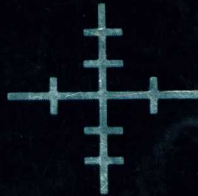


Assembly Language for Arcade Games and other Fast Spectrum Programs

Stuart Nicholls



FUEL:70

LASER:150

ENERGY:10

ASSEMBLY LANGUAGE FOR ARCADE GAMES
AND OTHER FAST SPECTRUM PROGRAMS

Assembly Language for Arcade Games and other Fast Spectrum Programs

Stuart Nicholls

McGRAW-HILL Book Company (UK) Limited

London · New York · St Louis · San Francisco · Auckland ·
Bogotá · Guatemala · Hamburg · Johannesburg · Lisbon ·
Madrid · Mexico · Montreal · New Delhi · Panama · Paris · San
Juan · São Paulo · Singapore · Sydney · Tokyo · Toronto

Published by
McGRAW-HILL Book Company (UK) Limited
MAIDENHEAD · BERKSHIRE · ENGLAND

British Library Cataloguing in Publication Data

Nicholls, S.A.

Assembly language for arcade games and other fast Spectrum programs.

1. Electronic games.
2. Sinclair ZX Spectrum (Computer).
3. Assembler language (Computer program language).

I. Title.

794.8'028'5424 GV1469.2

ISBN 0-07-084729-0

Library of Congress Cataloging in Publication Data

Nicholls, Stuart.

Assembly language for arcade games and other fast Spectrum programs.

1. Assembler language (Computer program language)
2. Computer games.
3. Sinclair ZX Spectrum (Computer) Programming.

I. Title.

QA76.73.A8N53 1984 001.64'2 83-24915

ISBN 0-07-084729-0

Copyright © 1984 McGraw-Hill Book Company (UK) Limited. All rights reserved.
No part of this publication may be reproduced, stored in a retrieval system, or
transmitted, in any form or by any means, electronic, mechanical, photocopying,
recording, or otherwise, without prior permission of McGraw-Hill Book Company
(UK) Limited, or of the original copyright holder.

1 2 3 4 5 CUP 8 6 5 4

Typeset by TC Photo-Typesetters, Maidenhead, England

Printed in Great Britain at the University Press, Cambridge

To my wife Fran for all her help in typing the original manuscript (a task carried out without understanding a word of it).

CONTENTS

Preface		ix
Chapter 1	Printing with colour	1
Chapter 2	PLOT, DRAW, and CIRCLE	8
	PLOT	8
	DRAW x,y,a	19
	CIRCLE x,y,r	19
Chapter 3	Counting	22
Chapter 4	Random numbers	29
Chapter 5	The keyboard	37
	Using the LAST KEY variable	37
	Using IN A, (C)	37
Chapter 6	Movement	42
	Foreground movement one character square	42
	Background movement one character square	44
	Pixel movement	60
Chapter 7	Music and sound effects	68
Chapter 8	ATTRIBUTE, SCREEN\$ and POINT	80
	ATTRIBUTE (line, column)	80
	SCREEN\$ (line, column)	82
	POINT	84
Chapter 9	The printer	85
	COPY	85
	LPRINT	85
	LLIST	86
Chapter 10	Program conversion	87
Appendix 1	Replacement ROM routines	114
Appendix 2	Spectrum Z80 machine code listings	119
Appendix 3	Decimal-hexadecimal conversion tables	125
Appendix 4	Useful calculator literals	127
Appendix 5	Memory map of display file	128
Appendix 6	Farewell program	129

PREFACE

So you have ploughed your way through handbooks or manuals on Z80 machine language programming for the Spectrum and have been given a fair knowledge of all the instructions! But just as they were getting interesting you end up with a few examples of how to add and subtract and are then left to fend for yourself.

This book will take you on to the next stage, carrying on where the others left off, and showing you how to *use* your knowledge of Spectrum routines to produce full working programs written entirely in machine code.

We shall look at the easier routines such as printing and also at the more complicated routines such as random numbers and the use of the calculator with floating point numbers.

The final Chapter shows how to apply the information given in the book to convert a BASIC games program entirely into machine code and, for those of you who cannot spend the time programming from the listings, an accompanying tape with demonstration programs of all the points discussed (including the BASIC and machine code versions of the main program for comparison) is available (see the last page in this book).

The machine code listings have all been produced using the McGraw-Hill Assembler. I cannot stress too highly that anyone who is serious about writing machine language programs should use an Assembler, since it does remove all the tedious table-searching normally associated with machine language and, more importantly, it enables you to modify your program easily when it crashes (which will happen all too often when you first start out).

1 PRINTING WITH COLOUR

Although printing in machine code is the most tedious and time-consuming of tasks, it is also one of the most important. A good program can be spoiled by a bad layout and display with perhaps ambiguous input prompts and, likewise, a mediocre program can be improved by a good layout. Before any program is developed, a good deal of thought must go into the visual presentation.

The Spectrum has 21 user-definable graphics (UDGs) so make full use of them. For example, if you are developing a space invaders type of game, do not stop at redefining the invaders, bombs, and shots; use the remaining characters to redefine numbers zero to nine to give futuristic characters for scoring or to create interesting landscapes. Use colours sensibly; do not put *red* and *magenta* together (or *yellow* and *white*) and it may be wise to run your program on a black and white set to see whether it still looks right since many Spectrum owners will not have easy access to a colour set.

As every program requires printing of information in one form or another, in this Chapter we will look at methods of converting BASIC PRINT into machine code.

The first BASIC program command usually involves the setting up of BORDER, PAPER, and INK permanent colours followed by CLS to put these colours on to the screen.

The BORDER colour is probably the easiest to set up in machine code and takes just five bytes. The A register is first loaded with the 'colour number' required and a call then made to the ROM routine 8859 d (229Bh). This routine sets the screen to the required border

```
10 BORDER 5: REM green
```

```
org 23760
23760 3E 05      ld a,5

CALL BORDER SET

23762 CD 9B 22   call 8859
23765 C9        ret
```

Program 1.1

and stores the colour value in the systems variable BORDCR 23624 (the value stored is in fact eight times the colour value) that is bits 5 – 3. The other bits are used to store the INK, BRIGHT, and FLASH parameters for the lower half of the screen (the *input lines*).

Program 1.1 shows the machine code version for the setting of BORDER 5 (green).

The permanent colours for PAPER and INK are held in the systems variable ATTR-P (23693). Bits 0–2 hold the INK colours and bits 3–5 the PAPER colours. The remaining bits 6 and 7 are used to indicate FLASH and BRIGHT. For FLASH 0 bit 7 is reset; for FLASH 1 bit 7 is set. BRIGHT 0 has bit 6 reset and BRIGHT 1 has bit 6 set.

Program 1.2 shows the setting of PAPER 3: INK 1: CLS and demonstrates the fact that ATTR-P (23693) can be regarded as IY + 83. The IY register is used as a pointer to systems variable and holds value 23610. If you wish to 'LD(23609), A' for example this can be considered as (IY – 1) or (IY + 255).

5 REM set up permanent screen	Call open channel 2	
colours		
10 PAPER 3: INK 1: CLS	23766 3E 02	ld a,2
	23768 CD 01 16	call 5633
	call CLS	
Ld (ATTR-P),colours		
	23771 CD 6B 0D	call 3435
23762 FD 36 53 19 ld (iy+83),25	23774 C9	ret

Program 1.2

The CLS command CALL 3435 clears the screen and sets the screen parameters as in the BASIC CLS *but* you will notice that before the call to CLS is made there is a call to ROM routine 5633. This is most important and will be used many times throughout this Chapter. The call is necessary to open channel 2 allowing information to go to the top half of the screen, that is, lines 0 to 21. Once our permanent screen parameters are set up, we can move on to actually printing on the screen.

Program 1.3 shows the method of printing a character on the screen in the *next* print position; this will be 0,0; if a CLS command has been executed. You will notice again that 'channel 2' requires opening first and that the code of the character to be printed is held in the A register.

The ROM routine to print a character is called using RST 16 d which in fact is a call to routine 5618 d. This routine prints the character in the A register and moves the print position along one place.

```

REM print a character using permanent colours
20 PRINT "A";

org 23760

Open channel 2

23760 3E 02      ld a,2
23762 CD 01 16   call 5633

LOAD ACC. CODE CHR A
23765 3E 41      ld a,65

CALL PRINT

23767 D7        rst 16
23768 C9        ret

```

Program 1.3

Program 1.4 expands the above to enable a 'short' word to be printed. Again channel 2 is opened and the A register loaded with each character in turn until the whole word is printed.

```

10 REM Print a string of chrs.
20 PRINT "HELLO"

org 23760

Open channel 2

23760 3E 02      ld a,2
23762 CD 01 16   call 5633
23765 3E 48      ld a,72
H
23767 D7        rst 16
23768 3E 45      ld a,69

E
23770 D7        rst 16
23771 3E 4C      ld a,76
L
23773 D7        rst 16
23774 3E 4C      ld a,76
L
23776 D7        rst 16
23777 3E 4F      ld a,79
O
23779 D7        rst 16
23780 C9        ret

```

Program 1.4

You can see however that the above method would become time and memory consuming if a long word or sentence were to be printed and would be better tackled in a different way.

There are several options open to us as are demonstrated in Program 1.5. The main idea is to hold the message as DATA to be read one character at a time until all of DATA is printed.

```

10 REM Print a LONG string
20 Print "HELP ME"

org 23760

23760 3E 02      ld a,2
23762 CD 01 16   call 5633
23765 11 E4 5C   ld de,DATA
23768 01 07 00   ld bc,7

Loop
23771 78        ld a,b
23772 B1        or c
23773 0B        dec bc
23774 C8        ret z
23775 1A        ld a,(de)
23776 13        inc de
23777 D7        rst 16
23778 18 F7     jr Loop
DATA
defb 72 69 76 80 32 77 69

```

String length

Program 1.5

In *version 1* the DE register holds the address of the start of DATA and is used as a pointer to each character; the BC register holds the length of the string and an RST 16 call is made for each character in turn until BC = 0.

```

org 23760                                CALL PRINT STRING
23760 3E 02                                ld a,2
23762 CD 01 16                            call 5633
23765 11 DF 5C                            ld de,DATA
23768 01 07 00                            ld bc,7
23771 CD 3C 20                            call 8252
23774 C9                                    ret
DATA
defb 72 69 76 80 32 77 69

```

Version 2 is in fact the same as version 1 except that the ROM routine 8252d is used which in fact is a copy of version 1 routine from address 23771 to 23778.

```

org 23760                                23774 13                                inc de
23760 3E 02                                ld a,2                                23775 18 F7                                jr Loop
23762 CD 01 16                            call 5633                            END
23765 11 E5 5C                            ld de,DATA                            23777 CB BF                                res 7,a
Loop                                        23779 D7                                rst 16
23768 1A                                    ld a,(de)                            23780 C9                                ret
23769 CB 7F                                bit 7,a                                DATA
23771 20 04                                jr nz,END                            defb 72 69 76 80 32 77 197
23773 D7                                rst 16

```

Version 3 can be used for printing strings of any length but is limited to character codes 0 to 127. The last character is used as a marker to indicate the end of the string by having 128 added to its code, in effect setting bit 7, then before a character is printed a check is made to see whether bit 7 is zero. If it is then the PRINT loop is repeated; if not, a jump is made to the END routine which resets bit 7,A, prints the character, and continues with the program, in this case RET.

Version 3 can be used as a 'PRINT subroutine' in a program where several messages can be stored together with end markers and all that would be necessary would be to load DE with the start address of the message required and to make a call to the PRINT subroutine. A similar, though more complicated, routine is used by the Spectrum for printing keywords and error messages; this routine starts at address 3082 d.

The error messages 1 to R as listed in the Spectrum manual Appendix B can be printed easily by using the RST 8 command. The byte following the RST 8 instruction points to the error message required. For example, if you require the message 'K invalid colour' which is in fact the twenty-first error message, then use:

```

RST 8
defb 19d (message number - 2)

```

There is no need for a return instruction as the ROM will automatically return to BASIC after printing the error.

We can now move on to printing characters or strings at a specific line and column, as in the BASIC PRINT AT line, column; 'message'. Again, if you look at Appendix A of the Spectrum manual, you will see that character codes 6 to 23 are control codes and in particular code 22 is the code for 'AT'. Now luckily for us the RST 16 instruction will recognize code 22 as 'PRINT AT' and will use the next two bytes of DATA as the x,y; coordinates to give the start position of the message. Program 1.6 demonstrates this using defb 22,3,5 for PRINT AT 3,5;

```

10 REM Print string at x,y          23768 01 0C 00      ld bc,12
20 PRINT AT 3,5;"Thank you"        23771 CD 3C 20      call 8252
                                   23774 C9              ret
                                   DATA
org 23760                            AT 3 5
23760 3E 02          ld a,2
23762 CD 01 16      call 5633      defb 22 3 5
23765 11 DF 5C      ld de,DATA      defs Thank you

```

Program 1.6

The same principle can be used for TAB (CODE 23). Note also that ENTER (CODE 13d) will give the same effect as NEWLINE.

So now we should be able to print any message in any position on lines 0 to 21 using the permanent colours. All that now remains to complete normal screen printing is to print messages in local colours, BRIGHT, FLASH, OVER, and INVERSE.

Program 1.7 shows how this is achieved and follows the same principles as Program 1.6, in that the control codes for PAPER, INK, BRIGHT, etc. are held in DATA with the byte following the control code specifying the parameters.

```

1000 REM Printing in colours          RESET PERMANENT COLOURS
1010 PRINT AT 4,7; PAPER 2; INK      AND BRIGHT 0 FLASH 0
4; FLASH 1; BRIGHT 1; OVER 1; IN    23774 CD 4D 0D      call 3405
VERSE 1;"Can you read this"        23777 C9              ret
org 23760                            DATA
23760 3E 02          ld a,2          defb 22 4 7 17 2 16 4 18 1
23762 CD 01 16      call 5633      defb 19 1 21 1 20 1
23765 11 E2 5C      ld de,DATA      defs Can you read this
23768 01 24 00      ld bc,36      defb 21 0 20 0
23771 CD 3C 20      call 8252

```

Program 1.7

After local colours are used it is necessary to reset the permanent colours for further printing. In the case of OVER and INVERSE this requires the resetting to be carried out using RST 16, but the PAPER, INK, BRIGHT, and FLASH can all be reset by a call to the ROM routine 3405d which copies the value in ATTR P (23693) into ATTR T (23695).

Finally, we consider printing on the 'lower half' of the screen, that is the input lines 22 and 23. This uses exactly the same method as described for printing on the upper half, except that a different channel requires opening before RST 16 is used. Program 1.8 gives an example of how this is achieved.

```

10 REM Printing on INPUT lines      WAIT
20 PRINT #0 AT 0,10;"Input line    23774 76          halt
0";AT 1,10;"Input line 1"
30 PAUSE 0

                                If key pressed then
                                BIT 5,FLAGS will be SET

                                23775 FD CB 01 6E bit 5,(iy+1)
                                23779 28 F9          jr z,WAIT
                                23781 FD CB 01 AE res 5,(iy+1)
                                23785 C9          ret
                                DATA

org 23760

Open channel -3

23760 3E FD          ld a,253
23762 CD 01 16      call 5633          defb 22 0 10
23765 11 EA 5C      ld de,DATA        defs Input line 0
23768 01 1E 00      ld bc,30          defb 22 1 10
23771 CD 3C 20      call 8252        defs Input line 1

```

Program 1.8

The channel in this case is -3 (253).

The BASIC program is interesting in that it is not mentioned in the manual that PRINT # 0; will print on the input lines. The PAUSE 0 instruction is necessary to stop the program without the instruction 0 O.K. which would remove the input line printing.

Similarly this machine code routine requires the equivalent of PAUSE 0. This is achieved by continually checking bit 5 of FLAGS (23611) or (IY + 1) after a HALT instruction. The HALT instruction in effect waits for the next KEYSKAN routine and, if a key is pressed, sets bit 5 of FLAGS. So we wait in the loop until bit 5 is set and then return to BASIC. Note that line 22 is counted as line 0 and line 23 as line 1.

One ROM routine that can be very useful for clearing printing on the input lines but leaves the upper screen intact, can be found at 3652, which is a 'clear specified number of lines routine' the number of lines are counted from the bottom of the screen and must be greater than 0. The number of lines is placed in the B register (1-24) and then the call made; the colours used to clear the lines are those held in ATTR P. The input lines can be cleared with the instructions

```
LD B, 2
CALL 3652
(Return)
```

Before moving on to the next Chapter there is an important point to note concerning user-definable graphics. If you have written a program that is intended to run on both 16K and 48K machines then the UDGs will need to be correctly located on each machine. The easiest way to achieve this is to set up the UDGs starting at address 32600, that is for a 16K machine, and then to use Program 1.9 to relocate the UDGs as necessary to the address held in the systems variable UDG (23675).

```
10 REM Transfer 16K UDG's to          90 IF bc<>0 THEN GO TO 50
correct location for 48K machine     100 REM return
20 LET de=PEEK 23675+256*PEEK
23676
30 LET h1=32600
40 LET bc=166
50 POKE de,PEEK h1
60 LET h1=h1+1
70 LET de=de+1
80 LET bc=bc-1
org 23760
23760 ED 5B 7B 5C ld de,(23675)
23764 21 58 7F ld h1,32600
23767 01 A8 00 ld bc,168
23770 ED B0 ldir
23772 C9 ret
```

Program 1.9

2 PLOT, DRAW, AND CIRCLE

PLOT

The BASIC command PLOT x,y involves the setting of *pixel* column x, row y in the permanent ink colour. Pixel position 0,0 is considered as the bottom left-hand corner of the screen and 175, 255 as the top right-hand corner.

In machine language there are two useful entry points to the ROM PLOT routine: the first of these, which is probably the more useful and the easier to use, is CALL 8933. Before the routine is called the B register must hold the value of y (in the range 0 to 175) and the C register must hold the value of x (0–255). Program 2.1 shows the simple BASIC command PLOT 75,125.

```
1000 REM Plot x,y
1010 PLOT 75,175
```

```
org 23760
23760 06 7D      ld b,125
23762 0E 4B      ld c,75
23764 CD E5 22   call 8933
23767 C9        ret
```

Program 2.1

Using decimal instead of hexadecimal numbers it is easier to follow the program by loading the B and C registers separately. We could of course have saved memory by using LD BC, 32075 but this would have been very confusing.

The second entry point is at 8924. This requires the values of x and y to be placed on the *calculator stack* with the value of y being the top value. Program 2.2 demonstrates this with the same PLOT 75,125.

```
org 23760
23760 3E 4B      ld a,75
                                STACK 175
                                23767 CD 28 2D      call 11560
                                CALL PLOT
                                23762 CD 28 2D      call 11560
                                23765 3E 7D      ld a,125
                                23770 CD DC 22      call 8924
                                23773 C9        ret
```

Program 2.2

The ROM routine at 11560 is used to place the value held in the A register on top of the calculator stack. With the PLOT routine entered at 8924, the top two values are removed from the stack and the pixel plotted. The calculator stack thus being reduced by two values.

The use of the calculator will be discussed more fully in Program 2.5. This second entry point is useful if the calculator has been used to manipulate a formula for plotting graphs etc.

Program 2.3 shows how PLOT OVER 1; x, y is achieved.

10 REM Plot OVER 1;x,y	23765 06 28	ld b,40	
20 PLOT OVER 1;25,40	23767 0E 19	ld c,25	
	23769 CD E5 22	call 8933	
org 23760	SET OVER 0		
SET OVER 1	SET BITS 0 & 1 OF P-FLAG		
SET BITS 0 & 1 OF P-FLAG			
	23772 AF	xor a	
23760 3E 03	ld a,3	23773 FD 77 57	ld (iy+87),a
23762 FD 77 57	ld (iy+87),a	23776 C9	ret

Program 2.3

This involves the setting of bits 0 and 1 of P FLAG (IY + 87) before the PLOT call is made, and then resetting 0 and 1 of P FLAG afterwards. If OVER 0 is not reset then any future plotting or printing will be carried out as OVER 1.

PLOT INVERSE is identical in form to PLOT OVER requiring the setting of bits 2 and 3 of P FLAG; the systems variable thus being set to value 12.

Using entry point 8933 we can write a program to plot a character anywhere on the screen. The only limitations being that the code of the character must be from 32 (space) to 127 (copyright), that is, one that has its eight-byte make-up held in the ROM character generator.

org 23760	Load L with code of CHR.	Byte count	
23760 2E 7F	ld l,127		
23762 26 00	ld h,0		
	Multiply x8	23772 06 08	ld b,8
		L3	
		Store pointer HL	
23764 29	add hl,hl		
23765 29	add hl,hl	23774 E5	push hl
23766 29	add hl,hl		
		Bit count	
	Add to (23606) to find start of character	23775 0E 08	ld c,8
		23777 7E	ld a,(hl)
23767 ED 5B 36 5C	ld de,(23606)	L2	
23771 19	add hl,de	23778 C5	push bc

Get plot position x,y		23798 C1	pop bc
		23799 0D	dec c
23779 ED 4B 0E 5D	ld bc, (XY)	23800 20 E8	jr nz, L2
Check if BIT SET		23802 C5	push bc
			Reset PLOT to x-8,y-1
23783 17	rla		
23784 F5	push af		
		23803 ED 4B 0E 5D	ld bc, (XY)
		23807 05	dec b
23785 30 05	jr nc, L1	23808 3E F8	ld a, 248
		23810 81	add a, c
If SET then PLOT		23811 4F	ld c, a
		23812 ED 43 0E 5D	ld (XY), bc
23787 C5	push bc	23816 C1	pop bc
23788 CD E5 22	call 8933	23817 E1	pop hl
23791 C1	pop bc	23818 23	inc hl
L1			
			Repeat until BYTE count=0
Move PLOT to x+1,y		23819 10 D1	djnz L3
		23821 C9	ret
23792 0C	inc c	XY	
23793 ED 43 0E 5D	ld (XY), bc	defb 100 20	
23797 F1	pop af		

Program 2.4

You can see from Program 2.4 that the code of the character required is placed in the L register (address 23761) and is then multiplied by eight and added to the value stored in the systems variable (23606) to find the start of the eight bytes that make up the character shape in the character generator. The HL register is then used as a *pointer* holding the address of the character byte being examined. The BC register is used as a counter for the eight bit \times eight bit matrix of the character and the A register is used to hold the byte being examined.

An RLA instruction is carried out eight times on each character byte and if a bit is set, that is, if after RLA the CARRY flag is set then the pixel is plotted.

After each RLA instruction the x, y coordinates are updated. The x, y start coordinates denote the top left-hand pixel position of the character being plotted. To remain on the screen therefore the value of y must be between 7 and 175. The value of x can be from 0 to 255 as any attempt to take x over 255 will result in the value being reset to 0, that is, if $C = 255$ then 'inc C' will give $C = 0$.

This program can be easily modified to plot characters with a 90 degrees anticlockwise rotation.

org 23760		23785 30 05	jr nc,L1
Load L with code of CHR.			
23760 2E 60	ld l,96		If SET then PLOT
23762 26 00	ld h,0		
Multiply x8		23787 C5	push bc
		23788 CD E5 22	call 8933
		23791 C1	pop bc
23764 29	add hl,h1		L1
23765 29	add hl,h1		
23766 29	add hl,h1		Move PLOT to x,y+1
Add to (23606) to find			
start of character		23792 04	inc b
		23793 ED 43 0E 5D	ld (XY),bc
23767 ED 5B 36 5C	ld de,(23606)	23797 F1	pop af
23771 19	add hl,de	23798 C1	pop bc
		23799 0D	dec c
Byte count		23800 20 E8	jr nz,L2
		23802 C5	push bc
			Reset PLOT to x+1,y-8
23772 06 08	ld b,8		
L3			
Store pointer HL			
23774 E5	push hl	23803 ED 4B 0E 5D	ld bc,(XY)
		23807 0C	inc c
Bit count		23808 3E F8	ld a,248
		23810 80	add a,b
23775 0E 08	ld c,8	23811 47	ld b,a
23777 7E	ld a,(hl)	23812 ED 43 0E 5D	ld (XY),bc
L2		23816 C1	pop bc
23778 C5	push bc	23817 E1	pop hl
		23818 23	inc hl
Get plot position x,y			Repeat until BYTE count=0
23779 ED 4B 0E 5D	ld bc,(XY)		
Check if BIT SET		23819 10 D1	djnz L3
		23821 C9	ret
23783 17	rla	XY	
23784 F5	push af	defb 100 20	

Program 2.5

In Program 2.5 the x,y coordinates are updated to give a *vertical* byte plot instead of a *horizontal* one as in Program 2.4; there the x,y start coordinates denote the *bottom* left-hand pixel, and so the range for y is 0 to 167.

Having studied the method of plotting a character we can now move on to our first serious program to *plot a message* anywhere on the screen and, to make it more useful, we shall allow the character height and width to be selected. A copy of the program is available on tape – LOAD 'PLOTDEMO'.

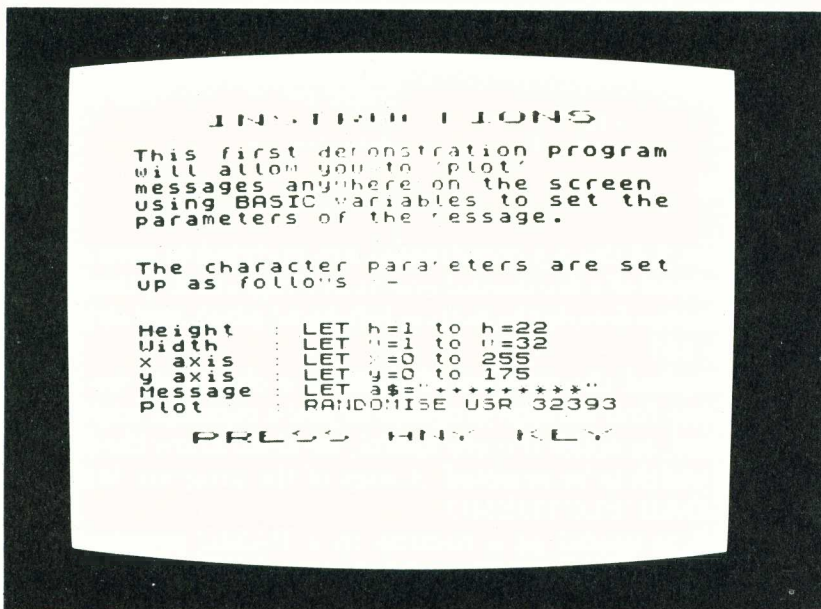
Because it is useful as a routine in a BASIC program I have attached it to a 'search' routine so that the parameters height/width, x/y plot positions, and message can be set up using

BASIC variables and so negating the need to POKE them into machine code. If you wish to use the routine within a machine code program then the search routine is unnecessary and can be omitted, the parameters can be copied directly into the printer buffer. The BASIC program is self-explanatory and allows the reader to experiment with different values of height/width, x,y, and message (a\$).

The machine code is an extension of Program 2.4. The values of x,y,h, and w are found using the search routine. This routine works through the variables store until the required variable CODE is found and then stores the value of the variable (0-255) in the *printer buffer*. Once the simple variable value has been stored, the LEN a\$ is found and stored, again (0-255), and finally the message is copied into the Printer buffer (a most useful place for storing data). The 'PLOT' routine includes a CATCH-ALL instruction and will reset any wrong parameters to acceptable ones, and consists of five nested loops:

- (L1) Plot a SET bit for WIDTH w
- (L2) Repeat L1 for eight bits
- (L3) Repeat HEIGHT h times
- (L4) Repeat for eight bytes
- (L5) Repeat until end of message

But note program comment on FOR/NEXT LOOP variables.



You may set up ink colours prior to the machine code call BUT remember to reset the permanent colour afterwards.

Any incorrect parameters will be corrected by the M/code, and if the message is too long/tall it will wrap around the screen.

NOTE

DO NOT USE THE VARIABLES X,Y,H,W AS FOR/NEXT LOOP CONTROLS AS THE PROGRAM WILL BECOME CORRUPTED.

The following program will ask for the parameters required and then plot your message.

PROGRAM: PLOT.M

```

5 OVER 0: INK 0: PAPER 6: BORD
ER 3: CLS
10 LET x=24: LET y=79: LET h=5:
LET w=2: LET a$="STOP THE TAPE"
15 INK 2: RANDOMIZE USR 32393
18 PAUSE 100
20 LET w=8: LET y=150: LET x=0:
LET a$="PLOT"
30 INK 1: RANDOMIZE USR 32393
35 LET a$="____": LET y=140
36 INK 1: RANDOMIZE USR 32393
40 FLASH 1: GO SUB 8000
50 FLASH 0
60 PAUSE 0
70 PAPER 7: INK 0: BORDER 7: CL
S
80 LET x=32: LET y=175: LET h=1
: LET w=2: LET a$="INSTRUCTIONS"
90 INK 2: RANDOMIZE USR 32393:
INK 0
100 PRINT AT 2,0;"This first dem
onstration program will allow you
to 'plot' messages anywher
e on the screen using BASIC varia
bles to set the parameters of the
message."
110 PRINT ""The character param
eters are setup as follows :-"
120 PRINT ""Height : LET h=1 t
o h=22""Width : LET w=1 to w=3
2"
130 PRINT "x axis : LET x=0 to
255""y axis : LET y=0 to 175"
140 PRINT "Message: LET a$=";CH
R$ 34;"*****";CHR$ 34
150 PRINT "Plot : RANDOMISE U
SR 32393"
160 GO SUB 8000
170 IF INKEY$<>"" THEN GO TO 17
0
175 IF INKEY$="" THEN GO TO 175
178 CLS
180 PRINT "You may set up ink co
lours prior to the machine code ca
ll BUT remember to reset the p
ermanent colour afterwards."
190 PRINT ""Any incorrect parame
ters will be corrected by the M
/code, and if the message is too
long/tall it will wrap around the
screen."
200 LET a$="NOTE": LET y=95: INK
2: RANDOMIZE USR 32393: INK 0
210 PRINT AT 12,0;"DO NOT USE TH
E VARIABLES x,y,h,w AS FOR/NEXT LO
OP CONTROLS AS THE PROGRAM WILL BE
COME CORRUPTED."
220 PRINT ""The following progra
m will ask for the parameters re
quired and then plot your message
."
230 GO SUB 8000
235 IF INKEY$<>"" THEN GO TO 23
5
240 IF INKEY$="" THEN GO TO 240
242 CLS : BORDER 4
245 PLOT 0,121: DRAW 255,0: DRAW
0,-73: DRAW -255,0: DRAW 0,73
250 PRINT AT 8,0;"HEIGHT 1-22 ?
__","WIDTH 1-32 ? __","X

```

axis 0-255 ?	___", "Y axis 0-1	32355 CB 76	bit 6, (h1)
75 ?	___", "MESSAGE ?", "	32357 20 0C	jr nz, L3
	----- INK	32359 23	inc hl
0-9 ?",,			
255 LET z\$="	"		
	"		
257 LET v\$="	"	32360 7E	ld a, (h1)
		32361 CB 7F	bit 7, a
260 PRINT OVER 1; AT 8, 0; z\$		32363 28 FA	jr z, -6
262 INPUT h: PRINT AT 8, 18; h: "		32365 11 06 00	ld de, 6
; OVER 1; AT 8, 0; v\$		32368 19	add hl, de
265 PRINT OVER 1; AT 9, 0; z\$: INP		32369 18 DF	jr L2
UT w: PRINT AT 9, 18; w: "		L3	
; OVER 1		32371 CB 7E	bit 7, (h1)
; AT 9, 0; v\$		32373 28 F6	jr z, -10
270 PRINT OVER 1; AT 10, 0; z\$: IN		32375 11 13 00	ld de, 19
PUT x: PRINT AT 10, 18; x: "		32378 19	add hl, de
; OVER 1		32379 18 D5	jr L2
; AT 10, 0; v\$		SET	
272 PRINT OVER 1; AT 11, 0; z\$: IN		32381 32 B0 5C	ld (23728), a
PUT y: PRINT AT 11, 18; y: "		32384 CD 4F 7E	call FIND
; OVER 1		32387 23	inc hl
; AT 11, 0; v\$		32388 23	inc hl
273 PRINT OVER 1; AT 12, 0; z\$: AT		32389 23	inc hl
13, 0; z\$: INPUT a\$: PRINT AT 13, 0;		32390 7E	ld a, (h1)
a\$; FOR k=LEN a\$ TO 31: PRINT "		32391 02	ld (bc), a
"; NEXT k: PRINT OVER 1; AT 12, 0		32392 C9	ret
; v\$; OVER 1; AT 13, 0; v\$			
275 PRINT OVER 1; AT 14, 0; z\$: IN			
PUT i: PRINT AT 14, 0; INK i; OVER			
1; v\$			
280 PAUSE 50			
290 CLS : INK I: RANDOMIZE USR 3	START		
3293: INK 0	32393 01 00 5B	ld bc, 23296	
300 PRINT #0; AT 0, 0; "PRESS 0 TO	32396 3E 78	ld a, 120	
STOP"; AT 1, 0; "ANY OTHER KEY FOR A	32398 CD 7D 7E	call SET	
NOTHER 60"	32401 03	inc bc	
310 PAUSE 0: IF INKEY\$="q" OR IN	32402 3E 79	ld a, 121	
KEY\$="0" THEN STOP	32404 CD 7D 7E	call SET	
320 CLS : GO TO 245	32407 03	inc bc	
8000 LET h=1: LET w=2: LET x=24:	32408 3E 68	ld a, 104	
LET y=15: LET a\$="PRESS ANY KEY":	32410 CD 7D 7E	call SET	
INK 2: RANDOMIZE USR 32393: INK	32413 03	inc bc	
0	32414 3E 77	ld a, 119	
8010 RETURN	32416 CD 7D 7E	call SET	
9000 STOP	32419 3E 41	ld a, 65	
9800 CLEAR 32334: "LOAD ""CODE : G	32421 32 B0 5C	ld (23728), a	
0 TO 5	32424 CD 4F 7E	call FIND	
9900 SAVE "PLOTDEMO" LINE 9800	32427 23	inc hl	
9950 SAVE "LARGE" CODE 32335, 265	32428 5E	ld e, (h1)	
org 23760 32335	32429 23	inc hl	
FIND	32430 56	ld d, (h1)	
32335 2A 4B 5C	32431 ED 53 04 5B	ld (23300), de	
L2			
32338 3A B0 5C			
32341 BE	32435 D5	push de	
32342 C8	32436 C1	pop bc	
32343 CB 6E	32437 23	inc hl	
32345 20 08	32438 11 05 5B	ld de, 23301	
32347 23	32441 ED B0	ldir	
32348 5E	32443 2A 00 5B	ld hl, (23296)	
32349 23	32446 AF	xor a	
32350 56	32447 7C	ld a, h	
32351 19	32448 DE B0	sbc a, 176	
32352 23	32450 38 04	jr c, +4	
32353 18 EF	32452 67	ld h, a	
L1	32453 22 00 5B	ld (23296), hl	

32456	22	B0	5C	ld (23728),hl	32527	F1	pop af			
32459	21	05	58	ld hl,23301	32528	C1	pop bc			
LP5					32529	10	D3	djnz LP2		
32462	E5			push hl	32531	3A	00	5B	ld a,(23296)	
32463	7E			ld a,(hl)	32534	21	B0	5C	ld hl,23728	
32464	26	00		ld h,0	32537	77			ld (hl),a	
32466	6F			ld l,a	32538	23			inc hl	
32467	29			add hl,hl						
32468	29			add hl,hl						
32469	29			add hl,hl	32539	AF			xor a	
32470	11	00	3C	ld de,15360	32540	7E			ld a,(hl)	
32473	19			add hl,de	32541	DE	B0		sbc a,176	
32474	06	08		ld b,8	32543	38	03		jr c,+3	
LP4					32545	77			ld (hl),a	
32476	C5			push bc	32546	18	08		jr +8	
32477	ED	4B	01	5B	ld bc,(23297)	32548	7E		ld a,(hl)	
LP3					32549	FE	00		cp 0	
32481	7E			ld a,(hl)	32551	20	02		jr nz,+2	
32482	E5			push hl	32553	36	B0		ld (hl),176	
32483	C5			push bc	32555	35			dec (hl)	
32484	06	08		ld b,8	32556	C1			pop bc	
LP2					32557	E1			pop hl	
32486	C5			push bc	32558	10	B1		djnz LP3	
32487	17			rla	32560	23			inc hl	
32488	F5			push af	32561	C1			pop bc	
32489	DA	F9	7E	jp c,PLOT	32562	10	A8		djnz LP4	
32492	2A	03	5B	ld hl,(23299)	32564	3A	03	5B	ld a,(23299)	
32495	3A	B0	5C	ld a,(23728)	32567	87			add a,a	
32498	85			add a,l	32568	87			add a,a	
32499	52	B0	5C	ld (23728),a	32569	87			add a,a	
32502	C3	0F	7F	jp SKIP	32570	6F			ld l,a	
PLOT					32571	3A	B0	5C	ld a,(23728)	
32505	ED	4B	B0	5C	ld bc,(23728)	32574	85		add a,l	
LP1					32575	32	00	5B	ld (23296),a	
32509	C5			push bc	32578	32	B0	5C	ld (23728),a	
32510	ED	4B	B0	5C	ld bc,(23728)	32581	3A	01	5B	ld a,(23297)
32514	C5			push bc	32584	32	B1	5C	ld (23729),a	
32515	CD	E5	22	call 8933	32587	E1			pop hl	
32518	C1			pop bc	32588	23			inc hl	
32519	0C			inc c	32589	3A	04	5B	ld a,(23300)	
32520	ED	43	B0	5C	ld (23728),bc	32592	3D		dec a	
32524	C1			pop bc	32593	C8			ret z	
32525	10	EE		djnz LP1	32594	32	04	5B	ld (23300),a	
SKIP					32597	C3	CE	7E	jp LP5	

Program 2.6

The machine code in Program 2.6 starts at 32393. The first routine, from 32393 to 32418, initializes the BC register as a pointer to the printer buffer and calls the SET subroutine to set the code of the variable being located into address 23728. The SET routine in turn calls the FIND routine to locate the variables code. The value of the variable (0-255) is then stored in the printer buffer address pointed to by the BC register. So variables x,y,h, and w are found and stored consecutively. The routine from 32419 to 32442 finds the code of variable a\$, the length (0-255) is then stored and finally the string copied. The routine from 32443 to 32458 checks the value of y

(and if it is greater than 175 sets it to $y - 176$) and then stores the values x and y back into 23296/7 and also 23728/9.

