sinclair

Sinclair Research Limited
Milton Hall · Milton · Cambridge CB4 4AE
Telephone 0223 862661 · Telex 817949

SINCLAIR ZX SPECTRUM 128 TECHNICAL DOCUMENTATION

INFORMATION FOR SOFTWARE HOUSES AND PERIPHERALS MANUFACTURERS

Issued by the Software and Peripherals
Group, Marketing Department

Revision 3, February 18.2.86

Registered Office: Milton Hall · Milton · Cambridge CB4 4AE
Registered No. [illegible] · VAT Registration No. [illegible]

Spectrum 128 Technical Guide

Released 18.2.86

1 INTRODUCTION

This document is a brief description for software developers of the main features of the Spectrum 128 microcomputer.

* * * * * * * * * * * * * * * * * *

2 HARDWARE SPECIFICATION

The Spectrum 128 is a Z80A based microcomputer with 128K of bank switched Ram and 32K Rom memory. Its external interfaces and outputs comprise.

1) Cassette port	As for Spectrum Plus
2) TV output	
3) Expansion bus	
4) RS-232/Midi-out port	Spectrum 128 specific
5) RGB-monitor output	
6) Keypad	
7) TV sound	

A major consideration of the Spectrum 128 design has been to make it as far as possible software and hardware compatible with the Spectrum. Where information is not given the system will behave in the same fashion as the Spectrum eg screen layout, cassette data format.

* * * * * * * * * * * * * * * * * *

3 MEMORY MAP

The Spectrum 128 contains 32K Rom and 128K Ram arranged in 16K byte pages.

The two ROM pages (0-1) are mapped into the bottom 16K (0-3FFF) of the Z80 memory map.

The eight RAM pages (0-7) are mapped into the top 16K (C000-FFFF) of the memory map, RAM page 5 is also mapped to the range 4000-7FFF while RAM page 2 is mapped to the range 8000-BFFF.

It is thus possible, though not very useful, to have the same RAM page mapped into two different address spaces within the Z80.

The RAM pages are divided into a video contended section, pages 4 to 7, and a non contended section pages 0 to 3.

The Spectrum 128 has two possible hardware screen bases, screen 0 and screen 1. Screen 0 is used by the system and resides in RAM page 5 corresponding to the normal Spectrum screen. The system software does not support the use of the second screen base, screen 1, which is realistically available only to machine code applications programs. The second screen resides in RAM page 7 and thus maps, when paged into the Z80 address space, onto memory address C000.

	Physical Memory			Z80 Memory	
RAM	page 7	screen 1	contended editor	Paging	Address
	page 6		contended	Ram page 0-7	C000-FFFF
	page 5	screen 0	contended	Ram page 2	8000-BFFF
	page 4		contended	Ram page 5	4000-7FFF
	page 3		uncontended	Rom page 0-1	0000-3FFF
	page 2		uncontended		
	page 1		uncontended		
	page 0		uncontended basic		
ROM	page 1	Spectrum			
	page 0	Editor			

* * * * * * * * * * * * * * * * * *

4 I/O MAP

The Spectrum i/o addresses reserve A4-A0 for use by Sinclair Research. These are active low and decoded by the presence of a single signal. Unused addresses in the range A4-A0 should be set high.

Line	Address	Function
A0	FE	Spectrum keyboard, cassette, loudspeaker and border.
A1	FD	Spectrum 128 paging, screen selection, sound and i/o.
A2	F8	ZX Printer
A3,A4	F7/EF	Interface One

The following addresses are only found on the Derby.

7FFD	D2-D0	Ram Page select
	D3	Screen select
	D4	Rom select

	05	lock (become a Spectrum)
BFFD	Sound chip -	data register write (register dependent)
FFFD	Sound chip -	data read (register dependent)
	-	address write 000XXXXX, where XXXX is the register selected 0-F

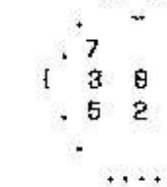
16-bit i/o is acheived on the Z80 by using the instruction OUT N,(C) which sends the value in register N to the 16-bit i/o address in BC.

* * * * * * * * * * * * * * * * * *

5 CONNECTOR SPECIFICATION

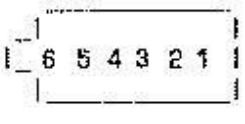
There are three connectors on the 128K Spectrum whose connections require additional information for use in interfacing external equipment.

1. RGB connector
This is the 8-way DIN (type 45326) RGB socket, viewed looking into the rear of the computer,



Pin	Signal	Level
1	Composite PAL	75ohms, 1.2 V p-p
2	0 Volts DC	
3	Bright	TTL
4	Composite sync	TTL
5	Vertical sync	TTL
6	Green	TTL
7	Red	TTL
8	Blue	TTL

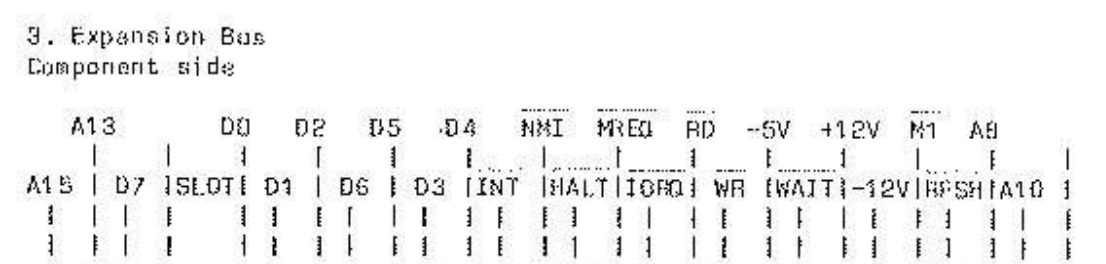
2. Serial/Midi connector
This is a BT-type socket, viewed from the left hand side of the machine, looking towards the socket.



Pin	Name	Function
1	GND	0 volt reference
2	TXD	input, data to computer
3	RXD	output, data from computer
4	DTR	input, flow control to computer
5	CTS	output, flow control from computer
6	+12V	power

Lines 2-5 all operate at 5V levels.The signal names seem to contradict their functions, but this is because the Spectrum 128 is acting as a DTE rather than a DCE.

