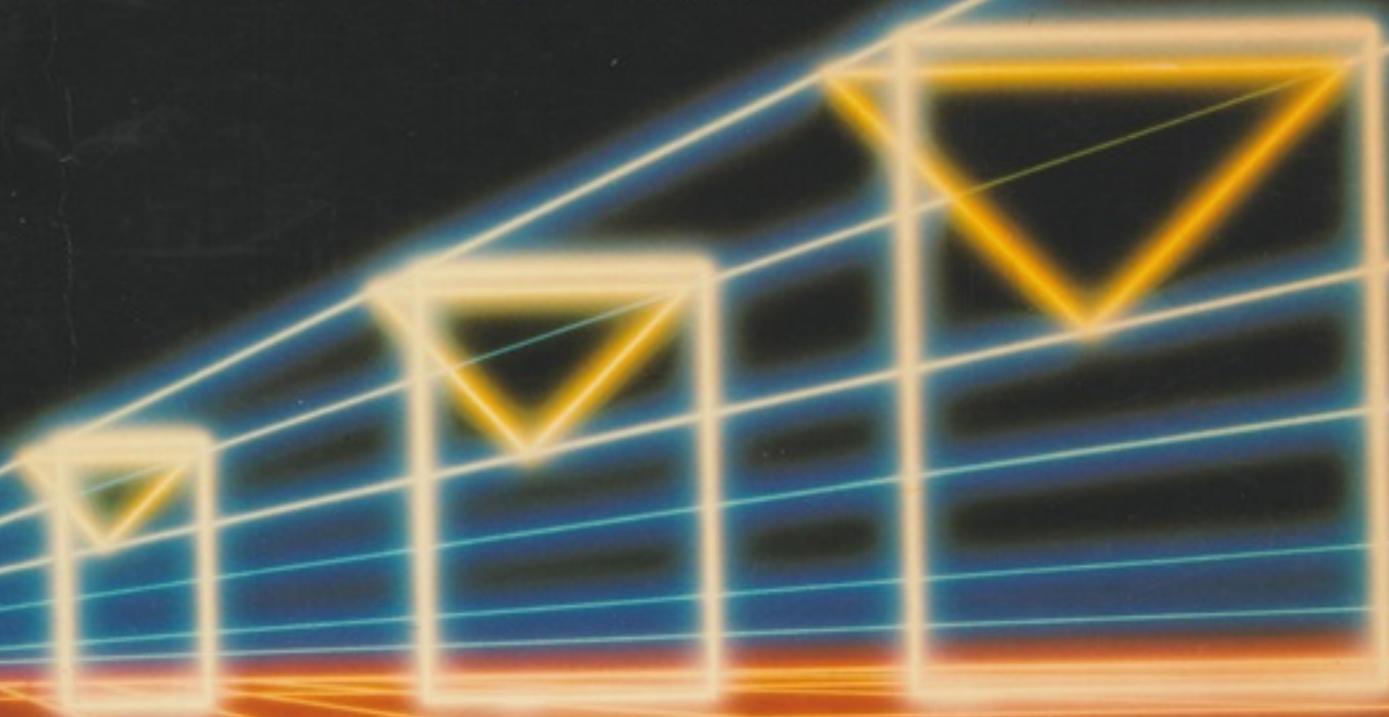


# THE SPECTRUM OPERATING SYSTEM

Steve Kramer



MICRO DEFS

1500

**The Spectrum  
Operating System**

# **The Spectrum Operating System**

Steve Kramer

 **MICRO PRESS**

First Published 1984 by  
Micro Press  
Castle House, 27 London Road  
Tunbridge Wells, Kent

© Steve Kramer 1984

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, recording or otherwise, for commercial gain without the prior permission of the publishers.

British Library Cataloguing in Publication Data  
Kramer, Steve

The Spectrum operating system.

1. Sinclair ZX Spectrum (Computer)—Programming

2. Operating systems (Computers)

I. Title

001.64'25 QA76.8.S625

ISBN 0-7447-0019-1

Typeset by Keyset Composition, Colchester  
Printed by MacKays of Chatham Ltd

# Contents

<i>Chapter 1</i>	<b>Getting Started</b>	1
<i>Chapter 2</i>	<b>Useful Call Addresses and How to Use Them: the 16K ROM</b>	3
<i>Chapter 3</i>	<b>The 8K Interface ROM</b>	25
<i>Chapter 4</i>	<b>The System Variables</b>	42
<i>Chapter 5</i>	<b>Input and Output Ports and Channels</b>	62
<i>Chapter 6</i>	<b>Using the Interrupts</b>	70
<i>Chapter 7</i>	<b>Extending BASIC with Interface 1</b>	75
<i>Chapter 8</i>	<b>The Calculator</b>	80
<i>Appendix A</i>	<b>Hex to Decimal Conversions</b>	93
<i>Appendix B</i>	<b>The Spectrum Memory Map</b>	95
<i>Appendix C</i>	<b>The Spectrum Screen Map</b>	96
<i>Appendix D</i>	<b>The Keyboard Map</b>	98
<i>Appendix E</i>	<b>The Spectrum Character Set</b>	99
<i>Appendix F</i>	<b>ROM Interrupt Vectors</b>	102
<i>Appendix G</i>	<b>Useful Subroutines</b>	104
<i>Index</i>		129

# Getting Started

While this book is primarily aimed at those people who have a reasonable understanding of assembly language programming, there will be a considerable amount of information which will be of use to the uninitiated programmer who would like to be able to gain access to the versatility of machine code, without having to learn how to write it. If you belong to the latter group I hope that this book will whet your appetite, and prompt you to start learning a little about assembly language programming. With the minimum amount of knowledge and using the routines in the following chapters, short machine code routines can be written and used very easily.

For anybody not familiar with the operating system of the Spectrum or who is basically bone idle (like me) and who prefers to use ROM routines and existing facilities — rather than re-inventing the wheel every time they want to write a program— I will be discussing how to use ROM calls in both the 16K ROM, which is in the Spectrum, and in the 8K ROM in the Microdrive Interface. In some cases, I shall give examples as well as descriptions of how to use them, except where they are just a simple CALL.

I shall also look at the system variables and show how to use them to advantage, and explain the use of interrupt-driven routines, which can allow sprites to be used on the standard Spectrum.

Unless you wish to convert the assembly language routines by hand into numbers and then poke them into your computer's memory, a prerequisite to being able to use this book is an Assembler program. I can recommend the Picturesque Editor Assembler and also their Monitor/Disassembler programs, which are both very easy to use and fast. I can also recommend Highsoft's Devpack 3 which will be even better with a few improvements which will probably have been made by the time

you read this chapter. For the more experienced user, the Devpack (which contains both Monitor/Disassembler and Editor/Assembler) is, in my opinion, at present unsurpassed. The latest versions of both these programs are Microdrive-compatible.

It is not my intention to teach assembly language programming as there are a great many books already available which cover this in detail. Two that I can recommend are Rodney Zaks' *Programming the Z80* and *Spectrum Machine Language for the Complete Beginner*. The first is not related to the Spectrum, but goes into far more detail than the second and also has concise descriptions of all the available op-codes.

To call a machine code program from BASIC you can either use a RANDOMIZE USR NN, a PRINT USR NN or a LET V = USR NN command, where NN is the entry point to the machine code program and V can be any numeric variable. On return from the machine code routine, the variable in the LET command will be equal to the BC register pair on exit, and with the PRINT command the value in the BC register pair will be printed to the current stream. In all cases the BC register pair will contain NN on entry to the machine code program.

Whenever a machine code program is called, it is advisable to save the contents of the H'L' register pair since the contents of this are necessary for a successful return to BASIC. The IY register pair should not normally be used as the ROM uses this to index the system variables, but it can be used so long as the interrupts are off and no ROM routines are used before it is restored to holding 23610 (5C3Ah).

This book is intended to be a reference work which you will turn to for information for as long as you have your Spectrum. I have made no attempt in the main text to be chatty or to hold your hand, but I hope the information required is there. Obviously it would be impossible to detail all the ROM routines and all the ways they could be used, so I have selected those that I am asked about most often and find most useful. Further reading for the serious programmer must be Dr Ian Logan's *Complete Spectrum ROM Disassembly*.

## Useful Call Addresses and How to Use Them: the 16K ROM

It is important before using ROM routines always to save the H' and L' registers, and restore them afterwards before a return to BASIC is made. Also the IY register must always contain the address of the system variable ERR NR 23610 (5C3Ah) whenever a ROM routine is used.

### Printing: RST 16 (10h)

The character whose code is in the A register will be printed to whichever stream is currently open. This can also be used to 'print' control codes (i.e. TAB, INK, OVER, etc.; see the Spectrum manual for details).

### Opening and Closing Streams (for RST 16 (10h)): CALL 5633 (1601h)

CALL 5633 (1601h) sets the output for RST 16 (10h) to the stream held in the A register when called. Normally A = 2 will print to the main screen, A = 3 will print to the printer and A = 1 or 0 will print to the lower screen. With the Interface 1 connected, other streams can be used for output to the Microdrives, network or other devices. Details are given on how to divert streams for your own purposes (e.g., for controlling an interface for other uses such as the Kempston or other Centronics printer interface)

in the section on **Expanding Tokens for Output** later in this chapter.

### Detecting if Break is Being Pressed: CALL 8020 (1F54h)

This call will return with the carry flag set if it is not being pressed, and with the carry flag reset if it is being pressed. (*Note:* This call tests for both CAPS SHIFT and SPACE being pressed.) If you want to test for SPACE alone see the section later on **Getting a Character from the Keyboard** for full details but, for the time being, you could use:

```
LD A,7FH
IN A,(0FEH)
RRA
JP NC,PRESSED
```

### Setting the Position for Printing Using RST 16 (10h): CALL 3545 (DD9h)

This routine requires the B register to hold the screen line number in the form:

B = 24 - line number

(i.e. if B = 24, this is the top line of the screen; if B = 1 this is the bottom line). Unfortunately due to an oversight in the writing of the ROM, you cannot use lines 23 and 24 for printing on the main screen. So you must use the lower section by setting up the output stream for RST 16 (10h) to be 1, and using the top two lines of the lower screen.

C = 33 - column number

(i.e. if C = 33 this is the left-most column; if C = 2 is the right-most).

This call automatically updates the system variables for the print positions on whichever stream you are using, as set up by

calling 5633 (1601h). Caution needs to be exercised on the last line of the main screen as a scroll message will be generated after printing on the last available position. This will cause a return to BASIC if answered with 'n' or SPACE. In addition to this, any attempt to print to stream 1 will cause scrolling up of the lower screen when the available space set up by the system variable DF SZ (23659) has been filled. This can produce some very unexpected results.

Obviously when printing to a printer you cannot set a line number so the B register is not used.

### Clearing the Screen (Whole Screen): CALL 3435 (D6Bh)

This call will clear the whole screen and reset the attributes to those in system variable ATTR P for the main screen and BORDCR for the lower screen (23693 and 23624, respectively; see the Spectrum manual for how these are made up).

### Clearing the Screen (Lower Screen): CALL 3438 (D6Eh)

This will clear the lower screen only and reset the attributes.

*Note:* Both 3435 (D6Bh) and 3438 (D6Eh) reset DF SZ to 2 and can corrupt the current channel used by RST 16 (10h) so this will need to be reset. The current print positions are set to the top left of the respective screens.

### Scrolling the Screen: CALL 3582 (DFEh)

This scrolls the screen up by one line, but leaves the current print position unaffected. Therefore if you continually print to the same line, scrolling the screen at the end of each line, you will have an effect like typing using a typewriter (with the printing taking place at the bottom and the print being moved up after a carriage return).

## Plotting to the Screen: CALL 8933 (22E5h)

The point addressed by the B register (Y 0-175) and the C register (X 0-255) will be plotted to the screen. The INK and PAPER colours can be set up previously by loading the system variables for the attributes with the colours you require or the current attributes will be used. OVER 1 can be set by SET 0,(IY + 87) or reset by RES 0,(IY + 87) and any point can be unplotted by SET 2,(IY + 87).

*Note:* It is perfectly possible to plot to the lower screen and this should be avoided, if not required, by setting the stream required with the 5633 (1601h) CALL.

## Getting a Number to a Stream

This is the most complicated routine so far and does not have a simple call address as such. There is a routine in the ROM for taking a 16-bit number from two addresses and outputting it in decimal form, but this routine has the disadvantage that the bytes must be stored 'about face' from the way that the Z80 stores them. This means that you must load the number into a register pair and then store it back into a space in memory the opposite way round (i.e. high-order byte first). A short program such as this would serve:

```
LD DE,(address where No. stored)
LD HL,SPARE WORD
LD (HL),D
INC HL
LD (HL),E
```

The ROM routine at 6696 (1A28h) can now be called with HL pointing to the address SPARE WORD which will then be output in ASCII to the current stream, but there are two drawbacks.

- 1) It will only be output correctly if it is less than 10 000 decimal and it must be an integer.
- 2) The number will be output with leading spaces so that chaining outputs to create a larger number will work only if

each number output is between 1000 and 9999, otherwise spaces will be introduced.

Those of you who have the *Spectrum Pocket Book* and have read about this call address, but not used it, will now be calling me an idiot. In that book, the author points out that the E register controls the way the number is formatted, but if you have used the routine you will know that this is not quite the case, as the E register is ignored. There are two ways round this problem.

Firstly, you could write your own routine to bypass the first part of the ROM routine, such as:

```
PUSH DE
LD D,(HL)
INC HL
LD E,(HL)
PUSH HL
EX DE,HL
LD E,20H
JP 1A30H
```

*Note:* The last instruction must be a jump (JP) and the routine you have written must be called, otherwise the stack will be mixed up as the return address is stored on the stack and the routine POPs from the stack the PUSHes you have made. You will see that the E register is loaded in your routine and as this is a duplicate of the start of the routine on ROM you can now see why the *Spectrum Pocket Book* is wrong. Now you can change the LD E, instruction in this routine to be either 48 decimal, in which case leading 0s are printed, or 255 when anything not significant is not printed.

"But wait a minute," you will be saying, "now I can see why the byte order has to be changed." Quite correct, if you reverse the order of loading the DE register pair or even load HL directly the number does not have to be reversed, so it is well worth while rewriting this short piece of code for incorporation into your own program as a subroutine for outputting numbers.

There is one other part to this routine in the ROM worth considering at this point and that is the part starting at 1A1Bh. This simply takes the number in the BC register pair and outputs that with no leading 0s or spaces; *much* more useful in many cases.

With a little thought, these routines now give you the possibility of outputting any size of number from your program. It is probably easier, however, to use the routines which take a value from the calculator stack (for details see Chapter 8).

### Getting a Character from the Keyboard

The RST 56 (38h) instruction is used by the Spectrum to scan the keyboard and update the FRAMES system variables, and it is called by the maskable interrupt routine. If all that is required is to see if a key has been pressed, bit 5 of FLAGS 23611 (5C3Bh) can be tested; if the bit is set a key has been pressed since it was last reset. Note that bit 5 needs to be explicitly reset.

The code of the last key pressed can be found from 23560 (5C08h) LAST K, detailed in Chapter 4. This is often sufficient for getting input from the keyboard, but it has disadvantages.

Firstly, it is updated only 50 times a second (60 if the mains frequency is 60 Hz, as in the USA) so you cannot do a RES 5 instruction immediately followed by a look at the bit since almost certainly, even if a key is being pressed, the keyboard will not have been scanned by an interrupt. Otherwise, if the interrupts are off, the keyboard will never be scanned unless you do so. Therefore something along the lines of:

```
LD HL,23611 ;FLAGS system Variable
RES 5,(HL)
LD A,FFH
LD (23552),A ;23552 is part of the
               system variable KSTATE

PUSH HL
RST 56
POP HL
XOR A
BIT 5,(HL)
JP Z,NO KEY
LD A,(23560) ;LAST K system variable
NO KEY ( whatever )
```

will return in the A register the code of the key being pressed at the label NO KEY or 0 if no key is being pressed. In the above routine, the LD HL,23611 instruction could have been omitted

and (IY + 1) tested instead, as the IY register contains the address of the system variable ERR NR (23610) (5C3Ah) and is used for addressing the system variables by the ROM. This is why if you use the IY register in your programs you *must* make sure the interrupts are off or directed to your own routines, and you must also reset the IY register to the correct address before using the ROM or reverting to the normal interrupt mode.

The second problem is that you cannot test for more than one key being pressed at a time. This is because if a combination of keys which is not a valid character is being pressed the Spectrum will ignore them. To overcome this you will need to write your own keyboard-scanning routine. Firstly, if you only want to find out if a key is being pressed without worrying about which key or how many keys, you could do something like this:

```
XOR A ;make sure that A contains 0
               to scan all keys
IN A,(FEH) ;FEH is input port from keyboard etc.
LD D,31
AND D ;Get rid of the etc.(the 3 high bits)
XOR D ;Flip the remaining bits
JR Z,NO_KEY
```

The keyboard can be scanned properly to discover which keys are being pressed by loading the A register with the value of the lines to be scanned prior to the IN A,(FEH) instruction.

Chapter 23 of the Spectrum manual explains how the keyboard is laid out, and it is also shown in Appendix D in this book, but in brief this is the layout. The first character in each case is BIT 0 and the second BIT 4, the hex number is to be loaded into the A register to set the bit for the line being scanned low:

CAPS SHIFT—V	FEH
A—G	FDH
Q—T	FBH
1—5	F7H
0—6	EFH
P—Y	DFH
ENTER—H	BFH
SPACE—B	7FH

So, for example, to read the keyboard for ENTER being pressed the program would be:

```
LD  A,0BFH
IN  A,(0FEH)
AND 1
JR  Z,ENTER_PRESSED
```

To see if more than one key is being pressed you could use the logical operators AND, OR, etc. if they are on the same line or bit test instructions. See the next CALL 5598 (15DEh) for getting normal characters easily.

### Wait Input: CALL 5598 (15DEh)

This is a most useful routine which allows for input from any stream that has an input address. Before use the stream from which the input is required must be opened by the routine at 5633, described earlier. When the Wait Input routine is CALLED, it in turn calls the input routine of the current channel. On return from this the carry flag is checked, if set the main CALLing program is returned to. If the carry flag was not set the 0 flag is then checked, and if set the process is repeated.

The routine is used by the Spectrum as the control for the current input subroutine, normally the keyboard input, but if CURCHL is set to point to a channel with the input address pointing to your input subroutine this will be used. (How to do this is explained in Chapter 5.) When used with stream 1 (the keyboard and lower screen), the routine will wait for a key to be pressed and return its code in the A register. This use is demonstrated in the DeBASE program in the Appendix G.

There is, however, a problem when using this to read the keyboard if the mode is changed. Each time the routine is called, TV FLAG 23612 (5C3Ch) IY + 2 is checked and if bit 3 is set the input buffer is copied to the edit area of the screen. This can be overcome by using the Key Input routine at 4264 (10A8h) directly and not via the Wait Input routine, as shown in the DeBASE program in Appendix G at the label INPUT. The program duplicates the Wait Input routine but ensures the MODE CHANGE flag is always reset. The Wait Input routine is used at the label

INPUTF and if the program is entered (or just this section of it) the problem can be demonstrated by pressing CAPS SHIFT and SYMBOL SHIFT to go into EXTENDED MODE. The last line that was typed in from the keyboard will then appear at the bottom of the screen.

When using these routines to read from the keyboard the interrupts must be on and the normal interrupt routine at 56 (38h) called within the interrupt cycle, otherwise no input will ever be received.

### Screen Copy to Printer: CALL 3756 (EACH)

This routine requires no presetting and a straightforward CALL will output a screen copy to the ZX printer.

### Printing Graphics to Printer: CALL 3789 (ECDh)

This is similar to the screen copy routine in that it uses the printer buffer and outputs its contents to the printer. It is used by the RST 16 routine which normally treats the buffer as one screen line (eight pixels high). If you place your graphics into the buffer one line at a time and then make a CALL to this address, the printer will copy the buffer to the printer.

*Note:* The layout is 32 bytes per pixel line with the next pixel line following immediately and not as the screen. The buffer is also cleared to 0s after a CALL.

### Clearing the Printer Buffer: CALL 3807 (EDFh)

This simply clears the buffer to 0s.

### Using the Beep: CALL 949 (3B5h)

The DE register pair holds the length of the output and the HL register pair the frequency. 0 is high for frequency and FFFFh is low. The problem with this routine is that the duration is

frequency-dependent, hence if you halve the frequency you double the duration for a given value in DE. The actual figures are calculated as follows:

$$\begin{aligned} HL &= (437\ 500/\text{frequency}) - 30.125 \\ DE &= \text{duration} * \text{frequency} \\ &(\text{duration should be in seconds}) \end{aligned}$$

The reason why the 30.125 is taken away from the HL calculation is that the routine itself takes 120.5 T states actually to generate the note and amend its own registers, etc.

Middle C is approximately 261 Hz so the value for HL would be about 1646 decimal and DE for one second would be about 261 decimal.

Remember that the interrupts from the ULA will occur 50 times a second (60 times in the USA) and they will corrupt the sound output if the routine is in the bottom 16K of RAM.

### Printing Messages: CALL 3082 (COAh)

On entry the DE register pair must hold the address of the message table's start marker, which must have BIT 7 set, and the A register must hold the entry number in a table of messages, each with BIT 7 set on the last byte. The first message has entry number 0.

If you wanted to print the message " I am " you might have a line in your program such as: MESSAGE DEFM " I am ", but this would carry on printing after the last space until it came to a byte with bit 7 set. You should therefore change the program line to 'MESSAGE DEFB 80H : DEFM " I am " : DEFB " " + 80H'. The routine will subtract the 80H and print the last space correctly, but will know that it has reached the end and return to the calling routine. So, the whole program might look something like this:

```
LD  A,0      ;(to print first message in table)
CALL FR_MES
...
;  REST OF PROGRAM GOES HERE
...
```

```
FR_MES LD  DE,MESSAGE
CALL 3082
RET
MESSAGE DEFD 80
DEFM "ENTRY"
DEFB " "+80H
DEFM "ENTRY 2"
DEFB " "+80H
```

This would print to the current stream at the current position the message <ENTRY>. To print <ENTRY2>, the A register would hold 1. The chevrons are only to show the limits of what will actually be printed, and will not appear when a message is printed. This is the routine used by the ROM to expand tokens and generate error messages when called from another routine at 2898 (B52h). The DeBASE program (see Appendix G) makes extensive use of this routine.

### Expanding Tokens for Output: CALL 2898 (B52h)

Whenever the Spectrum comes across a token code (anything with bit 7 set) it has to decide what to do with it, as it could either be a User-Defined Graphic, a block graphic or a word from the BASIC vocabulary. Normally this is taken care of automatically when using RST 16 (10h) (which is directed through this routine). If you have changed the address for output on a stream (e.g. to output to your own printer interface), whenever your routine is being used the A register will hold the code for the token and if you wish to expand it you will have to do it yourself. If you leave it as it stands you may get some peculiar results from the printer. This routine can be used to expand the keywords, but you will have to deal with the graphics yourself and make sure that no graphic codes are sent to the routine, otherwise the machine may lock up.

*Note:* When expanding, the routine itself makes repeated calls to the address pointed to by the stream in use and returns only after all the letters have been output. This means you will most likely jump to the routine so as to ensure the code in the A register on the final return is not output. This is shown in the following typical program for output to a printer interface:

```

INIT  LD  HL,(23631) ; CHANS (address of channel
      LD  BC,15      ; offset for stream 3
      ADD HL,BC      ; HL now points to the
                    ; location holding the
                    ; address to be called with
                    ; output on this stream
      LD  BC,START   ; START = address of the
                    ; start of your O/P routine
      LD  (HL),C
      INC HL
      LD  (HL),B
      RET           ; stream three is now
                    ; initialised to send its
                    ; output to your routine

START LD  B,A        ; save the ASCII code in B
      CP  165
      JP  NC,B52H    ; expansion routine in ROM
      CP  13
      JR  Z,CRLF     ; carriage return & line
                    ; feed
      CP  32
      RET C         ; anything below 32 is an
                    ; unprintable control code
      CP  128
      JR  C,PRINT   ; must be normal character
                    ; anything left here is a
                    ; graphic, and you must
                    ; deal with it as you wish
                    ; your O/P routine comes
                    ; here
PRINT RET          ; go back for next
                    ; character, if there
                    ; isn't one execution will
                    ; return to the original
                    ; calling program

```

This program is a very basic one; if you wish to send any printer control codes, it will not print them. To get round this you will have to carry out further checks and act accordingly. The first section only needs to be called once as it permanently alters where output to stream 3 is sent. The only times it needs re-initialising are either after a NEW command from the keyboard or if the output has been changed by another part of the program.

### Expanding Block Graphics: CALL 2878 (B3Eh)

If you want to create from the code of a block graphic the block graphic itself, the routine at 2878 (B3Eh) will do it for you. On entry the base address of eight spare bytes where you want the graphic constructed should be in the HL register pair, and the B register should contain the code of the block graphic. Two CALLS need to be made, the second immediately after the first, as each CALL constructs four bytes of the character. The first byte of your eight-byte block will be the top of the block graphic and HL will point to the byte after the last byte of the graphic. The register pairs altered by this routine are AF, HL and BC; no others are used by it.

### Drawing Circles: CALL 9005 (232Dh)

The routine to draw a circle requires the parameters of the circle to be on the calculator stack of the Spectrum. Thus the first thing to do is put the details onto the calculator stack. A routine located at 11560 (2D28h) will do this for us if we put the number we wish to stack into the A register (see Chapter 8 for fuller details on **Use of the Calculator**). All you need to know now is that it resets the IY register to point to ERR NR and corrupts most of the other registers so you should save them before calling this routine.

The parameters must be saved onto the calculator stack in the order X,Y,Z (Z = radius). The circle-drawing routine updates the system variables COORDS so if you do not want them changed

you should save them before drawing your circle and restore them afterwards. Thus, the routine to draw a circle would look something like this:

```
LD    HL,(23677) ;COORDS
PUSH HL          ;SAVE COORDS
LD    A,X        ;WHERE X = 0 - 255
CALL  2D28H     ;STACK A ROM ROUTINE
LD    A,Y        ;Y 0—175
CALL  2D28H
LD    A,Z        ;Z = RADIUS (MAKE SURE THAT
                ;THERE IS ROOM ON THE SCREEN
                ;OTHERWISE YOU WILL GET AN
                ;ERROR REPORT)

CALL  2D28H
CALL  232DH     ;DRAW CIRCLE
POP   HL
LD    (23677),HL ;RESTORE COORDS
```

If you wanted to draw your circle round the current COORDS position, you could have placed these onto the calculator stack, but you would still have to save them to the machine stack if you want to restore them unaffected.

### Drawing a Line: CALL 9146 (23BAh)

The ROM line-drawing routine proper starts at 9399 (24B7h) and takes its parameters from the calculator stack in the same way as the circle routine. In this case, however, it is easier to bypass the part which uses the calculator stack. The routine takes its start from COORDS so if you want to start from somewhere else you will have to load COORDS with your start position after first saving them if required for later restoration. Otherwise they will point to the end of your line. On entry the DE registers hold the signs of the DRAW parameters which are held in the BC registers: -1 (FFh) for negative, +1 (01h) for positive. The registers C and E hold X, whereas B and D hold Y.

The machine code equivalent to the BASIC line DRAW 0,175 would look like this:

```
LD    BC,AF00H ;175,0
LD    DE,0101H;+
CALL  24BAH
Or to DRAW -255,0
LD    BC,00FFH ;0,255
LD    DE,01FFH ;+,-
```

Note that these will draw from the current COORDS, and no attempt has been made to save them or alter them. After running the routine they will point to the last point plotted while drawing the line and an error will occur if you try to draw off the screen.

### Finding the Address of a Pixel: CALL 8874 (22AAh)

To find the address of the byte holding the pixel for a PLOT command this routine can be called with the BC register pair holding the X and Y coordinates (Y 0-175 in B and X 0-255 in C) will return with the HL register pair holding the address and the A register holding the bit position.

### Clearing Part of the Screen: CALL 3652 (E44h)

This routine will clear the number of lines set by the B register from the bottom of the screen (e.g., if B contained 1 then just the bottom line would be cleared, or if B contained 10 then the bottom ten lines would be cleared). The bottom of the screen is always line 24, not the lowest line of the main screen.

### Scrolling Part of the Screen: CALL 3584 (E00h)

To scroll part of the screen, first the B register has to be set to the number of lines to be scrolled - 1. Then the routine can be called. The bottom line after each call will be cleared and at least two lines

must be scrolled. Lines are again counted from the bottom of the full screen.

