CHARLES N. M. WILLIAMSON.

TRADE SECRET

INCLUDES:

- ☆ Making programs unlistable, unstoppable, invisible
- ☆ How to reveal such programs
- ☆ Examples, hints & tips
- ☆ Microdrive supplement

CHARLES N. M. WILLIAMSON.

TRADE SECRET

Copyright © G.A.Bobker, 1984

ZX GUARANTEED

29 Chadderton Drive
Unsworth
Bury
Lancashire

Distributed in Australasia by

MANUKAU COMPUTERS (N.Z.) LTD.

P.O.Box 76-310
Manukau City
New Zealand

COPYRIGHT: The contents of this book are copyrighted and all rights are reserved. This document may not, in whole or in part, be copied, photocopied, reprinted, translated, reduced to any electronic medium or machine readable form or reproduced in any manner without prior consent in writing from G.A.Bobker, ZX Guaranteed.

DISCLAIMER: Although every effort has been made to ensure that the contents of this book are technically and typographically correct, no responsibility can be accepted by G.A.Bobker, ZX Guaranteed, or its distributors for any loss or damage resulting from the use of the information, techniques or programs contained herein. The book is sold only on the basis of this understanding.

HIDING ITEMS IN A LINE

BY USING THE SAME INK/PAPER COLOUR

Putting the usual BASIC 'INK7:PAPER7' would not achieve our aim, but if the control codes for INK and PAPER are used then it would.

On page 183 of the Sinclair Spectrum Manual we can see that the control characters for INK and PAPER are 16 and 17. There are two methods of using these to hide an item.

The first method is to enter a line, but directly after the line number, get into E mode, then press the CAP SHIFT and 7 keys, and then continue the line. N.B. Set 'BORDER1' first. Assume our line is:

1Ø LET A = Ø

Enter this line as follows: Type '1Ø' and then get into E mode. Press the CAP SHIFT and 7 keys, don't type in the rest of the line, and press the ENTER key. Only the line number will be visible on video. As our action made the INK of the rest of the line white, the line is only invisible if the PAPER is also white. Note also that all lines after our line will also be affected. If you only want our line affected, then before pressing ENTER at the end of the line, get into E mode again and press the CAP SHIFT and Ø keys. This method can be used to make a line any colour you require and can be used in PRINT statements if desired. It is useless for concealing items as they will be revealed if the command PAPERØ is used. Try it and see!

An alternative method is to poke both the INK & PAPER to be the same colour. Again we'll use:

1Ø LET A = Ø

but this time we'll enter it with four spaces after the line number, thus:

1Ø ____ LET A = Ø

Provided that this is the first line in the program listing, then the memory location of these four spaces are 23759, 23760, 23761, 23762. In BASIC we normally use 'INK7:PAPER7'. To POKE in the INK/PAPER enter as a direct command the following:

```
POKE23759,16 : POKE 23760,7 : POKE 23761,17 : POKE 23762,7
```

Then press enter. Again only the line number is displayed and all lines after our line are affected. If you only wanted our

line to be affected then you could put four spaces at the end of our line and then POKE them to be the INK/PAPER you want the rest of the program to be. Note also that we could put more spaces at the start and end of our line and make it flash by POKEing in 18 and 1 (to ensure that this doesn't affect other lines, at the end of the line you must POKE in 18 & Ø). Obviously you'll have to calculate the locations of the spaces, but if you always EDIT your lines to temporarily be the first line of your program while POKEing in the control codes, you will find this easier. The first line in any BASIC program in the Spectrum always starts at location 23755. The last character in any line is the ENTER character, which is character 13. This can be found by PEEKing at values from 23755 upwards. A program which does this appears later in this booklet.

The main use of this 'hiding' is to either hide all of the program, or possibly to hide answers in adventure games. If a line should be:

```
INPUT A$ : IF A$ = "CORRECT" THEN GOTO 1ØØØ
```

and if the INK/PAPER colours are POKEd into the line before the ':' then you won't see that the answer should be "CORRECT" in this example.

REVEALING INVISIBLE LINES/ITEMS

LIST the program. If the first line is holding machine-code in a REM statement, then the line may only partially list, because the Spectrum can not decode some machine-codes into printable characters. In this case simply ask the Spectrum to list from the next line. If all is revealed then there's no problem. If you suspect there are hidden items, EDIT the line down onto a coloured border (it sometimes helps to to show that there are hidden items by revealing a block of all white characters in the EDITed line). Now move the cursor to the right - if the cursor appears not to move, or if the cursor disappears, then you know that there are hidden control characters in the line. Delete these mystery characters until the items are revealed.



DIRECTLY ENTERED ITEMS

In the original pogram, strings and/or variables could have been entered by direct commands, e.g.

```
LET A$ = "CORRECT ANSWER"
```

If you had in your listing 'LET A$ = "CORRECT ANSWER" ' then you could see this simply by LISTing the program. To prevent this from happening, the A$ could have been entered by a direct command (without a line number). If you suspect an 'answer' or other items have been hidden by this method, then go through the alphabet by using direct commands - 'PRINT A$', then try 'PRINT B$' etc., till 'PRINT Z$'. Also try 'PRINT A', then 'PRINT B' etc., till 'PRINT Z'. This could be awkward, since Spectrum variables (not strings) could be several letters long!