3. Expansion Bus
Component side



Underside

* * * * * * * * * * * * * * * * * *

6 SOUND CHIP

The sound generator on the Spectrum 128 is the General Instruments AY-3-8912. This device has three sound channels and an 8-bit i/o port which is used to control the RS232/MIDI port and keypad.

The sound chip contains sixteen registers which are selected by writing first to the address write port with the number of the register and then writing or reading from the data write or data read ports. Once a register has been selected with an address write, it can be accessed repeatedly by data read/writes until a new register is selected.

The basic clock input to the sound chip is at 1.7734(476)MHz accurate 1 in 10+4

The registers have the following properties

R0 - Fine Tone Control for Channel A
R1 - Coarse Tone Control for Channel A
R2 - Fine Tone Control for Channel B
R3 - Coarse Tone Control for Channel B
R4 - Fine Tone Control for Channel C
R5 - Coarse Tone Control for Channel C

The tone is a 12-bit value taken from the sum of R1,D3-D0 and R0,D7-D0. The extra bits in R1,D7-D4 are unused. The sound clock is divided by 16 to get the basic frequency unit for the Tone and Noise registers. With a twelve bit counter range frequencies in a range from 27Hz to 100kHz can be generated.

R6 - Noise Generator Control, D4-D0

The period of the noise source is taken by counting down the lower 5 bits of the noise register every sound clock period divided by 16.

R7 - Mixer and i/o control

This register controls both the mixing of noise and tone values for each channel and the direction of the 8-bit i/o port. A zero indicates the enabling of a tone or noise source.

D7 - not used
D6 - (1) input port, (0) output port (normally in output mode,bits 4-7 always set)

D5 - Channel C Noise ----
D4 - Channel B Noise |
D3 - Channel A Noise |
|--- A zero enables each channel
D2 - Channel C Tone |
D1 - Channel B Tone |
D0 - Channel A Tone ----

R8 - Amplitude Control Channel A
R9 - Amplitude Control Channel B
RA - Amplitude Control Channel C

These three registers control the amplitude of the sound and whether it is modulated by the envelope registers.

D4 - if one then the amplitude of the channel is taken from the envelope generator else the amplitude is taken from the value 0-min to 15-max in D3 to D0.

D3-D0 - amplitude of sound channel

RB - Envelope Coarse Period Control
RC - Envelope Fine Period Control

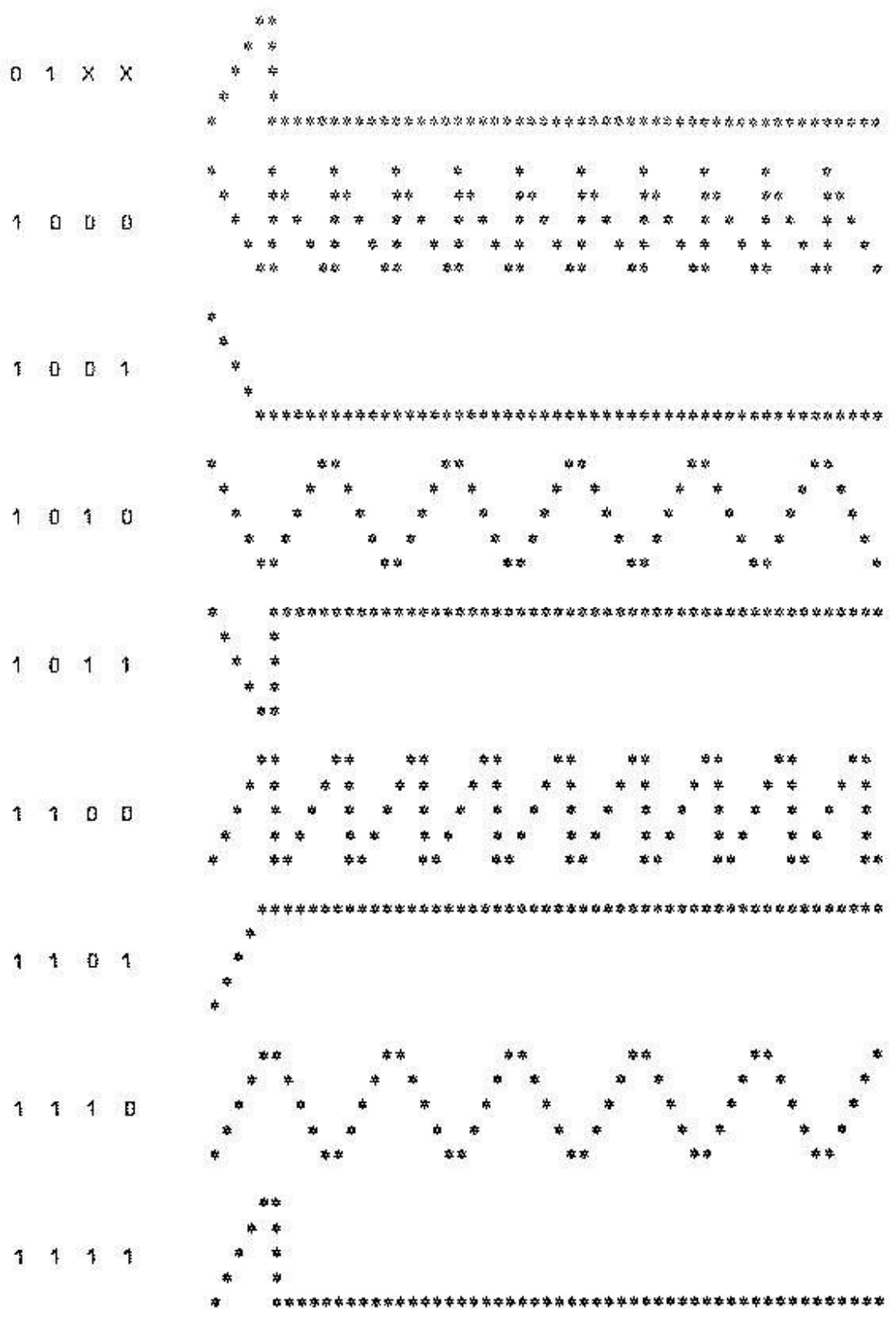
The 8-bit values in RB + RC are summed to generate a 16 bit number which is counted down in units of 256 times the sound clock. Envelope frequencies can be obtained thus between .1Hz and 6000Hz.

