough to follow as it is. Line 170 tidies up the last bit of the plates and, this done, the electrons have to be made to flow into one side of the battery and out of the other.

```

170 PRINT AT A,30; "■"
180 FOR J=3 TO 11
190 PRINT AT 0,J; "- "
200 PRINT AT 0,J+18; "- "
210 NEXT J
220 GOTO 10

```

Because the program cycles, all the print statements, even the first ones, have to be "PRINT AT " so that the characters are in the right place the next time round.

Looking back at this collection of slimmed down programs for the basic ZX81, there are a lot of different methods which can be used to make effective programs fit. Before we move on from this chapter, it might be as well to re-cap on the methods used and produce a summary of the collection of techniques.

First, and most obvious, is to write small programs. Not every program that can be made to fit into the machine will be effective,

as I think the carbon dating program will demonstrate, but the rest of the small programs have something more valuable to offer. Basically, these present the most difficult of the ideas and allow the students to work on the rest.

The big memory drain is the screen. If you can keep the screen file to a minimum then the programs stand a better chance of fitting. There is no need to restrict yourself to the top line though. Printing on the middle line improves the appearance of the display but costs very little extra because the other lines are not filled.

Loops are the very basis of efficient programming and, if a single loop can be made to do several jobs at once, then the programming will become even more economical. Keeping the number of numbers in the listing to a minimum will reduce the memory cost of the programs to a quite surprising extent. If the program can be adjusted so that the same number crops up over and over again, then this number can be stored in a variable, and the variable name can be used in the place of the number. While talking about variables, there is no need to store the value of a variable in the variables store, and to keep the line instructing the computer to do this storing in the program list. The same information is stored twice. Set up strings by means of direct commands and release more space for more important jobs. Be prepared, though, for less readable listings as a result. The other points that were mentioned were the use of several PRINT AT statements in a single line, and the building of mathematics into print lines, to calculate both the results that have to be printed and the coordinates at which the printing has to be carried out.

These techniques take practice. It will be difficult to remember all of them when you start to write 1K programs, but this crib list might be used to help you find ways of getting rid of the final few bytes of fat on your programs, that will make the difference between ones which will fit and those which won't.

10 *Building Block Programming*

Half way through developing a program you often need to insert a little bit extra. Usually the habit of writing at ten line number intervals will allow the new code to be inserted but the neat progression of the line numbers will be spoilt and the result is somehow a little less professional. Sometimes there is not enough room for the extra code and the use of GO TO is demanded. The machine is sent off to some less crowded section of the list and sent back after the job is done. This solution works but makes the program very difficult to read at a later time when some amendment is needed. Renumbering a portion of the program allows the extra space to be made available and, if done carefully, the join won't be noticed.

This is only the start of the benefit to be gained from the ability to renumber your programs. In the chapter "At Least As Good", the idea of collecting a library of subroutines was introduced. The ability to renumber these and then to merge them together into a composite program has important implications. Adopting this method of programming will prevent the feeling that you are "re-inventing the wheel" every time you solve similar problems in different programs. Once a good technique is developed it can be renumbered, catalogued and placed in the library to be merged into any program which needs the service. As your library grows it will give consistency to your software and will improve the rate at which you write. The need for de-bugging should decrease if the tried and de-bugged routines are reused. Altogether renumbering is an important technique and should be available. These suggestions are only reasonable as long as there is a merge facility and this is only provided on the Spectrum so renumbering programs for the ZX81 will have to be supported by a merge program. ZX81 users will find this at the end of the chapter.

Several renumber programs are listed during this chapter, all aimed at different needs. There is a quick and simple BASIC program which just copes with the line numbers and doesn't bother with the GO TOs or GO SUBS. For more complex programs there is a longer, slower but more powerful BASIC program to look after the GO TOs and GO SUBS automatically. Then there are machine code programs for both machines which are very fast and resistant to the NEW command. These remember the line number at the end of the last program that was renumbered. They might be useful for their extra speed and for renumbering your collection of routines so that all the segments are numbered sequentially.

Before starting the job of re-numbering we need some information from the user such as where to start, where to finish and the gap between the lines. The machine also needs to know the new figure for the first line.

first byte in the program

```
9000 LET A=16509
9010 PRINT "RENUMBER FROM LINE?"
```

initial line to be changed

```
9015 INPUT IN
9020 PRINT IN, "STARTING AT?"
```

new first line number

```
9025 INPUT ST
```

last line to be altered

```
9030 PRINT ST, "UP TO?"
9040 INPUT FI
```

interval between lines

```
9050 PRINT FI, "GAP?"
9060 INPUT GA
```

The Spectrum version of the program is very similar and a listing is given a little later together with the necessary explanation to cover the differences.

Even though the process is quick there is no point in wasting time so the first line in the ZX81 program switches the computer into FAST mode and then the value of the first line number is fed into variable B. Next C is loaded with the number of bytes to be skipped over to get to the next line number and then a few checks have to be made. If the line number is less than the first one to be changed, IN, then the business part of the program is by-passed. If the line number is larger than the finishing line or gets to the re-numbering routine itself, then the process stops.

```
9070 FAST
```

line number (MSB first)

```
9100 LET B=256*PEEK A+PEEK (A+1)
```

line length (MSB second)

```
9110 LET C=4+PEEK (A+2)+256*PEEK
(A+3)
```

by-pass if too early

```
9120 IF B<IN THEN GOTO 9170
```

stop if finished.

```
9130 IF B>8999 OR B>FI THEN GOTO
9190
```

And now the part that does the renumbering. The starting line number is divided by 256 and the INTeGer result is fed into the first of the two bytes. The remainder is fed into the second byte. The next line number is set up by adding the interval GA to ST. The value of the variable A is then set to the address of the next line number and the process repeats until the job is done.

MSB first

```
9140 POKE A,INT (ST/256)
```

then LSB

```
9150 POKE A+1,ST-INT (ST/256)*256
```

add the interval to the line number

```
9160 LET ST=ST+GA
```

add the line length to the starting point

```
9170 LET A=A+C
```

another line to be treated

```
9180 GOTO 9100
9190 SLOW
```

Here is the routine in action. The first version of the program has a ragged sequence of line numbers. These have been neatly ordered in steps of one in about three seconds.

**PROGRAM TO BE RENUMBERED
RENUMBER ROUTINE IN PLACE**

```
10 LET T=200
20 LET S=T-T
30 LET A=T/T
40 GOSUB 22
50 LET B=A+RND*28
60 LET C=INT (RND*#R)+1
70 PRINT AT C,B;" "
80 FOR J=1 TO 15
100 NEXT J
105 IF INKEY#="F" THEN GOSUB 13
110 PRINT AT C,B
115 GOTO 5
120 PRINT AT A,0;
123 FOR J=1 TO B+4
126 PRINT "-";
129 NEXT J
132 IF A=C THEN LET S=S+B
135 LET A=A+1
138 IF A=16 OR T<0 THEN GOTO 25
150 CLS
170 LET T=T-B
190 PRINT AT 18,0;"SCORE ";INT
S,"AMO ";INT T
210 PRINT AT A,0;"█"
```

```

230 RETURN
250 PRINT "LANDED."
270 PAUSE J*J
290 CLS
310 RUN

```

SPIDER INVADERS

```

1 LET T=200
2 LET S=T-T
3 LET A=T/T
4 GOSUB 22
5 LET B=1+RND*28
6 LET C=INT (RND*A)+1
7 PRINT AT C,B;"■"
8 FOR J=1 TO 15
9 NEXT J
10 IF INKEY$="F" THEN GOSUB 13
11 PRINT AT C,B;" "
12 GOTO 5
13 PRINT AT A,0;
14 FOR J=1 TO B+4
15 PRINT "-";
16 NEXT J
17 IF A=C THEN LET S=S+B
18 LET A=A+1
19 IF A=18 OR T<0 THEN GOTO 25
20 CLS
21 LET T=T-B
22 PRINT AT 18,0;"SCORE ";INT
S,"AMO ";INT T
23 PRINT AT A,0;"■"
24 RETURN
25 PRINT "LANDED."
26 PAUSE J*J
27 CLS
28 RUN

```

The program works by the way. It is called SPIDER INVADERS and will fit into the 1K ZX81. When you think that the spider is in your gunsights, press the firing button, F. You can't miss at first but scoring gets progressively more difficult. When the spider lands or you run out of ammunition, the game is over. Your score and remaining ammunition are displayed at the bottom of the screen at all times. To make a better score wait until the spider is in your sights but a long way off. You get extra points for long range marksmanship. For details of the space saving lines which made the program run in 1K, see the chapter on effective 1K programming.

The only real difference in the Spectrum renumber program is that the first byte of the program area may wander around in memory a little so we will have to check its whereabouts in the system variables. The information is in bytes 23635 and 23636. The program is a little more compact because of the Spectrum's ability to handle multi-statement lines. Spectrum users will have to wait for a while for their games program in this chapter. Here is the listing for now.

SPECTRUM BASE RENUMBER

```

9900 LET a=PEEK 23635+256*PEEK 2
3636
9910 INPUT "Renumber from line?"
,i: INPUT "Starting at line?",s
t: INPUT "Up to?",fi: INPUT "Gap
between lines?",ga
9920 LET b=256*PEEK a+PEEK (a+1)
: LET c=4+PEEK (a+2)+256*PEEK (a
+3): IF b<i THEN GO TO 9950
9930 IF b>9899 OR b>fi THEN GO T
O 9960
9940 POKE a,INT (st/256): POKE a
+1,st-INT (st/256)*256: LET st=s
t+ga
9950 LET a=a+c: GO TO 9920
9960 PRINT "Renumbered"

```

If you wanted to re-number a series of programs and then store them on tape in their new forms, the re-number routine would have to be added to the listings on the ZX81. The Spectrum can merge programs but still cannot delete part of a program automatically. The re-number routine has to be deleted, line by line, from each program that is treated. This would not be a problem if the re-number routine was written in machine code and stored in an area of memory which is safe from the effects of the NEW command. The ZX computers can be fooled into thinking that they have less memory than they in fact have. The system variable called RAMTOP can be poked with a smaller value for the end of the memory and the computer will only bother to clear the memory up to this new location. If the machine code is stored in the area above the new RAMTOP then it will be safe and can be used for many different programs. The program which is described later starts each new program off at a line number which is ten higher than the last line it re-numbered in the earlier run. This is to allow the construction of the sub-routine library mentioned earlier. If this facility is not required, then a simple POKE will start the next program off at line ten again.

To do justice to machine coding would take a book at least as long as this one, so only the barest of explanations is given for these sections of program. Readers who are interested in extending their programming in this direction may find the books in the list at the end of this chapter useful. The program is given in the form of a set of numbers which have to be poked into place somewhere convenient. The site chosen is the area of memory from 30000 onwards. This will leave a reasonable space in all the computers for the programs which are to be treated. If you wish to treat a long program then it is quite a simple task to adapt the program and make more space.

The machine code re-number routine looks like a long program but it occupies only fifty-seven bytes compared with the three hundred and fifty needed to hold the BASIC version of the routine.

MACHINE CODE RENUMBER ROUTINE MNEMONICS

LD BC, (30062)	next line number to use
LD A, B	most significant byte (MSB) in accumulator
LD DE, (30060)	next byte containing a line number
LD (DE), A	poke MSB into 1st byte of line number
INC DE	next byte of line number
LD A, C	least significant byte (LSB) in accumulator
LD (DE), A	poke LSB into 2nd line number byte
LD H, B	} transfer next line to HL register
LD L, C	
LD BC, (30064)	load interval into BC register
ADD HL, BC	add interval to next line number
LD (30062), HL	store next line number to use
INC DE	next byte, LSB of line length
LD A, (DE)	store in A
LD L, A	transfer to L of HL register
INC DE	move to MSB of line length
LD A, (DE)	load into A
LD H, A	transfer to H of HL register
LD BC, 4	4 bytes extra for the line length and number
ADD HL, BC	HL now contains the length of the line
LD BC, (30060)	first byte of line number
ADD HL, BC	add to convert HL contents to 1st byte of next line
LD (30060), HL	record in store for next loop
LD BC, (16396)	load start of the display file
SBC HL, BC	subtract start of DF from 1st byte of next line
JP NZ 30000	jump to start if last calculation was not zero
LD HL, 16509	address of 1st byte of the program area
LD (30060), HL	transfer to store for next USR call
RET	return to basic
30060/1	next byte containing a line number
30062/3	next line number to use
30064/5	interval to use

The mnemonics are not much use to you unless you have a machine code assembler but the following list of decimal numbers can be poked into a REM statement and SAVED onto tape.