**Input to Current Channel: CALL 5606 (15E6h)**

This routine picks up the address of the current channel from the system variable CURCHL, finds the address of the input sub-routine from the channel information area and then calls the required routine.

**Clearing the Calculator Stack and Workspace: CALL 5823 (16BFh)**

This routine can be very useful to ensure that the calculator has the maximum workspace and that there is nothing inadvertently left on the calculator stack. It uses only the HL register pair and will return with the contents of the system variable STKEND in this.

**SAVEing, LOADING and VERIFYing**

The Spectrum SAVE and LOAD routines are fairly straightforward and very easy to use if both saving and loading are to be carried out from within the program, if the exact length of the data is known and if it does not matter if control is returned to BASIC in the event of an error or BREAK being pressed. If the length is not known, or something other than data is to be handled, things become more complicated.

Normally when loading the Spectrum expects a header to be received before the main block, and it is this header that tells the Spectrum how to deal with the main data block which follows. It is much easier to save and load without the header but, as mentioned previously, this can only be done if the exact parameters are known.

The header is 19 bytes long (not 17 as most books will tell you), but only the 17 middle bytes need to be set up as the routines for SAVEing and LOADING create the first and last bytes themselves.

Byte 1 is always 00 for a header and the last byte is a parity byte which is generated within the routine, so there is no need to worry about it. Byte 2 holds a type number:

- 0 for a BASIC program      2 for a string array
- 1 for a numeric array      3 for a block of code

Bytes 3 to 12 hold the name. Bytes 13 and 14 hold the length of the main block. For a BASIC program this would be E LINE—PROG system variables. Bytes 15 and 16, for a code block, will hold the start address for loading the block. Bytes 15 and 16 for a BASIC program, will hold the auto-start line number if applicable or the start address if not. For an array, byte 16 holds the array name in the form of:

- Bits 0-4 the name (A = 1 to Z = 26)
- Bit 5 reset if the array is numeric
- Bit 6 set if the array is a string
- Bit 7 set

Bytes 17 and 18 hold the length of the program for a BASIC program (i.e. VARS—PROG system variables). The last byte is a parity byte, and this is created as the actual SAVEing or LOADING is being carried out.

**The Header**

BYTES	1	2	3.....12	13	14	15	16	17	18	19
	FLAG	TYPE	PROGRAM NAME	DATA LENGTH	START	PROGRAM LENGTH				PARITY
IX+		00	1.....10	11	12	13	14	15	16	17

To save onto tape either a header must be created as above and saved followed by the main data block, or if the parameters of the main data block are known, and will still be known when it comes to be loaded, the data block alone can be saved.

There are a number of entry points to the SAVE routine which can be used and each has its own merits and disadvantages. The first entry point to be considered is perhaps the easiest, but also the most fraught with problems. It is used by setting the IX register to point to the start of the header (byte 2 as described in the header information above) and the HL register pair must be set to point to the start of the main block to be saved. Once these registers have been set a CALL can be made to 2416 (0970h) and SAVEing of both the header and the main block will take place. The problems are as follows:

- 1) The message "Start tape then press any key" will appear. If any key except BREAK is pressed all will be well, otherwise control will be returned to BASIC via the error-handling routine.
- 2) The BREAK key is periodically tested during the SAVE routine. If pressed, a premature return is made to the BASIC error-handling routine, which can be embarrassing.
- 3) The header is saved in such a form so as to ensure that the SAVEd material can be loaded back by BASIC. Under some circumstances, this may be regarded as an advantage not a problem.

The next entry point can be regarded as the same as the last with the exception that it does not ask for or wait for a key to be pressed. This must be accessed via a *subroutine* in the calling program because correct operation depends upon correct loading of the machine stack. First set the IX and HL register pairs as above, then CALL the following routine from your program:

```
SAVE  PUSH  HL
      JP    2436 (0984H)
```

The return after SAVEing is complete will be made to the location following the CALL to SAVE. This is a neat way of SAVEing several successive blocks complete with headers, as repeated 'start tape' messages are avoided.

To SAVE without the possibility of BREAKing into the SAVE routine to return to BASIC, the normal start must be bypassed, and the header and data saved separately as blocks. The routine SA\_BYTES located at 1218 (04C2h) is normally called to do any SAVEing, with the A register holding 00 for a header or FFh for a

data block. The first thing that this routine does is preload the machine stack with the address of the SAVE/LOAD RETURN routine. The SAVE/LOAD RETURN routine enables the interrupts then looks to see if BREAK has been pressed. The error-handling routine is called in with an RST 8 if BREAK has been pressed, thereby making a return to BASIC. If BREAK has not been pressed return is to the original RETURN address, put onto the stack by the calling routine.

If the instructions to place the SAVE/LOAD RETURN routine are bypassed, the RETURN address on the machine stack will be that of your calling program, and control will be returned to that on exit from SAVEing, with the carry flag reset if there has been an attempt to BREAK into the SAVEing process, or set if not, and the interrupts will be *off*, so they must be re-enabled. You will also have to organise some form of message to make sure that the tape has been started, perhaps by using the message printing and Wait Key routines detailed elsewhere in this chapter.

To save a block of code by this method, the IX register must point to the start of the block, and the DE register must hold the length. The A register must hold either FFh for a data block or 00 for a header. A direct CALL can then be made to 1222 (04C6h). Remember that the DE register pair will have to hold 17 (11h) if a standard header is being saved. Blocks of data can be saved by this method without a header, but can only be loaded back if the length is known.

If a return to BASIC is not a worry, or is an advantage, then the same routine can be used, but from the start, which is at 1218 (04C2h). If BREAK is pressed control will then be returned to BASIC. Again you will have to make sure the tape is running.

## LOADing and VERIFYing

Data on tape can be loaded into the Spectrum in two forms; either with a header or without. Where there is a header this can be used either to provide all the parameters for the loading of the main data block to follow or, as when loading from BASIC, only the details which are either unknown, known and definitely correct, or to ensure that the correct data is being loaded. Where there is no header, all the details that would have been in the header section *must* be known before the data can be loaded.

It is possible to create a different type of header from that used by the Spectrum, to define the parameters of the following data, by SAVEing a fixed length of data as a main data block and writing one's own decoding routine, which will furnish the details when the main block is loaded. This is often a good idea as it prevents the possibility of a block being lost, because you have forgotten the length or where it was on the tape, and hence cannot reload it. This will also prevent the code being loaded by an 'unauthorised' user, unless he takes the time to write his own code to examine what you have done and manages to interpret it. A routine which creates a special header is given in the DeBASE program in Appendix G.

To load a data block preceded by a normal header the first thing that must be done is allocate a space of 34 bytes in memory. In the first 17 bytes of this a 'dummy' header must be created. This defines details which are required to match, before the following block is loaded. Once a match of the required details is made any non-matching information will be checked against the real header and, assuming that the changes are acceptable, these details for the LOADing will be taken from the 'dummy' header. Should the changes be unacceptable, a BASIC error report will be generated by an RST 8 instruction. The second 17 bytes will be filled from the tape with headers for comparison. Only when a match is made will this area become free for use again.

The dummy header is made up in exactly the same manner as the header saved on the tape and again, the first and last bytes of the full 19 are generated internally, so are not included in the 17-byte specifier. The first byte *must* be the same identifier as the first byte in the header of the data to be loaded (i.e. the type must match). This is held in the A register on entry to the LOAD BYTES subroutine and need only be set if calling this directly. If the types do not match, a new leader is waited for and the process repeated until the types do match. The second byte (the first of the 17 that need to be set up in the dummy header) again *must* match: 0 for a BASIC program, 1 for a numeric array, etc. If there is no match the next block will be waited for. The next ten bytes are the name. If the name is to be disregarded then the first of these bytes must be FFh, the names will then be taken as matching. The next two bytes are the length, if this is set at 0 then the length will be taken from the tape header otherwise the two lengths *must* match. For a block of code, bytes 15 and 16 will be the start address to load to,

whereas for a BASIC array, byte 15 will be ignored and byte 16 will be the name of the array in the same form as when SAVEing. For a BASIC program, byte 15 will be 0 and byte 16 80h. The last bytes can be effectively ignored.

Before attempting to use the dummy header, the low byte of the system variable T ADDR must be set to contain 01 to LOAD or set to 2 in order to VERIFY.

The HL register pair must contain the address to which the main block is to be accepted, or 0 if the information in the tape header is to be used. For an existing BASIC array, this will be the start of the array data, following the array name and length bytes in the variables area of a BASIC program.

Finally the IX register must be set to point to the first byte of the dummy header. The ROM routine at 1889 (761h) can now be called to LOAD or VERIFY both the header and main data block from the tape.

LOADing or VERIFYing without a header can be achieved only if the full parameters of the block to be accepted are known, when it is a very simple process.

First the A register is loaded with FFh which signals that a main block is to be accepted. Next the DE register pair is loaded with the total length of the block to be accepted. Then the IX register is set to point to the address to start LOADing or VERIFYing the main data block. This followed by the carry flag being set, signalling a LOAD or reset to VERIFY. Finally, the ROM routine at 1366 (556h) is called to do the actual operation.

The start of this routine preloads the machine stack in the same way as the SAVEing routine detailed earlier, giving an error return in the event of BREAK being pressed. It can be bypassed by CALLING the following short subroutine in your own program:

```
LOAD INC D      ;RESET 0 FLAG
      EX  AF,A'
      DEC D      ;RESTORE D
      DI        ;THE INTERRUPTS MUST BE
                ;OFF TO ENSURE THAT THE
                ;TIMING IS CONSTANT. THEY
                ;ARE NOT TURNED ON BY THE
                ;ROM ROUTINE BEFORE RETURN
```

```
LD  A,0FH
OUT (FEH),A ;SET BORDER AND EAR PORTS
JP  562H ;JUMP TO MAIN LOAD ROUTINE
```

The OUT instruction sets the BORDER colour to white with the three low bits; these can be changed as required to give a different colour. Bit 5 should be left set as this initialises the EAR port for indication of anything coming from the tape prior to the leader.

A tape-loading error can be detected by the carry flag being reset on return to the calling routine and, as an attempt to BREAK will result in an immediate return, if the test break routine is called after checking for errors this can be catered for.

*Note:* The interrupts will be off on return from loading.

The routine can also be used to read in a header, instead of a main data block, by loading the A register with 0 prior to calling it. This can be useful if you wish to create a different type of header and read it with your own routine. This is demonstrated in the DeBASE program in Appendix G.

## The 8K Interface ROM

With the addition of the 8K ROM in the Microdrive interface, Sinclair have opened up the possibility of expansion in the form of sideways ROM as used in the BBC microcomputer, as well as an easy route to expansion of the Sinclair BASIC. The first thing that needs to be understood is the mechanism by which the Z80 CPU can be used to address memory outside its normal direct addressing range of 64 kilobytes.

The CPU has two sets of information lines — A 1–16 (which it uses to tell the memory which byte it wishes to use) — called the address bus, and D 0–7, the data bus, which is used to read and write to the memory location pointed to on the address bus. Normally it would be impossible to access outside the 64K range, so some form of interception must be made which substitutes a different bank of memory in the same address range.

This is quite easy to do by choosing a byte of memory, looking for its address on the address bus, and when a match is made switching to the alternative memory. Execution will then continue at the next address as the program counter will move on to point to the next location as normal, but the data will be fetched from the new bank of memory.

In the Spectrum this is done by looking at the address bus for the program counter addressing location 8, which is the error-handling routine. Since this address is reached only by a restart (RST 8), the top of the stack will always hold the address of the instruction following the calling instruction (i.e. the return address). This is fetched by the shadow ROM routine, and the contents of that memory location examined. Values in the range 00 to 1Ah cause a return to the 16K ROM as these are normal error codes, but numbers between 1Bh and 32h are used as 'hook codes'. These call shadow ROM routines and will now be detailed. To use a hook code use RST 8 followed by DEFB hook code. All locations relate to the shadow ROM.

## Inputs

### Get Key: Hook Code 1Bh, Location 6617 (19D9h)

This is similar to the GET command in some BASICs (although the Sinclair BASIC does not have this command). It waits until a key is pressed and then returns with the code of the key pressed in the A register. The maskable interrupt should be on as the 16K ROM routine is used to scan the keyboard.

### RS232 Input: Hook Code 1Dh, Location 2945 (B81h)

To use this routine, first the BAUD rate must be set using the system variable BAUD 23747/8 (5CC3/4h) calculated as  $3\,500\,000 / (26 \times \text{baud rate}) - 2$ , 3 500 000 being the clock frequency of the Spectrum. Next SER\_FL 23751 (5CC7h) should be set to 0, and then the input routine can be called. The A register will hold the code of the character received and the carry flag will be set. The routine will wait for only a certain period for a code to be received. If it has to wait too long, or if the space key is pressed, it will return, but without the carry flag set.

### Network Input: Hook Code 2Fh, Location 6705 (1A31h)

Before use, a network channel must be opened and made current by the use of the Open Channel routine described later (see **Network Output** section). This routine will read a packet from the network, on entry the IX register should be set to the start of the network area and IX + 11, IX + 12, and IX + 13&14 must be set to the correct values for the block to be received (see the details on the network header in the section on **Network Output** using hook code 30H). The block number in IX + 13&14 will be incremented after each block is successfully received.

It would appear that it was intended that the carry flag should indicate if a packet has been read or, if there was an error, when the return to the calling routine is made. However, the carry flag can be corrupted by the resetting of the border colour on exit from the routine. A return will be made from this routine when a packet has successfully been received, with the carry flag reset. If

the time allowed for a packet to be received has elapsed, there is a checksum error. Alternatively, if BREAK is pressed, the carry flag would be set.

Because of the problem inherent in this routine, it is easier to use the 16K ROM routine at 5606 (15E6h). Remember to preserve the IX register before calling this routine as it would be corrupted otherwise. The carry flag will be set if the A register does not hold a received code.

## Outputs

### Print to Screen: Hook Code 1Ch, Location 6636 (19ECh)

To use this the character code must be in the A register prior to calling the routine. This routine is a direct equivalent to the 16K ROM routines for setting output to stream 2 (main screen) and RST 16 printing to current stream, which actually uses these 16K ROM routines. Used in connection with the Get Key routine (see **Inputs** section above) you can create a sort of 'dumb terminal'. The program to do this would look something like this:

```
RST      8
DEFB    1BH
RST      8
DEFB    1CH
```

The first RST 8 will wait for a key to be pressed and return with the code in the A register, and the second RST 8 will echo the character to the main screen (or whatever output to stream 2 has been directed to).

### Print to the Printer: Hook Code 1Fh, Location 6652 (19FCh)

This is identical to the print-to-screen routine above except that it uses stream 3 (normally the printer) instead of stream 2.

### RS232 Output: Hook Code 1Eh, Location 3162 (C5Ah)

Again this is used by putting the code to be output in the A register, but using the RS232 output port on the interface. The

baud rate is picked up from the system variable BAUD and the border colour from the system variable IOBORD (see Chapter 4 for the settings of these). No information will be sent unless the DTR (Data Terminal Ready) line is high.

The main entry point called by the hook code allows any value of the A register to be output; care therefore needs to be taken to avoid sending control codes unintentionally. The other useful entry point is at 3132 (C3Ch) which looks for unprintable ASCII codes and intercepts them: any codes below 32 (20h) will cause an immediate return, except for code 13 (0Dh) (carriage return) which will be output and followed by a line feed code 10 (0Ah). Codes from 128 (80h) are treated according to their type. Graphics characters will be output as a '?' and tokens will be expanded by a call to the 16K ROM routine at 3088 (C10h), which requires 165 (A5h) to be subtracted from the code prior to its being called.

## Network Output

### Open Channel: Hook Code 2Dh, Location 3753 (EA9h)

Before any data can be sent or received on the network, a network channel must be opened. This is done by using this call after first setting D\_STR1 23766 (5CD6h) and NSTAT 23749 (5CC5h) system variables to be the destination and sending station numbers, respectively. A network channel will then be created at the end of the CHANS area, and everything from the address contained in the PROG 23635/6 (5C53/4h) system variable to that in the STKEND 23653/4 (5C65/6h) system variable will be moved up by 276 bytes and the relevant system variables will be reset, assuming there to be room below RAMTOP 23730/1 (5CB2/3h). If there is not enough room, an error will be caused.

On return the IX register will point to the start of the new channel. The newly created channel is temporary and signified as such by bit 7 of IX + 4 being set. To make it permanent this bit should be reset, and it can now be used to send by loading CURCHL 23633/4 (5C51/2h) with the address in the IX register and using the RST 16 (10h) routine in the 16K ROM to output data, a byte at a time.

After the final data has been passed to the RST 16 (10h) routine, the channel should be closed by the use of the hook code 2Eh

routine (see **Close Network Channel** below), which will send any remaining buffer contents and reclaim the buffer area. It will *not* close any streams attached to the network channel, and care must be exercised to ensure that other streams and channels are not corrupted.

### Send Packet: Hook Code 30h, Location 3530 (DB2h)

This CALL allows a 'packet' to be sent over the network. Before use a network channel must be opened, the header and system variables must be set up, and the data to be sent placed in the buffer. To open a network channel use hook code 2Dh (see details above).

On calling this routine, first a 'scout' is sent which claims the network if it is free. Then the header is sent followed by the main data block. The header is held in the front of the network channel and is addressed by the IX register which points to the first location in the channel. Bytes IX + 00 to IX + 10 are set up by the channel-operating routine and bytes IX + 11 to IX + 18 are the header. The header is made up as follows:

IX + 11	The station receiving
IX + 12	The station sending
IX + 13 & 14	The block number
IX + 15	1 for the last block otherwise 0
IX + 16	The length of the buffer (0-255)
IX + 17	Checksum for the data block
IX + 18	Checksum for the header
IX + 19 & 20	USED FOR RECEIVING ONLY
IX + 19	The position of the last code taken from the buffer
IX + 20	The number of bytes available in the buffer
IX + 21 to IX + 275	are the data for sending, up to 255 bytes.

The checksums are created by the routine itself so do not have to be inserted and IX + 15 is loaded from the A register on entry to the routine, but all the other details must be set by the user. In addition to setting the header, the system variables D\_STR1 23766 (5CD6h) must be the destination station number and NSTAT 23749 (5CC5h) the sending station number. On each call to this routine the block number will be incremented. The base

address of the network channel is returned in the IX register by the channel open routine.

#### Close Network Channel: Hook Code 2Eh, Location 6692 (1A24h)

If this routine is called after a send operation, it will transmit any data remaining in the network buffer, marking it as an end-of-file block, but after a receive operation it will discard any remaining data in the buffer. The 270 bytes of the buffer area will then be reclaimed using the 16K ROM routine at 6632 (19E8h) but the channel information is left.

### Microdrive Output

#### Open Channel/Open File: Hook Code 22h, Location 6953 (1B28h)

Before data can be sent to a Microdrive, a Microdrive channel and a map area for the drive selected must be created. Hook code 2Bh was intended to serve this purpose, but there was an error in the program, although this hook code can be used to perform the operation. First set D\_STR1 23766/7 (5CD6/7h) to contain the drive number (1-8), N\_STR1 23770/1 (5CD4/5h) the length of the file name and T\_STR1 23772/3 (5CD6/7h) to the start address of the file name in memory. The H'L' register pair must then be saved and the hook code can then be called.

The channel will be opened as a read channel if the file name exists, otherwise it will be opened as a write channel. To make the channel a permanent one, the data must be incorporated into the STREAMs information. The following subroutine achieves this:

```

EXX          ;SAVE THE RETURN ADDRESS TO
             BASIC
PUSH HL
EXX
LD   A,S    ;S IS THE STREAM NUMBER FOR THE
             CHANNEL
RLA        ;DOUBLE IT

```

```

LD   HL,5C16H ;BASE ADDRESS OF STREAMS
LD   D,0      ;OFFSET FOR S STREAM TO THE DE
LD   E,A      ;REGISTER PAIR
ADD  HL,DE    ;NOW HL = STREAM LOCATION
PUSH HL      ;SAVE IT
RST  8        ;CALL HOOK CODE
DEFB 22H     ;22 (OPEN CHANNEL/FILE)
PUSH HL      ;ON RETURN FROM THIS ROUTINE
             HL = STREAM DISPLACEMENT
             ;THE OPEN CHANNEL ROUTINE
             LEAVES THE
             ;MICRODRIVE MOTOR RUNNING
RST  8        ;SO IT MUST BE TURNED OFF
DEFB 21H     ;DE = STREAM DISPLACEMENT
POP  DE      ;HL = STREAM LOCATION
POP  HL      ;SET S STREAM TO CORRECT
LD   (HL),E  ;DISPLACEMENT

INC  HL
LD   (HL),D
RES  7,(IX + 4) ;ON RETURN FROM THE HOOK
             CODE 22 IX = START OF CHANNEL
             AREA. RESETTING (IX + 4) MAKES
             THE CHANNEL PERMANENT
EXX          ;RESTORE THE RETURN ADDRESS TO
             BASIC

POP  HL
EXX
RET          ;FINISHED

```

To write to or read from a Microdrive, the channel created must be made current by loading the system variable CURCHL 23633/4 (5C51/2h) with the base address, returned in the IX register after opening the channel. The cartridge can then be written to with the 16K ROM RST 16 (10h) routine, or read from with the 16K routine at 5606 (15E6h), which returns with each character in the A register. If it is required to SAVE, VERIFY, MERGE, or LOAD with the Microdrive as opposed to using the cartridge as a file, this is explained later in this chapter.

**Write Record: Hook Code 26h, Location 4607 (11FFh)**

This hook code will write the contents of the Microdrive buffer to the next free sector on the Microdrive cartridge. Before calling the routine, the buffer must contain the information to be saved, the IX register must point to the start of the Microdrive channel. IX + 11 must contain the length of the data, IX + 13 the sector number (starting at 0 for the first sector of a record and automatically incremented each time a record is either sent or received), IX + 14 to IX + 23 the record name and IX + 25 the drive to be used. Locations IX + 82 to IX + 593 are available for the data to be placed for sending.

**Write Sector: Hook Code 2Ah, Location 6801 (1A91h)**

This performs the same action as above but looks for a sector of a record with the sector number in IX + 13 and, if it exists, overwrites it with the information currently in the buffer. If the sector does not exist, a 'FILE NOT FOUND' error will be generated. No check is made as to whether the sector is free or not, so care must be exercised to ensure that nothing that you want is already on the sector.

**Close Microdrive Channel: Hook Code 23h, Location 4777 (12A9h)**

This hook code performs the same action as the Close Network Channel (hook code 2Eh) but for the Microdrive and on entry the IX register must hold the base address of the file. If the channel was used for sending, any information left in the buffer will be sent as an 'end of file' message and the buffer will then be reclaimed, or if it was a read channel anything left in the buffer will be lost as the channel is reclaimed.

**Erase File: Hook Code 24h, Location 7534 (1D6Eh)**

This will erase a Microdrive file whose name is written in memory and pointed to by T\_STR1 with name length N\_STR1 on Microdrive D\_STR1.

**Microdrive Input**

*All the following 'read' routines return with the Microdrive motor still running and the maskable interrupt off.*

**Read Print Record: Hook Code 27h, Location 6679 (1A17h)**

This will read a record from a print file, the record number of which is held in IX + 13. The IX register must point to the start of the Microdrive channel, IX + 25 must hold the drive number and IX + 14 to IX + 23 the record name. Assuming the record sector to be present, it will be read into the buffer otherwise an error will be caused.

**Read Next Print Record: Hook Code 25h, Location 6665 (1A09h)**

This is as above but, when used after the previous Hook Code, it will automatically read in the next record of a print file, if there is one. Otherwise an error will be caused.

**Read Record Sector: Hook Code 28h, Location 6731 (1A4Bh)**

This will read a record from a file. The IX register must point to the start of the Microdrive channel, IX + 13 should be set, the record number (within the file) IX + 25 must hold the drive number and IX + 14 to IX + 23 the file name. Assuming the record sector to be present, it will be read into the buffer. If it is a print file, a return will be made with the carry flag reset and the record in the Microdrive buffer area. Should the record be other than a print record the carry flag will be set and the record cleared from the buffer.

**Read Next Record Sector: Hook Code 29h, Location 6790 (1A86h)**

This is as above but for the next sector of a file. It is similar to Hook Code 25h.

**Motor on/Motors off: Hook Code 21h, Location 6135 (17F7h)**

The A register should be loaded either with 0, to turn off any motors that are running, or with the drive number whose motor is to be started. If the drive is not present, an error will be caused. If the drive is present, the motor will be started and a return made, with the interrupts off.

**Reclaim Microdrive Channel: Hook Code 2Ch, Location 4292 (10C4h)**

The Microdrive channel area currently pointed to by the IX register will be reclaimed. Any streams currently using the channel will be closed and the Microdrive map area is also reclaimed if not in use from another channel. All memory above the reclaimed channel up to RAMTOP is moved down by the 627 bytes freed by the reclamation.

**Keyscan: Hook Code 20h, Location 6657 (1A01h)**

This returns with the carry flag set if any key is being pressed.

**Insert Variables: Hook Code 31h, Location 6568 (19A8h)**

This inserts the extra system variables needed for the Interface 1. In fact, it is a single RET instruction as the variables are created automatically when the first 16K error occurs.

**ROM 2: Hook Code 32h, Location 6564 (19A4h)**

Perhaps this is the most useful of all the hook codes as it allows the 8K ROM to be paged in at will; therefore any routine can be called. To use it the location of the routine you want accessed must be placed in the system variable HD\_11, the hook code can then be used. Unfortunately, the only register that can be passed directly by this hook code is the A register, but this is no real problem.

When the hook code is used, the address in HD\_11 23789/90 (5CED/Eh) is put on the machine stack followed by the return address for the 8K ROM to 16K ROM switchover (1792 (700h)). If you make the address in HD\_11 point back to your own program, you can then 'POP' the two return addresses off the stack and have the 8K ROM paged in, with your program in control. Any registers can now be used freely with 8K routines. To return to the 16K ROM all that you need to do is to CALL 1792 (700h), and the 8K ROM will be switched out.

*Note:* Many routines look at the syntax/run flag in FLAGS 23611 (5C3Bh) IY + 1, and cause a return to the BASIC interpreter in syntax time. Some of the Microdrive routines also return to BASIC via the address in ERR SP 23613/4 (5C3D/Eh) if they are completed successfully, or via an 8K ROM error if they have failed. The 16K error can be diverted to point back to the calling routine by altering the address in ERR SP so that it points to the return address of your routine on the machine stack. This is demonstrated in the DeBASE program in Appendix G, in the Microdrive SAVE/LOAD routines.

The following routines are called by the RST 8 DEF8 32h hook code after putting the location into HD\_11.

**Catalogue Cartridge: Hook Code 32h, Location 7256 (1C58h)**

Before use, save H'L' registers. D\_STR1 must contain the Microdrive number and S\_STR1 the stream number for the catalogue to be printed to. The call can then be made and after the catalogue has been done H'L' registers are restored.

**Format Cartridge: Hook Code 32h, Location 7022 (1B6EH)**

Once again, before use the H'L' registers must be saved. N\_STR1 must hold the length of the name to be given to the cartridge, and T\_STR1 the address of the first character of the name when the routine is CALLED. The H'L' registers should be restored afterwards for a successful return to BASIC.

**Run: Hook Code 32h, Location 2709 (A95H)**

This is the simplest of the routines. It loads a program from Microdrive 1 called 'Run'. All that is necessary is a jump to the routine in ROM. This can be made after the 16K ROM has been paged out in the way described in the **Hook Code 32h** section above.

The following routines for SAVEing, LOADing, VERIFYing and MERGing all require a header to be set up before they can be used. The details of the make-up of a Microdrive channel are given in the Microdrive and Interface 1 manual (see pages 47 and 48) so I shall not repeat this detail here. I shall, however, explain how a header is made so that the Microdrive can be used from machine code.