R0 - Envelope shape and cycling control

The lower four bits of this register control various functions of the envelope generator. They represent the following functions

D3 - Continue
D2 - Atteck
D1 - Alternate
D0 - Hold

Graphically they have the following effects on the envelope.

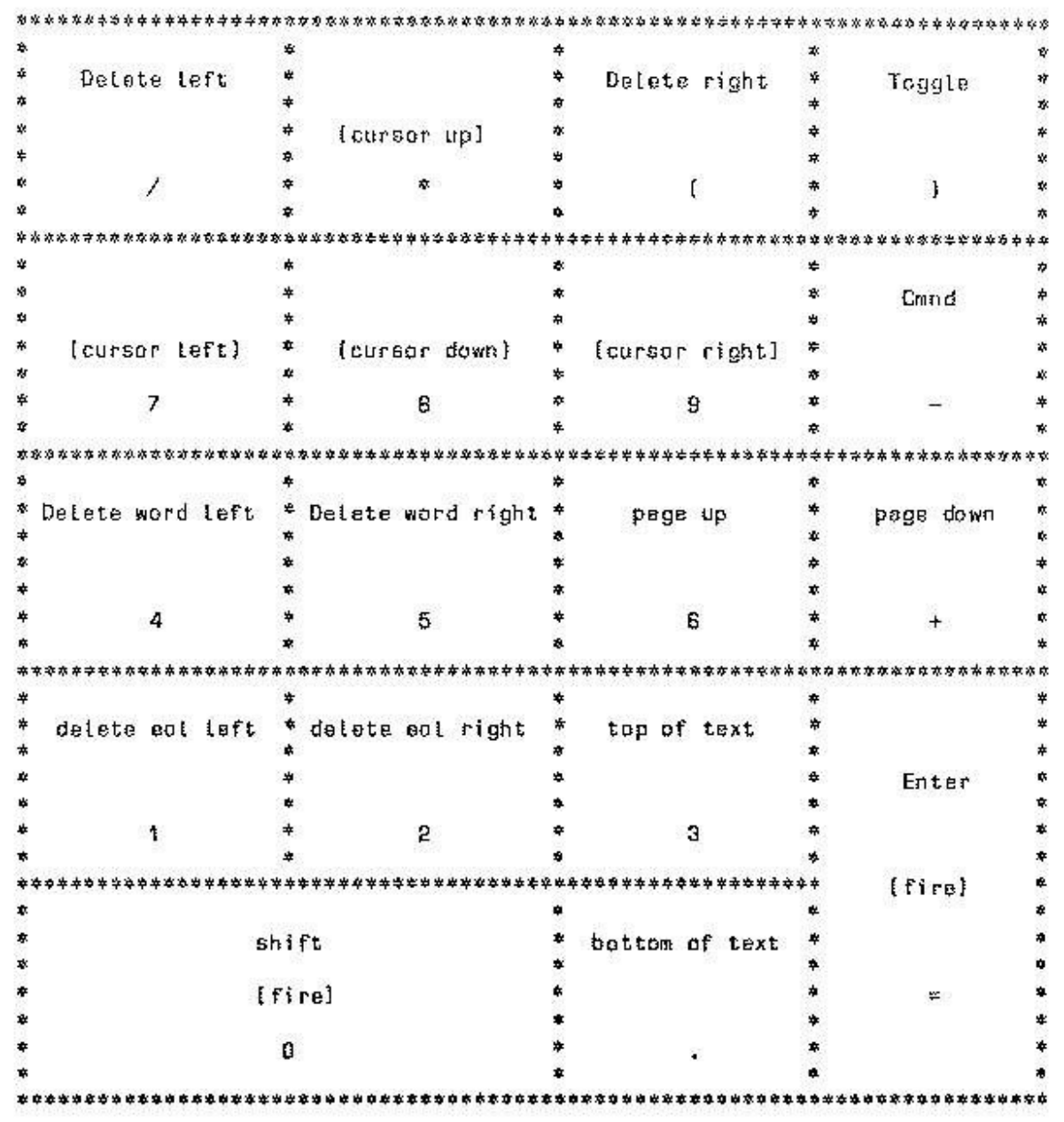

0 1 X X
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1

* * * * * * * * * * * * * * * * *

7 KEYPAD LAYOUT

The keypad has the following legends inscribed on its keys.

Top row ------ editor commands
Middle row ------ games keys
Bottom row ------ numeric keypad



It connects to the Spectrum 128 over a special bi-directional link, using a protocol which due to its complexity (and likely changing nature) we will not document but instead give the routine KPSCAN for its use. This has a ROM vector in the Editor rom at address 118H (see section on Rom vectors).

* * * * * * * * * * * * * * * * * *

8 SOFTWARE ORGANISATION

The Editor Rom contains the routines associated with the program editor, music control (PLAY), RS-232 and Ram disk. The Spectrum Rom contains the Basic language and the keypad routines.

Ram page 7 contains the ram storage for the editor and ram disk directory.

Ram page 0 is the ram used for program storage by Basic.

In normal operation the Spectrum 128 starts on power up in the Editor Rom and Editor Ram. While a Basic program is being entered the screen display information is held in the editor ram page, when a line is terminated or moved off in the editor, after being altered the memory map is changed to Spectrum Rom and Basic Ram, the line is passed to the Spectrum syntax handler and if it passes this check it is either entered into a program or if it is a direct command, it runs. While a program is running the computer is in the Spectrum Rom and Basic Ram, on termination it returns to the Editor Rom and Editor Ram and reenters the screen editor which then allows the user to modify the program.