237	75	110	117	120	237
91	108	117	18	19	121
18	96	105	237	75	112
117	9	34	110	117	19
26	111	19	26	103	1
4	0	9	237	75	108
117	9	34	108	117	237
75	12	64	237	66	194
48	117	33	125	64	34
108	117	201			

.Set up the REM statement at line 1 and key in two lines of dots.

```
1   REM .....
   .....
   .....
```

The machine code will be poked into this area temporarily. There will be a few dots left over at the end. Key in this program to do the loading.

```
2   FOR J = 16514 TO 16571
3   INPUT A
4   POKE J, A
5   NEXT J
```

The ZX81 will request 57 numbers when you RUN this program and it is easy to get them wrong, so it is wise to check that the program is correct before using the routine. Change the load program as follows:

```
3   delete
4   PRINT PEEK J,
```

and then RUN it again and check the results against the list of numbers shown above. When all is well, delete lines 2 to 5 and then add these lines to the REM statement. First there is a sub-routine which converts a number stored in A into a two byte number suitable for the ZX81 which is stored in two variables; B and C.

most significant byte

```
2 LET C=INT (A/256)
```

least significant byte

```
3 LET B=A-C*256
4 RETURN
```

This next part of the program sets the top of the memory area that the ZX81 bothers about to byte number 30000.

```
9 REM RAM TOP AT 29999
10 POKE 16388,47
20 POKE 16389,117
```

The machine code will be loaded from byte 30000 onward and this next section of the program reads the characters in the REM statement and then POKES them into their correct places.

```
29 REM LOAD MACHINE CODE
30 FOR J=1 TO 57
40 POKE 29999+J,PEEK (16513+J)
50 NEXT J
```

Next the routine has to be primed with the correct information and the following lines accept decimal numbers and then call the sub-routine at line 2 to convert them into a suitable form for POKeing into the bytes allocated for data storage. The first byte in the program area has to be the first place that the ZX81 looks for its first line number so the address of this byte is always fed into memory stores 30060/1.

```
59 REM SET UP ROUTINE
60 PRINT "KEY FIRST LINE NUMBE
R"
70 INPUT A
```

for two byte format

```
80 GOSUB 2
```

least significant byte

```
90 POKE 30062,B
```

most significant byte

```
100 POKE 30063,C
110 PRINT A,,"KEY GAP BETWEEN L
INES"
120 INPUT A
130 GOSUB 2
140 POKE 30064,B
150 POKE 30065,C
159 REM FIRST BYTE = 16503
160 POKE 30060,125
170 POKE 30061,64
```

Once the program has been used it is of no further use so it can be deleted by NEW. This is the reason for re-setting the RAMTOP variable earlier. The computer only clears its memory as far as RAMTOP when NEW is used. Your program will now be in place even if it is invisible and can be brought into play by the command: RANDUSR 30000 followed by NEWLINE. Here are the last few lines which save the package onto tape and then start it working at the right place when it is loaded into the computer.

to clear the machine ready for the first job

```
180 NEW
190 LET A$="RENUMBER MC"
200 SAVE A$
210 GOTO 10
```

To save the program, set up the recorder and then key GO TO 190.

The Spectrum version of the program differs slightly because of the different layout of the machine's memory. It is impossible to LIST the machine code because there is at least one character which gives an "invalid colour" error code. As with the ZX81 program, the decimal code is poked into the REM statement at line 1 but the address will have to be found by peaking the system variables again. Key in the REM and about sixty dots and then use the following program to load the numbers into position:

```

10 LET a = PEEK 23635 + 256 * PEEK 23636
20 FOR j = a + 5 TO a + 61
30 INPUT b
40 PRINT j , b
50 POKE j , b
60 NEXT j

```

Delete these lines, all except line 1, and then add the following BASIC program. When it is complete key GO TO 70 and then record the routine on tape. The program will run itself and then self-destruct when you load it from the tape. From then until you switch off any program can be renumbered by keying RANDOMISE USR 30000. If the numbers have to start from ten again then key the following direct commands:

```

POKE 30062, 10
POKE 30063, 0

```

Here are the decimal numbers to be poked into the REM statement at line 1 and the BASIC program to load it into place above RAMTOP:

237	75	110	117	120
237	91	108	117	18
19	121	18	96	105
237	75	112	117	9
34	110	117	19	26
111	19	26	103	1
4	0	9	237	75
108	117	9	34	108
117	237	75	12	64
237	66	194	48	117
33	125	64	34	108
117	201			

```

2 GO TO 20
10 LET c=INT (a/256): LET b=a-
c*256: RETURN
20 POKE 23730,47: POKE 23731,1
17
30 LET a=PEEK 23635+256*PEEK 2
3636: FOR j=0 TO 56: POKE 30000+
j,PEEK (a+5+j): NEXT j

```

```

34 GO SUB 10: POKE 30050,b: PO
KE 30051,c: POKE 30051,b: POKE 3
0052,c
40 INPUT "Key the first line n
umber ";a: GO SUB 10: POKE 30052
,b: POKE 30053,c
50 INPUT "key the gap between
the lines ";a: GO SUB 10: POKE 3
0054,b: POKE 30055,c
60 NEW
70 SAVE "renumber3" LINE 1

```

Its about time that Spectrum users came first so this time it's they who get to play the game, and quite a good one as well. This time we are going to look after the GO TO and GO SUB addresses as well as the line numbers. ZX81 users will find a version of this more sophisticated routine at the end of the chapter. There are still a couple of things to watch out for though. All the GO TO and GO SUB addresses must have four digits e.g. GO TO 0100, and the routine won't be able to cope with calculated addresses such as GO SUB A * 1000. A section in the routine recognises these problems for you and stops the run giving you the line number at which the mistake is to be found. Just as an added bonus there is a line at the end which lets you know how much memory is left to use.

The routine works by building up a string of all the characters representing the address of the jump. These are built up in order in A\$, altered as necessary and then fed back into the places from which they came. The start of the routine is full of the sub-routines and these have to be by-passed by line 9900. The routine at 9901 finds the start of the program in memory. 9902 turns a normal number in "b" into a two byte number suitable for the computer in "c" and "d", "c" holds the most significant byte. The 9903 routine makes sure that a "number" character is in place five bytes after the GO TO or GO SUB character. The "number" character has a code of 14. If one of these is not where it should be, then the machine sets "a" to -1 as a signal. The routine at 9904 adds the nine characters which make up the address of the jump, to A\$. 9905 prints the error message and stops the run and the last routine at 9906 works out the line number as described earlier on page 130.

by-pass the sub-routines

```
9900 GO TO 9910
```

find start of program

```
9901 LET a=PEEK 23635+256*PEEK 2
3636: RETURN
```

makes 2 byte numbers

```
9902 LET c=INT (b/256): LET d=b-
256*c: RETURN
```

checks for "number" characters.

```
9903 IF PEEK (J+5) <>14 THEN LET  
a=-1: RETURN
```

adds address of jump to A\$

```
9904 FOR k=1 TO 9: LET a$=a$+CHR  
$ PEEK (J+k): NEXT k: RETURN
```

prints error message

```
9905 PRINT "all GO TO & GO SUBs  
must be 4 digits long. There's  
a problem in line ";line: STOP
```

works out line number

```
9906 LET line=256*PEEK a+PEEK (a  
+1): RETURN
```

The first job is to arrange the first of the new line numbers and the gap between the lines. The start "st" is stored in "st1" because a copy of it will be needed after it has been altered. A\$ is set to the empty string. Line 9915 finds the first line number and checks that it is not part of the re-numbering routine itself. The program loops round many times and eventually will reach the routine at 9900 and at this point the computer is sent on to the next section because re-numbering the re-number routine would be disastrous.

find start of program
set up variables

```
9910 GO SUB 9901: INPUT "startin  
g at new line number? ";st: INPU  
T "With a gap of? ";ga: LET a$="":  
LET st1=st
```

find line number
move on if end of program

```
9915 GO SUB 9906: IF line>9898 T  
HEN GO TO 9940
```

GO TO has a code of 236 and GO SUB 237 on the Spectrum and the ZX81. This line checks each line of the program in turn for GO TO or GO SUB commands. When these are found the computer is sent to the sub-routine at 9903 for checking. If all is well the information is added to A\$, otherwise the machine jumps to the error message line at 9905 and stops.

searches the text of each line
adds information to A\$ if OK

```
9925 FOR j=a+4 TO a+2+PEEK (a+2)  
+256*PEEK (a+3): IF PEEK j=236 O  
R PEEK j=237 THEN GO SUB 9903: I  
F a=-1 THEN GO TO 9905  
9930 NEXT j
```

The line number and length are held in the same way in both the Spectrum and the ZX81. Line 9935 sets the variable "a" to the start of the next line and then jumps back to 9915 to search for more jump commands.

resets "a" to start of next line

```
9935 LET a=a+4+PEEK (a+2)+256*PEEK (a+3): GO TO 9915
```

When all the lines in front of the re-number routine have been searched the machine is sent to line 9940 where the start of the program is found again, and the value of the first line number is worked out. Again, this section loops and will eventually produce a line number in the re-number routine, at which point the computer is sent to the next section.

The computer now steps through the whole of A\$, once for each line, looking for jump addresses which match the line number held in the variable "line". If no match is found then the section which adjusts the addresses is by-passed.

start of program

```
9940 GO SUB 9901
```

line number

```
9945 GO SUB 9906: IF (line>9998) THEN GO TO 9970
```

check A\$ for address match

by-pass if not found

```
9950 FOR j=1 TO LEN a$ STEP 9: IF (line<>VAL a$(j TO j+3)) THEN GO TO 9965
```

B\$ will hold the new line number so the starting number in "st" is fed into B\$. "st" will be updated later on, as the routine loops. The address must have four characters and so zeros are added to the front of B\$ if it is too short. The characters are fed into the correct positions in A\$ and then "st" is turned into a two byte number by a call to the routine at 9902. This two byte value is fed into A\$ by means of the CHR\$ command.

new line number in B\$

```
9952 LET b$=STR$ st
```

ensure B\$ is 4 characters. Change A\$ with new line number and value

```
9955 IF LEN b$<>4 THEN LET b$="0"+b$: GO TO 9955  
9960 LET a$(j TO j+3)=b$: LET b=
```

```

st: GO SUB 9902: LET a$(j+7)=CHR
$d: LET a$(j+8)=CHR$c

```

The next line number to be used is generated by adding "ga" to "st" and then the first byte of the next line is found by adding the line length to the start of the old line position in memory. The program loops until the first line of the re-number routine is reached.

next line number to use

```

9965 NEXT j: LET st=st+ga: LET a
=a+4+PEEK (a+2)+256*PEEK (a+3):
GO TO 9945

```

In this last section the whole of A\$ is fed back into the program, nine bytes at a time, and the line numbers are changed. The first line number is retrieved from "st1", the start of the program is identified and the line number is worked out by the routine at 9906. This time, if the line number is above 9898 the program stops running with a report on the amount of memory left.

find start of program
re-set 1st line no.
find old line number
off-load a\$ into lines
remove first nine bytes of a\$

```

9970 GO SUB 9901: LET st=st1
9975 GO SUB 9906: IF line>9898 T
HEN GO TO 9990
9980 FOR j=a+4 TO a+4+PEEK (a+2)
+256*PEEK (a+3): IF PEEK j=236 O
R PEEK j=237 THEN FOR k=1 TO 9:
POKE j+k, CODE a$(k): NEXT k: LET
a#=a$(10 TO )

```

The new line number is turned into a two byte value and then fed into the line in place of the old version of the number. The memory location of the start of the next line is found and the computer is sent back. The routine finishes with a report that re-numbering is complete and information about the amount of memory remaining to be used. The routine takes some memory itself so 1476 bytes is added to allow for the space freed when the routine is deleted.

calculate the 2 byte value of the next line number
replace old number with new
set new line number and find start of next line loop
report on memory left

```

9965 NEXT j: LET b=st: GO SUB 99
02: POKE a,c: POKE a+1,d: LET st
=st+ga: LET a=a+4+PEEK (a+2)+256

```



```

*PEEK (a+3): GO TO 9975
9990 PRINT "Renumbered"... "Bytes
remaining=";1351+(PEEK 23732+256
*PEEK 23733)-(PEEK 23641+256*PEE
K 23642): PAUSE 500: CLS : LIST

```

The routine is protected by a STOP command and should not be called into play by accident. To use it key GO TO 9900, reply to the requests and then be prepared for a longer wait than you might have experienced with the simpler programs. The example program "GOTCHER" was re-numbered in fifty odd seconds.