**Making a Microdrive Header**

The Microdrive header is made up in much the same way as the tape header described in Chapter 2, but it is not necessary actually to set up the header in the Microdrive channel, as there is a routine in the ROM which will do this for you. All that has to be done is to set the system variables correctly. The subroutine to do this within your program would look something like this:

```
LD HL,23782 ;HD_00
LD (HL),T ;T = 0 for a program: 1 for a
numeric array: 2 for a string
array: 3 for code

INC HL
LD DE,LEN ;length of the main data block
LD (HL),E
INC HL
LD (HL),D
INC HL
LD DE,START ;start of data to be saved or dest
LD (HL),E ;when loading. 0 if information on
INC HL ;cartridge is to be used
LD (HL),D
the rest of this need only be carried out for
BASIC programs and arrays. Otherwise jump to GO_ON
```

```
INC HL
LD DE,DATA ;program length or variable name as
LD (HL),E ;described in chapter 2 in saveing
INC HL ;and loading
LD (HL),D
INC HL
LD DE,AUTO ;auto start line number, if any
LD (HL),E ;otherwise a number over 10000
INC HL
LD (HL),D
GO_ON LD HL,23769 ;make it a microdrive operation
LD (HL),"M" ;it must be a captial letter
SET BIT,(IY+124);bit 4 to load, bit 5 to save
bit 6 to merge, bit 7 to verify
LD HL,NLEN ;length of file name (max 10
letters characters or tokens)
LD (23770),HL;this is NSTR1
LD HL,NAME ;start address of file name
LD (23772),HL;T_STR1
LD HL,DRIVE ;the drive number (1 - 8)
LD (23766),HL;D_STR1
BIT 5,(IY+124);is it a save operation
JP NZ,1E7FH ;if so go into save routine in ROM
JP OBAF ;else ROM load/verify/merge routine
NAME DEFS 10 ;10 spare bytes for file name
```

The subroutine above *must* be CALLED from the main program. The RETURN after the operation has been performed is from the ROM routine to your calling routine, via the address on the machine stack pointed to by ERR SP (see the DeBASE program in Appendix G for details of how to ensure a return). The 16K ROM will be paged in by the SAVE or LOAD routine.

Any errors in the setting up of the header will cause either a BASIC error or a machine crash, so great care must be taken. As with most 8K ROM routines, it is important to save the H'L registers before using the routines and restore them afterwards.

When a machine code program using ROM routines is being written, it is very important to remember which ROM will be switched in at any given time. Not only are all the routines in ROM changed, but all the ReStarts in the 8K ROM are different from those in the 16K ROM.

It is possible to have a free exchange between the two ROMs by using Hook Code 32h from the 16K ROM and RST 10h from the 8K ROM. The use of Hook Code 32h has already been explained, but

details of how to use RST 10h and the other ReSTarts (RST) in the 8K ROM are given below:

```
RST 0  POP  HL          ;REMOVES THE RETURN
                          ADDRESS FROM THE
                          MACHINE STACK
      LD   (IY + 124),00 ;THIS IS FLAGS 3
      JP  700H          ;700H IS THE RETURN TO
                          THE 16K ROM
```

The address at the top of the stack at this stage should be the return to the routine that originally called the 8K ROM.

```
RST 8  LD   HL,(CHLADD) ;THIS HAS NO EFFECT
                          WHEN CALLED FROM 16K
      POP  HL          ;THE ADDRESS OF THE
                          ERROR CODE
      PUSH HL         ;SAVE IT AGAIN
      JP  009AH       ;THIS WILL CHECK WHAT IS
                          BEING EXECUTED AND
                          EITHER RETURN TO A 16K
                          EITHER RETURN TO A 16K
                          CODE OR EXECUTE A HOOK
                          CODE OR ACCESS A
                          CHANNEL.
```

This is of no practical use when the 8K ROM is switched in.

The RST 16 (10h) in the 8K ROM CALLs a 16K ROM routine, the address of which is held in a DEFINed word (DEFW) after ReSTart. After the 16K routine has been executed control is returned to the address after the DEFW in the CALLing routine.

```
RST 16  LD   (5CBAH),HL ;SAVE HL FOR
                          RESTORATION AFTER
                          RETURN
      (10H) POP  HL          ;COLLECT IN HL THE
                          ADDRESS OF THE DEFW
      PUSH DE         ;SAVE DE ON MACHINE
                          STACK
      JR   0081H
```

```
0081H  LD   E,(HL)        ;PUT THE ADDRESS TO BE
                          CALLED IN THE
      INC  HL             ;16K ROM INTO DE
      LD   D,(HL)
      INC  HL
                          ;HL NOW HOLDS THE
                          RETURN ADDRESS AFTER
                          THE DEFW
      EX  (SP),HL        ;MAKE THIS THE NEW
                          RETURN ADDRESS
      EX  DE,HL
      LD   HL,0          ;THIS WILL LET THE RST 8
                          ROUTINE KNOW
      PUSH HL           ;IT IS A RETURN FROM A
                          8K TO A 16K CALL
      LD   HL,8          ;THE ERROR RESTART
      PUSH HL           ;ONTO THE STACK FOR A
                          LATER RETURN
      LD   HL,5CB9H     ;THE START OF THE
                          SUBROUTINE
      PUSH HL           ;ONTO THE STACK FOR A
                          RETURN
      JP  0700H         ;A RET INSTRUCTION
                          THAT SWITCHES IN THE
                          16K ROM
0700H  RET              ;PAGE IN THE 16K ROM
                          AND RETURN
```

I have given the full listing for this as it is an interesting method of transferring control.

When the RETurn at 0700h is made, switching in the 16K ROM, the address collected from the stack is 5CB9h. This address contains 21h, the instruction LD HL,NN. As can be seen above the NN was the value in HL when the ReSTart was made. The next location — 5CBCh — contains CDh, the instruction CALL NN. Again this address was loaded above, from the DEFW after the RST 16 (10h). The return from the 16K routine will be to 5CBFh which contains 22h, the instruction LD (NN),HL. The NN in this case is 5CBAh, so the new value in the HL register is saved where the original value was. Finally there is the instruction RET. This will be a return to the address on the top of the machine stack,

now 8 (i.e. the error ReStart). This time when the address for the DEFBS is 'POPped' off the stack it will be 0. This will tell the ROM (8K now) that the HL register has to be loaded from (5CBAh), and after this there is yet another RETurn, this time to the address after the two DEFBS after the RST 16 (10h), seemingly so long ago.

The following ReStart can be used to generate the shadow ROM error messages in the same way as the RST 8 in the main ROM:

```

RST 24  BIT  7,(IY + 1)  ;THIS IS FLAGS
(18H)  RET                ;THE ZERO FLAG WILL BE
                                SET IF A BASIC LINE IS
                                BEING CHECKED FOR
                                SYNTAX
RST 32  RST  24          ;CHECK IF SYNTAX OR
(20H)  JR   Z,0068H      ;THIS WILL RESET THE
                                STACK POINTER TO THE
                                ADDRESS IN ERR_SP SET
                                X_PTR TO CH_ADD AND
                                RETURN TO THE MAIN
                                ROM LINE INPUT
                                ROUTINE VIA THE CLEAR
                                CALCULATOR ROUTINE,
                                AND INDICATE THE
                                POSITION OF THE
                                ERROR.
                                JR   003AH  ;THIS WILL GENERATE AN
                                ERROR REPORT. THE
                                SHADOW ROM ERROR
                                CODE WILL BE
                                COLLECTED FROM THE
                                BYTE AFTER THE
                                RESTART.

```

This ReStart can be used to generate a 16K error by loading ERR\_NR with the error code before calling it.

```

(28H)  JR   0040H        ;THIS IS TV FLAG
                                ;THIS WILL CHECK IF THE
                                ERROR OCCURRED IN
                                SYNTAX OR RUN TIME. IF
                                IN RUN TIME IT WILL
                                RETURN TO THE 16K
                                ERROR ROUTINE
                                OTHERWISE IT WILL
                                JUMP TO 0068H, AS
                                EXPLAINED ABOVE.
RST 48  JP   01F7H      ;CHECKS TO SEE IF THE
(30H)                                INTERFACE
                                VARIABLES ARE PRESENT
                                AND IF NOT INSERTS
                                THEM.
RST 56  EI              ;THIS IS THE NORMAL
(38H)                                MASKABLE INTERRUPT
                                WHICH WOULD SCAN
                                THE KEYBOARD IF THE
                                16K ROM WAS PAGED IN.
                                NOTE THAT THE
                                KEYBOARD WILL NOT BE
                                SCANNED WITH THIS.
                                RET

```

The non-maskable interrupt routine at 102 (66h) consists of the single instruction RETN.

More 8K ROM routines are discussed in Chapter 7 but there are few routines that are of use for other purposes. Their use and locations have already been detailed in the hook codes section.

## The System Variables

The BASIC interpreter requires some means of keeping track of where information is stored, and what it is supposed to be doing, since it is located in ROM (Read-Only Memory) and has to use RAM (Random Access Memory) to store this information. Since the ROM is unalterable, the addresses of the system variables have to be predefined in such a way that they can be accessed on a regular basis. In the Spectrum, this is achieved both by having their locations fixed in particular area of memory and also by indexing them with the IY register, which always points to 5C3Ah (ERR NR).

### 16K System Variables

#### KSTATE: Locations 23552–23559 (5C00h–5C07h)

These locations are used in reading the keyboard. They can be divided into two sets of four, and each set is treated the same as the other. Which set is actually used depends upon the state of the other set.

The CAPS SHIFT value of a key being pressed is stored in the first location of each set or, if no key is pressed, 255 (FFh) which flags the set as free for use. The count down to a set being free is stored in the second byte, initially set to 5, free when 0. The repeat delay is in the third byte, initially loaded from REPPER. The ASCII code of the key pressed when a set was in use is in the last location of that set until the countdown to the set being free reaches 0. In order to try to make some sense of this I will explain the routine that uses these variables.

If a key is pressed, the scanning subroutine looks to see whether the first set is empty. If it is it will then use this set, otherwise it will look at the second set. When a set is free then the

CAPS SHIFT code of the key being pressed will be saved in the first location. The second location is set to 5 (the countdown to free, which also serves to debounce the keyboard), the third location is loaded with the repeat delay REPDEL. The ASCII code of the key pressed is decoded, saved in the last location and copied into LAST K, bit 5 of FLAGS is set to signal a new key press and a return to the calling routine made. When neither set is free then a counter in the second byte of each set is decremented and the calling routine is returned to.

Once one of the counters reaches 0, the code of the key being pressed (if any) is saved and a comparison is made between the code of the keys in the two sets. If they match a key is being held down and then the routine allows for repeating of the key. This is not the key-debouncing routine, but the repeating key routine. The repeat delay being held in the third byte and initially loaded from the system variable REPDEL.

Once the delay for the first repeat is over (the routine must be called the number of times set by the delay counter), if the match has been continuous then the third byte is loaded from the system variable REPPER, the code is again passed to the system variable LAST K, bit 5 of FLAGS set and a return to the calling routine made. On subsequent calls to the keyboard-scanning routine, the whole process is repeated but with the shorter delay from REPPER until the codes no longer match. If the keys do not match then the treatment is the same as for a free set.

The net result of all this is that there is a two-key rollover, which allows for a second key being pressed before the delay for a repeat on the first set has elapsed, avoiding the loss of the second key if it is not held down long enough for the original set to become free.

#### LAST K: Location 23560 IY – 50 (5C08h)

This contains the code of the last key pressed.

#### REPDEL: Location 23561 IY – 49 (5C09h)

The number of times the keyboard-scanning routine must be called before a key repeats is contained at this location.

**REPPER: Location 23562 IY - 48 (5C0Ah)**

This contains the number of times the keyboard scan must be called before a repeating key repeats.

**DEFADD: Locations 23563/4 IY - 47 (5C0B/Ch)**

These locations hold either the address of a user-defined function being evaluated or 0.

**K DATA: Location 23565 IY - 45 (5C0Dh)**

This is used as temporary storage of the colour information when colour controls are being entered (i.e. the second byte, the colour number, after the E shift).

**TVDATA: Location 23566/7 IY - 44 (5C0E/Fh)**

These are as above but for output and used with AT and TAB values as well, hence the two bytes.

**STRMS: Locations 23568-23605 IY - 42 (5C10h-5C35h)**

These contain the addresses of the channels attached to streams. Initially streams -3 to 3 are in the first 14 bytes, but the information for extra streams (up to a total of 19) is added when they are opened, in the relevant place.

**CHARS: Locations 23606/7 IY - 4 (5C36/7h)**

These contain the address of the theoretical start of the character set, each character comprising of eight bytes; in fact, 256 less than the actual start as all characters are addressed relative to this position by multiplying their ASCII code by eight and ASCII 0-31 are not printable and thus are handled differently, not from this table.

The last valid ASCII code is 127 (7Fh), but the Spectrum uses the remaining 128 values of the full eight-bit range for tokens. These are dealt with separately and do not comprise part of this character set.

**RASP: Location 23608 IY - 2 (5C38h)**

This contains the length of the buzz generated when a basic line exceeds 22 lines or memory has run out.

**PIP: Location 23609 IY - 1 (5C39h)**

This contains the duration of the beep whenever a key is pressed.

**ERR NR: Location 23610 IY + 0 (5C3Ah)**

This contains one less than the report code that will be generated on an error. It can be used to generate your own reports or use the resident reports for your own purposes (see Chapter 2 for details).

**FLAGS: Location 23611 IY + 1 (5C3Bh)**

Bit 0 is set to suppress a leading space when a token is being printed if a space immediately precedes it, but is reset if a space is to be printed.

Bit 1 is set when stream 3 (normally the printer) is to be used for output from the RST 16 routine, but is reset for stream 2 (normally the main screen).

Bit 2 is set when printing is to be using L mode and is reset for K mode.

Bit 3 is set to signal L mode, normally when getting an input, but is reset to signal K mode.

Bit 5 is set if a new key has been pressed since it was last reset (see

details on Scanning the Keyboard in Chapter 2 and for KSTATE system variable).

Bit 6 is used to indicate if an expression is numeric or a string, being set for numeric and reset for a string, and is used by the BASIC interpreter.

Bit 7 is reset when the BASIC interpreter is checking a line for syntax on entry and is set when the program is being run.

**TV FLAG: Location 23612 IY + 2 (5C3Ch)**

Bit 0 is set if handling the lower screen and is reset for the main screen.

Bit 3 signals that the current mode may have changed and needs to be rechecked.

Bit 4 is set when an automatic listing is being printed, otherwise it is reset.

Bit 5 signals that the lower screen needs to be cleared (e.g. when a report is about to be printed).

**ERR SP: Locations 23613/4 IY + 3 (5C3D/Eh)**

These locations hold the address on the machine stack of the RETURN to be used in the event of an error in a BASIC command being executed. It is picked up by an error ReSTart and frequently changed by BASIC. It can be changed from machine code to make an error go to your own routine, as demonstrated in the DeBASE program in Appendix G, in the Microdrive SAVE/LOAD section. If an error occurs this must be reset and the error cancelled by making ERR NR hold 255 (FFh). Note that 0 OK is counted as an error.

**LIST SP: Locations 23615/6 IY + 5 (5C3F/40h)**

LIST SP is used to save the address of the stack pointer so that it

can be restored after automatic listing is completed. This is necessary because the listing can be terminated at different places, with different amounts on the machine stack (e.g., by a 'n' in response to a 'Scroll?').

**MODE: Location 23617 IY + 7 (5C41h)**

This determines the cursor used in an input, but it will only affect the first key press except when changed to force the graphics mode. However, it can be useful to get different cursors for input (e.g., if you POKE MODE with 32 you will get a flashing graphics input). Experimentation is the best way to make use of this facility as the system will not crash.

**NEWPPC: Locations 23618/9 IY + 8 (5C42/3h)**

This contains the line number of the next statement to be interpreted.

**NSPPC: Location 23620 IY + 10 (5C44h)**

This is the number of the statement in the line to be evaluated next. Poking first the line number then the statement number forces a jump in execution to that place.

**PPC: Locations 23621/2 IY + 11 (5C45/6h)**

This contains the line number of the statement currently being evaluated. Also it is used by an auto-run program loaded from the header.

**SUBPPC: Location 23623 IY + 13 (5C47h)**

This contains the statement number currently being evaluated.

**BORDCR: Location 23624 IY + 14 (5C48h)**

This contains the value for the border colour \*8 plus the attributes for the lower screen. Bits 7 and 6 can be used to make the lower screen flash or bright.

**E PPC: Locations 23625/6 IY + 15 (5C49/Ah)**

E PPC contains the number of the current line (i.e. the line that will be displayed with the cursor and brought into the editing area addressed by E LINE by an EDIT command).

**VARs: Locations 23627/8 IY + 17 (5C4B/Ch)**

This contains the address of the start of the variables area. When it is used in conjunction with E LINE, the total length of a BASIC program's variables can be calculated.

**DEST: Locations 23629/30 IY + 19 (5C4D/Eh)**

This contains the address of the first letter of the name of the variable currently in use. If a new variable is to be used, it is the 80h end marker immediately before the address in E LINE where the start of a new variable is being placed.

**CHANS: Locations 23631/2 IY + 21 (5C4F/50h)**

This contains the address of the start of the channel information area in which the details for open channels are kept.

**CURCHL: Locations 23633/4 IY + 23 (5C51/2h)**

These locations hold the address of the start of information in the channel information area for the channel currently in use.

**PROG: Locations 23635/6 IY + 25 (5C53/4h)**

This is the address of the start of the BASIC program area, which will be the byte following the channel information area and any input/output buffers being used by Interface 1.

**NXTLIN: Locations 23637/8 IY + 27 (5C55/6h)**

This is the address of the start of the next BASIC line in a program.

**DATADD: Locations 23639/40 IY + 29 (5C57/8h)**

This is the address of the terminator of the last data item used, or the start of the line given by a RESTORE command, or the first after it if it does not exist. This keeps track of which data item is to be used next and if there are no more data after this pointer it will cause an 'out of data' error.

**E LINE: Locations 23641/2 IY + 31 (5C59/Ah)**

E LINE is the address of the start of the editing area, which will be the start of any line in the edit area.

**K CUR: Locations 23643/4 IY + 33 (5C5B/Ch)**

This is the address of the cursor in the current line. It is used to edit or to create a new line in the edit area.

**CH ADD: Locations 23645/6 IY + 35 (5C5D/Eh)**

This is the address of the next character to be interpreted by the BASIC interpreter (n.b., any numerics are marked by a CHR\$ 14 and these are skipped).

**X PTR: Locations 23647/8 IY + 37 (5C5F/60h)**

These locations hold the address at which BASIC syntax has

failed when entering a new BASIC program line, marked by a '?'. It is also used to store information temporarily by the interpreter.

**WORKSP: Locations 23649/50 IY + 39 (5C61/2h)**

This is the address of temporary workspace, the first byte of free space created by a CALL 5717 (1655h) Make bc Space routine (see Chapter 5 for details, given in the section on **Standard Streams**).

**STKBOT: Locations 23651/2 IY + 41 (5C63/4h)**

These locations are the address of the bottom (start) of the calculator stack. This is explained fully in chapter 8.

**STKEND: Locations 23653/4 IY + 43 (5C65/6h)**

This is the address of the top (end) of the calculator stack. Again it is fully explained in Chapter 8.

**BREG: Location 23655 IY + 45 (5C67h)**

This is the B register for use by the calculator (see Chapter 8 for further details).

**MEM: Locations 23656/7 IY + 46 (5C68/9h)**

MEM provides the address of the calculator memory area (see Chapter 8 for further details).

**FLAGS2: Location 23658 IY + 48 (5C6Ah)**

Bit 0 is set if it is unnecessary for the main screen to be cleared when a line is being input into the editing area.

Bit 1 is set if the printer buffer has been used by a 16K ROM routine and is reset after it is cleared by the **Clearing Printer Buffer** routine described in Chapter 2.

Bit 2 set signals that the screen is clear.

Bit 3 set signals that CAPS LOCK is on.

Bit 4 set signals that K channel is in use.

**DF SZ: Location 23659 IY + 49 (5C6Bh)**

This is the number of lines allocated to the lower screen. It can cause the BASIC system to crash if set to less than three since one spare line is always left between the main screen and any reports. If there is nowhere for the reports to be printed, the interpreter hangs up.

**S TOP: Locations 23660/1 IY + 50 (5C6C/Dh)**

This address contains the line number of the first program line to be listed by a BASIC LIST command.

**OLDPPC: Locations 23662/3 IY + 52 (5C63/Fh)**

This address holds the line number containing the statement after that at which a BREAK or STOP was executed (i.e. the line that will be interpreted after a CONTINUE command).

**OSPCC: Location 23664 IY + 54 (5C70h)**

This is as above but provides the statement number within the line.

**FLAGX: Location 23665 IY + 55 (5C71h)**

The FLAGSX system variable is a rough equivalent of the FLAGS

system variable (location 23611 (5C3Bh)) and is used in its place in the interpreter when INPUT commands are being executed.

Bit 1 is set if the BASIC interpreter is handling a new variable.

Bit 5 is set when the 16K ROM is in input mode and is reset when in editing mode.

Bit 6 is set if a string is being handled.

Bit 7 is set if the BASIC interpreter is dealing with an INPUT LINE command.

**STRLEN: Locations 23666/7 IY + 56 (5C72/3h)**

These locations hold either the length of an existing string variable that is currently being used or, for a numeric or new string variable, the low byte will hold the letter of the variable (in the form described in the section on SAVEing and LOADing, see Chapter 2).

**T ADDR: Locations 23668/9 IY + 58 (5C74/5h)**

This normally holds the address of the next item in the syntax tables located from 6728 (1A48h) in the main ROM. However, it is also used for other purposes by some routines.

**SEED: Locations 23670/1 IY + 60 (5C76/7h)**

This is the seed for a random number. It is taken from the low two bytes of FRAMES if a RANDOMIZE command has no number, otherwise it is taken from the RANDOMIZE number.

**FRAMES: Locations 23672-23674 IY + 62 (5C78h-5C7Ah)**

At this address, a three-byte frames counter is held. It is set to 0 when the Spectrum is first switched on and is incremented each

time the normal interrupt routine is called. The least significant byte is in 23672 (5C78h).

**UDG: Locations 23675/6 IY + 65 (5C7B/Ch)**

This is the address of the first User-Defined Graphic.

**COORDS: Locations 23677/8 IY + 67 (5C7D/Eh)**

COORDS provides the X and Y coordinates of the last point plotted. This can be used to reset the start position for DRAWing or CIRCLE without having to PLOT the start; X (across) is in 23677 (5C7Dh).

**P POSN: Location 23679 IY + 69 (5C7Fh)**

This contains the column number of the next position to be used in the printer buffer (as S POSN: Locations 23688/9 IY + 78 (5C88/9h) below but for printer).

**PR CC: Location 23680 IY + 70 (5C80h)**

This is the low byte of the printer buffer current address. In effect, this is identical to DF CC: Locations 23684/5 IY + 48 (5C84/5h) and the printer buffer can be moved by changing the so-called unused byte below, which is in fact the high byte of the PR CC system variable. Unfortunately this is reset at the end of each print line to point to the original address; care therefore needs to be exercised if it has been moved.

**NOT USED: Location 23681 IY + 71 (5C81h)**

In effect, this is not used if the printer buffer is in the normal place (see PR CC: Location 23680 IY + 70 (5C80h)) but not free for use, except as above.

**ECHO E: Locations 23682/3 IY + 72 (5C82/4h)**

As **S POSN: Locations 23688/9 IY + 78 (5C88/9h)** this is the column and line number for the next print position, but for the input buffer. It is used when entering a BASIC line.

**DF CC: Location 23684/5 IY + 74 (5C84/5h)**

This contains the address of the top pixel line of the next print position. **DF CC** can be used to change the position of normal printed characters, by altering this to the next line down, but it can cause unexpected effects since the **RST 16 (10h)** routine simply adds 256 (100h) for each subsequent pixel line of a printed item. **The Spectrum screen map** in the appendix can be used to see the effect when moved down by more than one pixel line.

**DFCCL: Locations 23686/7 IY + 76 (5C86/7h)**

This is the lower screen version of **DFCC: Locations 23684/5 IY + 74 (5C84/5h)**.

**S POSN: Locations 23688/9 IY + 78 (5C88/9h)**

These locations hold column and line number for the next **PRINT** position on the main screen and set up by the **CALL 3545 (DD9h)** routine detailed in Chapter 2 (where 33 is left column and 24 is top line).

**SPOSNL: Locations 23690/1 IY + 80 (5C8A/Bh)**

Like **S POSN: Locations 23688/9 IY + 78 (5C88/9h)**, these locations hold the column and line number for lower screen.

**SCR CT: Location 23692 IY + 82 (5C8Ch)**

This holds 1 more than the number of times the screen can be scrolled without a 'scroll?' question. It should always be kept to a

high value since a machine code program could be broken into or crashed, if a negative response is given.

**ATTR P: Location 23693 IY + 83 (5C8Dh)**

This provides the global attributes for any screen printing, plotting, drawing, etc. It is set by a BASIC colour statement and can be loaded from within a machine code program, in which case all subsequent colours will be changed.

**MASK P: Location 23694 IY + 84 (5C8Eh)**

This is used as a mask to discriminate between transparent colour items and colour items to be taken from **ATTR P: Location 23693 IY + 83 (5C8Dh)**. For any bit set to 1, the attribute bit will be taken from the screen attribute for the current position, otherwise it will come from **ATTR P**.

**ATTR T: Location 23695 IY + 85 (5C8Fh)**

This provides the temporary attributes, set up by a colour item printed by a **RST16 (10h)** instruction.

**MASK T: Location 23696 IY + 86 (5C90h)**

This is used as a mask to discriminate between transparent colour items and colour items to be taken from **ATTR T: Location 23695 IY + 85 (5C8Fh)**. For any bit set to 1, the attribute bit will be taken from the screen attribute for the current position, otherwise it will come from **ATTR T**.

**P FLAG: Location 23697 IY + 87 (5C91h)**

This is the flag used to discriminate between the print parameters for any output to the screen by the ROM. The even bits are temporary bits, whereas the odd ones are permanent bits, each relating to the same items.

Bit 0/1 is set if OVER 1 is to be used.

Bit 2/3 is set if INVERSE is to be used.

Bit 4/5 is set if INK 9 is to be used.

Bit 6/7 is set if PAPER 9 is to be used.

**MEMBOT: Locations 23698–23727 IY + 88 (5C92h–5CAFh)**

This is the area used by the calculator to store values which cannot conveniently be kept on the calculator stack (see Chapter 8).

**Not Used: Locations 23728/9 IY + 118 (5CB0/1h)**

**RAMTOP: Locations 23730/1 IY + 120 (5CB2/3h)**

The main use of this address, which contains the address of the last byte of the BASIC area, will probably be to ensure that the calculator stack has sufficient space (see Chapter 8). It is set by a clear number command from BASIC which also initiates the machine stack to this address.

**P-RAMT: Locations 23732/3 IY + 122 (5CB4/5h)**

This contains the address of the last byte of physical RAM.

## 8K System Variables

There now follow the 8K ROM system variables which are only present when inserted by the Microdrive interface.

**FLAGS3: Location 23734 IY + 124 (5CB6h)**

Bit 0 is set if an extended command is being executed.

Bit 1 is set if a CLEAR# is being executed.

Bit 2 is set if the ERR SP system variable has been altered by the 8K ROM.

Bit 3 is set for NETWORK routines.

Bit 4 is set for LOAD routines by the 8K ROM.

Bit 5 is set for SAVE routines by the 8K ROM.

Bit 6 is set for MERGE routines by the 8K ROM.

Bit 7 is set for VERIFY routines by the 8K ROM.

**VECTOR: Locations 23735/6 IY + 125 (5CB7/8h)**

This contains the address to be jumped to if syntax has failed both the 16K and the 8K ROM interpreters. It is normally set to 496 (1F0h) which will cause a 16K ROM error. It can be changed to point to routines which check syntax further. This is covered in Chapter 7.

**SBRT: Locations 23737–23746 (5CB9h–5C2h)**

Strictly this is not a system variable but a short subroutine used by the 8K ROM to call 16K routines. The full details of this are given in Chapter 3 with the RST 16 routine in the 8K ROM.

**BAUD: Locations 23747/8 (5CC3/4h)**

This is the value used to set the BAUD rate for RS 232 input/output. It is calculated as follows:

$$\frac{3\,500\,000}{26 \times \text{baud rate}} - 2 = \text{BAUD}$$

3 500 000 being the clock rate of the Spectrum. This can be used to send or receive at non-standard baud rates. The default value is 12 (0Ch) which gives a baud rate of about 19 200.

**NTSTAT: Location 23749 (5CC5h)**

This is the network station number currently assigned to the Spectrum.

**IOBORD: Location 23750 (5CC6h)**

This holds the colour for the border during input/output operations, and is loaded with the colour number. It is normally 0 for black, but can be changed.

**SER\_FL: Locations 23751/2 (5CC7/8h)**

This is used during RS232 input. The first byte is a flag, set to 0 on entry to the input routine and set to 1 when a byte has been received. The second byte is the received byte, on return from the input routine.

**SECTOR: Locations 23753/4 (5CC9/Ah)**

This is used by the 8K ROM to count Microdrive sectors.

