

# SPECTRUM +2 ROM o DISASSEMBLY

The Spectrum ROMs are copyright Amstrad, who have kindly given permission to reverse engineer and publish Spectrum ROM disassemblies.



Image © Bill Bertram 2006

# CONTENTS

NOTES .....	10
Release Date .....	10
Disassembly Contributors .....	10
Markers .....	10
REFERENCE INFORMATION — PART 1 .....	10
128 BASIC Mode Limitations .....	10
Timing Information .....	10
I/O Details .....	11
Memory Paging .....	11
Memory Map .....	11
Shadow Display File .....	11
Contended Memory .....	11
Logical RAM Banks .....	11
AY-3-8912 Sound Generator .....	11
I/O Port A (AY-3-8912 Register 14) .....	12
Standard I/O Ports .....	12
Error Report Codes .....	12
Standard Error Report Codes .....	12
New Error Report Codes .....	12
System Variables .....	12
New System Variables .....	12
Standard System Variables .....	14
RAM Disk Catalogue .....	14
Editor Workspace Variables .....	14
Called ROM 1 Subroutines .....	18
RESTART ROUTINES — PART 1 .....	20
RST \$00 — Reset Machine .....	20
RST \$10 — Print A Character .....	20
RST \$18 — Collect A Character .....	20
RST \$20 — Collect Next Character .....	20
RST \$28 — Call Routine in ROM 1 .....	21
MASKABLE INTERRUPT ROUTINE .....	21
ERROR HANDLER ROUTINES — PART 1 .....	21
128K Error Routine .....	21
RESTART ROUTINES — PART 2 .....	21
Call ROM 1 Routine (RST \$28 Continuation) .....	21
RAM ROUTINES .....	22
Swap to Other ROM (copied to \$5B00) .....	22
Return to Other ROM Routine (copied to \$5B14) .....	22
Error Handler Routine (copied to \$5B1D) .....	23
'P' Channel Input Routine (copied to \$5B2F) .....	23
'P' Channel Output Routine (copied to \$5B34) .....	23
'P' Channel Exit Routine (copied to \$5B4A) .....	23
ERROR HANDLER ROUTINES — PART 2 .....	24
Call Subroutine .....	24
INITIALISATION ROUTINES — PART 1 .....	24
Reset Routine (RST \$00 Continuation, Part 1) .....	24
ROUTINE VECTOR TABLE .....	24
INITIALISATION ROUTINES — PART 2 .....	25
Fatal RAM Error .....	25
Reset Routine (RST \$00 Continuation, Part 2) .....	25
COMMAND EXECUTION ROUTINES — PART 1 .....	27
Execute Command Line .....	27
Return from BASIC Line Syntax Check .....	28
Parse a BASIC Line with No Line Number .....	28
ERROR HANDLER ROUTINES — PART 3 .....	29
Error Handler Routine .....	29
Error Handler Routine When Parsing BASIC Line .....	31
COMMAND EXECUTION ROUTINES — PART 2 .....	31
Parse a BASIC Line with a Line Number .....	31
ERROR HANDLER ROUTINES — PART 4 .....	32
New Error Message Vector Table .....	32
New Error Message Table .....	33

Print Message .....	33
INITIALISATION ROUTINES — PART 3 .....	34
The 'Initial Channel Information' .....	34
The 'Initial Stream Data' .....	34
ERROR HANDLER ROUTINES — PART 5 .....	34
Produce Error Report .....	34
Check for BREAK into Program .....	35
RS232 PRINTER ROUTINES .....	35
RS232 Channel Handler Routines .....	35
FORMAT Routine .....	36
Baud Rate Table .....	38
RS232 Input Routine .....	38
Read Byte from RS232 Port .....	38
RS232 Output Routine .....	41
Write Byte to RS232 Port .....	44
COPY Command Routine .....	45
Output Half Row .....	45
Output Nibble of Pixels .....	46
Output Characters from Table .....	46
Test Whether Pixel (B,C) is Set .....	47
EPSON Printer Control Code Tables .....	47
PLAY COMMAND ROUTINES .....	47
Command Data Block Format .....	47
Channel Data Block Format .....	48
Calculate Timing Loop Counter « RAM Routine » .....	50
Test BREAK Key .....	50
Select Channel Data Block Duration Pointers .....	50
Select Channel Data Block Pointers .....	51
Get Channel Data Block Address for Current String .....	51
Next Channel Data Pointer .....	51
PLAY Command (Continuation) .....	51
PLAY Command Character Table .....	52
Get Play Character .....	52
Get Next Note in Semitones .....	52
Get Numeric Value from Play String .....	53
Multiply DE by 10 .....	54
Find Next Note from Channel String .....	54
Play Command '!' (Comment) .....	54
Play Command 'O' (Octave) .....	54
Play Command 'N' (Separator) .....	55
Play Command '(' (Start of Repeat) .....	55
Play Command ')' (End of Repeat) .....	55
Get Address of Bracket Pointer Store .....	56
Play Command 'T' (Tempo) .....	57
Tempo Command Return .....	57
Play Command 'M' (Mixer) .....	58
Play Command 'V' (Volume) .....	58
Play Command 'U' (Use Volume Effect) .....	58
Play command 'W' (Volume Effect Specifier) .....	59
Play Command 'X' (Volume Effect Duration) .....	59
Play Command 'Y' (MIDI Channel) .....	59
Play Command 'Z' (MIDI Programming Code) .....	60
Play Command 'H' (Stop) .....	60
Play Commands 'a'..'g', 'A'..'G', '1'..'12', '&' and '_' .....	60
End of String Found .....	62
Point to Duration Length within Channel Data Block .....	62
Store Entry in Command Data Block's Channel Duration Length Pointer Table .....	62
PLAY Command Jump Table .....	63
Envelope Waveform Lookup Table .....	63
Identify Command Character .....	63
Semitones Table .....	63
Find Note Duration Length .....	64
Note Duration Table .....	64
Is Numeric Digit? .....	64
Play a Note On a Sound Chip Channel .....	64
Set Sound Generator Register .....	65
Read Sound Generator Register .....	66
Turn Off All Sound .....	66
Get Previous Character from Play String .....	66
Get Current Character from Play String .....	67