ZX81 users will not be able to run the GOTCHER program as it stands, and it probably will not be worth translating as it would have to run in slow mode and would be too easy at the low speed. Here are the listings of the game before and after re-numbering:

GOTCHER"as written"

```

10 CLS : LET bs=0
20 LET a=10: LET b=15: LET y=1
9: LET z=30: LET s=0
30 FOR j=1 TO 640: PRINT ".":
NEXT j
40 FOR j=0 TO 20 STEP 2: FOR k
=0 TO 31: PRINT AT j,k:"█": NEX
T k: NEXT j
50 FOR j=0 TO 20: PRINT AT j,0
;"█": AT j,31:"█": NEXT j
60 FOR j=2 TO 19 STEP 2: FOR k
=1 TO (RND*5)+1: PRINT AT j,(RND
*29)+1:".": NEXT k: NEXT j
70 LET b$="█"
80 PRINT AT a,b;"0": AT y,z;b$
90 PRINT AT y,z:".": IF a<y TH
EN LET b$=SCREEN$(y-1,z): IF b$
="." OR b$="0" OR b$="-" THEN LE
T y=y-1
100 IF a>y THEN LET b$=SCREEN$(
y+1,z): IF b$="." OR b$="0" OR
b$="-" THEN LET y=y+1
110 IF b<z THEN LET b$=SCREEN$(
y,z-1): IF b$="." OR b$="0" OR
b$="-" THEN LET z=z-1
120 IF b>z THEN LET b$=SCREEN$(
y,z+1): IF b$="." OR b$="0" OR
b$="-" THEN LET z=z+1
130 GO SUB 0140: GO SUB 0170: G
O SUB 0170: GO TO 0090
140 PRINT AT y,z;"█": PAUSE 10:
IF y=a AND z=b THEN PRINT AT 10
12;"GOTCHER": IF s>bs THEN LE
T bs=s
150 IF y=a AND z=b THEN PRINT A
T 21,15;"Record ";bs: FOR j=1 TO
500: NEXT j: CLS : GO TO 0020
160 RETURN
170 PRINT AT a,b;"_": IF INKEY$
=" " THEN GO TO 0220

```

```

180 IF INKEY$="5" THEN LET b$=5
CREEN$ (a,b-1): IF b$="." OR b$=
" " THEN LET b=b-1: IF b$="." TH
EN LET s=s+1
190 IF INKEY$="8" THEN LET b$=5
CREEN$ (a,b+1): IF b$="." OR b$=
" " THEN LET b=b+1: IF b$="." TH
EN LET s=s+1
200 IF INKEY$="7" THEN LET b$=5
CREEN$ (a-1,b): IF b$="." OR b$=
" " THEN LET a=a-1: IF b$="." TH
EN LET s=s+1
210 IF INKEY$="6" THEN LET b$=5
CREEN$ (a+1,b): IF b$="." OR b$=
" " THEN LET a=a+1: IF b$="." TH
EN LET s=s+1
220 PRINT AT 21,0;"Score ";s: P
RINT AT a,b;"0": RETURN

```

GOTCHER after renumbering

```

1 CLS : LET bs=0
2 LET a=10: LET b=15: LET y=1
9: LET z=30: LET s=0
3 FOR j=1 TO 640: PRINT ".":
NEXT j
4 FOR j=0 TO 20 STEP 2: FOR k
=0 TO 31: PRINT AT j,k;"■": NEX
T k: NEXT j
5 FOR j=0 TO 20: PRINT AT j,0
;"■": AT j,31;"■": NEXT j
6 FOR j=2 TO 19 STEP 2: FOR k
=1 TO (RND*5)+1: PRINT AT j,(RND
*29)+1;"." : NEXT k: NEXT j
7 LET b$="■"
8 PRINT AT a,b;"0": AT y,z;b$
9 PRINT AT y,z;"." : IF a<y TH
EN LET b$=SCREEN$ (y-1,z): IF b$
="." OR b$="0" OR b$="_" THEN LE
T y=y-1
10 IF a>y THEN LET b$=SCREEN$
(y+1,z): IF b$="." OR b$="0" OR
b$="_" THEN LET y=y+1
11 IF b<z THEN LET b$=SCREEN$
(y,z-1): IF b$="." OR b$="0" OR
b$="_" THEN LET z=z-1
12 IF b>z THEN LET b$=SCREEN$
(y,z+1): IF b$="." OR b$="0" OR
b$="_" THEN LET z=z+1
13 GO SUB 0014: GO SUB 0017: G
O SUB 0017: GO TO 0009
14 PRINT AT y,z;"■": BEEP .1,-
30: IF y=a AND z=b THEN PRINT AT
10,12;"SCOREEE": IF s>bs THEN
LET bs=s
15 IF y=a AND z=b THEN PRINT A
T 21,15;"Record ";bs: FOR j=1 TO
500: NEXT j: CLS : GO TO 0002
16 RETURN
17 PRINT AT a,b;"_": IF INKEY$
=" " THEN GO TO 0022

```

```

18 IF INKEY$="5" THEN LET b$=3
GREEN$ (a,b-1): IF b$="." OR b$=
" THEN LET b=b-1: IF b$="." TH
EN LET s=s+1
19 IF INKEY$="8" THEN LET b$=3
GREEN$ (a,b+1): IF b$="." OR b$=
" THEN LET b=b+1: IF b$="." TH
EN LET s=s+1
20 IF INKEY$="7" THEN LET b$=3
GREEN$ (a-1,b): IF b$="." OR b$=
" THEN LET a=a-1: IF b$="." TH
EN LET s=s+1
21 IF INKEY$="6" THEN LET b$=3
GREEN$ (a+1,b): IF b$="." OR b$=
" THEN LET a=a+1: IF b$="." TH
EN LET s=s+1
22 PRINT AT 21,0;"Score ";s: P
RINT AT a,b;"0": RETURN

```

The ZX81 version of the sophisticated re-numbering routine is very similar in its design. There are a few differences in the addresses and codes, but the main difference is in the way in which the earlier machine held its jump addresses. The Spectrum is simplicity itself in comparison. It takes a long time to work out the pattern to the way the addresses are coded and the lines 9390 to 9410 show how complex the storage is.

ZX81 FULL RENUMBER

```

9000 STOP
9010 GOTO 9100
9020 LET C=INT (B/256)
9030 LET D=B-256*C
9040 RETURN
9050 IF PEEK (J+5) <> 126 THEN LET
A=-1
9055 IF PEEK (J+5) <> 126 THEN RET
URN
9060 FOR K=1 TO 9
9063 LET A$=A$+CHR$ PEEK (J+K)
9066 NEXT K
9068 RETURN
9070 PRINT "ALL GOTO AND GOSUBS
MUST BE FOURDIGITS LONG. THERES
A PROBLEM IN LINE ";LINE
9075 SLOW
9080 STOP
9090 LET LINE=256*PEEK A+PEEK (A
+1)
9095 RETURN
9100 LET A=16509
9110 LET A$=""
9120 PRINT "RENUMBERING",,"STATI
NG AT NEW LINE NUMBER?",
9130 INPUT ST
9140 PRINT ST,,"WITH A GAP OF?",
9150 INPUT GA
9160 PRINT GA
9170 FAST
9180 LET ST1=ST
9200 GOSUB 9090
9210 IF LINE >= 9999 THEN GOTO 929
0
9230 FOR J=A+4 TO A+2+PEEK (A+2)
+256*PEEK (A+3)

```

```

9240 IF PEEK J=236 OR PEEK J=237
    THEN GOSUB 9050
9250 IF A=-1 THEN GOTO 9070
9260 NEXT J
9270 LET A=A+4+PEEK (A+2)+256*PE
EK (A+3)
9280 GOTO 9200
9290 LET A=16509
9300 GOSUB 9090
9310 IF LINE>=8999 THEN GOTO 946
0
9320 FOR J=1 TO LEN A$ STEP 9
9330 IF LINE<>VAL A$(J TO J+3) T
HEN GOTO 9420
9340 LET B$=STR$ ST
9350 IF LEN B$(<>)4 THEN LET B$=""0
"+B$
9360 IF LEN B$(<>)4 THEN GOTO 9350
9370 LET A$(J TO J+3)=B$
9380 LET B=128+INT (LN ST/LN 2+1
)
9385 LET A$(J+5)=CHR$ B
9390 LET C=ST#65536/(2*(A-128))
9400 LET D=C-256*INT (C/256)
9405 LET A$(J+7)=CHR$ D
9410 LET C=INT (C/256)-128
9415 LET A$(J+6)=CHR$ C
9420 NEXT J
9430 LET ST=ST+GA
9440 LET A=A+4+PEEK (A+2)+256*PE
EK (A+3)
9450 GOTO 9300
9460 LET A=16509
9470 LET ST=ST1
9480 GOSUB 9090
9490 IF LINE>=8999 THEN GOTO 999
0
9500 FOR J=A+4 TO A+2+PEEK (A+2)
+256*PEEK (A+3)
9510 IF PEEK J<>236 AND PEEK J<>
237 THEN GOTO 9560
9520 FOR K=1 TO 9
9530 POKE J+K, CODE A$(K)
9540 NEXT K
9550 LET A$=A$(10 TO )
9560 NEXT J
9570 LET B=ST
9580 GOSUB 9020
9590 POKE A,C
9600 POKE A+1,D
9610 LET ST=ST+GA
9620 LET A=A+4+PEEK (A+2)+256*PE
EK (A+3)
9630 GOTO 9480
9990 SLOW
9999 PRINT ,,"RENUMBERED",,"BYTE
S REMAINING=";1009+(PEEK 16388+2
56*PEEK 16389)-(PEEK 16404+256*P
EEK 16405)

```

To play GOTCHER, use the 5, 6, 7 and 8 keys to run away from the hunter in the maze. The maze is generated randomly for each new game and is filled with energy giving sandwiches! As you run you accumulate points for eating the sandwiches which seem to have fallen from a hole in the bottomless sack which the hunter always carries. The weight of such a sack slows him down on the straight but he is very fast around the corners and seems to think very quickly although his ideas are a little limited. It is easy to trap the hunter in a cul-de-sac but then he drops no more sandwiches. Eventually you will have to tease him out of the trap and make him chase you to drop some more food. A score of 200 is fair for a beginner but 1000 points is a reasonable target to aim for.

If you have a colour set then it is an easy task to brighten up the maze. You might wish to use the sound as well. If you replace the delay in line 140 of the original program with " BEEP .1 , 30 " you will get a sinister effect as the hunter follows you. He gets rather excited whenever you stop and this comes across in the sound. You could try some suitable music to play when you are eventually caught. The Mahler on page 136 of the Spectrum hand book seems fitting.

ZX81 users have been waiting patiently for their merge program and here it is at last. It is in machine code and sits above the RAMTOP. The merge program is in two halves but the two together only occupy sixty-four bytes. The first part moves programs above RAMTOP and allows the lower part of the memory to be cleared by the NEW command, ready for the next part of the program. When a new program is in place the second part of the machine code can be called into play to dump the earlier lines on top of the present program.

The program has to be loaded into a REM statement which has at least sixty-four dots in it. Set up a temporary program as shown in the listing. Add the other six lines and then run the program.

```
10 REM .....  
.....  
20 FOR J=16514 TO 16577  
30 INPUT A  
40 SCROLL  
50 PRINT A  
60 POKE J,A  
70 NEXT J
```

The machine will wait sixty-four times while you enter the machine code in decimal form. Here are the numbers to be entered:

DECIMAL CODE VERSION OF M/C

42	12	64	17	125
64	167	237	82	64
93	33	117	127	167
237	82	34	4	64
42	12	64	17	125
64	167	237	82	34
168	127	237	91	4
64	68	77	33	125
64	237	176	201	42
12	64	229	43	1
1	0	197	205	158
0	193	209	42	4
64	237	176	201	

When you list the program after the run the screen should look like this:

```

10 REM E=RNDRND) ?RND GOSUB ???5?
GOSUB ?6 RND E=RNDRND) ?RND GOSUB
?6 GOSUB ? RND ???5?RND GOSUB
TAN E=RNDRND FAST F VAL LN AT S
GN E RND GOSUB TAN I
20 FOR J=16514 TO 16577
30 INPUT A
40 SCROLL
50 PRINT A
60 POKE J,A
70 NEXT J

```

Delete all the lines from 20 on and then add the necessary lines to give the following listing:

MERGE 81

```

5 POKE 16388,220
6 POKE 16389,95
10 REM E=RNDRND) ?RND GOSUB ???5?
GOSUB ?6 RND E=RNDRND) ?RND GOSUB
?6 GOSUB ? RND ???5?RND GOSUB
TAN E=RNDRND FAST F VAL LN AT S
GN E RND GOSUB TAN I
20 FOR J=0 TO 64
30 POKE (32630+J),PEEK (16567+
J)
40 NEXT J
50 NEW

```

RUN the program this time and it should disappear! The code is safely stored high up in the memory and the rest of the machine is ready for building programs from the units that you have on tape. When the units have been merged together, the rest of the program may be typed in on top in the normal way. There is one point to bear in mind. The lines stored above RAMTOP will have to have higher line numbers than the lines onto which they are to be dumped. It is possible to re-number the programs after assembly but this will mean recording them, loading the RENUMBER program and then using it to tidy up the programs stored on tape. It is much simpler to build the segments up into a program from the high line numbers down.