The remaining machine code is an extension of the character plot routine with the additional loops to PLOT the pixel 'width w ' times for 'h lines', a check being made after each 'line' to ensure that the next PLOT position stays on the screen. The routine from 32540 to 32554 checks the next PLOT position and gives a wrap-around effect if it is off the bottom of the screen.

Without going too deeply at this stage into the CALCULATOR routines we can look at a program that makes use of PLOT at entry point 8924.

Program 2.7 shows how a sine wave can be plotted using CALCULATOR to manipulate the formula:

$$88 + 80 * \text{SIN} (A/128 * \text{PI})$$

```

1000 FOR A=0 TO 255
1010 PLOT A,88+80*SIN (A/128*PI)
1020 NEXT A

org 23760
LET A=0

23760 AF      xor a
LOOP
STORE A
23761 F5      push af
STACK A
23762 CD 28 2D call 11560
23765 3E 58   ld a,88
23767 CD 28 2D call 11560
23770 3E 50   ld a,80
23772 CD 28 2D call 11560
23775 F1      pop af
23776 F5      push af
23777 CD 28 2D call 11560
23780 3E 80   ld a,128
23782 CD 28 2D call 11560

STACK 88 80 A 128

USE CALCULATOR

23785 EF      rst 40
Divide top 2 values (A/128)

defb 5
Stack PI/2
defb 163
Duplicate top values
defb 49
Add top 2 values (PI)
defb 15
Multiply top 2 values
(A/128*PI)
defb 4
SIN top value SIN (A/128*PI)
defb 31
Multiply top 2 values
80*SIN (A/128*PI)
defb 4
Add top 2 values
88+80*SIN (A/128*PI)

defb 15
End calculator routine
Top 2 values are
A and 88+80*SIN (A/128*PI)
defb 56
CALL PLOT
23795 CD DC 22 call 8924
Get value A
23798 F1      pop af
23799 3C      inc a
Check not zero (256)
23800 FE 00   cp 0
23802 20 D5   jr nz,LOOP
23804 C9      ret

```

Program 2.7

The CALCULATOR routine can be made to perform complex arithmetic by the use of *literals*. These are in fact ROM calls to perform calculations involving, usually, the top two values on the calculator stack, but may also be applied to string handling. For

instance, if the top two values were 1234 and 2, and Literal 4 were called from within the CALCULATOR routine, then these two values would be removed from the stack, multiplied, and their product put back on top of the stack; the stack would therefore be reduced by one value and the top value would be 2468.

Positive integer numbers can be placed on the stack by means of:

```
STACK VAL A – CALL 11560 OR STACK VAL BC – CALL –  
11563.
```

In Program 2.7 I have used STACK VAL A to place in order, 'variable A', 88, 80, 'variable A', and 128.

The USE CALCULATOR routine is then called by the RST 40 instruction. The CALCULATOR routine works through the bytes following RST 40 and acts on each one calling further subroutines as required. In our program the next byte is 5, and Literal 5 means 'GOSUB DIVIDE'. The top two values on the stack are then taken, the top one divided into the one underneath and the answer placed back on the stack. The next byte in the routine is Literal 163; this tells the calculator to 'STACK PI/2'.

Literal 49 is DUPLICATE TOP VALUE

Literal 15 is ADD TOP TWO VALUES – thus restacking the sum

Literal 4 is MULTIPLY TOP TWO VALUES

Literal 32 is SINE OF TOP VALUE – thus replacing top value with its sine

There are many more literals that can be used in games programs and these will be discussed in future Chapters.

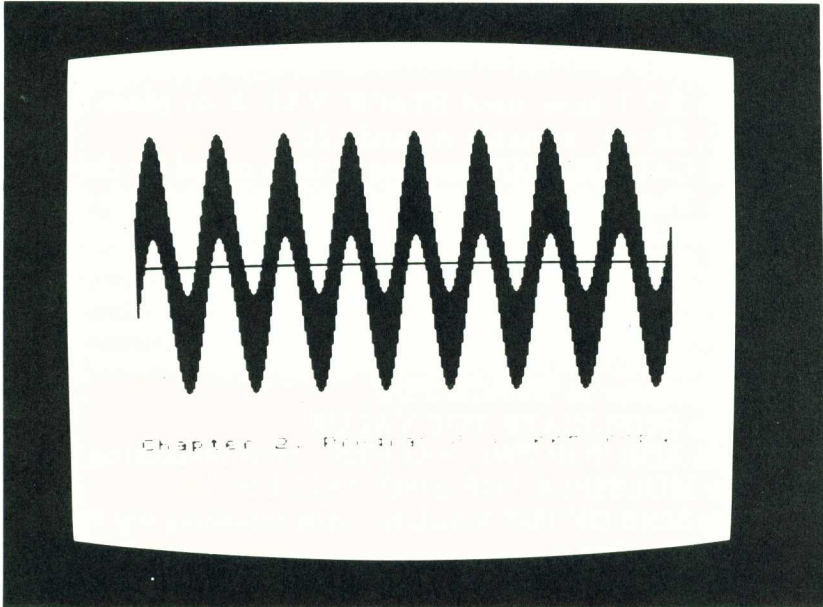
The last Literal 56 is 'END CALCULATOR ROUTINE CALL' which *must* be used to return to your machine code program. One other important point to bear in mind when using the CALCULATOR routine is that, before a 'safe' return to BASIC can be made, the stack must be cleared of all values previously stacked.

At the end of our CALCULATOR routine we are left with two values on the calculator stack, the top value being '88+80 * SIN (A/128 * P1) and the other value 'A'. The PLOT call 8924 then takes these two values, thus resetting the stack, and PLOTS the pixel. The value 'A' is then incremented and the LOOP repeated until 'A' = 256 (that is, 'A' is reset to 0).

You can see from this program that the CALCULATOR's literals are very useful and that a fairly complex formula has been dealt with in just eight bytes. In fact it takes more bytes to set up the stack than to perform the calculations.

DRAW

Again there are two entry points in ROM for the DRAW x, y command. The first point call 9402 requires the ABS value of x to be in the C register and the ABS value of y in the B register. The DE register is used to hold the SGN of x and y; the D register holding SGN x (1 if positive, 255 if negative) and the E register holding SGN y. Program 2.8 is a modified version of Program 2.7 to demonstrate the DRAW command.



```

5 PLOT 0,95: DRAW 255,0
10 FOR a=0 TO 255
20 PLOT a,120+40*SIN (a/16*PI)
DRAW 0,-50
30 NEXT a

```

```

23778 CD BA 24      call 9402
Restore H'L'
23781 D9            exx
23782 E1            pop hl
23783 D9            exx
23784 AF            xor a

```

```

org 23760
Plot 0,95
23760 06 5F        ld b,95
23762 0E 00        ld c,0
23764 CD E5 22     call 8933
Store H'L'
23767 D9          exx
23768 E5          push hl
23769 D9          exx
Draw 255,0
23770 06 00        ld b,0
23772 0E FF        ld c,255
23774 16 01        ld d,1
23776 1E 01        ld e,1

```

```

L1
23785 F5          push af
23786 CD 28 2D     call 11560
23789 3E 78        ld a,120
23791 CD 28 2D     call 11560
23794 3E 28        ld a,40
23796 CD 28 2D     call 11560
23799 F1          pop af
23800 F5          push af
23801 CD 28 2D     call 11560
23804 3E 10        ld a,16
23806 CD 28 2D     call 11560
23809 EF          rst 40
defb 5 163 49 15 4 31 4 15 56

```

23819 CD DC 22	call 8924	23831 1E 01	ld e,1
Store H'L'		23833 CD BA 24	call 9402
23822 D9	exx	Restore H'L'	
23823 E5	push hl	23836 D9	exx
23824 D9	exx	23837 E1	pop hl
DRAW 0,-50		23838 D9	exx
23825 06 32	ld b,50	23839 F1	pop af
		23840 3C	inc a
		23841 FE 00	cp 0
23827 0E 00	ld c,0	23843 20 C4	jr nz,L1
23829 16 FF	ld d,255	23845 C9	ret

Program 2.8

Note that before DRAW is called the HL prime register must be saved (in this instance by pushing it onto the stack) because it is used in the DRAW routine.

The value of the HL prime register must *not* be corrupted during a user-written machine code routine or the program will crash on returning to BASIC. The HL prime register value is restored after the DRAW command. DRAW OVER and INVERSE are set in exactly the same way as PLOT OVER/INVERSE.

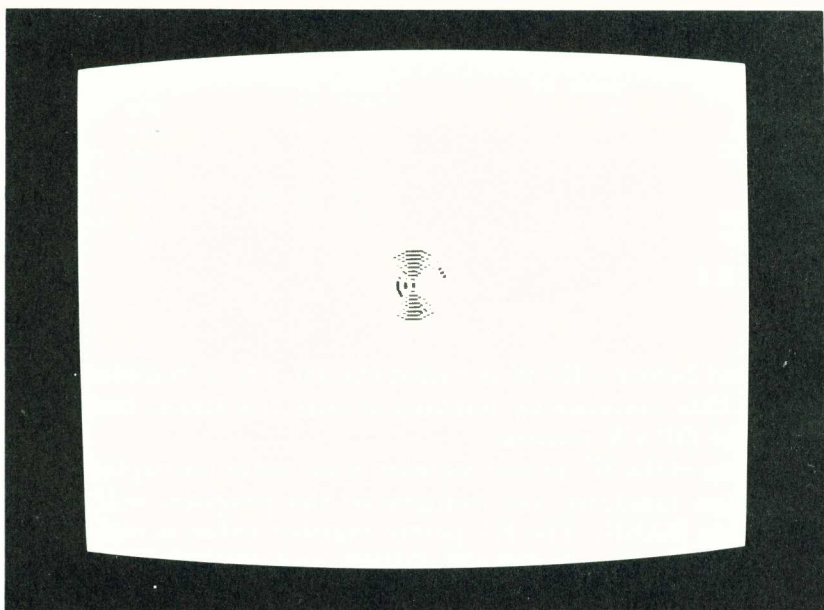
The second entry point for DRAW x,y requires the top two values on the calculator stack to have values x and y, again y being the top value – CALL 9335. Should you wish to use this second entry point then you will need to read Chapter 7 to acquaint yourself with the Spectrum method of storing five-byte *negative* numbers on the calculator stack.

DRAW x,y,a

The entry point for this BASIC command is 9108 and requires the value of x, y, and a to be placed on the calculator stack (in that order, that is, 'a' on top). Again Chapter 7 will explain how negative numbers are stacked. Again the HL prime register must not be corrupted.

CIRCLE x,y,r

The entry point for this BASIC command is 9005 and requires the values x, y, and r to be the top values on the calculator stack. The HL prime register is saved during the CIRCLE call. Program 2.9 demonstrates a method of drawing concentric circles to give a ripple effect.



```

10 REM Circle x,y,r
20 FOR B=1 TO 2
30 FOR A=1 TO 21 STEP 2
40 CIRCLE OVER 1;128,88,A
50 NEXT A
60 NEXT B

org 23760
For B=1 TO 2

23760 06 02      ld b,2
L2
23762 05          push bc

For A=1 TO 21

23763 3E 01      ld a,1
L1
23765 F5          push af

Set OVER 1

23766 FD 36 57 03 ld (iy+87),3

Stack DATA

23770 3E 80      ld a,128

23772 CD 28 2D   call 11560
23775 3E 58      ld a,88
23777 CD 28 2D   call 11560

23780 F1          pop af
23781 F5          push af
23782 CD 28 2D   call 11560
Store H'L' (IMPORTANT !)

23785 D9          exx
23786 E5          push hl
23787 D9          exx

Call CIRCLE

23788 CD 2D 23   call 9005

Restore H'L'

23791 D9          exx
23792 E1          pop hl
23793 D9          exx

23794 F1          pop af
STEP 2
23795 3C          inc a
23796 3C          inc a
23797 FE 17      cp 23
23799 20 DC      jr nz,L1
23801 C1          pop bc
23802 10 D6      djnz L2

Reset OVER 0

23804 FD 36 57 00 ld (iy+87),0
23808 C9          ret

```

Program 2.9

You will notice that the program is almost as slow as the BASIC. This is because the ROM CIRCLE routine is quite lengthy. If a circle is required in your machine code program it would be quicker to hold the plot points as DATA and to use a loop to plot each pixel.

3 COUNTING

Most games programs and many 'serious' ones require some form of counting. With games programs this is mainly used to keep an account of the score, lives lost, time taken, etc. I know three methods of counting and displaying the result on the screen. The first is so memory consuming and unsightly that only a brief outline of the method will be given. This method involves the setting up of the MAX number allowed as DATA. For instance if you were counting from 0 to 999999 then six bytes of DATA would be reserved and initialized to 000000. A print string routine would then be required to print all six zeros on the screen. When the counter is increased by one it would then be necessary to start at the units byte, check that it is not nine, add the one to the units and then print string. If the units were nine they would be reset to zero and the tens column checked for a nine, and so on. After printing the string a check must be made to see whether all six bytes are set at nine and if so the count ends. Counting up or down in steps greater than one would involve placing the 'increase by one' counter in a loop.

The second method is used for counting up or down within the range 0 to 65535 and is demonstrated in Program 3.1.

```
org 23760
Set counter to zero

23760 01 00 00    ld bc,0
L1
23763 ED 43 B0 5C ld (23728),bc

Stack value
23767 CD 2B 2D    call 11563

Open channel 2
23770 3E 02      ld a,2
23772 CD 01 16   call 5633

Set PRINT AT 11,16
23775 3E 16      ld a,22
23777 D7         rst 16

23778 3E 0B      ld a,11
23780 D7         rst 16

23781 3E 10      ld a,16
23783 D7         rst 16

Call PRINT VALUE ON STACK
23784 CD E3 2D   call 11747

Get value and add 1
23787 ED 4B B0 5C ld bc,(23728)
23791 03         inc bc

Return if zero
23792 78         ld a,b
23793 B1         or c
23794 C8         ret z

Check for key press
23795 FD CB 01 6E bit 5,(iy+1)

23799 2B DA      jr z,L1
23801 C9         ret
```

Program 3.1

In this program use is made of a ROM routine that prints the value on top of the calculator stack: call 11747. The spare bytes in the systems variables 23728/9 are used to hold the value of the counter; hence the range 0 to 65535. All we need to do is then fetch the current value and add one, put the new value back into the COUNTER, STACK the value, set the PRINT AT parameters and call PRINT VALUE ON STACK. If the value in BC becomes zero (that is, 65536) a return to BASIC is made.

I have also included an EXIT routine from address 23795 to 23800 so that a key press will also return to BASIC. This is advisable as we would otherwise be stuck in the LOOP until the counter had reached 65535 and that would take several minutes.

The third form of counting again involves the use of the calculator. Up to now we have regarded the calculator stack as just a method of storing values without going into too much detail about how they are stored. The values are actually stored on the stack in their five-byte form, that is, it takes five bytes of information to define any number that the computer can cope with. This is discussed in more detail in Chapter 7. In Program 3.2 we make use of this fact to count from 0 upwards in steps of 250 with no upper limit. In fact, however, once 99 999 999 has been reached the PRINT STACK VALUE routine will change to printing the value in its E form (1E + 9 etc.), but one hundred million is probably high enough for any games scoring.

```

org 23760                               Set PRINT AT 11,16
Use CALCULATOR                          23776 3E 16          ld a,22
                                           23778 D7            rst 16
23760 EF          rst 40                  23779 3E 0B        ld a,11
                                           23781 D7            rst 16
LITERAL STACK 0                          23782 3E 10        ld a,16
defb 160                                  23784 D7            rst 16

End use CALCULATOR                       Call PRINT VALUE ON STACK
defb 56                                   23785 CD E3 2D     call 11747

HL set to STACKEND-5                      STACK 250
after defb 56 instruction                  23788 3E FA        ld a,250

L1                                         23790 CD 28 2D     call 11560
Copy value into PRINTER BUFFER

23763 11 00 5B          ld de,23296
23766 61 05 00          ld bc,5
23769 ED B0            ldir

Open channel 2                             STACK VALUE
                                           23793 21 00 5B     ld hl,23296
                                           23796 ED 5B 65 5C  ld de,(23653)
                                           23800 01 05 00     ld bc,5
                                           23803 ED B0        ldir
                                           23805 ED 53 65 5C  ld (23653),de

23771 3E 02            ld a,2
23773 CD 01 16         call 5633
Use CALCULATOR

```

23809 EF	rst 40	23816 28 C9	jr z,L1
LITERAL ADD		Remove last value from STACK	
defb 15		23818 2A 65 5C	ld hl,(23653)
End use CALCULATOR		23821 2B	dec hl
defb 56		23822 2B	dec hl
		23823 2B	dec hl
Check for key press		23824 2B	dec hl
		23825 2B	dec hl
		23826 22 65 5C	ld (23653),hl
23812 FD CB 01 6E bit 5,(iy+1)		23827 C9	ret

Program 3.2

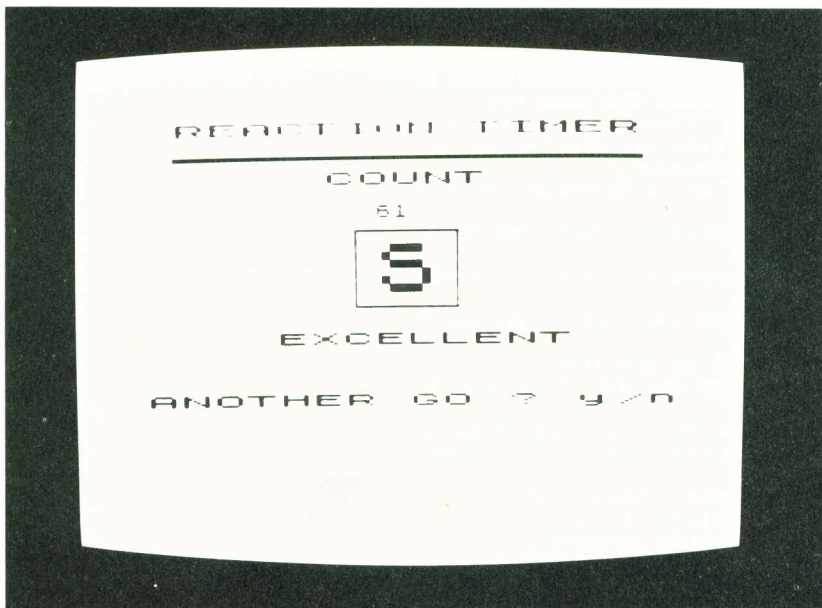
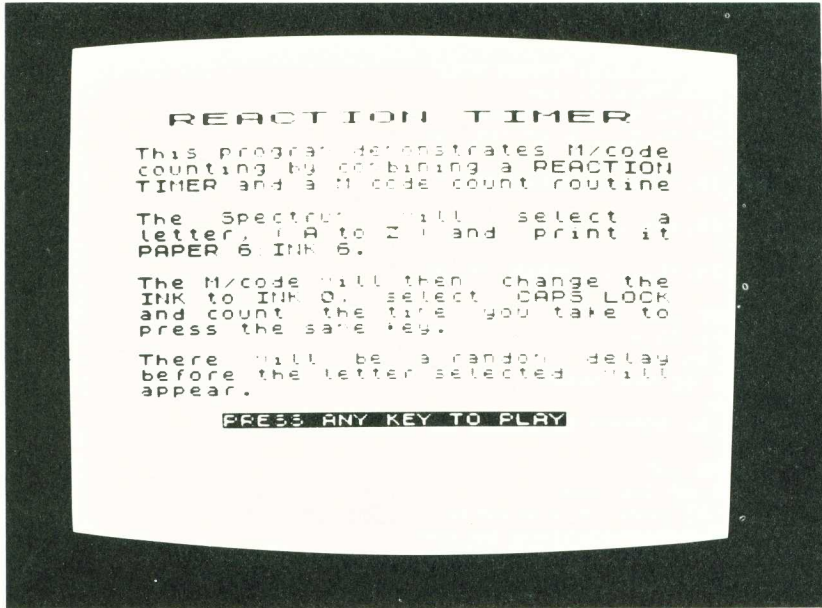
The program has several important routines that require explaining. The setting up of the value 0 is carried out on the calculator stack by defb 160, which is the literal STACK 0, the five bytes on top of the stack are then taken and copied into the printer buffer to be used as a store. This is carried out by an LDIR instruction; the value of HL is set for us by the calculator routine as (STACKEND - 5). The initial value is printed at 11, 16; by the PRINT VALUE routine and the stack is thus cleared.

And so to the ADD 250 routine. This requires the value of 250 to be placed on the stack by means of STACK VAL A, and then the 'stored' value placed on the stack. This is carried out using LDIR. The HL register is set to the start of the stored value; the DE register is set to STACKEND. After the LDIR instruction the new address of STACKEND must be put back into the systems variable 23653. The calculator is used to add these two values together and to repeat the STORE and PRINT VALUE routines. Again a key press routine is added to allow exit from the counting, but note that the stack is reset by removing the last value before a return to BASIC is made.

Counting down would require the start value to be placed on the stack (See Chapter 7 for numbers greater than 65535) and the Literal 3 SUBTRACT used. Note that in subtracting, the 'top' value is taken away from the one underneath. At the end of each count a check would be required to see whether the 'store' bytes had all been set to zero and, if so, the count ended (or if the second byte has bit 7 set the number is negative). In counting down it is also necessary to clear the screen of the existing number before printing the new one because, if a change is made from say 1000 to 999, the 'units zero' from 1000 would remain on the screen and the number would appear to be 9990.

The COUNTDEMO program, available on the tape, shows how the counter routine can be used for reaction timing.

The BASIC program is self-explanatory and is mainly involved with the printing of instructions. The DATA statements are used to hold parameters of x, y, h, w, and a\$ for LARGE printing as discussed in Chapter 2.




```

12>PAPER 6:CLS
15 DATA 87,143,1,2,"COUNT"
20 DATA 95,63,1,2,"SLOW"
30 DATA 71,63,1,2,"AVERAGE"
40 DATA 95,63,1,2,"GOOD"
50 DATA 63,63,1,2,"VERY GOOD"
60 DATA 63,63,1,2,"EXCELLENT"
80 DATA 47,63,1,2,"NOT TRYING"
90 DATA 0,31,1,2,"ANOTHER GO ?
y/n"
95 DATA 55,125,5,2,"THANK YOU"
96 DATA 103,79,3,2,"for"
97 DATA 71,50,5,2,"PLAYING"
100 DATA 24,79,5,2,"STOP THE TAP
E"
105 DATA 15,167,1,2,"REACTION TI
MER"
110 DATA 15,164,2,2,"-----
---"
120 DATA 24,15,1,2,"PRESS ANY KE
Y"
125 RESTORE 100
130 FOR a=1 TO 4
140 PAUSE 25
160 INK 2: GO SUB 9000
170 NEXT a
180 PAUSE 0: BORDER 5: PAPER 6:
INK 0: CLS
190 RESTORE 105
220 INK 0: GO SUB 9000
250 PRINT AT 3,0:"This program d
emonstrates M/codecounting by com
bining a REACTIONTIMER and a M/co
de count routine"
260 PRINT "The Spectrum will
select aletter, ( A to Z ) an
d print itPAPER 6:INK 6."
270 PRINT "The M/code will then
change theINK to INK 0, select
CAPS LOCKand count the time y
ou take topress the same key."
280 PRINT "There will be a r
andom delaybefore the letter sel
ected willappear."
290 PRINT AT 20,5:"PRESS ANY KEY
TO PLAY"
300 IF INKEY$<>" THEN GO TO 30
0
302 IF INKEY$="" THEN GO TO 302
305 CLS
310 RESTORE 105: INK 1: GO SUB 9
000
320 INK 5: GO SUB 9000
330 RESTORE 15: INK 2: GO SUB 90
00
350 INK 0: PLOT 102,114: DRAW 50
,0: DRAW 0,-38: DRAW -50,0: DRAW
0,38
380 LET z=INT (RND*26+65)
390 LET x=111: LET y=111: LET h=
4: LET w=4: LET a$=CHR$ z
400 INK 6: LET c=USR 32393
410 FOR a=1 TO RND*200+50: NEXT
a

```

```

420 INK 0
430 LET c=USR 23760
440 LET a=PEEK 23728+256*PEEK 23
729
450 IF a>=140 OR a=0 THEN RESTO
RE 80
460 IF a<140 THEN RESTORE 20
470 IF a<120 THEN RESTORE 30
480 IF a<100 THEN RESTORE 40
490 IF a<85 THEN RESTORE 50
500 IF a<70 THEN RESTORE 60
510 GO SUB 9000
7000 RESTORE 90: INK 1: GO SUB 90
00
7010 IF INKEY$<>" " THEN GO TO 70
10
7020 IF INKEY$="" THEN GO TO 702
0
7030 LET d$=INKEY$
7040 IF d$<>"y" THEN GO TO 8000
7050 LET c=USR 23842: GO TO 350
8000 RESTORE 95
8005 POKE 23843,20
8010 LET c=USR 23842
8015 POKE 23843,18
8030 FOR a=1 TO 3
8040 GO SUB 9000
8050 NEXT a
8060 STOP
9000 READ x,y,h,w,a$
9010 LET c=USR 32393
9020 RETURN
9800 CLEAR 32334: LOAD ""CODE : G
0 TO 5
9900 SAVE "REACTION" LINE 9800
9950 SAVE "LARGE"CODE 32335,265
10 REM >z? \ STEP }~ CLEAR THE
N 0 OVER !?X~ RETURN 6 ?60E! RETU
RN 1 NEXT ??? OR STEP +->? STEP
? BEEP >
BEEP >?xLEN (? CLEAR THEN ?n( 0
UT 1) \PEEK VERIFY GO SUB CVAL
\ CLEAR THEN 0o<>

```

org 23760

Find variable z

```

23760 3E 7A      1d a,122
Store in 23681
23762 01 81 5C  1d bc,23681
23765 CD 7D 7E  call 32381

```

Set CAPS LOCK

```
23768 FD CB 30 DE set 3,(iy+48)
```

Attribute colour swap

```

23772 21 00 58  1d h1,22528
L1
23775 7E        1d a,(h1)
23776 FE 36     cp 54
PAPER 6 INK 6
23778 20 02     jr nz,L2
23780 36 30     1d (h1),48

```

```

Replace PAPER & INK 0
L2
23782 23          inc hl
23783 7C          ld a,h
23784 FE 5B      cp 91
End of ATTRIBUTES ?
23786 20 F3      jr nz,L1

Start counting

23788 01 00 00   ld bc,0
Store count on stack
L3
23791 05          push bc
Place count on CALCULATOR
23792 CD 2B 2D   call 11563

Set PRINT AT 6,14
23795 3E 02      ld a,2
23797 CD 01 16   call 5633
23800 3E 16      ld a,22

23802 D7          rst 16
23803 3E 06      ld a,6
23805 D7          rst 16
23806 3E 0E      ld a,14
23808 D7          rst 16

Print count
23809 CD E3 2D   call 11747

Get count and add 1
23812 C1          pop bc
23813 03          inc bc

Check count < 65535
23814 78          ld a,b
23815 B1          or c
23816 2B 0F      jr z,END

Check key pressed

23818 FD CB 01 6E bit 5, (iy+1)

23822 28 DF      jr z,L3
Find INKEY#
23824 3A 0B 5C   ld a,(23560)
Compare a#
23827 21 81 5C   ld hl,23681
23830 BE          cp (hl)
23831 20 D6      jr nz,L3
END
Store count
23833 ED 43 B0 5C ld (23728),bc
Restore lower case
23837 FD CB 30 9E res 3,(iy+48)
23841 C9          ret
Clear bottom 18 lines
23842 06 12      ld b,18
23844 CD 44 0E   call 3652
23847 C9          ret

```

Program 3.3

4 RANDOM NUMBERS

Random numbers are an essential part of most games programs such as the random delay and random choice of letter in the REACTION TIMER program (Program 3.3) of Chapter 3. The setting up of a random number in machine code is actually very easy and, depending on the 'random/range' of the number, can be tackled in one of two ways.

The first method, which does not actually produce a true random number, but is good enough for many games, is to use the FRAMES and SEED variables in a way in which they were not intended to be used. That is, if the random number range is 0-1, 0-3, 0-7, 0-15, 0-31, 0-63, 0-127, or 0-255 then this method is worth considering. The routine requires the HL register to be loaded with the value of SEED (23670) the DE register to be loaded with SEED high byte/FRAMES low byte (23671), the two register values added together, and the sum stored back into the SEED variable for the next 'random number'. The A register is then loaded with either H or L and an 'AND a' carried out to mask out the unwanted bits; hence the random number range as above. Program 4.1 for instance gives an example of INT(RND * 32) as it will mask out bits 7-5 of the value in the A register and so leave numbers from 0 to 31.

```
org 23760
10 Random numbers
23760 06 0A      ld b,10
L1
23762 C5        push bc
Load HL, (SEED)
23763 2A 76 5C  ld hl,(23670)
23766 ED 5B 77 5C ld de,(23671)
23770 19        add hl,de
Get random number and put
on CALCULATOR STACK
23791 F1        pop af
23792 CD 28 2D  call 11560
23795 CD E3 2D  call 11747
Print it
Get number count
23798 C1        pop bc
Store Random number
23777 F5        push af
Set PRINT AT b,10
23778 3E 02     ld a,2
repeat until 10 numbers
23799 10 D9     djnz L1
23801 C9        ret
```

Program 4.1

PROGRAM 1 SAMPLE PRINT OUT

6
22
23
8
9
25
24
5
1
10

As previously stated this is useful if a 'not so random number' is required in one of the above ranges. It can be taken a stage further to evaluate, for example, $\text{INT}(\text{RND} * 23)$ by:

```
LD A, H
AND 15
LD H, A
LD A, L
AND 7
ADD A, H
```

You will notice in Program 4.1 that the B register is used to hold the loop count (ten times) and that this value is also used for the PRINT AT b, 10; routine. The sample print out shows the randomness of the routine.

To produce a random number in the same way as the BASIC command $\text{INT}(\text{RND} * n)$ we will need to use the RND ROM routine, but unfortunately this is not available with a call RND instruction as the routine has no return after it has been evaluated. The routine occupies ROM address 9725 to 9765 and uses the calculator, yet again, to modify the value of SEED. It starts by getting the existing value from SEED, and ends with the new value duplicated on top of the stack. The top value is removed and used to update SEED. The remaining value is modified further to give a value from 0 to 1 (but not 1) which is used for RND. To use this routine it is necessary to copy it into RAM and call it from your program as a subroutine. Program 4.2 shows this method used to produce $\text{INT}(\text{RND} * 12345)$ and print 44 answers using CHR\$ CODE 6 to print in two columns.

INT (RND*12345)		44 Random numbers
		23768 06 2C 1d b,44
		L1
org 23760		23770 C5 push bc
CLS		
		COPY of ROM 9725 to 9765
23760 3E 02	1d a,2	
23762 CD 01 16	call 5633	23771 ED 4B 76 5C 1d bc,(23670)
23765 CD 6B 0D	call 3435	23775 CD 2B 2D call 11563

```

23778 EF          rst 40
STACK (1)
defb 161

+
defb 15
STACK (75)
defb 52 55 22
*
defb 4
STACK (65537)
defb 52 128 65 0 0 128
LITERALS
defb 50 2
STACK (1)
defb 161
-
defb 3
Duplicate
defb 49
End CALC ROUTINE
defb 56
Put top value into BC
REGISTER (INT 0-65535)
23797 CD A2 2D    call 11682

```

```

Store new SEED
23800 ED 43 76 5C 1d (23670),bc
Manipulate EXPONENT BYTE

```

Program 4.2

SAMPLE PRINT OUT OF ABOVE PROG.

```

1857      3558
7666      7091
1037      3758
10296     6834
6415     12044
3058      7178
7727     11711
3057      7150
2940     10693

```

```

23804 7E          ld a,(hl)
23805 A7          and a
23806 28 03      jr z,L2
23808 D6 10      sub 16
23810 77          ld (hl),a
L2
STACK 12345
23811 01 39 30   ld bc,12345
23814 CD 2B 2D   call 11563
23817 EF          rst 40
*
defb 4
LITERAL 'INTEGER'
defb 39
End CALC ROUTINE
defb 56
Print value on STACK
23821 3E 02      ld a,2

23823 CD 01 16   call 5633
23826 CD E3 2D   call 11747
PRINT ,
23829 3E 06      ld a,6
23831 D7          rst 16

```

Get count

```

23832 C1          pop bc
23833 10 EF      djnz L1
23835 C9          ret

```

The RND routine uses several new literals which may be of use in other routines:

- LITERAL 161 is STACK NUMBER 1
- LITERAL 52 is STACK DATA (this is normally a number expressed in its compressed form – see Chapter 7)
- LITERAL 50 is $n - \text{mod } -m$
- LITERAL 2 is DELETE (the quotient)

The last two literals are probably of little use for games programs. The routine – CALL 11682 – is used by the ROM to remove the top value on the stack and to place the integer value in the BC register. The integer value is in this case rounded up or down as necessary to

the nearest whole number. There is also a similar routine – CALL 11733 which places the top value in the A register in a similar manner. The copied ROM routine ends at 23810.

The routine from 23811 to 23820 further modifies the RND value on the stack. The value 12345 is stacked and then multiplied by the RND value. A new literal is then used, literal 39; this removes any decimal value from the result of 12345 RND rounding *down* the value (as in BASIC RND), and so leaves INT (RND * 12345) as the top value on the stack. The PRINT VALUE routine is then used to print 44 values in two columns and the results show that these values are truly random.

Demonstration Program 4.3 shows how RND is actually used. In this case RND is used to print random-height skyscrapers in random characters of random colours and could be used in the 'city bomb' type of program. All the UDGs for a complete program are already set up and you may like to develop this program. The tape program 'RNDDEMO' uses the BASIC program to produce the first screen display; this is held on the screen for a few seconds, and then the machine code routine is used to print the next display demonstrating the speed of Machine Code as well as the RND function.

One point to remember when using BASIC RND with machine code USR calls is to use LET 'variable' = USR 'address' to call the machine code as RANDOMIZE USR 'address' will affect the random number generator. You may also use PRINT USR address if you don't mind the value held in the BC register being printed on return to BASIC. In the machine code routine the PAPER and INK colours are set up using LD (IY + 83), 47 instead of LD A, 47: LD (IY + 83) A, as in previous examples, thus saving memory and showing the usefulness of the IY register as a pointer. This is further demonstrated later in the program where 1Y is set to 23296, the start of the printer buffer, and used to LD (23296), 22 (AT) and LD (23299), 16 (INK). When altering the value held in IY, however, care must be taken that, if a ROM call is made that uses the IY register, the value should be reset to 23610 before the call is made or the ROM routine will be corrupted and may cause a crash.

Before the random number routine is called the A register is loaded with the RND * n number and this then stored at address 23681 (an unused address in the systems variables). This number is then picked up in the RND routine and used to obtain INT (RND * n), the value being placed in the A register by call 11733 before a return from the routine is made. The printer buffer is used to hold the PRINT information in order: AT (22), x, y, INK (16), b\$, a\$.

```

1 REM >? STEP 0" CLEAR 65/>?
STEP ?
10 BORDER 0: PAPER 5: INK 7: C
LS
20 FOR c=31 TO 0 STEP -1
30 LET l=19-INT (RND*10)
40 LET a%=CHR$ INT (RND*2+148)
45 LET b%=CHR$ INT (RND*4)
50 PRINT AT l-1,c;"▲"
60 FOR a=l TO 20
70 PRINT AT a,c;CHR$ 16+b%+a%
80 NEXT a
90 PRINT AT a,c;"■"
100 NEXT c
110 LET x=0: LET y=0: LET a$=""
120 PRINT AT x,y;a$
140 PRINT #0;" FUEL 10000 ■ BOMBS |
100 ■ SHOTS 100 ■"
150 RETURN
500 DATA 24,79,5,2,"STOP THE TA
PE"
510 DATA 15,167,1,2,"RANDOM NUM
BERS"
520 DATA 15,164,1,2,"_____
525 DATA 24,15,1,2,"PRESS ANY K
EY"
530 DATA 47,164,2,4,"BASIC"
540 DATA 31,164,3,2,"MACHINE CO
DE"
600 BORDER 4: PAPER 6: INK 0: C
LS
605 RESTORE 500
610 FOR a=1 TO 4
620 INK a: GO SUB 8000
630 PAUSE 50: NEXT a
640 PAUSE 0: CLS
645 PRINT INK 0;AT 8,0;"This p
rogram demonstrates theRND rou
tine in Machine code andcompa
s the speed of both BASICand Mac
hine code versions."
650 RESTORE 525
655 PAUSE 100
660 FLASH 1: INK 2: PAPER 7: GO
SUB 8000: FLASH 0
670 PAUSE 0: CLS
1000 LET z=1
1010 GO SUB 2000
1020 GO SUB 10
1025 LET z=z+1
1030 GO SUB 2000
1040 GO SUB 3000
1045 LET z=z+1
1050 GO TO 1010
2000 FOR a=1 TO 400: NEXT a: CLS
2010 PAPER 8: INK 0: RESTORE 530
+(10*(z/2=INT (z/2))) : GO SUB 80
00
2020 FOR a=1 TO 200: NEXT a: RET
URN
3000 LET c=USR 23760: GO TO 110
8000 READ x,y,h,w,a$
8010 LET c=USR 32393
8020 RETURN
9998 CLEAR 32334: LOAD "udg"CODE
USR "a": LOAD ""CODE : GO TO 50
0
9999 SAVE "RNDDEMO" LINE 9998: S
AVE "udg"CODE USR "a",168: SAVE
"LARGE"CODE 32335,265

```

org 23760

BORDER 0

23760 3E 00 ld a,0
23762 CD 9B 22 call 8859

PAPER 5 INK 7

23765 FD 36 53 2F ld (iy+83),47

CLS

23769 3E 02 ld a,2
23771 CD 01 16 call 5633
23774 CD 6B 0D call 3435

Set IY as POINTER

23777 FD 21 00 5B ld iy,23296

23781 FD 36 00 16 ld (iy+0),22
23785 FD 36 03 10 ld (iy+3),16

Reset IY

23789 FD 21 3A 5C ld iy,23610

Column Count

23793 3E 1F ld a,31
L1
23795 32 02 5B ld (23298),a

19-INT (RND*10)

23798 3E 0A ld a,10
23800 32 81 5C ld (23681),a
23803 CD 61 5D call RND
23806 6F ld l,a
23807 3E 13 ld a,19
23809 95 sub l
23810 32 01 5B ld (23297),a

INT (RND*2+148)

23813 3E 02 ld a,2
23815 32 81 5C ld (23681),a
23818 CD 61 5D call RND
23821 C6 94 add a,148
23823 32 05 5B ld (23301),a

INT (RND*4)

23826 3E 04 ld a,4
23828 32 81 5C ld (23681),a
23831 CD 61 5D call RND
23834 32 04 5B ld (23300),a

PRINT AT x-1,y CHR# 147

23837 3E 02 ld a,2
23839 CD 01 16 call 5633
23842 3E 16 ld a,22

```

23844 D7          rst 16
23845 3A 01 5B   ld a,(23297)
23848 3D          dec a
23849 D7          rst 16
23850 3A 02 5B   ld a,(23298)
23853 D7          rst 16
23854 3E 93      ld a,147
23856 D7          rst 16
L2

```

FOR a=x TO 20

```

PRINT AT a,y CHR$ 16+b$+a$
23857 11 00 5B   ld de,23296
23860 01 06 00   ld bc,6
23863 CD 3C 20   call 8252
23866 3A 01 5B   ld a,(23297)
23869 3C          inc a
23870 32 01 5B   ld (23297),a
23873 FE 15      cp 21
NEXT a
23875 20 EC      jr nz,L2

```

PRINT AT a,y CHR\$ 153

```

23877 3E 99      ld a,153
23879 32 05 5B   ld (23301),a
23882 3E 07      ld a,7
23884 32 04 5B   ld (23300),a
23887 11 00 5B   ld de,23296
23890 01 06 00   ld bc,6
23893 CD 3C 20   call 8252
23896 3A 02 5B   ld a,(23298)
23899 3D          dec a
23900 FE FF      cp 255

```

NEXT COLUMN

```

23902 20 93      jr nz,L1
23904 C9          ret

```

ROM RND ROUTINE

RND

```

23905 ED 4B 76 5C ld bc,(23670)
23909 CD 2B 2D    call 11563
23912 EF          rst 40
defb 161 15 52 55 22 4 52 128
defb 65 0 0 128 50 2 161 3 49 56
23931 CD A2 2D    call 11682
23934 ED 43 76 5C ld (23670),bc
23938 7E          ld a,(hl)
23939 A7          and a
23940 28 03      jr z,L3
23942 D6 10      sub 16
23944 77          ld (hl),a

```

INT (RND*PEEK (23681))

L3

```

23945 3A 81 5C   ld a,(23681)

```

```
23948 CD 28 2D    call 11560
23951 EF          rst 40
defb 4 39 56
```

VALUE to A

```
23955 CD D5 2D    call 11733
23958 C9          ret
```

Program 4.3



5 THE KEYBOARD

In machine code there are two methods of reading the keyboard. Both of them are useful but the method chosen depends on the type of program.