# SPECTRUM +2 ROM o DISASSEMBLY

Produce Play Error Reports .....	67
Play Note on Each Channel .....	68
Wait Note Duration .....	68
Find Smallest Duration Length .....	69
Play a Note on Each Channel and Update Channel Duration Lengths .....	69
Note Lookup Table .....	71
Play Note on MIDI Channel .....	73
Turn MIDI Channel Off .....	74
Send Byte to MIDI Device .....	74
CASSETTE / RAM DISK COMMAND ROUTINES — PART 1 .....	75
SAVE Routine .....	75
LOAD Routine .....	75
VERIFY Routine .....	75
MERGE Routine .....	75
RAM Disk Command Handling .....	76
RAM Disk VERIFY! Routine .....	77
RAM Disk MERGE! Routine .....	77
RAM Disk LOAD! Routine .....	78
RAM Disk Load Bytes .....	80
Get Expression from BASIC Line .....	80
Check Filename and Copy .....	80
Cassette / RAM Disk Command Handling .....	81
EDITOR ROUTINES — PART 1 .....	85
Relist the BASIC Program from the Current Line .....	85
Print All Screen Line Edit Buffer Rows to the Display File .....	88
Clear Editing Display .....	89
Shift All Edit Buffer Rows Up and Update Display File if Required .....	89
Shift All Edit Buffer Rows Down and Update Display File if Required .....	90
Insert Character into Edit Buffer Row, Shifting Row Right .....	91
Insert Character into Edit Buffer Row, Shifting Row Left .....	91
BASIC LINE AND COMMAND INTERPRETATION ROUTINES — PART 1 .....	92
The Syntax Offset Table .....	92
The Syntax Parameter Table .....	93
The 'Main Parser' Of the BASIC Interpreter .....	95
The Statement Loop .....	95
The 'Separator' Subroutine .....	96
The 'Statement Return' Subroutine .....	96
The 'Line Run' Entry Point .....	97
The 'Line New' Subroutine .....	97
REM Routine .....	97
The 'Line End' Routine .....	97
The 'Line Use' Routine .....	98
The 'Next Line' Routine .....	98
The 'CHECK-END' Subroutine .....	98
The 'STMT-NEXT' Routine .....	99
The 'Command Class' Table .....	99
The 'Command Classes — 0C, 0D & 0E' .....	99
The 'Command Classes — 00, 03 & 05' .....	100
The 'Command Class — 01' .....	100
The 'Command Class — 02' .....	100
The 'Command Class — 04' .....	100
The 'Command Class — 08' .....	101
The 'Command Class — 06' .....	101
Report C — Nonsense in BASIC .....	101
The 'Command Class — 0A' .....	101
The 'Command Class — 07' .....	101
The 'Command Class — 09' .....	102
The 'Command Class — 0B' .....	102
IF Routine .....	102
FOR Routine .....	102
READ Routine .....	103
DATA Routine .....	104
RUN Routine .....	104
CLEAR Routine .....	104
GO SUB Routine .....	105
RETURN Routine .....	105
DEF FN Routine .....	106
MOVE Routine .....	107
MENU ROUTINES — PART 1 .....	107
Run Tape Loader .....	107

# SPECTRUM +2 ROM o DISASSEMBLY

List Program to Printer .....	107
BASIC LINE AND COMMAND INTERPRETATION ROUTINES — PART 2 .....	108
SPECTRUM Routine .....	108
MENU ROUTINES — PART 2 .....	108
Main Menu — 48 BASIC Option .....	108
Set 'P' Channel Data .....	108
LOAD "" Command Bytes .....	109
BASIC LINE AND COMMAND INTERPRETATION ROUTINES — PART 3 .....	109
LLIST Routine .....	109
LIST Routine .....	109
RAM Disk SAVE! Routine .....	109
CAT! Routine .....	110
ERASE! Routine .....	110
RAM DISK COMMAND ROUTINES — PART 2 .....	110
Load Header from RAM Disk .....	110
Load from RAM Disk .....	111
PAGING ROUTINES — PART 1 .....	111
Page Logical RAM Bank .....	111
Physical RAM Bank Mapping Table .....	111
RAM DISK COMMAND ROUTINES — PART 3 .....	112
Compare Filenames .....	112
Create New Catalogue Entry .....	112
Adjust RAM Disk Free Space .....	113
Find Catalogue Entry for Filename .....	113
Find RAM Disk File .....	114
Update Catalogue Entry .....	114
Save Bytes to RAM Disk .....	115
Load Bytes from RAM Disk .....	116
Transfer Bytes to RAM Bank 4 — Vector Table Entry .....	118
Transfer Bytes from RAM Bank 4 — Vector Table Entry .....	119
PAGING ROUTINES — PART 2 .....	119
Use Normal RAM Configuration .....	119
Select RAM Bank .....	119
Use Workspace RAM Configuration .....	120
RAM DISK COMMAND ROUTINES — PART 4 .....	120
Erase a RAM Disk File .....	120
Print RAM Disk Catalogue .....	123
Print Catalogue Filename Data .....	124
Print Single Catalogue Entry .....	124
BASIC LINE AND COMMAND INTERPRETATION ROUTINES — PART 4 .....	125
LPRINT Routine .....	125
PRINT Routine .....	125
INPUT Routine .....	125
COPY Routine .....	125
NEW Routine .....	126
CIRCLE Routine .....	126
DRAW Routine .....	126
DIM Routine .....	126
Error Report C — Nonsense in BASIC .....	127
Clear Screen Routine .....	127
Evaluate Numeric Expression .....	127
Process Key Press .....	128
Find Start of BASIC Command .....	128
Is LET Command? .....	129
Is Operator Character? .....	129
Operator Tokens Table .....	129
Is Function Character? .....	129
Is Numeric or Function Expression? .....	130
Is Numeric Character? .....	130
PLAY Routine .....	130
UNUSED ROUTINES — PART 1 .....	131
Return to Editor .....	131
BC=HL-DE, Swap HL and DE .....	131
Create Room for 1 Byte .....	131
Room for BC Bytes? .....	132
HL = A*32 .....	132
HL = A*8 .....	132
Find Amount of Free Space .....	132
Print Screen Buffer Row .....	132
Blank Screen Buffer Content .....	133