The machine code is called into play by the following statements:

To store a program above RAMTOP	RAND USR 32630 NEW
To merge a stored program with lines in the program area	IF USR 32674 = 0 THEN STOP

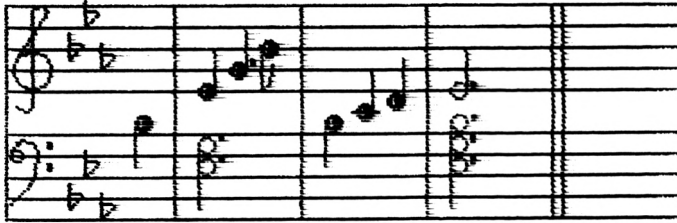
Machine Code References

Assembly Language Assembled - for the Sinclair ZX81, by Tony Woods. Published by The Macmillan Press, 1982.

Mastering Machine Code on your ZX81 or ZX80 by Tony Baker. Published by Database Consultancy.

Programming the Z80 by Rodney Zaks. Published by Sybex.

11 *Music*



Emma, my daughter, was having difficulty in remembering the names of the notes for her piano lessons. To help her out I wrote a short program for the ZX81 and this soon overcame the difficulty. Working on the computer added a little interest to a dull job. The ZX81 program is given a little further on but the bulk of this chapter is about the development of the original idea. The extra facilities available on the Spectrum made it possible to produce realistic music on the screen, even those excessively curly clefs, but most important of all there was sound. With sound came the possibility of aural work. Music teachers should be allowed to concentrate on the creative side of music and leave the drill and practice of aural testing to the computer.

Music is a very complex system and, though it has a mathematical basis, it was simpler to base the programs which follow on data held in the computer rather than on mathematical formulae. The programs which follow make extensive use of the Spectrum's DATA, READ and RESTORE statements. The most interesting feature of the programs, however, is their basic architecture. The idea of a library of sub-routines mentioned earlier has been extended so that the programs become merely a means of driving the computer through the available services. The "driver" programs are generally much shorter than the collection of routines which they drive. This approach means that the user is not confined to the programs given in this chapter. These can be thought of as examples of ways in which the library can be used. The hope is that readers will design driver programs of their own which will make the sub-routines dance to any tune.

The whole package will fit into the 48K Spectrum but is too large for the smaller machine. The individual programs have been written on the 16K computer to make sure that they all fit.

Indeed, many can be merged together in the 16K Spectrum but the last program, Rhythm, is larger and needs the machine to itself.

Here is the ZX81 program. It has a limited range of responses, nothing at all for a miss and "OK" for a hit. In spite of this it does its job and covers both bass and treble clefs.

BASS AND TREBLE

```

1 LET B$=" -----"
2 LET C$=" ----"
3 LET D$="BASS"
4 IF RND<.5 THEN LET D$="TREB
LE "
5 PRINT D$
6 PRINT C$,,C$,,C$,,B$,,B$,,
.B$,,B$,,B$,,C$,,C$
7 LET A=INT (RND*8)
8 LET B=A
9 LET C=7
10 IF RND<.5 THEN LET B=B+C
11 PRINT AT B,C;"#.#";TAB C;"# #
";TAB C;"#.#"
12 INPUT A$
13 IF D$<>"BASS" THEN LET A=A-
5
14 IF A<5 THEN LET A=A+C
15 IF A<5 THEN LET A=A+C
16 IF A>11 THEN LET A=A-C
17 IF CODE A$=C*C-A THEN PRINT
"OK"
18 FOR J=C-C TO C+C
19 NEXT J
20 CLS
21 RUN

```

From that humble beginning sprang the following mammoth collection of data and routines. The first to be covered is a section of service routines which do some very mundane jobs. The routine at line 2 organises the printing of all of the messages. By keeping the printing on line fifteen, the line of spaces automatically removes the old message. The last PRINT AT statement re-sets the print position for the next message. The pause seems just long enough for most purposes. The next routine at line 3 checks that z\$ contains a single digit between "1" and whatever is in e\$ at the time. If Z\$ is likely to cause trouble it is simply emptied. The driver program looks for an empty string before carrying on with its task. If one is found then the user is asked for another input. The routine at line 5 does a similar job for any collection of characters between the contents of d\$ and e\$. Z\$ is emptied if the input does not fall between the two limits. These last two routines are likely to be familiar but the next one is new. It turns lower case input in Z\$ into upper case. This is important when checking one string against another; to the Spectrum "a major" does not equal "A major".

```

1 GO TO 900
2 PAUSE 50: PRINT AT 15,0;"":
PRINT AT 15,0;: RETURN
3 IF Z$<"1" OR Z$>e$ OR LEN Z
$<>1 THEN LET Z$=""
4 RETURN
5 IF Z$<d$ OR Z$>e$ OR (LEN Z
$<>1 AND LEN Z$<>2) THEN LET Z$=""
6 RETURN
7 FOR J=1 TO LEN Z$: IF CODE
Z$(J)>90 THEN LET Z$(J)=CHR$(CO
DE Z$(J)-32)
8 NEXT J: RETURN

```

It was decided to DRAW the clefs and the stave rather than use the alternative graphics capability for two reasons. Firstly the code needed to produce the alternative graphics is very long and difficult to plan and secondly it is difficult to draw over characters already on the screen without leaving little holes in the display. Drawing the display is a little slower but produces more pleasing results.

This next routine draws the lines of the stave and then places the clefs on the lines. The lines are a character cell apart (eight picture points), hence the "STEP -8" in line 10. The top five lines are separated from the lower by a blank line for middle C so line 11 by-passes the drawing instruction after the first five lines. Line 14 draws the first bar line to tie the treble and bass staves together. The last two lines in the routine would take a lot of space to explain and this would be wasted because it is so much easier to appreciate by watching the lines in action. They draw the two clefs in place.

```

9 REM DRAW STAVE
10 FOR J=150 TO 70 STEP -8
11 IF J=110 THEN GO TO 13
12 PLOT 0,J: DRAW 255,0
13 NEXT J
14 PLOT 0,150: DRAW 0,-80
15 CIRCLE 10,125,7: PLOT 4,130
: DRAW 12,18: DRAW -6,0,2: DRAW
0,-36: DRAW -6,0,-2
16 CIRCLE 4,94,2: PLOT 2,96: D
RAW 10,0,-2,5: DRAW -10,-20,-1:
CIRCLE 16,98,1: CIRCLE 16,90,1
17 RETURN

```

Now follow two collections of lines which are not sub-routines but collections of data. Lines 20 to 22 contain the information about the number of semi-tones to be found between the notes of the three types of scales: major, harmonic and melodic minors. By READING the data and then adding it to the value of the first note in the scale, the machine will be able to go smoothly up, and then down, any scale that may be chosen.

```

19 REM SCALES DATA
20 DATA 2,2,1,2,2,2,1,-1,-2,-2
,-2,-1,-2,-2
21 DATA 2,1,2,2,2,2,1,-2,-2,-1

```

```

-2, -2, -1, -2, " (Melodic)"
20 DATA 2,1,2,2,1,3,1,-1,-3,-1
-2, -2, -1, -2, " (Harmonic)"

```

Lines 30 to 42 also contain data, this time on the number of semitones between the notes in an interval. The string data gives the name of the interval and the number data gives the number of semitones. They are arranged in this order so that they are in order of difficulty. There are three difficulty levels in the program and only those musicians working on level three will be expected to cope with diminished fifths or minor sixths.

```

29 REM INTERVAL DATA
30 DATA "unison",0
31 DATA "major 2nd",2
32 DATA "major 3rd",4
33 DATA "perfect 4th",5
34 DATA "perfect 5th",7
35 DATA "major 6th",9
36 DATA "major 7th",11
37 DATA "octave",12
38 DATA "minor 3rd",3
39 DATA "minor 7th",10
40 DATA "minor 2nd",1
41 DATA "augmented 4th or dimi
ished 5th",6
42 DATA "augmented 5th or mino
r 6th",8

```

Now it's time for a little more of the library which does some of the work. Line 50 draws a sharp and line 51 draws a flat. The signs are drawn in place, "a" positions across the screen and "b" positions up. These two co-ordinates are set automatically by other routines and generally the programmer need not worry about them. This illustrates the strength of this type of design where one sub-routine makes use of others in the collection.

```

49 REM DRAW SHARPS AND FLATS
50 PLOT a,b+5: DRAW 0,-10: PLO
T a+5,b+6: DRAW 0,-10: PLOT a-2,
b-3: DRAW 9,2: PLOT a-2,b+2: DRA
U 9,2: RETURN
51 PLOT a,b+6: DRAW 0,-10: PLO
T a,b+1: DRAW 5,0,-2: DRAW -5,-5
: RETURN

```

The Spectrum has to have a means of knowing how many sharps or flats to draw and the positions that they should occupy. The data between lines 60 and 77 allows the key signatures to be drawn. Each sign has to be drawn in the treble and the bass clef and these should be drawn in a vertical line. The variable "a" has to be read once and "b" needs to be found for the treble sign and then for the bass sign. The first piece of data is the "a" co-ordinate and the second two pieces are the two "b" co-ordinates.

```

59 REM KEY SIGNATURES
60 REM STAFF POS
61 DATA 24,150,94
62 DATA 30,138,82
63 DATA 36,155,96
64 DATA 42,142,86
65 DATA 48,130,102

```

```

66 DATA 54,146,90
67 DATA 60,134,78
70 REM STEMS
71 DATA 24,134,78
72 DATA 30,146,90
73 DATA 36,130,74
74 DATA 42,142,86
75 DATA 48,126,70
76 DATA 54,138,82
77 DATA 60,122,66

```

Without the ability to draw the notes there would not be much point to the program. The next four routines will place notes on the stave, above it and below it as well, and with their tails and stems pointing in the correct directions. The co-ordinates "a" and "b" are used again to direct the note to the correct position on the screen. When notes are drawn above or below the stave, extra lines called ledger lines are needed and these are provided by a routine at line 170. The routine at 80 draws the note in place and then hands over to line 170 to draw any necessary ledger lines. Line 81 fills in the circle drawn by 80 while 82 and 83 decide between them which way the stem should go, and then draw it and if necessary, 85 adds a tail for a quaver.

```

79 REM DRAW NOTES
80 LET x=0: CIRCLE a,b,3: GO T
O 170
81 CIRCLE a,b,2: CIRCLE a,b,1:
RETURN : REM fills note
82 IF (b>86 AND b<110) OR b>13
O THEN PLOT a-3,b: DRAW 0,-16: R
ETURN : REM stem up
83 PLOT a+3,b: DRAW 0,16: RETU
RN : REM stem down
85 IF (b>86 AND b<110) OR b>13
O THEN DRAW 4,10,1: REM draws ta
il
86 DRAW 4,-10,-1: RETURN

```

Later on, at line 300, there is a great deal of data about the scales. The sub-routine at line 90 picks a scale from this collection at random, but with the degree of difficulty in mind. At level 1 only a limited collection of scales are presented, while level 3 musicians will be expected to cope with all the scales, major and minor up to seven sharps and flats. The first line of this routine is rather complex. The selection of a scale is straightforward enough but it is difficult to see exactly what the variables "e" and "f" are for. "e" holds the number of sharps or flats that have to be drawn to make the key signature. The variable "f" is best described as "related" to the type of key, major or minor, with flats or sharps. More of this later.