MAKING LINES TOTALLY INVISIBLE

This is not possible in a BASIC program, but a method can be used for part of a program. Here is an interesting exercise to prove the point. First LOAD in a BASIC program. Now enter as a direct command:

```
POKE 23755,255
```

LIST and the screen is blank. Do not press NEW while trying this exercise. Type in, then RUN the following program:

```
1Ø FOR I = 1 TO 15
2Ø PRINT "fred"
3Ø NEXT I
```

This will print 'fred' fifteen times on the video.... big deal! Next, delete each of the lines one at a time - Don't press NEW!. Now enter as a direct command:

```
POKE 23755,Ø
```

Magic! Your original program re-appears. Actually, it was there all the time but the POKE 23755 made the line number greater than 9999, and the Spectrum can only LIST or RUN programs with line numbers not greater than 9999. It was used here just to clear the video while we tried out a short program. This method is used in some games to 'hide' the machine-code routine of the game. Let's assume we have some machine-code in a REM statement, and it would normally appear as:

```
1Ø REM ABCDEFG
```

The actual characters after the word REM are meaningless to look at as they are actual machine-code values..... in this case we'll assume they look like 'ABCDEFG'. To get machine-code to RUN in the Spectrum, we would have in the program somewhere a line giving one of the following commands:

```
RANDOMIZE USR XXXXX
PRINT USR XXXXX
LET X = USR XXXXX
RUN USR XXXXX
```

where XXXXX would be a five digit number. For the hidden machine-code method, line 1Ø in the example would be the first line in the program. Enter as a direct command:

```
POKE 23755,255
```

This makes the line number greater than 9999 so the line will disappear. Now we enter the rest of the program, such as:

```
1Ø RUN USR 23778
```

All we would see on the video is this new line 1Ø. The original line is in memory and is in fact after this line. The program, when RUN, would obey line 1Ø and the 'RUN USR 23778' calls the machine-code routine in the hidden line.

If you have a program which appears to LOAD as BASIC yet only lists a short program, including a 'call' to a USR routine, then suspect it's the 'hidden machine-code' type. If you want, delete each line one at a time, then POKE 23755,Ø and the REM line would be revealed (You could 'look' at it using PEEK or by using a disassembler such as Infrared made by ACS of Leeds). Remember also that machine-code routines could be 'hidden' in the print buffer or the UDG area.



MAKING PROGRAMS UNSTOPPABLE

Firstly we'll look at methods of ensuring programs auto-start and either won't stop or 'crash-out' if stopped.

In machine-code programs, the program stays in machine-code and is made to recognise only certain keys.... and it doesn't take any action on any other keys - thus the BREAK key has no effect, or can be made such that if the BREAK key is pressed, the program jumps to memory location zero.... this will cause the program to NEW.

In BASIC programs, all inputs are made to be from an INKEY$ command to ensure no messages, etc., are printed on the bottom two lines of the video, and the Spectrum is made to use all 24 lines for screen action. Under these conditions pressing the BREAK key, or an illegal key in the program, would make the Spectrum try to stop and print an error message on the bottom of the video. But, as the Spectrum is told it is to use all 24 lines, this results in the program 'crashing', usually with an all black screen.

If you look at the Spectrum manual, page 175 near the top, you'll see a system variable called DFSZ. This is normally set at two and tells the Spectrum to reserve the last two lines for error messages. DFSZ is location 23659 and can be POKEd to be Ø, thus allowing all 24 lines to be used by the program. Let's try an experiment using DFSZ. Enter the following program:

```
1Ø BORDER 1 : POKE 23659,Ø : POKE 23658,8
2Ø FOR I = 1 TO 24
3Ø PRINT I
4Ø NEXT I
5Ø IF INKEY$ = "S" THEN POKE 23659,2 : STOP
6Ø IF INKEY$ = "P" THEN STOP
7Ø GOTO 5Ø
```

RUN it and you'll observe that we have printed on all 24 lines of video. Press the 'S' key to safely STOP the program. Now re-RUN the program and this time press the 'P' key.... the program will 'crash'. Why? In line 1Ø we POKEd 23659 to be Ø, which told the Spectrum we will use all 24 lines for PRINTing (and therefore no room for error message printing). The 'POKE 23658,8' had also been used, but this was purely to make sure that the Spectrum was in upper-case (capital letters) so it could see the 'S' or 'P' in lines 5Ø or 6Ø. Pressing 'S' made line 5Ø POKE DFSZ back to two so it could STOP O.K. Pressing 'P' made line 6Ø STOP, but it stopped the program with DFSZ still with a value of zero.

USING ERR-SP

On page 174 of the Spectrum manual you'll find ERR-SP as location 23613 (actually, it is 23613 and 23614). This is used

by the Spectrum to let it know which error message to give if there is an error. If in your program you include:

```
POKE 23613,Ø
```

then if the program is STOPped, the Spectrum gets confused and will 'crash-out'. This is not always 100% foolproof since the Spectrum is constantly changing the value in ERR-SP. To be truly effective in a BASIC program, it is necessary to include the 'POKE 23613,Ø' in every FOR loop, before and after all GOTO, GOSUB and PRINT statements. Indeed, long PRINT statements need to be broken and have 'POKE 23613,Ø' placed in them.