Print Screen Buffer to Display File .....	133
Print Screen Buffer Characters to Display File .....	134
Copy A Character « RAM Routine » .....	135
Toggle ROMs 1 « RAM Routine » .....	136
Toggle ROMs 2 « RAM Routine » .....	136
Construct 'Copy Character' Routine in RAM .....	136
Set Attributes File from Screen Buffer .....	136
Set Attributes for a Screen Buffer Row .....	137
Swap Ink and Paper Attribute Bits .....	138
Character Data .....	139
KEY ACTION TABLES .....	139
Editing Keys Action Table .....	139
Menu Keys Action Table .....	140
MENU ROUTINES — PART 3 .....	140
Initialise Mode Settings .....	140
Show Main Menu .....	140
EDITOR ROUTINES — PART 2 .....	141
Return to Editor / Calculator / Menu from Error .....	141
Return to the Editor .....	141
Main Waiting Loop .....	142
Process Key Press .....	142
TOGGLE Key Handler Routine .....	143
Select Lower Screen .....	143
Select Upper Screen .....	144
Produce Error Beep .....	144
Produce Success Beep .....	144
MENU ROUTINES — PART 4 .....	145
Menu Key Press Handler Routines .....	145
Menu Key Press Handler — MENU .....	145
Menu Key Press Handler — SELECT .....	145
Menu Key Press Handler — CURSOR UP .....	145
Menu Key Press Handler — CURSOR DOWN .....	145
Menu Tables .....	146
Main Menu .....	146
Edit Menu .....	146
Calculator Menu .....	147
Tape Loader Text .....	147
Menu Handler Routines .....	147
Edit Menu — Screen Option .....	147
Edit Menu / Calculator Menu — Exit Option .....	147
Main Menu — Tape Loader Option .....	148
Edit Menu — Renumber Option .....	148
Edit Menu — Print Option .....	148
Main Menu — Calculator Option .....	148
EDITOR ROUTINES — PART 3 .....	149
Reset Cursor Position .....	149
Return to Main Menu .....	149
Main Screen Error Cursor Settings .....	149
Lower Screen Good Cursor Settings .....	149
Initialise Lower Screen Editing Settings .....	150
Initialise Main Screen Editing Settings .....	150
Handle Key Press Character Code .....	150
DELETE-RIGHT Key Handler Routine .....	150
DELETE Key Handler Routine .....	151
ENTER Key Handler Routine .....	151
TOP-OF-PROGRAM Key Handler Routine .....	152
END-OF-PROGRAM Key Handler Routine .....	152
WORD-LEFT Key Handler Routine .....	153
WORD-RIGHT Key Handler Routine .....	153
Remove Cursor .....	153
Show Cursor .....	154
Display Cursor .....	154
Fetch Cursor Position .....	154
Store Cursor Position .....	154
Get Current Character from Screen Line Edit Buffer .....	155
TEN-ROWS-DOWN Key Handler Routine .....	155
TEN-ROWS-UP Key Handler Routine .....	155
END-OF-LINE Key Handler Routine .....	156
START-OF-LINE Key Handler Routine .....	156
CURSOR-UP Key Handler Routine .....	157

CURSOR-DOWN Key Handler Routine .....	157
CURSOR-LEFT Key Handler Routine .....	158
CURSOR-RIGHT Key Handler Routine .....	158
Edit Buffer Routines — Part 1 .....	158
Find Closest Screen Line Edit Buffer Editable Position to the Right else Left .....	158
Find Closest Screen Line Edit Buffer Editable Position to the Left else Right .....	159
Insert BASIC Line, Shift Edit Buffer Rows Down If Required and Update Display File If Required .....	159
Insert BASIC Line, Shift Edit Buffer Rows Up If Required and Update Display File If Required .....	160
Find Next Screen Line Edit Buffer Editable Position to Left, Wrapping Above if Required .....	160
Find Next Screen Line Edit Buffer Editable Position to Right, Wrapping Below if Required .....	161
Find Screen Line Edit Buffer Editable Position from Previous Column to the Right .....	163
Find Screen Line Edit Buffer Editable Position to the Left .....	163
Find Start of Word to Left in Screen Line Edit Buffer .....	163
Find Start of Word to Right in Screen Line Edit Buffer .....	164
Find Start of Current BASIC Line in Screen Line Edit Buffer .....	165
Find End of Current BASIC Line in Screen Line Edit Buffer .....	165
Insert BASIC Line into Program if Altered .....	166
Insert Line into BASIC Program If Altered and the First Row of the Line .....	166
Insert Line into BASIC Program .....	166
Fetch Next Character from BASIC Line to Insert .....	169
Fetch Next Character Jump Table .....	170
Fetch Character from the Current Row of the BASIC Line in the Screen Line Edit Buffer .....	170
Fetch Character from Edit Buffer Row .....	172
Upper Screen Rows Table .....	172
Lower Screen Rows Table .....	172
Reset to Main Screen .....	172
Reset to Lower Screen .....	173
Find Edit Buffer Editable Position from Previous Column to the Right .....	173
Find Edit Buffer Editable Position to the Left .....	173
Fetch Edit Buffer Row Character .....	174
Insert Character into Screen Line Edit Buffer .....	174
Insert Blank Row into Screen Edit Buffer, Shifting Rows Down .....	176
Empty Edit Buffer Row Data .....	176
Delete a Character from a BASIC Line in the Screen Line Edit Buffer .....	177
Shift Rows Up to Close Blank Row in Screen Line Edit Buffer .....	179
DELETE-WORD-LEFT Key Handler Routine .....	180
DELETE-WORD-RIGHT Key Handler Routine .....	181
DELETE-TO-START-OF-LINE Key Handler Routine .....	182
DELETE-TO-END-OF-LINE Key Handler Routine .....	182
Remove Cursor Attribute and Disable Updating Display File .....	183
Previous Character Exists in Screen Line Edit Buffer? .....	183
Find Row Address in Screen Line Edit Buffer .....	183
Find Position within Screen Line Edit Buffer .....	184
Below-Screen Line Edit Buffer Settings .....	184
Set Below-Screen Line Edit Buffer Settings .....	184
Shift Up Rows in Below-Screen Line Edit Buffer .....	184
Shift Down Rows in Below-Screen Line Edit Buffer .....	185
Insert Character into Below-Screen Line Edit Buffer .....	186
Find Row Address in Below-Screen Line Edit Buffer .....	188
Delete a Character from a BASIC Line in the Below-Screen Line Edit Buffer .....	188
Above-Screen Line Edit Buffer Settings .....	189
Set Above-Screen Line Edit Buffer Settings .....	190
Shift Rows Down in the Above-Screen Line Edit Buffer .....	190
Shift Row Up into the Above-Screen Line Edit Buffer if Required .....	191
Find Row Address in Above-Screen Line Edit Buffer .....	192
BASIC Line Character Action Handler Jump Table .....	192
Copy a BASIC Line into the Above-Screen or Below-Screen Line Edit Buffer .....	193
Set 'Continuation' Row in Line Edit Buffer .....	194
BASIC Line Handling Routines .....	195
Find Address of BASIC Line with Specified Line Number .....	195
Create Next Line Number Representation in Keyword Construction Buffer .....	195
Fetch Next De-tokenized Character from Selected BASIC Line in Program Area .....	195
Copy 'Insert Keyword Representation into Keyword Construction Buffer' Routine into RAM .....	195
Insert Keyword Representation into Keyword Construction Buffer « RAM Routine » .....	196
Copy Keyword Characters « RAM Routine » .....	197
Identify Token from Table .....	197
Create Next Line Number Representation in Keyword Construction Buffer .....	198
Insert ASCII Line Number Digit .....	199
Find Address of BASIC Line with Specified Line Number .....	200
Move to Next BASIC Line .....	200