The next line of data to be read is set by the RESTORE command and then the details of the notes in the scale are fed into a\$. The pitch of the first note in the scale is fed into "c" and the vertical co-ordinate is fed into "b". To save a little memory the octave note is not shown in the data statements as it is simply a duplication of the first note. LET a\$=a\$a\$(TO 2) makes a\$ up to the full eight notes. Lines 91 and 92 push the start of the scale

down the stave an octave or two to make sure that the questions set later on examine all parts of the range.

```

90 LET d=299+(INT (RND*10+d1))+
1): LET e=7-INT ((329-d)/4): LET
f=INT (4*((d-300)/4-INT ((d-300
)/4)+.001)): RESTORE d: READ a$,
c, b: LET a$=a$+a$( TO 2)
91 IF RND<.66 THEN LET c=c-12:
LET b=b-28
92 IF RND<.33 THEN LET c=c-12:
LET b=b-28

```

In the scale DATA lines the notes are given and then followed by a space (not sharp or flat), an F (flat), an S (sharp), an N (natural) or an X (double sharp). The following line describes the scale of C minor:

```
DATA "C D EFF G AFBN",0,110
```

The name of the scale is held in b\$ and this starts off as the first character in a\$. The second character in a\$ is used to decide on the next part of the name, "C flat", "C sharp" or just plain "C". The next decision to make is the major/minor choice. The value of "f" is used to help here but the scales of C major and A minor are special cases which do not fit the mathematical pattern of the rest of the data lines. The variable "g" is used as a flag to indicate "major" or "minor".

```

99 REM GENERATE SCALE NAME
100 LET b$=a$(1): IF a$(2)="S"
THEN LET b$=b$+" sharp"
101 IF d=300 THEN LET b$=b$+" #
ajor": LET g=0: RETURN
102 IF d=301 THEN LET b$=b$+" #
inor": LET g=1: RETURN
103 IF a$(2)="F" THEN LET b$=b$
+" flat"
104 IF f=3 OR f=1 THEN LET b$=b
$+" minor": LET g=1: RETURN
105 LET b$=b$+" major": LET g=0
: RETURN

```

Printing the key signature means RESTOREing to the correct DATA line and then calling the routines which draw sharps or flats. First of all though, the value of "b" has to be stored in "b1" as the value will be corrupted by the routines which are called. Again C major and A minor are a pair of special cases as they have no sharps or flats. If these are detected, that is if "d" equals 300 or 301, the machine returns control to the driver program. The value of "f" is used to determine if flats or sharps are to be drawn. RESTORE 60 gives access to the data on sharps while RESTORE 70 makes the machine look up how to plot flats on the screen. You will remember that the variable "e" holds the number of sharps or flats there are in the scale, and this value is used as the limit of a loop which steps through the data, going off to print a sharp or a flat each step (GO SUB 50 or 51). The last line of the routine adds a little to "a" to make the first note print a little way into the screen and then digs the old value of "b" out of "b1" before RETURNing.

```

109 REM PRINT THE KEY SIGNATURE
110 LET b1=b: IF d=300 OR d=301
THEN LET a=32: RETURN
111 IF f>=2 THEN RESTORE 60: FO
R J=1 TO e: READ a,b: GO SUB 50:
READ b: GO SUB 50: NEXT J
112 IF f<2 THEN RESTORE 70: FOR
J=1 TO e: READ a,b: GO SUB 51:
READ b: GO SUB 51: NEXT J
113 LET a=a+16: LET b=b1: RETUR
N

```

It's about time the computer was made to use its voice so the next part of the library plays the scale that was selected at line 90. To warn the user that something is about to happen, a message is printed and so the PRINT AT 15,0; routine is called. RESTORE 20 selects the major scale but if the "minor" flag, "g" is set to 1 the computer ignores 20 and starts looking at line 21 for its data. If the user has opted for difficulty level 3 then there is a 50/50 chance that the Spectrum will move on again, from the melodic to the harmonic minor scale. The data is fed into an array so that it will be available for future use and if the scale is minor, the type of minor will be fed into c\$. Next, the bit we have been working towards, making the Spectrum sing. Line 122 makes the Spectrum BEEP each note of the scale and then play the keynote a little longer than the rest. The length of the note "1" is set at the time that the difficulty level is chosen and those opting for the more difficult questions will have to make do with very short notes. Just before returning the machine calls the routine at line 2 again to keep the display tidy.

```

119 REM PLAY A SCALE
120 GO SUB 2: PRINT "Here is th
e scale": PAUSE 25: RESTORE 20:
IF g=1 THEN RESTORE 21: IF d1=3
AND AND>.5 THEN RESTORE 22
121 DIM z(14): FOR J=1 TO 14: R
EAD z(J): NEXT J: IF g=1 THEN RE
AD c$
122 FOR J=1 TO 14: BEEP L,c: L
ET c=c+z(J): NEXT J: BEEP L+3,c
123 PAUSE 25: GO SUB 2: RETURN

```

The arpeggio is played in a similar manner. The second note will be flattened if the scale is minor and so the minor flag "g" is used to decide the pitch. Again, GO SUB 2 is used to print and then remove a message.

```

129 REM PLAY AN ARPEGGIO
130 GO SUB 2: PRINT "Here is th
e arpeggio": PAUSE 25: BEEP L,c:
IF g=0 THEN BEEP L,c+4:
131 IF g=1 THEN BEEP L,c+3
132 BEEP L,c+7: BEEP L,c+12: BE
EP L,c+7: IF g=1 THEN BEEP L,c+3
133 IF g=0 THEN BEEP L,c+4
134 BEEP 3*L,c: PAUSE 25: GO SU
B 2: RETURN

```

There is scope for a great deal more variety in the messages printed by the next two routines but the type of message used is a matter of personal choice. Line 140 indicates a correct response, adds one to the score and the number of questions attempted and then returns. Line 150 indicates an incorrect reply and starts to give the correct answer. One is added to the number of questions but not to the score, the message is removed and the control passes back to the main program.

```

140 GO SUB 2: PRINT "THAT'S RIG
HT": GO SUB 2: LET s=s+1: LET tt
=tt+1: RETURN
150 GO SUB 2: PRINT "WRONG, IT
WAS: -": GO SUB 2: LET tt=tt+1: R
ETURN

```

If the program is to ask reasonable questions about a particular scale it cannot generate notes at random but will have to select its notes from those available in the scale. In line 160 of the next routine the pitch of the keynote is transferred into "h" and "h1" is set to a random note. If the difficulty level, "di", is three then the pitch is set randomly up or down an octave. You may remember that the intervals within a scale were fed into an array z(14) at line 121. These are now used to set "h" to the pitch of the second note in an interval within a scale. Line 163 uses the note number to adjust the vertical co-ordinate of the note "b".

Each note in the scale is drawn four picture points above the previous one and so $h1 * 4$ is added to "b" to give the right vertical co-ordinate. If the pitch of the second note, "h", is outside the normal octave range, then "b" will have to be adjusted by 28 up or down to make the note appear in the right part of the stave. Line 165 prevents the generation of notes which are too low or high. If these are detected the machine is sent back for another try.

```

160 LET h=c: LET h1=INT (RAND*8)
: IF di=3 AND RAND<.67 AND h1<>0
AND h1<>7 THEN LET h=h-12: IF AN
D<.5 THEN LET h=h+24
161 IF h1=0 THEN GO TO 163
162 FOR j=1 TO h1: LET h=h+z(j)
: NEXT j
163 LET b=b+h1*4: IF h<c THEN L
ET b=b-28
164 IF h>c+12 THEN LET b=b+28
165 IF b<50 OR b>170 THEN GO TO
160
166 RETURN

```

The code which draws the ledger lines would not fit neatly between lines 80 and 89 so it has been tucked in here where there is a little more room. 110 is the vertical co-ordinate for middle C, 158 and 166 give the upper two ledger lines while 62 and 54 give the lower two. Ledger lines are drawn whenever values of "b" indicate that they are needed. The RETURN to the main program is from here whenever the sub-routine at 80 is called.

```

170 IF b=110 THEN PLOT a-5,b: D
RAW 10,0:
171 IF b>=158 THEN PLOT a-5,158
: DRAW 10,0: IF b>=166 THEN PLOT
a-5,166: DRAW 10,0
172 IF b<=62 THEN PLOT a-5,62:
DRAW 10,0: IF b<=54 THEN PLOT a-
5,54: DRAW 10,0
173 RETURN

```

The block of data that follows is the biggest in the set. The data covers all the scales which the programs have to deal with. In each line is to be found a string, the notes, and two numbers. The first number is the pitch of the key note and the second is the vertical co-ordinate of the key note on the staff.

```

300 DATA "C D E F G A B ",0,110
301 DATA "A B C D E F G ",0,130
302 DATA "G A B C D E F G ",7,120
303 DATA "E F G A B C D E ",4,110
304 DATA "F G A B F C D E ",0,122
305 DATA "D E F G A B F C ",0,114
306 DATA "D E F G A B C ",0,114
307 DATA "B C D E F G A ",11,13
4
308 DATA "B F C D E F G A ",10,13
4
309 DATA "G A B F C D E F G ",7,120
310 DATA "A B C D E F G ",0,130
311 DATA "F G A B C D E ",0,122
312 DATA "E F G A B F C D ",0,110
313 DATA "C D E F G A B F ",0,110
314 DATA "E F G A B C D E ",4,110
315 DATA "C D E F G A B ",1,110
316 DATA "A B F C D E F G ",0,130
317 DATA "F G A B F C D E ",0,122
318 DATA "B C D E F G A ",11,13
4
319 DATA "G A B C D E F G ",0,120
320 DATA "D E F G A B F C ",1,114
321 DATA "B F C D E F G A ",10,13
4
322 DATA "F G A B C D E F G ",0,120
323 DATA "D E F G A B C D E ",0,114
324 DATA "G A B F C D E F G ",0,120
325 DATA "E F G A B F C D E ",0,110
326 DATA "C D E F G A B C D E ",1,110
327 DATA "A B C D E F G A B C ",10,13
0
328 DATA "C D E F G A B F G ",11,13
3
329 DATA "A B F C D E F G A B ",8,130

```

You may not have noticed that the very first line of the program was

```
1 GO TO 900
```

but this is the pattern which has been adopted throughout the book. The segment of program at line 900 is not a menu. This would be awkward for a program such as this, designed to be put together like building bricks to a different pattern each time. The code which follows simply sets the variables needed for all the pro-

grams that can be put together from this collection of blocks. "e\$" is set to 3 because this is the maximum value that can be keyed in response to the invitation to choose the difficulty level. "l", the duration of the notes, is set to a value which is related to the degree of difficulty "di". The score is held in "s" and the total number of questions is held in "t".

```

900 PRINT "          SPECTRUM MUSIC
MUSIC"
910 PRINT "*****Chose a difficu
lty level*****between 1 and 3*****K
ey the number and then press*****E
NTER.": INPUT z$
920 CLS: LET d$="3": GO SUB 4:
IF z$="" THEN GO TO 900
930 LET di=VAL z$: LET l=.1*(4-
di): LET s=0: LET tt=s

```

The collection of routines is complete. They can now be put to use to produce a display like the following:



The second of these example displays was produced by the last of the driver programs which deals with the problem of reading a passage of rhythm correctly. The first example was produced by the next piece of code. The lines between 1000 and 1210 check the musician's knowledge of keys, the notes within a scale and their ability to distinguish between the different types of scales. An arpeggio and a scale are played after a staff and a key signature have been drawn. The usual procedure of giving the purpose of each sub-routine has been set aside for most of this chapter because it is necessary to become familiar with the effects of the routines in order to write driver programs.

```

1000 PRINT "In a moment you will
hear an""arpeggio and a scale.
""There are several questions
to""answer go study the display
well": PAUSE 250
1100 CLS : GO SUB 10: GO SUB 90:
GO SUB 110: GO SUB 130: GO SUB
120

```

In line 1110 the limit on the input to z\$ is fed into e\$ before the input is checked. If the input is inappropriate then the user is asked for another try. The "minor" flag, "g", is used to check the input for correctness. If the answer was wrong the sub-routine will start the correct answer: "Wrong...it was", but not finish the message. The rest of the message is to be found in b\$ and the PRINT statement in line 1130 finishes the job. If the difficulty level is 3 then the user will be expected to distinguish between harmonic and melodic minor scales. If "di" is less than 3 or the scale is a major one, then the lines 1150 to 1154 are not needed and are by-passed. The type of minor scale is recognised from the initial letter held in c\$.

```

1110 GO SUB 2: PRINT "Key 1 for
Major, 2 for Minor": INPUT z$: L
ET e$="2": GO SUB 3
1120 IF z$="" THEN GO TO 1110
1130 IF VAL z$(<>g)+1 THEN GO SUB
150: PRINT b$(LEN b$-5 TO ): GO
TO 1160
1140 IF di<3 OR g=0 THEN GO SUB
140: GO TO 1160
1150 GO SUB 2: INPUT "Harmonic (1
) OR Melodic (2)?": z$: LET e$="2"
: GO SUB 3: IF z$="" THEN GO TO
1150
1152 IF (VAL z$=1 AND c$(2)="H")
OR (z$="2" AND c$(2)="M") THEN
GO SUB 140: GO TO 1160
1154 GO SUB 150: PRINT c$

```

The tonic or keynote of the scale is to be found in a\$ (1) and a\$ (2) indicates a sharp or a flat note. The user is invited to key the tonic of the scale in lower case. The routine at line 7 converts this input into upper case before it is matched against the data in a\$ which is in upper case. The limits on the input to z\$ are set by setting d\$ to "a" and e\$ to "gs" for A natural and G sharp.

```

1160 GO SUB 2: PRINT "Key the to
nic of the scale": INPUT "Eg ""c
"" or ""ef"" (for E flat)": z$: GO
SUB 2: LET d$="a": LET e$="gs":
IF LEN z$=1 THEN LET z$=z$+" "
1170 GO SUB 5
1180 IF z$="" THEN GO SUB 2: GO
TO 1170
1190 GO SUB 7
1200 IF z$=a$( TO 2) THEN GO SUB
140: GO TO 1220
1210 GO SUB 150: PRINT b$