**CHADD\_: Locations 23755/6 (5CCB/Ch)**

This is the 8K ROM equivalent to CH\_ADD (Locations 23645/6) for the 16K ROM. It is used to store the address of the CH\_ADD address while the extended syntax is being checked and then replaced if necessary.

**NTRESP: Location 23757 (5CCDh)**

NTRESP is the response code given to the network.

The next eight bytes make up the network header which is explained in Chapter 3.

**NTDEST: Location 23758 (5CCEh)**

This contains the station to which network output is directed.

**NTSRCE: Location 23759 (5CCFh)**

This contains the station sending on the network.

**NTNUMB: Locations 23760/1 (5CD0/1h)**

This contains the number of the network block currently being passed.

**NTTYPE: Location 23762 (5CD2h)**

This holds the identifier for a network block, 0 for a normal block or 1 for the final block.

**NTLEN: Location 23763 (5CD3h)**

This holds the length of the network block being transmitted.

**NTDCS: Location 23764 (5CD4h)**

This contains the checksum for the data block to follow.

**NTHCS: Location 23765 (5CD5h)**

This holds the checksum for the seven bytes of the header.

The next eight bytes make up the first file specifier.

**D\_STR1: Locations 23766/7 (5CD6/7h)**

For Microdrive transactions, this provides the drive number (as two bytes).

For network transactions, this provides the destination station number.

For RS232 transactions, this provides the baud rate.

(For a fuller description of the use of D\_STR1 see Chapter 3, where each use is discussed when explaining the routines which use it.)

**S\_STR1: Location 23768 (5CD8h)**

This contains the stream number (0–15).

**L\_STR1: Location 23769 (5CD9h)**

This contains channel type in upper case.

**N\_STR1: Locations 23770/1 (5CDA/Bh)**

N\_STR1 holds the length of the file name.

**T\_STR1: Location 23772/3 (5CDC/Dh)**

**T\_STR1 holds the address of the start of the file name.**

The next eight bytes are used by LOAD and MOVE commands.

**D\_STR2: Locations 23774/5 (5CDE/Fh) to T\_STR2: Location 23780/1 (5CE4/5h)**

These eight bytes are the same as previous eight which make up the first file specifier.

The following bytes are direct equivalents to the header bytes for the 16K ROM routines, but they are used by the 8K ROM (for an explanation of their uses see Chapter 3).

**HD\_00: Location 23782 (5CE6h)**

This holds the file type where: 0, program; 1, numeric array; 2, string array; 3, code.

**HD\_0B: Locations 23783/4 (5CE7/8h)**

The length of data is held in these locations.

**HD\_0D: Locations 23785/6 (5CE9/Ah)**

This holds the start of data.

**HD\_0F: Locations 23787/8 (5CEB/Ch)**

This holds the array name or program length.

**HD\_11: Locations 23789/90 (5CED/Eh)**

This holds the auto-start line number. It can also be used by hook code 32h (see Chapter 3).

**COPIES: Location 23791 (5CEFh)**

This dictates the number of copies made by a SAVE command.

# Input and Output Ports and Channels

The standard Spectrum has all the address and data bus lines available on the edge connector at the back, and the BASIC allows for communication with the outside world by means of the IN and OUT commands. These use the IN A,(C) and OUT A,(C) instructions on the Z80 CPU allowing the full address bus to be used for discriminating which peripheral device is to be controlled.

Port 254 (FEh) is used as an output for the BORDER colour, for the tape output and to drive the internal loud (soft?) speaker. It also handles the keyboard and the tape input. A brief résumé of this was given in the chapter on the 16K ROM, and I shall discuss it fully later in this chapter.

Port 251 (FBh) runs the ZX printer and is used for both input and output.

Port 247 (F7h) passes data for both the network and the RS232 communications input/output.

Port 239 (EFh) controls the Microdrive and the handshaking on the RS232 input/output of the Interface 1.

Port 231 (E7h) handles the Microdrive data for both read and write.

This last port is one reason for some peripherals being incompatible with the Microdrive interface. The Spectrum manual failed to mention that it would be used. I shall now consider the details of the more useful ports in turn.

## Port 254 (FEh) 1111110 BIN

The keyboard is read through this port on bits 0–4. Each line of keys is split into two sections of five keys and for each section the key on the outside is read on bit 0 and the key nearest the middle bit 4. Each bit is returned high (1) unless a key is being pressed, in which case the respective bit is low (0).

The address lines are used to determine which half row is read by an IN instruction. The bit to be set low for each line is set out below:

A0	CAPS SHIFT	to V	254 (FEh)
A1	A	to G	253 (FDh)
A2	Q	to T	251 (FBh)
A3	1	to 5	247 (F7h)
A4	0	to 6	239 (EFh)
A5	P	to Y	223 (DFh)
A6	ENTER	to H	191 (BFh)
A7	BREAK/SPACE	to B	127 (7Fh)

It is unfortunate that bits 5–7 of the data read in are set in an unpredictable manner and, in fact, the reading of the keyboard has been changed in issue 3 Spectrums, giving rise to some problems in programs which employ scruffy keyboard scanning which fails to exclude these. The Spectrum's own keyboard-scanning routine ignores them. This last fact opens up possibilities for their use for other purposes: for example as defined function keys as found on the BBC and other computers, scanned by an interrupt-driven routine.

It is possible to read more than one line of the keyboard at a time, by setting the relevant bits low before reading in the data lines, but this does not allow discrimination between the different lines of keys.

*Note:* Extreme care needs to be used when reading for more than one key pressed at a time. Sinclair have not put any protection against feedback between lines on the keyboard. This means that if, for example, the 'A', 'S' and 'W' keys are pressed at the same time, a scan of the line Q to T will show Q as being pressed, even though it is not. This is because while pressing two keys on

different address lines but with the same data bit the lines are effectively connected together. Therefore any other key pressed on either line, will set the respective bit low on both lines, so long as both the other keys are being pressed.

Bit 6 is the input from the tape socket and has a tendency to stay low, although it can be reset by an OUT instruction. Care needs to be exercised since any output to bits 0-2 will set the border colour. The safest way of setting all bits to 1 is to output 248 (F8h), but (1) it will only be temporary, and (2) it should never be necessary if proper decoding is practised.

For the output, bits 0,1 and 2 control the border colour, bit 0 controls the blue, bit 1 the red and bit 2 the green, all colours being made up from a mix of these.

Colour	Number	Binary
BLACK	0	00000000
BLUE	1	00000001
RED	2	00000010
MAGENTA	3	00000011
GREEN	4	00000100
CYAN	5	00000101
YELLOW	6	00000110
WHITE	7	00000111

Bit 3 controls the microphone socket. Remember that to get anything more than a click this needs to be turned on and off repeatedly to give a tone. The 'MIC' printed on the rear of the Spectrum is slightly misleading, since this gives output to the mic input of a tape recorder and plugging a microphone in would be a waste of time.

Bit 4 controls the loudspeaker inside the Spectrum and the same comments as above regarding use apply.

A program that demonstrates the use of the internal speaker and the 'EAR' input by reading speech or music in, storing it in memory and allowing its subsequent replay through the loudspeaker is given at the end of this chapter.

#### Port 251 (FBh) 11111011 BIN

Bit 0 INPUT is the busy line of the ZX printer, low (0) is busy.

Bit 1 OUTPUT high (1) slows the motor, low (0) speeds it up again.

Bit 2 OUTPUT low (0) starts the motor.

Bit 6 INPUT high (1) if the printer is not connected.

Bit 7 OUTPUT high (1) does the actual printing, one bit at a time.

#### Port 247 (F7h) 11110111 BIN

Bit 0 OUTPUT serial data for both network and RS232;  
INPUT serial data from network.

Bit 7 INPUT serial data from RS232.

#### Port 239 (EFh) 11101111 BIN

The write protect tab on a Microdrive cartridge can be checked by IN A,(239) AND 1; the 0 flag will be set if the cartridge is protected.

The presence of a drive can be checked by reading bit 2 of port 239, after selecting the drive to be checked; it will be reset if the drive is present.

The RS232 DTR line is on bit 3 of port 239 and the CTS line is on bit 4 of port 239.

It is most unlikely that it will be necessary to use ports 247, 239 and 231, other than from within the 8K ROM with the exception of checking the presence of an ancillary device.

The remaining ports are available for use by other ancillary devices. Remember that if a printer or other interface is connected it will use a port or ports for information transfer. Two of the most common parallel printer interfaces are the MOREX (which also has a bidirectional RS232 interface built in, and which I can highly recommend) and the Kempston, which use the following ports:

**Morex Ports 251 (FBh) 11111011 BIN & 127 (7Fh) 01111111 BIN****Port 251 (FBh) 11111011 BIN**

Bits 0-7 OUTPUT Centronics Data.

Bit 0 INPUT Centronics Busy.

Bit 1 INPUT RS232 DTR.

Bit 7 INPUT RS232 RX data.

**Port 127 (7Fh) 01111111 BIN**

Bit 0 OUTPUT Centronics strobe.

Bit 1 OUTPUT RS232 TX data.

Bit 2 OUTPUT RS232 CTS.

**Kempston Ports 58047 (E2BFh), 57535 (E0BFh) & 58303 (E3BFh)**

The Kempston interface can be used only with IN (C) and OUT (C) instructions since it looks at the full 16 bits of the address bus. Routines to send a single character to each of these interfaces Centronics output are given in the **Useful Subroutines** appendix.

**Port 58047 (E2BFh)**

Bit 0 INPUT is the busy line.

**Port 57535 (E0BFh)**

Bits 0-7 OUTPUTS Centronics data.

**Port 58303 (E3BFh)**

Bits 0-3 OUTPUTS Centronics strobe.

**Standard Streams**

Stream K -3/253 (FDh) is a duplicate of streams 0 and 1.

Stream S -2/254 (FEh) is a duplicate of stream 2.

Stream R -1/255 (FFh) is used to write to workspace, and will place the code currently in the A register into the address contained in the system variable K CUR 23643/4 (5C5B/Ch), and increment the address in K CUR. This is not as useful as it seems at first, since the make room routine is called first, and this moves up all memory above the address in K CUR as far as RAMTOP, making the stream useless for putting anything into anywhere above RAMTOP, or anywhere else that must not be moved. The routine starts at 3969 (F81h) and the stream can be used only for output; any attempt to use it for input will cause an error.

Stream 0 and Stream 1 K are normally attached to the lower screen, and the keyboard. They also allow input to be made to their channels.

Stream S 2 is for output only, normally to the screen.

Stream P 3 is for output only, normally the printer.

There are 19 streams available and each has to be attached to a channel; each stream has two bytes in the streams area of the system variables, starting at 23569 (5C10h) for stream -3, containing the displacement of the channel that it is attached to, from the base of the channels area. Remember that the first stream is -3 so that to find the address of the channel for stream 0 you will have to look at 23574/5 (5C16/7h).

A channel consists of a minimum of five bytes; the first two hold the address of the output routine, the second two the address of the input routine and the final byte the upper case letter of the channel code (S,K,P, etc.). The channels associated with the Microdrive Interface are substantially longer than the minimum and their format is shown in the handbook provided with the interface.

### Record and Play Back Routine

Pass 1 errors: 00

```

10 : THIS SHORT PROGRAM ALLOWS SPEECH
20 : TO BE RECORDED FROM THE TAPE RECORDER INTO MEMORY
30 : THROUGH THE EAR INPUT
40 : THE RECORDED SPEECH CAN THEN BE PLAYED BACK THROUGH
50 : THE SPECTRUM'S SPEAKER FROM MEMORY
60 : THIS IS A VERY CRUDE PROGRAM BUT IS QUITE EFFECTIVE
70 :
80 :
C350 90 LIMIT EDU 50000 ; THE BOTTOM LIMIT OF AVAILABLE MEMORY
0005 100 TIME EDU 5 ; THE DELAY BETWEEN EACH BIT
FDE6 110 FIRST EDU 64998 ; THE FIRST BYTE OF MEMORY TO USE
FDE6 120 DRG 65000
FDEB 21E6FD 130 START LD HL,FIRST ; CALCULATE THE AMOUNT OF MEMORY
FDEB 1150C3 140 LD DE,LIMIT ; AVAILABLE FOR USE
FDEE E5 150 PUSH HL
FDEF A7 160 AND A
FDF0 ED52 170 SBC HL,DE
FDF2 E5 180 PUSH HL
FDF3 C1 190 POP BC
FDF4 E1 200 POP HL
FDF5 F3 210 DI ; INTERRUPTS OFF TO ENSURE EVEN TIMING
FDF6 2E7F 220 LISTEN LD A,(#FE) ; WAIT FOR A SOUND
FDF8 DBFE 230 IN A,(#FE)
FDF9 CB77 240 BIT 6,A
FDFC 2BF6 250 JR Z,LISTEN
FDFE C5 260 PUSH BC
FDFE 0608 270 BYTEIN LD B,B ; READ IN 8 BITS
FE01 2B 280 DEC HL
FE02 D6FE 290 HEAR IN A,(#FE) ; A BIT AT A TIME
FE04 17 300 RLA
FE05 17 310 RLA
FE06 CB16 320 RL (HL) ; AND TRANSFER IT TO MEMORY
FE08 100A 330 DJNZ PAUSE ; PAUSE AND GET NEXT BIT
FE0A D1 340 POP DE ; BUT IF 8 BITS READ IN THEN CHECK
FE0B 1B 350 DEC DE ; ROOM AND DECREMENT COUNTER
FE0C 7A 360 LD A,D
FE0D B3 370 OR E
FE0E D5 380 PUSH DE
FE0F 20EE 390 JR NZ,BYTEIN ; IF STILL ROOM GO BACK FOR ANOTHER BYTE
FE11 C1 400 POP BC ; OTHERWISE FINISH
FE12 FB 410 EI
FE13 C9 420 RET
FE14 0E05 430 PAUSE LD C,TIME
FE16 0D 440 DELAY DEC C
FE17 20FD 450 JR NZ,DELAY
FE19 1BE7 460 JR HEAR
FE1B 21E6FD 470 SPEAK LD HL,FIRST ; THIS IS THE SAME AS FOR LISTEN
FE1E 1150C3 480 LD DE,LIMIT
FE21 E5 490 PUSH HL
FE22 A7 500 AND A
FE23 ED52 510 SBC HL,DE
FE25 E5 520 PUSH HL
FE26 C1 530 POP BC
FE27 E1 540 POP HL
FE28 C5 550 PUSH BC
FE29 F3 560 DI
FE2A 0608 570 BYTEOUT LD B,B ; OUTPUT 8 BITS
FE2C 2B 580 DEC HL
FE2D 7E 590 LD A,(HL) ; NOTE ALL BITS ARE PUT OUT AND THIS
FE2E 0F 600 RRCA ; WILL CHANGE THE BORDER COLOUR
FE2F 0F 610 RRCA
FE30 0F 620 RRCA
FE31 D3FE 630 BITOUT OUT (#FE),A ; A BIT AT A TIME, AS READ IN
FE33 07 640 RLCA
    
```

```

FE2A 0E05 590 LD C,TIME
FE2B 0D 600 DEC C
FE27 20FD 610 JR NZ,HOLD
FE29 10F6 620 DJNZ BITOUT
FE2B C1 630 POP BC
FE2C 0B 700 DEC BC
FE2D 79 710 LD A,C
FE2E B0 720 OR B
FE2F C5 730 PUSH BC
FE40 20E8 740 JR NZ,BYTEOUT
FE42 C1 750 POP BC
FE43 FB 760 EI
FE44 C9 770 RET
    
```

Pass 2 errors: 00

```

BITOUT FE21 BYTEIN FDF6
BYTEOUT FE2A DELAY FE16
FIRST FDE6 HEAR FE02
HOLD FE26 LIMIT C350
LISTEN FDF6 PAUSE FE14
SPEAK FE1B START FDEB
TIME 0005
    
```

Table used: 170 rows 12M

00000000

7

## Using the Interrupts

The start-up sequence in the Spectrum, which clears the memory and sets the system variables, also initialises the interrupt register to hold 63 (3Fh) and sets the interrupt mode to 1 (IM1). The setting of the I register appears unnecessary since the IM1 mode has no use for it — because in this mode any interrupt does a RST 56 (38h) — but the novel manner in which the ULA in the Spectrum handles the display makes bits 6 and 7 of the I register important.

On every machine instruction cycle, the Z80 executes a memory refresh operation during which the contents of the I register are put out on high eight bits of the address bus and the memory request line is activated. The ULA generates an interrupt each time it wishes to update the screen. This makes the CPU execute the interrupt service routine, assuming the interrupts are enabled. This is normally the keyboard scan and frames counter update, but if the Interface ROM is switched in all that happens is the interrupts are enabled and an immediate RETURN made, without the keyboard being scanned or anything else done.

When the interrupt routine is completed the CPU returns to whatever it was doing previously. If this involves a read or write instruction to the memory between 16384 (4000h) and 32767 (7FFFh), which the ULA checks by looking at the top two lines of the address bus, and the  $\overline{\text{MREQ}}$  line, the ULA stops the clock on the CPU while it completes the screen update.

If the I register has the top bit reset and bit 6 set the ULA gets confused, due to the refresh of dynamic memory during T3 and T4 of an M1 (instruction fetch) cycle. This activates the  $\overline{\text{MREQ}}$  line and causes the I register to be put out on the top eight bits of the address bus. The ULA then thinks that the CPU is doing a read or write operation to this area of the RAM, even though it has tried to stop this, and the ULA omits its own read to update the display, causing break-up of the picture. The I register cannot

therefore hold a value which has the top two bits set in this manner, in other words any number from 64 to 127 (40h to 7Fh) inclusive, if this break-up of the picture is to be avoided.

By setting IM2, it is possible to use the interrupts for your own purposes, so long as you do a RST 56 (38h) at the end of your own interrupt-servicing routine, which will enable the interrupts before returning to the CALLING program — if you want the keyboard scanned and frames counter updated — and end with a RETI instruction.

If you have not used the RST 56 (38h) within the interrupt routine, you must execute an EI instruction before the RETI if you want the interrupts enabled to CALL the interrupt service routine again. Remember that you will have to reset the IM1 mode and enable the interrupts before you return to BASIC unless you are using a RST 56 (38h) within the interrupt routine.

The IM2 mode is somewhat convoluted. On receipt of an interrupt, which occurs 50 (60 in the USA) times a second, the CPU saves the address of the next instruction in the program that it is executing on the machine stack, and disables any further interrupts. It then looks at the location pointed to by the data bus + (256 × the I register) and jumps to the address which is contained in this location + (256 times the contents of the following location). It is normally regarded as bad practice to have bit 0 on the data bus high for use as a pointer in IM2 since the vector should always start in an even numbered address, but unfortunately with the Spectrum there is no choice.

*Example.* The I register contains 10 (0Ah) and the data bus will hold 255 (FFh).  $10 \times 256 = 2560$  and  $2560 + 255 = 2815$ , therefore the address to be jumped to will be taken from the contents of address 2815 + (256 × the contents of address 2816). Address 2815 contains 34 and address 2816 contains 128. You can see this for yourself by PEEKing on your Spectrum, as these are in the ROM. So the address jumped to will be  $34 + (256 \times 128)$ , which is 32802.

Similarly, if the I register held 6:

$$6 \times 256 = 1536, \text{ and } 1536 + 255 = 1791.$$

$$1791 \text{ contains } 221 \text{ and } 1792 \text{ contains } 113.$$

$$\therefore 221 + (113 \times 256) = 29149, \text{ so the jump will be to } 29149.$$

Alternatively, if you have a 48K Spectrum and the I register held 200:  $200 \times 256 = 51200$ , and  $51200 + 255 = 51455$ . So the jump will be to the address that you put into this and the following location, in normal Z80 fashion low byte first.

This can be represented by imagining the interrupt as an invisible instruction in the program being run. At the moment of the interrupt the invisible instruction is executed as if it were a DI followed by a CALL instruction in the address immediately prior to the address pointed to by the I register and the data bus, the address being CALLED is in the next two bytes in the standard Z80 low byte first order. The instruction, being invisible, cannot place its own return address on the machine stack, hence the address after the last instruction executed in your program goes onto the stack, and it is this address that will be returned to after the RETI instruction at the end of the interrupt service routine.

The RETI instruction must be preceded by an EI instruction. The reason for the DI being incorporated in the CALL performed by the interrupt is to ensure that, should the interrupt service routine be longer in execution time than the delay between two interrupts, the program does not become tied up in a loop.

It is quite easy to write a program which changes the address jumped to by an interrupt by loading the vector bytes (the two addresses looked at to determine where the jump is made to) with the desired address within the program.

*Note:* Whenever interrupt routines are used it is of vital importance that any registers which are used by the interrupt routine are preserved on entry, and restored before going back to the main program. No attempt should be made to pass data to and from the interrupt routine in registers.

Because of the limitation on the values that can be held in the I register, there are only a limited number of addresses that can be jumped to in the 16K Spectrum, and these are dictated by the contents of the ROM. An added problem when using ROM-vectored interrupts is caused by the Microdrive interface which changes the vector whenever it is paged in. A list of the vectors for the Issue 2 Spectrum and the Issue 1 Microdrive interface is given in Appendix F, but if you are unsure which issue you have, or you have a different issue, these *must* be checked. For commercial software, it is dangerous to use ROM vectors as any changes and future additions could make your software unusable.

Typical uses of interrupt routines are SPRITE control and CONTINUOUS SOUND within a program. Since it is known how often an interrupt will be generated, it is easy to calculate the speed of movement for a SPRITE and, since it will be independent of any other operation within the program, the speed will normally remain constant.

The use of ROM routines within an interrupt routine is complicated by the possibility of the interface ROM being switched in at the time of the interrupt, and this *must* be allowed for when the routine is written. For example, if a SPRITE routine relies on a 16K ROM call to plot the sprite to the screen by use of the PLOTting CALL at 8933 (22E5h) when the Interface ROM is paged in, the CALL will be to 8933 in the Interface ROM. This is an address that does not exist. This will inevitably cause the program to crash.

One way of solving the problem is to incorporate into the interrupt routine a check to see which ROM is present. The easiest way of checking is to look at a ROM address which contains a different value in each ROM. I tend to use address 20 (14h), which contains 213 (D5h) in the Interface ROM and 255 (FFh) in the 16K ROM. The appropriate action for each ROM can then be taken.

If the ROM call is to the 16K ROM, it can be called directly when the 16K ROM is present and via the RST 16 instruction when the 8K ROM is paged in. For the 8K ROM, this can be used with hook code 32h (details are given in Chapter 3).

I have given a simple SPRITE program in Appendix G. This program moves a group of four pixels, bouncing them off the four edges of the screen, emitting a sound and changing the border colour, demonstrating this way round the problem.

As a full understanding of how to use interrupts is so important, if any use at all is to be made of them, I suggest that you enter the program, using your assembler. The listing is taken direct from my assembler to ensure that there are no errors; it is slightly unusual in that *hex* numbers are prefixed by a # and *binary* numbers by a % in the source code. The routine can be relocated if a new vector is calculated and the I register is changed to suit. Once you have entered and assembled the program, *before* attempting to run it, turn back to this page and read on.

The first problem you may encounter is, if you have a 16K Spectrum, that the vector will have to be ROM-based. Unfortu-

nately at the time of writing this I have found no way that a 16K Spectrum with the Microdrive Interface can be used.

Before doing anything further save both the source and the object code to tape or Microdrive and, if you have a Microdrive, remove the cartridge. To initiate the SPRITE the routine labelled SETUP in the listing must be called. If the vector has not been changed, RANDOMIZE USR 51457. You should now be able to see a single black dot moving about the screen. Should it not be there check your code again. The presence of the SPRITE should have no effect on anything else the computer is doing, so a program can be entered and run as normal, as long as it does not encroach on the memory used by the interrupt routine.

Some of the BASIC instructions that act on the interrupts can now be demonstrated. Enter the BASIC line:

```
10 BEEP 5,60 : FOR N = 1 TO 100 : NEXT N : GOTO 10
```

When this is run you will see that the SPRITE stops while the BEEP is being executed. Other commands which disable the interrupts are those involved with SAVEing and LOADing.

The SPRITE will travel at the rate of 50 pixels per second in the UK (60 pixels per second in the USA), so in the UK it will take 3.5 seconds to travel from the bottom of the screen to the top.

## Extending BASIC with Interface 1

With the addition of Interface 1, any BASIC instruction which fails the Sinclair syntax checking is normally vectored back to the error-handling routines, via the address held in the new system variable VECTOR. This can be changed to point to an address in RAM. This allows a program to be written which can check further, and act upon any instructions that it is programmed to deal with. Before any use can be made of this facility it is necessary to understand the method by which lines are checked, so that the extra routine can make the same checks.

As soon as ENTER is pressed after entering a BASIC line into the input buffer, the ROM interpreter is called into action. This RUNs the line, but stops short of actually doing the commands, because the syntax flag (bit 7 in FLAGS) is reset. If any error occurs, it is flagged with a '?', and the error must be corrected before the line can be inserted into a program.

The same process is carried out in run time but, since the syntax flag is set, the command is acted upon. Remember that whenever a routine in RAM is being used as part of the BASIC interpreter, the 8K ROM will be paged in.

There are routines in both the ROMs which check for the syntax flag and return to the calling routine only in run time. These are most useful when adding commands, as demonstrated by the routines at the end of this chapter. The basic (no pun intended) criteria for checking the syntax are given below, and ROM routines have been used as far as possible.

The address of the character at which the syntax failed, so far as the ROM is concerned, will be present in the system variable CH\_ADD on arrival at the extended syntax checker in RAM. Because of this it is important to ensure that the first character of the added BASIC command fails normal syntax. If this was not

the case, the interpreter would start execution during run time, and it would be almost impossible to regain control. For this reason, it is simplest to use a non-alphabetic character to start the additional command, which gets the system out of the normal K or command mode, ensuring syntax failure. The "\*" and "!" symbols are ideally suited to this.

The first thing that must be checked is the command characters. This is done by getting each character in turn and ensuring that it is the one expected. The 16K ROM routine at 24 (18h) will get the current character in the a register, and the routine at 32 (20h) will collect the next character and advance CH\_ADD by one. Each character can then be checked and, if it is not correct, the error handler can be called in by jumping to the original VECTOR which was 496 (1F0h).

Then:

- 1) If there should be a numeric expression next, the 16K routine at 7298 (1C82h) can be used. This evaluates the next expression as numeric; if it is not an error will be caused. In run time it will also put the value onto the calculator stack.
- 2) If there should be two numeric expressions next separated by a comma, the 16K routine at 7290 (1C7Ah) can be used. This acts as above but stacks both numbers in run time.
- 3) If there should be a string expression next (either in quotes or a \$) then the 16K routine at 7308 (1C8Ch) can be used. Again this acts as above, but generating an error if the expression is not a string, and in run time stacking the string parameters as described at the end of Chapter 8.

*Note:* In all three of the above routines, the first character of the expression to be evaluated must be in the A register, and CH\_ADD must contain its address, before CALLing the routine. After the evaluation, the A register will contain the first character after the expression and CH\_ADD its address. Mathematical operators are allowed as are the use of variables in the expressions evaluated, and this can be used to advantage to find details of variables. For example, the expression X\$(1,4 TO 9) or A\*(B + (C/(D + E))) would be allowed, the result being stacked in run time.

When evaluation is completed, a CALL 1463 (5B7h) will check that the current character is the end (a colon or carriage return code 13 (0Dh)), and return to the interpreter in syntax time, or to your routine for execution in run time, if it is, and generate an error if not.

In run time, your routine will have to act upon the command, collecting information from the calculator stack as necessary, and then return to the interpreter via a JP 1473 (5C1h) instruction, with CH\_ADD pointing to the next character for interpretation.