Check if at End of BASIC Program .....	201
Compare Line Numbers .....	201
Clear BASIC Line Construction Pointers .....	201
Find Address of BASIC Line .....	201
Fetch Next De-tokenized Character from BASIC Line in Program Area .....	202
Edit Buffer Routines — Part 2 .....	204
Keywords String Table .....	204
Indentation Settings .....	204
Set Indentation Settings .....	205
Store Character in Column of Edit Buffer Row .....	205
'Enter' Action Handler Routine .....	205
'Null Columns' Action Handler Routine .....	205
Null Column Positions .....	205
Indent Edit Buffer Row .....	206
Print Edit Buffer Row to Display File if Required .....	206
Shift Up Edit Rows in Display File if Required .....	206
Shift Down Edit Rows in Display File if Required .....	207
Set Cursor Attribute Colour .....	207
Restore Cursor Position Previous Attribute .....	207
Reset 'L' Mode .....	208
Wait for a Key Press .....	208
MENU ROUTINES — PART 5 .....	208
Display Menu .....	208
Plot a Line .....	209
Print "AT B,C" Characters .....	210
Print String .....	210
Store Menu Screen Area .....	210
Restore Menu Screen Area .....	210
Store / Restore Menu Screen Row .....	211
Move Up Menu .....	212
Move Down Menu .....	212
Toggle Menu Option Selection Highlight .....	212
Menu Title Colours Table .....	213
Menu Title Space Table .....	213
Menu Sinclair Stripes Bitmaps .....	213
Sinclair Strip 'Text' .....	213
Print the Sinclair stripes on the menu .....	213
Print '128 BASIC' Banner .....	214
Print 'Calculator' Banner .....	214
Print 'Tape Loader' Banner .....	214
Print Banner .....	214
Clear Lower Editing Display .....	214
RENUMBER ROUTINE .....	215
Tokens Using Line Numbers .....	216
Parse a Line Renumbering Line Number References .....	216
Count the Number of BASIC Lines .....	219
Skip Spaces .....	220
Create ASCII Line Number Representation .....	220
Insert Line Number Digit .....	221
EDITOR ROUTINES — PART 4 .....	221
Initial Lower Screen Cursor Settings .....	221
Initial Main Screen Cursor Settings .....	221
Set Main Screen Editing Cursor Details .....	222
Set Lower Screen Editing Cursor Details .....	222
UNUSED ROUTINES — PART 2 .....	222
Print 'AD' .....	222
EDITOR ROUTINES — PART 5 .....	222
Store Cursor Colour .....	222
Set Cursor Position Attribute .....	222
Restore Cursor Position Attribute .....	223
Shift Up Edit Rows in Display File .....	223
Shift Down Edit Rows in Display File .....	223
Print a Row of the Edit Buffer to the Screen .....	224
Clear Display Rows .....	225
Find Rows and Columns to End of Screen .....	226
Find Rows to End of Screen .....	226
Get Attribute Address .....	226
Exchange Colour Items .....	227
EDITOR ROUTINES — PART 5 .....	227
Tokenize BASIC Line .....	227