REMOVING THE 'CRASH' POKES FROM PROGRAMS

To remove the 'POKE 23613,Ø' and/or 'POKE 23659,Ø' from a BASIC program, the first thing to do is LOAD the program and not let it auto-start. This is achieved simply by using MERGE instead of LOAD. Look through the listing and delete all the offending POKEs from the program. You cannot use MERGE on machine-code programs or on very long BASIC programs. To now SAVE this new version of the BASIC program, and still make it auto-start, you must use:

```
SAVE "name" LINE 10
```

assuming it is desired to auto-start at line 1Ø. N.B. LINE in a SAVE statement is the keyword LINE which is on button '3' of your Spectrum.

Without knowledge of machine-code it is impossible to alter these programs. You can easily STOP normal machine-code programs by using the STOP program supplied on the 'ØØ7SPY' cassette from ZX-GUARANTEED at 29 Chadderton Drive, Unsworth, Bury, Lancashire, or in Australasia from Manakau Computers (N.Z.) Ltd (ØØ7SPY also has on the tape a genuine Kopyall Kopykat program). It is essential to use such a program to find where in memory the machine-code is normally put, and the length of the program. The method of STOPping is to LOAD the machine-code into a different area of memory by instead of putting 'LOAD "" CODE', use 'LOAD "" CODE 3ØØØØ. This would put the program into location 3ØØØØ onwards and the program then would not auto-start. To 'look' at the codes you can either PEEK at them or use a disassembler as stated previously. Please note that the program is still in machine-code. A simple program to PEEK at the above could be:

```
1Ø FOR I = 3ØØØØ TO 66666
2Ø PRINT I ; "=" ; PEEK I
3Ø NEXT I
```

BASIC PROGRAMS DISGUISED AS MACHINE-CODE

Since a BASIC program occupies bytes in the computer, it is just as easy to tell the Spectrum to SAVE it as bytes (machine-code). One example of this is the well known 'Time-Gate' gate. The first part is loaded by 'LOAD "" CODE', but if STOPped, shows it is actually in BASIC. If a program had included the POKE 23659,Ø in it, it couldn't be stopped without crashing (we could stop it!).

Auto-start bytes programs, including BASIC ones disguised as bytes, always start below the normal user memory area. This is because they are SAVEd with some, or all of the system variables. Although these are from 23552, most programmers SAVE from 235ØØ. To convert a disguised BASIC program back to BASIC so it can be studied, we first need to STOP the program. Next we want to SAVE it without the stuff below the user memory area. This is a little complicated to do, so what we'll do is first write a BASIC program, make it unstoppable, then convert it to machine-code format. Finally we will turn it back into just the BASIC. This is rather pointless, but the method used does explain how to apply it to an existing game you want to study. Enter the following and then SAVE it on tape before RUNning:

```
1Ø BORDER 2 : POKE 23659,Ø
2Ø FOR I = 1 TO 1Ø
3Ø PRINT "BASIC PROGRAM"
4Ø NEXT I
5Ø PAUSE Ø
```

SAVE on cassette by:

```
SAVE "TEST" CODE 235ØØ,5ØØ : RUN
```

Note that when SAVEing a bytes program you must include in the SAVE statement not just the name and keyword CODE, but also the start address of the program plus the length. The program is less than 5ØØ long but it is best to SAVE greater than the program length, as it is a BASIC program really and we need to SAVE part of the work area of memory. 5ØØ is a round figure to use in our example. The ': RUN' at the end is one method of ensuring this program will auto-start when LOADed in.

Having SAVEd the program, prove that it works by LOADing it with 'LOAD "" CODE'. You cannot STOP the program without a 'crash'. We will now make the program STOP and then convert it back to BASIC.

Ensure the machine is empty by powering off then on again. LOAD our 'TEST' program by:

```
LOAD "" CODE 3ØØØØ
```

(The value 3ØØØØ has been chosen for convenience here - it could have been any value over about 245ØØ). Our 'TEST' is now in 3ØØØØ onwards. The user memory area of a Spectrum is 23755 onwards. Since we have relocated our 'TEST' into 3ØØØØ onwards, we have to calculate the new address. Originally it was 235ØØ onwards, and addresses of 235ØØ to 23754 inclusive are to be deleted from our program. This means we don't want the first 255 bytes. Therefore to SAVE the part we do want, use:

```
SAVE "TEST" CODE 3Ø255,246
```

and only the original BASIC area should now be on the tape, but still held as bytes. You cannot simply LOAD this into the user area with a 'LOAD "" CODE 23755'. It is essential to first ensure the user area in this case is loaded with sufficient bytes to at least cover the number we intend to LOAD in. Sound difficult? All we need to do is enter a line such as:

```
1 REM 111111111 (etc.)
```

Make the REM about 246 bits long or longer. The easiest way to do this is to put 32 1's, then 32 2's, up to 32 7's. This is very easy as the video will look like:

```
  1 REM 11111111111111111111111
11111111122222222222222222222222
222222222  (etc.)
```

Now LOAD the new 'TEST' tape by:

```
LOAD "" CODE 23755
```

and the BASIC is revealed. N.B. If doing this with a long program, it would not be practical to make up a very long REM line without the aid of a 'REM create' program. Such a program has been included near the end of this booklet.