Using the LAST KEY variable

The Spectrum ROM contains a KEYSKAN and DECODE routine which is automatically called 50 times every second. The routine also updates the FRAMES count.

The character code of the last key pressed is stored in the variable LAST KEY (23560) and if it is a new key then bit 5, (IY + 1) will be set. Incidentally the HALT command waits for the next KEYSKAN call to be made before continuing with the machine code routine.

With the above information we can write a program the equivalent of PAUSE 0.

```
org 23760

WAIT FOR KEY PRESS

L1
23760 FD CB 01 6E bit 5, (iy+1)
23764 2B FA      jr z, L1
23766 FD CB 01 AE res 5, (iy+1)

CONTINUE WITH PROGRAM

23770 C9      ret
```

Program 5.1

You can see that in Program 5.1 bit 5, (IY + 1) is continually checked to see whether it is set and only when it is (indicating a new key press) will the program continue. Note that it is necessary to reset bit 5, (IY + 1) afterwards, otherwise if a similar PAUSE routine is used before the next KEYSKAN call bit 5 would still appear set and the PAUSE will be skipped.

Using IN A, (C)

This is identical to the BASIC IN command, reading the half row

specified by the value held in the BC register, and is discussed in the Spectrum manual. The value returned in A indicates which keys are pressed in that half row. Each half row contains five keys, and bits 4–0 of A hold the state of these keys (bits 7–5 are normally set) bit 0 is used to indicate the state of the ‘outside key’ in the half row and bit 4 the state of the ‘inside key’.

With no keys pressed the bits are set and A will hold 255. If a key is pressed its bit will be reset. For example if key 0 were pressed then bit 0 would be reset and A would hold value 254. You can see that with this method we have the ability to read more than one key being pressed at the same time by either checking the value of A or the individual bits of A. This would be useful in a game for two or more players or for moving things in two directions (for example upwards and forwards) at the same time. Program 5.2 demonstrates the IN command used to wait until key 0 is pressed.

```
org 23760

READ HALF ROW 6-0

L1
23760 01 FE EF      ld bc,61438
23763 ED 78         in a,(c)

CHECK KEY 0 PRESSED

23765 CB 47         bit 0,a
23767 C8            ret z
23768 18 F6         jr L1
```

Program 5.2

The ROM KEYSKAN call is made 50 times every second *even during our own machine code program* and, in doing so, slows down the program. We can stop this interruption to our program if we need extra speed, for example in pixel-scrolling, by the use of the DI instruction. However this will mean that while the interrupt is disabled we are unable to use the routine LAST KEY to read the keyboard (unless we call the ROM routine ourselves) and must resort to using the IN instruction. Program 5.3 shows the DI instruction being used.

```
org 23760                                CHECK KEY 0 PRESSED

DISSABLE                                23766 CB 47          bit 0,a
                                           23768 20 F7          jr nz,L1

23760 F3          di

ENABLE

READ HALF ROW 6-0

                                           23770 FB           ei

L1
23761 01 FE EF      ld bc,61438
23764 ED 78         in a,(c)                                23771 C9           ret
```

Program 5.3

Note that the EI instruction *must* be used before a return to BASIC or the keyboard will be useless. Try the program without the EI instruction and see what happens. No lasting damage will occur, but the Spectrum will have to be switched off then on again to make the keyboard function.

Program 5.4 shows the method of producing PAUSE n to give a timed delay or, as in the BASIC, continue if a key is pressed. The program uses the fact that the keyscans occur every 0.02 seconds and that HALT waits for a keyscan.

```

org 23760                                23768 20 05      jr nz,L2
                                           REDUCE COUNTER
SET 10 SECOND DELAY                    23770 0B          dec bc
23760 01 F4 01      ld bc,500           23771 7B          ld a,b
L1                                         23772 B1          or c
PAUSE LOOP                               23773 20 F4      jr nz,L1
23763 76             halt                L2
CHECK KEY PRESS                          23775 FD CB 01 AE res 5,(iy+1)

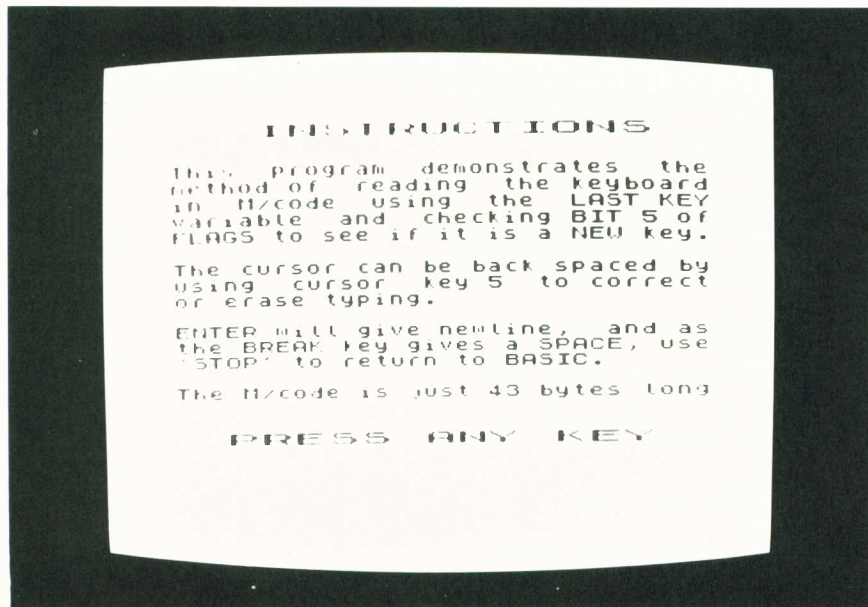
23764 FD CB 01 6E bit 5,(iy+1)          23779 C9          ret

```

Program 5.4

Program 5.5 is available on tape 'KEYDEMO' and makes use of the LAST KEY method of reading the keyboard to produce a simple typewriter program that allows erasure and modification of type.

The current print position is indicated by a flashing cursor. Note that CAPS SHIFT and '2' moves the print position to the next



PRINT COMMA TAB; CAPS SHIFT and '5' gives a backspace (but note the ROM error that you cannot backspace from the beginning of line 1 to the end of line 0) and ENTER gives a new line. In machine code the BREAK key is inoperable and so I have used 'STOP' to return to BASIC.

As with BASIC INKEY\$, neither the double-shifted keys nor the graphics mode keys are read. Should you wish to check for, or print, double-shifted keys then you will need to set up a DATA table of unshifted characters and their double-shifted equivalents. Then if LD A, (LAST KEY) returns value 14 in A, indicating both shift keys pressed, a routine to read the next key press and convert its unshifted character to its double-shifted one, by means of the DATA table, will be required.

```

10 REM >? STEP ?
20>DATA 24,79,5,2,"STOP THE TAP
E"
30 DATA 7,170,2,3,"TYPEWRITER"
40 DATA 7,150,2,3,"-----"
50 DATA 24,15,1,2,"PRESS ANY KE
Y"
60 DATA 39,170,1,2,"INSTRUCTION
S"
65 RESTORE
70 BORDER 4: PAPER 6: CLS
80 FOR A=1 TO 4
85 PAUSE 20
90 INK A: PAPER 8: GO SUB 8000
100 NEXT A
110 PAUSE 0: PAPER 7: CLS
120 INK 2: PAPER 8: GO SUB 8000
130 INK 0: PRINT AT 3,0;"This p
rogram demonstrates the method o
f reading the keyboard in M/cod
e using the LAST KEY variable
and checking BIT 5 of FLAGS to se
e if it is a NEW key."
140 PRINT "The cursor can be ba
ck spaced by using cursor key 5
to corrector erase typing."
150 PRINT "ENTER will give newl
ine, and as the BREAK key gives a
SPACE, use 'STOP' to return to BA
SIC."
160 PRINT "The M/code is just 4
3 bytes long"
200 RESTORE 50: INK 2: GO SUB 80
00
210 PAUSE 0: BRIGHT 1: CLS
220 INK 0: RANDOMIZE USR 23760
230 STOP
8000 READ x,y,h,w,a$
8010 RANDOMIZE USR 32393
8020 RETURN
9990 CLEAR 32334: LOAD ""CODE : G
O TO 20
9999 SAVE "KEYDEMO" LINE 9990: SA
VE "LARGE"CODE 32335,265

```

org 23760

Open channel 2

23760 3E 02 ld a,2
23762 CD 01 16 call 5633

SET OVER 1

L1
23765 FD 36 57 03 ld (iy+87),3

FLASH CURSOR IN WAIT LOOP

23769 3E 8F ld a,143
23771 D7 rst 16
23772 3E 08 ld a,8
23774 D7 rst 16
23775 3E 8F ld a,143
23777 D7 rst 16

23778 3E 08 ld a,8
23780 D7 rst 16

WAIT FOR KEY PRESS

23781 FD CB 01 6E bit 5,(iy+1)
23785 2B EA jr z,L1
23787 FD CB 01 AE res 5,(iy+1)

SET OVER 0

23791 FD 36 57 00 ld (iy+87),0

GET LAST KEY

23795 3A 08 5C ld a,(23560)

CHECK FOR 'STOP'

23798 FE E2 cp 226
23800 CB ret z

PRINT CHR#

23801 D7 rst 16
23802 1B D9 jr L1

Program 5.5

6 MOVEMENT

Now that we are able to print our display and read the keyboard we can look at methods of animation. This of course is an essential part of any 'real time' games program. In this Chapter we will cover methods of giving the impression of movement by:

1. Moving the foreground – ships, aliens, people, etc.
2. Moving the background – stars, planets, skyscrapers, etc.
3. Moving both 1 and 2.

The type of movement used by most software writers is 'character square'; this allows faster games and use of all colours. The screen is divided into 24 rows of 32 columns, and, when a character is moved, it travels from its present position to the adjacent row/column; this movement is fairly coarse but not jerky. If pixel movement is used then, especially with foreground movement, the characters require replotting. If you have used the BASIC PLOT command you will realize that this would normally restrict us to the use of two colours (PAPER and INK).

Foreground movement one character square

The first step is to define (or initialize) the character start position then, using either of the methods discussed in Chapter 5, to read the keyboard, and update the character position accordingly. The updating of the character position is usually carried out on a 'duplicate set' of start position data. When the updating is complete we will have two sets of data: the first holding the actual screen position and the second the new position. Any checks, for example, to see whether the new position is still on the screen, are carried out and the new data amended. To give the impression of flicker-free movement we remove the existing character and reprint it *immediately* in its new position. The method I generally use is to set OVER 1 and REPRINT the character in the *old* position, thus removing it, and, still using OVER 1 reprint in the *new* position. This will allow the character to be moved over background without erasing it.

org 23760
DISSABLE KEYBOARD

23760 F3 di
23761 18 18 jr START
UDG DATA

DATA1
CHR# 144
defb 0 0 15 24 49 226 225 224
CHR# 145
defb 24 60 255 255 153 255 231
defb 60
CHR# 146
defb 0 0 240 24 140 71 135 7
START

SET UP UDG's

23787 ED 5B 7B 5C ld de, (23675)

23791 21 D3 5C ld hl, DATA1
23794 01 18 00 ld bc, 24
23797 ED B0 ldir
23799 18 0A jr BEGIN

Print CHR# DATA
AT x,y INK 8 PAPER 8
CHR# 144 145 146

DATA2
defb 22 0 0 16 8 17 8
defb 144 145 146
BEGIN

BORDER 5
23811 FD 36 0E 28 ld (iy+14), 40

PAPER 1 INK 7
23815 FD 36 53 0F ld (iy+83), 15
CLS
23819 3E 02 ld a, 2

23821 CD 01 16 call 5633
23824 CD 6B 0D call 3435

PRINT STRING

23827 3E 02 ld a, 2
23829 CD 01 16 call 5633
23832 11 F9 5C ld de, DATA2
23835 01 0A 00 ld bc, 10
23838 CD 3C 20 call 8252
L7

TRANSFER DATA2 TO STORE

23841 11 00 5B ld de, 23296
23844 21 F9 5C ld hl, DATA2
23847 01 0A 00 ld bc, 10
23850 ED B0 ldir

GET x,y FROM STORE

23852 2A 01 5B ld hl, (23297)

CHECK KEY PRESSED

23855 01 FE EF ld bc, 61438
23858 ED 78 in a, (c)
23860 CB 47 bit 0, a
23862 20 01 jr nz, L1
23864 24 inc h
L1
23865 CB 4F bit 1, a
23867 20 01 jr nz, L2
23869 25 dec h
L2
23870 01 FE F7 ld bc, 63486
23873 ED 78 in a, (c)
23875 CB 47 bit 0, a
23877 20 01 jr nz, L3
23879 2D dec l
L3
23880 01 FE FB ld bc, 64510

23883 ED 78 in a, (c)
23885 CB 47 bit 0, a
23887 20 01 jr nz, L4
23889 2C inc l
L4

CHECK STILL ON SCREEN

23890 7D ld a, 1
23891 FE 16 cp 22
23893 20 03 jr nz, L5
23895 2D dec l
23896 18 05 jr L6
L5
23898 FE FF cp 255
23900 20 01 jr nz, L6
23902 2C inc l
L6
23903 7C ld a, h
23904 FE 1E cp 30
23906 20 03 jr nz, L8

23908 25 dec h
23909 18 05 jr L9
L8
23911 FE FF cp 255
23913 20 01 jr nz, L9
23915 24 inc h
L9

CHECK NO MOVE

23916 ED 5B 01 5B ld de, (23297)
23920 AF xor a
23921 E5 push hl
23922 ED 52 sbc hl, de
23924 E1 pop hl
IF NO MOVE THEN SKIP
PRINT OVER 1
23925 28 2C jr z, L12

STORE NEW x,y PARAMETERS

23927 22 01 5B	ld (23297),h1	DELAY LOOP	
SET OVER 1		23963 21 88 13	ld hl,5000
23930 FD 36 57 03	ld (iy+87),3	L11	
ERASE OLD POSITION		23966 2B	dec hl
23934 11 F9 5C	ld de,DATA2	23967 7C	ld a,h
23937 01 0A 00	ld bc,10	23968 B5	or 1
23940 CD 3C 20	call 8252	23969 20 FB	jr nz,L11
REPRINT IN NEW POSITION		CHECK 'SPACE' PRESSED	
23943 11 00 5B	ld de,23296	L12	
23946 01 0A 00	ld bc,10	23971 01 FE 7F	ld bc,32766
23949 CD 3C 20	call 8252	23974 ED 78	in a,(c)
UPDATE DATA2 FOR NEXT TIME		23976 CB 47	bit 0,a
23952 11 F9 5C	ld de,DATA2	23978 C2 21 5D	jp nz,L7
23955 21 00 5B	ld hl,23296	RESET OVER 0	
23958 01 0A 00	ld bc,10	23981 FD 36 57 00	ld (iy+87),0
23961 ED B0	ldir	MAKE KEY BOARD ACTIVE	
		23985 FB	ei
		23986 C9	ret

Program 6.1A

Program 6.1A demonstrates the principle as outlined above. A spaceship-like character is moved around the screen using keys 9/0 for left/right and 1/Q for up/down. 'Space' will return to BASIC. The spaceship is able to move in two directions at once and can fly around the screen without disturbing any background that you may care to set up. The character is printed INK 8; PAPER 8; and so will take on the colours of the squares as defined by the attribute of that square. The speed of movement is governed by the delay loop, and a check is made to see whether the ship is stationary. If it is, then the reprint routine is missed out as this would cause the ship to flash by being continually erased and reprinted at the same place. You can modify the program to enable the ship to fly off the screen and reappear on the other side by resetting 'y' if the ship exceeds the screen parameters.

Multiple character movement is basically the same as above with each character having its own set of PRINT AT DATA. The machine code routine is very fast as can be demonstrated by using LD HL, 1 for the delay loop, when the TV will not have the chance to print all the positions as the character moves. You will find that most programs need some sort of delay loop to hold the picture for a moment before moving on.

Background movement one character square

Whereas foreground movement requires the character position to

be held in memory and movements tracked, background movement usually requires the shifting of the entire screen display (or blocks of it) in a known direction without keeping track of exactly where everything is.

This movement is normally in one of four directions – up, down, left, and right – and usually involves both the characters and their attributes. If a screen background is shifted by one character position we have a further choice concerning the characters shifted off the screen. We can either lose them and print spaces in the row/column created on the other side of the screen, or we can store them with their attributes and reprint them in their corresponding place on the other side of the screen thus giving a wrap-around effect which is most useful in games programs.

The following Programs 6.1B to 6.16 demonstrate ROLL and SCROLL in all four directions; firstly the attributes and then the screen characters. These programs are easily modified to ROLL or SCROLL parts of the screen leaving the rest stationary. Note the peculiar layout of the screen display which is split up into three blocks of eight rows. Each row is further divided into eight lines of 32 bytes per line. The memory map starts at row 1 line 1 and 'maps' the 32 bytes of that line, then moves to row 2 line 1, row 3 line 1 and so on up to row 8 line 1 thus mapping 256 bytes. We then jump back to row 1 line 2, row 2 line 2, row 3 line 2, etc., and end up with row 8 line 8, thus mapping 2K bytes.

We then move to the middle eight rows which are dealt with in the same way, and finally to the last eight rows, again mapped in the same way. This does make the character rolling and scrolling a little complicated and requires moving 6K bytes.

```

org 23760
23760 11 00 58    1d de,22528
23763 21 20 58    1d hl,22560
23766 01 E0 02    1d bc,736
23769 ED B0       1dir
23771 06 20       1d b,32
23773 3A 8D 5C    1d a,(23693)
                L2
23776 12          1d (de),a
23777 13          inc de
23778 10 FC       djnz L2
23780 C9         ret

```

Program 6.1B

```

org 23760
23760 11 FF 5A    1d de,23295
23763 21 DF 5A    1d hl,23263
23766 01 E0 02    1d bc,736
23769 ED B8       1ddr
23771 06 20       1d b,32
23773 3A 8D 5C    1d a,(23693)
                L7
23776 12          1d (de),a
23777 13          inc de
23778 10 FC       djnz L7
23780 C9         ret

```

Program 6.2

org 23760		23769 01 1F 00	ld bc,31
23760 06 18	ld b,24	23772 ED B0	ldir
23762 21 00 58	ld hl,22528	23774 3A 8D 5C	ld a,(23693)
L12		23777 12	ld (de),a
23765 E5	push hl	23778 C1	pop bc
23766 D1	pop de	23779 10 F0	djnz L12
23767 C5	push bc	23781 C9	ret
23768 23	inc hl		

Program 6.3

org 23760		23769 01 1F 00	ld bc,31
23760 06 18	ld b,24	23772 ED B8	lddr
23762 21 FF 5A	ld hl,23295	23774 3A 8D 5C	ld a,(23693)
L14		23777 12	ld (de),a
23765 E5	push hl	23778 C1	pop bc
23766 D1	pop de	23779 10 F0	djnz L14
23767 C5	push bc	23781 C9	ret
23768 2B	dec hl		

Program 6.4

org 23760		23772 01 E0 02	ld bc,736
23760 21 00 58	ld hl,22528	23775 ED B0	ldir
23763 E5	push hl	23777 06 20	ld b,32
23764 D1	pop de	L17	
23765 06 20	ld b,32	23779 F1	pop af
L16		23780 2B	dec hl
23767 7E	ld a,(hl)	23781 77	ld (hl),a
23768 F5	push af	23782 10 FB	djnz L17
23769 23	inc hl	23784 C9	ret
23770 10 FB	djnz L16		

Program 6.5

org 23760		23772 01 E0 02	ld bc,736
23760 21 FF 5A	ld hl,23295	23775 ED B8	lddr
23763 E5	push hl	23777 06 20	ld b,32
23764 D1	pop de	L25	
23765 06 20	ld b,32	23779 F1	pop af
L24		23780 23	inc hl
23767 7E	ld a,(hl)	23781 77	ld (hl),a
23768 F5	push af	23782 10 FB	djnz L25
23769 2B	dec hl	23784 C9	ret
23770 10 FB	djnz L24		

Program 6.6

org 23760		23769 23	inc hl
23760 06 18	ld b,24	23770 01 1F 00	ld bc,31
23762 21 00 58	ld hl,22528	23773 ED B0	ldir
L32		23775 12	ld (de),a
23765 E5	push hl	23776 C1	pop bc
23766 D1	pop de	23777 10 F2	djnz L32
23767 C5	push bc	23779 C9	ret
23768 7E	ld a,(hl)		

Program 6.7

org 23760		23769 2B	dec hl
23760 06 18	ld b,24	23770 01 1F 00	ld bc,31
23762 21 FF 5A	ld hl,23295	23773 ED B8	lddr
L33		23775 12	ld (de),a
23765 E5	push hl	23776 C1	pop bc
23766 D1	pop de	23777 10 F2	djnz L33
23767 C5	push bc	23779 C9	ret
23768 7E	ld a,(hl)		

Program 6.8

org 23760		23769 23	inc hl
23760 06 C0	ld b,192	23770 01 1F 00	ld bc,31
23762 21 00 40	ld hl,16384	23773 ED B0	ldir
L1		23775 12	ld (de),a
23765 E5	push hl	23776 C1	pop bc
23766 D1	pop de	23777 10 F2	djnz L1
23767 C5	push bc	23779 C9	ret
23768 7E	ld a,(hl)		

Program 6.9

org 23760		23769 2B	dec hl
23760 06 C0	ld b,192	23770 01 1F 00	ld bc,31
23762 21 FF 57	ld hl,22527	23773 ED B8	lddr
L1		23775 12	ld (de),a
23765 E5	push hl	23776 C1	pop bc
23766 D1	pop de	23777 10 F2	djnz L1
23767 C5	push bc	23779 C9	ret
23768 7E	ld a,(hl)		

Program 6.10

org 23760		23769 23	inc hl
23760 21 00 40	ld hl,16384	23770 01 1F 00	ld bc,31
23763 AF	xor a	23773 ED B0	ldir
23764 06 C0	ld b,192	23775 12	ld (de),a
L13		23776 C1	pop bc
23766 C5	push bc	23777 10 F3	djnz L13
23767 E5	push hl	23779 C9	ret
23768 D1	pop de		

Program 6.11

org 23760		23773 21 01 40	ld hl,16385
L2		23776 11 00 40	ld de,16384
23760 21 00 40	ld hl,16384	23779 01 FF 17	ld bc,6143
23763 11 20 00	ld de,32	23782 ED 80	ldir
23766 06 C0	ld b,192	23784 EB	ex de,hl
L1		23785 36 00	ld (hl),0
23768 36 00	ld (hl),0		
23770 19	add hl,de	23787 C9	ret
23771 10 FB	djnz L1		

Program 6.11A

org 23760		23769 2B	dec hl
23760 21 FF 57	ld hl,22527	23770 01 1F 00	ld bc,31
23763 AF	xor a	23773 ED B8	lddr
23764 06 C0	ld b,192	23775 12	ld (de),a
L14		23776 C1	pop bc
23766 C5	push bc	23777 10 F3	djnz L14
23767 E5	push hl	23779 C9	ret
23768 D1	pop de		

Program 6.12

```

org 23760
23760 11 00 40   ld de,16384
23763 21 20 40   ld hl,16416
23766 01 E0 07   ld bc,2016
23769 ED B0      ldir
23771 06 08      ld b,b
23773 21 00 48   ld hl,18432
23776 11 E0 40   ld de,16608
L3
23779 C5         push bc
23780 E5         push hl
23781 D5         push de
23782 01 20 00   ld bc,32
23785 ED B0      ldir
23787 D1         pop de
23788 E1         pop hl
23789 C1         pop bc
23790 14         inc d
23791 24         inc h
23792 10 F1      djnz L3

23794 11 00 48   ld de,18432
23797 21 20 48   ld hl,18464
23800 01 E0 07   ld bc,2016
23803 ED B0      ldir
23805 06 08      ld b,b
23807 21 00 50   ld hl,20480
23810 11 E0 48   ld de,18656
L4
23813 C5         push bc

23814 E5         push hl
23815 D5         push de
23816 01 20 00   ld bc,32
23819 ED B0      ldir
23821 D1         pop de
23822 E1         pop hl
23823 C1         pop bc
23824 14         inc d
23825 24         inc h
23826 10 F1      djnz L4
23828 11 00 50   ld de,20480
23831 21 20 50   ld hl,20512

23834 01 E0 07   ld bc,2016
23837 ED B0      ldir
23839 06 08      ld b,b
23841 21 E0 50   ld hl,20704
L5
23844 C5         push bc
23845 E5         push hl
23846 06 20      ld b,32
L6
23848 36 00      ld (hl),0
23850 23         inc hl
23851 10 FB      djnz L6
23853 E1         pop hl
23854 C1         pop bc
23855 24         inc h
23856 10 F2      djnz L5
23858 C9         ret

```

Program 6.13

```

org 23760
23760 21 00 40   ld hl,16384
L18
23763 06 20      ld b,32
L19
23765 7E         ld a,(hl)
23766 F5         push af
23767 23         inc hl
23768 10 FB      djnz L19
23770 AF         xor a
23771 6F         ld l,a
23772 24         inc h
23773 7C         ld a,h
23774 FE 48      cp 72
23776 20 F1      jr nz,L18
23778 11 00 40   ld de,16384
23781 21 20 40   ld hl,16416
23784 01 E0 07   ld bc,2016
23787 ED B0      ldir
23789 06 08      ld b,b

23791 21 00 48   ld hl,18432
23794 11 E0 40   ld de,16608
L20
23797 C5         push bc
23798 E5         push hl
23799 D5         push de
23800 01 20 00   ld bc,32
23803 ED B0      ldir

23805 D1         pop de
23806 E1         pop hl
23807 C1         pop bc
23808 14         inc d
23809 24         inc h
23810 10 F1      djnz L20
23812 11 00 48   ld de,18432
23815 21 20 48   ld hl,18464
23818 01 E0 07   ld bc,2016
23821 ED B0      ldir
23823 06 08      ld b,b
23825 21 00 50   ld hl,20480
23828 11 E0 48   ld de,18656

L21
23831 C5         push bc
23832 E5         push hl
23833 D5         push de
23834 01 20 00   ld bc,32
23837 ED B0      ldir
23839 D1         pop de
23840 E1         pop hl
23841 C1         pop bc
23842 14         inc d
23843 24         inc h
23844 10 F1      djnz L21
23846 11 00 50   ld de,20480
23849 21 20 50   ld hl,20512
23852 01 E0 07   ld bc,2016

```

23855 ED B0	ldir	23864 2B	dec hl
23857 21 FF 57	ld hl,22527	23865 10 FB	djnz L23
L22		23867 3E FF	ld a,255
23860 06 20	ld b,32	23869 6F	ld l,a
L23		23870 25	dec h
23862 F1	pop af	23871 7C	ld a,h
		23872 FE 4F	cp 79
		23874 20 F0	jr nz,L22
		23876 C9	ret
23863 77	ld (hl),a		

Program 6.14

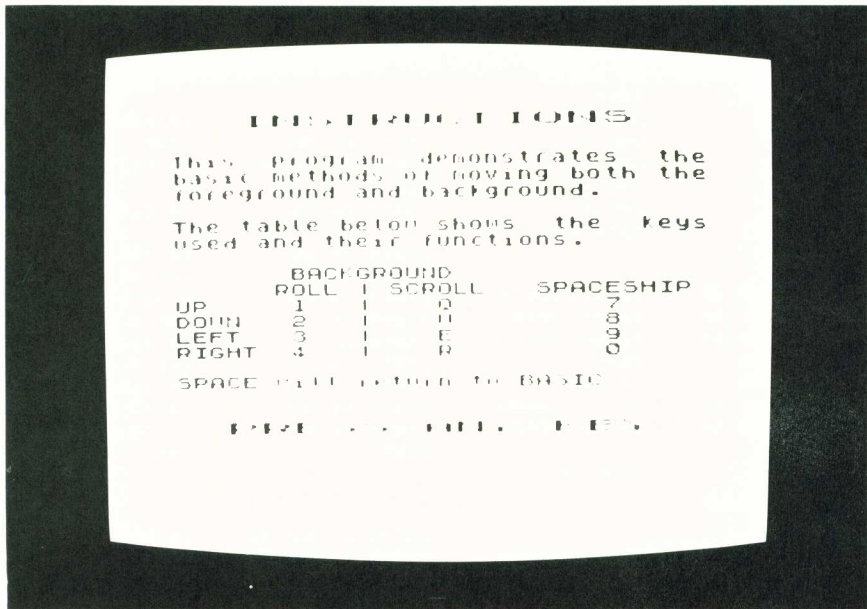
org 23760		23814 E5	push hl
23760 11 FF 57	ld de,22527	23815 D5	push de
23763 21 DF 57	ld hl,22495	23816 01 20 00	ld bc,32
23766 01 E0 07	ld bc,2016	23819 ED B0	ldir
23769 ED B8	lddr	23821 D1	pop de
23771 06 08	ld b,8	23822 E1	pop hl
23773 21 E0 48	ld hl,18656	23823 C1	pop bc
23776 11 00 50	ld de,20480	23824 14	inc d
L8		23825 24	inc h
23779 C5	push bc	23826 10 F1	djnz L9
23780 E5	push hl	23828 11 FF 47	ld de,18431
23781 D5	push de	23831 21 DF 47	ld hl,18399
23782 01 20 00	ld bc,32		
23785 ED B0	ldir		
23787 D1	pop de	23834 01 E0 07	ld bc,2016
23788 E1	pop hl	23837 ED B8	lddr
23789 C1	pop bc	23839 21 00 40	ld hl,16384
23790 14	inc d	L10	
23791 24	inc h	23842 06 20	ld b,32
23792 10 F1	djnz L8	L11	
		23844 36 00	ld (hl),0
		23846 23	inc hl
23794 11 FF 4F	ld de,20479	23847 10 FB	djnz L11
23797 21 DF 4F	ld hl,20447	23849 AF	xor a
23800 01 E0 07	ld bc,2016	23850 6F	ld l,a
23803 ED B8	lddr	23851 24	inc h
23805 06 08	ld b,8	23852 7C	ld a,h
23807 21 E0 40	ld hl,16608	23853 FE 48	cp 72
23810 11 00 48	ld de,18432	23855 20 F1	jr nz,L10
L9		23857 C9	ret
23813 C5	push bc		

Program 6.15

org 23760		23792 21 E0 48	ld hl,18656
23760 21 FF 57	ld hl,22527	23795 11 00 50	ld de,20480
L26		L28	
23763 06 20	ld b,32	23798 C5	push bc
L27		23799 E5	push hl
23765 7E	ld a,(hl)	23800 D5	push de
23766 F5	push af	23801 01 20 00	ld bc,32
23767 2B	dec hl	23804 ED B0	ldir
23768 10 FB	djnz L27	23806 D1	pop de
23770 3E FF	ld a,255	23807 E1	pop hl
23772 6F	ld l,a	23808 C1	pop bc
23773 25	dec h	23809 14	inc d
23774 7C	ld a,h	23810 24	inc h
23775 FE 4F	cp 79	23811 10 F1	djnz L28
23777 20 F0	jr nz,L26	23813 11 FF 4F	ld de,20479
23779 11 FF 57	ld de,22527	23816 21 DF 4F	ld hl,20447
23782 21 DF 57	ld hl,22495	23819 01 E0 07	ld bc,2016
23785 01 E0 07	ld bc,2016	23822 ED B8	lddr
23788 ED B8	lddr	23824 06 08	ld b,8
23790 06 08	ld b,8	23826 21 E0 40	ld hl,16608

23829 11 00 48	ld de,18432	23856 ED B8	lddr
		23858 21 00 40	ld hl,16384
L29		L30	
23832 C5	push bc	23861 06 20	ld b,32
23833 E5	push hl	L31	
23834 D5	push de	23863 F1	pop af
23835 01 20 00	ld bc,32		
23838 ED B0	ldir	23864 77	ld (hl),a
23840 D1	pop de	23865 23	inc hl
23841 E1	pop hl	23866 10 FB	djnz L31
23842 C1	pop bc	23868 AF	xor a
23843 14	inc d	23869 6F	ld l,a
23844 24	inc h	23870 24	inc h
23845 10 F1	djnz L29	23871 7C	ld a,h
23847 11 FF 47	ld de,18431	23872 FE 48	cp 72
23850 21 DF 47	ld hl,18399	23874 20 F1	jr nz,L30
23853 01 E0 07	ld bc,2016	23876 C9	ret

Program 6.16



```

1 REM >> STEP 1" CLEAR 65/>>
STEP 1
500>DATA 24,79,5,2,"STOP THE TAP
E"
510 DATA 0,167,6,4,"MOVEMENT"
520 DATA 0,130,2,4,"_____"
525 DATA 24,15,1,2,"PRESS ANY KE
Y"
550 DATA 31,170,1,2,"INSTRUCTION
S"
600 BORDER 4: PAPER 6: INK 0: CL
S
605 RESTORE 500
610 FOR a=1 TO 4
620 INK a: GO SUB 8000

```

```

630 PAUSE 50: NEXT a
640 PAUSE 0: CLS
642 INK 2: GO SUB 8000
645 PRINT INK 0;AT 3,0;"This p
rogram demonstrates thebasic me
thods of moving both theforegroun
d and background."
646 PRINT "The table below show
s the keysused and their functi
ons."
647 PRINT
650 PRINT "          BACKGROUND"
655 PRINT "          ROLL : SCROLL
SPACESHIP"
660 PRINT "UP      1   :   0
7"
670 PRINT "DOWN    2   :   W
8"
680 PRINT "LEFT    3   :   E
9"
690 PRINT "RIGHT   4   :   R
0"
695 PRINT "SPACE will return to
BASIC"
700 RESTORE 525: INK 2: GO SUB 8
000
705 IF INKEY$<>" THEN GO TO 70
5
710 IF INKEY$=" " THEN GO TO 710
715 POKE 30046,0: POKE 30047,0
720 RANDOMIZE USR 30001
725 CLS : RESTORE 730: PAPER 8:
INK 0: GO SUB 8000
730 DATA 0,80,3,1,"ENTER 'GOTO 7
15' for another go."
740 STOP
8000 READ x,y,h,w,a$
8010 LET c=USR 32393
8020 RETURN
9998 CLEAR 30000: LOAD "udg"CODE
USR "a": LOAD ""CODE : GO TO 500
9999 SAVE "MOVEDEMO" LINE 9998: S
AVE "udg"CODE USR "a",168: SAVE "
M/code"CODE 30001,2767

```

Program 6.17

org 40000 30001		BEGIN	
30001 F3	di	30055 11 5D 75	ld de,DATA2
30002 CD D0 5C	call 23760	30058 01 0A 00	ld bc,10
30005 18 18	jr START		
DATA1			
defb 0 0 15 24 49 226 225 224		30061 CD 3C 20	call 8252
defb 24 60 255 255 153 255 231		30064 11 00 5B	ld de,23296
defb 60 0 0 240 24 140 71 135 7		30067 21 5D 75	ld hl,DATA2
START		30070 01 0A 00	ld bc,10
30031 ED 5B 7B 5C	ld de,(23675)	30073 ED B0	ldir
30035 21 37 75	ld hl,DATA1		
30038 01 18 00	ld bc,24	MOVE	
30041 ED B0	ldir	30075 2A 01 5B	ld hl,(23297)
30043 18 0A	jr BEGIN	30078 01 FE EF	ld bc,61438
DATA2		30081 ED 78	in a,(c)
defb 22 0 0 16 8 17 8 144 145		30083 FE FF	cp 255
defb 146		30085 28 5D	jr z,LP12
		30087 CB 47	bit 0,a