Fetch Next Character and Character Status from BASIC Line to Insert .....	232
Is Lowercase Letter? .....	233
Copy Keyword Conversion Buffer Contents into BASIC Line Workspace .....	233
Insert Character into Keyword Conversion Buffer .....	233
Insert Character into BASIC Line Workspace, Handling '>' and '<' .....	234
Insert Character into BASIC Line Workspace, Handling 'REM' and Quotes .....	236
Insert Character into BASIC Line Workspace With Space Suppression .....	237
Insert a Character into BASIC Line Workspace .....	238
Room for BC Bytes? .....	240
Identify Keyword .....	241
Copy Data Block .....	241
Get Numeric Value for ASCII Character .....	242
Call Action Handler Routine .....	242
PROGRAMMERS' INITIALS .....	243
UNUSED SPACE .....	243
END OF ROM MARKER .....	243
REFERENCE INFORMATION — PART 2 .....	244
Routines Copied/Constructed in RAM .....	244
Construct Keyword Representation .....	244
Copy Keyword Characters .....	245
Identify Token .....	245
Insert Character into Display File .....	246
Standard Error Report Codes .....	247
Standard System Variables .....	248
Memory Map .....	250
I Register .....	250
Screen File Formats .....	250
Display File .....	250
Attributes File .....	251
Address Conversion Between Display File and Attributes File .....	251
Standard I/O Ports .....	251
Port \$FE .....	251
Cassette Header Format .....	251
AY-3-8912 Programmable Sound Generator Registers .....	252
Registers 0 and 1 (Channel A Tone Generator) .....	252
Registers 2 and 3 (Channel B Tone Generator) .....	252
Registers 4 and 5 (Channel C Tone Generator) .....	252
Register 6 (Noise Generator) .....	252
Register 7 (Mixer — I/O Enable) .....	252
Register 8 (Channel A Volume) .....	252
Register 9 (Channel B Volume) .....	253
Register 10 (Channel C Volume) .....	253
Register 11 and 12 (Envelope Period) .....	253
Register 13 (Envelope Shape) .....	253
Register 14 (I/O Port) .....	254
Socket Pin Outs .....	254
RS232/MIDI Socket .....	254
Keypad Socket .....	255
Monitor Socket .....	255
Edge Connector .....	256
Sound Socket .....	256
ROM 0 Differences Between Models .....	256

## NOTES

### Release Date

4th August 2017

This file was automatically derived from the Spectrum 128 ROM 0 disassembly, using a conversion utility created by Paul Farrow.

Any enhancements or corrections should only be made to the Spectrum 128 ROM 0 disassembly and then the utility used to automatically regenerate the Spectrum +2 listing.

### Disassembly Contributors

Matthew Wilson ([www.matthew-wilson.net/spectrum/rom/](http://www.matthew-wilson.net/spectrum/rom/))

Andrew Owen ([cheveron-AT-gmail.com](mailto:cheveron-AT-gmail.com))

Geoff Wearmouth ([gwearmouth-AT-hotmail.com](mailto:gwearmouth-AT-hotmail.com))

Rui Tunes

Paul Farrow ([www.fruitcake.plus.com](http://www.fruitcake.plus.com))

### Markers

The following markers appear throughout the disassembly:

[...] = Indicates a comment about the code.

???? = Information to be determined.

For bugs, the following marker format is used:

[BUG - xxxx. Credit: yyyy] = Indicates a confirmed bug, with a description 'xxxx' of it and the discoverer 'yyyy'.

[BUG? - xxxx. Credit: yyyy] = Indicates a suspected bug, with a description 'xxxx' of it and the discoverer 'yyyy'.

Since many of the Spectrum 128 ROM routines were re-used in the Spectrum +2 and +3, where a bug was originally identified in the Spectrum +2 or +3 the discoverer is acknowledged along with who located the corresponding bug in the Spectrum 128.

For every bug identified, an example fix is provided and the author acknowledged. Some of these fixes can be made directly within the routines affected since they do not increase the length of those routines. Others require the insertion of extra instructions and hence these cannot be completely fitted within the routines affected. Instead a jump must be made to a patch routine located within a spare area of the ROM.

Fortunately there is 0.5K of unused routines located at \$2355-\$2555 (ROM 0) which are remnants of the original Spanish 128, and another unused routine located at \$3F6A-\$3F75 (ROM 0). This is sufficient space to implement all of the bug fixes suggested.

## REFERENCE INFORMATION — PART 1

### 128 BASIC Mode Limitations

There are a number of limitations when using 128 BASIC mode, some of which are not present when using the equivalent 48 BASIC mode operations. These are more design decisions than bugs.

- The RAM disk VERIFY command does not verify but simply performs a LOAD.
- The renumber facility will not renumber line numbers that are defined as an expression, e.g. GO TO VAL "10".
- The printer output routine cannot handle binary data and hence EPSON printer ESC codes cannot be sent.
- The Editor has the following limitations:
- Variables cannot have the same name as a keyword. This only applies when entering a program and not when one is loaded in.
- Line number 0 is not supported and will not list properly. It is not possible to directly insert such a line, not even in 48 BASIC mode, and so line number 0 is not officially supported.
- There is a practical limitation on the size of lines that can be entered. It is limited to 20 indented rows, which is the size of the editing buffers. Typed lines greater than 20 rows get inserted into the BASIC program, but only the first 20 rows are shown on screen. Editing such a line causes it to be truncated to 20 rows. There is no warning when the 20 row limit is exceeded.
- It is not possible to directly enter embedded control codes, or to correctly edit loaded in programs that contain them. Loaded programs that contain them will run correctly so long as the lines are not edited.
- It is not possible to embed the string of characters ">=", "<=" or "<>" into a string or REM statement without them being tokenized (this is perhaps more an oversight than a design decision).
- In 48 BASIC mode if the line '10 REM abc: PRINT xyz' is typed then the word PRINT is stored as a new keyword since the colon (arguably incorrectly) reverts to 'K' mode. In 128 BASIC mode, typing the same line stores each letter as a separate character.

### Timing Information

Clock Speed = 3.54690 MHz (48K Spectrum clock speed was 3.50000 MHz) Scan line = 228 T-states (48K Spectrum was 224 T-states).

TV scan lines = 311 total, 63 above picture (48K Spectrum had 312 total, 64 above picture).

## I/O Details

### Memory Paging

Memory paging is controlled by I/O port:

\$7FFD (Out) - Bits 0-2: RAM bank (0-7) to page into memory map at \$C000.

Bit 3 : 0=SCREEN 0 (normal display file in bank 5), 1=SCREEN 1 (shadow display file in bank 7).

Bit 4 : 0=ROM 0 (128K Editor), 1=ROM 1 (48K BASIC).

Bit 5 : 1=Disable further output to this port until a hard reset occurs.

Bit 6-7 : Not used (always write 0).

The Editor ROM (ROM 0) always places a copy of the last value written to port \$7FFD into new system variable BANK\_M (\$5B5C).

### Memory Map

ROM 0 or 1 resides at \$0000-\$3FFF.

RAM bank 5 resides at \$4000-\$7FFF always.

RAM bank 2 resides at \$8000-\$BFFF always.