If you are trying this on a commercial cassette, and it only gives you a few lines of BASIC and ends with a statement such as 'RANDOMISE USR XXXXX' then the BASIC is most likely only used to load the rest of the program which is genuinely in machine-code. Or, if the BASIC revealed consists mainly of a giant REM statement holding meaningless characters, then the characters are actually machine-code for the program.



HINTS AND TIPS FOR THE SPECTRUM

To ensure that a program starts and runs in capital letter mode, put near the start of the program:

```
POKE 23658,8
```

If you want it to be in small letter mode, then use:

```
POKE 23658,Ø
```

To make the keyboard bleeper easier to hear in a program, put near the start of the program:

```
POKE 236Ø9,5Ø
```

To SCROLL rapidly through a listing, when the message SCROLL? appears on the bottom of the screen, press CAP SHIFT and key 3 and it will scroll two screens full.

To UNPLOT a point on the video you can use 'INVERSE 1' in a PLOT statement. To plot near the centre of the screen you use:

```
PLOT 8Ø,8Ø
```

To UNPLOT this you can use:

```
PLOT INVERSE 1 ; 8Ø,8Ø
```

The DRAW command can have a third value added to the end, and will draw through an 'arc'. Try this 'just for fun' program:

```
1Ø INK RND * 5: PLOT 8Ø,8Ø: DRAW OVER 1; 1ØØ,1Ø,8ØØ
2Ø CLS: INK RND*5: PLOT 8Ø,8Ø: DRAW OVER 1; 1ØØ,1Ø,7ØØ: RUN
```

To SCROLL on the Spectrum, include in your program a line with:

```
RANDOMISE USR 3582
```

Every time you GOTO this line the video will SCROLL up by one line. Try to make your lines a full 32 characters as this scroll tends to scroll-up the border colour as well if it's a short line. Multi-colour borders are possible on the Spectrum but only if the program is doing nothing else. Add this sub-routine to your programs and each time you GOSUB 9ØØØ it will produce multi-coloured borders for a short period:

```
9ØØØ FOR I = 1 TO 2ØØ
9Ø1Ø BORDER 1: BORDER 2: BORDER 3: BORDER 4
       BORDER 5: BORDER 6: BORDER 0: PAUSE 1
9Ø2Ø NEXT I: RETURN
```

The 'PAUSE 1' at the end of line 9Ø1Ø is essential for a steady video. Note that this FOR loop uses the letter 'I'. If 'I' is used in your program, then pick a different letter for this FOR loop.

BYTE SAVING PLUS SOPHISTICATED PROGRAMMING

Byte saving techniques can be very useful to the 16K Spectrum owner and very useful for the long business type programs. It should be noted that some byte saving methods, while saving bytes, may tend to RUN slower than normal methods. Using VAL or CODE does slow down a program and therefore should not be used in routines associated with the ZX-Printer (as this is slow enough). Don't take my word about which saves most bytes. Prove it by using the following program, which will show how the Spectrum holds the first line of a program in its memory area:

```
1Ø LET A = Ø
2Ø FOR I = 23755 TO 66666
3Ø PRINT PEEK I ; "." ;
4Ø IF PEEK (I+2) = 2Ø THEN GOTO 6Ø
5Ø NEXT I
6Ø PRINT : PRINT : LIST
```

Line 4Ø is just a crude method of making the program STOP after the first line is printed out. Line 3Ø prints each location with a full stop after it for clarity. RUN the program and the video will display:

```
Ø . 1Ø . 11 . Ø . 241 . 65 . 61 . 48 . 14 . Ø . Ø . Ø . Ø . Ø . 13
```

Line No.	Length of the data including end of line	Let	A	=	Ø	Indicates number next	Method Spectrum uses to hold numbers	End of Line

The line number is always held as two bytes and the start of the program is always address 23755.

The second two bytes are not displayed but tell the Spectrum the number of bytes in the line, including the end-of-line character which is a 13 (note that these two bytes do not include themselves or the two bytes used for the line in this count).

Look in the Spectrum manual on page 183 at the character set. Observe that 241 = LET, 65 = A, etc. The line is longer

than expected since the Spectrum holds the number in its 'floating-point' five digit method plus the 13 'end-of-line' indicator. If we change line 1Ø to be:

```
LET A = VAL "Ø"
```

and note that VAL is a keyword above key 'J' and stands for 'value'. Now RUN the program and, surprise, surprise, the line is actually shorter by three bytes! VAL, and then putting the number in inverted commas, can be used to replace any number in a program and will save three bytes each time it is used. In a 48K program, this could result in a total saving of several thousand bytes. You cannot use VAL on the line number itself.

An alternative to VAL is to use the keyword instead. Try it with the following examples:

```
LET A = CODE "H"
  LET A = 72
```

The CODE of 'H' is 72 so both lines are the same to the Spectrum. Using CODE instead of VAL is only an advantage if the original number was to be between 1ØØ and 255.... try it and see!