30089	20	01		jr nz,LP1	30187	28	3E	jr z,SCR
30091	24			inc h	30189	FE	03	cp 3
LP1					30191	28	3A	jr z,SCR
30092	CB	4F		bit 1,a	30193	FE	0C	cp 12
30094	20	01		jr nz,LP2	30195	28	36	jr z,SCR
30096	25			dec h	30197	FE	0F	cp 15
LP2					30199	28	32	jr z,SCR
30097	CB	57		bit 2,a	30201	F5		push af
30099	20	01		jr nz,LP3	30202	FD	36 57 03	ld (iy+87),3
					30206	11	5D 75	ld de,DATA2
					30209	01	0A 00	ld bc,10
30101	2C			inc l	30212	CD	3C 20	call 8252
LP3					30215	F1		pop af
30102	CB	5F		bit 3,a	30216	CB	47	bit 0,a
30104	20	01		jr nz,LP4				
30106	2D			dec l	30218	F5		push af
LP4					30219	CC	09 77	call z,UROLL
30107	7D			ld a,l	30222	F1		pop af
30108	FE	16		cp 22	30223	CB	4F	bit 1,a
30110	20	03		jr nz,LP5	30225	F5		push af
30112	2D			dec l	30226	CC	56 78	call z,DROLL
30113	18	05		jr LP6	30229	F1		pop af
LP5					30230	CB	57	bit 2,a
30115	FE	FF		cp 255	30232	F5		push af
30117	20	01		jr nz,LP6	30233	CC	E3 78	call z,LROLL
30119	2C			inc l	30236	F1		pop af
LP6					30237	CB	5F	bit 3,a
30120	7C			ld a,h	30239	CC	F7 78	call z,RROLL
30121	FE	1E		cp 30	30242	11	5D 75	ld de,DATA2
30123	20	03		jr nz,LP7	30245	01	0A 00	ld bc,10
30125	25			dec h	30248	CD	3C 20	call 8252
30126	18	05		jr LP8	SCR			
					30251	01	FE FB	ld bc,64510
LP7					30254	ED	78	in a,(c)
30128	FE	FF		cp 255	30256	E6	0F	and 15
30130	20	01		jr nz,LP8	30258	28	3E	jr z,DELAY
30132	24			inc h				
LP8					30260	FE	03	cp 3
30133	ED	5B 01 5B		ld de,(23297)	30262	28	3A	jr z,DELAY
30137	AF			xor a	30264	FE	0C	cp 12
30138	E5			push hl	30266	28	36	jr z,DELAY
30139	ED	52		sbc hl,de	30268	FE	0F	cp 15
30141	E1			pop hl	30270	28	32	jr z,DELAY
30142	28	24		jr z,LP12	30272	F5		push af
30144	22	01 5B		ld (23297),hl	30273	FD	36 57 03	ld (iy+87),3
OVER1					30277	11	5D 75	ld de,DATA2
30147	FD	36 57 03		ld (iy+87),3	30280	01	0A 00	ld bc,10
30151	11	5D 75		ld de,DATA2	30283	CD	3C 20	call 8252
30154	01	0A 00		ld bc,10	30286	F1		pop af
30157	CD	3C 20		call 8252	30287	CB	47	bit 0,a
30160	11	00 5B		ld de,23296	30289	F5		push af
30163	01	0A 00		ld bc,10	30290	CC	8A 76	call z,USCR
30166	CD	3C 20		call 8252	30293	F1		pop af
30169	11	5D 75		ld de,DATA2	30294	CB	4F	bit 1,a
					30296	F5		push af
30172	21	00 5B		ld hl,23296	30297	CC	01 77	call z,DSCR
30175	01	0A 00		ld bc,10	30300	F1		pop af
30178	ED	B0		ldir	30301	CB	57	bit 2,a
LP12								
30180	01	FE F7		ld bc,63486	30303	F5		push af
30183	ED	78		in a,(c)	30304	CC	77 77	call z,LSCR
30185	E6	0F		and 15				

30307	F1	pop af	30421	D5	push de
30308	CB 5F	bit 3,a			
30310	CC A0 77	call z,RSCR			
30313	11 5D 75	ld de,DATA2	30422	01 20 00	ld bc,32
30316	01 0A 00	ld bc,10	30425	ED B0	ldir
30319	CD 3C 20	call 8252	30427	D1	pop de
DELAY			30428	E1	pop hl
30322	21 10 27	ld hl,10000	30429	C1	pop bc
L1			30430	14	inc d
30325	2B	dec hl	30431	24	inc h
30326	7C	ld a,h	30432	10 F1	djnz L4
30327	B5	or l	30434	11 00 50	ld de,20480
30328	20 FB	jr nz,L1	30437	21 20 50	ld hl,20512
30330	01 FE 7F	ld bc,32766	30440	01 E0 07	ld bc,2016
30333	ED 78	in a,(c)	30443	ED B0	ldir
30335	CB 47	bit 0,a	30445	06 08	ld b,8
30337	C2 7B 75	jp nz,MOVE	30447	21 E0 50	ld hl,20704
30340	FD 36 57 00	ld (iy+87),0	L5		
30344	FB	ei	30450	C5	push bc
			30451	E5	push hl
30345	C9	ret	30452	06 20	ld b,32
USCR			L6		
30346	11 00 58	ld de,22528	30454	36 00	ld (hl),0
30349	21 20 58	ld hl,22560	30456	23	inc hl
30352	01 E0 02	ld bc,736			
30355	ED B0	ldir	30457	10 FB	djnz L6
30357	06 20	ld b,32	30459	E1	pop hl
30359	3A 8D 5C	ld a,(23693)	30460	C1	pop bc
L2			30461	24	inc h
30362	12	ld (de),a	30462	10 F2	djnz L5
30363	13	inc de	30464	C9	ret
30364	10 FC	djnz L2	DSCR		
30366	11 00 40	ld de,16384	30465	11 FF 5A	ld de,23295
30369	21 20 40	ld hl,16416	30468	21 DF 5A	ld hl,23263
30372	01 E0 07	ld bc,2016	30471	01 E0 02	ld bc,736
30375	ED B0	ldir	30474	ED B8	lddr
30377	06 08	ld b,8	30476	06 20	ld b,32
30379	21 00 48	ld hl,18432	30478	3A 8D 5C	ld a,(23693)
30382	11 E0 40	ld de,16608	L7		
L3			30481	12	ld (de),a
30385	C5	push bc	30482	13	inc de
			30483	10 FC	djnz L7
30386	E5	push hl	30485	11 FF 57	ld de,22527
30387	D5	push de	30488	21 DF 57	ld hl,22495
30388	01 20 00	ld bc,32	30491	01 E0 07	ld bc,2016
30391	ED B0	ldir	30494	ED B8	lddr
30393	D1	pop de			
30394	E1	pop hl	30496	06 08	ld b,8
30395	C1	pop bc	30498	21 E0 48	ld hl,18656
30396	14	inc d	30501	11 00 50	ld de,20480
30397	24	inc h	L8		
30398	10 F1	djnz L3	30504	C5	push bc
30400	11 00 48	ld de,18432	30505	E5	push hl
30403	21 20 48	ld hl,18464	30506	D5	push de
30406	01 E0 07	ld bc,2016	30507	01 20 00	ld bc,32
30409	ED B0	ldir	30510	ED B0	ldir
30411	06 08	ld b,8	30512	D1	pop de
30413	21 00 50	ld hl,20480	30513	E1	pop hl
30416	11 E0 48	ld de,18656	30514	C1	pop bc
L4			30515	14	inc d
30419	C5	push bc	30516	24	inc h
30420	E5	push hl	30517	10 F1	djnz L8

30519	11 FF 4F	ld de,20479	30613	23	inc hl
30522	21 DF 4F	ld hl,20447	30614	01 1F 00	ld bc,31
30525	01 E0 07	ld bc,2016	30617	ED B0	ldir
30528	ED B8	lddr	30619	12	ld (de),a
30530	06 08	ld b,8	30620	C1	pop bc
30532	21 E0 40	ld hl,16608	30621	10 F3	djnz L13
			30623	C9	ret
			RSCR		
30535	11 00 48	ld de,18432	30624	06 18	ld b,24
L9			30626	21 FF 5A	ld hl,23295
30538	C5	push bc	L14		
30539	E5	push hl			
30540	D5	push de			
30541	01 20 00	ld bc,32	30629	E5	push hl
30544	ED B0	ldir	30630	D1	pop de
30546	D1	pop de	30631	C5	push bc
30547	E1	pop hl	30632	2B	dec hl
30548	C1	pop bc	30633	01 1F 00	ld bc,31
30549	14	inc d	30636	ED B8	lddr
30550	24	inc h	30638	3A 8D 5C	ld a,(23693)
30551	10 F1	djnz L9	30641	12	ld (de),a
30553	11 FF 47	ld de,18431	30642	C1	pop bc
30556	21 DF 47	ld hl,18399	30643	10 F0	djnz L14
30559	01 E0 07	ld bc,2016	30645	21 FF 57	ld hl,22527
30562	ED B8	lddr	30648	AF	xor a
30564	21 00 40	ld hl,16384	30649	06 C0	ld b,192
L10			L14		
30567	06 20	ld b,32	30651	C5	push bc
L11			30652	E5	push hl
			30653	D1	pop de
			30654	2B	dec hl
30569	36 00	ld (hl),0	30655	01 1F 00	ld bc,31
30571	23	inc hl	30658	ED B8	lddr
30572	10 FB	djnz L11	30660	12	ld (de),a
30574	AF	xor a			
30575	6F	ld l,a			
30576	24	inc h			
30577	7C	ld a,h	30661	C1	pop bc
30578	FE 48	cp 72	30662	10 DD	djnz L14
30580	20 F1	jr nz,L10	30664	C9	ret
30582	C9	ret	UROLL		
LSCR			30665	21 00 58	ld hl,22528
30583	06 18	ld b,24	30668	E5	push hl
30585	21 00 58	ld hl,22528	30669	D1	pop de
L12			30670	06 20	ld b,32
30588	E5	push hl	L16		
30589	D1	pop de	30672	7E	ld a,(hl)
30590	C5	push bc	30673	F5	push af
30591	23	inc hl	30674	23	inc hl
30592	01 1F 00	ld bc,31	30675	10 FB	djnz L16
30595	ED B0	ldir	30677	01 E0 02	ld bc,736
30597	3A 8D 5C	ld a,(23693)	30680	ED B0	ldir
			30682	06 20	ld b,32
			L17		
30600	12	ld (de),a	30684	F1	pop af
30601	C1	pop bc	30685	2B	dec hl
30602	10 F0	djnz L12	30686	77	ld (hl),a
30604	21 00 40	ld hl,16384	30687	10 FB	djnz L17
30607	AF	xor a			
30608	06 C0	ld b,192			
L13					
30610	C5	push bc	30689	21 00 40	ld hl,16384
30611	E5	push hl	L18		
30612	D1	pop de	30692	06 20	ld b,32
			L19		

30694	7E	ld a, (hl)	30794	10 FB	djnz L23
30695	F5	push af	30796	3E FF	ld a, 255
30696	23	inc hl	30798	6F	ld l, a
30697	10 FB	djnz L19	30799	25	dec h
30699	AF	xor a	30800	7C	ld a, h
30700	6F	ld l, a	30801	FE 4F	cp 79
30701	24	inc h	30803	20 F0	jr nz, L22
30702	7C	ld a, h	30805	C9	ret
30703	FE 4B	cp 72	DROLL		
30705	20 F1	jr nz, L18	30806	21 FF 5A	ld hl, 23295
30707	11 00 40	ld de, 16384	30809	E5	push hl
30710	21 20 40	ld hl, 16416	30810	D1	pop de
30713	01 E0 07	ld bc, 2016	30811	06 20	ld b, 32
30716	ED B0	ldir	L24		
30718	06 08	ld b, 8	30813	7E	ld a, (hl)
30720	21 00 48	ld hl, 18432	30814	F5	push af
30723	11 E0 40	ld de, 16608	30815	2B	dec hl
			30816	10 FB	djnz L24
			30818	01 E0 02	ld bc, 736
			30821	ED B8	lddr
			30823	06 20	ld b, 32
L20					
30726	C5	push bc			
30727	E5	push hl			
30728	D5	push de			
30729	01 20 00	ld bc, 32	L25		
30732	ED B0	ldir	30825	F1	pop af
30734	D1	pop de	30826	23	inc hl
30735	E1	pop hl	30827	77	ld (hl), a
30736	C1	pop bc	30828	10 FB	djnz L25
30737	14	inc d	30830	21 FF 57	ld hl, 22527
30738	24	inc h	L26		
30739	10 F1	djnz L20	30833	06 20	ld b, 32
30741	11 00 48	ld de, 18432	L27		
30744	21 20 48	ld hl, 18464	30835	7E	ld a, (hl)
30747	01 E0 07	ld bc, 2016	30836	F5	push af
30750	ED B0	ldir	30837	2B	dec hl
30752	06 08	ld b, 8	30838	10 FB	djnz L27
30754	21 00 50	ld hl, 20480	30840	3E FF	ld a, 255
30757	11 E0 48	ld de, 18656	30842	6F	ld l, a
L21			30843	25	dec h
30760	C5	push bc	30844	7C	ld a, h
			30845	FE 4F	cp 79
			30847	20 F0	jr nz, L26
			30849	11 FF 57	ld de, 22527
			30852	21 DF 57	ld hl, 22495
30761	E5	push hl			
30762	D5	push de			
30763	01 20 00	ld bc, 32			
30766	ED B0	ldir			
30768	D1	pop de	30855	01 E0 07	ld bc, 2016
30769	E1	pop hl	30858	ED B8	lddr
30770	C1	pop bc	30860	06 08	ld b, 8
30771	14	inc d	30862	21 E0 48	ld hl, 18656
30772	24	inc h	30865	11 00 50	ld de, 20480
30773	10 F1	djnz L21	L28		
30775	11 00 50	ld de, 20480	30868	C5	push bc
30778	21 20 50	ld hl, 20512	30869	E5	push hl
30781	01 E0 07	ld bc, 2016	30870	D5	push de
30784	ED B0	ldir	30871	01 20 00	ld bc, 32
30786	21 FF 57	ld hl, 22527	30874	ED B0	ldir
L22			30876	D1	pop de
30789	06 20	ld b, 32	30877	E1	pop hl
L23			30878	C1	pop bc
30791	F1	pop af	30879	14	inc d
30792	77	ld (hl), a	30880	24	inc h
30793	2B	dec hl	30881	10 F1	djnz L28

30883	11	FF 4F	ld de,20479	30941	7C	ld a,h
30886	21	DF 4F	ld hl,20447	30942	FE 48	cp 72
30889	01	E0 07	ld bc,2016	30944	20 F1	jr nz,L30
30892	ED	B8	lddr	30946	C9	ret
				LROLL		
30894	06	08	ld b,8	30947	06 D8	ld b,216
30896	21	E0 40	ld hl,16608	30949	21 00 40	ld hl,16384
30899	11	00 48	ld de,18432	L32		
L29				30952	E5	push hl
30902	C5		push bc	30953	D1	pop de
30903	E5		push hl	30954	C5	push bc
30904	D5		push de	30955	7E	ld a,(hl)
30905	01	20 00	ld bc,32			
30908	ED	B0	ldir	30956	23	inc hl
30910	D1		pop de	30957	01 1F 00	ld bc,31
30911	E1		pop hl	30960	ED B0	ldir
30912	C1		pop bc	30962	12	ld (de),a
30913	14		inc d	30963	C1	pop bc
30914	24		inc h	30964	10 F2	djnz L32
30915	10	F1	djnz L29	30966	C9	ret
30917	11	FF 47	ld de,18431	RROLL		
30920	21	DF 47	ld hl,18399	30967	06 D8	ld b,216
30923	01	E0 07	ld bc,2016	30969	21 FF 5A	ld hl,23295
30926	ED	B8	lddr	L33		
30928	21	00 40	ld hl,16384	30972	E5	push hl
L30				30973	D1	pop de
				30974	C5	push bc
30931	06	20	ld b,32	30975	7E	ld a,(hl)
L31				30976	2B	dec hl
30933	F1		pop af	30977	01 1F 00	ld bc,31
30934	77		ld (hl),a	30980	ED B8	lddr
30935	23		inc hl	30982	12	ld (de),a
30936	10	FB	djnz L31	30983	C1	pop bc
30938	AF		xor a	30984	10 F2	djnz L33
30939	6F		ld l,a			
30940	24		inc h	30986	C9	ret

Program 6.18

Note that in these routines extensive use is made of the block transfer instructions LDIR and LDDR and that after such instructions the DE and HL registers hold the finish address +1 (LDIR) or -1 (LDDR). Use is made of this fact to save reloading HL or DE with specific a address for further program routines. The ATTR. UP SCROLL (Program 6.1B) for example uses the fact that after LDIR the DE register holds the address of the 'START of LINE 24' attribute and so it is unnecessary to reload DE with this address. The existing line 24 attributes are replaced with the permanent screen attributes held in the systems variable ATTR P (23693).

To demonstrate each type of movement I have written Program 6.17 which is available on tape 'MOVEDEMO'. Since the program is almost 1000 bytes long it is not practical to hold it all in a REM statement (this would be more than a screenful) so I have written it for use above RAMTOP with start at address 30001 (Program 6.18). The machine code in the REM statement is taken from the program

on random numbers (Program 4.1) and is used to set up the background. The program is self-explanatory and calls the various routines depending on the keys pressed. The attributes are moved at the same time as the display characters. You can move the background and the ship diagonally and if two opposing keys are pressed then the routine is skipped. A return to BASIC is made by pressing 'SPACE'. Note that the vertical movement is rather jerky and not really suitable in this form owing to the transition from one screen zone to another. The programs, in fact, momentarily put line 1 into line 8 before scrolling line 9 into line 8.

The screen UP/DOWN – SCROLL and ROLL programs (Programs 6.13 to 6.16) were written in a form that should have been easy for you to follow with no subroutines to make them position dependent. To give a smoother movement these routines should be tackled in a different way, that is, scrolling one row at a time, rather than the top line of each row then the second line of each row etc. as at present. To show how this is achieved I have rewritten the SCREEN UP SCROLL program (Program 6.19A). As you can see it involves one subroutine and a sub-subroutine and is rather more complicated. Perhaps you would like to rewrite the SCREEN DOWN SCROLL routine in the same form as an exercise to make sure that you fully understand its workings.

```

org 23760
23760 21 00 40   ld hl,16384
23763 CD 01 5D   call SECT
23766 EB        ex de,hl
23767 21 00 48   ld hl,18432
23770 E5        push hl
23771 CD 15 5D   call UP
23774 E1        pop hl
23775 CD 01 5D   call SECT
23778 EB        ex de,hl
23779 21 00 50   ld hl,20480
23782 E5        push hl
23783 CD 15 5D   call UP
23786 E1        pop hl
23787 CD 01 5D   call SECT
23790 21 E0 50   ld hl,20704
L4
23793 E5        push hl
23794 06 20     ld b,32
L3

23796 36 00     ld (hl),0
23798 23       inc hl
23799 10 FB     djnz L3
23801 E1       pop hl
23802 24       inc h
23803 7C       ld a,h
23804 FE 58     cp 88

23806 20 F1     jr nz,L4
23808 C9       ret
SECT
23809 06 07     ld b,7
L1
23811 C5       push bc
23812 01 20 00 ld bc,32
23815 E5       push hl
23816 D1       pop de
23817 09       add hl,bc
23818 E5       push hl
23819 D5       push de
23820 CD 15 5D call UP
23823 D1       pop de

23824 E1       pop hl
23825 C1       pop bc
23826 10 EF     djnz L1
23828 C9       ret
UP
23829 06 08     ld b,8
L2
23831 C5       push bc
23832 E5       push hl
23833 D5       push de
23834 01 20 00 ld bc,32
23837 ED B0     ldir
23839 D1       pop de
23840 E1       pop hl

```



```

23841 C1          pop bc
23842 14          inc d
23843 24          inc h
23844 10 F1       djnz L2
23846 C9          ret

```

Program 6.19A

If you compare Programs 6.13 and 6.19A you will see that we have reduced the memory required from 99 bytes to 87 bytes but even this is not as compact as it could be. To show how we can reduce the memory and running time still further *and* give an extra useful feature we will consider the UP ROLL Program 6.14 which as it stands takes 117 bytes. At present the program handles each character row, line by line and has the 'fault' of temporarily transferring row 1 to row 8 and row 9 to row 17. We could rewrite it on the lines of Program 6.21 to overcome this problem but let us consider an entirely different approach that will give a program only 76 bytes long.

Instead of handling the screen one row at a time, we could write a program that rolls one column at a time. The routine would then be repeated for 32 columns. This would give a feature of being able to select which columns you require rolling either in blocks or singly, and what is more, would not require 256 bytes of memory to store the top row of the screen as Program 6.14 does but rather only eight bytes for the single top column character. Program 6.19B shows how this reduction in memory is achieved.

```

org 23760
CHAPTER 6
PROGRAM 19B

SCREEN UP ROLL 'n' COLUMNS
EXAMPLE

ROLL COLUMNS 5 TO 14 INC

23760 AF          xor a

LD DE,16383+START COLUMN

1st COLUMN = 1

23761 11 04 40    ld de,16388

LD B,No. of columns

23764 06 0A       ld b,10
L5
23766 D5          push de
23767 C5          push bc

STORE TOP LINE CHR ON STACK

23768 D5          push de
23769 E1          pop hl
23770 06 08       ld b,8
L6
23772 7E          ld a,(hl)
23773 F5          push af
23774 24          inc h
23775 10 FB       djnz L6

TRANSFER 7 LINES

L4
23777 06 07       ld b,7
L2

23779 C5          push bc
23780 21 20 00    ld hl,32
23783 19          add hl,de
23784 E5          push hl

TRANSFER 8 BYTES

23785 06 08       ld b,8
L1
23787 7E          ld a,(hl)
23788 12          ld (de),a
23789 24          inc h

```

23790 14	inc d	23813 10 FA	djnz L3
23791 10 FA	djnz L1	23815 D1	pop de
23793 D1	pop de	23816 18 D7	jr L4
23794 C1	pop bc		
23795 10 EE	djnz L2		

REMOVE CHR FROM STACK
AND PLACE IN BOTTOM SCREEN ROW

SCREEN ZONE TRANSFER

23797 21 20 07	ld hl,1824	END	
		23818 11 20 00	ld de,32
23800 19	add hl,de	23821 ED 52	sbc hl,de
23801 7C	ld a,h	23823 06 08	ld b,8
		L7	
CHECK FOR OUT OF SCREEN		23825 F1	pop af
ie START OF ATTRIBUTES		23826 77	ld (hl),a
		23827 25	dec h
23802 FE 58	cp 88	23828 10 FB	djnz L7
23804 28 0C	jr z,END		
23806 E5	push hl	MOVE TO NEXT COLUMN AND REPEAT	
23807 06 08	ld b,8		
L3		23830 C1	pop bc
23809 7E	ld a,(hl)	23831 D1	pop de
23810 12	ld (de),a	23832 13	inc de
23811 24	inc h	23833 10 BB	djnz L5
23812 14	inc d	23835 C9	ret

Program 6.19B

You will note that we start Program 6.19B with the DE register pair holding the address of the top byte of the start column and the B register holding the number of columns to be rolled. As listed, the program is completely self-contained and can be positioned anywhere in memory.

Even this is not the end of the memory saving. Would you believe that it was possible to reduce Program 6.19B to just 58 bytes *and* add an extra facility of being able to define the rows required, thus giving a roll window routine? Well, Program 6.19C does just that.

The memory saving is achieved by checking for the end of a screen zone using the BIT 0,H instruction. If BIT 0,H is not zero then we have come to the end of a zone and, instead of adding 32d to DE to move up one line within a zone, we add 1824 to move to the 'first' line in the next zone. Using this method we can define how many rows require rolling by the number of times loop L4 is repeated. The start ROW/COLUMN is defined by the display address held in the DE register pair (23671/3). To calculate this start address use the following formula:

$$16384 + (1824 * \text{INT}(\text{ROW}/8)) = ((\text{ROW} - \text{INT}(\text{ROW}/8)) * 32) + \text{COLUMN}$$

In this formula the Spectrum ROW/COLUMN numbers apply, that is 1st row = 0 1st column = 0.

Example: ROW 17 COLUMN 5

$$\begin{aligned}
 &= 16384 + (1824 * 2) + ((17 - 6) * 32) + 5 \\
 &= 16384 + 3648 + 32 + 5 \\
 &= 20069
 \end{aligned}$$

```

org 23760                                23779 C5                                push bc
CHAPTER 6                                23780 21 20 00                            ld hl,32
PROGRAM 19C                              23783 19                                add hl,de

UP ROLL WINDOW                            CHECK SCREEN ZONE CHANGE

EXAMPLE
COLUMNS 4 TO 10
ROWS 5 TO 18

23760 AF                                xor a

LD DE,SCREEN START ADDRESS
ie COLUMN/ROW START ADDRESS

23761 11 83 40                            ld de,16515

LD B,No of columns

23764 06 07                                ld b,7

L6
23766 D5                                push de
23767 C5                                push bc
23768 D5                                push de
23769 E1                                pop hl
23770 06 08                            ld b,8
L1
23772 7E                                ld a,(hl)
23773 F5                                push af
23774 24                                inc h
23775 10 FB                            djnz L1

LD B,No of rows-1!
23777 06 0D                            ld b,13
L4
23784 CB 44                                bit 0,h
23786 28 04                                jr z,L2
23788 21 20 07                            ld hl,1824
23791 19                                add hl,de
L2
23792 E5                                push hl
23793 06 08                            ld b,8
L3
23795 7E                                ld a,(hl)
23796 12                                ld (de),a
23797 14                                inc d
23798 24                                inc h
23799 10 FA                            djnz L3
23801 D1                                pop de
23802 C1                                pop bc
23803 10 E6                            djnz L4
23805 06 08                            ld b,8
L5
23807 25                                dec h
23808 F1                                pop af

23809 77                                ld (hl),a
23810 10 FB                            djnz L5
23812 C1                                pop bc
23813 D1                                pop de
23814 D1                                pop de
23815 13                                inc de
23816 10 CC                            djnz.L6
23818 C9                                ret

```

Program 6.19C

To roll the whole screen DE should contain 16384, the B register (before L6) should contain 32 and the B register (before L4) should contain 23. This program is easily adapted to produce UP/DOWN SCROLL and DOWN ROLL routines.

Pixel movement

This type of movement for foreground characters is not really suitable for games programs as it does tend to be rather slow. The movement is reduced by a factor of eight and if you are checking for, say, a character hitting an obstacle then each bit of the character shape must be checked using the machine code equivalent of POINT

x,y to see whether its new position is clear before replotting. So if a character is 16×16 pixels in size then even if the outside pixel positions only are checked this would mean perhaps 100 calls to the POINT routine before moving the character. This all takes time and that means that speed of movement is drastically reduced. Background movement, however, is possible one pixel at a time as long as the attributes remain fixed (these cannot be moved one pixel).

Demonstration program 'PIXEL DEMO' is available on tape and is our first arcade games program. This demonstrates the slowness of pixel movement and also the fact that there is a great deal of flicker due to the background movement. The balloon must be removed from the screen during background movement and a POINT check made before replotting it. This time delay is noticeable but has been kept as short as possible by the choice of balloon in which only the outline is necessary, thus reducing the number of POINT checks and PLOT calls to a minimum. The principles are the same as in character square movement in that the x,y PLOT positions are stored as data, a duplicate set of data updated and checked, and the balloon erased using OVER 1 and replotted using the duplicate data. Note that the balloon is able to move off the left and right sides of the screen in a wrap-around way. In fact this is necessary to escape the arrows on the bottom line.

For those of you with colour sets, the program is in black and white mainly because if more than two colours were used the balloon would change colour as it moved from one attribute square to another and perhaps become indistinct but also because it looks equally good on a black and white set.

The program demonstrates the principles behind left and right screen roll in that each character byte is rotated left RLA or right RRA and, if a bit is transferred into CARRY, it is picked up by the adjacent character byte. This will be discussed more in Chapter 9.

To demonstrate pixel down roll the following program (Program 6.20) will plot a sine wave and draw a line on the top of the screen then down roll it until we end up with a very smooth moving black wavy 'road'. This could be used in a motor-racing type of program. It also demonstrates the method of putting non-integer steps into a loop (see Chapter 7).

```

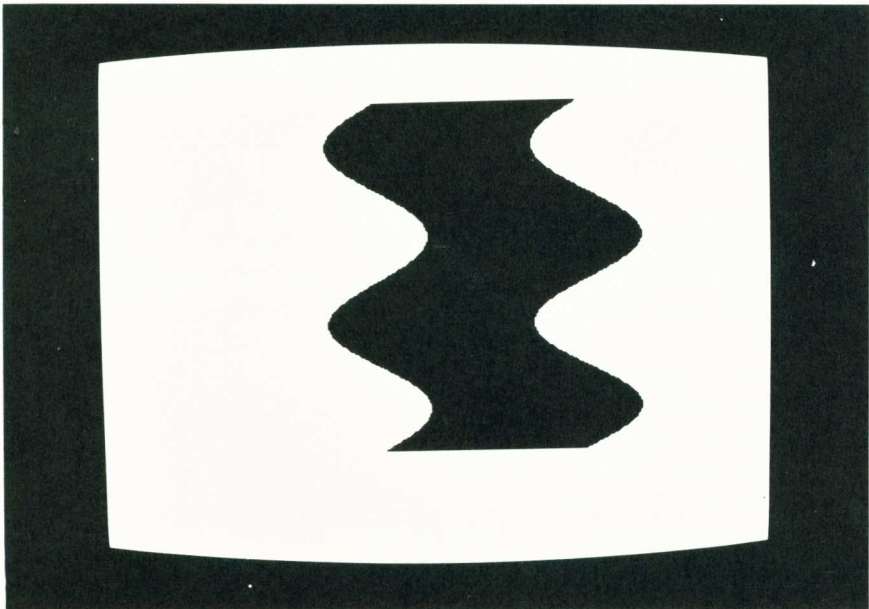
10 FOR a=0 TO 4*PI STEP PI/44      org 23760
PLOT 100+SIN a*25,175: DRAW
100,0                               SET a=0 ( on calculator )
30 REM CALL ONE PIXEL DOWN ROL     23760 EF          rst 40
L                                     defb 160 56
40 NEXT a

```

STORE value a in DATA		23813 CD DC 22	call 8924
L1		SAVE H'L'	
23763 11 35 5D	ld de,DATA		
23766 01 05 00	ld bc,5	23816 D9	exx
23769 ED B0	ldir	23817 E5	push hl
		23818 D9	exx
STACK 4*PI (on calc)			
23771 EF	rst 40	LD B,0 LD C,100	
defb 163 56			
23774 34	inc (hl)	23819 01 64 00	ld bc,100
23775 34	inc (hl)		
23776 34	inc (hl)	LD DE, +ve +ve	
SUB a-4*PI		23822 11 01 01	ld de,257
23777 EF	rst 40	CALL DRAW	
defb 3 56			
CHECK 1st MANTISSA BYTE FOR -ve ANSWER		23825 CD BA 24	call 9402
		RESTORE H'L'	
23780 23	inc hl	23828 D9	exx
23781 7E	ld a,(hl)	23829 E1	pop hl
23782 CB 7F	bit 7,a	23830 D9	exx
23784 20 04	jr nz,L2	STACK DATA a	
CLEAR CALCULATOR if -ve		23831 CD 24 5D	call STACK
23786 CD F1 2B	call 11249	a+PI/44	
RETURN TO BASIC		23834 EF	rst 40
23789 C9	ret	stack PI/2	
		defb 163	
CLEAR CALCULATOR		stack 22	
L2		defb 52 53 48	
23790 CD F1 2B	call 11249	divide	
23793 CD 3A 5D	call ROLL	defb 5	
STACK DATA a		add	
		defb 15	
23796 CD 24 5D	call STACK	end calc routine	
		defb 56	
100+SIN a*25=x		23842 18 AF	jr L1
23799 EF	rst 40	STACK a SUB	
sine		STACK	
defb 31		23844 21 35 5D	ld hl,DATA
stack 25		23847 ED 5B 65 5C	ld de,(23653)
defb 52 53 72		23851 01 05 00	ld bc,5
mult		23854 ED B0	ldir
defb 4		23856 ED 53 65 5C	ld (23653),de
stack 100		23860 C9	ret
		DATA a STORE	
defb 52 55 72		DATA	
add		defb 0 0 0 0 0	
defb 15		DOWN ROLL	
stack 175 (=y)		ROLL	
defb 52 56 47 56		23866 06 20	ld b,32
CALL PLOT x,y		23868 21 A0 57	ld hl,22432

L3			23939 10 FB	djnz L4
23871 7E	ld a,(hl)		23941 C9	ret
23872 F5	push af			
23873 23	inc hl	DOWN		
23874 10 FB	djnz L3	23942 C5		push bc
23876 21 BF 56	ld hl,22207	23943 06 07		ld b,7
23879 11 BF 57	ld de,22463	L5		
23882 06 06	ld b,6	23945 C5		push bc
23884 CD 86 5D	call DOWN	23946 E5		push hl
23887 21 E0 4F	ld hl,20448	23947 E5		push hl
23890 11 00 50	ld de,20480	23948 01 20 00		ld bc,32
23893 01 20 00	ld bc,32	23951 ED B8		lddr
		23953 D1		pop de
		23954 E1		pop hl
		23955 25		dec h
		23956 C1		pop bc
23896 ED B0	ldir	23957 10 F2		djnz L5
23898 21 FF 4E	ld hl,20223	23959 7C		ld a,h
23901 11 FF 4F	ld de,20479	23960 C6 08		add a,8
23904 06 08	ld b,8	23962 67		ld h,a
23906 CD 86 5D	call DOWN	23963 7D		ld a,1
23909 21 E0 47	ld hl,18400	23964 D6 20		sub 32
23912 11 00 48	ld de,18432	23966 6F		ld l,a
23915 01 20 00	ld bc,32	23967 E5		push hl
23918 ED B0	ldir			
23920 21 FF 46	ld hl,18175	23968 E5		push hl
23923 11 FF 47	ld de,18431	23969 01 20 00		ld bc,32
23926 06 08	ld b,8	23972 ED B8		lddr
23928 CD 86 5D	call DOWN	23974 E1		pop hl
23931 06 20	ld b,32	23975 D1		pop de
23933 21 1F 40	ld hl,16415	23976 25		dec h
L4		23977 C1		pop bc
23936 F1	pop af	23978 10 DA		djnz DOWN
23937 77	ld (hl),a	23980 C9		ret
23938 2B	dec hl			

Program 6.20



The PIXEL DOWN ROLL routine at the end of Program 6.20 has again been written so that it should be easy for you to see the procedure necessary. This can be rewritten on the lines of Program 6.19C to pixel down roll a window, at the same time making it 'position independent' and using less memory. In its present form it takes 115 bytes (+ 32 bytes to store the bottom line) but rewritten as Program 6.20A takes just 44 bytes (+ 1 to store the bottom column byte).

```

org 23760
CHAPTER 6
PROGRAM 20A

DOWN ROLL WINDOW

EXAMPLE

COLUMNS 4 TO 10
LINES 4 TO 39

REM LINE 1/COLUMN 1=TOP LH.

23760 11 83 46    ld de,18051

LD B,Columns

23763 06 07    ld b,7
L2

23765 C5        push bc

23766 D5        push de
23767 1A        ld a,(de)
23768 F5        push af
23769 D5        push de
23770 E1        pop hl
23771 25        dec h

23772 06 23    ld b,35
L1

23774 E5        push hl
23775 7E        ld a,(hl)
23776 12        ld (de),a
23777 D1        pop de
23778 7C        ld a,h
23779 25        dec h
23780 E6 07    and 7
23782 20 0A    jr nz,L3
23784 7D        ld a,l
23785 D6 20    sub 32

23787 6F        ld l,a
23788 38 04    jr c,L3
23790 7C        ld a,h
23791 C6 08    add a,8
23793 67        ld h,a
L3

23794 10 EA    djnz L1
23796 F1        pop af
23797 12        ld (de),a
23798 D1        pop de
23799 C1        pop bc
23800 13        inc de
23801 10 DA    djnz L2
23803 C9        ret

```

LD B,Lines-1 !

Program 6.20A

The number of columns to be rolled is held in the B register (before L2) and the number of lines - 1 in the B register (before L1). The DE register is loaded with the screen address of the first column bottom byte and is calculated using:

LET A = LINE (0 is first line)

LET B = INT (A/64)

LET C = A - (b * 64)

LET D = INT (C/8)

LET E = C - (D * 8)

LET F = COLUMN (0 is first column)

ADDRESS = 16384 + (1824 * B) + (256 * E) + (32 * (D - 1)) + F

EXAMPLE LINE 38 : COLUMN 3

A = 38 : B = 0 : C = 38 : D = 3 : E = 6 : F = 3

ADDRESS = 16384 + (1824 * 0) + (256 * 6) + (32 * 4) + 3 = 18051

As this formula is rather lengthy you could write a machine code program to do the calculation for you. Start with the HL register pair holding the LINE/COLUMN number and end with this converted to the screen address in DE so that you may go straight into the ROLL routine (or alternatively have a table showing the individual addresses of the lines/columns for the screen memory map.

Why not try to write such a routine? It could be useful in many programs, for example, to POKE characters directly onto the screen. If you get stuck, then Program 6.20B gives one method and takes just 29 bytes (this would be 28 bytes if LINE/COLUMN were placed in HL using LD HL, 034Fh). I have attached the program to Program 6.20A.