The result of this switching back and forth is that the Basic user when peeking and poking within the machine will only have access to the Spectrum Rom and Basic Ram.

Great care should be taken if a program indulges in paging of Rom or Ram to make sure that on return to Basic the system is set up as it was left. A program which pages Ram should also make sure that it hasn't paged out its stack space.

A Silicon disc facility is available to the Basic programmer, allowing him to save files into the Ram pages which are normally unused except by machine code programs. Entry points to this facility are not available to machine code programmers, on the assumption that they will be able to make faster and more economical use of the additional Ram space by use of code designed specifically for their application. Access from Basic is by use of the standard filing system commands with an exclamation mark(!) following the initial command. The only command not supported is VERIFY.

```
eg      LOAD ! "makpaint" CODE
        SAVE ! "demopic" SCREEN$
        MERGE ! "basprog"
```

* * * * * * * * * * * * * * * * * *

9 ROM VECTORS

In the Spectrum 128 Rom there are a number of useful Rom vectors, suitable for use by machine code programs. No other routines in the Editor Rom should be used by programmers as the contents and organisation of the rom are liable to change.

Routine	Address	Function
KPSCAN	11BH	Scan the Spectrum 128 keypad
MSHOOK	11BH	Play music strings
MIDI_SEND	11EH	Send a MIDI character
RSIN	121H	Receive an RS232 character
OUT_T	124H	Send a token to RS232
OUT_T2	127H	Send a character to RS232
SCRDMP	12AH	Epson compatible RS232 screen dump

These routines will corrupt all unused registers

```
*****************************************
*    KPSCAN - EDITOR ROM 11BH           *
*****************************************
```

call this routine to service the keypad

NB interrupts must be disabled on entry

exit conditions are as follows :

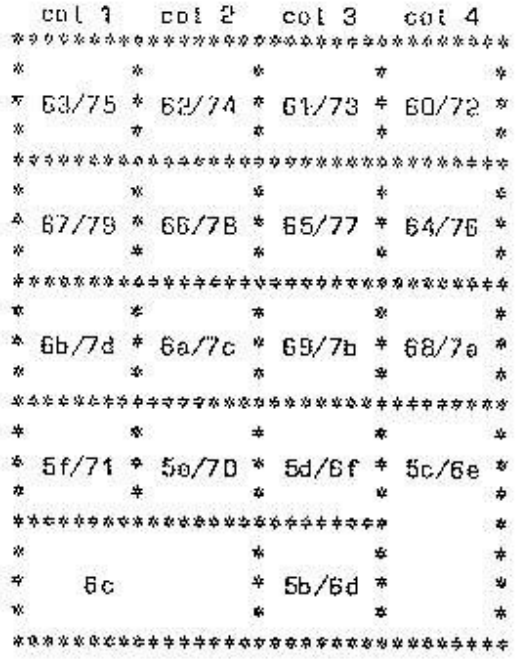
ROW01, ROW23, and ROW45 are left holding the instantaneous keypad image

z flag is clear: something's wrong
- no pp device is connected
- no keypad is connected
- more than 1 key is pressed (not including 0/SHIFT)

z flag is set: everything's ok,
an intermediate key code is returned in E
if no key was pressed then E is 10000000
else E is 0kkkkkkk

intermediate key codes are returned as follows :

calculator legend	row	col	key only	key and 0/SHIFT
0	1	1		6c (108)
.	1	3	5b (91)	6d (109)
ENTER	2	4	5c (92)	6e (110)
3	2	3	5d (93)	6f (111)
2	2	2	5e (94)	70 (112)
1	2	1	5f (95)	71 (113)
)	5	4	60 (96)	72 (114)
(	5	3	61 (97)	73 (115)
*	5	2	62 (98)	74 (116)
/	5	1	63 (99)	75 (117)
-	4	4	64 (100)	76 (118)
9	4	3	65 (101)	77 (119)
8	4	2	66 (102)	78 (120)
7	4	1	67 (103)	79 (121)
+	3	4	68 (104)	7a (122)
6	3	3	69 (105)	7b (123)
5	3	2	6a (106)	7c (124)
4	3	1	6b (107)	7d (125)



****
* MSHOOK - EDITOR ROM 118H *
****

This entry point allows access to the music system from machine code. It requires strings to be set up in the same form as the Basic PLAY command.

The data structure required is described below.

Index registers

The music routines in the Editor Rom use two control blocks accessed indirectly via IY and IX.

IY points to a control block which contains 'system' information about all the strings that are currently being interpreted. This block must be at least 60 bytes long. The variables in this block are :

Offset

```
CT_CHAN0        EQU     0               ; value of IX for channel 0
CT_CHAN1        EQU     CT_CHAN0+2      ; --------||------------ 1
CT_CHAN2        EQU     CT_CHAN1+2      ; --------||------------ 2
CT_CHAN3        EQU     CT_CHAN2+2
CT_CHAN4        EQU     CT_CHAN3+2
CT_CHAN5        EQU     CT_CHAN4+2
CT_CHAN6        EQU     CT_CHAN5+2
CT_CHAN7        EQU     CT_CHAN6+2
CT_FLAGS        EQU     CT_CHAN7+2      ; Flags
                                        ; bit i set if string i finished
CT_Q0           EQU     CT_FLAGS+1      ; queue pointer for channel 0
CT_Q1           EQU     CT_Q0+2         ; --------||------------ 1
```

```
CT_Q2       EQU     CT_Q1+2         ; ----------||---------- 2
CT_Q3       EQU     CT_Q2+2
CT_Q4       EQU     CT_Q3+2
CT_Q5       EQU     CT_Q4+2
CT_Q6       EQU     CT_Q5+2
CT_Q7       EQU     CT_Q6+2
CT_CHAN     EQU     CT_Q7+2         ; temp store for chan indicator
CT_TEMP     EQU     CT_CHAN+1       ; temp store for copy of flags
CT_QTEMP    EQU     CT_TEMP+1       ; temp storage for pointer to Q ptr
CT_EVENT    EQU     CT_QTEMP+2      ; length of current event in T states
CT_TEMPO    EQU     CT_EVENT+2      ; no of dec BC loops for 1 T state
CT_ENV      EQU     CT_TEMPO+2      ; current envelope shape/cycle byte
CT_MIXT     EQU     CT_ENV+1        ; temporary mixer mask
CT_CODE     EQU     CT_MIXT+1       ; RAM code for tempo adjustment
```