Another method of saving is to use variables. If your program was:

```
1Ø LET A = Ø : LET B = Ø : LET C = Ø
```

you could use instead:

```
1Ø LET A = Ø : LET B = A : LET C = A
```

This is a super saver as both 'B' and 'C' don't need VAL etc., and will not have a 'floating-point' number after them.

If in a program you are constantly using the same value over three times, it will save bytes if a variable is set to the value and you use it instead. For example, suppose your program includes the following in it:

```
PRINT AT 2Ø,6; "xcvcvcx"
FOR I = 1 TO 2Ø
PAUSE 2Ø
GOSUB 2Ø etc.
```

If you include near the start of the program, say, 'LET X = 2Ø' then whenever a '2Ø' is used in the program, use 'X' instead. The 'PAUSE 2Ø' would thus become 'PAUSE X'.

Ø is a special case. Instead of 'LET A = Ø' you could use the keyword 'PI' by using 'LET A = PI - PI'.

Instead of:

```
IF A = Ø THEN GOTO 1ØØØ
```

you could use the shorter line:

```
IF NOT A THEN GOTO 1ØØØ
```

(NOT is a keyword on the 'S' key). The Spectrum understands 'NOT A' as meaning 'if A is not a value', i.e. if A = Ø. You can also use:

```
IF A THEN GOTO 1ØØØ
```

instead of using 'IF A <> Ø' etc. The Spectrum sees the 'IF A' here as meaning 'if A is a value then take action'. Note that this method can only be used to check if a value is either a Ø or something else. Even a negative value is taken as being not Ø.

TRUE/FALSE BRACKETS

This next method is using the most powerful combined action line, which can also be used in most computers, although its action is very rarely explained. We'll try it in a demo program. Suppose we have a program to 'GOTO 1ØØ' if the input equals one, 'GOTO 2ØØ' if the input equals two, 'GOTO 3ØØ' if the input equals three. Enter this program:

```
 1Ø INPUT A
 2Ø IF NOT A THEN STOP
 3Ø IF A = 1 THEN GOTO 1ØØ
 4Ø IF A = 2 THEN GOTO 2ØØ
 5Ø IF A = 3 THEN GOTO 3ØØ
 6Ø GOTO 1Ø
1ØØ PRINT "WAS ONE" : GOTO 1Ø
2ØØ PRINT "WAS TWO" : GOTO 1Ø
3ØØ PRINT "WAS THREE" : GOTO 1Ø
```

RUN the program and one, two, or three to check if it works O.K. STOP the program and delete lines 3Ø, 4Ø, and 5Ø. Instead use:

```
3Ø GOTO (1ØØ AND A = 1) + (2ØØ AND A = 2) + (3ØØ AND A = 3)
```

Note that the AND is a keyword. Line 3Ø will now cause the program to GOTO whichever statement in brackets is true. This type of line does not necessarily save bytes. WARNING: Use this type of statement with care - if none of the conditions are true, then line 3Ø will be understood by the Spectrum as a 'GOTO Ø'. This multi-line could instead be:

```
3Ø PRINT ("WAS ONE" AND A = 1) + ("WAS TWO" AND A = 2)
        + ("WAS THREE" AND A = 3)
```

This obviously saves bytes as lines 1ØØ, 2ØØ, and 3ØØ are now not needed. If in such a PRINT line none of the brackets are true, then a blank line would be printed.

A variation on the above, which is used a lot in games and can save a really massive amount of bytes, is the multi-conditional line for moving things around the screen. Consider this program which simply moves an asterisk across the screen by using the '5' and '8' keys. The asterisk must stay between columns five and ten in our program. Enter it and try it:

```
1Ø LET X = 1Ø
2Ø LET Y = 5
3Ø PRINT AT X,Y ; "*"
4Ø PRINT AT X,Y ; " "
5Ø IF INKEY$ = "8" THEN LET Y = Y + 1
6Ø IF INKEY$ = "5" THEN LET Y = Y - 1
7Ø IF Y > 1Ø THEN LET Y = 1Ø
8Ø IF Y < 5 THEN LET Y = 5
9Ø GOTO 3Ø
```

Lines 5Ø and 6Ø decide if left or right; lines 7Ø and 8Ø set the limits of travel. Lines 5Ø, 6Ø, 7Ø, and 8Ø can be combined into one true/false action line as follows:

```
5Ø LET Y = Y + (INKEY$ = "8" AND Y < 1Ø)
       - (INKEY$ = "5" AND Y > 5)
```

The left half of this line makes Y = Y + 1 if the conditions are true; the right half makes Y = Y - 1 if this alternative condition is true. In other words, the bracket condition acts as a '1' if the condition is true - if it is false then the bracket condition acts as a 'Ø'. Now suppose that both conditions are false. The statement then appears to the Spectrum as 'LET Y = Y + (Ø) - (Ø)', which means that Y is unchanged. Very good! That's precisely what we require if no key is pressed. This conditional line in the above program, in the place of lines 5Ø, 6Ø, 7Ø, and 8Ø, saves a massive 56 bytes. In a real game, the same sort of thing would be used to affect 'X' so it could move the asterisk up and down as well. Thus the total bytes saved would be 112 by the two conditional lines used. Suppose we wanted to alter the 'Y' by two instead of one each time. Simply multiply the brackets, thus:

```
LET Y = Y + 2 * (INKEY$ = "8" AND Y < 1Ø)
      - 2 * (INKEY$ = "5" AND Y > 5)
```

There is no limit to the number of conditions in the brackets:

```
LET Y = Y + ((INKEY$ = "5" OR INKEY$ = "6") AND Y < 1Ø)
```

etc., is a valid statement if you want 'Y' to be '+1' when INKEY$ is '5' or '6', and provided that 'Y' is less than ten.