Study this program until you are confident that you know how it works as it will give you a better understanding of the display file.

```

org 23760                                23785 5F          ld e,a
CHAPTER 6                                23786 7D          ld a,l
PROGRAM 20B                              23787 83          add a,e
                                           23788 5F          ld e,a

DOWN ROLL WINDOW

EXAMPLE

                                LD B,Columns
COLUMNS 3 TO 9
LINES 56 TO 79

REM LINE 0/COLUMN 0=TOP LH.

LD H,Line
23760 26 4F          ld h,79

LD L,Column
23762 2E 03          ld l,3

                                LD B,Lines-1 !
23764 11 00 40      ld de,16384
23767 7C            ld a,h
23768 E6 07          and 7
23770 B2            or d
23771 57            ld d,a
23772 7C            ld a,h
23773 E6 C0          and 192
23775 1F            rra
23776 1F            rra
23777 1F            rra
23778 82            add a,d
23779 57            ld d,a
23780 7C            ld a,h
23781 E6 38          and 56
23783 17            rla
23784 17            rla

                                23789 06 07          ld b,7
                                L2
                                23791 C5          push bc
                                23792 D5          push de
                                23793 1A          ld a,(de)
                                23794 F5          push af
                                23795 D5          push de
                                23796 E1          pop hl
                                23797 25          dec h

                                23798 06 17          ld b,23
                                L1
                                23800 E5          push hl
                                23801 7E          ld a,(hl)
                                23802 12          ld (de),a
                                23803 D1          pop de
                                23804 7C          ld a,h

                                23805 25          dec h
                                23806 E6 07          and 7
                                23808 20 0A          jr nz,L3
                                23810 7D          ld a,l
                                23811 D6 20          sub 32
                                23813 6F          ld l,a
                                23814 38 04          jr c,L3
                                23816 7C          ld a,h

```



```

23817 C6 08      add a,8
23819 67        ld h,a
L3
23820 10 EA      djnz L1
23822 F1        pop af
23823 12        ld (de),a
23824 D1        pop de
23825 C1        pop bc
23826 13        inc de
23827 10 DA      djnz L2
23829 C9        ret

```

Program 6.20B

These two programs for pixel movement (Programs 6.20A and 6.20B) show methods of rolling the background left/right and down. I will leave it to you to produce programs to UP ROLL and SCROLL. It must be remembered that in machine code there is no correct method. Each person will have his or her own personal way of doing things, choosing registers, and deciding whether to push/pop, or store value etc. and as long as the program works to the individual's satisfaction then all is well. Many of the programs discussed so far may well be rewritten to save memory or to reduce running time but in doing so they may not be so easy to follow for someone beginning programming.

There is one more useful ROM routine that is not available from BASIC and that is SCROLL n lines with attributes, where n must be greater or equal to two (but not greater than 24 of course).

Program 6.21 shows the routine being used to clear lines 16 to 24 inclusive (nine lines) by calling the routine nine times. Note that the number of lines to be scrolled are counted from line 24. This routine could be used instead of our user-written one in the 'MOVEDEMO' program. The B register holds the number of lines to be scrolled - 1 and then a call to 3584 is made. The bottom line is overprinted with spaces using colours from ATTR P.

```

org 23760
23760 06 09      ld b,9
L1
23762 C5        push bc
23763 06 08      ld b,8
23765 CD 00 0E   call 3584
23768 C1        pop bc
23769 10 F7      djnz L1
23771 C9        ret

```

Program 6.21

Finally, here is a routine to roll a screen left (or right) by half a character at a time. This program, although short, is rather more difficult to understand and makes use of the rotate left decimal (or rotate right decimal) instruction. This instruction involves the nibble-handling of the A register and the contents of the address in

the HL register pair. RLD transfers bits 0 to 3, (HL) into bits 7 to 4, (HL); bits 7 to 4, (HL) into bits 0 to 3, A; and bits 0 to 3, A into bits 0 to 3, (HL). Incidentally it affects none of the 'flags'. You can see from Program 6.22 that this instruction is used on each screen line using the A register to store the right-hand nibble and to place it into the next character left-hand nibble. The first character left-hand nibble is blanked out because the A register holds 0 at the start of each line. At the end of each line the A register holds the left-hand nibble which will be shifted off the screen. This is used to add to the line start byte to give a wrap-around effect. If you required a continually changing background you could hold the replacement right-hand nibble in data form to be added to the right-hand byte instead of using the nibble shifted off the screen as above. I will leave the RIGHT ROLL routine for you to write (one tip – start at the beginning of the display file).

org 23760		Rotate left a, (hl)
CHAPTER 6		nibble instruction
PROGRAM 22		
		L2
HALF CHARACTER SQUARE		23771 ED 6F rld
LEFT ROLL ROUTINE		23773 2B dec hl
		23774 10 FB djnz L2
23760 F3 di		Get line start in de
Start at end of display		
23761 21 FF 57 ld hl,22527		23776 D1 pop de
No. of screen lines		Store present scr. pos.
23764 06 C4 ld b,196		23777 EB ex de,hl
L1		Add LH nibble to
23766 C5 push bc		RH byte
Reset a=0		23778 86 add a, (hl)
		23779 77 ld (hl),a
23767 AF xor a		Restore scr. pos. in hl
No. of bytes/line		23780 EB ex de,hl
23768 06 20 ld b,32		23781 C1 pop bc
Store line start		23782 10 EE djnz L1
		23784 FB ei
23770 E5 push hl		23785 C9 ret

Program 6.22

7 MUSIC AND SOUND EFFECTS

In this chapter we will be looking at ways to improve your animated graphics with interesting sounds and 'music'. But firstly we must take a look at the Spectrum way of storing numbers in floating point form and also how to convert a floating point number to its compressed form. If your maths are a little rusty then do not worry too much about the theory as the Spectrum can help. Put simply, any number whether decimal, integer, positive, or negative can be expressed in a five-byte form using a set procedure to define each byte. The first byte is called the *exponent* and the remaining four bytes are called the *mantissa*.

The conversion of a 'normal' number to its five-byte form involves four stages as follows:

1. Express the number in its *binary* form.

For example 11.375 would become 1011.011

Remember that $0.1 \text{ binary} = \frac{1}{2}$

$0.01 \text{ binary} = \frac{1}{4}$

etc.

2. Find the *exponent*.

Move the decimal point either left or right until it is to the left of the first 1 in the binary number and count the number of moves taken.

The EXPONENT will be $128 +$ the number of moves taken.

If the moves are to the left then the 'moves' are positive.

If the moves are to the right then the 'moves' are negative.

In our example the exponent = $128 + 4 = 132$

and we are left with .1011011

3. Alter the bit to the right of the decimal point to indicate the sign of the original number.

If the number was positive then the bit is reset.

If the number was negative then the bit is left set.

In our example the number was positive and becomes .0011011

4. Find the *mantissa*.

Discard the decimal point and divide the remaining binary number into four bytes, adding zero as necessary to make up to 32 bits.

00110110 00000000 00000000 00000000

and then convert each byte back to its decimal number.

And so it can be seen that in our example

11.375 = 132 54 0 0 0

If by now you are thoroughly confused then, as I have said, the Spectrum can help. The Spectrum has a useful method (useful to us, that is) of storing simple variables in that they are stored immediately after the variables' code in their five-byte floating point form. So all we really need to do is to write a short program to ENTER a number and then PEEK at the variable store to find the five-byte form.

```
5 GO TO 5000
10 REM PROGRAM TO PRINT
   FLOATING POINT FORM OF ANY
   NUMBER
20 PRINT AT 3,0;"THE FOLLOWING
SHORT PROGRAM WILLPRINT THE FLOAT
ING POINT FORM OFANY NUMBER, DECI
MAL, INTEGER ANDNEGATIVE NUMBERS.
   INTEGER NUMBERS HA
VE THE DECIMAL0.00000000001 ADDE
D TO CONVERTTO A FLOATING POINT
NUMBER BUTNOT AFFECT ITS VALUE.
"
25 PRINT ?? "COMPRESSED FORM NUM
BERS ARE ALSO GIVEN FOR USE WITH
CALCULATORLITERAL 34H (52d).
"
40 RETURN
100 INPUT "NUMBER:";TAB 9;n
200 LET n$="NUMBER:"
300 LET b$="FLOATING POINT : "
400 LET c$="COMPRESSED FORM:"
500 LET d$="0"
1003 IF N<>0 THEN GO TO 1010
1005 PRINT n$;TAB 9;d$; PAPER 6;0
"b$;TAB 9;d$;"0 ";d$;"0 ";d$;"0
";d$;"0 ";d$;"0"
1006 PRINT c$;TAB 9;d$;"0 ";d$;"
176";d$;"0"
1007 GO TO 100
1010 LET N=N+.000000000001
1020 PRINT n$;TAB 9;d$; PAPER 6;N
1025 PRINT b$;
1030 LET X=PEEK 23627+256*PEEK 23
628
1040 FOR A=1 TO 5
1050 PRINT TAB 5+4*A;d$;PEEK (X+A
);
1060 NEXT A
1065 PRINT ?
1070 DIM a(5)
1075 LET q=0
1076 PRINT c$;
1080 FOR b=5 TO 1 STEP -1
1085 LET a(b)=PEEK ((PEEK 23627+2
56*PEEK 23628)+b)
1086 IF b=2 THEN GO TO 1100
1090 IF q=0 AND a(b)=0 THEN NEXT
b
```

```

1100 LET q=q+1
1110 NEXT b
1120 LET q=q-2
1125 LET x=(a(1)-80)
1126 IF x>=64 THEN GO TO 2000
1130 LET e=x+(q*64)
1140 PRINT TAB 9;d$;e;
1150 FOR b=1 TO q+1
1160 PRINT TAB 9+b*4;d$;a(1+b);
1170 NEXT b
1180 PRINT
1190 GO TO 100
2000 PRINT TAB 9;d$;q*64;TAB 13;d
$;a(1)-80;
2010 FOR b=1 TO q+1
2020 PRINT TAB 13+b*4;d$;a(1+b);
2030 NEXT b
2040 PRINT : GO TO 100
5000 DATA 24,79,5,2,"STOP THE TAP
E"
5100 DATA 15,167,7,4,"NUMBERS"
5200 DATA 15,130,2,4,"-----"
5250 DATA 24,15,1,2,"PRESS ANY KE
Y"
5500 DATA 31,170,1,2,"INSTRUCTION
S"
6000 BORDER 4: PAPER 6: INK 0: CL
S
6010 LET n=0
6050 RESTORE 5000
6100 FOR a=1 TO 4
6200 INK a: GO SUB 8000
6300 PAUSE 50: NEXT a
6400 PAUSE 0: CLS
6420 INK 2: GO SUB 8000
6450 INK 0: PAPER 7: GO SUB 10
7000 RESTORE 5250: INK 2: GO SUB
8000
7005 IF INKEY$<>" THEN GO TO 70
05
7006 IF INKEY$="" THEN GO TO 700
6
7010 CLS : GO TO 100
8000 READ x,y,h,w,a$
8010 LET c=USR 32393
8020 RETURN
9998 CLEAR 32334: LOAD "udgs"CODE
USR "a",168: LOAD ""CODE : GO TO
5000
9999 SAVE "NUMBERS" LINE 9998: SA
VE "udgs"CODE USR "a",168: SAVE "
LARGE"CODE 32335,265
<336@ S.A.Nicholls

```

Program 'Numbers'

This program is available on tape with colours held as part of strings n\$ b\$ c\$ and d\$.

When RUN, the program will print the five-byte floating point *and* compressed form of any number. The compressed form will be discussed later. Note that 0 is treated as a special case in which all bytes are 0, and also that 0.5 and $\frac{1}{2}$ give different results.

By now you are probably wondering what this has to do with music and sounds. The answer is that the BASIC BEEP command makes use of the calculator stack and any number stored on the stack must be in its five-byte form.

We can now look at the method of converting the BASIC command BEEP duration, pitch into machine code. This entails stacking the value of duration and then pitch in their five-byte forms on top of the calculator stack and then calling the BEEP routine from address 03F8h (1016d).

If we take, for example, BEEP .1,11, then using the NUMBERS program we can convert .1 and 11 to their five-byte form:

```
.1 = 125 76 204 204 204
11 = 132 48 0 0 0
```

We must now find a way of stacking these ten bytes consecutively on the calculator stack. One method would be to find the start address to which the DATA must be transferred by PEEKing at the system variable STACKEND and then to use an LDIR instruction as shown in Program 7.1.

```
org 23760                                23772 ED 53 65 5C 1d (23653),de
CHAPTER 7
PROGRAM 1                                CALL BEEP

BEEP .1,11

FIND STACKEND                            23776 CD F8 03    call 1016
                                           23779 C9          ret
DATA
0.1 (5 byte form)
23760 ED 5B 65 5C 1d de,(23653)          defb 125 76 204 204 204
23764 21 E4 5C    1d hl,DATA
23767 01 0A 00    1d bc,10
23770 ED B0      ldir
                                           11 (5 byte form)
                                           defb 132 48 0 0 0

UPDATE STACKEND
```

Program 7.1

The RESET is needed to reload STACKEND with the new address of the top of the stack. Remember that LDIR ends with HL and DE holding the *address* of the last byte transferred +1. The BEEP routine when called from address 03F8h takes the top two values from the stack and manipulates them to give the required duration and pitch. The stack is thus reset to give a safe return to BASIC.

Another method of stacking integer numbers is to use LD A,n and CALL STACK A subroutine 2D28h (11560d) for numbers 0 to 255, or LD BC,nn and CALL STACK BC subroutine 2D2Bh (11563d) for numbers 0 to 65535. Program 7.1 could then be written as Program 7.2.

```
org 23760
CHAPTER 7
PROGRAM 2
```

```
23772 ED 53 65 5C 1d (23653),de
23776 3E 0B          1d a,11
```

```
BEEP .1,11
```

```
STACK value a
```

```
FIND STACKEND
```

```
23778 CD 28 2D      call 11560
```

```
23760 ED 5B 65 5C 1d de,(23653)
23764 21 E9 5C      1d h1,DATA
23767 01 05 00      1d bc,5
23770 ED B0          1dir
```

```
CALL BEEP
```

```
23781 CD F8 03      call 1016
23784 C9            ret
DATA
0,1 (5 byte form)
defb 125 76 204 204 204
```

```
UPDATE STACKEND
```

Program 7.2

Lastly, for those who wish to take floating point numbers one stage further, we can convert them to their *compressed form* and use the calculator literal 34h (52d) to stack data in this compressed form.

The stages for converting five-byte floating numbers are as follows:

1. Starting from the fifth byte, remove all zero numbers until a non-zero number is reached *or* until the second byte is reached.
2. The quotient is then the number of remaining bytes *less 1*.
3. Subtract 50h (80d) from the exponent and, if the remainder is greater than or equal to 40h (64d) then go to instruction 6.
4. Change the exponent to (exponent - 80d) + (quotient * 64d).
5. Go to instruction 7.
6. The exponent becomes two bytes. The first byte is the quotient *64d; the second byte is the exponent - 80d.
7. Place the remaining bytes after the modified exponent byte(s).

To convert a compressed number back to its five-byte form then the following stages are taken:

1. Divide the first byte by 64d.
2. If there is a remainder then the exponent will be 80d + remainder.
3. If there is no remainder then the exponent will be 80d + the second byte.
4. The quotient is used to find how many extra bytes follow the exponent. The number of extra bytes is the quotient + 1.

As an example $11 = 52\ 48$ in compressed form.

52 divided by $64 = 0$ remainder 52

quotient = 0

further bytes = 1

As there was a remainder exponent = $52 + 80 = 132$

1 further byte specified = 48

Make up to 5 bytes with 0 0 0

So $52\ 48 = 132\ 48\ 0\ 0\ 0$

Again if all this is too much for you at present then the NUMBERS program will do all the converting for you. Using compressed form numbers BEEP .1,11 can be converted to machine code as shown in program 7.3.

```

org 23760                                STACK 11 (COMP. FORM)
CHAPTER 7
PROGRAM 3                                defb 52 52 48

                                           END CALC

BEEP .1,11

USE CALCULATOR

23760 EF          rst 40                defb 56
                                           CALL BEEP
STACK 0.1 (COMP. FORM)
defb 52 237 76 204 204 204            23771 CD F8 03      call 1016
                                           23774 C9          ret

```

Program 7.3

In Program 7.3 RST 28h (40d) calls the CALCULATOR routine and defb 38h (56d) is the END CALC literal.

The FANFARE BEEP for the 'CROSS' program is a combination of all three methods, namely, holding numbers as DATA, using stack A and using literal 34h (52d) to stack compressed form numbers. In this program it is fortunate that the mantissa bytes of the compressed form of decimals .1 .8 and .05 are the same, so that the exponent byte only needed to be stored as DATA.

```

1 REM CHAPTER 7                          VALUE
2 REM PROGRAM 4                          defb 0 76 204 204 204 56
REM FANFARE BEEP                         23780 E1          pop hl
10 FOR a=1 TO 8                           23781 7E          ld a,(hl)
20 READ b,c                               23782 E5          push hl
30 BEEP b,c                               23783 CD 28 2D   call 11560
50 NEXT a                                 23786 CD F8 03   call 1016
60 DATA .1,11,.1,11,.8,16,.05,         23789 E1          pop hl
11,.05,16,.05,11,.05,16,1,20           23790 C1          pop bc

org 23760
23760 06 07          ld b,7
23762 21 00 5D      ld hl,DATA
LOOP
23765 7E          ld a,(hl)
23766 32 DE 5C     ld (VALUE),a
23769 23          inc hl
23770 C5          push bc
23771 E5          push hl
23772 EF          rst 40
defb 52

23791 23          inc hl
23792 10 E3       djnz LOOP
23794 3E 01       ld a,1
23796 CD 28 2D   call 11560
23799 3E 14       ld a,20
23801 CD 28 2D   call 11560
23804 CD F8 03   call 1016
23807 C9          ret
DATA
defb 237 11 237 11 240 16 236
defb 11 236 16 236 11 236 16

```

Program 7.4

So far I have shown you how to convert any BASIC BEEP command into machine code. This is fine for 'music' but will not produce anything like a decent sound effect. For this type of sound we will need to enter the BEEP routine at a different point, 03B5h (949d). After the values for duration and pitch have been removed from the stack they are manipulated and end up with 'pitch' in the HL register and 'duration' in the DE register. So a program could be written to give a falling tone by incrementing the value stored in HL and calling the BEEP routine repeatedly in a 'FOR/NEXT loop'. This can be achieved as shown in Program 7.5.

```

org 23760                                CALL BEEP
CHAPTER 7
PROGRAM 5

SOUND 1

23760 06 FF          ld b,255
23762 21 01 00      ld hl,1
LOOP
23765 11 01 00      ld de,1
23768 E5            push hl
23769 C5            push bc

23770 CD B5 03      call 949
23773 C1            pop bc
23774 E1            pop hl
23775 23            inc hl
23776 10 F3         djnz LOOP

23778 C9            ret

```

Program 7.5

You will notice when Program 7.5 is RUN that as the pitch gradually changes the duration also changes. This is because the duration is in fact a function of both values in HL and DE registers. To even out the duration we must decrease the value held in DE as the value in the HL register increases. So we arrive at Program 7.6.

```

org 23760                                23770 C5            push bc
CHAPTER 7                                23771 CD B5 03      call 949
PROGRAM 6                                23774 C1            pop bc
                                           23775 E1            pop hl
SOUND 2                                  23776 D1            pop de
                                           23777 23            inc hl
23760 06 64          ld b,100           23778 1B            dec de
23762 21 01 00      ld hl,1           23779 10 F3         djnz LOOP
23765 11 65 00      ld de,101
LOOP
23768 D5            push de           23781 C9            ret
23769 E5            push hl

```

Program 7.6

We have one last method of producing sound from the Spectrum and that is by sending 'signals' directly to the speaker switching it high or low in the same way as the ROM routine does. There seems little point in using this method for musical notes as we would need to know the frequency of the high/low switching for each note and would in fact duplicate the ROM routine. However it does have a use in games programs for producing a 'white noise' type of sound for explosions etc.

The instruction to communicate with the speaker is OUT (254), A. This instruction controls not only the speaker but also the temporary border colour in the following way:

1. If bit 0,A is set then blue is on.
If bit 0,A is reset then blue is off.
2. If bit 1,A is set then red is on.
If bit 1,A is reset then red is off.
3. If bit 2,A is set then green is on.
If bit 2,A is reset then green is off.
4. If bit 4,A is set then speaker is 'high'.
If bit 4,A is reset then speaker is 'low'.

You can see from the above that if bit 4 were set and reset at regular intervals then a note would be produced dependent on that interval.

To produce true white noise we would need random interval switching but this would mean a lengthy routine to produce a random number and check bit 4,A each time causing a 'long' pause between switching. We must therefore find a quick method of producing 'random numbers'. This is not as hard as it sounds, for we are only interested in the state of bit 4,A and for our purposes the ROM bytes are random enough. With this in mind we can write a program to look at each ROM address in turn checking the state of bit 4 of the number held in that address and thus switch the speaker high or low accordingly.

```

org 23760
23760 21 00 00    ld hl,0
L1
23763 7E        ld a,(hl)
23764 00        nop
23765 D3 FE     out (254),a
23767 00        nop
23768 00        nop
23769 D3 FE     out (254),a
23771 00        nop
23772 00        nop
23773 D3 FE     out (254),a
23775 00        nop
23776 23        inc hl
23777 7C        ld a,h
23778 FE 3C     cp 60
23780 20 ED     jr nz,L1
23782 C9        ret

```

Program 7.7

There are several points to note in Program 7.7.

1. The three OUT (254), A instructions and NOPs give a set delay. You may like to remove or add more and hear the effect.
2. The length of sound is governed by the CP 60 instruction. If this were reduced to say CP 40 the length of sound would be similarly reduced. Anything above 60 runs into an area of ROM with not so random numbers. Try it and see what happens.
3. The border colour is also temporarily affected as bits 0 – 2 are also changed at random. To keep the border the same colour you would need to replace bits 0 – 2 with bits 3 – 5 of the value held in the systems variable BORDER.

If you want to experiment with producing music by this method then remember that the ROM interrupt routine is called 50 times

per second and will upset your frequency so you will need to disable it (DI) before your routine and enable it (EI) afterwards. This in fact is what the ROM routine does and explains why you cannot BREAK a BASIC program during execution of a BEEP command.

I leave the rest to your imagination and ingenuity in altering the values held in HL and DE registers to produce wonderful and weird effects, but to give you some idea of sounds possible using the above method the tape contains a Program SOUND which has five different sounds accessed by pressing keys 1 to 5. The flashing border does give an extra dimension to sounds 1 and 5 and is very simple to program.

One last point – do keep sound effects as short as possible since they do halt all moving graphics during their execution. The best place for sounds is in delay loops used to slow down the moving graphics.



```

10 REM ? RETURN RUN GO SUB X P
PRINT THEN G TO ?] LET PRINT THEN
0 TO JJLET PRINT THEN W TO DJ
LET PRINT THEN _ TO _] LET PRINT
THEN g TO [ ] LET >? STEP £"? RE
TURN @ GO SUB X THEN G>=?<>
2 OR !d?
25>INK 2:PAPER 7:BORDER 4:CLS
30 LET x=16: LET y=100: LET w=2
: LET h=4: LET a$="STOP THE TAPE"
40 PAPER 8: RANDOMIZE USR 32393
50 PRINT #0;AT 0,3;"PRESS ANY K
EY TO START"

```

```

60 PAUSE 0
65 CLS
70 LET x=8: LET y=175: LET w=5:
LET h=15: LET a$="SOUND!"
80 INK 2: RANDOMIZE USR 32393
90 PRINT AT 19,2;"Press keys 1
to 5 for sounds"
100 PRINT AT 21,3;"SPACE will re
turn to BASIC"
110 RANDOMIZE USR 23760
120 CLS
130 LET x=24: LET y=100: LET h=5
: LET w=1: LET a$="To re RUN ente
r GOTO 65"
140 RANDOMIZE USR 32393
150 STOP
9998 CLEAR 32334: LOAD ""CODE : G
O TO 25
9999 SAVE "SOUNDS" LINE 9998: SAV
E "LARGE"CODE 32335,265: STOP
<336 @ S.A.Nicholls

```

```

org 23760
START
23760 01 FE F7    ld bc,63486
23763 ED 78      in a,(c)
23765 F5         push af
23766 CB 47      bit 0,a
23768 CC 07 5D   call z,BEEP1
23771 F1         pop af
23772 F5         push af
23773 CB 4F      bit 1,a
23775 CC 29 5D   call z,BEEP2
23778 F1         pop af
23779 F5         push af
23780 CB 57      bit 2,a
23782 CC 44 5D   call z,BEEP3
23785 F1         pop af
23786 F5         push af
23787 CB 5F      bit 3,a
23789 CC 5F 5D   call z,BEEP4
23792 F1         pop af

23793 F5         push af
23794 CB 67      bit 4,a
23796 CC 7B 5D   call z,BEEP5
23799 F1         pop af
23800 3E 07      ld a,7
23802 CD 9B 22   call 8859
23805 01 FE 7F   ld bc,32766
23808 ED 78      in a,(c)
23810 CB 47      bit 0,a
23812 C8         ret z
23813 18 C9      jr START
BEEP1
23815 06 32      ld b,50
L1
23817 C5         push bc
23818 21 64 00   ld hl,100
L2
23821 11 0A 00   ld de,10
23824 E5         push hl
23825 7D         ld a,l
23826 E6 07      and 7

```

```

23828 CD 9B 22    call 8859
23831 CD B5 03    call 949
23834 E1          pop hl
23835 11 0A 00    ld de,10
23838 AF          xor a
23839 ED 52      sbc hl,de
23841 7C          ld a,h
23842 B5          or l
23843 20 E8      jr nz,L2
23845 C1          pop bc
23846 10 E1      djnz L1
23848 C9          ret
BEEP2
23849 21 00 00    ld hl,0
L3
23852 7E          ld a,(hl)
23853 00          nop
23854 D3 FE      out (254),a
23856 00          nop
23857 00          nop
23858 D3 FE      out (254),a

```

```

23860 00          nop
23861 00          nop
23862 D3 FE      out (254),a
23864 00          nop
23865 00          nop
23866 D3 FE      out (254),a
23868 00          nop
23869 23          inc hl
23870 7C          ld a,h
23871 FE 3C      cp 60
23873 20 E9      jr nz,L3
23875 C9          ret

```

```

BEEP3
23876 06 05      ld b,5
L5
23878 C5          push bc
23879 21 32 00    ld hl,50
L4
23882 11 0A 00    ld de,10
23885 E5          push hl
23886 CD B5 03    call 949
23889 E1          pop hl
23890 11 32 00    ld de,50
23893 19          add hl,de
23894 7C          ld a,h
23895 FE 04      cp 4
23897 20 EF      jr nz,L4
23899 C1          pop bc
23900 10 E8      djnz L5
23902 C9          ret

```

```

BEEP4
23903 06 0A      ld b,10
L6
23905 C5          push bc
23906 21 00 04    ld hl,1024
L7
23909 11 01 00    ld de,1
23912 E5          push hl
23913 CD B5 03    call 949
23916 E1          pop hl
23917 AF          xor a
23918 11 10 00    ld de,16

```

```

23921 ED 52      sbc hl,de
23923 7C        ld a,h
23924 B5        or l
23925 20 EE      jr nz,L7
23927 C1        pop bc
23928 10 E7      djnz L6
23930 C9        ret
BEEF5
23931 21 01 00   ld hl,1
L8
23934 11 05 00   ld de,5
23937 E5        push hl
23938 CD B5 03   call 949
23941 E1        pop hl
23942 23        inc hl
23943 7C        ld a,h
23944 FE 02      cp 2
23946 20 F2      jr nz,L8
23948 06 FA      ld b,250
L9
23950 C5        push bc

23951 78        ld a,b
23952 E6 06      and 6
23954 CD 9B 22   call 8859
23957 21 90 01   ld hl,400
23960 11 01 00   ld de,1
23963 CD B5 03   call 949
23966 C1        pop bc
23967 10 ED      djnz L9
23969 C9        ret

```

Program 'Sound'

8 ATTRIBUTE, SCREEN\$ AND POINT

As you may know if you have tried BASIC games programming, there will come a time when you will need to find the character in a particular screen position. This is useful for example if checking that a missile has hit its target or that you have landed on the moon. In BASIC there are three methods of doing this, namely:

ATTRIBUTE which checks the INK: PAPER: FLASH: BRIGHT state of a character square and returns a number from 0 to 255.

SCREEN\$ which checks the character in a particular square (but only characters CODE 32 to 127; the rest returns an empty string).

POINT which checks the state of a screen pixel: 0 = unplotted 1 = plotted.

All three above methods are available in machine code by calls to the particular ROM routines.

ATTRIBUTE (line, column)

The machine code routine – call 9603 – requires the line number in the C register and the column number in the B register. The value of the ATTRIBUTE after the routine is called ends up as the top value on the calculator stack. The value of the ATTRIBUTE is made up in the following way:

Value = 128 * (FLASH number) + 64 * (BRIGHT number) + 8 * (PAPER colour) + INK colour

You can see from the above formula that the value has a range of 0 to 255, and so may easily be removed from the stack into the A register using the 'CALC VALUE TO A' routine – call 11733 – and then checked by a CP n instruction.

Program 8.1 shows the above method used to PRINT AT 0, 20; PAPER 1; INK 7; FLASH 1; CHR\$ 144 (which has been redefined as a man). The program then checks each screen position, shown by a > sign, until it finds ATTR value 143 and then returns to BASIC.

```
org 23760
CHAPTER 8
PROGRAM 1
```

```
ATTRIBUTE CHECK
```

```
SET UP UDG
```

```
23760 ED 5B 7B 5C 1d de, (23675)
23764 21 42 5D 1d h1, DATA3
23767 01 08 00 1d bc, 8
23770 ED B0 1dir
```

```
CLEAR SCREEN
```

23772 3E 02	ld a,2	23832 32 40 5D	ld (COL),a
23774 CD 01 16	call 5633		
23777 CD 68 0D	call 3435		GET COLUMN/LINE IN BC
23780 3E 02	ld a,2		
23782 CD 01 16	call 5633	23835 ED 4B 3F 5D	ld bc, (LINE)
			CALL ATTR. & GET VALUE IN A
PRINT UDG		23839 CD 83 25	call 9603
		23842 CD D5 2D	call 11733
23785 11 2E 5D	ld de,DATA1		
23788 01 10 00	ld bc,16		CHECK IF 143
23791 CD 3C 20	call 8252	23845 FE 8F	cp 143
SET OVER 1			MOVE TO NEXT POSN.
23794 FD 36 57 03	ld (iy+87),3	23847 20 D1	jr nz,L1
RESET START AT 0,0			RESET OVER 0
23798 AF	xor a	23849 FD 36 57 00	ld (iy+87),0
23799 32 40 5D	ld (COL),a	23853 C9	ret
PRINT >			PRINT UDG DATA
L1			DATA1
23802 11 3E 5D	ld de,DATA2		defb 22 0 20 17 1 16 7 18 1
23805 01 04 00	ld bc,4		defb 144 18 0 17 7 16 0
23808 CD 3C 20	call 8252		PRINT > DATA
DELAY			DATA2
23811 21 FF FF	ld hl,65535		defb 22
L2			LINE
23814 2B	dec hl		defb 0
23815 7C	ld a,h		COL
23816 B5	or l		defb 0 62
23817 20 FB	jr nz,L2		UDG DATA
REMOVE >			DATA3
23819 11 3E 5D	ld de,DATA2		defb 24 153 126 153 24 36 36 102
23822 01 04 00	ld bc,4		
23825 CD 3C 20	call 8252		
INC COLUMN			
23828 3A 40 5D	ld a, (COL)		
23831 3C	inc a		

Program 8.1

When run, Program 8.1 gives the appearance of a missile moving slowly towards a man and returns to BASIC when the man is hit. You could of course have a hit routine instead of the RETURN instruction (with sound effects). This method of checking screen positions for targets etc. is OK but a bit limiting in that all your targets must have the same ATTRIBUTE value.

SCREEN\$ (line, column)

As with the ATTR routine the line number is held in the C register and the column number in the B register before call 9528 is made. The routine places the *parameters of the string* on top of the calculator stack in five-byte form. The use of the calculator stack to hold string parameters is a new idea as so far we have only used it for holding five-byte numbers. Suffice it to say that there are calculator literals for manipulating strings, but instead of putting the 'string' on the stack their 'parameters' are held in a five-byte form. To put the parameters of the string on to the stack, the BC register pair holds the length of the string and the DE register pair holds the start address of the string. For simple strings the A register holds value 0. A call to the routine – call 10929 – will place the parameters in the correct order on the stack. Similarly the routine – call 11249 – will take the last 'value' from the stack and place it into A,B,C,D, and E registers; again BC will hold the string length and DE the start address.

Program 8.2 shows the routines being used to print the character @ at SCREEN position 0, 20; and when run will end with two @s at 0, 20; and 0, 21;. Instead of the PRINT STRING – call 8252 – we could have used LD A,(DE) to check the actual character in position 0, 20; and then to take the necessary actions.

However there is one limitation in the BASIC SCREEN\$ routine, and that is it will only find characters in the range CODE 32 – CODE 127; anything else will return an empty string. This in effect rules out using this method for finding UDGs and in BASIC this is a big handicap in games programs. In machine code this is no problem as there are 'quick' ways to fool the ROM and place the UDGs into the character set. The systems variable CHARS – 23606/7 usually holds the address of the start of the character set –256, that is, 15360 and is used by the SCREEN\$ routine to find the character. We could set this variable to point to the start of the UDGs –256 before the SCREEN\$ call is made (in effect making UDG 'A' into CODE 32, UDG 'B' into CODE 33, etc.) and then reset it to its normal value afterwards. In this way we can check through the UDGs to find the character we want.

```
org 23760                23762 CD 01 16      call 5633
CHAPTER 8                23765 CD 6B 0D      call 3435
PROGRAM 2                 23768 3E 02        ld a,2
                           23770 CD 01 16      call 5633

SCREEN$ (0,20)

CLEAR SCREEN              PRINT AT 0,20 @

                           23773 3E 16        ld a,22
23760 3E 02               ld a,2
                           23775 D7         rst 16
```

```

23776 3E 00      ld a,0          23789 CD 38 25      call 9528
23778 D7         rst 16
23779 3E 14      ld a,20        CALL 'VALUE TO A,B,C,D,E
23781 D7         rst 16
23782 3E 40      ld a,64        23792 CD F1 2B      call 11249
23784 D7         rst 16

CALL SCREEN$

23785 06 14      ld b,20        23795 CD 3C 20      call 8252
23787 0E 00      ld c,0         23798 C9           ret

```

Program 8.2

Program 8.3 is in effect the same as Program 8.1 but this time we are checking for CHR\$ 144 before returning to BASIC.

```

org 23836          23846 CD F1 2B      call 11249
CHAPTER 8
PROGRAM 3

SCREEN$ FOR UDG's          CHECK FOR CHR$ 32
                           i.e. CHR$ 144 (UDG 'A')

23760 TO 23835          23849 1A           ld a,(de)
SAME AS PROG. 1        23850 FE 20        cp 32

SET CHARS. TO UDG's - 256      RESET CHARS. TO NORMAL

23836 2A 7B 5C      ld h1,(23675)    23852 21 00 3C      ld h1,15360
23839 25            dec h           23855 22 36 5C      ld (23606),h1
23840 22 36 5C      ld (23606),h1

CHECK SCREEN$          REST OF PROGRAM
                           AS PROGRAM 1

23843 CD 38 25      call 9528        23847 TO END OF DATA3

```

Program 8.3

You could of course copy the character set into RAM and keep CHARS permanently pointing to the start -256. In this way you can forget the UDGs and redefine any character you wish in the new set (leave the alphabet and numbers unchanged so that text will be unaltered). This method will allow about 50 redefinable characters if you limit yourself to using text in *upper case* only and a minimum of punctuation marks. The SCREEN\$ routine will recognize these characters as they will appear in the new character set.

There is one very important point to note when using the calculator for string manipulation and that is that use is made of the workspace to hold the string and/or the result of the calculation. If this workspace is not cleared regularly then you may find that an out of memory report will suddenly appear in your program for no apparent reason or, worse still, the workspace will run into your machine code routine and your program will crash. In practice it is advisable to clear the workspace each time the SCREEN\$ routine is called. The ROM routine - call 5823 - will clear the workspace for you (and also clear the calculator stack). So Program 8.3 should

have call 5823 after LD A, (DE) but remember to save the value in A before the call.

POINT

The BASIC POINT x,y, returns a value of 1 if the pixel x,y is set (that is PLOTted) or 0 if reset (PLOT OVER). The machine code routine – call 8910 – requires the x coordinate to be held in the C register and the y coordinate in the B register before the call is made. As with the previous routines the result ends up as the last value on the calculator stack, and, as this value is either 0 or 1, we can unstack it into the A register to compare it to a required value. Program 8.4 shows how this is used to check screen position 0,175, that is, the top left-hand corner. It first prints a black square at 0,0; then returns to BASIC as POINT 0,175 = 1. Use is made of this routine in the 'balloon' program to POINT each of the balloon plot positions before replotting the balloon. If the routine returns a value of 1 then the HIT routine is called.

```
org 23760
CHAPTER 8
PROGRAM 4
POINT x,y

23760 3E 02      ld a,2
23762 CD 01 16  call 5633
23765 CD 6B 0D  call 3435
23768 3E 02      ld a,2

23770 CD 01 16  call 5633
23773 3E 8F      ld a,143
23775 D7         rst 16
23776 0E 00      ld c,0
23778 06 AF      ld b,175
23780 CD CE 22  call 8910
23783 CD D5 2D  call 11733
23786 FE 01      cp 1
23788 CB         ret z
```

Program 8.4

9 THE PRINTER

Not many programs make use of the printer but it can be handy sometimes to have a copy of the screen (to prove to friends that you have achieved the highest scores etc.). So with this in mind we will look at the three BASIC commands COPY, LPRINT, and LLIST and their machine code equivalents.

COPY

Normally the COPY routine will pass the contents of the first 22 lines of the screen directly to the printer. In machine code however we can improve on this and specify how many screen lines we want, and the start line, so for instance we could copy just three lines starting from line seven.

Program 9.1 demonstrates the method used to copy all 24 lines since some games programs have the score held in the bottom two screen lines and will not be printed with a direct COPY command. You can see that before call 3762 the HL register pair holds the line start address, in our example – this is line 1, and the B register holds the number of lines required $\times 8$ ($24 \times 8 = 192$).

```
.org 23760  
CHAPTER 9  
PROGRAM 1
```

```
COPY n SCREEN LINES
```

```
HL=SCREEN LINE START  
B=LINES  $\times$  8
```

Program 9.1

```
EXAMPLE 24 LINES
```

```
23760 21 00 40      ld hl,16384  
23763 06 C0        ld b,192  
23765 F3           di  
23766 CD B2 0E     call 3762  
23769 C9           ret
```

LPRINT

This instruction can be considered the same as PRINT with the only difference being that control characters will not work, of course, and that a different channel requires to be opened before the RST 16 instruction to send the output to the printer instead of to the screen. So to LPRINT "A", for example, first OPEN CHANNEL 3, LD A, CODE "A" then use RST 16 to LPRINT it.