To demonstrate the addition of commands the following program adds !CALLnn and !FRE. !CALLnn will call a machine code routine at the address after the !CALL. !FRE will return the free memory.

```

!CALL  RST 16
        DEFW 24      ; get character
        CP  "!"     ; is it a !
        JR  NZ,ERROR ; if not allow error
        CALL NEXT_CH ; get next character
        CP  "C"     ; is it C
        JR  NZ,!FRE ; if not, jump and see
                    ; if it's a !FRE

        CALL NEXT_CH
        CP  "A"     ; each character is
        JR  NZ,ERROR ; checked, the error is
        CALL NEXT_CH ; allowed to pass if
        CP  "L"     ; a match fails
        JR  NZ,ERROR
        CALL NEXT_CH
        CP  "L"
        JR  NZ,ERROR
        JR  ISCALL ; to reach here the word
                    ; must match

!FRE   CP  "F"     ; same again for !FRE
        JR  NZ,ERROR ;
        CALL NEXT_CH
        CP  "R"
        JR  NZ,ERROR
        CALL NEXT_CH
        CP  "E"
        CALL NEXT_CH
        JR  Z,!SFRE ; it is !FRE
        JP  496     ; the original 'VECTOR'
ERROR  CALL NEXT_CH
        RST 16

```

```

DEFW 7298 ; evaluates next basic
           expression as numeric.
           causes error if not.
           In run time value is
           put on calc. stack
CALL 1463 ; checks for end of
           basic statement.
           and returns only in
RST 16 ; run time
DEFW 11682 ; stack to BC, see chap 8
JR C,ERROR ; carry set if over 65535
LD (DEST),BC; the address to CALL
RST 16
DEST DEFS 2 ; and put it here
JR FINIS ; 16K ROM will be present
           when called routine is
           executed
ISFRE CALL 1463 ; exits in syntax time
LD HL,00 ; clear HL
ADD HL,SP ; add the address in SP
LD DE,(23653);this is STKEND
SBC HL,DE ; sub from address in
PUSH HL ; SP to give free space
           for basic
POP BC ; result now in BC
RST 16
DEFW 11563 ; stack BC, see chapt 8
RST 16
DEFW 11747 ; prints value on stack
FINIS JP 1473 ; return to basic
NEXT_CH RST 16
DEFW 32 ; ROM NEXT_CH routine
AND 223 ; make it upper case
RET

```

Normally extended BASIC commands or functions will be ended by a jump back to the main interpreter as above and it is most important that it is the 8K ROM which is switched when this jump is made, otherwise the program will crash.

## The Calculator

The Spectrum contains a powerful calculator in the ROM which can be used to advantage by the machine code programmer. As it has 66 different routines, I shall be examining in detail only the more useful ones. To use the calculator it is important to understand the form in which the Spectrum handles numbers, how to place them so that the calculator can access them and how to retrieve the answer to the completed calculation.

All numbers being used by the calculator are stored as five bytes in either binary floating point representation or small integer representation.

### Small Integer Representation

The first byte is always 0.

The second byte is the sign, 255 (FFh) for negative or 0 for positive.

The third and fourth bytes are the actual number in standard Z80 format, low byte first.

The final byte is always 0.

The number 0 is regarded as being positive.

### Floating Point Representation

The first byte is the exponent: this is the number of times the binimal point has been moved to the left to make bit 7 set and to the right of the binimal point, and bit 7 of the exponent shows which way the binimal point has been moved. If it is set, the point has been moved to the left, as in the example below.

Take the number 126 decimal, 7Eh, which in binary is 01111110. The binimal point (were we to show it) would be on the

right. It takes seven moves to the left to make the most significant set bit on the right of the binimal point. To follow the process:

```
0 moves 0 1 1 1 1 1 1 0
1 move  0 1 1 1 1 1 1.0
2 moves 0 1 1 1 1 1.1 0
3 moves 0 1 1 1 1.1 1 0
and so on until after
7 moves 0.1 1 1 1 1 1 0
```

Any bits left to the left of the binimal point will always be reset and so can be discarded, as can the binimal point since it is known where it is and it will always be there for any number. We normally do this ourselves with decimal numbers, albeit unknowingly, by not showing a decimal point to the right of an integer (a whole number) as everybody knows that that is where it would be.

This process gives us the part of the number known as the *mantissa* which, for the example given, is 11111100 in binary and the exponent (the number of times the binimal point has been moved to the left) is seven. The most significant bit of the mantissa will always be set so this bit can be used to show the sign of the number; it is set for a negative number and reset for a positive number.

The *exponent* is expressed in signed binary, bit 7 set if the binimal point has been moved to the left, and reset if it has been moved to the right. Thus, for the example above the mantissa was 7, binary 00000111, but the point was moved to the left so bit 7 must be set giving 10000111 or 135 decimal. Now the number can be shown in its full five-byte form:

	Exponent	Mantissa			
Binary	10000111	01111100	00000000	00000000	00000000
Decimal	135	124	0	0	0
Hex	87	7C	0	0	0

Normally the number given in the above example would have been stored in small integer representation, but it has been used for simplicity.

The calculator uses its own stack on which it keeps any numbers upon which it is working, and the first thing that must be done before any calculations can be performed, is to put the

numbers to be manipulated onto the calculator stack. This can be tackled in three basic ways:

- 1) A number can be placed on the stack from a register or register pair, using a ROM routine to convert it to the form that the calculator requires.
- 2) The number can be changed into the form that the calculator understands and then put onto the stack.
- 3) The number can be written to memory in ASCII representation and the BASIC syntax checker used to read it and place it onto the calculator stack in the correct form.

Each method has its own advantages and disadvantages and each lends itself to different types of numbers. I shall now consider their use in turn.

- 1) For small integers, there are two routines that can be used:

**CALL 11560 (2D28h)**

This is employed by putting the number to be transferred to the calculator stack into the A register. Obviously the range is limited to 0-255 (0-FFh) and the number must be positive.

**CALL 11563 (2D2Bh)**

This will accept numbers in the range 0-65535 (0-FFFFh) from the BC register pair. Again the number can only be positive, unless the start of the routine is bypassed. The CALL is then made to 11569 (2D31h) with the A register holding 0, and the E register set to 255 (FFh) and the number will be transferred to the calculator as negative.

- 2) For numbers in five-byte form ready for use by the calculator the following routine is used:

**CALL 10934 (2AB6h)**

The five-byte representation of the number as described above in **Floating Point Representation** must be in the registers A, E, D, C, B; the exponent in the A register and the mantissa in the other four registers in order.

- 3) ASCII representation

**CALL 11419 (2C9Bh)**

Whenever the BASIC interpreter comes across a number in a BASIC line being entered, it places the five-byte binary form of the number after the ASCII version, ready for use later when the program is being run. The routine used for this can be made to do the conversion for a machine code program written by the user. This saves all the problems of converting numbers manually or writing a routine to convert them in your program, and is well worthy of detailed consideration.

To use this routine to load the calculator stack the BASIC system variable CH\_ADD 23645 (5C5Dh) must contain the address of the first character of the number to be stacked, and the A register must hold the code of the character pointed to by CH\_ADD. The first character will normally be the most significant digit of a decimal number but, because the routine is that used for BASIC line scanning, it could be the BIN token 196 (C4h) if the following digits are a binary number. A binary number can only be evaluated to 16 digits, decimal 65535 (FFFFh). Any attempt to exceed this will result in a jump to the BASIC error-handling routine. E format can be used if it is so wished, and the number should be arranged exactly as it would be in a BASIC line.

After the characters comprising the number, and the exponent if used, a defined byte of value 13 (0Dh) should be added. This lets the routine know that it has reached the end of the number and that there is nothing else to look at.

There are two ROM routines the reverse of the two routines to Stack A and Stack BC. These take the last entry on the calculator stack and compress it into a rounded integer if possible. If the number was too large then the carry flag will be set on return, and if the number is negative the 0 flag is reset. For a positive number the 0 flag will be set. The number will be deleted from the stack by changing the pointer, but the DE register pair still points to it in memory allowing it to be reclaimed if recovery was not successful, although it is easier to duplicate the number before attempting recovery, and then delete the copy on a successful operation. The call addresses are:

```
STACK TO A: CALL 11733 (2DD5H)
STACK TO BC: CALL 11685 (2DA5H)
```

There is also a reverse to the routine in (2) above located at 11249 (2BF1h) which returns the last value on the stack to the same registers. This routine also deletes the number from the stack but since it cannot fail to recover a number, the flags are not set. A duplicate must be made on the stack if the number is still required to be on the stack after recovery.

```
STACK TO A, E, D, C, B: CALL 11249 (2BF1H)
```

When numbers are generated inside the program the routine to print a number from the stack can be used to 'print' the number to a space in memory — as opposed to printing it to the screen — in ASCII form. The maximum number of spare memory locations that will be needed for a single number is 14.

This is achieved by writing a subroutine which places the contents of the A register into the next memory location each time it is called, opening a channel which points to your own subroutine, and making this channel current by putting its base address in CUR\_CHL 23633 (5C51h). The ROM routine at 11747 (2DE3h) can then be called to print the number. The subroutine will then be called with each character of the number in turn, which it places into successive memory locations, ready for use later. Remember that the indexing for the address in which the characters are being saved cannot be held in a register between calls to your subroutine, nor can it be saved on the machine stack, therefore two bytes of memory must be allowed for saving this address. A program to do this might look something like this:

```
SET_UP LD HL,SPACE
      LD (SP_WD),HL
      LD HL,(23633) ;CUR_CHL
      PUSH HL
      LD DE,START ; make current channel to
      LD C,(HL) ; your subroutine and save
      LD (HL),E ; original address for
      INC HL ; later restoration
      LD B,(HL)
      LD (HL),D
      PUSH BC
```

```
CALL 11747 ; the routine to print
           numbers
POP BC ; restore original address
POP HL ; and current channel
LD (HL),C ; restore channel
INC HL ; destination
LD (HL),B
;REST OF PROGRAM GOES HERE
START LD HL,(SP_WD)
      LD (HL),A
      INC HL
      LD (SP_WD),HL
      RET
SPACE DEFS 16
SP_WD DEFW SPACE
```

This is the routine used above and mentioned previously. When called it takes the top entry from the calculator stack and outputs it in ASCII, as a decimal number, to the current stream. The number is removed from the stack.

Whenever the calculator is used it is important to ensure that the calculator stack is kept balanced. The calculator itself always behaves in a predictable manner (when correctly used) and imbalances can lead to erroneous results. Room must also be left between RAMTOP and STKBOT for the calculator stack to expand upwards and the machine stack to grow downwards without their colliding.

Since some calculator routines use the calculator recursively, it will be necessary to leave more space than that which will apparently be used. It is best to err on the large side for the space allowed because of this fact. Should there be any doubt as to what is on the stack after completing a set of operations it can be cleared by a CALL 5823 (16BFh), but remember that this will delete *everything* that was on the stack.

## Use of the Calculator

In order to understand how to operate the calculator, it may help to think of it as a separate processor, with its own instruction set, which is switched in by a RST 40 (28h) and switched out by an ENDCALC, op-code 56 (38h).

During the time that the calculator is switched on, its op-codes are taken from the memory following the RST 40. These are a series of DEFINED Bytes in a normal Z80 program. When the calculator gets an ENDCALC instruction, control is returned to the Z80 and this continues execution of the program from the address following the ENDCALC. Some op-codes require operands, and some can be used only at the start of a series, since they require the Z80 registers to be set in a particular manner in order to operate.

Each time the stack is used its size changes by five bytes. Therefore, every time a number is placed on the stack a check is made to see that there is room. Should there be inadequate space a BASIC error will be caused.

The calculator is not limited to numeric operation, it is also used to perform the BASIC string and VAL functions. These will be discussed later.

The op-codes that the calculator understands are set out below with descriptions of their operation. In each case the change to the calculator stack is given in bytes — five bytes are used for each value — for the calculator's execution of a single op-code.

X is the value below Y on the calculator stack, Y being the last value put onto the calculator stack. The result of a calculation is always left on the top of the calculator stack, and this result is represented by Z. For example, for the subtract op-code (03) if X = 5 and Y = 9, X would be placed onto the calculator stack followed by Y, the RST 40 (28h) would be followed by a defined byte 03. After the operation, the calculator stack is -5, the answer is on the top as Z. Therefore both X and Y have been deleted.

Jumps are made from the location of the distance operand, the standard Z80 manner.

#### Op-code Function Operation

00	JUMP TRUE	Jumps the distance (2s complement notation) in the operand (the defined byte after the 00 op-code) if Y is non-zero. (STACK CH. -5)
01	EXCHANGE	Reverses the order of X and Y on the stack (STACK CH. 0)

02	DELETE	Removes Y from the stack. Z = X (STACK CH. -5)
03	SUBTRACT	$X - Y = Z$ (STACK CH. -5)
04	MULTIPLY	$X * Y = Z$ (STACK CH. -5)
05	DIVIDE	$X / Y = Z$ (STACK CH. -5)
06	TO POWER	$X ^ Y = Z$ (STACK CH. -5)
07	BINARY OR	$X \text{ OR } Y$ . If = 0 then Z = X, otherwise Z = 1 (STACK CH. -5)
08	BINARY AND	X AND Y. Both X and Y must be numbers. If Y = 0 then Z = 0, otherwise Z = X. There is a separate op-code to deal with strings 16 (10h) (STACK CH. -5)

The sequence of op-codes from 09 to 14 (0Eh) deals with numeric values. Each code can be used only as the first operation after the rst 40 (28h) as the b register must contain the op-code at the moment the actual operation is performed.

There is a second set of op-codes for string comparisons; all return Z = 1 if true or Z = 0 if false (STACK CH. -5).

09	Y <= X	10 (0Ah) Y >= X	11 (0Bh) Y <> X
12	(0Ch) Y > X	13 (0Dh) Y < X	14 (0Eh) Y = X
15	(0FH) ADDITION	$X + Y = Z$	(STACK CH. -5)

In the following sequence of op-codes, one or both of the X and Y values must hold the parameters of a string. Details of how to get these parameters and put them on the calculator stack are given later. The part or parts that are string parameters are shown with a \$ symbol.

#### Op-code Function Operation

16	(10h) \$ AND No.	$X\$ \text{ AND } Y$ . If Y = then Z\$ will be an empty string, otherwise Z\$ = X\$ (STACK CH. -5)
----	------------------	---

The op-codes 17-22 (11h-16h) are the string equivalents of op-codes 09-14 (0Eh), called with the B register holding the op-code. Again Z = 1 if true or Z = 0 if false. (STACK CH -5)

17	(11h)	Y\$ <= X\$	18	(12h)	Y\$ >= X\$	19	(13h)	Y\$ <> X\$
20	(14h)	Y\$ > X\$	21	(15h)	Y\$ < X\$	22	(16h)	Y\$ = X\$
23	(17h)	ADDITION	X\$ + Y\$ = Z\$. The two \$strings are concatenated in workspace and the new parameters are returned in Z\$. Remember that if there is inadequate room for the two strings to be copied into an expanded workspace a BASIC error will be caused (STACK CH. -5).					
24	(18h)	VAL\$	VAL\$ Y\$ = Z\$. The new string is created in workspace in the same manner as above. The op-code must also be contained in the B register at the time of use. Any errors will cause a BASIC error. This is the routine used by the BASIC interpreter and is subject to all the syntax checking procedure (STACK CH. 0)					
25	(19h)	USR\$	Z = USR\$. Again used by the BASIC interpreter and subject to syntax checking. Y\$ must contain the parameters of a \$string containing a single letter from A to U. Z will be the address of that user-defined graphic on completion (STACK CH. 0)					
26	(1Ah)	READ IN	This routine allows a single byte to be put into WORKSPACE through any stream 0-15 (0Fh), taken from Y. The byte is regarded as a string and Z\$ is the parameters of this string. If the carry flag is not set by the input routine, no action is taken and Z\$ is returned as a null string					
27	(1Bh)	NEGATE	Z = Y but with the sign changed (STACK CH. 0)					

28	(1Ch)	CODE	Z = CODE Y\$. As used by the BASIC interpreter (STACK CH. 0)					
29	(1Dh)	VAL	Z = VAL Y\$. As used by the BASIC interpreter and in fact accesses the line scanner to get the result into Z. Subject to BASIC errors. (STACK CH. 0)					
30	(1Eh)	LEN	Z = LEN Y\$. This is easier to do without using the calculator, all it does is stack the length bytes of the string parameters (STACK CH. 0)					

The following algebraic functions all return the answer (Z) on the top of the calculator stack and the stack size is unchanged.

<i>Op-code</i>	<i>Operation</i>	
31 (1Fh)	Z = SIN Y	
32 (20h)	Z = COS Y	
33 (21h)	Z = TAN Y	
34 (22h)	Z = ASN Y	
35 (23h)	Z = ACS Y	
36 (24h)	Z = ATN Y	
37 (25h)	Z = LN Y	
38 (26h)	Z = EXP Y	
39 (27h)	Z = INT Y	
40 (28h)	Z = SQR Y	
41 (29h)	Z = SGN Y	
42 (2Ah)	Z = ABS Y	
43 (2Bh)	PEEK	Z = PEEK Y. As used by the BASIC interpreter (STACK CH. 0)
44 (2Ch)	IN	Z = IN Y. This performs the Z80 IN A,(C) instruction after taking Y from the stack into the BC register pair as an integer (STACK CH. 0)
45 (2Dh)	USR No.	<i>Caution</i> This will cause an execution jump to the address Y. Used by BASIC to jump to machine code. The return address will be 11563 (2D2Bh), the stack BC routines

		which will put the value in the BC register on return on the calculator stack. This is an interesting op-code since it opens up the possibility of using routines in ROM and RAM recursively from within the calculator (STACK CH. 0)	
46	(2Eh) STR\$	Z\$ = Y. The top value on the calculator stack is printed to WORKSPACE and evaluated as a string, the parameters of which are then put on the calculator stack (STACK CH. 0)	
47	(2Fh) CHR\$	Z\$ = CHR\$ Y. If $0 < Y < 255$ then a single space is made in WORKSPACE and the value transferred there as one byte. This is then interpreted as a string and the parameters returned as Z\$ (STACK CH. 0)	
48	(30h) NOT	Z = 1 if Y = 0, otherwise Z = 0 (STACK CH. 0)	
49	(31h) DUPLICATE	Z = Y. Y is duplicated (STACK CH. +5)	
50	(32h) X MOD Y	This returns Z = INT (X/Y) and underneath Z (where X was originally) X - INT (X/Y) (STACK CH. 0)	
51	(33h) JUMP	This simply does the equivalent of a Z80 JR (Jump Relative) instruction, but for the calculator. The jump length is taken from the location following the op-code. The calculator stack is untouched (STACK CH. 0)	
52	(34h) STK DATA	This allows a value to be read in from the locations following the op-code. Bits 6 and 7 of the first byte (the exponent) following the op-code determines the number of	
			bytes which follow to make the mantissa, 00 BIN for 1 to 11 BIN for 4. The floating point number is then read to the calculator stack (STACK CH. +5)
53	(35h) DEC JR NZ	This is a direct equivalent to the Z80 DJNZ op-code, but for the calculator. The B register is taken as being BREG system variable. <i>Not safe to use</i> , since this routine is extensively used within the calculator for its own purposes, and the value of BREG cannot be guaranteed (STACK CH. 0)	
54	(36h) Y < 0	Z = 1 if Y < 0, otherwise Z = 0 (STACK CH. 0)	
55	(37h) Y > 0	Z = 1 if Y > 0, otherwise Z = 0 (STACK CH. 0)	
56	(38h) ENDCALC	Returns control to the Z80 at the next address (STACK CH. 0)	
57	(39h) GET ARG1	Used internally to the calculator to establish the value Y (which for the purposes of Z being the result will be Z) of SIN Y or COS Y (STACK CH. 0)	
58	(3Ah) TRUNCATE	Z = INT Y, where Z is Y truncated towards 0 (STACK CH. 0)	
59	(3Bh) FP CALC 2	Used by the interpreter to perform a single calculator operation. Of no practical use for the purposes of this book	
60	(3Ch) E TO F.P.	Z = Y E (Exponent) A register (STACK CH. 0)	
61	(3Dh) RE STACK	Z = the floating point version of Y, where Y could have been a small integer (STACK CH. 0)	
62	(3Eh) SERIES	Used internally to the calculator to generate a Chebyshev polynomial. Of no practical use as it is called by the routines that need it	

- 63 (3Fh) STACK NO. As above, but used to stack constants
- 64 (40h) ST MEM. Used to store in the memory area. On entry the A register must hold C0 to C5 according to which of the five memory locations is to be used (STACK CH. -5)
- 65 (41h) REC MEM. Recall from memory. The reverse of the above routine (STACK CH. +5)

For string operations the parameters can be passed to the calculator stack using the routine at 10934 (2AB6h) mentioned earlier. The BC register pair holds the length of the string, the DE register pair the start address and the A register the name (in the form mentioned in chapter 2 under SAVEing LOADing and VERIFYing.

To assist with the understanding of the calculator a routine that will allow experimentation with the calculator is given in the **Useful Subroutines** appendix.

# Hex to Decimal Conversions

## MSB

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	256	512	768	1024	1280	1536	1792	2048	2304	2560	2816	3072	3328	3584	3840
1	4096	4352	4608	4864	5120	5376	5632	5888	6144	6400	6656	6912	7168	7424	7680	7936
2	8192	8448	8704	8960	9216	9472	9728	9984	10240	10496	10752	11008	11264	11520	11776	12032
3	12288	12544	12800	13056	13312	13568	13824	14080	14336	14592	14848	15104	15360	15616	15872	16128
4	16384	16640	16896	17152	17408	17664	17920	18176	18432	18688	18944	19200	19456	19712	19968	20224
5	20480	20736	20992	21248	21504	21760	22016	22272	22528	22784	23040	23296	23552	23808	24064	24320
6	24576	24832	25088	25344	25600	25856	26112	26368	26624	26880	27136	27392	27648	27904	28160	28416
7	28672	28928	29184	29440	29696	29952	30208	30464	30720	30976	31232	31488	31744	32000	32256	32512
8	32768	33024	33280	33536	33792	34048	34304	34560	34816	35072	35328	35584	35840	36096	36352	36608
9	36864	37120	37376	37632	37888	38144	38400	38656	38912	39168	39424	39680	39936	40192	40448	40704
A	40960	41216	41472	41728	41984	42240	42496	42752	43008	43264	43520	43776	44032	44288	44544	44800
B	45056	45312	45568	45824	46080	46336	46592	46848	47104	47360	47616	47872	48128	48384	48640	48896
C	49152	49408	49664	49920	50176	50432	50688	50944	51200	51456	51712	51968	52224	52480	52736	52992
D	53248	53504	53760	54016	54272	54528	54784	55040	55296	55552	55808	56064	56320	56576	56832	57088
E	57344	57600	57856	58112	58368	58624	58880	59136	59392	59648	59904	60160	60416	60672	60928	61184
F	61440	61696	61952	62208	62464	62720	62976	63232	63488	63744	64000	64256	64512	64768	65024	65280

LSB

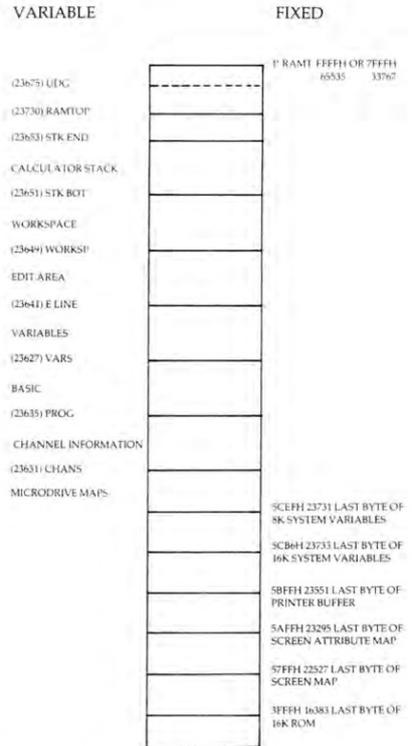
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

NIBBLES

HEX	DEC	BIN	HEX	DEC	BIN
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	A	10	1010
3	3	0011	B	11	1011
4	4	0100	C	12	1100
5	5	0101	D	13	1101
6	6	0110	E	14	1110
7	7	0111	F	15	1111

Appendix B

# The Spectrum Memory Map





## The Keyboard Map

	D4	D3	D2	D1	D0
254 (FEH)	V	C	X	Z	C/S
253 (FDH)	G	F	D	S	A
251 (FBH)	T	R	E	W	Q
247 (F7H)	5	4	3	2	1
239 (EFH)	6	7	8	9	0
223 (DFH)	Y	U	I	O	P
191 (BFH)	H	J	K	L	ENT
127 (7FH)	B	N	M	S/S	B/S

## The Spectrum Character Set

Code	Character	Special Notes
0&1	None	Used only after INK, PAPER, OVER, INVERSE, FLASH, BRIGHT AT or TAB
2-5	None	Used only after INK, PAPER, AT or TAB
6		Does a TAB on the screen to the next half screen position.
7	EDIT	Code returned by the keyboard input routine is used if CAPS SHIFT and 1 are pressed. Not printable. Often the BELL code on printers and terminals
8	BACKSPACE	Code returned by the keyboard input routine if CAPS SHIFT and '5' are pressed. Can be printed to give a backspace on the screen. Also recognised by most printers, as it is the ASCII backspace code
9	RIGHTSPACE	Code returned by the keyboard input routine if CAPS SHIFT and '8' are pressed. Not usable to give a screen rightspace as the Spectrum does not update the print positions after its use. ASCII horizontal TAB code
10	DOWNSPACE	As above but for CAPS SHIFT and '6'. Is the ASCII code for line feed
11	UPSPACE	As above for CAPS SHIFT and '7'. ASCII vertical TAB code

12	DELETE	As above for CAPS SHIFT and '0'. ASCII Form Feed code
13	ENTER	Returned when the ENTER key is pressed, performs a carriage return and line feed when printed to the screen. Is also the ASCII code for carriage return
14		Precedes a number in a BASIC program line. Not of any practical use. ASCII Shift Out code
15		Not used in the Spectrum. ASCII Shift In code
16		Ink control code used to precede the number for the INK colour. For example to change the on-screen printing colour for all following characters to red, you would use the RST 16 routine with the A register holding 16 followed by 2. Note <i>NOT</i> ASCII 2 but value 2. This is demonstrated in the DeBASE program.
17		As above but for PAPER
18		As above for FLASH, the following code can be only 0 or 1
19		As above for BRIGHT
20		As above for INVERSE
21		As above for OVER
22		AT control code, must be followed by the line and column values
23		As above but TAB, needing only the column value

Codes 24 to 31 are not used by the Spectrum; code 27 however is the escape code which is used by most peripherals, followed by a letter to indicate the action required. While the CHR\$ 27 has been standardised as the escape code, the codes that follow it have *not* been standardised.

The remaining codes are all character representations, with

codes 32 to 126 conforming to standard ASCII and these are shown in the Spectrum manual.

Code 127 — the Spectrum copyright symbol — is the ASCII delete code, *be careful!!*

## Appendix F

**ROM Interrupt Vectors**

	16K ROM Mk 2 Spectrum	8K ROM Mk 1 Interface
I = 0	20430	23755
I = 1	52818	8501
I = 2	22269	25888
I = 3	39020	4196
I = 4	10419	52486
I = 5	2294	6701
I = 6	29149	51711
I = 7	16039	32459
I = 8	2088	14353
I = 9	65129	58170
I = 10	22802	1200
I = 11	58889	2039
I = 12	52183	59861
I = 13	52503	8205
I = 14	14367	49921
I = 15	27928	58884
I = 16	51984	32742
I = 17	8729	32477
I = 18	52481	4821
I = 19	49749	21965
I = 20	25705	56790
I = 21	51673	45182
I = 22	51568	65535
I = 23	12493	57851
I = 24	15582	61947
I = 25	23842	60952
I = 26	13824	52189
I = 27	7306	16129
I = 28	49947	6400
I = 29	2344	4870
I = 30	26575	65535
I = 31	3360	57855
I = 32	52513	23755
I = 33	33485	8501
I = 34	544	25888
I = 35	49537	4196
I = 36	8527	52486
I = 37	23670	6701
I = 38	20448	51711
I = 39	288	32459
I = 40	32348	14353
I = 41	58154	58170
I = 42	19754	1200
I = 43	23653	2039
I = 44	7117	59861
I = 45	55781	8205
I = 46	23713	49921
I = 47	4569	58884
I = 48	60208	32742
I = 49	57640	32477
I = 50	13627	4821

	16K ROM Mk 2 Spectrum	8K ROM Mk 1 Interface
I = 51	13256	21965
I = 52	1560	56790
I = 53	57124	45182
I = 54	34307	65535
I = 55	41231	57851
I = 56	65535	61947
I = 57	65535	60952
I = 58	65535	52189
I = 59	65535	16129
I = 60	255	6400
I = 61	0	4870
I = 62	285	65535
I = 63	60	255

## Useful Subroutines

### Calculator Routines

To assist with the understanding and use of the calculator the following routines will allow experimentation with the calculator. The first routine is for demonstration purposes.

```

EXX          ; YOU MUST ALWAYS SAVE
             ; H'L'
PUSH HL     ; IF A SUCCESSFUL RETURN IS
             ; TO
EXX         ; BE MADE
CALL PRSTK  ; PRINT STACK START AND
             ; END
LD BC,X     ; FIRST NUMBER
PUSH BC     ; SAVE IT
CALL 2D2BH ; STACK X
POP BC      ; RESTORE X
CALL 2D2BH ; STACK X AGAIN
CALL 2DE3H ; PRINT X
LD A,20H   ; PRINT A SPACE
RST 10H    ;
LD HL,NUMBER ; BASE ADDRESS OF THE
             ; NUMBER IN ASCII FORM
CALL STKNUM ; STACK NUMBER (THIS WILL
             ; BE Y)
RST 28H    ; CALL THE CALCULATOR
DEFB 31H   ; DUPLICATE Y
DEFB 38H   ; TURN OFF THE
             ; CALCULATOR
CALL 2DE3H ; PRINT Y
RST 28H    ; TURN ON CALCULATOR

```

YOU CAN PUT A SERIES OF DEFINED BYTES HERE TO EXPERIMENT WITH THE CALCULATOR

```

DEFB 38H          ; END CALC
CALL 2DE3H        ; PRINT RESULT
CALL PRSTK        ; PRINT STKBOT AND STKEND
                  ; TO SEE IF THE STACK WAS
                  ; BALANCED
EXX               ; RESTORE H'L'
POP HL
EXX
RET

NUMBER
DEFM "1234.567"
DEFB 13          ; THERE MUST ALWAYS BE A
                  ; DEFB 13 TO LET THE
                  ; ROUTINE KNOW THAT IT
                  ; HAS FINISHED THE NUMBER

```

This subroutine will print the address of the start and end of the calculator stack:

```

PRSTK LD BC,(23651) ; THIS IS STKBOT
      CALL 2D2BH   ; STACK IT
      LD A,"B"
      RST 10H
      LD A," "
      RST 10H
      CALL 2DE3H  ; PRINT STKBOT
      LD A," "
      RST 10H
      LD A,"T"
      RST 10H
      LD A," "
      RST 10H
      LD BC,(23653) ; THIS IS STKEND
      CALL 2D2BH
      CALL 2DE3H   ; PRINT STKEND
      LD A,0DH
      RST 10H
      RET

```

This routine will stack a number in ASCII form.