Even if you don't use the above yourself, hopefully you'll understand its action so you could alter it yourself if desired.

USING REM TO HOLD DATA AND/OR MACHINE-CODE

DATA statements are very wasteful of bytes as well as being very slow in operation. Most games printed in magazines use DATA statements to hold the UDG shapes and/or machine-code values. Each time the game is played there is a pause while waiting for the DATA line to be loaded into appropriate locations. This method explained below would greatly reduce the delay by a very noticeable amount of time. This example is only short, and the increase in speed not discernable. Do please try this exercise as it could be very useful for manipulating bytes in the Spectrum. Firstly we'll make a short program to put a 'man' into the user definable graphics 'A':

```
2Ø RESTORE : DATA 16,56,146,254,16,4Ø,68,13Ø
3Ø FOR I = Ø TO 7
4Ø READ A : POKE USR "A" + I , A
5Ø NEXT I
6Ø PRINT AT 1Ø , 16 ; "A"
```

N.B. This must be the graphics 'A' and not just the letter 'A'. To get this, get into 'G' mode, then press the 'A' key.

RUN the program, then STOP it. Add in the following lines:

```
1Ø REM 12345678
```

{ This is to put 8 somethings into REM to allow for the DATA statement.

```
1ØØ FOR I = 2376Ø TO 23767
11Ø POKE I, PEEK (USR "A" + I - 2376Ø)
12Ø NEXT I
```

RUN the program again. STOP, then delete all the lines except for lines 1Ø and 6Ø. Enter the following extra lines:

```
2Ø FOR I = 2376Ø TO 23767
3Ø POKE USR "A" + I - 2376Ø , PEEK I
4Ø NEXT I
```

Normally at this stage the program would be SAVEd. What we have achieved is two things. Firstly, a saving of 56 bytes, which would be substantially more in a real program. Secondly we would have a dramatic increase in starting-up time in a real program. Study the above, then try it on your next

'magazine' program.

DIRECTLY ENTERED STRINGS AND/OR VARIABLES

In some instances instead of using lines such as 'LET A$ = "HI THERE"' or 'LET A = 27', you could enter these as direct commands (i.e. without a line number). Provided the program auto-starts and does not have a CLEAR in it, then the Spectrum will know that each time you say 'PRINT A$' that 'A$' is still 'HI THERE'. Obviously saves bytes as a line defining this is not in the program. Can only be used with care as there are lots of instances whereby such directly entered variables and/or strings could be deleted or changed.

MULTI-STATEMENTS

The Spectrum allows you to use multi-statement lines with actions separated by the ':'. This saves bytes, but long multi lines are difficult to read and/or EDIT. Do not put a GOTO in the middle of a multi-statement line and also remember that if an IF statement condition is not met, the entire line will not continue to the next section.

MULTI-PRINT

AT can be used several times in a single line, thus:

```
1Ø PRINT AT 5 , 1Ø ; "THIS"
2Ø PRINT AT 7 , 1Ø ; "IS"
3Ø PRINT AT 9 , 1Ø ; "GREAT"
```

could all be in a single line, which would save bytes, thus:

```
1Ø PRINT AT 5 , 1Ø ; "THIS" ; AT 7 , 1Ø ; "IS" ;
                   AT 9 , 1Ø ; "GREAT"
```

RIDICULOUS BUT USEFUL TIP

Due to the unreliability of the printer to move the paper up.... if you have this problem, instead of entering the direct command COPY, use instead:

```
PAUSE 15Ø : COPY
```

This will give you time to grab the paper ready to pull it through the printer!

QUICK PROGRAM TO USE SPECTRUM AS XYLOPHONE

```
1Ø LET A = CODE (INKEY$) - 48
2Ø IF A > 8 OR A < 1 THEN GOTO 1Ø
3Ø BEEP .17 , A + 15 : GOTO 1Ø
```

The sequence for 'London Bridge' is: 5654 345 234 345 5654 345 2531. Repeat until fingers fall off!

USEFUL UTILITY PROGRAMS (1)