IX points to a buffer for the string currently being processed. For 8 strings there will be 8 of these buffers and the higher level software switches the value of IX between them. Note that the values of IX are stored at the start of the IY control buffer. An IX buffer must be at least 55 bytes long. The buffer variables are :

```
MV_CURR     EQU     0               ; current MIDI note
MV_MIDI     EQU     MV_CURR+1       ; MIDI channel number (byte)
MV_CHAN     EQU     MV_MIDI+1       ; channel number (0,1 or 2) (byte)
MV_OCTAVE   EQU     MV_CHAN+1       ; current octave (byte)
MV_VOL      EQU     MV_OCTAVE+1     ; current volume (byte)
MV_NOTE     EQU     MV_VOL+1        ; current note code (byte)
MV_ADD      EQU     MV_NOTE+1       ; pointer to next char (word)
MV_END      EQU     MV_ADD+2        ; pointer to end of string (word)
MV_REPEAT   EQU     MV_END+2        ; pointer to last repeat (word)
MV_FLAG     EQU     MV_REPEAT+2     ; misc flags (byte)
MV_OPEN     EQU     MV_FLAG+1       ; open bracket stack(byte+5 words)
MV_CLOSE    EQU     MV_OPEN+11      ; close bracket stack(byte+5 words)
MV_PEND     EQU     MV_CLOSE+11     ; notes in queue (byte)
MV_QUEUE    EQU     MV_PEND+1       ; start of queue (20 words)
```

The string interpreter

In order to provide string interpretation for machine code programmers an entry point to the code has been provided at the global MUSIC_HOOK. On entry to this point the calling code must have set up a control block at IY and music buffers for each string to be interpreted (up to a maximum of 8 strings). The control block must have the following parameters set :

```
(IY+CT_CHAN0)   must contain the value of IX for the first string.
(IY+CT_CHAN1)   ----------------------------------2nd----------,
                        etc up to 8 strings if necessary.
(IY+CT_FLAGS)   must have reset bits for strings to be played and
                set bits for absent strings
                Bit 0 is the first string etc.
```

On entry the code will set up the default tempo to 120 crotchets per min.

Each music buffer must have the following parameters set :

(MV_MIDI)	OFFH if MIDI output is not required otherwise the MIDI channel number (0..15)
(MV_CHAN)	The channel number for this string. The first string is channel 0 and so on.
(MV_OCTAVE)	The default octave for the base of the 2 octave range. A value 5 gives note code c as middle C.
(MV_VOL)	The volume for the GI chip on this channel (0..15)
(MV_NOTE)	The default note type [5=crotchet]
(MV_ADD)	16 bit address of the first code in the string. This value should also be copied to (MV_REPEAT) if repeat is required to start from the beginning of the string.
(MV_END)	Pointer to the next byte after the end of the string.
(MV_OPEN)	Must contain 0 (byte)
(MV_OPEN+1)	Must contain the 16 bit start address of the string.
(MV_CLOSE)	Must contain OFFH.

On entry to the code interrupts must be disabled. The code will execute a RET on correct termination of all strings. Any errors will jump to the internal error handling routine and control will be lost. The routine will corrupt all normal registers. IY will be returned as the correct pointer to the system variables. The alternate register set is unaffected.

```
*****************************************
*  MIDI_SEND   EDITOR ROM  11E          *
*****************************************
```

This routine outputs a byte to the RS-232 port acting as a MIDI-OUT port

The MIDI output routine is accessed via a call to the global address MIDI_SEND. The byte to be sent is in A and is sent immediately. The routine corrupts A,BC,DE and L. Interrupts must be off to ensure correct timing.

```
*****************************************
*  RSIN        EDITOR ROM  121          *
*****************************************
```

RS-232 Receive character routine

If a character is received then C-flag is set and the character is returned in A, else no character C-flag clear

The receive system expects 8-data bits, no parity and 1 stop-bit.

Corrupts all registers

```
*****************************************
*  OUT_T       EDITOR ROM  124          *
*****************************************
```

RS-232 Send a token routine

This routine takes as input a token in A

The baud rate is set up by poking into the Editor system variable BAUD a value equivalent to a bit time in T-states divided by 26. From Basic it can be set up with the command FORMAT "P";speed.

This routine sends data in the format, 8-data, no-parity, 2 stop bits.

Corrupts all registers

```
*****************************************
*  OUT_T2      EDITOR ROM  127          *
*****************************************
```

RS-232 Send a character routine

This routine takes as its input a character in A

The baud rate is set up by poking into the Editor system variable BAUD a value equivalent to a bit time in T-states divided by 26

This routine sends data in the format, 8-data, no-parity, 2 stop bits.

Corrupts all registers

```
*****************************************
*  SCRDMP      EDITOR ROM  12A          *
*****************************************
```

Dump the screen image to an Epson-compatible printer

Corrupts all registers

This takes no parameters and sends a bit image of the current screen to an Epson compatible printer over the RS-232 interface.

The baud rate is set up by poking into the Editor system variable BAUD a value equivalent to a bit time in T-states divided by 26

* * * * * * * * * * * * * * * * * *

10 MUSIC AND SOUND

This section describes how to use the Spectrum 128 music system from Basic, this will allow you to experiment with setting up sound effects. The strings used for machine code programs are identical in format to the Basic strings described below for the PLAY command.

The Spectrum 128 has two different ways to make music and sound effects. Both work through the sound channel of a TV or can be sent to an external amplifier via the ear-phone socket on the cassette interface. There is no internal speaker.

The first method uses the Spectrum BASIC BEEP command. But 128 BASIC also has the PLAY command which allows you to make sophisticated music with up to three notes at once. It also gives you more control over the sound of each individual note than is possible using BEEP.