Program 9.2 shows the above method used to LPRINT a string of

characters. Should you wish to tabulate your LPRINTing, the TAB number should be POKEd into systems variable PRCC 23680 before the RST 16 instruction.

```

org 23760                23765 11 DF 5C      1d de,DATA
CHAPTER 9                23768 01 11 00      1d bc,17
PROGRAM 2

LPRINT STRING

OPEN CHANNEL 3          23771 CD 3C 20      call 8252
                        23774 C9          ret
                        DATA
                        defs This is program 2

23760 3E 03              1d a,3
23762 CD 01 16          call 5633

```

Program 9.2

LLIST

Although I cannot think of any application in which a listing would be required in the middle of a program, I have included this because it is one of the printer commands and very easy to program. All that is needed is call 6133. Program 9.3 demonstrates this by assembling the BASIC program and then LLISTing it.

```

org 23760                30>REM org 23760
CHAPTER 9                40 REM !CHAPTER 9;!PROGRAM 3;!
PROGRAM 3                50 REM ! LLIST ;!
LLIST                    60 REM call 6133
                        70 REM ret
23760 CD F5 17           call 6133
23763 C9                 ret
    10 REM STEP PRINT
    20 REM go
                        9000 REM finish
                        9010 RANDOMIZE USR 58000
                        9020 COPY
                        9030 LPRINT
                        9040 RANDOMIZE USR 23760

```

Program 9.3

10 PROGRAM CONVERSION

If you have managed to reach this Chapter with a clear understanding of all the previous ones then congratulations – this one should be no problem. If you are not too sure about any of the previous routines then I would advise you to reread the particular Chapter before proceeding.

In this Chapter I hope to demonstrate the method of producing a games program completely in machine code using a 'working' BASIC program as a guide.

No doubt you have your own method of producing a BASIC program. Perhaps you have an idea and develop it actually on the Spectrum adding routines as you think of them and ending with a completely unstructured, but working, program that only you can follow. This method is satisfactory but you may find that, if you try to amend the program a few weeks later, not even *you* can follow it. I find that the best method is to produce a flow diagram from the basic idea and then to add routines as necessary ending with a complete structured flow diagram from which the BASIC program can be written and perhaps refined with colour, sound, and UDGs. More importantly, however, this gives a program that is easy to convert to machine code.

The program we shall convert in this Chapter is called 'CROSS' and started out as a program to control a man past obstacles to a safe home. From this idea the flow diagram was produced. This involves the main routine to set up the variables and print instructions and the display and then a loop to check on the movement of the man and his 'new' screen position. There are two points of exit from this main loop:

1. If the man is hit we enter the HIT routine and check lives; if there are no lives left the game ends, otherwise we re-enter the main loop.
2. The HOME routine which also checks the number of homes filled and adds spiders or increases the speed before returning to the main loop.

From this flow diagram the BASIC version of 'CROSS' was produced. If you compare the actual program with the flow diagram you should be able to spot the various routines.

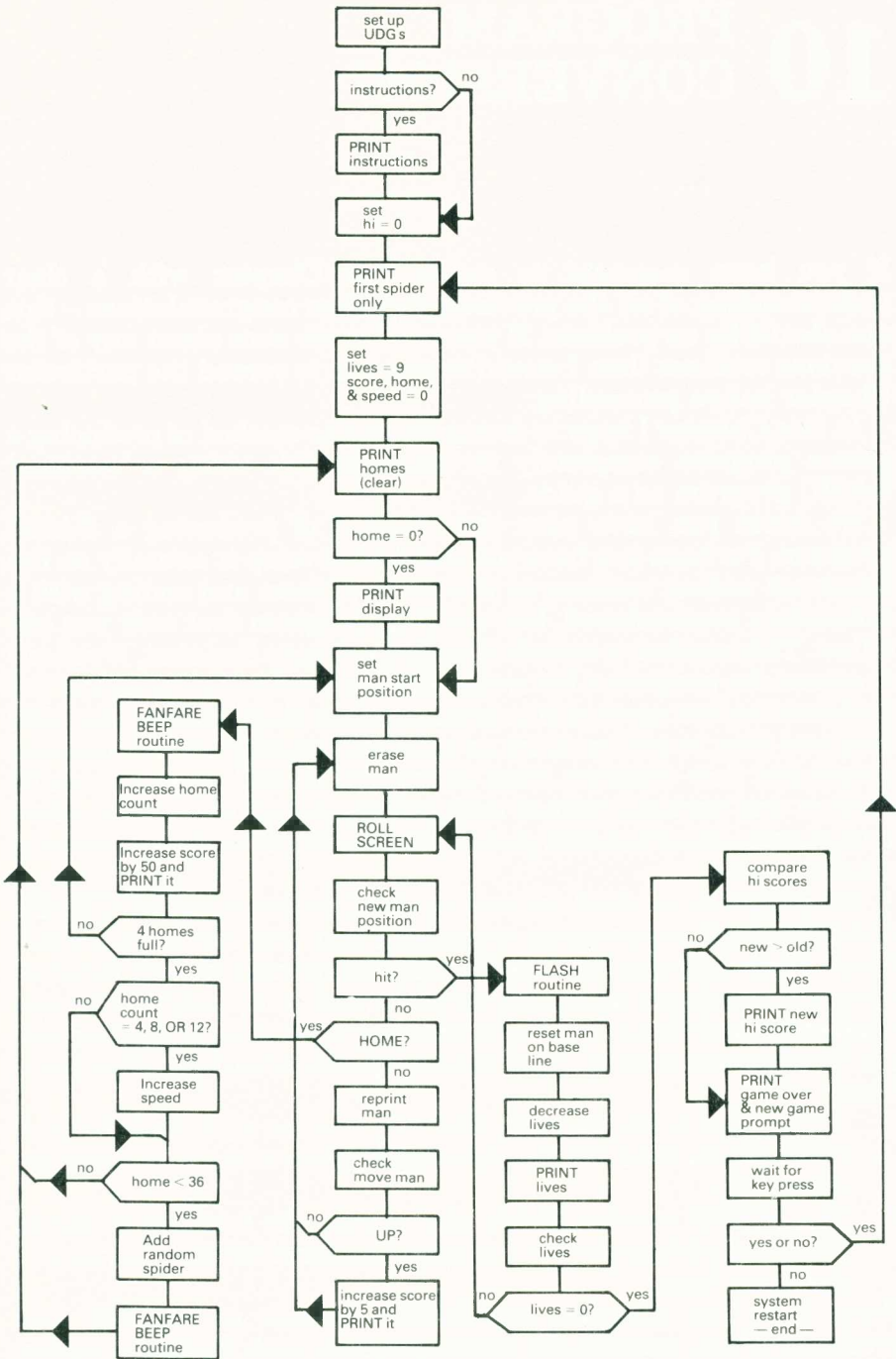


Figure 10.1 Flow diagram for 'Cross' program

The program, although very simple, is greatly improved by making full use of the UDGs, colours, and sound. It does cheat very slightly by using a machine code routine to roll screen lines left or right one pixel at a time, but a BASIC version of this would have been incredibly slow.

The machine code is POKEd into the correct memory addresses by the BASIC program. This could have been carried out using a LOAD 'machine' code XX, n instruction to load the code directly from tape.

Before continuing with this Chapter make sure that you completely understand the program workings as we will now work through it line by line converting it to machine code.

```

20 CLEAR 32243: GO TO 40
25 BEEP .01,b-a
30 PRINT OVER 1: PAPER 8: INK
8: AT a,02;"*": RETURN
40 PRINT AT 11,5:"Please wait
a moment"
50 FOR a=32244 TO 32494
60 READ b: POKE a,b: NEXT a
70 DATA 14,8,229,17,31,0,25,12
6,237,82,31,6,32,126,31,119,35,1
6,250,225,36,13,32,234,201
80 DATA 14,8,175,229,17,31,0,2
37,82,126,25,23,6,32,126,23,119,
43,16,250,225,36,13,32,233,201
90 DATA 33,95,64,205,13,126,33
,128,64,205,244,125,33,128,64,20
5,244,125,33,223,64,205,13,126,3
3,0,72,205,244,125,33,0,72,205,2
44,125,33,0,72,205,244,125
100 DATA 58,121,92,0,0,0,0,0,0,
0,230,2,40,20,33,64,72,205,244,1
25,33,64,72,205,244,125,33,64,72
,205,244,125,24,18,33,95,72,205,
13,126,33,95,72,205,13,126,33,95
,72,205,13,126
110 DATA 33,128,72,205,244,125,
33,192,72,205,244,125,33,192,72,
205,244,125,33,31,80,205,13,126,
33,31,80,205,13,126,33,95,80,205
,13,126,201
120 DATA 33,128,72,205,244,125,
33,192,72,205,244,125,33,31,80,2
05,13,126,33,95,80,205,13,126,20
1
130 DATA 33,95,64,205,13,126,33
,128,64,205,244,125,33,0,72,205,
244,125,201
140 DATA 33,95,64,205,13,126,33
,223,64,205,13,126,33,128,72,205
,244,125,33,192,72,205,244,125,2
01
150 LET a=PEEK 23675+256*PEEK 2
3675
160 FOR b=a TO a+167
170 READ c: POKE b,c: NEXT b
180 DATA 15,18,34,127,255,255,4
0,16,128,64,32,254,254,255,40,16
190 DATA 127,127,127,127,127,25
5,21,8,254,254,254,254,255,255,6
4,126
200 DATA 0,248,196,196,254,254,
40,16,24,24,36,126,60,90,165,66
210 DATA 56,40,146,124,56,56,40
,108,1,2,4,127,127,255,20,8
220 DATA 240,72,68,254,255,255,
20,8,0,31,35,35,127,127,20,8

```



```

570 PRINT PAPER 0; INK 7;"-----
580 PRINT PAPER 0; INK 5;"-----
590 PRINT PAPER 0; INK 7;"=====
600 PRINT PAPER 0; INK 4;"-----
610 PRINT PAPER 0; INK 7;"-----
620 PRINT PAPER 0; INK 6;"-----
630 PRINT PAPER 4;"#####
640 PRINT PAPER 4;"#####

650 PRINT PAPER 1; INK 7;" SCOR
E ";AT 21,11;" MEN "; PAPER 5; I
NK 0; lives; PAPER 1; INK 7;" HI-
SCORE "
660 LET x1=20: LET y1=16: LET x
2=x1: LET y2=y1
670 PRINT PAPER 8; INK 8;AT x1,
y1;" "
680 RANDOMIZE USR 32295
690 IF SCREEN$(x2,y2)=" " THEN
GO TO 680
700 LET a=x2: FOR b=25 TO 35: G
O SUB 25: GO SUB 25: NEXT b
730 FOR a=x2 TO 20 STEP 2: GO S
UB 25: GO SUB 25: NEXT a
740 LET lives=lives-1: PRINT AT
21,16;lives
750 LET x2=20
760 IF lives<>0 THEN GO TO 680
770 IF hi>score THEN GO TO 790
780 LET hi=score: PRINT AT 21,2
7;hi
790 PRINT FLASH 1; PAPER 7;AT 1
2,0;" " GAME OVER

800 PRINT AT 14,0;" Another ga
me ? (y)es (n)o "
840 IF INKEY$="n" THEN RANDOMIZ
E USR 0
850 IF INKEY$<>"y" THEN GO TO 8
40
860 PRINT PAPER 5;AT 21,7;"
" GO TO 415
880 IF x2<>0 THEN GO TO 1050
890 PRINT PAPER 8; INK 8;AT x1,
y1;" " PRINT AT x2,y2;" "
1020 IF SCREEN$(10,a+1)=" " THEN
900 RESTORE 920
910 FOR a=1 TO 8: READ b,c: BEE
P b,c: NEXT a
920 DATA .1,11,.1,11,.8,16,.05,
11,.05,16,.05,11,.05,16,1,20
930 LET home=home+1: LET score=
score+50: PRINT AT 21,7;score
950 IF home/4<>INT (home/4) THE
N GO TO 660
960 IF home=4 THEN POKE 32425,0
970 IF home=8 THEN POKE 32450,0
980 IF home=12 THEN POKE 32459,
0
985 IF home>36 THEN GO TO 450
990 LET a=RD*30
1000 LET a=a+1
1005 IF a>31 THEN LET a=0
1010 IF SCREEN$(10,a)=" " THEN G
O TO 1000
1020 IF SCREEN$(10,a+1)=" " THEN
GO TO 1000
1030 PRINT PAPER 4;AT 10,a;" "
1035 RESTORE 920: FOR a=1 TO 8:
READ b,c: BEEP b,c: NEXT a

```


```

1040 GO TO 450
1050 PRINT PAPER 8; INK 8; AT x2,
y2;"*"
1060 LET x1=x2: LET y1=y2
1070 IF INKEY$("<>"1) THEN GO TO 1
1080 BEEP .001,33
1090 LET x2=x2-2: LET score=score+5: PRINT AT 21,7;score
1100 LET y2=y2+(INKEY$="0" AND y2("<>"31)-(INKEY$="9" AND y2("<>"0)
1110 GO TO 670

```

USER DEFINABLE GRAPHICS :

```

AB = 
CDE = 
F = 
G = 
HI = 
JKL = 
M = 
N = 
OP = 
QRS = 
TU = 

```

Program 'Cross basic'

```

org 32244
CHAPTER 10
PROGRAM >CROSS<

```

[BASIC] M/CODE

RIGHT LINE ROLL

```

RIGHT
32244 0E 08      ld c,8
L1
32246 E5        push hl
32247 11 1F 00  ld de,31
32250 19        add hl,de
32251 7E        ld a,(hl)
32252 ED 52    sbc hl,de
32254 1F        rra
32255 06 20    ld b,32
L2
32257 7E        ld a,(hl)
32258 1F        rra
32259 77        ld (hl),a
32260 23        inc hl
32261 10 FA    djnz L2
32263 E1        pop hl
32264 24        inc h
32265 0D        dec c
32266 20 EA    jr nz,L1
32268 C9        ret

```

LEFT LINE ROLL

```

LEFT
32269 0E 08    ld c,8
L3
32271 AF        xor a
32272 E5        push hl
32273 11 1F 00  ld de,31
32276 ED 52    sbc hl,de
32278 7E        ld a,(hl)
32279 19        add hl,de
32280 17        rla
32281 06 20    ld b,32
L4
32283 7E        ld a,(hl)
32284 17        rla
32285 77        ld (hl),a
32286 2B        dec hl
32287 10 FA    djnz L4
32289 E1        pop hl
32290 24        inc h
32291 0D        dec c
32292 20 E9    jr nz,L3
32294 C9        ret

```

START ROUTINE

```

HL holds :-
Line start for RIGHT ROLL
Line end for LEFT ROLL

```

SPEED 0			32392 CD F4 7D	call RIGHT
32295 21 5F 40	ld hl,16479		32395 21 C0 48	ld hl,18624
32298 CD 0D 7E	call LEFT		32398 CD F4 7D	call RIGHT
32301 21 80 40	ld hl,16512		32401 21 C0 48	ld hl,18624
32304 CD F4 7D	call RIGHT		32404 CD F4 7D	call RIGHT
32307 21 80 40	ld hl,16512		32407 21 1F 50	ld hl,20511
32310 CD F4 7D	call RIGHT		32410 CD 0D 7E	call LEFT
32313 21 DF 40	ld hl,16607		32413 21 1F 50	ld hl,20511
32316 CD 0D 7E	call LEFT		32416 CD 0D 7E	call LEFT
32319 21 00 48	ld hl,18432		32419 21 5F 50	ld hl,20575
32322 CD F4 7D	call RIGHT		32422 CD 0D 7E	call LEFT
32325 21 00 48	ld hl,18432		32425 C9	ret
32328 CD F4 7D	call RIGHT	SPEED 1		
32331 21 00 48	ld hl,18432			
32334 CD F4 7D	call RIGHT	32426 21 80 48	ld hl,18560	
Move SPIDER line				
Uses FRAMES counter				
			32429 CD F4 7D	call RIGHT
			32432 21 C0 48	ld hl,18624
			32435 CD F4 7D	call RIGHT
			32438 21 1F 50	ld hl,20511
32337 3A 79 5C	ld a,(23673)		32441 CD 0D 7E	call LEFT
32340 00	nop		32444 21 5F 50	ld hl,20575
32341 00	nop		32447 CD 0D 7E	call LEFT
32342 00	nop		32450 C9	ret
32343 00	nop	SPEED 2		
32344 00	nop			
32345 00	nop			
32346 00	nop	32451 21 5F 40	ld hl,16479	
32347 E6 02	and 2	32454 CD 0D 7E	call LEFT	
32349 28 14	jr z,L5	32457 21 80 40	ld hl,16512	
32351 21 40 48	ld hl,18496	32460 CD F4 7D	call RIGHT	
32354 CD F4 7D	call RIGHT	32463 21 00 48	ld hl,18432	
32357 21 40 48	ld hl,18496	32466 CD F4 7D	call RIGHT	
32360 CD F4 7D	call RIGHT	32469 C9	ret	
32363 21 40 48	ld hl,18496	SPEED 3		
32366 CD F4 7D	call RIGHT			
32369 18 12	jr L6			
L5				
32371 21 5F 48	ld hl,18527			
32374 CD 0D 7E	call LEFT	32470 21 5F 40	ld hl,16479	
32377 21 5F 48	ld hl,18527	32473 CD 0D 7E	call LEFT	
		32476 21 DF 40	ld hl,16607	
		32479 CD 0D 7E	call LEFT	
32380 CD 0D 7E	call LEFT	32482 21 80 48	ld hl,18560	
32383 21 5F 48	ld hl,18527	32485 CD F4 7D	call RIGHT	
32386 CD 0D 7E	call LEFT	32488 21 C0 48	ld hl,18624	
L6		32491 CD F4 7D	call RIGHT	
32389 21 80 48	ld hl,18560	32494 C9	ret	

Program 'Machine Code for Cross basic'

Machine code conversion

BASIC LINE 20

This will not be necessary in the final machine code version because there will be no need to reset RAMTOP to protect the machine code.

BASIC LINES 25 AND 30

This is a subroutine used in the HIT routine and in machine code this will be placed with the HIT routine. The subroutine is placed at the start of the BASIC program to reduce the time that the Spectrum takes to find it. When a subroutine is called the Spectrum starts from the beginning of the BASIC program and works through the line numbers until the required routine is located. You can see that it would take longer to find if it had a high line number thus increasing the time taken to carry out lines 700 to 730 which call this routine at least 22 times.

BASIC LINE 40

In BASIC it takes several seconds for the Spectrum to 'LOAD' the machine code and set up the UDGs so this message is displayed during that period. In machine code, however, this is instantaneous and the message is therefore unnecessary.

BASIC LINES 50 TO 140

This is the routine to POKE the machine code 'ROLL ROUTINE' into memory addresses 32244 to 32494, and again is not required in machine code as it will automatically be correctly located along with the rest of the machine code program.

BASIC LINES 150 TO 270 SET UP UDGs

This is the first routine that will need converting. We could of course use SAVE 'UDG' CODE USR 'a', 168 : LOAD 'UDG' CODE USR 'a', 168 but this would mean saving and loading two sets of machine code routine, namely the main game and the UDGs. In order that the machine code will easily fit into the 16K Spectrum we will begin the machine code program at address 5DC1h (24001d). As with the BASIC we will hold each byte of the UDGs in DATA form and read through this DATA transferring each number to its correct place in the UDG area of memory. The simplest method as you will see is to use the LDIR instruction. The DE register pair is loaded with the address of UDG start as held in the system variable 5C7Bh (23675d) and the HL register pair holds the address of the start of DATA. Routine DATA = 5DC1h to 5E68h and machine code routine = 5E69h (24169d) to 5E73h (24180d). Note that the routine is called using RANDOMIZE USR 24169 *not* USR 24001 which is DATA start.

BASIC LINE 280

This is a simple printing statement and is easily converted to machine code using the print DATA ROM call (203Ch – 8252d) as discussed in Chapter 1. So DATA = 5E77h (24183d) to 5EA1h (24225d). This is jumped over by the instruction 'JR 5EA2h' at memory address 5E75/6h. The print DATA routine = 5EA2h(24226d) to 5EAFh.

BASIC LINE 290 AND 295 MACHINE CODE 5EB0 TO 5EBE

In this routine we wait until a new key press is registered and check the key for 'y' (the HALT command is not absolutely necessary). If the key pressed is not 'y' then we skip the instructions by jumping (NZ) to 608Ah (24714d).

BASIC LINE 300: 1 CLS MACHINE CODE 5EBF TO 5EC6

This routine is straightforward and is discussed in Chapter 1.

BASIC LINES 300:2 TO 330

This again is carried out using DATA store and PRINT string routines. The first set of DATA is held at 5ECA to 5F72 with the PRINT DATA routine at 5F73 to 5F83. The second set of DATA is held from 5F84 to 5FF9 and the PRINT DATA routine at 5FFA to 6002. The DATA could have been lumped together with just one PRINT DATA routine.

The PAUSE 0 routine is 6003 to 600C; note that bit 5, (IY + 1) has to be reset from the previous PAUSE 0 routine. Try removing the RES 5,(IY + 1) and 'see' the result.

BASIC LINE 370:1 CLS ROUTINE 600E TO 6012

This uses the alternative methods of clearing the whole screen by POKEing zeros into all screen addresses (not affecting the ATTRIBUTES).

BASIC LINES 370:2 TO 390

DATA for print routine = 6016 to 6075

PRINT data routine = 6076 to 607E

This is followed again by the PAUSE 0 routine.

BASIC LINE 400

Using the set BORDER colour routine and set permanent attribute

colour as in Chapter 1, we can convert this line as routine 608A to 609D.

The CLS call puts the colours on the screen as in BASIC.

BASIC LINE 410

This line sets the initial value of the high score. We will assume that the value will not be greater than 65 535 and as such use the first two bytes of the printer buffer to hold its value (address 5B00/01 – 23296/7). This is easily set to zero by LD HL, 0000h and LD(5B00),HL and is shown in routine 609C to 60A1.

BASIC LINE 420

DATA for printing = 60A4 to 60C8
PRINT DATA routine = 60C9 to 60D6

BASIC LINE 430

This sets up the initial values of 'lives', 'score', and 'home'. The lives can be held as one byte, the score as two bytes, and the home as one byte. We will again use the printer buffer to hold these values. Address 5B02 holds 'lives', 5B03/4 holds 'score', and 5B05 holds 'home'. These initial values are stored using routine 60D7 to 60E5.

BASIC LINE 440

This line resets the speed to zero by replacing the RET instructions at the correct points in the LINE ROLL subroutine. Routine: 60E6 to 60F0.

BASIC LINE 450

DATA for printing = 60F4 to 6128
PRINT DATA routine = 6129 to 6136

BASIC LINE 455

A check is made to see whether home <> 0, that is whether the screen display has to be printed or left as it is just clearing the homes. We need to fetch the value stored in 'home' 5B05 and check that it is not zero using 'AND A'. If this does not set the zero flag we jump to the PRINT display routine. If zero flag is set we skip the PRINT display routine and jump straight into the game proper. Routine: 6137 to 6140.

BASIC LINES 460 TO 640

This is the longest routine in the program and demonstrates just how tedious printing a display actually is.

DATA for printing 6141 to 63D1

PRINT DATA routine 63D2 to 63DF

BASIC LINE 650

You will notice that this PRINT routine contains the variable 'lives', so before we can use our PRINT data routine, the correct value of 'lives' must be placed into the DATA for printing. This is achieved by using the routine 63E0 to 63E9 which just collects the value 'lives' from 5B02 then adds 30h (48d) to convert it to the correct ASCII value and finally 'POKEs' it into DATA, address 6401.

DATA for printing 63EA to 640F

PRINT DATA routine 6410 to 641D

BASIC LINE 660

This line sets the initial values of the man screen print position x1, y1 and the 'new' position used in the SCREEN\$ routine x2, y2. Again we will use the printer buffer to hold these values

x1 = 5B06

y1 = 5B07

x2 = 5B08

y2 = 5B09

This is easily carried out in machine code by LD HL,1014h: LD(5B06), HL : LD(5B08),HL. You will notice from the above how much easier it is to use hexadecimal numbers and not decimal ones. It is quite clear that with LD HL, 1014h that H will hold 10h (16d) and L will hold 14 (20d). Routine 641E to 6426.

BASIC LINE 670

As we are printing just one character whose position is dependent on the variables x1, y1, it is easier to use LD A,n : RST 10h than a PRINT DATA routine. Routine 6427 to 6445.

BASIC LINE 680

Routine 6446 to 6448. We will leave the ROLL routine at address (32295) 7E27.

BASIC LINE 690

We will use the routine described in Chapter 8 to find SCREEN\$(x2, y2) then unstack the 'value' into the A,B,C,D,E, registers. The character can then be found by LD A,(DE) and compared to 20h (32). Remember that we must clear the workspace before continuing with the program. Routine 6449 to 645D. If the character is not a space then we jump to the HIT routine that starts at address 6548h. The BASIC main loop jumps to line 880.

BASIC LINE 880

Here we check that the man is not on the top screen line, that is, 'home'. In machine code we get the value x2 by LD A,(5B08) and compare it to 0. If it is zero we jump to the home routine at address 66A6. Routine 645E to 6465. The BASIC main loop jumps to line 1050.

BASIC LINE 1050

Because we are printing one character using two variables x2, y2, we will use LD A, n : RST 10h (as line 670). Routine 6466 to 6484.

BASIC LINE 1060

This is a case of collecting the value held in 5B08/9 and placing it in 5B06/7. Routine 6485 to 648A.

BASIC LINE 1070

At this point we will change the BASIC to improve the game. Instead of using the LAST KEY variable to read the keyboard, we will read it directly using the IN A,(C) command and so allow all keys to be read and diagonal moves made.

To read key '1' we need to read half row 1 to 5 and as described in Chapter 5 the routine required is 648B to 6493.

BASIC LINE 1080

Because the BEEP is not important in its length and pitch we need not be precise in the machine code conversion. We can use the BEEP routine call 03B5(949d) instead of using the calculator. Routine 6494 to 649C.

BASIC LINE 1090

Here we are updating the line position (x2) of the man so we collect the value in 5B08, decrease it by two, and put the new value back

into 5B08. The score is increased by collecting the value in 5B03/4, adding five and putting the result back into 5B03/4. Routine 649D to 64AF. To print the score we will use the STACK BC and PRINT VALUE ON STACK routines, but first we define the 'PRINT AT' parameters. Routine 64B1 to 64C8.

BASIC LINE 1100

Again using the IN A,(C) instruction we can read the keyboard and update the value of y 2 (5B09). Routine 64C9 to 64F1. I have added a routine to check for key '6' being pressed to give a return to BASIC facility at this point. (This is necessary to 'SAVE' the program once it is running properly.) Routine 64F2 to 64F4.

This completes the main loop but you will find that, if this program is run, it will be so fast as to make the man almost invisible as he flashes on and off between screen rolls. Because of this we need our delay loop to hold the man for a moment as suggested in Chapter 6. Instead of our normal loop we can add a traffic noise sound effect as routine 64F5 to 650B. The final JP 6427 will take us back to the beginning of the main loop (line 670).

Hit routine

BASIC LINE 700

Here we introduce another variable $a = x2$. We will use address 5CB0 to hold this value (an unused address in the systems variables). The FOR/NEXT loop variable 'b' can be replaced by LD B, 19h(25d). This value is then placed in address 5CB1 to be used in the next routine. Routine 6548 to 6561. The GOSUB 25 call is replaced by call 650F.

BASIC LINE 25

This is easily converted using the calculator to stack .01 . b - a is simply LD HL,(5CB0) : LD A,H: SUBL; the values a and b being stored consecutively at 5CB0/B1. We then use the stack value in 'A' routine 2D28 and call BEEP routine 03F8. Routine 650F to 6521.

BASIC LINE 30

This is another case of using LD A,n : RST 10h as there are two variables in the PRINT routine. Routine 6522 to 6547.

BASIC LINE 730

Variable a is already stored at 5CB0, so we can set up a loop taking

this value and increasing it by two after each pass until the value reaches 14h(20d). Routine 6562 to 6573.

BASIC LINE 740

Here we get the value of 'lives' from address 5B02, reduce it by one, and return the new value to 5B02. To print this value we just set the PRINT AT parameters then get the value, add 30h(48d) to convert it to ASCII code, and print it. Routine 6574 to 658E.

BASIC LINE 750

Routine 658F to 6593. This involves resetting value x2(5B08) to 14h(20d).

BASIC LINE 760

Routine 6594 to 659A. This is quite straightforward. If 'lives' not zero we jump to 6446, that is, back into the main loop.

End game routine

BASIC LINE 770

This involves getting the value of hi score and score and checking which is greater. It is quite easily done by using SBC HL, DE and then checking the state of the carry flag. Routine 659B to 65A. If hi score is greater than score, we jump to 6611.

BASIC LINE 780

This involves transferring score 5B03/4 to hi score 5B0/01. Routine 65A7 to 65AC. To print the hi score we use the STACK VALUE IN BC and PRINT TOP VALUE ON CALCULATOR STACK routines having first set the PRINT AT parameters. Routine 65AD to 66C2.

BASIC LINES 790 TO 800

DATA for printing = 65C5 to 6610
PRINT DATA routine = 6611 to 6619

BASIC LINE 840 TO 850

Here we use LAST KEY variable to wait for a key press and take action accordingly. Routine 661A to 6632. If the key pressed is not 'y' (79h) or 'n' (6Eh) then we jump back into the WAIT loop.

BASIC LINE 860

This is carried out using LD A,n : RST 10h followed by JP 60C9 to return to the main loop reprinting the display but leaving the 'hi score' unaltered. Routine 6633 to 6655.

Home routine

BASIC LINE 890

Here we use the PRINT DATA routine but first putting the required values of x1, y1 and x2, y2 into the DATA.

DATA for PRINTING 669A to 66A5

SET value of x1,y1,x2,y2 66A6 to 66B1

PRINT DATA routine 66B2 to 66BF

BASIC LINE 900

This of course is unnecessary in machine code as the DATA start does not need to be reset.

BASIC LINES 910 TO 920

FANFARE BEEP call 666A

The routine is covered in detail in Chapter 7 and so needs no further comment. Routine 66C0 to 66C2.

BASIC LINES 930

Another set of routines to collect current values of 'home' and 'score' and update each one followed by a routine to print the score using the STACK VALUE IN BC and PRINT TOP VALUE ON STACK routines. Update routine 66C3 to 66D3. Print score routine 66D4 to 66E9.

BASIC LINE 950

This is a check to see whether all four homes are full and is easier to program in machine code than in BASIC. Routine 66EA to 66F1. You can see that we first LD A, (5B05) that is, 'home' count, then mask it with AND 03h. If we are left with a non-zero answer it will show that the original value of 'home' was not a multiple of four. If the result is not zero we jump back into the main loop JP 641E.

BASIC LINE 960 TO 980

These routines are used to increase the speed of the game when 'home' = 4,8, and 12. This is done by removing the RETURN

instructions in the ROLL subroutine allowing certain lines to be moved by an extra pixel: line 960 = 66F2 to 66FF, line 970 = 6700 to 6709, and line 980 = 670A to 6712.

BASIC LINE 985

We now check that 'home' is not greater than 36. If it is then the 'ADD EXTRA SPIDER' routine is missed and we return to the main loop. We simply subtract 37 from the value of 'home' and if the carry flag is not set (that is home=37) then we JP NC, 6129. Routine 6714 to 6718.

BASIC LINE 990

This routine is discussed in Chapter 4 and needs no further explanation except to point out that in this case we use the modified SEED value in BC and mask 1Fh (31d) : LD A,C : AND 1Fh then subtract two and check for the carry flag set. This method was chosen because a true random number was not necessary. Routine 6719 to 6740.

BASIC LINE 1000

Routine 6741

All that is required is to increment the value in 'a'.

BASIC LINE 1005

A check is made that the value of 'a' is not greater than 31, that is, off the screen; if it is then 'a' is reset to 0. Routine 6742 to 6746.

BASIC LINE 1010

A check is made that the screen position (10,a) is clear using the SCREEN\$ routine. This entails placing the value 'a' into the B register and 0ah (10d) into the C register, calling STACK VALUE IN BC and SCREEN\$ routines then removing the parameters from the stack into A,B,C,D,E registers and finally checking for SCREEN\$ (10,a) = 20h(32d). If all is satisfactory then we get the original value of 'a' (POP AF) and continue. If the SCREEN\$ routine indicates that the position is not clear then we JR 6741 thus stepping to the next screen position. Routine 6747 to 675E.

BASIC LINE 1020

This is a repeat of the previous routine except that the value of 'a' is

incremented before the SCREEN\$ routine. Routine 675F to 6776. Note that the value of 'a' is saved before the SCREEN\$ routine (6760 PUSH AF) to be used in the next routine.

BASIC LINE 1030

The value 'a' is POPped back into AF and decreased. This is then placed into the PRINT DATA, and the PRINT routine called. Routine 6777 to 6792.

BASIC LINE 1030

This is a repeat of the FANFARE BEEP and just requires CALL 666A. Routine 6793 to 6795.

BASIC LINE 1040

Finally we jump back into the main loop using JP 6129. Routine 6796 to 6798.

You can see from the above that it really is very simple to convert a BASIC program to machine code *as long as you take it one step at a time* and plan your program with machine code in mind.

One last point that you will no doubt have noticed is that it is better to work in hexadecimal rather than decimal as for instance many of the routines require the loading of BC with line/column parameters. Take another look at the LEFT/RIGHT ROLL subroutine. The decimal screen addresses mean very little, with no apparent pattern, but if you look at the hexadecimal equivalents you will see some sort of pattern giving a clearer indication of what is actually happening.