REM CREATE: This is very useful as mentioned previously for the 'disguised' BASIC programs. It is useful anytime a long REM is required. The action is to combine the bytes in line one with those of line two. The easiest way to use it is to EDIT line one to be the new line two, then RUN. Each time you repeat this, the length of line one doubles. If you particularly want a certain length REM then note that if you make line two:

```
2 REM ....        (just four things after the keyword REM)
```

then when you RUN, it will add ten to the existing length of line one. When the length is as desired, enter the direct command:

```
GOTO 1ØØØ
```

then wait until the O.K. message appears on the bottom of the screen. The routine in 1ØØØ will zero out all the bytes in the REM (when used for BASIC programs disguised as bytes, it isn't necessary to zero REM). When all done, all except line one can be deleted. N.B. Line nine and line 999 are not essential but I suggest these are left in so that if the program is used at a later date you'll know how to use it.

```
    1 REM ..........................................
    2 REM ..........................................
    9 REM  EDIT 1 TO 2 THEN RE-RUN  ← (this line in INVERSE)
   1Ø LET A = PEEK 23757 + 256 * PEEK 23758
   2Ø LET B = A + 4
   3Ø LET C = 23757 + B
   4Ø LET D = PEEK C + 256 * PEEK (C + 1) + 4
   5Ø LET E = A + D
   6Ø POKE 23758, INT (E / 256)
   7Ø POKE 23757, E - PEEK 23758 * 256
   8Ø POKE 23754 + B, 128
   9Ø PRINT AT 10,5 ; E - 2 ; "BYTES IN LINE 1"
  1ØØ PAUSE 15Ø : CLS : LIST 1 : STOP
  999 REM  NEXT LINE ZEROES REM     ← (this line in INVERSE)
 1ØØØ FOR I=2376Ø TO 65367:IF PEEK I<>13 THEN POKE I,Ø:NEXT I
```

N.B. When the REM gets very long, the error buzzer sounds - ignore it. Do SAVE this on tape - it is a very useful program.

USEFUL UTILITY PROGRAMS (2)

This program, when called, will print the length of a program, the number of variables and the free memory space. Apart from its obvious use, it is useful while developing programming techniques to help you determine which is the shortest programming method. The program is intentionally all in one line, line 9999, so that it can sit above your program safely. To use at any time, enter:

```
GOTO 9999
```

The program itself occupies 219 bytes. Sorry it is difficult to read due to the VAL statements, but the program needs to be as short as possible - enter it carefully and SAVE on tape before use.

```
9999 CLS : PRINT "PROGRAM..." ; PEEK VAL "23627" + VAL
     "256" * PEEK VAL "23628" - ( PEEK VAL "23635" +
     VAL "256" * PEEK VAL "23636" ) ' "VARIABLES." ;
     PEEK VAL "23641" + VAL "256" * PEEK VAL "23642" -
     ( PEEK VAL "23627" + VAL "256" * PEEK VAL "23628"
     ) - VAL "1" ' "FREE AREA." ; PEEK VAL "2373Ø" +
     VAL "256" * PEEK VAL "23731" - ( PEEK VAL "23641"
     + VAL "256" * PEEK VAL "23642" )
```

N.B. The ' is the apostrophe which is on key 7. The "." in the above represents one space. Merge the above onto your program, then 'GOTO 9999': or RUN the program in an empty machine by 'RUN 9999'.

The program prints:

In a 48K machine -	PROGRAM 219	In a 16K -	PROGRAM 219
	VARIABLES Ø		VARIABLES Ø
	FREE AREA 41392		FREE AREA 8624

TIP In programs using the first line as line Ø. To EDIT such a line you can give it a number by entering:

```
POKE 23756,1
```

To make the first line number a zero:

```
POKE 23756,Ø
```

MICRODRIVE SUPPLEMENT

The major problem with the microdrive is that the vast majority of the existing software cannot be transferred to it without, in most cases, major program changes. Add to this the fact that the interface unit is bolted securely to the Spectrum case (Sinclair has no faith in the ability of his edge connectors!). This means you cannot simply remove the microdrive unit. There does not exist a command to truly make the Spectrum behave as if the microdrive is not present, but, if you enter:

RANDOMIZE USR Ø

you can then LOAD tapes which will only RUN without the microdrive (some Spectrums appear to be a bit deaf, and it is best to enter this command at least three times!!!).

Tapes designed to work with the microdrive will also give problems if they are LOADed when the Spectrum thinks the microdrive is not present. Part way through the program the Spectrum may say "yes it is".
Solution: If you do want a microdrive present, then issue a microdrive command before LOADing. The simplest method is to enter:

CLS#

Note that if you ever enter NEW, then the Spectrum forgets the microdrive is present. As an example, if using the 'ØØ7 SPY' cassette, get into the habit of instead of LOADing by 'LOAD""' use instead 'CLS# : LOAD""'. If loading, for example, a long game, then first do a 'RANDOMIZE USR Ø'. Do it three times, then LOAD your game.

Warning: Adverts which state 'microdrive compatible' only mean that the tape will LOAD and RUN if a microdrive is fitted or not. It does not mean that the program can be copied to cartridge. In the case of KopyKat adverts, it does not mean that they can copy tapes to cartridge.... if in doubt, write and ask them first. Such ads are not intended to mislead, but regrettably they do.

MELBOURNE HOUSE and the SUNSHINE GROUP (which owns PCW) both supply excellent books on the microdrive and there is little point in this booklet covering the in-depth sections they do.

The microdrive uses an extra 58 bytes for additional system variables and this results in your user memory area being moved up from its original start location of 23755 to be 23813 instead. This is disastrous for machine-code programs which have either been written to start low down in memory and/or are in a REM statement. The 'RANDOMIZE USR XXXXX' statements (and the 'CALLS' in the machine-coding) will be out by 58

bytes at least. In some programs simply adding 58 to all RANDOMIZE statements may solve the problem. It is not just the extra 58 bytes which we have to beware of. When the Spectrum is doing a microdrive instruction it creates a buffer of approximately 593 bytes. Little information about the location of this buffer can be gleaned from the various microdrive books, but it appears that it possibly moves the BASIC while using the buffer, but puts it back correctly after the command is done. If there is machine-code near the start of the user area, it will get lost or corrupted (don't write saying it must use the workspace - if it did it would not affect only machine-code written in low memory!!!).