ON ENTRY THE HL REGISTER PAIR MUST CONTAIN THE ADDRESS OF THE START OF THE NUMBER TO BE STACKED

```

STKNUM LD    DE, (23645) ; THIS IS CH_ADD
        PUSH DE          ; SAVE IT
        LD    (23645), HL ; POINT CH_ADD TO THE
                        ; START OF THE NUMBER
        LD    A, (HL)    ; PUT THE FIRST
                        ; CHARACTER IN A
        CALL 2C9BH      ; STACK THE NUMBER
        POP  DE         ; RESTORE CH_ADD TO
                        ; ITS ORIGINAL
        LD    (23645), DE ; VALUE
        RET

```

## Morex and Kempston Interface Drivers

On entry to these subroutines the ASCII code to be output to the Centronics lines should be in the A register. No expansion will be made; see the section in Chapter 2 on EXPANDING TOKENS FOR OUTPUT if it is required.

This is the Morex Interface Centronics Output Routine:

```

        PUSH AF          ; SAVE CHARACTER
                        ; CODE
BUSY   IN    A, (0FBH)   ; READ BUSY LINE FROM
                        ; PRINTER
        AND  1           ; TEST BIT 0
        JR   NZ, BUSY   ; IF NOT LOW PRINTER
                        ; BUSY
        POP  AF          ; RESTORE CHARACTER
        OUT (0FBH), A   ; SEND IT
        LD  A, 1        ; STROBE PRINTER

```

```

OUT    (7FH), A
XOR   A
OUT    (7FH), A
RET                                ; FINISHED

```

This is the Kempston Interface Centronics Output Routine:

```

        PUSH BC          ; SAVE BC REGISTER PAIR
                        ; AS THEY ARE USED FOR
                        ; I/O
        PUSH AF          ; SAVE CHARACTER
                        ; CODE
        LD    BC, 0E2BFH ; BUSY PORT
BUSY   IN    A, (C)     ; READ BUSY LINE
        RRA          ; BIT 0 TO CARRY
        JR   C, BUSY   ; IF NOT LOW PRINTER
                        ; BUSY
        POP  AF          ; RESTORE CHARACTER
        DEC  B          ; CHANGE PORT TO 02BFH
        DEC  B
        OUT (C), A     ; SEND CHARACTER
        LD  A, 00EH    ; STROBE DATA FOR
                        ; SENDING
        LD  B, 0E3H    ; CHANGE PORT TO
                        ; E3BFH, STROBE PORT
        OUT (C), A     ; STROBE PRINTER
        INC  A
        OUT (C), A
        POP  BC          ; RESTORE BC REGISTER
                        ; PAIR
        RET                                ; FINISHED

```

## Interrupt Driven Sprite Routine

Pass 1 errors: 00

```

10 : DEMONSTRATION PROGRAM FOR SPRITES
20 : SEE CHAPTER 6
30 :
40 : ORG $1455
50 : ENT $145F
60 :

```

```

70 : 51455 CONTAINS THE INTERRUPT VECTOR
80 :
C8FF 2CC9 90 : DEFW 51500
C901 2EC8 110 SETUP LD A,200
C903 ED47 120 LD I,A
C905 ED48B3C9 130 LD BC,(COORD)
C909 CD87C9 140 CALL PLOT
C90C E05E 150 JM 2
C90E C9 160 RET
170
180 : THIS IS THE ACTUAL PROGRAM, STARTING AT THE ADDRESS
190 : WHERE THE INTERRUPT HAS BEEN VECTORED
200
C92C 210 ORG 51500
220 :
230 : FIRST SAVE ALL REGISTERS THAT WILL BE USED
240 :
C92C E5 250 PUSH HL
C92D C5 260 PUSH BC
C92E D5 270 PUSH DE
C92F F5 280 PUSH AF
290
300 : SAVE TV FLAG AS IT MAY BE ALTERED LATER
310 :
C930 2A3C5C 320 LD A,(23612)
C933 F5 330 PUSH AF
340
350 : AND SAVE CO_ORDS AS THEY WILL BE CHANGED AND THEY MUST BE
360 : RESTORED BEFORE RETURNING
370 :
C934 2A7D5C 380 LD HL,(23677)
C937 E5 390 PUSH HL
C938 ED48B3C9 400 LD BC,(COORD) : (THE SPRITE CO-ORDINATES)
C93C C5 410 PUSH BC : SAVE THEM
420 : CALL THE ROUTINE TO PLOT THE SPRITE
430 : AS EACH PLOT OF THE SPRITE IS OVER : THIS WILL ERASE
440 : THE PREVIOUS POSITION
C93D CD87C9 450 CALL PLOT
C940 C1 460 POP BC : RESTORE THE SPRITE COORDS
470 :
480 : I IS USED AS A FLAG REGISTER
490 : D TELL THE SPRITE WHICH WAY IT MUST MOVE
500 : BIT 0 IS SET FOR UP, RESET FOR DOWN
510 : BIT 1 IS SET FOR LEFT AND RESET FOR RIGHT
520 :
C941 2A85C9 540 LD HL,(FLAG)
C944 C845 550 BIT 0,L
C946 2847 560 JR 2,UP
C948 05 570 DOWN DEC B : MOVE SPRITE DOWN 1 PIXEL
C949 234C 580 JR NZ,LEFT : IF ZERO FLAG SET IT HAS REACHED THE
C94B C885 590 RES 0,L : BOTTOM OF THE SCREEN AND ITS
C94D 1848 600 JR LEFT : DIRECTION IS CHANGED
C94F 04 610 UP INC B : UP ONE PIXEL
C950 78 620 LD A,B : HAS SPRITE REACHED TOP ?
C951 FE4E 630 CP 174
C952 2302 640 JR NZ,LEFT
C955 C8C5 650 SET 0,L : IT HAS SO CHANGE DIRECTION
C957 C84D 660 LEFT BIT 1,L : SAME CHECKS AGAIN FOR LEFT/RIGHT
C959 2807 670 JR 2,RIGHT
C95B 0D 680 DEC C
C95C 200C 690 JR NZ,CPLD
C95E C88D 700 RES 1,L
C960 1848 710 JR CPLD
C962 0C 720 RIGHT INC C
C963 78 730 LD A,C
C964 FEFE 740 CP 254
C966 2002 750 JR NZ,CPLD
C968 C8CD 760 SET 1,L

```

```

770
780 : THE NEW CO-ORDINATES FOR THE SPRITE ARE SAVED
790 :
C96A ED48B3C9 800 CPLD LD (COORD),BC : AND SO ARE THE FLAGS
C96E 2285C9 810 LD (FLAG),HL
820
C971 CD87C9 830 CALL PLOT : PLOT THE NEW SPRITE POSITION
840 :
850 :
860 : ALL THE PLOTTING IS DONE SO THE SYSTEMS VARIABLES
870 : ARE NOW RESTORED TO THEIR ORIGINAL STATE
880 :
890 :
C974 E1 900 NRET POP HL
C975 227D5C 910 LD (23677),HL
C978 F1 920 POP AF
C979 223C5C 930 LD (23612),A
940
950 : THE REGISTERS ARE NOW RESTORED
C97C F1 960 POP AF
C97D 01 970 POP DE
C97E C1 980 POP BC
C97F E1 990 POP HL
1000
1010 : AND THE KEYBOARD SCANNED AS IT HAS NOT BEEN
1020 : BECAUSE THE INTERRUPT WAS RELOCATED
1030 :
C980 FF 1040 RST #28
C981 ED4D 1050 RETI
C982 0060 1060 COORD DEFW 0
C985 0300 1070 FLAG DEFW 0
1080 :
1090 : D IS GOING TO BE USED AS A COUNTER, AS B IS IN USE
1100 :
1110 :
C987 484 1120 PLOT LD B,4
1130 :
1140 :
1150 : SAVE THE CO-ORDINATES AS THEY ARE CORRUPTED BY THE R.O.M.
1160 : ROUTINES
1170 :
1180 :
C989 C5 1190 PUSH BC
C98A 7A 1200 LOOP LD A,D
C98B D5 1210 PUSH DE : SAVE THE COUNTER
1220
1230 : THE ACTUAL PLOT IS MADE FOUR TIMES, MOVING EACH TIME
1240 : TO CREATE A LARGER SPRITE
C98C FE04 1250 CP 4
C98E 284F 1260 JR 2,LOOP4
C990 FE12 1270 CP 2
C992 2001 1280 JR NZ,LOOP1
C994 0C 1290 INC C
C995 FE02 1300 LOOP1 CP 2
C997 2001 1310 JR NZ,LOOP2
C999 04 1320 INC B
C99A FE01 1330 LOOP2 CP 1
C99C 2001 1340 JR NZ,LOOP4
C99E 0D 1350 DEC C
1360
1370 : A TEST MUST NOW BE MADE TO SEE WHICH R.O.M. IS PRESENT
1380 :
C99F 2A1400 1390 LOOP4 LD A,(#144) : PICK A R.O.M. LOCATION THAT IS
1400 :
1410 :
1420 : DIFFERENT IN EACH R.O.M.
C9A2 FDC0286 1420 RES 0,(IY+2) : SIGNAL COLOURS TO BE FROM MAIN SCREEN
C9A6 C5 1430 PUSH BC
C9A7 FED5 1440 CP #D5 :
C9A9 290C 1450 JR 2,RDM2 : R.O.M. 2 IS FLAGGED IN SO DONT CALL THE
1460 : MAIN R.O.M. ROUTINE DIRECT

```

```

1470
1480 : TO GET HERE THE MAIN R.O.M. MUST BE PAGED IN
1490 :SD DIRECT CALL D.K.
1500
C9A6 CDAB00 1510 CALL #DAD : COLLECT THE COLOURS FOR THE PLOT
C9A8 FDC857C6 1520 SET 0,(IY+B7) : SET OVER 1 WITH F FLAG
C9B2 CDE522 1530 CALL B933: DO THE PLOT
C9B5 1B08 1540 JR CONT1: JUMP THE INDIRECT BN TO 1st CALLS
1550
1560 : THE INTERFACE R.O.M. IS PAGED IN IF THIS IS REACHED
C9B6 4E0D 1570 RMO2 RST 16: THE 16 K ROUTINES MUST BE
C9BA FDC857C6 1580 DEFW #DAD : CALLED INDIRECTLY
C9BE 07 1600 RST 16
C9BF FDC857B6 1610 CDNT1 RES 0,(IY+B7): RESET OVER 1
C9C2 C1 1620 POP BC : RESTORE LATEST PLOT POSITION
C9C4 01 1630 POP DE : AND THE COUNTER
C9C5 15 1640 DEC D
C9C6 20C2 1650 JR NZ,LOOP
1660
C9C8 C1 1670 POP BC : RESTORE STARTING PLOT POSITION
C9C9 7B 1680 LD A,B : AND SEE IF AN EDGE HAS BEEN
C9CA A7 1690 AND A : REACHED
C9CB 2B0D 1700 JR Z,PING : AND IF SO TAKE THE NECESSARY
C9CD FEAE 1710 DF 17A : ACTION
C9CF 2B09 1720 JR 3,PING
C9D1 7B 1730 LD A,C
C9D2 A7 1740 AND A
C9D3 2B05 1750 JR Z,PING
C9D5 FEFE 1760 DF 25A
C9D7 2B01 1770 JR Z,PING
C9D9 C9 1780 RET : OTHERWISE RETURN
1790
C9DA C5 1800 PING PUSH BC : SAVE THE PLOT CO-ORDINATES
C9DB 0600 1810 LD B,0 : SO THAT B CAN BE USED AS A COUNTER
C9DD 7B 1820 PINGL LD A,B : SEND OUTPUT TO THE SPEAKER
1830
1840 :SEND OUTPUT TO THE SPEAKER
C9DE D3FE 1850 OUT (#FE),A : REPEATEDLY CHANGING THE STATE
C9E0 10FB 1860 DJNZ PINGL : TO MAKE A NOISE
C9E2 C1 1870 POP BC
1880
1890 :NOW SEE WHERE THE CALL CAME FROM
1900 :
C9E3 2174C9 1910 LD HL,#RET
C9E5 7D 1920 LD A,L
C9E7 E1 1930 POP HL
C9E8 E5 1940 PUSH HL
C9E9 AD 1950 XOR L
1960
1970 : IF THE RETURN WAS NOT MADE HERE EVERY ALTERNATE TIME
1980 : THIS ROUTINE WAS CALLED ONLY THE EVEN NUMBER COLOURS
1990 :WOULD BE SET
2000 :
C9EA C8 2010 RET Z
2020
2030 : INCREASE THE BORDER COLOUR. BITS 0-2
2040 :
C9EB 2AF5C9 2050 LD A,(COL)
C9EE 2C 2060 INC A
C9EF 2AF5C9 2070 LD (COL),A
2080 :
2090 : AND SET THE BORDER
2100 :
C9F2 D3FE 2110 OUT (#FE),A
C9F4 C9 2120 RET
C9F5 00 2130 CDL DEFE 0
    
```

Pass 1 errors: 00

```

COL C9F5 CONT1 C9BF
COORD C9E7 CPLEDT C9AA
DOWN C946 FLAG C9B5
LEFT C957 LOOP C9BA
LOOP1 C995 LOOP2 C99A
LOOP4 C94F NRET C974
PING C9DA PINGL C9D0
PLOT C9B7 RIGHT C9A2
RMO2 C9B7 SETUP C901
UP C94F
    
```

Table used: 22B from 680

### DeBASE

DeBASE is made up from a number of separate routines building up to a crude, but efficient, data base program, with routines for saving to and loading from both tape and Microdrive.

The program will allow entries to be input, printed out, found, erased and changed. There is no constraint on the number of separate records nor on the size of each record, except that it must fit on the screen without scrolling it. Any part of any record may be found, and the cursor will be placed at the start of the details that were looked for. A record may be changed, enlarged or shrunk after its original creation, without destroying or altering any other records. After a record deletion all space used by the record is reclaimed.

The program uses only 1 byte of memory per character plus 1 byte per record for the end marker of the record; 32 000 bytes are available for records as written, with plenty of room for added facilities.

Pass 1 errors: 00

```

10 : THIS SUBROUTINE WILL PRINT ANYTHING WHICH STARTS WITH
20 : A BYTE WITH BIT 7 SET UP TO THE NEXT BYTE WITH BIT 7 SET.
30 : IT HAS THREE MAIN ENTRY POINTS AND TWO METHODS OF USE
40 : IF THE A REGISTER HOLDS 255 (#FF) ON ENTRY, THE DE REGISTER
50 : MUST HOLD THE ADDRESS OF THE FIRST CHARACTER OF A MESSAGE
60 : WHICH MUST BE PRECEDED BY A BYTE WITH BIT 7 SET.
70 : FOR ALL OTHER VALUES OF A THE MESSAGE OF THAT NUMBER
80 : FROM A TABLE OF MESSAGES STARTING WITH A MARKER BYTE
90 : WILL BE PRINTED
100 :
110 :
F424 120 MC1 DRG 62500
F425 F5 150 PMESS PUSH AF : THIS ENTRY POINT SETS THE PRINT POSITION
F426 D5 160 PUSH DE : TO THE TOP LEFT OF THE SCREEN
F426 3E02 170 LD A,2
    
```

```

F428 CD0116 180 CALL #1601 : ENSURE MAIN SCREEN CURRENT, OTHERWISE
190 : THE PRINT POSITION COULD BE SET
200 : FOR THE WRONG SCREEN, LEAVING THE MAIN
210 : SCREEN AT ITS PREVIOUS POSITION
F428 012118 220 LD BC,#1821
F42E CD090D 230 CALL 3545 : USE THE R.O.M. CALL DETAILED IN CHAPTER 2
F431 D1 240 POP DE
F432 F1 250 POP AF
251 :
252 :
F437 F5 260 PRINAT PUSH AF : THIS ENTRY LEAVES THE CURRENT PRINT
F438 D5 270 PUSH DE : POSITION AND USES THE MAIN SCREEN
F43E 3E02 280 LD A,2
F437 CD0116 290 CALL #1601
F43A 3EFF 320 LD A,255 : SET SCROLLS BEFORE scroll1 ? MESSAGE
F43C 328C5C 330 LD (23692),A : THIS IS THE SYSTEM VARIABLE SCR CT.
F43F 3E0D 340 LD A,13 : ENTER (CARRIAGE RETURN AND LINE FEED)
F441 D7 350 RST 16 : THE PRINT RESTART
F442 D1 360 POP DE
F443 F1 370 POP AF
F444 3C 380 PRINT INC A : THIS ENTRY USES THE CURRENT STREAM
F445 18 390 DEC DE : SET DE TO POINT TO THE START MARKER
F446 2B04 400 JR Z,PRINTP : IF A WAS #FF THEN USE THE ADDRESS IN DE
F44B 3D 410 DEC A
F449 110000 420 LD DE,MESS-1 : POINT DE TO THE START OF THE MESSAGES
F44C CD0A0C 430 PRINTP CALL 30B2 : PRINT MESSAGE R.O.M. ROUTINE
F44F C9 440 RET
0001 450 MESS. EQU 1
    
```

Pass 2 errors: 00

```

MC1 C256 MESS 0001
PMESS F424 PRINAT F433
PRINT F444 PRINTP F44C
    
```

Pass 1 errors: 00

```

450 : THIS SUBROUTINE WILL SEARCH MEMORY FOR A STRING WHICH
460 : MATCHES THE STRING STARTING AT SCHAR (SEARCH CHARACTER)
470 : AND ENDING WITH A MARKER BYTE OF 255 (MFF)
480 : IF A MATCH IS FOUND THE START OF THE RECORD CONTAINING
490 : THE MATCH IS THEN FOUND, BY STEPPING BACK TO A BYTE
500 : WITH BIT 7 SET. IT IS THEN PRINTED BY THE PREVIOUS
510 : SUBROUTINE.
520 :
530 : IF THE END OF EACH RECORD IS MARKED WITH A #BD
540 : THEN IF THE CONTENTS OF SCHAR IS #FF (IE FIND NOTHING)
550 : EACH RECORDS END WILL BE FOUND INSTEAD
560 :
570 : ON EXIT THE BC REGISTER PAIR WILL EITHER HOLD THE ADDRESS
580 : OF THE START OF THE RECORD CONTAINING THE STRING
590 : OR 0 IF THE STRING WAS NOT FOUND
600 :
610 :
F450 620 MC2 ORG #F450
F450 21CBF4 630 FINDIT LD HL,SCHAR : THE MAIN ENTRY POINT
F453 7E 640 LD A,(HL)
F458 FEFF 650 CP #FF
F45E 2006 660 JR NZ,SEARCH
F45B 23 670 INC HL
F459 77 680 LD (HL),A
F45A 28 690 DEC HL
F45B 3E8D 700 LD A,#BD
F45D 77 710 LD (HL),A
F45E E9 720 SEARCH PUSH HL
F45F 213275 730 LD HL,30092
F462 01D0B4 740 LD BC,34000
    
```

```

F465 D1 750 POP DE
F466 D5 760 LOCKON PUSH DE
F467 EDB1 770 LOCKON CPJR : LOOK FOR THE FIRST CHARACTER
F469 2046 780 JR NZ,NOTFND : IF THE STRING IS NOT FOUND
F46B 2B 790 DEC HL
F46C 22EEF4 800 LD (LOCEND),HL : SAVE THE LOCATION WHERE FOUND
F46F ED43ECF4 810 LD (LOCEND),BC : AND THE COUNT LEFT TO GO
F473 D1 820 POP DE
F474 D5 830 PUSH DE
F475 23 840 MAYBE INC HL : NOW EACH CHARACTER IS TESTED IN TURN
F476 EB 850 EX DE,HL
F477 23 860 INC HL
F478 7E 870 LD A,(HL)
F479 FEFF 880 CP 255 : AND WHEN THE END OF THE STRING
F47B 2B0D 890 JR Z,FOUND : IS FOUND THEN THE STRING HAS
900 : BEEN FOUND
F47D EB 910 EX DE,HL
F47E 7E 920 CP (HL)
F47F 2BF4 930 JR Z,MAYBE
F481 2AEEF4 940 LD HL,(LOCEND) : IF THE MATCH FAILS THEN LOCK
950 : FOR THE FIRST CHARACTER AGAIN
F484 23 960 INC HL
F48B 3ACBF4 970 LD A,(SCHAR)
F48D 18DD 980 JR LOCKON
990 :
F48A E1 1000 FOUND POP HL
F48B 2AEEF4 1010 LD HL,(LOCEND) : FIRST IT IS CHECKED IF THE FIND
F48E 2B 1020 RACK DEC HL : IS IN THE ALLOWED AREA
F48F 22F0F4 1030 LD (MEMPOS),HL
F492 23 1040 INC HL
F493 E5 1050 PUSH HL
F494 ED5BF3F4 1060 LD DE,(LIMIT)
F498 A7 1070 AND A
F499 ED52 1080 SBC HL,DE
F49B 3014 1090 JR NC,NOTFND : AND IF NOT IT IS DISALLOWED
F49D E1 1100 POP HL
F49E 2B 1110 BACKMD DEC HL : BEFORE THE ENTRY CAN BE PRINTED
F49F CB7E 1120 BIT 7,(HL) : THE START MUST BE FOUND
F4A1 29FB 1130 JR Z,BACKMD
F4A3 23 1140 INC HL
F4A4 E5 1150 PUSH HL
F4A5 E5 1160 PUSH HL
F4A6 CD6B00 1170 CALL 3425 : CLEAR THE WHOLE SCREEN
F4A8 D1 1180 POP DE
F4A9 3EFF 1190 LD A,#FF : SIGNAL TO PRINT ROUTINE THE ADDRESS
F4AC CD3AF4 1200 CALL PMESS : IN DE IS TO BE USED
F4AF C1 1210 POP BC
F4B0 C9 1220 RET
F4B1 3E03 1230 NOTFND LD A,Z : PRINT MESSAGE 3
F4B3 CD33F4 1240 CALL PRINAT
F4B6 E1 1250 POP HL
F4B7 011000 1260 LD BC,0
F4BA C9 1270 RET
1280 :
F4BB 21CBF4 1290 CONTX LD HL,SCHAR : THIS IS THE ENTRY POINT TO LOOK FOR A
F4BE E5 1300 PUSH HL : FURTHER OCCURENCE OF THE FIND STRING
F4BF 7E 1310 LD A,(HL)
F4C0 ED48ECF4 1320 LD BC,(LOCEND)
F4C4 2AEEF4 1330 LD HL,(LOCEND)
F4C7 23 1340 INC HL
F4CB C367F4 1350 JP LOCKON
1360 :
1370 :
F4CB 1380 SCHAR DEFS 33
F4EC 0000 1390 LOCEND DEFW 0 : THE COUNT LEFT AFTER A SEARCH
F4EE 3275 1400 LOCEND DEFW 30002 : THE LOCATION OF A FIND
F4F0 3175 1410 MEMPOS DEFW 30001 : ONE BEFORE THE FIND LOCATION
F4F2 C0F3 1420 LIMIT DEFW 62400 : THE LIMIT OF MEMORY THAT CAN BE USED
F4F4 1420 PMESS EQU #F424 : MESSAGE PRINTING ROUTINE, SEE PREVIOUS
F4F3 1440 PRINAT EQU #F433 : SUBROUTINE IN THIS APPENDIX
    
```

Pass 2 errors: 00

```

BACKMD F49E CONTL F48B
FINDIT F450 FOUND F48A
LIMIT F4F2 LDCNT F4EC
LDFND F4EE LOOKMD F466
LDDON F467 MAYBE F475
MC2 C770 MEMPOS F4F0
NOTFND F4B1 PMESS F424
PRINAT F433 SCHRR F4CB
    
```

Pass 1 errors: 00

```

1360 ; THIS SUBROUTINE WILL ERASE A RECORD WHEN USED WITH THE
1370 ; PREVIOUS SUBROUTINE
1380 ;
1390 ; ON EXIT THE BC REGISTERS WILL CONTAIN THE AMOUNT OF MEMORY
1400 ; RECLAIMED
1410 ;
1420 ;
F4CB 1420 MCS DRG #F4CB
F4CB CDCBF4 1440 XREC CALL FINDIT ; THIS IS USED TO FIND
F4CE 78 1450 XRECON LD A,B ; THE ENTRY TO BE ERASED
F4CF B1 1460 DR C
F4D0 CB 1470 RET I ; BC = 0 IF NOT FOUND
F4D1 E5 1480 PUSH HL
F4D2 C5 1490 PUSH BC
F4D3 3E02 1500 LD A,2 ; MESSAGE 2
F4D5 CDD5F4 1510 CALL PRINAT
F4D8 3E7F 1520 CHECKB LD A,127 ; THIS IS ONE WAY OF SCANNING
1530 ; FOR KEYS PRESSED, SEE CHAPTER 2
F4DA DBFE 1540 IN A,(#FE)
F4DC 1F 1550 RRA
F4DD 3030 1560 JR NC,BREAK
F4DF 3E7F 1570 LD A,127
F4E1 DBFE 1580 IN A,(#FE)
F4E3 CB5F 1590 BIT 3,A
F4E5 2B2E 1600 JR Z,NEXTR
F4E7 3EFE 1610 LD A,254
F4E9 DBFE 1620 IN A,(#FE)
F4EB CB57 1630 BIT 2,A
F4ED 20E9 1640 JR NZ,CHECKB
F4EF C1 1650 POP BC
F4F0 E1 1660 POP HL
F4F1 C5 1670 PUSH BC
F4F2 C5 1680 PUSH BC ; BC = START OF RECORD
F4F3 E1 1690 POP HL
F4F4 010000 1700 LD BC,0
F4F7 CB7E 1710 F_END BIT 7,(HL)
F4F9 23 1720 INC HL
F4FA 03 1730 INC BC ; THE END OF THE RECORD
F4FB 2BFA 1740 JR Z,F_END ; MUST BE FOUND
F4FD D1 1750 POP DE ; NOW EVERYTHING ABOVE THE
F4FE C5 1760 PUSH BC ; RECORD UP TO LIMIT CAN BE BROUGHT
F4FF E5 1770 PUSH HL ; DOWN TO DELETE THE RECORD
F500 C1 1780 POP BC
F501 E5 1790 PUSH HL
F502 2A02F5 1800 LD HL,(LIMIT)
F505 A7 1810 AND A
F506 ED42 1820 SBC HL,BC
F508 E5 1830 PUSH HL
F509 C1 1840 POP BC
F50A E1 1850 POP HL
F50B EDB0 1860 LDIR
F50D C1 1870 POP BC ; THE AMOUNT OF GAINED SPACE
F50E C9 1880 RET
F50F E1 1890 BREAK POP HL ; CLEAR THE STACK AND SHOW NOT FOUND
F510 C1 1900 POP BC
    
```

```

F511 010000 1910 LD BC,0
F514 C9 1920 RET
F515 E1 1930 NEYTR POP HL ; RESTORE THE OLD DETAILS AND LOOK ON
F516 C1 1940 POP BC
F517 CD17F5 1950 CALL CONTL
F51A C3CEFA 1960 JP XRECON
    
```