The PLAY command can be used to let your Spectrum 128 play tunes on many types of synthesizer and other electronic musical instruments, such as drum machines. The Spectrum 128 is MIDI OUT (Musical Instrument Digital Interface) compatible, which allows it to be connected to any other equipment which conforms to the MIDI IN standard.

Programming sounds

Making music and sound effects with PLAY is simple. You just type in the series of notes that make up a tune, then ask the Spectrum 128 to PLAY them. You can also include instructions that tell the Spectrum 128 what sort of tone you want for the sound.

To hear some of the wide range of sounds that you can make, type in each of the two programs below, RUN it, then try the other example. In the following examples where it can be confused with letter 'O', zero is shown with an underline (eg '0')

MUSIC

```
10 LET
a$="T180O6[CDEC][5EF7G][3GAGF5EC]5Cg7C9CgC"
20 LET
b$="O4[CDEC][5EF7G][3GAGF5EC]5Eb7E9EbE"
30 LET c$="O3[7CG][7CG][7CG]5GD7G9GDG"
40 PLAY a$,b$,c$
```

SOUND EFFECTS

```
10 LET a$="M8UX350W507[[[C]]]":PLAY
a$:PAUSE 25
20 LET a$="M56UX5000W103[[[C]]]":PLAY
a$:PAUSE 25
30 LET a$="M56W201N8C":PLAY a$:PAUSE 25
```

Using the PLAY command

In the examples above, you will see that each time the PLAY command is used, it

is followed by up to three different letters, each followed by a $ sign in a statement like

```
PLAY a$,b$,c$
```

Each of these is the name of a string (a series of characters) which you have already given to the computer earlier in the program. The strings tell the Spectrum 128 which sounds to make.

PLAY controls three separate sound channels called A, B, and C and there can be up to three strings, one for each channel. In the MUSIC example given above, a$ tells channel A to play the melody line, b$ tells channel B to play a harmony, and c$ tells channel C to play a bass part. In the SOUND EFFECTS example, only one noise is used at a time (although up to three can be), so each one is in channel A and the command is simply PLAY a$.

In fact any of the channels can produce either a musical tone or noise, so you can mix sound effects in with your music (see Channel selection).

Constructing strings

Composing music and sound effects on the Spectrum 128 is simply a matter of creating strings containing the information you want. You can see how this works in the MUSIC example above. Each string is created with the LET command, followed by the name of the string, and its contents enclosed in "" marks. Try this example, which plays just one note - an A.

```
LET a$="a":PLAY a$
```

Any music program using PLAY also uses LET to tell it what to play, as you can see by looking at the earlier examples. The reason why these programs look more complicated than the example above is that the strings enclosed in the "" marks contain many letters and numbers to define a longer tune or more complex sound.

Any musical sound has a pitch and duration. It also has a volume and tone quality. The strings in the examples above contain information about all of these. The summary opposite lists each possible command, and they are explained in detail below.

PLAY command summary

This is a brief list of the commands which can be contained in a PLAY string. All letters except note names must always be in capitals.

String entry	Function
c-b or C-B	Gives pitch of note within current octave range
$	Flattens note following it
#	Sharpens note following it
O	Followed by number 0 to 8 sets current octave range

1-12	Sets length of notes
&	Denotes a rest
N	Separates two numbers
V	Followed by a number 0 to 15 sets volume of notes
W	Followed by a number 0 to 7 sets volume effect
U	Turns on volume effect in any string
X	Followed by a number 0 to 65535 sets duration of volume effect
T	Followed by a number from 60 to 240 sets tempo of music
( )	Enclose repeated phrase
! !	Enclose a comment
H	Stops a PLAY command
M	Followed by a number from 1 to 63 selects channels
Y	Followed by a number from 1 to 16 turns on a MIDI channel
Z	Followed by a number sends that number as a MIDI programming code

Setting the pitch

As you saw above, you set the pitch of any note by giving its musical name - eg, C E G. Sharp notes are prefixed by # (eg #C) and flat notes by $. Your 'instrument' covers two octaves in the key of C, and uses letters c to b for the notes in the lower octave, C to B in capitals for the higher one.

Any number of notes within these two octaves can be played one after another, eg

```
10 LET a$="cfedafgCFEDAFGCC"
20 PLAY a$
```

If you want to span more than just two octaves, you can change the overall pitch of your 'instrument' by using the octave command O followed by a number from 0 to 8. If you do not specify an octave (as in the example above) it is automatically set to 5 (the range containing middle C). The octave command remains in force for all notes which follow it until a new octave command is given.

This program lets you hear the same tune played in a higher octave (just add the 07 to your earlier program)

```
10 LET a$="07cfedafgCFEDAFGCC"
20 PLAY a$
```

Try changing the octave number progressively to hear the full pitch range which your Spectrum 128 can produce. Notice that some of the very lowest notes in octaves 0 and 1 will not be reproduced at the right pitch except through the MIDI output. The Spectrum 128 will simply play them at the lowest pitch possible.

Since each pitch range covers two octaves, two adjacent ranges overlap. For example, the high part of 04 contains the low part of 05 (see the diagram below). Note that you can extend any range slightly by using a series of sharps (####) or flats ($$$$) to raise the pitch of individual high notes or lower the pitch of low ones.

Note duration

If you do not specify the length of each note, they will all be played at the same length (as crotchets) as in the examples above. You can fix the length of any note or series of notes by prefixing it with a number from 1 to 12. This program lets you hear the different note duration with numbers from 1 to 9.

```
10 LET a$="1C2C3C4C5C6C7C8C9C"
20 PLAY a$
```

1 is the shortest note and 9 the longest. They are related to musical convention according to the following table.

Number	Note Name
1	semi-quaver
2	dotted semi-quaver
3	quaver
4	dotted quaver
5	crotchet
6	dotted crotchet
7	minim
8	dotted minim
9	semi-breve

Each of these controls the length of all notes which follow it until you give a new number code.

You can also use the numbers from 10 to 12 to specify triplet notes (three notes played in the time normally used for two).