```

All these programs are designed to drop through to the next if there is another program in place. The programs can be used one after the other or, as in the case of the next program, several times in succession. The repetition is achieved simply by means of a FOR/NEXT loop. The screen display is still intact so the same key signature is used for further questions but this is not necessary. A quick call to the routines at 10, 90 and 110 will select a new scale, present a fresh display and re-set all the necessary variables. An arpeggio is played and a note selected

from the scale. The note is sounded and the user is invited to indicate the note number within the scale, 1 for the key note, 8 for the octave, 5 for the dominant etc. After the response has been checked and the result printed on line 15, the note is drawn on the staff and the user is asked to name it. The answer has to be correct within the scale, no help is given with the accidentals in minor keys. Line 1400 converts the answer to upper case and adds a space to the answer, if necessary, and then the result is compared with the characters held in a\$ and pointed to by the variable "h1": a\$((h1+1)*2-1 TO (h1+1)*2).

```

1300 FOR k=1 TO 5: GO SUB 130: G
0 SUB 160
1330 BEEP l,h: GO SUB 2: PRINT "
key the scale position(1 to 8)":
INPUT z$: LET e$="8": GO SUB 3:
IF z$="" THEN GO TO 1330
1340 IF VAL z$=h1+1 THEN GO SUB
140: GO TO 1360
1350 GO SUB 150: PRINT "note ";h
1+1
1380 GO SUB 80: GO SUB 82: LET a
=a+16: LET b=b1
1390 INPUT "Name the note.":z$:
GO SUB 2: LET d$="a": LET e$="gs
": GO SUB 5: IF z$="" THEN GO TO
1390
1400 GO SUB 7: IF LEN z$=1 THEN
LET z$=z$+" "
1410 IF z$=a$((h1+1)*2-1 TO (h1+
1)*2) THEN GO SUB 140: GO TO 143
0
1420 GO SUB 150: PRINT a$((h1+1)
*2-1 TO (h1+1)*2)
1430 NEXT k

```

On the tape which accompanies this book the library of sub-routines and data is called MUSIC, the last program is called MUSIC 1 and so the obvious name for the next program is MUSIC 2. There are two parts to MUSIC 2, one which uses the staff and one which uses a different style of display for variety. The new display is in the form of a list of intervals with a cursor which moves up and down the list and so is similar to some of the displays given earlier in the book. First though, the code which plays a scale and then selects a note from it. The note is fed into a randomly selected array element and then the rest of the array is filled with bogus notes. The user is asked to indicate which of the notes printed on the staff is the one which is sounding. Lines 1515 and 1516 prevent the generation of notes which may cause problems with the display or which are too high or low for good sound. The answer is used as the element number of array z(3). If the contents of the array match the value of "b" then the answer is right. If the number was wrong then the machine does not name the note but it prints it in place on the staff.

```

1500 FOR n=1 TO 5: CLS : GO SUB
10: GO SUB 90: GO SUB 110: GO SU
5 120: GO SUB 160: LET b1=b: DIM
y(3): LET y(INT (RND*3)+1)=b

```

```

1510 FOR k=1 TO 3: IF y(k)=0 THEN
N LET y(k)=b+4*INT ((RND*3)+1)*4
: IF RND<.5 THEN LET y(k)=b-INT
((RND*3)+1)*4
1515 IF y(k)<50 THEN LET y(k)=y(k)
+32
1516 IF y(k)>170 THEN LET y(k)=y
(k)-32
1520 NEXT k: FOR k=1 TO 3: LET b
=y(k): GO SUB 80: GO SUB 82: LET
a=a+12: NEXT k: LET b=b1
1530 BEEP l,h: PRINT "Which note
: 1, 2 or 3?": INPUT z$: GO SUB 2:
LET e$="3": GO SUB 3: IF z$=""
THEN GO TO 1530
1540 IF y(VAL z$)=b THEN GO SUB
140: GO TO 1560
1550 GO SUB 150: LET a=a+24: GO
SUB 80: GO SUB 81: GO SUB 82: GO
SUB 2
1560 NEXT n

```

The list and cursor are printed using the contents of the data lines between 30 and 42. Line 1600 RESTORES the DATA line to be used next to 30. The twelve strings are read and printed on the screen to form the list. The number variable, "a" is read each time as well but is not used. This is necessary to prevent the computer trying to read this number variable as a string and giving an error message. Next, a line of data is chosen at random but the later lines are only chosen if the difficulty level selected is high.

```

1600 FOR n=1 TO 10: CLS : RESTOR
E 30: FOR j=0 TO 12: READ a$,a:
PRINT AT j,1:a$: NEXT j
1610 RESTORE 30+INT (RND*8): IF
di>1 AND RND>.5 THEN RESTORE 36+
INT (RND*3)
1620 IF di=3 AND RND>.7 THEN RES
TORE 41+INT (RND*2)

```

At line 1630 the name of the interval is loaded into a\$, the number of semitones in the interval is read into "c" and then a random note is chosen between the C below and the C above middle C. The line on which the cursor is to be printed is fed into the variable "a" and this is set initially to 5. The random note is sounded and then the interval is sounded by adding the number of semitones in the interval to "b" and sounding the resulting note. Next, instructions are given to enable the user to move the cursor and to choose the interval name.

```

1630 READ a$,c: LET b=INT (RND*2
4)-12
1640 LET a=5: BEEP l,b: PAUSE l*
5: BEEP l,b+c
1650 PRINT AT 16,0:"Key A to mov
e up the list." "Key Z to move d
own." "Key P to choose the inter
val."

```

Line 1660 places the cursor in position and the short delay prevents the cursor moving too quickly up and down the list. Without this short delay the cursor is not very controllable. The keyboard is "live" at this stage, the ENTER key is not needed to confirm the input. This is achieved by means of the INKEY\$ command. The last statement in line 1660 makes the machine wait until a key is pressed. As soon as a keystroke is made the machine prints out the cursor with a space, knocks one off the value of "a" if the "A" key is being pressed and then makes sure that "a" doesn't fall below 0 so that the cursor is not pushed off the top of the screen. Line 1680 does a similar but opposite set of jobs if the "Z" key is being pressed and 1690 sends the computer off to another part of the program if the "P" key is in operation.

```

1660 PRINT AT a,0;"█": PAUSE 10:
IF INKEY$="" THEN GO TO 1660
1670 PRINT AT a,0;" ": LET b$=IN
KEY$: PRINT AT a,0;" ": IF b$="A
" THEN LET a=a-1: IF a<0 THEN LE
T a=0
1680 IF b$="Z" THEN LET a=a+1: I
F a>12 THEN LET a=12
1690 IF b$="P" THEN GO TO 1710
1700 GO TO 1660

```

If "P" was pressed then the computer will end up here. The data is fed into c\$ and then checked against a\$. The value of the cursor control variable "a" not only points to the list, it also points to the data lines. If the answer is wrong then the machine has to print the correct answer and this is to be found in a\$.

```

1710 RESTORE 30+a: READ c$: IF c
#(<>a$ THEN GO SUB 150: PRINT a$:
PAUSE 30: GO TO 1730
1720 GO SUB 140
1730 NEXT n

```

MUSIC 3 is a stave based program so the first job to be done is to clear the screen, print a stave, select a key and print the key signature. This program will examine the musician's ability to recognise chords in different inversions. It would be better to play the chord rather than the arpeggio but the Spectrum has only one voice so an arpeggio it will have to be. An array, z(6) is set up to hold the pitch and note positions of the three notes which will go to make up the chord. The pitch is fed into the last three elements. The variable "h" values have the following meanings. 3 represents the root position, 1 the 1st inversion and 2 the second inversion. The values fed into the array are decided by the value of "h" and the value of the "minor" flag "g". If "g" is set to 1 then the scale is minor and the third note will have to be flattened.

```

1800 FOR n=1 TO 5: CLS : GO SUB
10: GO SUB 90: GO SUB 110: LET b
1=b: DIM z(6)
1810 LET h=INT (RAND*3)+1: IF h=3
THEN LET z(1)=b: LET z(4)=c: LE
T z(2)=b+6: LET z(5)=c+4: LET z(
3)=b+16: LET z(6)=c+7: IF f=1 OR
f=3 THEN LET z(5)=c+3
1820 IF h=2 THEN LET z(1)=b-20:
LET z(4)=c-8: LET z(2)=b-12: LET
z(5)=c-5: LET z(3)=b: LET z(6)=
c: IF f=1 OR f=3 THEN LET z(4)=c
-9
1830 IF h=1 THEN LET z(1)=b-12:
LET z(4)=c-5: LET z(2)=b: LET z(
5)=c: LET z(3)=b+8: LET z(6)=c+4
: IF f=1 OR f=3 THEN LET z(6)=c+
3

```

If di = 1 then some extra help is needed and the notes are printed on the stave by line 1840 before line 1850 plays the notes. The input is checked against "h" and the same variable is used to indicate the right answer if the user was mistaken. After hearing the chord the user should be able to identify the scale and so the machine requests a further input and then checks this against b\$. The switching between a\$ and z\$ in line 1900 is to enable the routine at line 7 to change the first letter only of the reply from lower to upper case.

```

1840 IF di=1 THEN LET b=z(1): GO
SUB 80: LET b=z(2): GO SUB 80:
LET b=z(3): GO SUB 80
1850 BEEP l,z(4): BEEP l,z(5): B
EEP l,z(6)
1860 GO SUB 2: PRINT "Broken cho
rds": "Key 1 for first inversion
": "2 for second inversion": "3 fo
r root position"
1870 INPUT z$: LET a$="3": GO SU
B 3: IF z$="" THEN GO TO 1870
1880 IF VAL z$=h THEN GO SUB 140
: GO TO 1900
1890 GO SUB 150: PRINT h: PAUSE
30:
1900 GO SUB 2: PRINT "Name the k
ey (eg e f (at major)": INPUT a$:
LET z$=a$(1): GO SUB 7: LET z$=z
$+a$(2 TO ): IF z$=b$ THEN GO SU
B 140: GO TO 1920
1910 GO SUB 150: PRINT h$: PAUSE
30
1920 CLS : NEXT n

```

The last of these sample driver programs is a little longer than the others and a lot less dependent on the library of routines. This is just as well because the program and the whole library would not fit into the 16K Spectrum. The routines which the rhythm program, MUSIC 4, needs are:

- The services between lines 2 and 8;
- Note drawing between lines 80 and 86;

The correct/incorrect routines at 140 and 150; and
 The ledger lines routine between lines 170 and 173.

As an alternative to this last item the routine at line 80 could have the "GO TO 170" replaced with a RETURN.