Any RAM bank may reside at \$C000-\$FFFF.

### Shadow Display File

The shadow screen may be active even when not paged into the memory map.

### Contended Memory

Physical RAM banks 1, 3, 5 and 7 are contended with the ULA.

### Logical RAM Banks

Throughout ROM 0, memory banks are accessed using a logical numbering scheme, which maps to physical RAM banks as follows:

Logical Bank	Physical Bank
\$00	\$01
\$01	\$03
\$02	\$04
\$03	\$06
\$04	\$07
\$05	\$00

This scheme makes the RAM disk code simpler than having to deal directly with physical RAM bank numbers.

### AY-3-8912 Sound Generator

The AY-3-8912 sound generator is controlled by two I/O ports: \$FFFD (Out) - Select a register 0-14.

\$FFFD (In) - Read from the selected register.

\$BFFD (In/Out) - Write to the selected register. The status of the register can also be read back.

The AY-3-8912 I/O port A is used to drive the RS232 and Keypad sockets.

Register	Function	Range
0	Channel A fine pitch	8-bit (0-255)
1	Channel A course pitch	4-bit (0-15)
2	Channel B fine pitch	8-bit (0-255)
3	Channel B course pitch	4-bit (0-15)
4	Channel C fine pitch	8-bit (0-255)
5	Channel C course pitch	4-bit (0-15)
6	Noise pitch	5-bit (0-31)
7	Mixer	8-bit (see end of file for description)
8	Channel A volume	4-bit (0-15, see end of file for description)
9	Channel B volume	4-bit (0-15, see end of file for description)
10	Channel C volume	4-bit (0-15, see end of file for description)
11	Envelope fine duration	8-bit (0-255)
12	Envelope course duration	8-bit (0-255)
13	Envelope shape	4-bit (0-15)
14	I/O port A	8-bit (0-255)

See the end of this document for description on the sound generator registers.

## I/O Port A (AY-3-8912 Register 14)

This controls the RS232 and Keypad sockets.

Select the port via a write to port \$FFFD with 14, then read via port \$FFFD and write via port \$BFFD. The state of port \$BFFD can also be read back.

Bit 0: KEYPAD CTS (out) - 0=Spectrum ready to receive, 1=Busy

Bit 1: KEYPAD RXD (out) - 0=Transmit high bit, 1=Transmit low bit

Bit 2: RS232 CTS (out) - 0=Spectrum ready to receive, 1=Busy

Bit 3: RS232 RXD (out) - 0=Transmit high bit, 1=Transmit low bit

Bit 4: KEYPAD DTR (in) - 0=Keypad ready for data, 1=Busy

Bit 5: KEYPAD TXD (in) - 0=Receive high bit, 1=Receive low bit

Bit 6: RS232 DTR (in) - 0=Device ready for data, 1=Busy

Bit 7: RS232 TXD (in) - 0=Receive high bit, 1=Receive low bit

See the end of this document for the pinouts for the RS232 and KEYPAD sockets.

## Standard I/O Ports

See the end of this document for descriptions of the standard Spectrum I/O ports.

## Error Report Codes

### Standard Error Report Codes

See the end of this document for descriptions of the standard error report codes.

### New Error Report Codes

a — MERGE error	MERGE! would not execute for some reason - either size or file type wrong.
b — Wrong file type	A file of an inappropriate type was specified during RAM disk operation, for instance a CODE file in LOAD!"name".
c — CODE error	The size of the file would lead to an overrun of the top of memory.
d — Too many brackets	Too many brackets around a repeated phrase in one of the arguments.
e — File already exists	The file name specified has already been used.
f — Invalid name	The file name specified is empty or above 10 characters in length.
g — File does not exist	[Never used by the ROM].
h — File does not exist	The specified file could not be found.
i — Invalid device	The device name following the FORMAT command does not exist or correspond to a physical device.
j — Invalid baud rate	The baud rate for the RS232 was set to 0.
k — Invalid note name	PLAY came across a note or command it didn't recognise, or a command which was in lower case.
l — Number too big	A parameter for a command is an order of magnitude too big.
m — Note out of range	A series of sharps or flats has taken a note beyond the range of the sound chip.
n — Out of range	A parameter for a command is too big or too small. If the error is very large, error L results.
o — Too many tied notes	An attempt was made to tie too many notes together.
p — © 1986 Sinclair Research Ltd	This error is given when too many PLAY channel strings are specified. Up to 8 PLAY channel strings are supported by MIDI devices such as synthesisers, drum machines or sequencers. Note that a PLAY command with more than 8 strings cannot be entered directly from the Editor. The Spanish 128 produces "p Bad parameter" for this error. It could be that the intention was to save memory by using the existing error message of "Q Parameter error" but the change of report code byte was overlooked.

## System Variables

### New System Variables

These are held in the old ZX Printer buffer at \$5B00-\$5BFF.

Note that some of these names conflict with the system variables used by the ZX Interface 1.

SWAP	EQU \$5B00	20	Swap paging subroutine.
YOUNGER	EQU \$5B14	9	Return paging subroutine.
ONERR	EQU \$5B1D	18	Error handler paging subroutine.
PIN	EQU \$5B2F	5	RS232 input pre-routine.
POUT	EQU \$5B34	22	RS232 token output pre-routine. This can be patched to bypass the control code filter.
POUT2	EQU \$5B4A	14	RS232 character output pre-routine.
TARGET	EQU \$5B58	2	Address of subroutine to call in ROM 1.
RETADDR	EQU \$5B5A	2	Return address in ROM 0.