10	triplet semi-quaver
11	triplet quaver
12	triplet crotchet

Each of these only applies to the three notes following it and must be followed by their three names, eg

```
10 LET a$="11ACE"
```

A rest (no note playing) is specified by a & and has the same length as the current note. For example

```
10 LET a$="7A&&8C&&&E"
```

is five minims with equal pauses between them.

Tied notes can be indicated by giving the two note durations connected by an underscore character and the note name, eg

```
10 LET a$="3_5A"
```

The second note duration you give will also apply to any following codes until you give another duration code.

The N command

In some of the examples you will see the letter N used to introduce a series of notes within the string, eg

```
LET a$="O7N1CDE"
```

N is used in cases where two sets of numbers would otherwise clash. In the example above, O is set to octave 7, then a series of notes is given, starting with the duration code 1.

Without the N code, the Spectrum 128 would read the octave code as 71 - obviously not what was intended!

Note volume

The overall volume of the sound is controlled by the volume setting of your TV or amplifier. You can control the volume of individual notes and phrases within the tune by using the V command. V followed by a number from 0 to 15 sets the following note(s) to a constant volume. The lower the number, the quieter the sound, with V0 completely silent (V0 is a useful way of stopping one channel playing while others continue). V15 is the maximum and is used automatically by the Spectrum 128 if you do not specify a level.

The low volumes are very quiet and you will normally use 10 to 15 unless you are outputting to an amplification system or via the MIDI port to a synthesizer. Try running this program

```
10 LET a$="V10cdefgabCDEFGAB"
20 PLAY a$
```

Now try changing the number after the V to a new value to hear the difference

Volume effects

Instead of you just setting each note to a fixed volume, PLAY also lets you change the volume of the sound while it is playing. For example, you can make a note start suddenly and then die away (like a piano) or make a sound effect rise and fall in volume (like a steam train).

This effect is controlled by the letter W which can be included in any of the strings controlled by the PLAY command. You must also include the letter U in each string where you want to use the effect. You cannot use it if the string

already has a volume setting (if it contains a V) - the volume command will overrule the effect.

The W must be followed by a number from 0 to 7 which controls how the sound builds up (the attack) or falls off (the decay). This is the full range of numbers and what they do

```
0 single decay then off
1 single attack then off
2 single decay then hold
3 single attack then hold
4 repeated decay
5 repeated attack
6 repeated attack-decay
7 repeated decay-attack
```

This program plays the same note with each effect in turn to let you hear what they sound like

```
10 LET a$="UX1000W0C&W1C&W2C&W3C&W4C&W5C&W6C&W7C"
20 PLAY a$
```

Notice the U to turn on the effect, then the series of W numbers.

There is one other new command used here, the letter X. This can be followed by a number from 0 to 65535 to set the length of the sound effect - the larger the number, the longer the effect is drawn out.

You do not have to include an X setting. If you do not, the Spectrum 128 will automatically choose the longest. In general, repetitive effects (W4 to 7) are more effective with quite short settings, eg X300. 'Single shot' effects (W0 to W3) need a longer period, eg X1000. Try changing the value after X in the program above to hear the difference.

Tempo

The speed at which a piece of music is played can be set with the command T followed by the number of crotchet beats per minute (bpm) in the range 60 to 240. The command controls the speed at which all notes are played, but can only be included in channel A (the first string after the PLAY command) otherwise it is ignored, eg

```
10 LET a$="T180cdefg"
20 PLAY a$
```

If no tempo is specified, the music will be played at 120 bpm.

Repeated phrases

Any musical phrase can be repeated by enclosing the appropriate string or part of a string in brackets. For example

```
10 LET a$="abC(DEFG)"
```

will repeat the last four notes. If there is an unequal number of brackets, the phrase will be repeated back to the last bracket. If there is only a closing

bracket, the phrase will be repeated back to the beginning of the string, eg

```
10 LET a$="abCDEFG)"
```

will repeat all seven notes. Double closing brackets

```
10 LET a$="G2CEGA))"
```

will cause an 'infinite' repeat. This is particularly useful for things like repetitive bass lines. To turn off an 'infinite' repeat you use the H command.

The H command

An H included in any string immediately turns off the PLAY command. The main use of this is where you have an infinitely repeated bass line in one string. You can stop this at the end of the tune by putting an H on the end of the string which plays the melody.

Comments

You can include reminders and comments anywhere you like by using !! marks. Anything written after a ! will be ignored until the next ! or the " at the end of the string is reached, eg

```
10 LET a$="abCDEFG!chorus!aCEaDG"
```

Channel selection

The command M is used to select which of the three channels are in operation and whether these give noise or musical tones.

You can have a maximum of three channels in use at any one time, but it does not matter whether they are all tone, all noise, or a mixture of both.

Your choice is entered with a number following the M, worked out like this

	Tone channels			Noise channels		
	A	B	C	A	B	C
Number	1	2	4	8	16	32

Mark each channel you want to turn on, and note down its number. Then just add them together to get the code you should use after the M. For example, if you want to use tone channels A, B, and C, you add the numbers 1+2+4=7, so you use the command M7. In the same way, M56 would turn on noise channels A, B, and C.

Noise can be used on any channel but the most wide-ranging frequencies are available in channel A. For the best results put your sound effects in the string which controls this channel - a$, the first string after the PLAY command.

Controlling musical instruments

Whenever the PLAY command is running, a signal can be sent to the MIDI port in the RS232 socket at the front of the computer. This output will drive any MIDI

compatible musical instrument such as many makes of synthesizer and drum machine, so that the instrument will play the music which has been programmed into the Spectrum 128.

Using the MIDI output lets you play more complicated music with up to eight notes at a time instead of three. All you do is to follow the PLAY command with the names of up to eight strings, (a$ to h$, for example) each of which is constructed as described above.

Do not try to send music to the MIDI port unless you have already connected the instrument. To send the output to the MIDI port, each string should include the letter Y followed by a channel number from 1 to 16. If you use the same number in each string, up to eight notes can be played by one instrument at the same time. If you use different numbers, you can control up to eight different instruments at the same time. If you do not put a number after the Y, the Spectrum 128 will send all instructions to channel 1.