Pass 2 errors: 00

```

*WARNING* FINDIT absent
*WARNING* PRINAT absent
*WARNING* LIMIT absent
*WARNING* CONTL absent
Table used: 147 from 267
    
```

Pass 1 errors: 00

```

1970 ; THIS ROUTINE ALLOWS KEYBOARD INPUT TO BE PLACED INTO MEMORY
1980 ; AS WELL AS ECHOED TO THE SCREEN.
1990 ;
2000 ;
2010 ; IT Caters FOR FORWARD AND BACKWARD CURSOR MOVEMENT
2020 ; AND INSERTION AND DELETION OF CHARACTERS
2030 ;
2040 ; IT ALSO OFFERS A HELP PAGE AND A MENU
2050 ;
2060 ; AS SHOWN HERE IT IS FOR USE WITH THE OTHER ROUTINES
2070 ; IN THIS SECTION AS A DATABASE,
2080 ; BUT IT CAN EASILY BE TAILORED TO SUIT MANY PURPOSES.
2090 ;
2100 ; THE ROUTINE CHECKS FOR MEMORY OVER THE LIMIT BEING USED
2110 ; AND WILL STOP INPUT AT THIS STAGE
2120 ;
2136 ; I HAVE PUT A SECOND INPUT ROUTINE IN FOR FIND INPUT
2140 ; AS WELL AS EXTRA INPUT ROUTINES FOR SELECTION OF OPTIONS
2150 ; THIS IS INTENDED TO DEMONSTRATE THE DIFFERENT WAYS
2160 ; OF GETTING INPUT, AND SOME OF THE PROBLEMS MENTIONED
2170 ; IN THE MAIN TEXT OF THE BOOK
2180 ;
2190 ;
2200 ; THIS IS THE CALL ADDRESS FROM BASIC.
2210 ;
F51D 2220 MCA DRG #F51D
F51D 2E40 2230 UBR LD A,#A0 ; FIRST FREE MEMORY IS CHECKED
F51F 2A1FF5 2240 LD HL,(LIMIT)
F522 EDB22F5 2250 LD DE,(CHPOS)
F526 A7 2260 AND A
F527 ED52 2270 SBC HL,DE
F529 0234F5 2280 JP NC,NOTFUL
F52C 3E09 2290 LD A,9 ; MESSAGE 9
F52E C02EF5 2300 CALL PRINAT
F531 C0DF6 2310 JP PMESS4
F534 E5 2320 NOTFUL PUSH HL ; AND ALL MEMORY ABOVE THE LAST RECORD
F535 C1 2330 POP BC ; AND BELOW LIMIT IS CLEARED
F536 D5 2340 PUSH DE ; AND MARKED AS FREE
F537 E1 2350 POP HL
F53B 13 2360 INC DE
F539 77 2370 LD (HL),A
F53A EDB0 2380 LDIR
F53C 2A22F5 2390 INPUTA LD HL,(CHPOS) ; AND THE START OF THE NEXT RECORD
F53F 2B 2400 DEC HL
F540 7E 2410 LD A,(HL)
F541 FE80 2420 CP #80
F543 2B01 2430 JR Z,ATBEG
F545 23 2440 INC HL
F546 3680 2450 ATBEG LD (HL),#80
F548 224BF5 2460 LD (CUPOS),HL ; IS MARKED, AND THE START NOTED
F54B CD6B00 2470 CALL 3435 ; THE SCREEN IS CLEARED
F54E CD4EF5 2480 CALL OPEN1
    
```

```

F551 012118 2490 LD BC,#1821 : AND THE PRINT POSITION IN THE
F554 CD0970 2500 CALL 2545 : LOWER SCREEN SET TO THE TOP LEFT
F557 2E07 2510 LD A,7 : MESSAGE ?
F559 CD59F5 2520 CALL PRINT
F55C 3E01 2530 LD A,1 : THE COPYRIGHT
F55E CD5EF5 2540 CALL PMESS AND THE EDIT MODE FLAG IS CLEARED
F561 AF 2550 XOR A :
F562 2262F5 2560 LD (WFLAG),A
F565 01C116 2570 LD BC,#1621 : SET LINE 2 COLUMN 0
F568 CD0970 2580 CALL 2545
F56B 3E3E 2590 LD A,10 :
F56D 07 2600 RST 16
F56E CD02F7 2610 CALL CURSOR
F571 CD4EF5 2620 INPUT CALL OPEN1
F574 CD4810 2630 INF1 CALL #10AB : THIS IS THE KEYBOARD INPUT
F577 2B06 2640 JR C,KEYPRE : ROUTINE, CARRY SET IF NEW KEY PRESSED
F579 FDC8029E 2650 RES 3,(IY+2) : THIS IS TV FLAG, SEE WAIT INPUT CHAP.2
F57D 1BFS 2660 JR INF1
F57F FE08 2670 KEYPRE CF B : NOW ANY SPECIAL FUNCTION KEYS PICKED
F581 2B4E 2680 JR Z,BAD1
F583 FE09 2690 CP 9
F585 CA11F7 2700 JP Z,RIGHT1
F588 FE0C 2710 CP 12 : CAPS SHIFT AND 0
F58A DAE6F7 2720 JP Z,DELETE
F58D FECS 2730 CP 197 : OR
F58F 0B 2740 RET Z
F590 FECD 2750 CP 205 : STEP
F592 DACFF7 2760 JP Z,HELP
F595 FE6 2770 CP 198
F597 CABEF7 2780 JP Z,AND
F599 FE3 2790 CP 195 : NOT
F59C CAD9F6 2800 JP Z,LOADM
F59F FECC 2810 CF 204 : TO
F5A1 CA87F7 2820 JP Z,LPRINT
F5A4 2A48F5 2830 LD HL,(CUPOS)
F5A7 23 2840 INC HL : NOW IT IS CHECKED THAT THE END OF A
F5A8 F5 2850 PUSH AF : RECORD HAS NOT BEEN REACHED, IF IT HAS
F5AA 2A62F5 2860 LD A,(WFLAG) : NO MORE INPUT IS ALLOWED
F5AC A7 2870 AND A
F5AD 2B06 2880 JR Z,NOTHEN : BUT IF NOT IN EDIT MODE IT MUST BE
F5AF 7E 2890 LD A,(HL) : FREE MEMORY
F5B0 FE8D 2900 CP #8D
F5B2 CA90F7 2910 JP Z,ENDREC
F5B5 F1 2920 D,THEN POP AF
F5B6 FE2 2930 CP 226 : STOP WHICH ENDS INPUT
F5B8 2B3B 2940 JR Z,STOP
F5BA 77 2950 LD (HL),A : THE CHARACTER CODE IS PUT INTO THE
F5BB 2248F5 2960 LD (CUPOS),HL : CURRENT MEMORY POSITION AND THIS IS
2970 : UPDATED FOR NEXT TIME
F5BE FE0D 2980 CP 13
F5C0 2006 2990 JR NZ,KEYOUT : IF AN ENTER REMOVE THE OLD CURSOR
F5C2 CDC2F5 3000 CALL OPEN2
F5C5 CD02F7 3010 CALL CURSOR
F5C8 CDC2F5 3020 KEYOUT CALL OPEN2
F5CB 07 3030 RST 16 : ECHO THE KEY TO THE SCREEN
F5CC CD02F7 3040 CALL CURSOR : AND MOVE THE CURSOR ALONG
F5CF 18A0 3050 JR INPUT : GO BACK FOR THE NEXT KEY
3060 :
3070 :
3080 BACK1 LD HL,(CUPOS) : THIS MOVES THE CURSOR AND CURRENT
F5D1 2A48F5 3090 LD A,(HL) : MEMORY POSITION BACK
F5D4 7E 3100 BIT 7,A
F5D5 CB7F 3110 JR NZ,INPUT
F5D7 2098 3120 CP 13
F5D9 FE0D 3130 DEC HL
F5DB 2B 3140 LD (CUPOS),HL
F5DD 2248F5 3150 JR NZ,B_ON
F5DF 2005 3160 CALL BACK : BUT GOING BACK PAST AN ENTER
F5E1 CDE1F5 3170 CALL BACK : MEANS A RE-PRINT AND CHANGE OF MODE
F5E4 1B6F 3180 JR FINRET :
F5E6 CDC2F5 3180 B ON CALL OPEN2
    
```

```

F5E9 CD02F7 3190 CALL CURSOR
F5EC 2E08 3200 LD A,B
F5EE 07 3210 RST 16
F5EF CD02F7 3220 CALL CURSOR
F5F2 C371F5 3230 JP INPUT : BACK FOR NEXT CHARACTER
3240 :
3250 :
F5F5 E5 3260 STOP PUSH HL : IF STOP IS PRESSED THE END OF THE
F5FA 2B 3270 DEC HL : RECORD MUST ONLY BE MARKED
F5F7 7E 3280 ENDIS LD A,(HL) : IF IT WAS A NEW ENTRY, BUT IN ALL
F5FB 27 3290 INC HL : CASES THE START OF FREE MEMOR
F5F9 FE80 3300 CP #80 : FOUND AND THE POINTER UPDATED
F5FB 20FA 3310 JR NZ,ENDIS
F5FD 2B 3320 DEC HL
F5FE 2222F5 3330 LD (CHPOS),HL
F601 E1 3340 POP HL
F602 2A62F5 3350 LD A,(WFLAG)
F605 A7 3360 AND A
F606 25 3370 INC HL
F607 2007 3380 JR NZ,STOPDN
F609 2222F5 3390 LD (CHPOS),HL
F60C 2B 3400 DEC HL
F60D 2E8D 3410 LD A,#8D
F60F 77 3420 LD (HL),A
F610 CD800 3430 STOPDN CALL 3435 : THE SCREEN IS CLEARED
F613 AF 3440 XOR A : THE MODE FLAG CLEARED
F614 2262F5 3450 LD (WFLAG),A
F617 2E04 3460 LD A,4 : AND THE MENU DISPLAYED
F619 CD5EF5 3470 CALL PMESS
F61C CD4EF5 3480 MENUIN CALL OPEN1
F61F CD0E15 3490 CALL #15DE : THE WAIT INPUT ROUTINE, SEE CHAPTER 2
F622 FE00 3500 CP 15
F624 CA1DF5 3510 JP Z,USR
F627 FE20 3520 DR #20
F629 FE45 3530 CP "e"
F62B 2B10 3540 JR Z,XINPUT
F62D FE66 3550 CP "f"
F62F 2B1E 3560 JR Z,FINPUT
F631 FE63 3570 CP "c"
F633 CAD9F6 3580 JP Z,LOADM
F636 FE73 3590 CP "e"
F638 CA38F6 3600 JP Z,SAVEI
F63B 1BDF 3610 JR MENUIN
3620 :
3630 :
F63D CD80F6 3640 XINPUT CALL FINPT : THIS IS THE ERASE ENTRY ROUTINE START
F640 CD80F6 3650 CALL YREC
F645 2A22F5 3660 LD HL,(CHPOS) : THE GAINED SPACE IS ALLOWED FOR BY
F646 A7 3670 AND A : MOVING THE START OF FREE MEMORY
F647 ED42 3680 SBC HL,BC : DOWN BY THAT AMOUNT
F649 2222F5 3690 LD (CHPOS),HL
F64C C30FF5 3700 JP PMESS4
3710 :
3720 :
F64F CD80F6 3730 FINPUT CALL FINPT : THE START OF THE FIND ENTRY ROUTINE
F652 C852F6 3740 CALL FINDIT
F655 78 3750 FINRET LD A,B : IF THE ENTRY IS NOT FOUND GOTO MENU
F656 B1 3760 DR C
F657 CADFF6 3770 JP Z,PMESS4
F65A 0B 3780 DEC BC
F65B ED4348F5 3790 LD (CUPOS),BC : OTHERWISE SET EDIT MODE WITH THE
F65F 2A5FF6 3800 LD HL,(MEMPOS) : CURSOR UNDER THE START OF THE FIND
F662 A7 3810 AND A
F663 ED42 3820 SBC HL,BC
F665 2262F6 3830 LD (CURSP),HL
F668 2E01 3840 LD A,1
F66A 2262F5 3850 LD (WFLAG),A
F66D CD4EF5 3860 CALL OPEN1
F670 012118 3870 LD BC,#1821
    
```

```

F673 C0090D 2880 CALL 3545
F676 3E09 3890 LD A,8
F67B CD59F5 3900 CALL PRINT
F67B 3E07 3910 LD A,7
F67D CD59F5 3920 CALL PRINT
F680 CDC2F5 3930 CALL OPEN2
F685 01211B 3940 LD BC,#1821
F686 C0090D 3950 CALL 3545
F689 3E0D 3960 LD A,13
F68B D7 3970 RST 16
F68C CD02F7 3980 CALL CURSOR
F68F ED485F6 3990 LD BC,(CURSP)
F692 7B 4000 RIMORE LD A,8
F694 B1 4010 OR C
F695 CA71F5 4020 JP Z,INPUT
F698 C5 4030 PUSH BC
F699 CD17F7 4040 CALL RIGHTS
F69C D1 4050 POP BC
F69D 0B 4060 DEC BC
F69E 1BF3 4070 JR RIMORE
4080 ;
4090 ;
F6A0 CD680D 4100 FINPT CALL 3435 ; THIS IS SIMILAR TO THE INPUT
F6A3 3E05 4110 LD A,5 ; INPUT ROUTINE EARLIER BUT IT DOES NOT
F6A5 CD5EF5 4120 CALL PMESS ; ALLOW ANY ALTERATIONS
F6A8 3E3E 4130 LD A,62 ; AND LIMITS THE INPUT BY STOPPING
F6AA D7 4140 RST 16 ; WHEN THE POSITION REACHES IN_LIM
F6AB 21C8F6 4150 LD HL,SCHAR+32
F6AE 22AEF6 4160 LD (IN_LIM),HL
F6B1 21A8F6 4170 LD HL,SCHAR
4180 ;
4190 ;
F6B4 E5 4200 INPUTF PUSH HL ; THE ACTUAL INPUT SUBROUTINE
F6B5 CD4EF5 4210 CALL OPEN1
F6B8 CDDE15 4220 CALL #15DE
F6BB E1 4230 POP HL
F6BC FE0D 4240 CP 13 ; TERMINATED BY PRESSING ENTER
F6BE 2B16 4250 JR Z,SETFIN
F6C0 FE20 4260 CP 32
F6C2 3BF0 4270 JR C,INPUTF
F6C4 77 4280 LD (HL),A
F6C5 23 4290 INC HL
F6C6 E5 4300 PUSH HL
F6C7 CDC2F5 4310 CALL OPEN2
F6CA D7 4320 RST 16
F6CB E1 4330 POP HL
F6CC EB 4340 EX DE,HL
F6CD 2AAEF6 4350 LD HL,(IN_LIM)
F6D0 A7 4360 AND A
F6D1 ED52 4370 SBC HL,DE
F6D3 EB 4380 EX DE,HL
F6D4 20DE 4390 JR NZ,INPUTF
F6D6 36FF 4400 SETFIN LD (HL),255
F6D8 C9 4410 RET
4420 ;
4430 ;
F6D9 CD09F6 4440 LOOKM CALL CONTL ; THIS LOOKS FOR A FURTHER OCCURENCE OF
F6DC C255F6 4450 JP FINRET ; A FIND STRING
4460 ;
4470 ;
F6DF 0603 4480 PMESS4 LD B,3 ; CLEAR THE BOTTOM THREE SCREEN LINES
F6E1 CD440E 4490 CALL 3452 ; CHAPTER 2 EXPLAINS
F6E4 3E04 4500 LD A,4 ; THE MENU
F6E6 CD2EF5 4510 CALL PRINAT
F6E9 C31DF6 4520 JP MENUIN
4530 ;
4540 ;
4550 ;
4560 ;
4570 ; THE CONTROL CODES
    
```

```

F6EC F5 4580 OVER1 PUSH AF
F6ED E5 4590 PUSH HL
F6EE 3E15 4600 LD A,21
F6F0 D7 4610 RST 16
F6F1 3E01 4620 LD A,1
F6F3 D7 4630 RST 16
F6F4 E1 4640 POP HL
F6F5 F1 4650 POP AF
F6F6 C9 4660 RET
F6F7 F5 4670 OVER0 PUSH AF
F6F8 E5 4680 PUSH HL
F6F9 3E15 4690 LD A,21
F6FB D7 4700 RST 16
F6FC 3E00 4710 LD A,0
F6FE D7 4720 RST 16
F6FF E1 4730 POP HL
F700 F1 4740 POP AF
F701 C9 4750 RET
F702 F5 4760 CURSOR PUSH AF ; THIS PRINTS THE CURSOR AND MOVES
F703 CDECF6 4770 CALL OVER1 ; THE PRINT POSITION BACK OVER IT
F706 3E5F 4780 LD A,95
F708 D7 4790 RST 16
F709 3E08 4800 LD A,8
F70B D7 4810 RST 16
F70C CD7F6 4820 CALL OVER0
F70F F1 4830 POP AF
F710 C9 4840 RET
F711 CD17F7 4850 RIGHT1 CALL RIGHTS
F714 C371F5 4860 JP INPUT
F717 CDC2F5 4870 RIGHTS CALL OPEN2 ; THIS SUBROUTINE MOVES THE CURSOR
F71A 2A8F5 4880 LD HL,(CURPOS) ; RIGHT ONE POSITION AND POINTS THE
F71D 22 4890 INC HL ; EMPTY POINTER TO THE NEW CHARACTER
F71E 7E 4900 LD A,(HL)
F71F CB7F 4910 BIT 7,A
F721 C0 4920 RET NZ
F722 FE7F 4930 CP 127
F724 C8 4940 RET Z
F725 FE0D 4950 CP 13
F727 2B02 4960 JR Z,ONRITE
F729 FE20 4970 CP 32
F72B DB 4980 RET C
F72E 22A8F5 4990 ONRITE LD (CURPOS),HL
F72F CD02F7 5000 CALL CURSOR
F732 FE0D 5010 CP 15
F734 2B05 5020 JR Z,RITEON
F736 CDECF6 5030 CALL OVER1
F739 3E20 5040 LD A,32
F73B D7 5050 RITEON RST 16
F73C CD02F7 5060 CALL CURSOR
F73F C9 5070 RET
5080 ;
5090 ;
F740 E5 5100 ENDREC PUSH HL ; IF ENTRY IS ATTEMPTED BEYOND THE
F741 0602 5110 LD B,2 ; END OF A RECORD A WARNING IS FLASHED
F743 CD440E 5120 CALL 3652 ; THE BOTTOM OF THE SCREEN IS CLEARED
F746 3E06 5130 LD A,6
F748 CD5EF5 5140 CALL PMESS
F74B FE 5150 EI ;
F74C 0632 5160 LD B,50 ; JUST IN CASE IF THEY WERE OFF
F74E 76 5170 WAIT1 HALT THIS 1 SEC PAUSE WOULD BE FOREVER
F74F 10FD 5180 DJNZ WAIT1
F751 E1 5190 POP HL
F752 F1 5200 POP AF
F755 2153F7 5210 REENTR LD HL,LOCEND
F756 35 5220 DEC (HL)
F757 2B 5230 DEC HL
F758 26 5240 INC (HL)
F759 CD09F6 5250 JP LOOKM
5260 ;
5270 ;
    
```

```

F75C 2A49F5 5280 BELETE LD HL,(CUPOS) : THIS WORKS THE SAME AS THE DELETE
F75F 22 5290 INC HL : RECORD BUT FOR ONE CHARACTER
F760 7E 5300 LD A,(HL)
F761 C87F 5310 BIT 7,A
F763 C271F5 5320 JF NZ,INPUT
F766 E5 5330 PUSH HL
F767 E5 5340 PUSH HL
F768 2A1FF5 5350 LD HL,(LIMIT)
F76B D1 5360 POP DE
F76C A7 5370 AND A
F76D E052 5380 SBC HL,DE
F76F E5 5390 PUSH HL
F770 C1 5400 POP BC
F771 D1 5410 POP DE
F772 D871F5 5420 JF C,INPUT
F775 C871F5 5430 JF Z,INPUT
F778 D5 5440 PUSH DE
F779 E1 5450 POP HL
F77A 22 5460 INC HL
F77B EDB0 5470 LDIR
F77D 2A22F5 5480 LD HL,(CHPOS)
F780 2B 5490 DEC HL
F781 2222F5 5500 LD HL,(CHPOS),HL
F784 2A48F5 5510 LD HL,(CUPOS)
F787 22 5520 INC HL
F78B C0E1F5 5530 CALL BACK
F78C C355F6 5540 JP FINRET
5550 :
5560 :
F78E 2A48F5 5570 AND LD HL,(CUPOS) : AND THIS IS THE SAME IN REVERSE
F791 22 5580 INC HL
F792 E5 5590 PUSH HL
F793 2A1FF5 5600 LD HL,(LIMIT)
F796 D1 5610 POP DE
F797 A7 5620 AND A
F798 E052 5630 SBC HL,DE
F79A E5 5640 PUSH HL
F79B C1 5650 POP BC
F79C D5 5660 PUSH DE
F79D E05B1FF5 5670 LD DE,(LIMIT)
F7A1 D5 5680 PUSH DE
F7A2 E1 5690 POP HL
F7A3 2B 5700 DEC HL
F7A4 E08B 5710 LDIR
F7A6 2A22F5 5720 LD HL,(CHPOS)
F7A9 22 5730 INC HL
F7AA 2222F5 5740 LD HL,(CHPOS),HL
F7AD E1 5750 POP HL
F7AE 3E20 5760 LD A,32
F7B0 77 5770 LD (HL),A
F7B1 C0E1F5 5780 CALL BACK
F7B4 C355F6 5790 JP FINRET
5800 :
5810 :
F7B7 2A48F5 5820 LPRINT LD HL,(CUPOS) : THIS USES STREAM THREE IN AN IDENTICAL
F7BA 22 5830 INC HL : WAY AS STREAM 2 FOR THE SCREEN.
F7BB C0E1F5 5840 CALL BACK : IF AN INTERFACE IS CONNECTED
F7BE C5 5850 PUSH BC : WHICH RECOGNISES LPRINT IT WILL WORK.
F7BF 3E03 5860 LD A,3
F7C1 C06116 5870 CALL #1601
F7C4 D1 5880 POP DE
F7C5 D5 5890 PUSH DE
F7C6 1B 5900 DEC DE
F7C7 AF 5910 XOR A
F7C8 C0C8F7 5920 CALL PRINTP
F7CB C1 5930 POP BC
F7CC C355F6 5940 JP FINRET
5950 :
5960 :
F7CF C0680D 5970 HELP CALL 3425
    
```

```

F7DD 3E0A 5980 LD A,10 : THE HELP PAGE
F7DA C05E5F5 5990 CALL PMESS
F7D7 3E8F 6000 WAITE LD A,#BF
F7D8 DBFE 6010 IN A,(#FE)
F7DB E80C 6020 AND I
F7DD 20FB 6030 JR NZ,WAITE
F7DF 2A48F5 6040 LD HL,(CUPOS)
F7E2 2252F7 6050 LD HL,(LOCEND),HL
F7E5 22 6060 INC HL
F7E6 7E 6070 LD A,(HL)
F7E7 21ABF6 6080 LD HL,SCHAR
F7EA 77 6090 LD (HL),A
F7EB 22 6100 INC HL
F7EC 36FF 6110 LD (HL),#FF
F7EE C352F7 6120 JP REENTR
    
```

Pass 2 errors: 00

- \*WARNING\* LIMIT absent
- \*WARNING\* CHPOS absent
- \*WARNING\* PRINAT absent
- \*WARNING\* CUPOS absent
- \*WARNING\* OPENI absent
- \*WARNING\* PRINT absent
- \*WARNING\* PMESS absent
- \*WARNING\* WFLAG absent
- \*WARNING\* OPEN2 absent
- \*WARNING\* BACK absent
- \*WARNING\* SAVEL absent
- \*WARNING\* XREC absent
- \*WARNING\* FINDIT absent
- \*WARNING\* MEMPOS absent
- \*WARNING\* CURSP absent
- \*WARNING\* SCHAR absent
- \*WARNING\* IN\_LIM absent
- \*WARNING\* CONTLK absent
- \*WARNING\* LOCEND absent
- \*WARNING\* PRINTP absent

Table used: 751 from 1219

Pass 1 errors: 00

```

6130 : THIS IS THE START OF THE SAVE AND LOAD ROUTINES
6140
6150 : FIRST THE LENGTH OF THE RECORDS IS CALCULATED.
6160 : WITH AN OVERHEAD FOR AN END-MARKER
6170 : AND LENGTH BYTES FOR RE-ENTRY AFTER LOADING
6180 :
6190 :
F7F1 2A1F7 6200 M5 ORG #F7F1
F7F4 222E75 6210 SAVEL LD HL,(CHPOS) : FIRST THE END OF RECORDS IS PUT INTO
F7F7 112E75 6220 LD (2999B),HL : THE START OF THE AREA TO BE SAVED OR
F7FA A7 6230 LD DE,2999B : LOADED
F7FB A7 6240 AND A
F7FC E052 6250 SBC HL,DE
F7FD 22 6260 INC HL
F7FE 22 6270 INC HL
F7FF E5 6280 PUSH HL
F800 3E06 6290 LD A,11
F802 C002FB 6300 CALL PMESS
F805 0619 6310 LD B,25 : 25 INTERRUPTS 1/2 SECOND
F807 FB 6320 EI : JUST IN CASE
6330 :
6340
6350 :
6360 :
6370 :
6380 :
        THIS WOULD BE BAD PLANNING
        BUT IT IS TO DEMONSTRATE THE
        USE OF THE HALT FOR TIMING
    
```

```

FB08 76      6390 HOLDIT HALT ;           TO MAKE SURE THAT S IS NOT TAKEN
FB09 10FD    6400      DJNZ HOLDIT ;     FROM THE MAIN MENU KEYFUSH
                                                NOW THE KEYBOARD IS SCANNED
6410 ;
6420 ;
FB08 2E8F    6430 SOLDEC LD  A,#BF
FB0D DBFE    6440      IN  A,(#FE)
FB0F E602    6450      AND  2
FB11 CA11FB  6460      JP   2,LOAD
FB14 2EFD    6470      LD   A,#FD
FB16 DBFE    6480      IN  A,(#FE)
FB18 E602    6490      AND  2
FB1A 20EF    6500      JR   NZ,SOLDEC
6510 ;

```

Pass 2 errors: 00

```

*WARNING* CHPOS absent
*WARNING* PMESS absent
*WARNING* LOAD absent
*WARNING* M_OR_T absent
*WARNING* LORET absent
*WARNING* HLSAV absent
*WARNING* BCSAV absent
*WARNING* MHEAD absent
Table used: 243 from 521

```

Pass 1 errors: 00

```

6520 ; THIS IS THE DECISION MAKER FOR TAPE OR MICRODRIVE
6530 ;
6540 ;
FB1C        6550      DRG  #FB1C
FB1C C1     6560 SAVE  POP  BC
FB1D CD1DFB 6570      CALL M_OR_T ;     RETURNS WITH CARRY SET IF M PRESSED
                                                ALSO SETS THE HEADER
6580 ;
6590 ;
FB20 385B   6600      JR   C,MSAVE
6610 ;
6620 ; THIS IS TO SAVE A HEADER AND A MAIN DATA BLOCK TO TAPE
6630 ; THE HEADER IS SPECIAL. SEE SAVING & LOADING
6640 ; IN CHAPTER 2
6650 ;
FB22 05     6660 TSAVE  PUSH BC ;     BC = LENGTH OF THE MAIN BLOCK
                                                IX IS ALREADY SET TO THE START
6670 ;
FB23 110D00 6680      LD   DE,13 ;     THE HEADER IS ONLY 13 BYTES LONG
6690      XDR  A ;     A TO BE 0 SIGNALLING HEADER
FB26 AF     6700      CALL 1222 ;     R.O.M. ROUTINE DETAILED IN CHAPTER 2
                                                TO SAVE HEADER
6710 ;
FB2A D1     6720      POP  DE ;     RESTORE THE LENGTH OF THE MAIN BLOCK
FB2B DD21E75 6730      LD   IX,29998 ;     POINT IX TO THE
                                                ADDRESS TO START SAVING FROM
6740 ;
FB2F 2EFF   6750      LD   A,#FF ;     SIGNAL MAIN BLOCK
FB31 CDC604 6760      CALL 1222 ;     USE THE R.O.M. ROUTINE AGAIN
6770      EI ;     IT RETURNS WITH INTERRUPTS OFF
FB35 C335FB 6780      JP   LORET