# SPECTRUM +2 ROM o DISASSEMBLY

BANK_M	EQU \$5B5C	1	Copy of last byte output to I/O port \$7FFD.
RAMRST	EQU \$5B5D	1	Stores instruction RST \$08 and used to produce a standard ROM error.
RAMERR	EQU \$5B5E	1	Error number for use by RST \$08 held in RAMRST.
BAUD	EQU \$5B5F	2	Baud rate timing constant for RS232 socket. Default value of 11. [Name clash with ZX Interface 1 system variable at \$5CC3]
SERFL	EQU \$5B61	2	Second character received flag: Bit 0 : 1=Character in buffer. Bits 1-7: Not used (always hold 0). Received Character.
COL	EQU \$5B63	1	Current column from 1 to WIDTH.
WIDTH	EQU \$5B64	1	Paper column width. Default value of 80. [Name clash with ZX Interface 1 Edition 2 system variable at \$5CB1]
TVPARS	EQU \$5B65	1	Number of inline parameters expected by RS232 (e.g. 2 for AT).
FLAGS3	EQU \$5B66	1	Flags: [Name clashes with the ZX Interface 1 system variable at \$5CB6] Bit 0: 1=BASIC/Calculator mode, 0=Editor/Menu mode. Bit 1: 1=Auto-run loaded BASIC program. [Set but never tested by the ROM] Bit 2: 1=Editing RAM disk catalogue. Bit 3: 1=Using RAM disk commands, 0=Using cassette commands. Bit 4: 1=Indicate LOAD. Bit 5: 1=Indicate SAVE. Bit 6; 1=Indicate MERGE. Bit 7: 1=Indicate VERIFY.
N_STR1	EQU \$5B67	10	Used by RAM disk to store a filename. [Name clash with ZX Interface 1 system variable at \$5CDA] Used by the renumber routine to store the address of the BASIC line being examined.
HD_00	EQU \$5B71	1	Used by RAM disk to store file header information (see RAM disk Catalogue section below for details). [Name clash with ZX Interface 1 system variable at \$5CE6] Used as column pixel counter in COPY routine. Used by FORMAT command to store specified baud rate. Used by renumber routine to store the number of digits in a pre-renumbered line number reference. [Name clash with ZX Interface 1 system variable at \$5CE7]
HD_0B	EQU \$5B72	2	Used by RAM disk to store header info - length of block. Used as half row counter in COPY routine. Used by renumber routine to generate ASCII representation of a new line number.
HD_0D	EQU \$5B74	2	Used by RAM disk to store file header information (see RAM disk Catalogue section below for details). [Name clash with ZX Interface 1 system variable at \$5CE9]
HD_0F	EQU \$5B76	2	Used by RAM disk to store file header information (see RAM disk Catalogue section below for details). [Name clash with ZX Interface 1 system variable at \$5CEB] Used by renumber routine to store the address of a referenced BASIC line.
HD_11	EQU \$5B78	2	Used by RAM disk to store file header information (see RAM disk Catalogue section below for details). [Name clash with ZX Interface 1 system variable at \$5CED] Used by renumber routine to store existing VARS address/current address within a line.
SC_00	EQU \$5B7A	1	Used by RAM disk to store alternate file header information (see RAM disk Catalogue section below for details).
SC_0B	EQU \$5B7B	2	Used by RAM disk to store alternate file header information (see RAM disk Catalogue section below for details).
SC_0D	EQU \$5B7D	2	Used by RAM disk to store alternate file header information (see RAM disk Catalogue section below for details).
SC_0F	EQU \$5B7F	2	Used by RAM disk to store alternate file header information (see RAM disk Catalogue section below for details).
OLDSP	EQU \$5B81	2	Stores old stack pointer when TSTACK in use.
SFNEXT	EQU \$5B83	2	End of RAM disk catalogue marker. Pointer to first empty catalogue entry.
SFSPACE	EQU \$5B85	3	Number of bytes free in RAM disk (3 bytes, 17 bit, LSB first).
ROW01	EQU \$5B88	1	Stores keypad data for row 3, and flags: Bit 0 : 1=Key '+' pressed. Bit 1 : 1=Key '6' pressed. Bit 2 : 1=Key '5' pressed. Bit 3 : 1=Key '4' pressed. Bits 4-5: Always 0. Bit 6 : 1=Indicates successful communications to the keypad. Bit 7 : 1=If communications to the keypad established.

## SPECTRUM +2 ROM 0 DISASSEMBLY

ROW23	EQU \$5B89	1	Stores keypad key press data for rows 1 and 2: Bit 0: 1=Key ')' pressed. Bit 1: 1=Key '(' pressed. Bit 2: 1=Key '*' pressed. Bit 3: 1=Key '/' pressed. Bit 4: 1=Key '-' pressed. Bit 5: 1=Key '9' pressed. Bit 6: 1=Key '8' pressed. Bit 7: 1=Key '7' pressed.
ROW45	EQU \$5B8A	1	Stores keypad key press data for rows 4 and 5: Bit 0: Always 0. Bit 1: 1=Key '.' pressed. Bit 2: Always 0. Bit 3: 1=Key '0' pressed. Bit 4: 1=Key 'ENTER' pressed. Bit 5: 1=Key '3' pressed. Bit 6: 1=Key '2' pressed. Bit 7: 1=Key '1' pressed.
SYNRET	EQU \$5B8B	2	Return address for ONERR routine.
LASTV	EQU \$5B8D	5	Last value printed by calculator.
RNLINE	EQU \$5B92	2	Address of the length bytes in the line currently being renumbered.
RNFIRST	EQU \$5B94	2	Starting line number when renumbering. Default value of 10.
RNSTEP	EQU \$5B96	2	Step size when renumbering. Default value of 10.
STRIP1	EQU \$5B98	32	Used as RAM disk transfer buffer (32 bytes to \$5BB7). Used to hold Sinclair stripe character patterns (16 bytes to \$5BA7).
			...
TSTACK	EQU \$5BFF	n	Temporary stack (grows downwards). The byte at \$5BFF is not actually used.

## Standard System Variables

These occupy addresses \$5C00-\$5CB5.

See the end of this document for descriptions of the standard system variables.

## RAM Disk Catalogue

The catalogue can occupy addresses \$C000-\$EBFF in physical RAM bank 7, starting at \$EBFF and growing downwards.

Each entry contains 20 bytes:

Bytes \$00-\$09: Filename.

Bytes \$0A-\$0C: Start address of file in RAM disk area.

Bytes \$0D-\$0F: Length of file in RAM disk area.

Bytes \$10-\$12: End address of file in RAM disk area (used as current position indicator when loading/saving).

Byte \$13 : Flags:

Bit 0 : 1=Entry requires updating.