MUSIC 4 prints eight bars of notes, asks the user to tap the rhythm presented and then plays the result and gives an assessment of the attempt. In case eight bars of mainly quavers in seven four time are generated, quite a lot of space will have to be made available and so a storage array of one hundred elements is set up, z(100). The variable "t" holds the number of quavers in a bar, "t1" holds the number of quavers to the end of the bar, "t2" holds the current bar number. "a" and "b" have their normal meaning when the computer is printing notes, the screen co-ordinates. If "di" is set to 2 then there is a chance that 6/8 time will be selected and if "di" is equal to 3 then it is possible that 5/4 and 7/4 will crop up. A loop is set up to count through the notes and if the end of a bar is recognised ("t1" is less than or equal to zero), the computer is sent off to line 2070 to draw a bar line.

```

2000 GO SUB 2: PRINT "Here are 8
ight bars of notes": LET t=2+(IN
T (RND*3)+2): LET t2=0: LET a=12
: LET b=158: IF di>1 AND RND<.3
THEN LET t=12
2010 IF di=3 THEN LET t=t+2
2020 LET t1=t: DIM z(100): FOR j
=1 TO 100: IF t1<=0 THEN GO TO 2
070
  
```

Line 2030 starts by making the first note a crotchet, z(j)=2, but changes its mind and makes it a quaver if RND<.5 (given half a chance that is). At line 2040 it might change again and make the note a minim, z(j)=4. If the note is a minim or a crotchet, if the difficulty level is higher than 1 and if one chance in four comes up, then the note is made into a dotted version of itself, worth half as much again. The final note value is knocked off the value of "t1" and if, by some chance the note has ended up being worth five quavers then it is turned into a minim, value 4, and the next note is made a quaver.

```

2030 LET z(j)=2: IF RND<.5 THEN
LET z(j)=1
2040 IF RND<.5 THEN LET z(j)=4
2050 IF di>1 AND RND>.75 AND z(j)
)=2 OR z(j)=4 THEN LET z(j)=z(j)
*.5
2060 LET t1=t1-z(j)
2070 IF t1<=0 THEN LET z(j)=z(j)
+t1:
2080 IF z(j)=5 THEN LET z(j)=4:
LET j=j+1: LET z(j)=1:
  
```

The end of a bar is recognised if "t1" is equal to or less than zero. "t1" is set back to a full bar of quavers again, one is added to the bar number "t2" and then the computer checks to make sure that it is not at the end of the eight bars before going back

to the start of the loop. If the music is at an end then "j" is set to the limit value of 100 so that the NEXT "j" will send the machine straight on. If "t" is set to 12 then the time must be 6/8 and this is printed in front of the line of notes. For any other time signature all the machine has to do is print "t/2" on one line and "4" under it on the next. The apostrophe simply forces another line of print.

```

2090 IF t1<=0 THEN LET t1=t: LET
t2=t2+1: LET j=j+1:
2100 IF t2=8 THEN LET j=100
2110 NEXT j: IF t=12 THEN PRINT
AT 1,0;6'8: GO TO 2130
2120 PRINT AT 1,0;1/2'4

```

The notes to be drawn are indicated by the values held in the array elements. The "CIRCLE a+5, b+2, 1" in lines 2160 and 2180 draw in the dots. If the array element is empty then this signals the end of a bar so a bar line is drawn. If the next element is also empty then the end of the music must be here and the computer can leave the loop by setting "j" to the limit value of 100.

```

2130 FOR j=1 TO 100
2140 IF z(j)=1 THEN GO SUB 80: G
O SUB 81: GO SUB 82: GO SUB 85
2150 IF z(j)=2 THEN GO SUB 80: G
O SUB 81: GO SUB 82
2160 IF z(j)=3 THEN GO SUB 80: G
O SUB 81: GO SUB 82: CIRCLE a+5,
b+2,1
2170 IF z(j)=4 THEN GO SUB 80: G
O SUB 82
2180 IF z(j)=6 THEN GO SUB 80: G
O SUB 82: CIRCLE a+5,b+2,1
2190 IF z(j)=0 THEN PLOT a,b+10:
DRAW 0,-20: IF a>180 THEN LET a
=0: LET b=120
2200 IF z(j)=0 AND z(j+1)=0 THEN
LET j=100

```

On a machine which makes its noises at constant volume it is difficult to indicate the accent normally heard at the start of a bar. The solution to this difficulty is to make the first note of the bar full length and to shorten the others. The time that would be lost in doing this is made up by padding with pauses. The pause length can be matched to the note length by multiplying the length by 50. These next two lines beat the time of the piece on middle C. The LET a=a+12 at the beginning is to move the draw position on one note's worth to the right before the machine loops back to the start of the cycle shown in the previous section of code.

```

2210 LET a=a+12: NEXT j: GO SUB
22: PRINT "Here is the time"
2220 FOR j=1 TO 3: BEEP 2*t,0: F
OR k=1 TO t/2-1: BEEP 1,0: PAUSE
t*50: NEXT k: NEXT j

```

To make things easier for those struggling on difficulty level 1, the next segment plays the rhythm for the musician to copy.


```

2230 IF d1>1 THEN GO TO 2275
2240 GO SUB 2: PRINT "Here is th
e rhythm": LET k=0: FOR j=1 TO 1
00: BEEP z(j)+1,0: IF z(j)=0 AND
k=1 THEN LET j=100: GO TO 2250
2250 IF z(j)=0 THEN LET k=1: GO
TO 2270
2260 LET k=0
2270 NEXT j

```

"y" will end up holding the number of notes in the piece. "w" will hold the value of the shortest note in the piece after the computer has completed this next loop.

```

2275 LET y=0: LET w=99: FOR j=1
TO 100: IF z(j)<>0 THEN LET y=y+
1
2280 IF z(j)<>0 AND z(j)<w THEN
LET w=z(j)
2290 NEXT j: GO SUB 2: PRINT "No
w key the rhythm on any key"

```

Now that the real notes are held in the array z(100) the musician can enter his or her version of the rhythm and these will end up in array y(y), dimensioned to the number of notes in the piece. First the machine is made to count as fast as it can, but only after a key is pressed (line 2310). The machine keeps adding 1 to "x" while the note is pressed, while the note is released and it only stops when the next note is pressed. In line 2330 the Spectrum checks to make sure that it is not dealing with the last note in the piece (j=y). If this is the case then the counting stops as soon as the note is released. The total value of "x" is transferred to the array element y(j) before the loop doubles back.

```

2300 DIM y(y): FOR j=1 TO y:
2310 LET x=0: IF INKEY$="" THEN
GO TO 2310
2320 LET x=x+1: IF INKEY$<>"" TH
EN GO TO 2320
2330 LET x=x+1: IF INKEY$="" AND
j<>y THEN GO TO 2330
2340 LET y(j)=x

```

To cover the inevitable delay caused by the next loop a message is printed for the user to read. The loop, meanwhile, is finding the shortest note in the user's version of the rhythm and storing it in the variable "w1". This done, the computer divides each array element by the length of the shortest note and stores the new value, correct to the nearest whole number, in the array again.

```

2350 NEXT j: GO SUB 2: PRINT "No
w here is your rhythm": LET w1=9
9: FOR j=1 TO y: IF y(j)<w1 THEN
LET w1=y(j)/w
2355 NEXT j
2360 FOR j=1 TO y: LET y(j)=INT
(y(j)/w1+.501): NEXT j

```

Line 2380 plays the rhythm entered at the keyboard and then the final loop checks the arrays against each other, ignoring the empty elements in the first which mark the bar lines, giving one

mark where the match is exact and giving half a mark where it is close. To finish, the marks are added to the total score "s" and the number of notes are added to the number of questions.

```
2370 GO SUB 2:
2380 FOR J=1 TO y: BEEP y(J)+1,0
: NEXT J
2390 LET a=0: LET z=1: FOR J=1 T
O y: IF z(z)=0 THEN LET z=z+1
2400 IF y(J)=z(z) THEN LET a=a+1
2410 IF y(J)=z(z)-1 OR y(J)=z(z)
+1 THEN LET a=a+.5
2420 LET z=z+1: NEXT J: GO SUB 2
: PRINT "score=";a;" out of ";y
2430 LET s=s+INT a: LET tt=tt+y:
PAUSE 100: CLS : GO TO 2000
```

Operating instructions are not very appropriate for these programs. They are designed to stand alone and to do their jobs if the users simply follow the instructions presented on the screen. The amount of musical knowledge required to use the programs on difficulty level 1 is quite small. The package should be an enjoyable way of improving the users' skill and to some extent, their technical knowledge as well.

Here is a detailed account of the sub-routines, data blocks and variables used in the package.

<i>Line Numbers</i>	<i>Routine/Data type</i>
2	Blank line, reset PRINT AT to 15, 0;
3	Checks z\$ for a digit between 1 and VAL e\$
5	Checks z\$ for characters between d\$ and e\$
7	Converts lower case in z\$ to upper case
10	Draws stave and clefs
20	Scale interval DATA
30	Interval DATA
50	Draws sharps
51	Draws flats
60	Key signature DATA (sharps)
70	Key signature DATA (flats)
80	Draws note
81	Fills in note
82	Draws stem
85	Draws tail
90	Select a new scale and finds the name of the scale
110	Prints the key signature
120	Plays a scale
130	Plays an arpeggio
140	Indicates a correct answer
150	Indicates a wrong answer
160	Selects a note from a scale and generates co-ordinates
170	Draws ledger lines as necessary
300	Scale DATA

MUSIC VARIABLES

<i>Variable name</i>	<i>Description</i>
a & b (b1)	Screen co-ordinates for notes, sharps, flats etc.
c (c1)(c2)	Pitch of notes being considered
d	Data line containing scale information
e	Number of sharps or flats in the scale
f	Related to the type of scale: sharp, flat, major, minor
g	Major/minor flag
h (h1)(h2)	Pitch of notes within intervals
l	Length of note (related to di)
m	Inversion of the chord
s	Score
t (t1)(t2)	Time signature, amount of bar left, bar number within sequence
tt	Number of questions asked
w (w1)	Shortest note in rhythm (in user's rhythm)
x	Rhythm counter
y	Number of notes in passage
j, k & n	Loop counters
z(14)	Scale intervals
z(6)	Chord notes
z(100)	Notes in rhythm passage
y(3)	Random notes
y(y)	Notes in user's rhythm
a\$	Notes in scale
b\$	Name of scale
c\$	Type of minor scale
d\$ e\$	Limits of input
z\$	Temporary input

Appendix: ZX81 Hardware Modifications

These are all very simple modifications. They require no more electronic expertise than the ability to solder neatly. None of the programs in the other chapters require any of these modifications in order to make them run smoothly, but each suggestion more than pays for the small amount of trouble involved.

POWER SUPPLY

The Sinclair power supply gives an output of 9 volts DC via a 3.5 mm jack plug, with the tip of the plug positive. It is very convenient because all the works are contained within its very large, plug type body. The only lead is the 9 volt supply to the ZX81. If you bought a ZX81 kit without a power supply and wish to build one, then this one has definite advantages. If you have a power supply and plan to use your computer for long periods, you might still consider the four or five pounds that the components will cost, worth spending for the following reason.

The ZX81 voltage regulator is provided with a small heat sink which gets very hot. It is reasonable to assume that a machine which works at a high temperature and does a particular job, will not be as reliable as one which does the same job at a much lower temperature. The ZX81 will work as long as it is fed with DC current at a voltage between seven and eleven volts. The higher the feed voltage, the harder the voltage regulator has to work to drop it down to the five volts needed by the chips. The more work it has to do the hotter it gets and, as likely as not, the shorter will be the mean time between failures of the ZX81, baking quietly in all this waste heat. If the power pack were designed to deliver the minimum of seven volts, then all would be well until the first power crisis, when the mains voltage is dropped by twenty or so

volts. This would drop the output of the power pack below the seven volts required and switch off the machine. A good compromise is to design a power pack to run at eight volts. This has a significant cooling effect.

Some users have reported a mains pack failure after running the computer for long periods of time. The fuse inside the power pack fails and is difficult to replace because it is soldered in place. Another problem which has been mentioned is that the power supply issued with the machine sometimes causes a slow ripple to pass through the picture. The more meaty, lower voltage power supply eliminates the ripple effect from those machines which suffer from it, and at the same time is less likely to blow fuses.

You will need:

- 1) line supply-to-8V transformer rated at 2A
- 1) rectifier bridge circuit rated at 2 amps, 50 p.i.v.
- 1) 2000 μ F electrolytic capacitor with a working voltage of at least 16 volts
- 1) 1 amp fuse and fuse holder.

If you are not too sure about building electrical equipment, then you should get an electrician to check over the power pack before you plug it in. You will be dealing with mains voltage and this can be very dangerous. The main things to check are that the transformer is connected the right way round, and that the tip of the plug to the ZX81 is positive. If the thick wires of the transformer are connected across the mains, then the ZX81 will blow up before the fuse has a chance to blow! Make sure that the thin wires of the primary coil are connected to the mains. Most transformers will have the primary and secondary connections marked.

The two amp rating on all the components takes into account the possibility of further peripherals being developed for the ZX81. These are appearing on the market at the time of writing and, by the time you read this, reviews should have appeared in the computing magazines. Many of these devices offer possibilities for the users who are interested in making the ZX81 work for its living and so it is wise to make provision for the power demands they will make. To comply with the relevant safety regulations, the unit must be mounted in a stout plastic (*not* metal) box, and all mains-carrying terminals must be insulated as well. The unit should be connected to the mains via a fused plug, fitted with a 1 amp or a 3 amp fuse.

When you have finished the power pack and wish to check the voltage and polarity of the plug, don't be alarmed at the high voltage reading on your multi-meter. The voltage will drop to its normal level when the power pack is loaded.

EXTRA MEMORY

At first, when the ZX81 was introduced, it was thought that the 3K RAM pack that many ZX80 users possessed would not work on the ZX81. A very minor modification to the printed circuit will make the extension work on both the ZX80 and ZX81. Remove the three plastic rivets and take off the back half of the case. The chips will now be visible and it is important to work on the side of the board where you can see the components.