Programs which LOAD into the top of memory will very rarely work with a microdrive, and usually give the message 'HOOK CODE ERROR' and/or lock up.

CANNOT TRANSFER TO MICRODRIVE ANY OF THE FOLLOWING

MACHINE-CODE CALLING NEXT SECTION: This includes headerless files which are always called off tape by machine-code. Obviously, the microdrive could not do this. Sorry, but most chess programs are of this type (the original PSION chess is normal machine-code).
EXTREMELY LONG PROGRAMS: To be on the safe side, you need to allow 58 bytes for extra system variables, 593 bytes for the buffer and about 100 bytes to ensure you're not too close to RAMTOP!
MACHINE-CODE LOW IN MEMORY: In general, if the machine-code is below about 245ØØ in user memory, it'll give problems.

ARE YOU IN MICRODRIVE MODE?

The simplest way to check is to remember that each line number in a program uses two bytes. If the first line in your program is number 1Ø, then the first two bytes of user memory hold Ø and 1Ø. Therefore PEEK at byte two in user memory. If there's no program in your Spectrum, enter this one line:

```
1Ø LET A = Ø
```

Now, as a direct command, enter:

```
PRINT PEEK 23814
```

If it PRINTs '1Ø' on the video, then you're in microdrive mode. If instead you find that the 'old' PEEK, that is:

```
PRINT PEEK 23756
```

gives you an answer of '10', then you are not in microdrive mode (if not in microdrive mode, enter 'CLS#' then re-try).

EXAMPLE PROGRAMS - MICRODRIVE MODIFICATIONS

The REM CREATE program, if you wish to RUN it on a microdrive Spectrum, must have all the values which begin with 237xx lifted by 58. In the LENGTH/VARIABLES/FREE AREA program, alter only the addresses which begin with 237xx. The other addresses are system variables and are not changed if the microdrive is there or not.

All the other 'test' programs in this booklet will remain correct provided that you enter 'RANDOMIZE USR Ø' before entering them.

COPYING TAPES TO MICRODRIVE

ZX-GUARANTEED (or in Australasia, Manukau Computers (N.Z.) Ltd) can supply a cassette which can copy up to 4Ø5ØØ bytes of machine-code to cartridge. An alternative is to use a header reader to get the relevant information from your tape (the ZXG cassette also contains a header reader). Why only machine-code? In most programs, the first section is a short BASIC program containing commands such as 'LOAD "" CODE' to 'call' the machine-code from off the tape. This means you must first be able to STOP the BASIC so you can change the command to 'call' the machine-code off the microdrive instead. In other words, instead of saying 'LOAD ""', enter:

MERGE ""

(MERGE is the keyword on button 'T'). In virtually all BASIC programs on tapes, this will LOAD but not RUN the BASIC (obviously ensure your Spectrum is empty before this). Now LIST the program and change all the 'LOAD "" CODE' commands to:

LOAD * "m" ; 1 ; "name" CODE S

where 'S' is the 'CODE START' the header reader give you. It is not really necessary to put this 'S' value at the end, but in some cases we'll have to re-locate a program in order to copy it to microdrive, and by quoting the start value, the Spectrum will ensure programs LOAD into the correct place.

Play the machine-code programs into the header reader to get the start and lengths of them. If they are in user memory area (which in practice usually means they are at a start address of around 245ØØ upwards), we can simply LOAD them into the

Spectrum by 'LOAD "" CODE', and then SAVE them on microdrive with:

SAVE * "m" ; 1 ; "name" CODE S , L

where 'S' equals start and 'L' equals length of the code (information gleaned from the header reader). If the code start was within the system variable area (23552 to 23733), then we have to re-locate by instead of initially saying 'LOAD "" CODE', instead use:

LOAD "" CODE 3ØØØØ

This will safely LOAD the code into 3ØØØØ. We must SAVE to the microdrive from this 'wrong' area by:

SAVE * "m" ; 1 ; "name" CODE 3ØØØØ , L

with 'L' being the length. It does not matter that we've SAVEd onto the microdrive in the wrong area as we've put the correct code start into our LOAD command and this will 'drag' the code back to its correct point. If you want to miss out the 'screen' from a program you must delete the 'LOAD "" CODE' which originally LOADed it in BASIC. If you want to SAVE the 'screen' to your microdrive, then first re-locate it to, say, 3ØØØØ, or else you'll lose the bottom two lines of your picture when you type in your command.

The best way to find out if a program can be transferred to a microdrive is to try transferring it as above, then check if it RUNs O.K.!!!



NOTES

NOTES

Copyright © G.A.Bobker, 1984

ZX GUARANTEED

29 Chadderton Drive
Unsworth
Bury
Lancashire

Distributed in Australasia by

MANUKAU COMPUTERS (N.Z.) LTD.

P.O.Box 76-310
Manukau City
New Zealand