All that is necessary now is for you to dream up your own brilliant and original program and start converting it to machine code. I wish you good luck and hope you enjoy many happy hours of programming with not too many crashes.

```

org 42000 24001                                defb $6c
                                                defb $01 $02 $04 $7f $7f $ff $14
                                                defb $08
                                                defb $f0 $48 $44 $fe $ff $ff $14

SDC1h
defb $0f $12 $22 $7f $ff $ff $28              defb $08
defb $10                                       defb $00 $1f $23 $23 $7f $7f $14
defb $80 $40 $20 $fe $fe $ff $28             defb $08
defb $10                                       defb $7f $7f $7f $7f $ff $ff $02
defb $7f $7f $7f $7f $7f $ff $15           defb $01
defb $08                                       defb $fe $fe $fe $fe $fe $ff $a8
defb $ff $fe $fe $fe $ff $ff $40           defb $10
defb $80                                       defb $10 $29 $c7 $00 $26 $00 $00
defb $00 $f8 $c4 $c4 $fe $fe $28           defb $00
defb $10                                       defb $00 $44 $ff $44 $44 $ff $44
defb $18 $18 $24 $7e $3c $5a $a5
defb $42
defb $38 $28 $92 $72 $38 $38 $28

```

```

defb $00
defb $00 $22 $55 $8f $97 $a3 $a0
defb $00
defb $00 $44 $aa $f1 $e9 $c5 $05
defb $00
defb $10 $10 $10 $fe $3f $1f $0f
defb $07
defb $00 $00 $00 $00 $1e $ff $ff
defb $ff
defb $60 $7c $54 $78 $7f $ff $fe
defb $7c

```

```

defb $00 $00 $03 $02 $0f $3f $ff
defb $00
defb $06 $0c $98 $f0 $e0 $55 $ff
defb $00

```

5E69h

```

24169 ED 5B 7B 5C ld de,($5c7b)
24173 21 C1 5D ld hl,$5dc1
24176 01 A8 00 ld bc,$00a8
24179 ED B0 ldir
24181 18 2B jr +$2b

```

5E77h

```

defb $16 $0b $03 $44 $6f $20 $79
defb $6f
defb $75 $20 $77 $61 $6e $74 $20
defb $69

```

```

defb $6e $73 $74 $72 $75 $63 $74
defb $69
defb $6f $6e $73 $3f $16 $0d $0b
defb $28
defb $79 $29 $65 $73 $16 $0f $0b
defb $28
defb $6e $29 $6f

```

5EA2h

```

24226 3E 02 ld a,$02
24228 CD 01 16 call $1601
24231 11 77 5E ld de,$5e77
24234 01 2B 00 ld bc,$002b
24237 CD 3C 20 call $203c
24240 76 halt
24241 FD CB 01 6E bit 5,(iy+$01)
24245 28 F9 jr z,-$07
24247 3A 08 5C ld a,($5c08)
24250 FE 79 cp $79

```

```

24252 C2 8A 60 jp nz,$608a

```

5EBFh

```

24255 3E 02 ld a,$02

```

```

24257 CD 01 16 call $1601
24260 CD 6B 0D call $0d6b
24263 C3 73 5F jp $5f73

```

5ECA

```

defb $16 $00 $0b $4f $42 $4a $45
defb $43
defb $54 $0d $0d $54 $6f $20 $67
defb $75
defb $69 $64 $65 $20 $61 $20 $96
defb $20
defb $61 $63 $72 $6f $73 $73 $20
defb $61

```

```

defb $20 $72 $6f $61 $64 $20 $61
defb $6e
defb $64 $20 $61 $72 $69 $76 $65
defb $72
defb $2c $61 $76 $6f $69 $64 $69
defb $6e
defb $67 $20 $90 $91 $20 $92 $93
defb $94
defb $20 $95 $20 $a0 $a1 $a1 $a2
defb $20
defb $a4 $0d $0d $41 $20 $9e
defb $9f
defb $20 $70 $61 $74 $72 $6f $6c
defb $73
defb $20 $74 $68 $65 $20 $63 $65
defb $6e
defb $74 $72 $61 $6c $20 $69 $73
defb $6c
defb $61 $6e $64 $2e $0d $0d $54
defb $68
defb $65 $72 $65 $20 $61 $72 $65

```

```

defb $20
defb $34 $20 $48 $4f $4d $45 $53
defb $20
defb $74 $6f $20 $62 $65 $20 $66
defb $69
defb $6c $6c $65 $64 $2e $20 $69
defb $2e
defb $65 $2e $20 $67 $61 $70 $73
defb $20
defb $69 $6e $20 $66 $65 $6e $63
defb $65
defb $20 $9d $9d $9d $20 $9d $9d
defb $9d $0d

```

5F73h

```

24435 3E 02 ld a,$02
24437 CD 01 16 call $1601
24440 11 CA 5E ld de,$5eca
24443 01 A9 00 ld bc,$00a9

```

```

24446 CD 3C 20 call $203c

```

24449 C3 FA 5F jp \$5ffa defb \$20
 defb \$20 \$20 \$20 \$20 \$20 \$20 \$20
 defb \$20

5F84h

defb \$0d \$4f \$6e \$63 \$65 \$20 \$61
 defb \$6c
 defb \$6c \$20 \$34 \$20 \$48 \$4f \$4d
 defb \$45
 defb \$53 \$20 \$61 \$72 \$65 \$20 \$66
 defb \$69
 defb \$6c \$6c \$65 \$64 \$20 \$74 \$68
 defb \$65
 defb \$20 \$73 \$70 \$65 \$65 \$64 \$20
 defb \$77
 defb \$69 \$6c \$6c \$20 \$69 \$6e \$63
 defb \$72
 defb \$65 \$61 \$73 \$65 \$2c \$20 \$61
 defb \$6e
 defb \$20 \$65 \$78 \$74 \$72 \$61 \$20

defb \$20 \$20 \$20 \$3c \$30 \$3e \$12
 defb \$01
 defb \$16 \$0b \$06 \$31 \$12 \$00 \$20
 defb \$32
 defb \$20 \$33 \$20 \$34 \$20 \$35 \$20
 defb \$36
 defb \$20 \$37 \$20 \$38 \$30 \$12 \$01
 defb \$39
 defb \$12 \$00 \$20 \$12 \$01 \$30 \$12
 defb \$00
 defb \$16 \$12 \$05 \$50 \$72 \$65 \$73
 defb \$73
 defb \$20 \$61 \$6e \$79 \$20 \$6b \$65
 defb \$79
 defb \$20 \$74 \$6f \$20 \$50 \$4c \$41
 defb \$59

defb \$65
 defb \$20 \$61 \$64 \$64 \$65 \$64 \$20
 defb \$61
 defb \$6e \$64 \$20 \$74 \$68 \$65 \$20
 defb \$48
 defb \$4f \$4d \$45 \$53 \$20 \$77 \$69
 defb \$6c
 defb \$6c \$65 \$6d \$70 \$74 \$79 \$16
 defb \$12
 defb \$09 \$50 \$72 \$65 \$73 \$73 \$20
 defb \$61
 defb \$6e \$79 \$20 \$6b \$65 \$79

6076h

24694 11 16 60 ld de,\$6016
 24697 01 60 00 ld bc,\$0060
 24700 CD 3C 20 call \$203c
 24703 FD CB 01 AE res 5,(iy+\$01
 24707 76 halt
 24708 FD CB 01 6E bit 5,(iy+\$01
 24712 28 F9 jr z,-\$07

5FFAh

24570 11 84 5F ld de,\$5f84
 24573 01 76 00 ld bc,\$0076
 24576 CD 3C 20 call \$203c
 24579 FD CB 01 AE res 5,(iy+\$01
 24583 76 halt

608Ah

24714 3E 05 ld a,\$05
 24716 CD 9B 22 call \$229b
 24719 3E 68 ld a,\$68
 24721 32 8D 5C ld (\$5c8d),a
 24724 3E 02 ld a,\$02
 24726 CD 01 16 call \$1601
 24729 CD 6B 0D call \$0d6b

24584 FD CB 01 6E bit 5,(iy+\$01
 24588 28 F9 jr z,-\$07

609Ch

600Eh

24590 06 18 ld b,\$18
 24592 CD 44 0E call \$0e44
 24595 C3 76 60 jp \$6076

24732 21 00 00 ld hl,\$0000
 24735 22 00 5B ld (\$5b00),hl
 24738 18 25 jr +\$25

6016h

defb \$16 \$07 \$0b \$43 \$4f \$4e \$54
 defb \$52
 defb \$4f \$4c \$53 \$0d \$0d \$20 \$20
 defb \$20
 defb \$20 \$20 \$20 \$5e \$20 \$20 \$20

60A4h

defb \$11 \$04 \$16 \$0a \$00 \$20 \$20
 defb \$20
 defb \$20 \$20 \$20 \$20 \$20 \$20 \$20
 defb \$20
 defb \$20 \$20 \$20 \$20 \$9e \$9f \$20
 defb \$20


```

defb $20 $20 $20 $20 $20 $20 $20
defb $20
defb $20 $20 $20 $20 $20

```

60C9h

```

24777 3E 02      ld a,$02

```

```

24779 CD 01 16   call $1601
24782 11 A4 60   ld de,$60a4
24785 01 25 00   ld bc,$0025
24788 CD 3C 20   call $203c

```

60D7h

```

24791 3E 09      ld a,$09
24793 32 02 5B   ld ($5b02),a
24796 21 00 00   ld hl,$0000
24799 22 03 5B   ld ($5b03),hl
24802 AF         xor a
24803 32 05 5B   ld ($5b05),a

```

60E6h

```

24806 3E C9      ld a,$c9
24808 32 A9 7E   ld ($7ea9),a
24811 32 C2 7E   ld ($7ec2),a

```

```

24814 32 D5 7E   ld ($7ed5),a
24817 C3 29 61   jp $6129

```

60F4h

```

defb $16 $00 $00 $11 $04 $9d $9d
defb $9d
defb $9d $11 $07 $20 $11 $04 $9d
defb $9d
defb $9d $9d $9d $9d $11 $07 $20
defb $11
defb $04 $9d $9d $9d $9d $9d $9d
defb $9d
defb $11 $07 $20 $11 $04 $9d $9d
defb $9d
defb $9d $9d $9d $9d $11 $07 $20
defb $11
defb $04 $9d $9d $9d $9d

```

6129h

```

24873 3E 02      ld a,$02
24875 CD 01 16   call $1601
24878 11 F4 60   ld de,$60f4
24881 01 35 00   ld bc,$0035
24884 CD 3C 20   call $203c

```

6137h

```

24887 3A 05 5B   ld a,($5b05)
24890 A7         and a
24891 C2 1E 64   jp nz,$641e
24894 C3 D2 63   jp $63d2

```

6141h

```

defb $11 $04 $10 $05 $8c $8c $8c
defb $8c
defb $8c $8c $8c $8c $8c $8c $8c
defb $8c
defb $8c $8c $8c $8c $8c $8c $8c
defb $8c
defb $8c $8c $8c $8c $8c $8c $8c
defb $8c
defb $8c $8c $8c $8c $11 $05 $10
defb $00
defb $20 $a0 $a1 $a1 $a2 $20 $20
defb $20
defb $a0 $a1 $a1 $a1 $a2 $20 $20
defb $20
defb $20 $a0 $a1 $a1 $a2 $20 $20
defb $20
defb $a0 $a1 $a1 $a2 $20 $20 $20
defb $20
defb $10 $07 $20 $20 $20 $9c $9c
defb $9c
defb $20 $20 $20 $20 $20 $9c $9c
defb $9c
defb $9c $9c $20 $20 $20 $20 $20

```

```

defb $9c
defb $9c $9c $20 $20 $20 $20 $20
defb $20
defb $9c $10 $02 $20 $20 $a3 $a4
defb $20
defb $20 $20 $20 $20 $a3 $a4 $20
defb $20
defb $20 $a3 $a4 $20 $20 $20 $20
defb $20
defb $20 $a3 $a4 $20 $10 $07 $9c
defb $9c
defb $20 $20 $20 $9c $20 $20 $20
defb $20
defb $20 $20 $9c $9c $9c $9c $9c
defb $20
defb $20 $20 $20 $20 $20 $20 $20
defb $9c
defb $9c $9c $20 $20 $20 $20 $10
defb $01

```

```

defb $a1 $a2 $20 $20 $20 $20 $20
defb $a0
defb $a1 $a1 $a2 $20 $20 $20 $20
defb $20
defb $20 $a0 $a1 $a1 $a1 $a1 $a2
defb $20
defb $20 $20 $20 $20 $a0 $a1 $a1

```

defb \$a1
 defb \$10 \$07 \$20 \$20 \$20 \$9c \$9c
 defb \$20
 defb \$20 \$9c \$9c \$9c \$9c \$9c \$20
 defb \$20
 defb \$20 \$9c \$9c \$20 \$20 \$20 \$20
 defb \$20
 defb \$20 \$9c \$9c \$9c \$20 \$20 \$20
 defb \$20
 defb \$20 \$20 \$10 \$00 \$a4 \$20 \$20
 defb \$20
 defb \$20 \$20 \$20 \$20 \$20 \$20 \$a3
 defb \$a4
 defb \$20 \$20 \$20 \$20 \$20 \$a3 \$a4
 defb \$20
 defb \$20 \$20 \$20 \$20 \$20 \$a3 \$a4
 defb \$20
 defb \$20 \$20 \$20 \$a3 \$11 \$04 \$9d
 defb \$9d
 defb \$9d \$9d \$9d \$9d \$9d \$9d \$9d
 defb \$9d
 defb \$9d \$9d \$9d \$9d \$9d \$9d \$9d
 defb \$9d
 defb \$9d \$9d \$9d \$9d \$9d \$9d \$9d
 defb \$9d
 defb \$9d \$9d \$9d \$9d \$9d \$9d \$9d
 defb \$9d
 defb \$9d \$9d \$9d \$9d \$9d \$9d \$11
 defb \$00
 defb \$10 \$07 \$16 \$0b \$00 \$9d \$9d
 defb \$9d
 defb \$9d \$9d \$9d \$9d \$9d \$9d \$9d
 defb \$9d
 defb \$9d \$9d \$9d \$9d \$9d \$9d \$9d
 defb \$9d
 defb \$9d \$9d \$9d \$9d \$9d \$9d \$9d
 defb \$9d
 defb \$9d \$9d \$9d \$9d \$9d \$9d \$9d
 defb \$9d
 defb \$9d \$9d \$9d \$9d \$9d \$10 \$03
 defb \$20
 defb \$90 \$91 \$20 \$95 \$20 \$20 \$90
 defb \$91
 defb \$20 \$20 \$92 \$93 \$92 \$93 \$94
 defb \$20
 defb \$95 \$20 \$95 \$20 \$20 \$20 \$92
 defb \$93
 defb \$94 \$20 \$20 \$20 \$20 \$20 \$20
 defb \$10
 defb \$07 \$2d \$2d \$2d \$2d \$2d \$2d
 defb \$2d
 defb \$2d \$2d \$2d \$2d \$2d \$2d \$2d
 defb \$2d
 defb \$2d \$2d \$2d \$2d \$2d \$2d \$2d
 defb \$2d
 defb \$2d \$2d \$2d \$2d \$2d \$2d \$2d
 defb \$2d
 defb \$2d \$10 \$05 \$20 \$90 \$91 \$20
 defb \$20
 defb \$20 \$20 \$90 \$91 \$20 \$20 \$20

defb \$91
 defb \$20 \$20 \$20 \$10 \$07 \$3d \$3d
 defb \$3d
 defb \$3d \$3d \$3d \$3d \$3d \$3d \$3d
 defb \$3d
 defb \$3d \$3d \$3d \$3d \$3d \$3d \$3d
 defb \$3d
 defb \$3d \$3d \$3d \$3d \$3d \$3d \$3d
 defb \$3d
 defb \$3d \$3d \$3d \$3d \$3d \$10 \$04
 defb \$98
 defb \$20 \$20 \$20 \$20 \$20 \$97 \$98
 defb \$20
 defb \$20 \$20 \$97 \$98 \$20 \$20 \$20
 defb \$20
 defb \$20 \$20 \$20 \$97 \$98 \$20 \$20
 defb \$20
 defb \$20 \$20 \$20 \$20 \$20 \$20 \$97
 defb \$10
 defb \$07 \$2d \$2d \$2d \$2d \$2d \$2d \$2d
 defb \$2d
 defb \$2d \$2d \$2d \$2d \$2d \$2d \$2d
 defb \$2d
 defb \$2d \$2d \$2d \$2d \$2d \$2d \$2d
 defb \$2d
 defb \$2d \$2d \$2d \$2d \$2d \$2d \$2d
 defb \$2d
 defb \$2d \$10 \$06 \$9a \$9b \$20 \$20
 defb \$20
 defb \$97 \$98 \$20 \$20 \$97 \$98 \$20
 defb \$95
 defb \$20 \$20 \$95 \$20 \$20 \$99 \$9a
 defb \$9b
 defb \$9a \$9b \$20 \$20 \$20 \$95 \$20
 defb \$20
 defb \$20 \$20 \$99 \$11 \$04 \$10 \$00
 defb \$9d
 defb \$9d \$9d \$9d \$9d \$9d \$9d
 defb \$9d
 defb \$9d \$9d \$9d \$9d \$9d \$9d \$9d
 defb \$9d
 defb \$9d \$9d \$9d \$9d \$9d \$9d \$9d
 defb \$9d
 defb \$04 \$20 \$20 \$20 \$20 \$20 \$20
 defb \$20
 defb \$20 \$20 \$20 \$20 \$20 \$20 \$20
 defb \$20
 defb \$20 \$20 \$20 \$20 \$20 \$20 \$20
 defb \$20
 defb \$20 \$20 \$20 \$20 \$20 \$20 \$20
 defb \$20

defb \$20
 defb \$90 \$91 \$20 \$20 \$90 \$91 \$20
 defb \$20
 defb \$20 \$20 \$20 \$20 \$20 \$20 \$90

63D2h
 25554 3E 02 1d a,\$02
 25556 CD 01 16 call \$1601

25559 11 41 61	ld de,\$6141	25667 3E 20	ld a,\$20
25562 01 91 02	ld bc,\$0291	25669 D7	rst \$10
25565 CD 3C 20	call \$203C		
63E0h			
25568 3A 02 5B	ld a,(\$5b02)	6446h	
25571 C6 30	add a,\$30	25670 CD 27 7E	call \$7e27
25573 32 01 64	ld (\$6401),a		
25576 18 26	jr +\$26		
63EAh			
defb \$11 \$01 \$10 \$07 \$20 \$53 \$43		6449h	
defb \$4f		25673 ED 4B 08 5B	ld bc,(\$5b08)
defb \$52 \$45 \$20 \$16 \$15 \$0b \$20		25677 CD 38 25	call \$2538
defb \$4d		25680 CD F1 2B	call \$2bf1
defb \$45 \$4e \$20 \$11 \$05 \$10 \$00		25683 1A	ld a,(de)
defb \$39		25684 F5	push af
		25685 CD BF 16	call \$16bf
		25688 F1	pop af
		25689 FE 20	cp \$20
		25691 C2 48 65	jp nz,\$6548
defb \$11 \$01 \$10 \$07 \$20 \$48 \$49		645Eh	
defb \$2d			
defb \$53 \$43 \$4f \$52 \$45 \$20		25694 3A 08 5B	ld a,(\$5b08)
		25697 FE 00	cp \$00
		25699 CA A6 64	jp z,\$66a6
6410h			
25616 3E 02	ld a,\$02		
25618 CD 01 16	call \$1601		
25621 11 EA 63	ld de,\$63ea		
25624 01 26 00	ld bc,\$0026		
25627 CD 3C 20	call \$203c		
641Fh			
25630 21 14 10	ld hl,\$1014		
25633 22 06 5B	ld (\$5b06),hl		
25636 22 08 5B	ld (\$5b08),hl		
6427h			
25639 3E 02	ld a,\$02		
25641 CD 01 16	call \$1601		
25644 3E 11	ld a,\$11		
25646 D7	rst \$10		
25647 3E 08	ld a,\$08		
25649 D7	rst \$10		
25650 3E 10	ld a,\$10		
25652 D7	rst \$10		
25653 3E 08	ld a,\$08		
25655 D7	rst \$10		
25656 3E 16	ld a,\$16		
25658 D7	rst \$10		
25659 3A 06 5B	ld a,(\$5b06)		
25662 D7	rst \$10		
25663 3A 07 5B	ld a,(\$5b07)		
25666 D7	rst \$10		
		25726 3A 09 5B	ld a,(\$5b09)
		25729 D7	rst \$10
		25730 3E 96	ld a,\$96
		25732 D7	rst \$10
		6485h	
		25733 2A 08 5B	ld hl,(\$5b08)
		25736 22 06 5B	ld (\$5b06),hl
		648Bh	

25739	01	FE	F7	ld bc,\$f7fe	25836	32	09	5B	ld (\$5b09),a
25742	ED	78		in a,(c)	25839	C3	F5	64	jp \$64f5
25744	CB	47		bit 0,a					
25746	20	1C		jr nz,+\$1c					

64F2h									
25842	CB	67		bit 4,a					
25844	C8			ret z					

6494h									
25748	21	32	00	ld hl,\$0032					
25751	11	05	00	ld de,\$0005					
25754	CD	B5	03	call \$03b5					

64F5h

25845	06	0A		ld b,\$0a
25847	11	78	00	ld de,\$0078
25850	21	01	00	ld hl,\$0001
25853	E5			push hl
25854	D5			push de
25855	C5			push bc
25856	CD	B5	03	call \$03b5
25859	C1			pop bc
25860	D1			pop de
25861	E1			pop hl
25862	7D			ld a,l
25863	3C			inc a
25864	6F			ld l,a
25865	00			nop
25866	10	F1		djnz -\$0f
25868	C3	27	64	jp \$6427

649Dh				
25757	3A	08	5B	ld a,(\$5b08)
25760	3D			dec a
25761	3D			dec a
25762	32	08	5B	ld (\$5b08),a
25765	2A	03	5B	ld hl,(\$5b03)
25768	23			inc hl
25769	23			inc hl
25770	23			inc hl
25771	23			inc hl
25772	23			inc hl
25773	22	03	5B	ld (\$5b03),hl
25776	00			nop
25777	3E	02		ld a,\$02

25779	CD	01	16	call \$1601
25782	3E	16		ld a,\$16
25784	D7			rst \$10
25785	3E	15		ld a,\$15
25787	D7			rst \$10
25788	3E	07		ld a,\$07
25790	D7			rst \$10
25791	ED	4B	03 5B	ld bc,(\$5b03)
25795	CD	2B	2D	call \$2d2b
25798	CD	E3	2D	call \$2de3

650Fh

25871	EF			rst \$28
-------	----	--	--	----------

6510h

defb	\$34	\$ea	\$23	\$d7	\$0a	\$3d	\$38
------	------	------	------	------	------	------	------

64C9h				
25801	01	FE	EF	ld bc,\$effe
25804	ED	78		in a,(c)
25806	CB	47		bit 0,a
25808	20	0E		jr nz,+\$0e
25810	3A	09	5B	ld a,(\$5b09)
25813	FE	1F		cp \$1f
25815	28	01		jr z,+\$01

6517h

25879	2A	B0	5C	ld hl,(\$5cb0)
25882	7C			ld a,h
25883	95			sub l
25884	CD	2B	2D	call \$2d2b
25887	CD	F8	03	call \$03f8
25890	3E	02		ld a,\$02
25892	CD	01	16	call \$1601
25895	3E	15		ld a,\$15
25897	D7			rst \$10
25898	3E	01		ld a,\$01
25900	D7			rst \$10
25901	3E	11		ld a,\$11

25817	3C			inc a
25818	32	09	5B	ld (\$5b09),a
25821	C3	F5	64	jp \$64f5
25824	CB	4F		bit 1,a
25826	20	0E		jr nz,+\$0e
25828	3A	09	5B	ld a,(\$5b09)
25831	FE	00		cp \$00
25833	28	01		jr z,+\$01
25835	3D			dec a

25903	D7			rst \$10
25904	3E	08		ld a,\$08
25906	D7			rst \$10
25907	3E	10		ld a,\$10
25909	D7			rst \$10

25910	3E	08	ld a,\$08	658Fh		
25912	D7		rst \$10	25999	3E 14	ld a,\$14
25913	3E	16	ld a,\$16	26001	32 02 5B	ld (\$5b02),a
25915	D7		rst \$10			
25916	3A	B0 5C	ld a,(\$5cb0)			
25919	D7		rst \$10			
25920	3A	09 5B	ld a,(\$5b09)	6594h		
25923	D7		rst \$10	26004	3A 02 5B	ld a,(\$5b02)
25924	3E	96	ld a,\$96	26007	A7	and a
25926	D7		rst \$10	26008	C2 46 64	jp nz,\$6446
25927	C9		ret			
6548h				659Bh		
25928	3A	08 5B	ld a,(\$5b08)	26011	AF	xor a
				26012	2A 03 5B	ld hl,(\$5b03)
				26015	ED 5B 00 5B	ld de,(\$5b00)
				26019	ED 52	sbc hl,de
25931	32	B0 5C	ld (\$5cb0),a			
25934	06	19	ld b,\$19	26021	3B 6A	jr c,+\$6a
25936	78		ld a,b			
25937	C5		push bc			
25938	32	B1 5C	ld (\$5cb1),a			
25941	CD	0F 65	call \$650f	65A7h		
25944	CD	0F 65	call \$650f	26023	2A 03 5B	ld hl,(\$5b03)
25947	C1		pop bc	26026	22 00 5B	ld (\$5b00),hl
25948	04		inc b	26029	E5	push hl
25949	78		ld a,b	26030	C1	pop bc
25950	FE	23	cp \$23	26031	CD 2B 2D	call \$2d2b
25952	20	EF	jr nz,-\$11	26034	3E 02	ld a,\$02
				26036	CD 01 16	call \$1601
				26039	3E 16	ld a,\$16
				26041	D7	rst \$10
6562h				26042	3E 15	ld a,\$15
25954	CD	0F 65	call \$650f	26044	D7	rst \$10
25957	CD	0F 65	call \$650f	26045	3E 1B	ld a,\$1b
25960	3A	B0 5C	ld a,(\$5cb0)	26047	D7	rst \$10
25963	3C		inc a	26048	CD E3 2D	call \$2de3
25964	3C		inc a	26051	18 4C	jr +\$4c
25965	32	B0 5C	ld (\$5cb0),a			
25968	FE	14	cp \$14			
25970	20	EE	jr nz,-\$12	6505h		
				defb	\$12 \$01 \$11 \$07 \$16 \$0c \$00	
				defb	\$20	
				defb	\$20 \$20 \$20 \$20 \$20 \$20 \$20	
				defb	\$20	
6574h				defb	\$20 \$20 \$47 \$41 \$4d \$45 \$20	
25972	3A	02 5B	ld a,(\$5b02)	defb	\$20	
25975	3D		dec a	defb	\$4f \$56 \$45 \$52 \$20 \$20 \$20	
25976	32	02 5B	ld (\$5b02),a	defb	\$20	
25979	3E	02	ld a,\$02	defb	\$4f \$56 \$45 \$52 \$20 \$20 \$20	
25981	CD	01 16	call \$1601	defb	\$20 \$20 \$20 \$20 \$20 \$20 \$20	
25984	3E	16	ld a,\$16	defb	\$16	
25986	D7		rst \$10	defb	\$0e \$00 \$12 \$00 \$20 \$20 \$41	
25987	3E	16	ld a,\$16	defb	\$6e	
25989	D7		rst \$10	defb	\$6f \$74 \$68 \$65 \$72 \$20 \$67	
25990	3E	10	ld a,\$10	defb	\$61	
25992	D7		rst \$10	defb	\$6d \$65 \$20 \$3f \$20 \$28 \$79	
25993	3A	02 5B	ld a,(\$5b02)	defb	\$29	
25996	C6	30	add a,\$30	defb	\$65 \$73 \$20 \$20 \$28 \$6e \$29	
25998	D7		rst \$10	defb	\$6f	

```

defb $20 $20 $20 $20

6611h
26129 11 C5 65 ld de,$65c5
26132 01 4C 00 ld bc,$004c
26135 CD 3C 20 call $203c

661Ah
26138 76 halt
26139 FD CB 01 6E bit 5,(iy+$01
26143 28 F9 jr z,-$07
26145 FD CB 01 AE res 5,(iy+$01
26149 3A 0B 5C ld a,($5c0b)
26152 FE 6E cp $6e
26154 20 03 jr nz,$+03
26156 CD 00 00 call $0000
26159 FE 79 cp $79

26161 20 E7 jr nz,-$19

6633h
26163 3E 02 ld a,$02
26165 CD 01 16 call $1601
26168 3E 11 ld a,$11
26170 D7 rst $10
26171 3E 05 ld a,$05
26173 D7 rst $10
26174 3E 16 ld a,$16
26176 D7 rst $10
26177 3E 15 ld a,$15
26179 D7 rst $10
26180 3E 07 ld a,$07
26182 D7 rst $10
26183 3E 20 ld a,$20
26185 D7 rst $10
26186 3E 20 ld a,$20
26188 D7 rst $10

26189 3E 20 ld a,$20
26191 D7 rst $10
26192 3E 20 ld a,$20
26194 D7 rst $10
26195 C3 C9 60 jp $60c9

6656h
defb $00 $00 $00 $00 $00 $00 $ed
defb $0b
defb $ed $0b $f0 $10 $ec $0b $ec
defb $10
defb $ec $0b $ec $10

666Ah
26218 06 07 ld b,$07
26220 21 5C 66 ld hl,$665c
26223 7E ld a,(hl)

26224 32 78 66 ld ($6678),a
26227 23 inc hl
26228 C5 push bc
26229 E5 push hl
26230 EF rst $28
defb $34 $ec $4c $cc $cc $cc $38
26238 E1 pop hl
26239 7E ld a,(hl)
26240 E5 push hl
26241 CD 28 2D call $2d28
26244 CD F8 03 call $03f8
26247 E1 pop hl
26248 C1 pop bc
26249 23 inc hl
26250 10 E3 djnz -$1d
26252 3E 01 ld a,$01
26254 CD 28 2D call $2d28
26257 3E 14 ld a,$14
26259 CD 28 2D call $2d28
26262 CD F8 03 call $03f8
26265 C9 ret

669Ah
defb $11 $08 $10 $08 $16 $02 $04
defb $20
defb $16 $00 $04 $96

66A6h
26278 2A 06 5B ld hl,($5b06)
26281 22 9F 66 ld ($669f),hl
26284 2A 08 5B ld hl,($5b08)
26287 22 A3 66 ld ($66a3),hl
26290 3E 02 ld a,$02
26292 CD 01 16 call $1601
26295 11 9A 66 ld de,$669a
26298 01 0C 00 ld bc,$000c
26301 CD 3C 20 call $203c
26304 CD 6A 66 call $666a

66C3h
26307 3A 05 5B ld a,($5b05)
26310 3C inc a
26311 32 05 5B ld ($5b05),a
26314 2A 03 5B ld hl,($5b03)
26317 11 32 00 ld de,$0032
26320 19 add hl,de
26321 22 03 5B ld ($5b03),hl
26324 E5 push hl
26325 3E 02 ld a,$02
26327 CD 01 16 call $1601

```

26330	3E 16	ld a,\$16	defb \$a1 \$0f \$34 \$37 \$16 \$04 \$34
26332	D7	rst \$10	defb \$80
26333	3E 15	ld a,\$15	defb \$41 \$00 \$00 \$80 \$32 \$02 \$a1
26335	D7	rst \$10	defb \$03 \$31 \$38
26336	3E 07	ld a,\$07	
26338	D7	rst \$10	
26339	C1	pop bc	

6733h

26340	CD 2B 2D	call \$2d2b	26419	CD A2 2D	call \$2da2
26343	CD E3 2D	call \$2de3	26422	ED 43 76 5C	ld (\$5c76),bc
			26426	79	ld a,c
66EAh			26427	E6 1F	and \$1f
26346	3A 05 5B	ld a,(\$5b05)	26429	D6 02	sub \$02
26349	E6 03	and \$03	26431	38 05	jr c,+\$05
26351	C2 1E 64	jp nz,\$641e			

6741h

66F2h			26433	3C	inc a
26354	AF	xor a			
26355	3A 05 5B	ld a,(\$5b05)	6742h		
26358	FE 04	cp \$04	26434	FE 20	cp \$20
26360	20 06	jr nz,+\$06	26436	20 01	jr nz,+\$01
26362	AF	xor a	26438	AF	xor a
26363	32 A9 7E	ld (\$7ea9),a			
26366	18 19	jr +\$19			

6700h			6747h		
26368	FE 08	cp \$08	26439	F5	push af
26370	20 06	jr nz,+\$06	26440	47	ld b,a
26372	AF	xor a	26441	0E 0A	ld c,\$0a
26373	32 C2 7E	ld (\$7ec2),a	26443	CD 38 25	call \$2538
26376	18 0F	jr +\$0f	26446	CD F1 2B	call \$2bf1
			26449	1A	ld a,(de)
670Ah			26450	F5	push af
26378	FE 0C	cp \$0c	26451	CD BF 16	call \$16bf
26380	20 06	jr nz,+\$06	26454	F1	pop af
26382	AF	xor a	26455	FE 20	cp \$20
26383	32 D5 7E	ld (\$7ed5),a	26457	28 03	jr z,+\$03
26386	18 05	jr +\$05	26459	F1	pop af
			26460	18 E3	jr -\$1d
			26462	F1	pop af

6714h

675Fh

26388	D6 25	sub \$25	26463	3C	inc a
26390	D2 29 61	jp nc,\$6129	26464	F5	push af
5718h			26465	47	ld b,a
26393	ED 4B 76 5C	ld bc,(\$5c76)	26466	0E 0A	ld c,\$0a
26397	CD 2B 2D	call \$2d2b	26468	CD 38 25	call \$2538
26400	EF	rst \$28	26471	CD F1 2B	call \$2bf1
			26474	1A	ld a,(de)
6721h			26475	F5	push af
			26476	CD BF 16	call \$16bf
			26479	F1	pop af
			26480	FE 20	cp \$20
			26482	28 03	jr z,+\$03
			26484	F1	pop af
			26485	18 CA	jr -\$36

6777h			26503 CD 01 16	call \$1601
26487 F1	pop af		26506 11 7E 67	ld de,\$677e
26488 3D	dec a		26509 01 07 00	ld bc,\$0007
26489 32 82 67	ld (\$6782),a		26512 CD 3C 20	call \$203c
26492 18 07	jr +\$07			

6793h				
26515 CD 6A 66			call \$666a	

677Eh
defb \$11 \$04 \$16 \$0a \$19 \$9e \$9f

6785h			6796h	
26501 3E 02	ld a,\$02		26518 C3 29 61	jp \$6129

Program 'Cross (machine code)'

Appendix 1

Replacement ROM routines

The Spectrum ROM routines are written in such a way that any wrong parameters are spotted and the correct error report given. For example, if you try to input PRINT AT 24, 33; "Message" then the PRINT routine will detect that line 24 column 33 is out of screen.

When writing machine code programs it can be assumed that the correct parameters will always be used and so the ROM routines can be rewritten without the error detection etc., which makes them faster and completely under programmer control.

The following six programs show how CLS., ATTR. SET, PRINT, PLOT, UNPLOT, SCR\$ and POINT can be rewritten. The PRINT routine handles characters 32 to 127 inclusive (although it could be adapted to PRINT any character) and actually POKEs the character on to the screen thus making it unnecessary to OPEN CHANNEL 2 and allows printing on lines 22 and 23. The unused variable addresses 23728/9 are used to hold the current PRINT position.

The SCR\$ routine detects characters 32 to 127 on lines 0 to 21 as the BASIC version but also allows lines 22 and 23 to be checked. The PLOT and UNPLOT routines have the same parameters as the BASIC, with 0, 0 as the bottom left-hand corner of line 21. If you change the value in address 23774 to 191 then the whole screen can be PLOTted and 0,0 will be the bottom left-hand corner of line 23. (Omitting instructions LD A, 175 at address 23774/5 will make 0,0 the top left-hand corner of line 1). The above also applies to the POINT routine. The address to be modified is 23777. You will see that the PLOT, UNPLOT and POINT routines use identical FIND subroutines to find the pixel to be checked; therefore if you were using all three routines they could share this common subroutine and save more memory. You could increase speed still more by adding DI (disable interrupt) at the start of these routines, or indeed at the beginning of your program, and EI at the end (or when a KEYSKAN is required). With these programs as a guide you could write a CIRCLE routine (one that runs faster than the Spectrum's).

```
org 23760                                23771 C9                                ret
CLS ( Slow )

23760 21 00 40    ld hl,16384    CLS ( Fast )
L1
23763 36 00      ld (hl),0      23772 21 00 40    ld hl,16384
23765 23        inc hl          23775 11 01 40    ld de,16385
23766 7C        ld a,h          23778 01 FF 17    ld bc,6143
23767 FE 58     cp 88          23781 36 00      ld (hl),0
23769 20 FB     jr nz,L1    23783 ED B0      ldir
Program A1.1    23785 C9                                ret
```

Appendix 1—continued

org 23760
ATTRIBUTE FILL

EXAMPLE
INK 2 PAPER 5
BRIGHT 1 FLASH 1

2+40+64+128=234

Program A1.2

org 23760
PRINT STRING ROUTINE
POKES CHR. ONTO SCREEN
DOES NOT USE RST 16d

ADDRESS 23728 HOLDS COLUMN
ADDRESS 23729 HOLDS LINE

```
23760 11 E4 5C    ld de,DATA
L4
23763 1A         ld a,(de)
23764 CB 7F     bit 7,a
23766 20 06     jr nz,L3
23768 CD 02 5D  call PRINT
23771 13         inc de
23772 18 F5     jr L4
L3
23774 CB BF     res 7,a
23776 CD 02 5D  call PRINT
23779 C9         ret
```

DATA
defs McGRAW-HILL BOOK Co.
defs (UK) Ltd

128+CHR 46="."

defb 174

PRINT ROUTINE

GET LINE/COLUMN IN HL

```
PRINT
23810 D5         push de
23811 2A B0 5C   ld hl,(23728)
23814 E5         push hl
23815 F5         push af
```

CHECK OUT OF SCREEN

```
23816 AF         xor a
```

```
23817 7C         ld a,h
23818 D6 18     sub 24
23820 38 02     jr c,OK
23822 CF         rst 8
defb 4
```

```
23760 3E EA     ld a,234
23762 21 00 58  ld hl,22528
23765 11 01 58  ld de,22529
23768 01 FF 02  ld bc,767
23771 77         ld (hl),a
23772 ED B0     ldir
23774 C9         ret
```

FIND SCREEN POSITION

```
OK
23824 11 00 40  ld de,16384
23827 19         add hl,de
23828 7C         ld a,h
23829 E6 07     and 7
23831 0F         rrca
23832 0F         rrca
23833 0F         rrca
23834 B5         or 1
23835 6F         ld l,a
23836 3E F8     ld a,248
23838 A4         and h
23839 67         ld h,a
```

FIND START OF CHR.
IN CHR. GENERATOR

```
23840 F1         pop af
23841 E5         push hl
23842 11 00 3C   ld de,15360
23845 6F         ld l,a
23846 26 00     ld h,0
23848 29         add hl,hl
23849 29         add hl,hl
23850 29         add hl,hl
23851 19         add hl,de
23852 D1         pop de
```

POKE CHR. ONTO SCREEN

```
23853 06 08     ld b,8
L1
23855 7E         ld a,(hl)
23856 12         ld (de),a
```

```
23857 23         inc hl
23858 14         inc d
23859 10 FA     djnz L1
```

UPDATE SCREEN POSITION

```
23861 E1         pop hl
23862 2C         inc l
23863 CB 6D     bit 5,l
23865 28 03     jr z,L2
```

Appendix 1—continued

```

23867 CB AD      res 5,1
23869 24         inc h
L2
23870 22 B0 5C   ld (2372B),h1
23873 D1         pop de
23874 C9         ret

```

Program A1.3

org 23760

PROGRAM TO REPLACE SCREEN\$
REQUIRES H=LINE, L=COLUMN
RETURNS WITH A REGISTER
HOLDING CHR CODE
OR 0 IF CHR NOT FOUND

```

23760 26 0A      ld h,10
23762 2E 07      ld l,7
23764 CD D8 5C   call SCR$
23767 C9         ret

```

SCREEN\$ ROUTINE

SCR\$

FIND SCREEN ADDRESS OF CHR

```

23768 11 00 40   ld de,16384

```

```

23771 19         add hl,de
23772 7C         ld a,h
23773 E6 07      and 7
23775 0F         rrca
23776 0F         rrca
23777 0F         rrca
23778 B5         or l
23779 6F         ld l,a
23780 3E F8      ld a,248
23782 A4         and h
23783 67         ld h,a

```

CHECK CHR WITH CHR GEN.

```

23784 11 00 3D   ld de,15616
L3
23787 06 08      ld b,9
23789 0E 00      ld c,0

```

Program A1.4

org 23760

PROGRAM TO REPLACE ROM PLOT
REQUIRES H=y, L=x

```

23760 26 00      ld h,0
23762 2E 5F      ld l,95
23764 CD D8 5C   call PLOT
23767 C9         ret
PLOT

```

CHECK FOR DE=16384

```

23791 CB 72      bit 6,d
23793 20 11      jr nz,NFOUND

```

```

23795 E5         push hl
L2
23796 1A         ld a,(de)
23797 BE         cp (hl)
23798 20 01      jr nz,L1
23800 0C         inc c
L1
23801 13         inc de
23802 24         inc h
23803 10 F7      djnz L2
23805 E1         pop hl
23806 CB 59      bit 3,c
23808 20 04      jr nz,FOUND
23810 18 E7      jr L3
NFOUND
23812 AF         xor a
23813 C9         ret

```

CONVERT TO CHR CODE

FOUND

```

23814 B7         or a
23815 21 00 3C   ld hl,15360
23818 EB         ex de,hl
23819 ED 52      sbc hl,de
23821 06 03      ld b,3
L4
23823 CB 1C      rr h
23825 CB 1D      rr l
23827 10 FA      djnz L4
23829 7D         ld a,l
23830 3D         dec a
23831 C9         ret

```

```

23768 CD DE 5C   call FIND
23771 B6         or (hl)
23772 77         ld (hl),a
23773 C9         ret

```

FIND SCREEN BIT
RET WITH HL=BYTE, A=BIT

FIND

Appendix 1—continued

23774 3E AF	ld a,175	23803 57	ld d,a
23776 94	sub h		
23777 67	ld h,a	23804 7C	ld a,h
23778 11 00 40	ld de,16384	23805 E6 C0	and 192
23781 7D	ld a,l	23807 1F	rra
23782 E6 07	and 7	23808 1F	rra
23784 F5	push af	23809 1F	rra
23785 7D	ld a,l	23810 B2	or d
23786 E6 F8	and 248	23811 57	ld d,a
23788 1F	rra	23812 F1	pop af
23789 1F	rra	23813 2E 80	ld l,128
23790 1F	rra	L1	
23791 5F	ld e,a	23815 A7	and a
23792 7C	ld a,h	23816 FE 00	cp 0
23793 E6 38	and 56	23818 28 05	jr z,L2
23795 17	rla	23820 CB 1D	rr l
23796 17	rla	23822 3D	dec a
23797 B3	or e	23823 18 F6	jr L1
23798 5F	ld e,a	L2	
23799 7C	ld a,h	23825 7D	ld a,l
23800 E6 07	and 7	23826 EB	ex de,hl
23802 B2	or d	23827 C9	ret