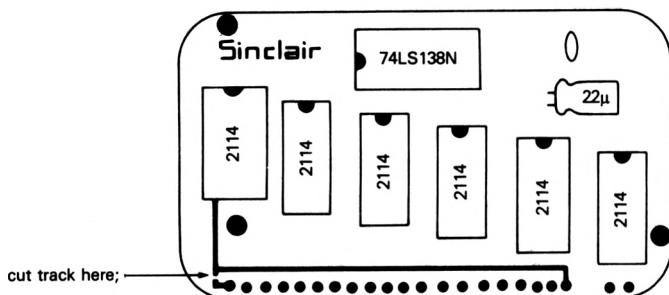


Figure A.3

All that is needed to make the 3K RAM pack work on *both* machines is to cut the track shown. To test the modification, plug in the pack, switch on and key in the following line:

```
10 DIM A ( 102 , 6 )
```

and RUN the program. You should be rewarded with a report code 0/10 indicating that the array of 612 numbers (102×6) has been successfully set up. Such an array requires at least 3060 bytes of space in the RAM. On the ZX80 the smaller RAM pack will give 4K of memory but the ZX81 switches off the 1K of memory on the main board when the extra RAM is attached. If the RAM pack is plugged in permanently the two 2114 chips can be removed and used in a 3K memory board. (Just a thought for those readers who have a small memory pack with less than the full complement of six 2114 chips installed!)

If you thought that the previous modification was simple, what will you think of the next? The press has been full of reports of programs being lost from the ZX81 whilst the 16K RAM pack is in place. The main cause of this problem seems to be the fact that the RAM pack is mounted at 90° to the main board and is thus liable to move a little on its mounting points. The lead in the solder coating of the edge connector contacts will become coated with lead oxide quite quickly and make the surface conduct badly in places. Slight movement of the RAM pack on a less than scrupulously clean solder track causes momentary loss of power and more than momentary

loss of program. There are three cures. The first is to fit the ZX81 with "radio spares" self-adhesive feet. These lift the computer high enough to ensure that the RAM pack is clear of the ground and the connector is not flexed when you press the keys. This is usually a complete cure for RAM problems. The second is to steady the RAM pack on its mountings with a piece of plasticine between the back of the computer and the front of the RAM pack. The appearance of the second cure is a little less technical than might be desired but the improvement in reliability is worth all the odd comments from onlookers. The third cure will be given in detail under the heading "eliminating the tangle".

A PROPER KEYBOARD

The low cost of the ZX81 is, to a considerable extent, due to the low cost of the keyboard. There are advantages to the installation of a normal keyboard; there is more room for "person sized" fingers and it is possible to "touch type" on a keyboard which is not almost completely flat. There are on offer, several beepers which indicate to the typist each time a keystroke has been successful. Such annoying devices would be unnecessary if you could feel a key making contact.

Keyboards specially designed for the ZX81 are sold by several manufacturers, and these are in general the best bet. From time to time you will come across old keyboards in junk shops, on jumble sale stands at computer shows, or in surplus stores. Most of these are coded in some way and need to be stripped down to remove the intricate interconnections of the keys under the board. If you attempt such a job, make sure that no hidden connections remain, by checking *all* the keys with a multimeter for each keystroke.

When the keyboard has been stripped down so that it is no more than a set of isolated switches, it can be wired up as shown in the diagram (Figure A.4). Choose two contacts per key from the several you will probably have and use the same two on every key. Connect up one set of contacts as shown in the top diagram and connect the second set up as shown in the lower drawing. If you wish, you can make extra keys available if there are more than forty buttons on your board. These could form a number pad with an extra decimal point if you intend doing a lot of number work on the computer. Bring out the fly leads and connect them to the underside of the computer board as shown. The connections will be easier to sort out if you can use a different colour for each connection. If you use the underside of the board, then the original keyboard will still function and the two can be used together if required. This might be useful for games.

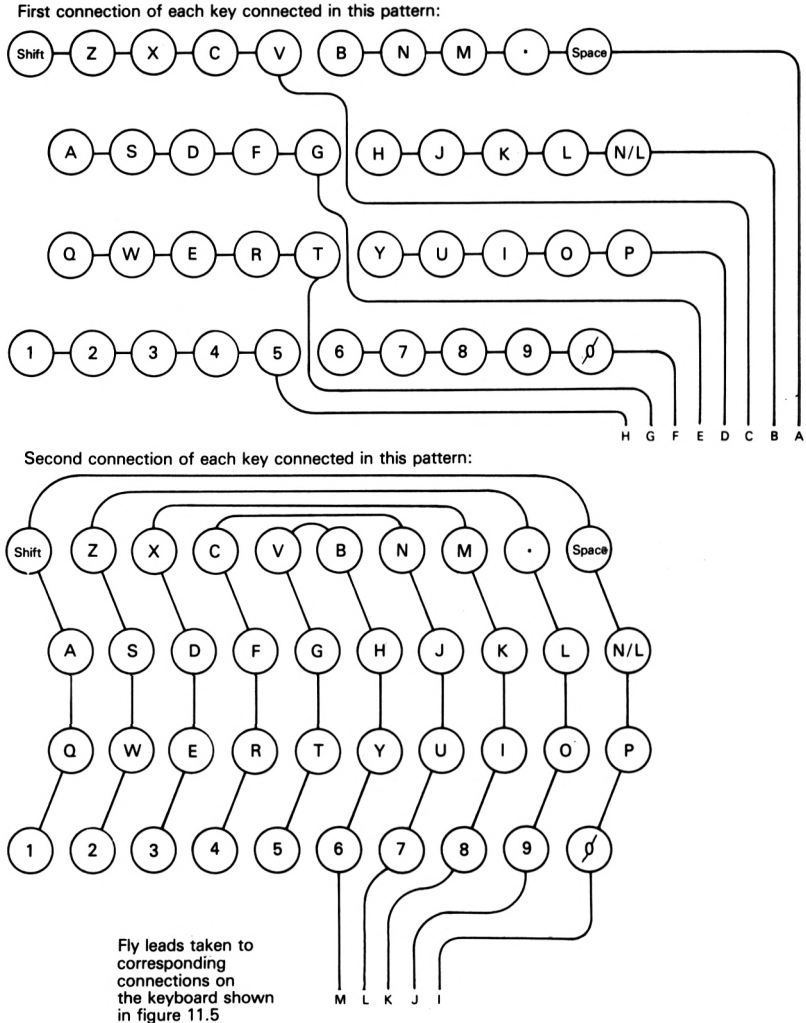


Figure A.4 Underside of the keyboard and interconnection of the keys

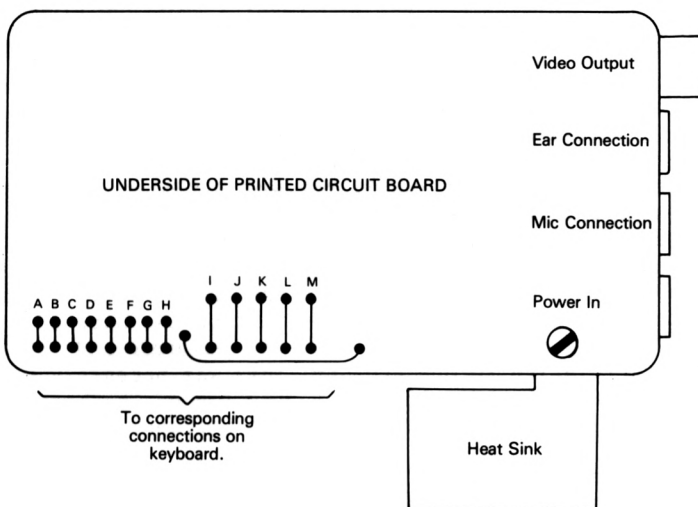


Figure A.5 Linking the ZX81 to a normal keyboard

This is the largest project in the chapter and the wiring will take some time to complete and will need careful checking. The cost is likely to be low and, as the likelihood of damage to the equipment is very small; don't be put off. The computer will remain available during the project as the bulk of the work is done on the keyboard alone. When attaching the leads to the computer some important precautions must be taken. The chips will not take kindly to being blasted with AC, even the small amount that leaks from a grounded iron. The safest plan is to heat the iron thoroughly and then switch off both the computer and the iron before attempting a joint. The tinned leads can be quickly soldered in place before the iron cools and the iron can be soon brought back to temperature between each lead. This sounds more involved than it really is. There are only thirteen leads to solder. Alternatively, use a ceramic-shafted iron.

Re-designating the keys is a job that can be off-loaded onto an artistic friend. The job consists of filling the recess of each key (they are usually concave) with something like car body filler, sanding this smooth and priming it in some way to receive the ink drawing of the key characters and finally, sealing with a durable, transparent film. The self adhesive Sinclair keyboard can be cut up and used but the area of adhesive is small and the characters tend to move around on the keys and look untidy.

ELIMINATING THE TANGLE

Even with the plasticine pad in place, I have had problems when a lead has been joggled while I have been sorting through papers on the table. The cure was a little elaborate but resulted in a much more professional looking set up and no further program losses. The ZX81 was stripped out of its plastic box and mounted securely in a case built on the pattern of an executive brief case. The deck of the case was laid out to take my tape recorder and the keyboard, tilted slightly forward for more comfortable typing. Underneath the deck all the wiring was soldered in place, not plugged. The most important job of this type was to solder the 16K RAM pack to the printed circuit board of the computer using ribbon cable. This meant removing the RAM pack from its case, opening out the two halves to make it lie flat on the base of the case, and then securing it in place so that the computer and the memory did not move relative to one another. This was necessary because the soldering was "tack" soldering to the top of the board to very small areas of strip, and such joints are not very robust.

It is not necessary to go to such lengths if you don't feel up to the task. The RAM pack can be connected to the board in the normal way in the box and secured in place to avoid the memory-losing joggling. The main benefit of the case is the lack of tangle and the more orderly nature of the computing area, which is conducive to more ordered programming.

More Real Applications for the ZX81 and ZX Spectrum

Two software cassettes are available to accompany this book. One contains programs for the ZX81, the other programs for the ZX Spectrum. Each costs £9.00.

ZX81 cassette

Side 1 contains

"DIR" "REACTIONS"
"TILL" "GAS LAWS"
"STATISTICS" "DOPPLER"
"CRICKET" "TRIANGLE"
"CARBON DATING" "PERISTALSIS"
"HALF LIFE" "ELECTROLYSIS"

Side 2 contains

"DIR" "NOTES"
"RENUMBER G" "MERGE 81"
"RENUMBER MC" "DRAW 1"
"RENUMBER B" "DRAW 2"
"SPIDER INVADERS" "HEART"

ISBN 0 333 34583 5

£9.00 (inc. VAT)

ZX Spectrum cassette

Contains

"dir" "gotcher"
"till" "music"
"statistics" "music 1"
"cricket" "music 2"
"renumber1" "music 3"
"renumber2" "music 4"
"renumber3"

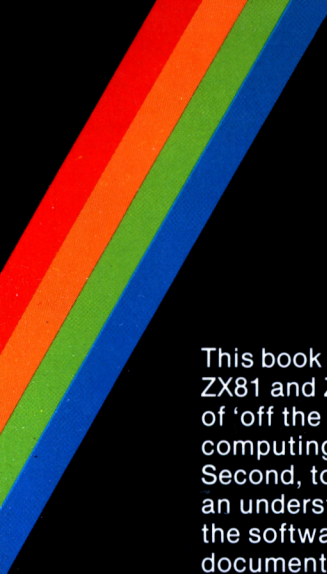
ISBN 0 333 34584 3

£9.00 (inc. VAT)

*These cassettes are available through all major bookshops...
but in case of difficulty order direct from*

Globe Book Services
Canada Road
Byfleet
Surrey KT14 7JL

£9.00 each
(please specify machine)



This book has two basic aims. First, to provide ZX81 and ZX Spectrum owners with a source of 'off the peg' programs which do real computing work in a wide field of applications. Second, to give the reader an insight into and an understanding of the techniques on which the software is built. To this end, full documentation and detailed commentary is provided.

The ZX computers have enormous potential that is often unexploited. This potential can be realised if programs for the ZX81 and Spectrum are made to be as good as software written for other machines. The 'building-block' approach described by the author will enable readers to utilise the ZX machines' capabilities more fully and to facilitate and speed up the production of high quality software of their own.

Randle Hurley is also author of *The Sinclair ZX81 — Programming for real applications*. (Macmillan, 1981).

Software Tapes Available

See inside the back of this book for details of the cassettes available to accompany it.

MORE RESEARCH SHOWS THAT FOR THE **NUMBER** OF **WORDS** IN **SENTENCES** AND **PARAGRAPHS** IN **WRITING** TO **IMPROVE** READING **COMPREHENSION** IN **ADULTS** WITH **READING** DIFFICULTIES, **TECHNIQUES** SUCH **AS** **TEACHING** **THESE** **TECHNIQUES** **ARE** **THE** **MOST** **EFFECTIVE** **AND** **EFFICIENT** **WAYS** **TO** **IMPROVE** **READING** **COMPREHENSION** **IN** **ADULTS** **WITH** **READING** **DIFFICULTIES**.

AMSTRAD

CPC



MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA



<https://acpc.me/>

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

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

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