```

Pass 2 errors: 00

```

*WARNING* M_OR_T absent
*WARNING* LORET absent
*WARNING* HLSAV absent
*WARNING* BCSAV absent
*WARNING* MHEAD absent
Table used: 160 from 407

```

Pass 1 errors: 00

```

6790 ;
6800 ;
6810 ;
6820 ;
6830 ;
FB38        6840      DRG  #FB38
FB3B 2149FB 6850 RSWAP LD  HL,ROMOUT
FB3B 22ED5C 6860      LD  (22789),HL ; THE SUBROUTINE USED BY THE R.O.M.
                                                IS USED BY HOOK CODE 22
6870 ;
6880 ;
FB3E D9     6890      EXX ;
FB3F 223FFB 6900      LD  (HLSAV),HL ; SO SET TO RETURN TO THIS ROUTINE
                                                THE STACK IS TO BE MUCKED ABOUT
6910 ;
6920 ;
FB42 ED4343FB 6920      LD  (BCSAV),BC
6930 ;
6940 ;
FB47 CF     6950      RST  8 ;     USE THE HOOK CODE
FB48 32     6960      DEFB #32
FB49 E1     6970 ROMOUT POP  HL ;     THE TWO RETURN ADDRESSES
6980 ;
FB4A E1     6980      POP  HL ;     MUST BE REMOVED
FB4B D1     6990      POP  DE ;     AND THE RETURN FROM THIS SUBROUTINE
69A0 ;
FB4C 2A3D5C 69B0      LD  HL,(22613) ; THIS IS ERR SP SEE CHAPTER 4
69C0 ;
FB4F E5     69C0      PUSH HL ;     SAVE THE POSITION OF THE ERROR RETURN
                                                ON THE STACK FOR LATER
7000 ;
FB50 216AFB 7010      LD  HL,MSLRET
7020 ;
FB53 E5     7020      PUSH HL ;     STACK THE NEW ERROR RETURN ADDRESS
7030 ;
FB54 ED733D5C 7030      LD  (22613),SP ; AND POINT ERR SP TO THAT
7040 ;
FB58 D5     7040      PUSH DE ; RESTORE THE RETURN ADDRESS
7050 ;
FB59 C9     7050      RET

```

Pass 2 errors: 00

```

*WARNING* HLSAV absent
*WARNING* BCSAV absent
*WARNING* MHEAD absent
*WARNING* LORET absent
Table used: 124 from 287

```

Pass 1 errors: 00

```

7060 ; THIS IS THE SUBROUTINE TO SAVE TO MICRODRIVE, SEE CHAP 2
7070 ;
7080 ;
7090 ;
FB5A        7090      DRG  #FB5A
FB5A CDSAFB 7100 MSAVE CALL RSWAP
FB5D CDSDFB 7110      CALL MHEAD ;     SET THE HEADER
6860 CDS6FB 7120      CALL VARSET ;     AND THE SYSTEM VARIABLES
FB63 FDCB7CEE 7130      SET  C,(IV+124) ; SIGNAL SAVE WITH FLAGS 3
6870 CDF71E 7140      CALL #1E7F ;     IN FACT A CALL IS NOT NEEDED
                                                AS RETURN IS VIA THE 16# ERROR 0 OF
7150 ;
FB6A E1     7150 MSLRET POP  HL ;     RESTORE THE ERR SP SYSTEM VARIABLE
FB6B 223D5C 7170      LD  (22613),HL ; ERR SP
7180 ;
FB6E FB     7180      EI ;     TOTALLY UNNECESSARY BUT SAFE
7190 ;
FB6F D9     7190      EXX
FB70 2A70FB 7200      LD  HL,(HLSAV) ; AND THE HL REGISTERS
7210 ;
FB73 D9     7210      EXX ;     NOTE THAT THE 16# R.O.M. IS NOW
7220 ;
FB74 C374FB 7220      JP  LORET ;     PAGED IN, BECAUSE OF THE RETURN VIA
                                                A BASIC ERROR IN THE 16# R.O.M.
7230 ;

```

Pass 2 errors: 00

```

*WARNING* RSWAP absent
*WARNING* MHEAD absent
*WARNING* VARSET absent
*WARNING* HLSAV absent
*WARNING* LORET absent
Table used: 99 from 188

```

```

Pass 1 errors: 00
FB77 7240          ORG #FB77
FB77 21095C       7250 VARSET LD HL,23789 ; THIS IS DETAILED FULLY IN CHAPTER 3
FB7A 364D        7260 LD (HL), "M"
FB7C 210A00      7270 LD HL,10
FB7E 220A5C      7280 LD (23770),HL
FB82 2172F9      7290 LD HL,SCHAR+1
FB85 220C5C      7300 LD (23772),HL
FB88 210100      7310 LD HL,1
FB8B 220A5C      7320 LD (23766),HL
FB8E C9         7330 RET
7340 ;
7350 ;
7360 ;
7370 ; THIS IS THE ROUTINE TO LOAD FROM A MICRODRIVE
7380 ; FULL DETAILS ARE GIVEN IN CHAPTER 3
7390 ;
7400 ;
FB8F 010000      7410 MLOAD LD BC,0 ; THE DETAILS ON THE HEADER ARE TO
FB92 CD92F8      7420 CALL RSWAP ; BE USED AS THEY ARE NOT KNOWN
FB95 CD45F8      7430 CALL MHEAD
FB98 CD77F8      7440 CALL VARSET
FB9B FDC87CE6    7450 SET 4,(1Y+124) ; THIS SIGNALS LOAD
FB9F CD4F08      7460 CALL #BAF ; CALL THE LOAD ROUTINE
FBA2 C3A2F8      7470 JP MSLRET ; IN FACT IT IS UNLIKELY TO GET HERE
7480 ;
7490 ;
7500 ;
7510 ; THIS SETS THE MICRODRIVE HEADER, FULL DETAILS ARE IN
7520 ; CHAPTER 3
7530 ;
7540 ;
FBAA 21E45C      7550 MHEAD LD HL,23782
FBAB 3603        7560 LD (HL),3
FBAA 23         7570 INC HL
FBAB ED585CF9    7580 LD DE,(BCSAV)
FBAF 73         7590 LD (HL),E
FB80 23         7600 INC HL
FB81 72         7610 LD (HL),D
FB82 23         7620 INC HL
FB83 112E75     7630 LD DE,29998
FB86 73         7640 LD (HL),E
FB87 23         7650 INC HL
FB88 72         7660 LD (HL),D
FB89 23         7670 INC HL
FB8A 3680       7680 LD (HL),#B0
FB8C C9         7690 RET
7700 ;
7710 ;
7720 ;
7730 ;
7740 ; THE LOAD DECISION BETWEEN MICRODRIVE AND TAPE
7750 ;
7760 ;
7770 ; LOADING FROM TAPE AS DETAILED IN CHAPTER 2
7780 ;
FBBD C1         7790 LOAD POP BC
FBBE CD05F9     7800 CALL M_DR_T
FBC1 38CC       7810 JR C,MLOAD
FBC3 AF         7820 TLOAD RDR A ; SIGNAL HEADER
FBC4 37         7830 SCF ; SIGNAL LOAD
FBC5 110D00     7840 LD DE,13
FBC8 14         7850 INC D
FBC9 08         7860 EX AF,AF'
FBCA 15         7870 DEC D
FBCB F3         7880 DI
FBCC D02180F9   7890 LD IX,SCHAR+15
FBD0 CD6205     7900 CALL #562
FBD3 FB         7910 EI
    
```

```

FB04 CD541F     7920 CALL #020
FB07 D0         7930 RET NC ; BREAK PRESSED
FB08 D02171F9  7940 TESTHE LD IX,SCHAR ; THE HEADER NAME IS NOW CHECKED
FB0C 2180F9     7950 LD HL,SCHAR+15
FB0F 060E       7960 LD B,11
FB11 7E         7970 TESTLE LD A,(HL)
FB12 D8E000     7980 CF (1X+00)
FB15 20BC       7990 JR NZ,LOAD ; MISMATCH SO WRONG HEADER
FB17 D023       8000 INC IX ; THE HEADER MATCHES SO DO THE LOAD
FB19 23         8010 INC HL
FB1A 10F5       8020 DJNZ TESTLE
FB1C D0212E75  8030 LD IX,29998
FB1E 5E         8040 LD C,(HL)
FB1F 23         8050 INC HL
FB21 56         8060 LD D,(HL)
FB23 37         8070 SCF ; SIGNAL LOAD
FB24 3EFF       8080 LD A,#FF ; MAIN DATA BLOCK
FB26 14         8090 INC D
FB27 08         8100 EX AF,AF'
FB28 15         8110 DEC D
FB29 F3         8120 DI
FB2A CD6205     8130 CALL #562
FB2D FB         8140 EI
FB2E CD541F     8150 CALL #020
F901 D0         8160 RET NC ; BREAK PRESSED
F902 C325F9     8170 JP LORET ; LOADED SO RE-ENTER
8180 ;
8190 ;
8200 ;
8210 ; THIS ROUTINE ASKS FOR THE FILENAME AND IF MICRODRIVE
8220 ; OR TAPE IS TO BE USED
8230 ;
8240 ;
F905 CD2EF9     8250 M_DR_T CALL SAVET
F908 C5         8260 PUSH BC
F909 D0E5       8270 PUSH IX
F90B 3E0C       8280 LD A,12
F90D CD0DF9     8290 CALL PRINAT
F910 3E7F       8300 MIORTA LD A,#7F
F912 D8FE       8310 IN A,(#FE)
F914 E604       8320 AND 4
F916 37         8330 SCF
F917 2B08       8340 JR Z,MOTFIN
F919 3E7F       8350 LD A,#7F
F91B D8FE       8360 IN A,(#FE)
F91D E610       8370 AND 16
F91F 20E7       8380 JR NZ,MIORTA
F921 D0E1       8390 MOTFIN POP IX
F923 C1         8400 POP BC
F924 C9         8410 RET
8420 ;
8430 ;
8440 ;
8450 ; THIS PICKS UP THE LENGTH DETAILS FROM THE
8460 ; START OF THE MEMORY LOADED
8470 ; OR FROM WHAT WAS PUT THERE WHEN IT WAS SAVED
8480 ;
8490 ;
F925 2A2E75     8490 LORET LD HL,(29998)
F928 2260F9     8500 LD (CHPOS),HL
F92B C325F9     8510 JP USR
8520 ;
8530 ;
8540 ;
8550 ; THIS ROUTINE CREATES A HEADER IN THE SCHAR AREA OF MEMORY
8560 ;
8570 ;
F92E 2171F9    8580 SAVET LD HL,SCHAR
F931 C5         8590 PUSH BC
F932 E5         8600 PUSH HL
F933 E5         8610 PUSH HL
    
```

```

F934 2603 B620 LD (HL),J
F936 C5 B630 PUSH BC
F937 060A B640 LD A,10
F939 2E20 B650 LD A,32
F93B 23 B660 BETHED INC HL
F93C 77 B670 LD (HL),A
F93D 10FC B680 DJNZ BETHED
F93F E5 B690 PUSH HL
F940 226AF9 B700 LD (IN_LIM),HL
F943 E1 B710 POP HL
F944 C1 B720 POP BC
F945 23 B730 INC HL
F946 71 B740 LD (HL),C
F947 23 B750 INC HL
F948 70 B760 LD (HL),B
F949 CD6B0D B770 CALL 3425
F94C 2E0D B780 LD A,13
F94E CD4EF9 B790 CALL PMESS
F951 E1 B800 POP HL
F952 23 B810 INC HL
F953 CD53F9 B820 CALL INPUF
F956 2620 B830 LD (HL),32
F958 0DE1 B840 POP IX
F95A C1 B850 POP BC
F95B C9 B860 RET
      8870 :
      8880 :
      8890 :
F95C 0000 B900 BCSAV DEFW 0
F95E 0000 B910 CUFOS DEFW 0
F960 2175 B920 CHFOS DEFW 20001
F962 0000 B930 HLSAV DEFW 0
F964 0000 B940 IN_LIM DEFW 0
F966 26F2 B950 LIMIT DEFW 62000
F968 0000 B960 LDCNT DEFW 0
F96A 0000 B970 LDCFND DEFW 0
F96C 00 B980 WFLAG DEFW 0
F96D 2175 B990 MEMPOS DEFW 20001
F96F 2118 9000 CURSP DEFW #1821
F971 9010 9010 SCHAR DEFS 33
F972 80 9020 MESSH1 DEFB #80
F973 80 9030 MESS DEFB #80
F974 44654241 9040 MESS1 DEFM "DBASE COPYRIGHT SGR 1984"
F975 80 9050 MESS DEFB #80
F976 1001 9060 MESS2 DEFW #110
F977 50524553 9070 DEFM "PRESS SPACE TO ABORT"
F978 0D 9080 DEFB 13
F979 5820544F 9090 DEFM "X TO ERASE OR N FOR NEXT RECORD"
F97A 1000 9100 DEFW #10
F97B 80 9110 DEFB #80
F97C 1002 9120 MESS3 DEFW #210
F97D 4E4F5420 9130 DEFM "NOT FOUND"
F97E 1000 9140 DEFW #10
F97F 80 9150 DEFB #80
F980 1001 9160 MESS4 DEFW #110
F981 20202020 9170 DEFM "MENU"
FA0A 0D0D 9180 DEFW #0D0D
FA0C 50524553 9190 DEFM "PRESS"
FA11 0D0D 9200 DEFW #0D0D
FA13 46202020 9210 DEFM "F TO FIND"
FA21 0D0D 9220 DEFW #0D0D
FA23 45202020 9230 DEFM "E TO ERASE"
FA24 0D0D 9240 DEFW #0D0D
FA34 43202020 9250 DEFM "C TO CONTINUE SEARCH"
FA4D 0D0D 9260 DEFW #0D0D
FA4F 454E5445 9270 DEFM "ENTER TO MAKE ANOTHER ENTRY"
FA68 0D0D 9280 DEFW #0D0D
FA6D 53202020 9290 DEFM "S TO SAVE OR LOAD"
FAB3 1000 9300 DEFW #10
FAB5 80 9310 DEFB #80
    
```

```

FAB6 454E5445 9320 MESS5 DEFM "ENTER DETAILS TO FIND"
FAB8 80 9330 DEFB #80
FAB9 1002 9340 MESS6 DEFW #210
FA9E 0D 9350 DEFB 13
FA9F 454E4420 9360 DEFM "END OF RECORD, NO MORE INPUT"
FABB 0D 9370 DEFB 13
FABC 504F5353 9380 DEFM "POSSIBLE"
FAC4 80 9390 DEFB #80
FAC5 1001 9400 MESS7 DEFW #110
FAC7 50524553 9410 DEFM "PRESS "
FACD 1002 9420 DEFW #210
FACF 53544F50 9430 DEFM "STOP "
FADA 1001 9440 DEFW #110
FAD6 464F5220 9450 DEFM "FOR MENU, "
FAE0 1002 9460 DEFW #210
FAE2 02 9470 DEFB 2
FAE3 544F20 9480 DEFM "TO "
FAE4 1001 9490 DEFW #110
FAE6 544F2050 9500 DEFM "TO PRINTER "
FAF3 1002 9510 DEFW #210
FAF5 53544550 9520 DEFM "STEP "
FAFA 1001 9530 DEFW #110
FADC 464F5220 9540 DEFM "FOR HELP"
FB04 A0 9550 DEFB #A0
FB05 1002 9560 MESS8 DEFW #310
FB07 594F5520 9570 DEFM "YOU ARE IN EDIT MODE"
FB16 80 9580 MCF9 DEFB #80
FB1C 1002 9590 MESS9 DEFW #210
FB1E 594F5520 9600 DEFM "YOU JUST RAN OUT OF MEMORY"
FB38 0D 9610 DEFB 13
FB39 53415645 9620 DEFM "SAVE THE RECORDS OR SOMETHING"
FB56 1000 9630 DEFW #10
FB58 80 9640 DEFB #80
FB59 574B454E 9650 MESS10 DEFM "WHEN YOU ARE IN EDIT MODE"
FB72 0D 9660 DEFB 13
FB73 50524553 9670 DEFM "PRESSING "
FB7C 1002 9680 DEFW #210
FB7E 4E4F5420 9690 DEFM "NOT "
FB82 1000 9700 DEFW #010
FB84 57494C4C 9710 DEFM "WILL FIND THE NEXT OCCURENCE OF THE LAST STRING"
FB85 20202020 9720 DEFM " THAT WAS SOUGHT"
FB86 0D0D 9730 DEFW #0D0D
FB8C 1002 9740 DEFW #210
FB8A 414E4420 9750 DEFM "AND "
FB8E 1000 9760 DEFW #010
FB90 57494C4C 9770 DEFM "WILL INSERT A CHARACTER"
FB8F 0D 9780 DEFB 13
FB8B 41542054 9790 DEFM "AT THE CURRENT CURSOR POSITION."
FC07 0D0D 9800 DEFW #0D0D
FC09 44454C45 9810 DEFM "DELETE WILL REMOVE THE CHARACTER"
FC29 0D 9820 DEFB 13
FC2A 41542054 9830 DEFM "AT THE CURRENT CURSOR POSITION."
FC49 0D0D 9840 DEFW #0D0D
FC4B 49462054 9850 DEFM "IF THERE IS NO SPACE IN A RECORD"
FC68 0D 9860 DEFB 13
FC6C 594F5520 9870 DEFM "YOU ARE ALTERING USE THE INSERT"
FC8B 0D 9880 DEFB 13
FC8C 45554E43 9890 DEFM "FUNCTION TO MAKE SOME SPACE."
FCAB 0D0D 9900 DEFW #0D0D
FC9A 544B4520 9910 DEFM "THE CURSOR KEYS ALLOW YOU TO"
FC86 0D 9920 DEFB 13
FC87 4D4F5645 9930 DEFM "MOVE THROUGH THE TEXT BUT YOU"
FCE4 0D 9940 DEFB 13
FCE5 43414E4E 9950 DEFM "CANNOT GO BACK PAST AN ENTER."
FD02 0D0D 9960 DEFW #0D0D
FD04 1004 9970 DEFW #410
FD06 50524553 9980 DEFM "PRESS ENTER TO RETURN TO TEXT"
FD23 1000 9990 DEFW #10
FD25 80 10000 DEFB #80
FD26 50524553 10010 MESS11 DEFM "PRESS S TO SAVE L TO LOAD"
    
```

```

FD3F 8D      10020      DEFB #BD
FD40 50524553 10030 MESS12 DEFM "PRESS M FOR M/DRIVE"
FD53 0D      10040      DEFB 13
FD54 AF522054 10050      DEFM "OR T FOR TAPE"
FD61 8D      10060      DEFB #BD
FD62 4F4E5055 10070 MESS13 DEFM "INPUT FILE-NAME"
FD71 8D      10080      DEFB #BD
FD72 8D      10090 ZEND  DEFB #BD

```

Pass 2 errors: 00

\*WARNING\* RSWAP absent  
 \*WARNING\* MSLRET absent

# Index

address bus	25, 62, 70	2898	13
address lines	63	3082	12
algebraic functions	89	3435	5
A register		3438	5
calculator	82, 84	3545	4
Interface 1	76	3582	5
8K ROM routines	26-9, 34	3584	17
16K ROM routines	3, 9, 13, 22-4	3652	17
standard streams	67	3756	11
ASC II code	6, 42-5, 82-5	3789	11
Assembler program	1	3807	11
assembly language programming	1	5598	10
automatic listing	46-7	5606	18
		5633	3, 6
BASIC	2-3, 5, 18-23, 46, 51, 75-9	5823	18
error	18, 20, 22, 83, 86, 88	8020	4
interpreter	42, 46, 49, 52, 88-90	8874	17
BASIC LIST command	51	8933	6
BBC microcomputer	25	9146	16
BC register pair	2	calling routine	43, 75
beep	11-12	CAPS LOCK	51
binary numbers	83	CAPS SHIFT	4, 11, 42-3
binomial point	80-1	carriage return	100
block graphics	15	carry flag	4, 10, 23-4, 26-7, 33-4, 83, 89
BREAK key	20	Catalogue Cartridge routine	35
B register	4-6, 17	Centronics	3, 65-6
buffer	11, 30, 32-3	channels	18, 30, 32, 62-9
		channel type	60
calculator	80-92, 104-6	characters	8-10
CALL 10934	82	character set	99-101
CALL 11419	83	Chebyshev polynomial	92
CALL 11560	82	circles	15
CALL 11563	82	Close Microdrive channel routine	32
op-codes	86-91	Close Network channel routine	30
routines	104-6	colours	44, 55, 58, 64, 100
use of	86-92	command mode	76
calculator stack	15-16, 18, 56, 82-92, 105	<i>Complete Spectrum ROM Disassembly</i>	(Logan) 2
Call addresses	3-24	CONTINUOUS SOUND	73
949	11	COORDS	15-17
2878	15		

- C register 6
- current channel 18
- data bus 25
- Data Terminal Ready 28
- DeBASE program 10, 13, 22, 24, 35, 111-28
- DE register 21
- DEFined word 39-40
- DI instruction 72
- El instruction 71-2
  - ENDCALC instruction 86
  - Erase File routine 32
  - E register 6
  - exponent 81
- file name 60
- floating point representation 80-2, 91
- Format Cartridge routine 35
- Get Key routine 26-7
- graphics 11, 13, 15
- header 18-24, 29, 59
  - dummy 22-3
  - microdrive 36-41
- hex to decimal conversions 93-4
- Highsoft's Devpack 3-4
- HL register 40
- hook codes 25-36
  - Catalogue Cartridge (32h) 35
  - Close Microdrive Channel (23h) 32
  - Close Network Channel (2Eh) 30
  - Erase File (24h) 32
  - Format Cartridge (32h) 35
  - Get Key (1Bh) 26
  - Insert Variables (31h) 34
  - Keyscan (20h) 34
  - Motor on/Motors off (21h) 34
  - Network Input (2Fh) 26
  - Open Channel (2Dh) 28
  - Open Channel/Open File (22h) 30-1
  - Print to Printer (1Fh) 27
  - Print to Screen (1Ch) 27
  - Read Next Print Record (25h) 33
  - Read Print Record (27h) 33
  - Read Next Record Sector (29h) 33
  - Read Record Sector (28h) 33
- Reclaim Microdrive Channel (2Ch) 34
- ROM 2 (32h) 34
- RS232 Input (1Dh) 26
- RS232 Output (1Eh) 27
- Run (32h) 36
- Send Packet (30h) 29
- Write Record (26h) 32
- Write Sector (2Ah) 32
- I' register 3
- IM1 mode 71
- IM2 mode 71
- indexing 84
- INPUT commands 52
- input routine 58
- Insert Variables routine 34
- Interface 1 75-9
- interrupts 8-9, 11, 21, 41, 53, 70-4
- interrupt vectors 102-3
- I register 70-3
- IX register 20-1, 23, 26-9, 32-4
- IY register 2, 3, 9, 15, 42
- Kempston Interface 3, 65, 106-7
- keyboard 8-9
  - map 98
  - scanning 43, 63
- key-debouncing routine 43
- Key Input routine 10
- Keyscan routine 34
- LET command 2
- line-drawing 16
- line feed code 28
- LOAD BYTES subroutine 22
- loading 18-24
- L' register 3
- machine code program 1, 2, 38, 55, 80, 83
- machine crash 38
- mantissa 81, 91
- memory 25, 42
- memory map 95-7
- memory request line 70
- message printing 12-13, 21
- Microdrive 30-1, 56, 72, 74, 111
  - channel 33, 36
  - header 36-41
  - Interface 1, 62, 67
  - microphone socket 64
  - MODE CHANGE flag 10
- MOREX Interface 65, 106-7
- Motor on/Motors off routine 34
- MREQ line 70
- network channel 28-9
- Network Input routine 26
- NN command 2
- numbers 6-8
- Open Channel/Open File routine 30
- Open Channel routine 26, 28
- opening and closing streams 3
- Picturesque Editor Assembler 1
- pixels 11, 17, 54, 73, 74, 96
- plotting to the screen 6
- ports
  - Kempston 66-7
  - Morex 65-6
  - 127 66
  - 231 62
  - 239 62, 65
  - 247 62, 65
  - 251 62, 64-5
  - 254 62-4
  - 57535 66
  - 58047 66
  - 58303 66
- PRINT command 2
- printer 11, 45, 65
  - buffer 11, 51, 53
- print positions 4-5
- Print to Printer routine 27
- Print to Screen routine 27
- Programming the Z80 (Zaks)* 2
- random numbers 52
- Read Next Print Record routine 33
- Read Next Record Sector routine 33
- Read Print Record routine 33
- Read Record Sector routine 33
- Reclaim Microdrive Channel routine 34
- record deletion 111
- repeating key routine 43
- ReStarts 38-41
- RET instruction 40
- RETI instruction 71-2
- ROM 2 routine 34
- ROM, 8K 25-41
  - inputs 26-7
  - microdrive header 36-41
  - microdrive input 33-6
  - microdrive output 31-2
  - network output 28-30
  - output 27-8
- ROM interpreter 75-7
- RS232 Input routine 26
- RS232 Output routine 27
- RST
  - 10h 38
  - 16 (10h) 3-4
  - 56 (28h) 8
- Run routine 36
- run time 77
- SAVE/LOAD RETURN routine 21
- saving 18-21
- screen 6, 10
  - clearing 17
  - lower 5
  - scrolling 5, 17
  - whole 5
- screen copy 11
- screen map 54
- scrolling 5, 17
- Send Packet routine 29
- shadow ROM 25, 40
- Sinclair BASIC 25-6
- small integer representation 80
- Spectrum Machine Language for the Complete Beginner (Zaks)* 2
- Spectrum Pocket Book* 7
- SPRITE 73-4
- standard streams 66-7
- start-up sequence 70
- stream number 60
- string operations 92
- subroutines 36, 38, 84, 104-28
  - calculator routines 104-6
  - DeBASE 11-28
  - interrupt driven sprite routine 107-11
  - Kempston Interface 106-7
  - Morex Interface 106-7
  - syntax checking 88
  - syntax flag 75
  - syntax/run flag 35
  - system variables 1, 42-61
  - 8K system 56-61
  - 16K system 42-56
  - ATTR P 55
  - ATTR T 55
  - BAUD 26, 57-8, 60
  - BORDCR 48
  - BREG 50

CH ADD	49	N_STR1	60
CH_ADD	75-6, 83	NTTYPE	59
CHADD_	58	NXTLIN	49
CHANS	48	OLDPPC	51
CHARS	44	OSPCC	51
COORDS	53	P FLAG	55
COPIES	61	PIP	45
CURCHL	10, 18, 48	PPC	47
DATADD	49	P POSN	53
DEFADD	44	P_RAMT	56
DEST	48	PR CC	53
DF CC	54	PROG	49
DFCCL	54	RAMTOP	56, 67
DFSZ	51	RASP	45
D_STR1	60	REPDEL	43
D_STR2	61	REPPER	42-4
ECHO E	54	S BRT	57
E LINE	48-9	SCR CT	54
EPPC	48	SECTOR	58
ERR NR	45	SEED	52
ERR SP	46	SEED_FL	58
FLAGS	45	S POSN	54
FLAGS 2	50	SPOSNL	54
FLAGS 3	57	S TOP	51
FLAGX	51	STRLEN	52
FRAMES	52	STRMS	44
HD_0B	61	S_STR1	60
HD_0D	61	STKBOT	50
HD_0F	61	STKEND	50
HD_00	61	SUBPPC	47
HD_11	61	T ADDR	52
JOBORD	58	T_STR1	60
K CUR	49, 67	TVDATA	44
KDATA	44	TVFLAG	46
KSTATE	42	UDG	53
LAST K	43	VARS	48
LIST SP	46	VECTOR	57, 75-6
L_STR1	60	WORKS P	50
MASK P	55	X PTR	49
MASK T	55		
MEM	50	tape socket	64
MEMBOT	56	tape-loading error	24
MODE	47	token code	13
NEWPPC	47		
NOT USED	53	ULA	70
NSPPC	47		
NTDCS	59	verifying	21-4
NTDEST	59		
NTHCS	59	Wait Input routine	10
NTLEN	59	Wait Key routines	21
NTNUMB	59	workspace	18
NTRESP	59	Write Record routine	32
NTSRCE	59	Write Sector routine	32
NTSTAT	58	Z80	6, 70, 72, 86