Program A1.5

org 23760		23791 5F	ld e,a
PROGRAM TO REPLACE		23792 7C	ld a,h
ROM PLOT OVER 1		23793 E6 38	and 56
REQUIRES H=y, L=x		23795 17	rla
		23796 17	rla
23760 26 00	ld h,0	23797 B3	or e
23762 2E 5F	ld l,95	23798 5F	ld e,a
23764 CD D8 5C	call PLOT OVER	23799 7C	ld a,h
23767 C9	ret	23800 E6 07	and 7
PLOT OVER		23802 B2	or d
23768 CD DE 5C	call FIND		
23771 AE	xor (hl)	23803 57	ld d,a
23772 77	ld (hl),a	23804 7C	ld a,h
23773 C9	ret	23805 E6 C0	and 192
		23807 1F	rra
FIND SCREEN BIT		23808 1F	rra
RET WITH HL=BYTE, A=BIT		23809 1F	rra
		23810 B2	or d
FIND		23811 57	ld d,a
23774 3E AF	ld a,175	23812 F1	pop af
		23813 2E 80	ld l,128
23776 94	sub h	L1	
23777 67	ld h,a	23815 A7	and a
23778 11 00 40	ld de,16384	23816 FE 00	cp 0
23781 7D	ld a,l	23818 28 05	jr z,L2
23782 E6 07	and 7	23820 CB 1D	rr l
23784 F5	push af	23822 3D	dec a
23785 7D	ld a,l	23823 18 F6	jr L1
23786 E6 F8	and 248	L2	
23788 1F	rra	23825 7D	ld a,l
23789 1F	rra	23826 EB	ex de,hl
23790 1F	rra	23827 C9	ret

Program A1.6

Appendix 1—continued

org 23760			23794 7C	ld a,h
PROGRAM TO REPLACE			23795 E6 38	and 56
ROM POINT			23797 17	rla
REQUIRES H=y, L=x			23798 17	rla
			23799 B3	or e
23760 26 00	ld h,0		23800 5F	ld e,a
23762 2E 5F	ld l,95		23801 7C	ld a,h
23764 CD D8 5C	call POINT		23802 E6 07	and 7
23767 C9	ret			
POINT				
23768 CD E0 5C	call FIND		23804 B2	or d
23771 A6	and (hl)		23805 57	ld d,a
23772 C8	ret z		23806 7C	ld a,h
23773 3E 01	ld a,1		23807 E6 C0	and 192
23775 C9	ret		23809 1F	rra
			23810 1F	rra
FIND SCREEN BIT			23811 1F	rra
RET WITH HL=BYTE, A=BIT			23812 B2	or d
			23813 57	ld d,a
FIND			23814 F1	pop af
			23815 2E 80	ld l,128
			L1	
23776 3E AF	ld a,175		23817 A7	and a
23778 94	sub h		23818 FE 00	cp 0
23779 67	ld h,a		23820 28 05	jr z,L2
23780 11 00 40	ld de,16384		23822 CB 1D	rr l
23783 7D	ld a,1		23824 3D	dec a
23784 E6 07	and 7		23825 18 F6	jr L1
23786 F5	push af		L2	
23787 7D	ld a,1		23827 7D	ld a,1
23788 E6 FB	and 248		23828 EB	ex de,hl
23790 1F	rra			
23791 1F	rra			
23792 1F	rra		23829 C9	ret
23793 5F	ld e,a			

Program A1.7

Appendix 2

Spectrum (Z80) machine code listings

NOTES: (HL)—the number held in the address to which register pair HL points
 NN is entered as second number first, for example, 4000 is entered as 00 40 (in hex.)
 Register A—the general-purpose accumulator

Decimal	Bytes	Hex	Mnemonic	Description
0	1	00	nop	No operation
1	3	01	ld bc,NN	Load BC with NN
2	1	02	ld (bc),a	Store A to (BC)
3	1	03	inc bc	Increment BC by 1
4	1	04	inc b	Increment B by 1
5	1	05	dec b	Decrement B by 1
6	2	06	ld b,N	Load B with N
7	1	07	rlca	Rotate left circular A
8	1	08	ex af,af	Set prime AF active
9	1	09	add hl,bc	BC + HL → HL
10	3	0A	ld a,(bc)	Load A with number in location (BC)
11	1	0B	dec bc	Decrement BC by 1
12	1	0C	inc c	Increment C by 1
13	1	0D	dec c	Decrement C by 1
14	2	0E	ld c,N	Load C with N
15	1	0F	rrca	Rotate A right circular
16	2	10	djnz x	Decrement B and JR if B ≠ 0, + or - x
17	3	11	ld de,NN	Load DE with NN
18	1	12	ld (de),a	Store A to (DE)
19	1	13	inc de	Increment DE by 1
20	1	14	inc d	Increment D by 1
21	1	15	dec d	Decrement D by 1
22	2	16	ld d,N	Load D with N
23	1	17	rla	Rotate A left thru carry
24	2	18	jr x	Unconditional jump relative, + or - x
25	1	19	add hl,de	DE + HL → HL
26	3	1A	ld a,(de)	Load A with location (DE)
27	1	1B	dec de	Decrement DE by 1
28	1	1C	inc e	Increment E by 1
29	1	1D	dec e	Decrement E by 1
30	2	1E	ld e,N	Load E with N
31	1	1F	rra	Rotate A right thru carry
32	2	20	jr nz, x	Jump relative if non-zero, + or - x
33	3	21	ld hl,NN	Load HL with NN
34	3	22	ld (NN),hl	Store HL to location NN
35	1	23	inc hl	Increment HL by 1
36	1	24	inc h	Increment H by 1
37	1	25	dec h	Decrement H by 1
38	2	26	ld h,N	Load H with N
39	1	27	daa	Decimal adjust A
40	2	28	jr z, x	Jump relative if zero, + or - x
41	1	29	add hl,hl	HL + HL → HL
42	3	2A	ld hl,(NN)	Load HL with location (NN)
43	1	2B	dec hl	Decrement HL by 1
44	1	2C	inc l	Increment L by 1
45	1	2D	dec l	Decrement L by 1
46	2	2E	ld l,N	Load L with N
47	1	2F	cpl	Complement A (1's comp.)
48	2	30	jr nc, x	Jump relative if no carry, + or - x
49	3	31	ld sp,NN	Load stack pointer with NN
50	3	32	ld (NN),a	Store A to location NN
51	1	33	inc sp	Increment SP by 1
52	1	34	inc (hl)	Increment (HL) by 1
53	1	35	dec (hl)	Decrement (HL) by 1
54	2	36	ld (hl),N	Store N to (HL)
55	1	37	scf	Set carry flag
56	2	38	jr c, x	Jump relative if carry, + or - x
57	1	39	add hl,sp	SP + HL → HL
58	3	3A	ld a,(NN)	Load A with location (NN)
59	1	3B	dec sp	Decrement SP by 1
60	1	3C	inc a	Increment A by 1
61	1	3D	dec a	Decrement A by 1

Appendix 2—continued

<i>Decimal</i>	<i>Bytes</i>	<i>Hex</i>	<i>Mnemonic</i>	<i>Description</i>
62	2	3E	ld a,N	Load A with N
63	1	3F	ccf	Complement carry flag
64	1	40	ld b,b	Move B to B
65	1	41	ld b,c	Move C to B
66	1	42	ld b,d	Move D to B
67	1	43	ld b,e	Move E to B
68	1	44	ld b,h	Move H to B
69	1	45	ld b,l	Move L to B
70	1	46	ld b,(hl)	Move (HL) to B
71	1	47	ld b,a	Move A to B
72	1	48	ld c,b	Move B to C
73	1	49	ld c,c	Move C to C
74	1	4A	ld c,d	Move D to C
75	1	4B	ld c,e	Move E to C
76	1	4C	ld c,h	Move H to C
77	1	4D	ld c,l	Move L to C
78	1	4E	ld c,(hl)	Move (HL) to C
79	1	4F	ld c,a	Move A to C
80	1	50	ld d,b	Move B to D
81	1	51	ld d,c	Move C to D
82	1	52	ld d,d	Move D to D
83	1	53	ld d,e	Move E to D
84	1	54	ld d,h	Move H to D
85	1	55	ld d,l	Move L to D
86	1	56	ld d,(hl)	Move (HL) to D
87	1	57	ld d,a	Move A to D
88	1	58	ld e,b	Move B to E
89	1	59	ld e,c	Move C to E
90	1	5A	ld e,d	Move D to E
91	1	5B	ld e,e	Move E to E
92	1	5C	ld e,h	Move H to E
93	1	5D	ld e,l	Move L to E
94	1	5E	ld e,(hl)	Move (HL) to E
95	1	5F	ld e,a	Move A to E
96	1	60	ld h,b	Move B to H
97	1	61	ld h,c	Move C to H
98	1	62	ld h,d	Move D to H
99	1	63	ld h,e	Move E to H
100	1	64	ld h,h	Move H to H
101	1	65	ld h,l	Move L to H
102	1	66	ld h,(hl)	Move (HL) to H
103	1	67	ld h,a	Move A to H
104	1	68	ld l,b	Move B to L
105	1	69	ld l,c	Move C to L
106	1	6A	ld l,d	Move D to L
107	1	6B	ld l,e	Move E to L
108	1	6C	ld l,h	Move H to L
109	1	6D	ld l,l	Move L to L
110	1	6E	ld l,(hl)	Move (HL) to L
111	1	6F	ld l,a	Move A to L
112	1	70	ld (hl),b	Move B to (HL)
113	1	71	ld (hl),c	Move C to (HL)
114	1	72	ld (hl),d	Move D to (HL)
115	1	73	ld (hl),e	Move E to (HL)
116	1	74	ld (hl),h	Move H to (HL)
117	1	75	ld (hl),l	Move L to (HL)
118	1	76	halt	HALT
119	1	77	ld (hl),a	Move A to (HL)
120	1	78	ld a,b	Move B to A
121	1	79	ld a,c	Move C to A
122	1	7A	ld a,d	Move D to A
123	1	7B	ld a,e	Move E to A
124	1	7C	ld a,h	Move H to A
125	1	7D	ld a,l	Move L to A
126	1	7E	ld a,(hl)	Move (HL) to A
127	1	7F	ld a,a	Move A to A
128	1	80	add a,b	B+A→A
129	1	81	add a,c	C+A→A
130	1	82	add a,d	D+A→A
131	1	83	add a,e	E+A→A

Appendix 2—continued

Decimal	Bytes	Hex	Mnemonic	Description
132	1	84	add a,h	H+A→A
133	1	85	add a,l	L+A→A
134	1	86	add a,(hl)	(HL)+A→A
135	1	87	add a,a	A+A→A
136	1	88	adc a,b	B+A+carry→A
137	1	89	adc a,c	C+A+carry→A
138	1	8A	adc a,d	D+A+carry→A
139	1	8B	adc a,e	E+A+carry→A
140	1	8C	adc a,h	H+A+carry→A
141	1	8D	adc a,l	L+A+carry→A
142	1	8E	adc a,(hl)	(HL)+A+carry→A
143	1	8F	adc a,a	A+A+carry→A
144	1	90	sub b	A-B→A
145	1	91	sub c	A-C→A
146	1	92	sub d	A-D→A
147	1	93	sub e	A-E→A
148	1	94	sub h	A-H→A
149	1	95	sub l	A-L→A
150	1	96	sub (hl)	A-(HL)→A
151	1	97	sub a	A-A→A
152	1	98	sbc a,b	A-B-carry→A
153	1	99	sbc a,c	A-C-carry→A
154	1	9A	sbc a,d	A-D-carry→A
155	1	9B	sbc a,e	A-E-carry→A
156	1	9C	sbc a,h	A-H-carry→A
157	1	9D	sbc a,l	A-L-carry→A
158	1	9E	sbc a,(hl)	A-(HL)-carry→A
159	1	9F	sbc a,a	A-A-carry→A
160	1	A0	and b	A and B→A
161	1	A1	and c	A and C→A
162	1	A2	and d	A and D→A
163	1	A3	and e	A and E→A
164	1	A4	and h	A and H→A
165	1	A5	and l	A and L→A
166	1	A6	and (hl)	A and (HL)→A
167	1	A7	and a	A and A→A
168	1	A8	xor b	A exclusive or B→A
169	1	A9	xor c	A exclusive or C→A
170	1	AA	xor d	A exclusive or D→A
171	1	AB	xor e	A exclusive or E→A
172	1	AC	xor h	A exclusive or H→A
173	1	AD	xor l	A exclusive or L→A
174	1	AE	xor (hl)	A exclusive or (HL)→A
175	1	AF	xor a	A exclusive or A→A
176	1	B0	or b	A or B→A
177	1	B1	or c	A or C→A
178	1	B2	or d	A or D→A
179	1	B3	or e	A or E→A
180	1	B4	or h	A or H→A
181	1	B5	or l	A or L→A
182	1	B6	or (hl)	A or (HL)→A
183	1	B7	or a	A or A→A
184	1	B8	cp b	Compare A:B
185	1	B9	cp c	Compare A:C
186	1	BA	cp d	Compare A:D
187	1	BB	cp e	Compare A:E
188	1	BC	cp h	Compare A:H
189	1	BD	cp l	Compare A:L
190	1	BE	cp (hl)	Compare A:(HL)
191	1	BF	cp a	Compare A:A
192	1	C0	ret nz	Return if non-zero
193	1	C1	pop bc	Pop BC from stack
194	3	C2	jp nz,NN	Jump to NN if non zero
195	3	C3	jp NN	Unconditional jump to NN
196	3	C4	call nz,NN	Call NN if non-zero
197	1	C5	push bc	Push BC onto stack
198	2	C6	add a,N	A+N→A
199	1	C7	rst 0	Call 0000 start routine
200	1	C8	ret z	Return if zero
201	1	C9	ret	Return

Appendix 2—continued

Decimal	Bytes	Hex	Mnemonic	Description
202	3	CA	jp z,NN	Jump to NN if zero
203		CB		See special set of CB routines
204	3	CC	call z,NN	Call NN if zero
205	3	CD	call NN	Call NN
206	2	CE	adc a,N	A + N + carry → A
207	1	CF	rst 8	Call 0008 error routine
208	1	D0	ret nc	Return if carry = 0
209	1	D1	pop de	Pop DE from stack
210	3	D2	jp nc,NN	Jump to NN if carry = 0
211	2	D3	out (N),a	Output A to port N
212	3	D4	call nc,NN	Call NN if carry = 0
213	1	D5	push de	Push DE onto stack
214	2	D6	sub N	A - N → A
215	1	D7	rst 16	Call 0010 print routine
216	1	D8	ret c	Return if carry = 1
217	1	D9	exx	Set prime B-L active (exchange registers)
218	3	DA	jp c,NN	Jump to NN if carry = 1
219	2	DB	in a,(N)	Input to A from port N
220	3	DC	call c,NN	Call NN if carry = 1
221		DD	prefixes instructions using ix	See special set of DD routines
222	2	DE	sbc a,N	A - N - carry → A
223	1	DF	rst 24	Call 0018 character routine (1)
224	1	E0	ret po	Return if overflow/parity flag = 0
225	1	E1	pop hl	Pop HL from stack
226	3	E2	jp po,NN	Jump to NN if overflow/parity flag = 0
227	1	E3	ex (sp),hl	Exchange (SP) and HL
228	3	E4	call po,NN	Call NN if overflow/parity flag = 0
229	1	E5	push hl	Push HL onto stack
230	2	E6	and N	A and N → A
231	1	E7	rst 32	Call 0020 character routine (2)
232	1	E8	ret pe	Return if overflow/parity flag = 1
233	1	E9	jp (hl)	Jump to location (HL)
234	3	EA	jp pe,NN	Jump to NN if overflow/parity flag = 1
235	1	EB	ex de,hl	Exchange DE and HL
236	3	EC	call pe,NN	Call NN if overflow/parity flag = 1
237		ED		See special set of ED routines
238	2	EE	xor N	A exclusive or N → A
239	1	EF	rst 40	Call 0028 calculator routine
240	1	F0	ret p	Return if sign flag = 0
241	1	F1	pop af	Pop AF from stack
242	3	F2	jp p,NN	Jump to NN if sign flag = 0
243	1	F3	di	Disable interrupts
244	3	F4	call p,NN	Call NN if sign flag = 0
245	1	F5	push af	Push AF onto stack
246	2	F6	or N	A or N → A
247	1	F7	rst 48	Call 0030 work space routine
248	1	F8	ret m	Return if sign flag = 1
249	1	F9	ld sp,hl	Move HL to SP
250	3	FA	jp m,NN	Jump to NN if sign flag = 1
251	1	FB	ei	Enable interrupts
252	3	FC	call m,NN	Call NN if sign flag = 1
253		FD	prefixes instructions using iy	See special set of FD routines
254	2	FE	cp N	Compare A:N
255	1	FF	rst 56	Call 0038

Appendix 2—continued

DD 09	ADD IX,BC	DD CB d 06	RLC	(IX+d)
DD 19	ADD IX,DE	DD CB d 0E	RRC	(IX+d)
DD 21 +dddd	LD IX,+dddd	DD CB d 16	RL	(IX+d)
DD 22 addr	LD (addr),IX	DD CB d 1E	RR	(IX+d)
DD 23	INC IX	DD CB d 26	SLA	(IX+d)
DD 29	ADD IX,IX	DD CB d 2E	SRA	(IX+d)
DD 2A addr	LD IX,(addr)	DD CB d 3E	SRL	(IX+d)
DD 2B	DEC IX	DD CB d 46	BIT	0,(IX+d)
DD 34 d	INC (IX+d)	DD CB d 4E	BIT	1,(IX+d)
DD 35 d	DEC (IX+d)	DD CB d 56	BIT	2,(IX+d)
DD 36 d +dd	LD (IX+d),+dd	DD CB d 5E	BIT	3,(IX+d)
DD 39	ADD IX,SP	DD CB d 66	BIT	4,(IX+d)
DD 46 d	LD B,(IX+d)	DD CB d 6E	BIT	5,(IX+d)
DD 4E d	LD C,(IX+d)	DD CB d 76	BIT	6,(IX+d)
DD 56 d	LD D,(IX+d)	DD CB d 7E	BIT	7,(IX+d)
DD 5E d	LD E,(IX+d)	DD CB d 86	RES	0,(IX+d)
DD 66 d	LD H,(IX+d)	DD CB d 8E	RES	1,(IX+d)
DD 6E d	LD L,(IX+d)	DD CB d 96	RES	2,(IX+d)
DD 70 d	LD (IX+d),B	DD CB d 9E	RS	3,(IX+d)
DD 71 d	LD (IX+d),C	DD CB d A6	RES	4,(IX+d)
DD 72 d	LD (IX+d),D	DD CB d AE	RES	5,(IX+d)
DD 73 d	LD (IX+d),E	DD CB d B6	RES	6,(IX+d)
DD 74 d	LD (IX+d),H	DD CB d BE	RES	7,(IX+d)
DD 75 d	LD (IX+d),L	DD CB d C6	SET	0,(IX+d)
DD 77 d	LD (IX+d),A	DD CB d CE	SET	1,(IX+d)
DD 7E d	LD A,(IX+d)	DD CB d D6	SET	2,(IX+d)
	ADD A,(IX+d)	DD CB d DE	SET	3,(IX+d)
DD 8E d	ADC A,(IX+d)	DD CB d E6	SET	4,(IX+d)
DD 96 d	SUB (IX+d)	DD CB d EE	SET	5,(IX+d)
DD 9E d	SBC A,(IX+d)	DD CB d F6	SET	6,(IX+d)
DD A6 d	AND (IX+d)	DD CB d FE	SET	7,(IX+d)
DD AE d	XOR (IX+d)	DD E1	POP	IX
DD B6 d	OR (IX+d)	DD E3	EX	(SP),IX
DD BE d	CP (IX+d)	DD E5	PUSH	IX
		DD E9	JP	(IX)
		DD F9	LD	SP,IX

ED Instructions

ED 40 IN B,(C)	ED 50 IN D,(C)	ED 60 IN H,(C)		EDA0 LDI	EDB0 LDIR
ED 41 OUT (C),B	ED 51 OUT (C),D	ED 61 OUT (C),H		EDA1 CPI	EDB1 CPIR
ED 42 SBC HL,BC	ED 52 SBC HL,DE	ED 62 SBC HL,HL	ED 72 SBC HL,SP	EDA2 INI	EDB2 INIR
ED 43 (addr),BC	ED 53 (addr),DE	ED 63 (addr),HL	ED 73 (addr),SP	EDA3 OUTI	EDB3 OTIR
ED 44 NEG					
ED 45 RETN					
ED 46 IM 0	ED 56 IM 1	ED 66 IM 2			
ED 47 LD I,A	ED 57 LD A,I	ED 67 RRD			
ED 48 IN C,(C)	ED 58 IN E,(C)	ED 68 IN L,(C)	ED 78 IN A,(C)	EDA8 LDD	EDB8 LDDR
ED 49 OUT (C),C	ED 59 OUT (C),E	ED 69 OUT (C),L	ED 79 OUT (C),A	EDA9 CPD	EDB9 CPDR
ED 4A ADC HL,BC	ED 5A ADC HL,DE	ED 6A ADC HL,HL	ED 7A ADC HL,SP	EDAA IND	EDBA INDR
ED 4B LD BC,(addr)	ED 5B LD DE,(addr)	ED 6B LD HL,(addr)	ED 7B LD SP,(addr)	EDAB OUTD	EDBB OTRD
ED 4D RETI					
ED 4F LD R,A	ED 5F LD A,R	ED 6F RLD			

Appendix 3

Decimal-hexadecimal conversion table

Decimal 0–255 Hexadecimal 00–FF, Low byte

Dec.	Hex.	Dec.	Hex.	Dec.	Hex.	2's C.	Dec.	Hex.	2's C.
0	00	64	40	128	80	-128	192	C0	-64
1	01	65	41	129	81	-127	193	C1	-63
2	02	66	42	130	82	-126	194	C2	-62
3	03	67	43	131	83	-125	195	C3	-61
4	04	68	44	132	84	-124	196	C4	-60
5	05	69	45	133	85	-123	197	C5	-59
6	06	70	46	134	86	-122	198	C6	-58
7	07	71	47	135	87	-121	199	C7	-57
8	08	72	48	136	88	-120	200	C8	-56
9	09	73	49	137	89	-119	201	C9	-55
10	0A	74	4A	138	8A	-118	202	CA	-54
11	0B	75	4B	139	8B	-117	203	CB	-53
12	0C	76	4C	140	8C	-116	204	CC	-52
13	0D	77	4D	141	8D	-115	205	CD	-51
14	0E	78	4E	142	8E	-114	206	CE	-50
15	0F	79	4F	143	8F	-113	207	CF	-49
16	10	80	50	144	90	-112	208	D0	-48
17	11	81	51	145	91	-111	209	D1	-47
18	12	82	52	146	92	-110	210	D2	-46
19	13	83	53	147	93	-109	211	D3	-45
20	14	84	54	148	94	-108	212	D4	-44
21	15	85	55	149	95	-107	213	D5	-43
22	16	86	56	150	96	-106	214	D6	-42
23	17	87	57	151	97	-105	215	D7	-41
24	18	88	58	152	98	-104	216	D8	-40
25	19	89	59	153	99	-103	217	D9	-39
26	1A	90	5A	154	9A	-102	218	DA	-38
27	1B	91	5B	155	9B	-101	219	DB	-37
28	1C	92	5C	156	9C	-100	220	DC	-36
29	1D	93	5D	157	9D	-99	221	DD	-35
30	1E	94	5E	158	9E	-98	222	DE	-34
31	1F	95	5F	159	9F	-97	223	DF	-33
32	20	96	60	160	A0	-96	224	E0	-32
33	21	97	61	161	A1	-95	225	E1	-31
34	22	98	62	162	A2	-94	226	E2	-30
35	23	99	63	163	A3	-93	227	E3	-29
36	24	100	64	164	A4	-92	228	E4	-28
37	25	101	65	165	A5	-91	229	E5	-27
38	26	102	66	166	A6	-90	230	E6	-26
39	27	103	67	167	A7	-89	231	E7	-25
40	28	104	68	168	A8	-88	232	E8	-24
41	29	105	69	169	A9	-87	233	E9	-23
42	2A	106	6A	170	AA	-86	234	EA	-22
43	2B	107	6B	171	AB	-85	235	EB	-21
44	2C	108	6C	172	AC	-84	236	EC	-20
45	2D	109	6D	173	AD	-83	237	ED	-19
46	2E	110	6E	174	AE	-82	238	EE	-18
47	2F	111	6F	175	AF	-81	239	EF	-17
48	30	112	70	176	B0	-80	240	F0	-16
49	31	113	71	177	B1	-79	241	F1	-15
50	32	114	72	178	B2	-78	242	F2	-14
51	33	115	73	179	B3	-77	243	F3	-13
52	34	116	74	180	B4	-76	244	F4	-12
53	35	117	75	181	B5	-75	245	F5	-11
54	36	118	76	182	B6	-74	246	F6	-10
55	37	119	77	183	B7	-73	247	F7	-9
56	38	120	78	184	B8	-72	248	F8	-8
57	39	121	79	185	B9	-71	249	F9	-7
58	3A	122	7A	186	BA	-70	250	FA	-6
59	3B	123	7B	187	BB	-69	251	FB	-5
60	3C	124	7C	188	BC	-68	252	FC	-4
61	3D	125	7D	189	BD	-67	253	FD	-3
62	3E	126	7E	190	BE	-66	254	FE	-2
63	3F	127	7F	191	BF	-65	255	FF	-1

Appendix 3—continued

Decimal 0–65 280 Hexadecimal 00–FF, high byte

Decimal	Hex.	Decimal	Hex.	Decimal	Hex.	Decimal	Hex.
0	00	16 384	40	32 768	80	49 152	C0
256	01	16 640	41	33 024	81	49 408	C1
512	02	16 896	42	33 280	82	49 664	C2
768	03	17 152	43	33 536	83	49 920	C3
1 024	04	17 408	44	33 792	84	50 176	C4
1 280	05	17 664	45	34 048	85	50 432	C5
1 536	06	17 920	46	34 304	86	50 688	C6
1 792	07	18 176	47	34 560	87	50 944	C7
2 048	08	18 432	48	34 816	88	51 200	C8
2 304	09	18 688	49	35 072	89	51 456	C9
2 560	0A	18 944	4A	35 328	8A	51 712	CA
2 816	0B	19 200	4B	35 584	8B	51 968	CB
3 072	0C	19 456	4C	35 840	8C	52 224	CC
3 328	0D	19 712	4D	36 096	8D	52 480	CD
3 584	0E	19 968	4E	36 352	8E	52 736	CE
3 840	0F	20 224	4F	36 608	8F	52 992	CF
4 096	10	20 480	50	36 864	90	53 248	D0
4 352	11	20 736	51	37 120	91	53 504	D1
4 608	12	20 992	52	37 376	92	53 760	D2
4 864	13	21 248	53	37 632	93	54 016	D3
5 120	14	21 504	54	37 888	94	54 272	D4
5 376	15	21 760	55	38 144	95	54 528	D5
5 632	16	22 016	56	38 400	96	54 784	D6
5 888	17	22 272	57	38 656	97	55 040	D7
6 144	18	22 528	58	38 912	98	55 296	D8
6 400	19	22 784	59	39 168	99	55 552	D9
6 656	1A	23 040	5A	39 424	9A	55 808	DA
6 912	1B	23 296	5B	39 680	9B	56 064	DB
7 168	1C	23 552	5C	39 936	9C	56 320	DC
7 424	1D	23 808	5D	40 192	9D	56 576	DD
7 680	1E	24 064	5E	40 448	9E	56 832	DE
7 936	1F	24 320	5F	40 704	9F	57 088	DF
8 192	20	24 576	60	40 960	A0	57 344	E0
8 448	21	24 832	61	41 216	A1	57 600	E1
8 704	22	25 088	62	41 472	A2	57 856	E2
8 960	23	25 344	63	41 728	A3	58 112	E3
9 216	24	25 600	64	41 984	A4	58 368	E4
9 472	25	25 856	65	42 240	A5	58 624	E5
9 728	26	26 112	66	42 496	A6	58 880	E6
9 984	27	26 368	67	42 752	A7	59 136	E7
10 240	28	26 624	68	43 008	A8	59 392	E8
10 496	29	26 880	69	43 264	A9	59 648	E9
10 752	2A	27 136	6A	43 520	AA	59 904	EA
11 008	2B	27 392	6B	43 776	AB	60 160	EB
11 264	2C	27 648	6C	44 032	AC	60 416	EC
11 520	2D	27 904	6D	44 288	AD	60 672	ED
11 776	2E	28 160	6E	44 544	AE	60 928	EE
12 032	2F	28 416	6F	44 800	AF	61 184	EF
12 288	30	28 672	70	45 056	B0	61 440	F0
12 544	31	28 928	71	45 312	B1	61 696	F1
12 800	32	29 184	72	45 568	B2	61 952	F2
13 056	33	29 440	73	45 824	B3	62 208	F3
13 312	34	29 696	74	46 080	B4	62 464	F4
13 568	35	29 952	75	46 336	B5	62 720	F5
13 824	36	30 208	76	46 592	B6	62 976	F6
14 080	37	30 464	77	46 848	B7	63 232	F7
14 336	38	30 720	78	47 104	B8	63 488	F8
14 592	39	30 976	79	47 360	B9	63 744	F9
14 848	3A	31 232	7A	47 616	BA	64 000	FA
15 104	3B	31 488	7B	47 872	BB	64 256	FB
15 360	3C	31 744	7C	48 128	BC	64 512	FC
15 616	3D	32 000	7D	48 384	BD	64 768	FD
15 872	3E	32 256	7E	48 640	BE	65 024	FE
16 128	3F	32 512	7F	48 896	BF	65 280	FF

Appendix 4

Useful calculator literals

Literal d	Function	Action
1	EXCHANGE	Top two values swapped
3	SUBTRACT	Top value subtracted from the one underneath
4	MULTIPLY	Multiply top two values
5	DIVIDE	Top value divided into the one underneath
15	ADD	Add top two values
31	SIN	Sine of top value
32	COS	Cosine of top value
33	TAN	Tangent of top value
39	INT	Integer of top value
40	SQR	Square root of top value
49	DUPLICATE	Duplicate top value and place on top of stack
52	STK	Stack data held in compressed form (see Chapter 7)
56	END	Finish calculator routine and return to machine code program
61	RESTACK	Replace top value with its floating point form
160	0	Stack 0
161	1	Stack 1
162	$\frac{1}{2}$	Stack $\frac{1}{2}$
163	$\pi/2$	Stack $\pi/2$
164	10	Stack 10

Appendix 5

Memory map of display file

Line	Column																																				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					
0	4060 4100 4200 4300 4400 4500 4600 4700	4700 4800 4900 5000 5100	8	16	5200 5300 5400 5500 5600 5700																																
1	4020 4120 4220 4320 4420 4520 4620 4720	4720 4820 4920 5020 5120 5220 5320 5420 5520 5620 5720	9	17																																	
2	4140 4240 4340 4440 4540 4640 4740	4740 4840 4940 5040 5140 5240 5340 5440 5540 5640 5740	10	18																																	
3	4060 4160 4260 4360 4460 4560 4660 4760	4760 4860 4960 5060 5160 5260 5360 5460 5560 5660 5760	11	19																																	
4	4080 4180 4280 4380 4480 4580 4680	4780 4880 4980 5080 5180 5280 5380 5480 5580 5680	12	20																																	
5	4040 4140 4240 4340 4440 4540 4640 4740	4740 4840 4940 5040 5140 5240 5340 5440 5540 5640	13	21																																	
6	4000 4100 4200 4300 4400 4500 4600 4700	4700 4800 4900 5000 5100 5200 5300 5400 5500 5600 5700	14	22																																	
7	4100 4200 4300 4400 4500 4600 4700	4700 4800 4900 5000 5100 5200 5300 5400 5500 5600 5700	15	23																																	

Example : Fifth byte of character in line 13, column 27 = 4BA0 + 1B = 4B BB h

Appendix 6

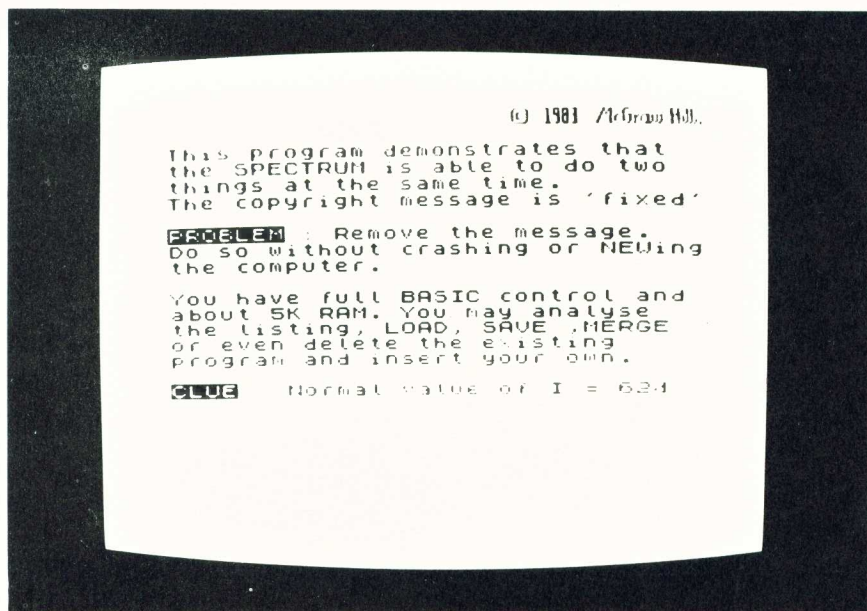
Farewell program

For those of you who like taking programs apart to see how they work, the tape contains a machine code program called PROBLEM. The object of the program is to demonstrate the fact that the Spectrum can run a machine code routine while giving full BASIC control to the programmer.

When loaded, the computer will display a copyright message in the top right-hand corner of the screen. This copyright message will remain on the screen at all times. Your problem is to find out how it is achieved and to remove the message without crashing or NEWing the Spectrum.

The program is such that it would be impossible to produce a useful listing and it has several features that make it difficult to trace the machine code. To remove the message you will need a suitable machine code routine.

If you successfully remove the message and disassemble the routine you will see that it opens up a whole new area of programming possibilities. Good luck.



Appendix 6—continued

```
10 RANDOMIZE USR (PEEK (PEEK 23627+256*PEEK 23628+1)+(256*(PEEK (PEEK 23627+256*PEEK 23628+2))))
20 CLEAR 29100
30 PRINT AT 2,0;"This program demonstrates that the SPECTRUM is able to do two things at the same time.";"The copyright message is 'fixed'"
35 PRINT
40 PRINT "PROBLEM : Remove the message.";"Do so without crashing
```

```
or NEWingthe computer."
50 PRINT
60 PRINT "You have full BASIC control and about 5K RAM. You may analyse the listing, LOAD, SAVE ,MERGE or even delete the existing program and insert your own."
70 PRINT "CLUE : Normal value of I = 62d"
80 STOP
90 LOAD ""CODE : GO TO 10
```

Program 'Problem'

Software to accompany this book . . .

A cassette tape, for 16K and 48K Spectrums, is available to go with this book. It includes many useful programs, as well as demonstrations of the techniques described in the book. And there is a full machine-code version of CROSS, an arcade game of professional quality!

Contents of the cassette tape are:—

- PLOTDEMO — plots messages anywhere on the screen, using BASIC to set height and width.
- REACTION — a reaction timer to demonstrate fast machine-code counting.
- RNDEMO — produces random height 'skyscrapers' using machine-code.
- KEYDEMO — machine-code fast keyscan.
- MOVEDEMO — demonstrates independent high-resolution foreground and background movement.
- PIXELDEMO — demonstrates movement at pixel level.
- NUMBERS — utility to print Sinclair floating-point form of any number.
- SOUND — five different machine-code sounds, including white noise.
- CROSS-BASIC — 'Cross' game in BASIC and machine-code.
- CROSS-M/C — Full machine-code version of 'Cross'.
- PROBLEM — If you've understood the book you should be able to erase the rather persistent McGraw-Hill logo that is always on the screen!

The tape is distinctively packed to match this book, and comes complete with a card enabling you to use the free McGraw-Hill software support service.

ARCADE PROGRAMMING

Just £6.95 inc. V.A.T.

(07 084730 4)

McGraw-Hill books and software should be available where you bought this book, but in case of difficulty send direct to:

**McGraw-Hill Book Company (UK) Limited,
Shoppenhangers Road, Maidenhead, Berkshire.**

Writing in Assembly Language is not enough!

It is not hard to learn Assembly Language, and writing machine code for the Spectrum is not difficult. Knowing *what* to write is another matter.

Professional Secrets

This is not a book for beginners. You are expected to have a working knowledge of Z80 Assembly Language, and of course a knowledge of BASIC. But for the first time, Stuart Nicholls describes the techniques of writing arcade games and other fast-moving programs for the Spectrum. Controlled high-resolution movement on the screen, separate movement of foreground and background, collision detection, randomizing and counting, are just a few of the subjects dealt with in this book. Routines for sound include "lasers", white noise (for explosions etc) and music.

A Treasure Chest for Programmers!

Complete, useful Assembly Language routines are given to demonstrate all the techniques described in the book. There are routines you can incorporate into your own programs. And there is even a selection of "fast ROM routines" that replace common Spectrum ROM calls with much faster equivalents.

Assembler Software

Any serious machine code programmer needs an Assembler. All programs in this book are written in standard Zilog Z80 mnemonics, and we recommend the McGraw-Hill *ZX Spectrum Machine Code Assembler*, which will assemble the programs in exactly the form in which they are given.

Learning Assembly Language

If you are new to Assembly Language, *Learn and Use Assembly Language on the ZX Spectrum* by Tony Woods (McGraw-Hill, 1983) makes ideal preparatory reading.

McGRAW-HILL Book Company (UK) Limited

MAIDENHEAD · BERKSHIRE · ENGLAND

07 084729 0