Most MIDI instruments power up in OMNI mode so that they play the notes on channel 1 as well as the channel to which they are directed. To get the notes on the correct channels, the MIDI instrument must be put into POLY mode. You can do this by directly altering the instrument's MIDI channel, or by including the appropriate MIDI programming code in the string. The programming codes for your instrument should be listed in its handbook and can be sent to it from the Spectrum 128 by including the letter Z in the string, followed by the MIDI code number.

If your synthesizer understands key velocity (so that the program you send it controls the volume at which it plays) this is interpreted from the V settings in the strings. It is calculated at eight times the volume set by the V command, so it takes the values 0,8,16...120. The key release velocity when a note is switched off is sent as 64. If the synthesizer is not designed to understand key velocity, it will play the same volume regardless.

* * * * * * * * * * * * * * * * * *

11 SPECTRUM/SPANISH 128/UK 128 DIVERGENCES

The Spectrum 128 runs in two distinct modes, the first and start-up state is as a 128K machine, the second is as a 48K Spectrum. When running in 48K mode the only detectable difference is that previously unused space in the Spectrum rom now contains the keypad scanning routines. In Spectrum 128 mode the buffer for the ZX printer is used for extra system variables, so programs that use this area for code space may not function. On the edge connector interface the Z80 clock signal is not connected on Spanish machines. Also on the edge connector the Spectrum signals VIDEO, NOT-Y, V and U are not connected as the screen signals are handled through the RGB connector.

A potential source of peripheral incompatibility is the start up mode of the system which is Editor Rom, and Basic Ram, this may cause intelligent peripherals which contain their own Rom code to fail if they assume that on power up the Spectrum Rom is present and either use it for data or jump into it after their own initialisation process. Of course there is no correspondence between code in the Spectrum Rom and the Editor Rom so the likely result of such action is a system crash.

The Spectrum portion of the Rom found in the Spectrum 128 is very similar to the standard Rom found in 48K Spectrums. The major difference is that some of the empty space at the end of the Rom (previously FF hex) has been filled with routines to control the keypad. Care has been taken to ensure that locations which previously may have been used to vector mode 2 interrupts remain unaltered.

In all versions of the Spectrum an interaction occurs between the contents of the interrupt vector page or I register and the video controller. In detail, that during memory refresh cycles the I register contents are output on the high eight address lines. When the address that is formed as a result points at video ram the video controller confuses memory refresh cycles with memory access cycles. The end result is noise on the screen and potential random memory corruption.

If you are writing a Spectrum program, I register values 40H-7FH will cause this effect. On the Spectrum 128 I register values between C0H-FFH may also cause the effect when ram pages 4-7 (video) are in the memory map from C000-FFFFH.

Stated simply, any programmer using Z80 interrupt mode 2 should only vector interrupts into addresses between 8000H and 8FFFH. Otherwise they risk untraceable program failures and screen display corruption.

There are a number of minor physical and electrical improvements that will take place between the development machines and the UK production version.

The Z80 clock signal is brought out to the edge connector (Whoops).

Software developers using Spanish machines will experience difficulties in obtaining good sound and picture quality on UK televisions. The reason for this is that the sound carrier on Spanish television is modulated at a different frequency to that on UK TV.

The pinout of the RGB connector will be altered, adding a PAL colour composite signal and V-SYNC, deleting the existing monochrome composite.

Spectrum 128 mode System Variables

These are the new variables associated with 128K mode and they reside in the 48K printer bufffer. The most useful of these to third parties will be BAUD, which allow you to set up the RS-232 speed and ROW01,ROW23,ROW45 which give access to the keypad.

Variable	Address	Function
SWAP	5B00	ROM swapping subroutines
YOUNGER	5B14	
ONERR	5B1D	
PIN	5B2F	
POUT	5B34	
POUT2	5B4A	
TARGET	5B58	Address of subroutine in old ROM
RETADDR	5B5A	Return address in new ROM
BANKM	5B5C	Copy of last byte output to bank
RAMRST	5B5D	RST 8 instruction
RAMERR	5B5E	Error number or old ROM

BAUD	5B5F	Bit period in T states / 26
SERFL	5B61	Second-character-received-flag and data
COL	5B63	Current column from 1 to width
WIDTH	5B64	Paper column width
TVPARS	5B65	Number of parameters exepected by RS232
FLAGS3	5B66	Bit 0 ... Calculator/Edit mode Bit 1 ... BASIC line changed Bit 2 ... Silicon File open for write Bit 3 ... Silicon/cassette SLVM Bit 4 ... Load Bit 5 ... Save Bit 6 ... Merge Bit 7 ... Verify
N_STR1	5B67	SLVM Name

SLVM header blocks

HD_00	5B71	Type code
HD_0B	5B72	Length of block
HD_0D	5B74	Start of block
HD_0F	5B76	Program length
HD_11	5B78	Line number
SC_00	5B7A	Second set for LOAD,VERIFY,MERGE
SC_08	5B7B	
SC_0D	5B7D	
SC_0F	5B7F	
XLOC	5B71	Screen dump variables
YLOC	5B72	(Dual use of variables)
OLDSP	5B81	Old SP when TSTACK is used
SFNEXT	5B83	Pointer to last (empty) entry in directory
SFSPACE	5B85	Number of bytes left (17 bit)

The following variables return a keyped image when KPSCAN is called

ROW01	5B8B	pdmm1111 - present, device, micro, row1
ROW23	5B89	22223333 - row2, row3
ROW45	5B8A	44445555 - row4, row5
SYNRET	5B8B	Return address for ONERR
LASTV	5B8D	Last value printed by calculator
RC_LINE	5B92	
RC_START	5B94 (word)	The value of the new start line
RC_STEP	5B96 (word)	The step size between lines
TSTACK	5BFF	Temporary stack used when memory paging

* * * * * * * * * * * * * * * * *

12 FINAL

Any queries arising from this documentation should be sent to our new address.

Technical Support,
Sinclair Research Ltd,
Milton Hall,
Milton,
Cambridgeshire,
CB4 4AE.

Andrew Cummins ---- 10/2/86