Bits 1-7: Not used (always hold 0).

The catalogue can store up to 562 entries, and hence the RAM disk can never hold more than 562 files no matter how small the files themselves are.

Note that filenames are case sensitive.

The shadow screen (SCREEN 1) also resides in physical RAM bank 7 and so if more than 217 catalogue entries are created then SCREEN 1 will become corrupted [Credit: Toni Baker, ZX Computing Monthly].

However, since screen 1 cannot be used from BASIC, it may have been a design decision to allow the RAM disk to overwrite it.

The actual files are stored in physical RAM banks 1, 3, 4 and 6 (logical banks 0, 1, 2, 3), starting from \$C000 in physical RAM bank 1 and growing upwards.

A file consists of a 9 byte header followed by the data for the file. The header bytes have the following meaning:

Byte \$00 : File type - \$00=Program, \$01=Numeric array, \$02=Character array, \$03=Code/Screen\$.

Bytes \$01-\$02: Length of program/code block/screen\$/array (\$1B00 for screen\$).

Bytes \$03-\$04: Start of code block/screen\$ (\$4000 for screen\$).

Bytes \$05-\$06: Offset to the variables (i.e. length of program) if a program. For an array, \$05 holds the variable name.

Bytes \$07-\$08: Auto-run line number for a program (\$80 in high byte if no auto-run).

## Editor Workspace Variables

These occupy addresses \$EC00-\$FFFF in physical RAM bank 7, and form a workspace used by 128 BASIC Editor.

\$EC00	3	Byte 0: Flags used when inserting a line into the BASIC program (first 4 bits are mutually exclusive). Bit 0: 1=First row of the BASIC line off top of screen. Bit 1: 1=On first row of the BASIC line.
--------	---	---

## SPECTRUM +2 ROM o DISASSEMBLY

		<p>Bit 2: 1=Using lower screen and only first row of the BASIC line visible.</p> <p>Bit 3: 1=At the end of the last row of the BASIC line.</p> <p>Bit 4: Not used (always 0).</p> <p>Bit 5: Not used (always 0).</p> <p>Bit 6: Not used (always 0).</p> <p>Bit 7: 1=Column with cursor not yet found.</p> <p>Byte 1: Column number of current position within the BASIC line being inserted. Used when fetching characters.</p> <p>Byte 2: Row number of current position within the BASIC line is being inserted. Used when fetching characters.</p>
\$EC03	3	<p>Byte 0: Flags used upon an error when inserting a line into the BASIC program (first 4 bits are mutually exclusive).</p> <p>Bit 0: 1=First row of the BASIC line off top of screen.</p> <p>Bit 1: 1=On first row of the BASIC line.</p> <p>Bit 2: 1=Using lower screen and only first row of the BASIC line visible.</p> <p>Bit 3: 1=At the end of the last row of the BASIC line.</p> <p>Bit 4: Not used (always 0).</p> <p>Bit 5: Not used (always 0).</p> <p>Bit 6: Not used (always 0).</p> <p>Bit 7: 1=Column with cursor not yet found.</p> <p>Byte 1: Start column number where BASIC line is being entered. Always holds 0.</p> <p>Byte 2: Start row number where BASIC line is being entered.</p>
\$EC06	2	Count of the number of editable characters in the BASIC line up to the cursor within the Screen Line Edit Buffer.
\$EC08	2	Version of E_PPC used by BASIC Editor to hold last line number entered.
\$EC0C	1	Current menu index.
\$EC0D	1	<p>Flags used by 128 BASIC Editor:</p> <p>Bit 0: 1=Screen Line Edit Buffer (including Below-Screen Line Edit Buffer) is full.</p> <p>Bit 1: 1=Menu is displayed.</p> <p>Bit 2: 1=Using RAM disk.</p> <p>Bit 3: 1=Current line has been altered.</p> <p>Bit 4: 1=Return to calculator, 0=Return to main menu.</p> <p>Bit 5: 1=Do not process the BASIC line (used by the Calculator).</p> <p>Bit 6: 1=Editing area is the lower screen, 0=Editing area is the main screen.</p> <p>Bit 7: 1=Waiting for key press, 0=Got key press.</p>
\$EC0E	1	<p>Mode:</p> <p>\$00 = Edit Menu mode.</p> <p>\$04 = Calculator mode.</p> <p>\$07 = Tape Loader mode. [Effectively not used as overwritten by \$FF]</p> <p>\$FF = Tape Loader mode.</p>
\$EC0F	1	Main screen colours used by the 128 BASIC Editor - alternate ATTR_P.
\$EC10	1	Main screen colours used by the 128 BASIC Editor - alternate MASK_P.
\$EC11	1	Temporary screen colours used by the 128 BASIC Editor - alternate ATTR_T.
\$EC12	1	Temporary screen colours used by the 128 BASIC Editor - alternate MASK_T.
\$EC13	1	<p>Temporary store for P_FLAG:</p> <p>Bit 0: 1=OVER 1, 0=OVER 0.</p> <p>Bit 1: Not used (always 0).</p> <p>Bit 2: 1=INVERSE 1, INVERSE 0.</p> <p>Bit 3: Not used (always 0).</p> <p>Bit 4: 1=Using INK 9.</p> <p>Bit 5: Not used (always 0).</p> <p>Bit 6: 1=Using PAPER 9.</p> <p>Bit 7: Not used (always 0).</p>
\$EC14	1	Not used.
\$EC15	1	Holds the number of editing lines: 20 for the main screen, 1 for the lower screen.
\$EC16	735	<p>Screen Line Edit Buffer. This represents the text on screen that can be edited. It holds 21 rows,</p> <p>with each row consisting of 32 characters followed by 3 data bytes. Areas of white space that do not contain any editable characters (e.g. the indent that starts subsequent</p> <p>rows of a BASIC line) contain the value \$00.</p> <p>Data Byte 0:</p>

## SPECTRUM +2 ROM 0 DISASSEMBLY

		Bit 0: 1=The first row of the BASIC line.
		Bit 1: 1=Spans onto next row.
		Bit 2: Not used (always 0).
		Bit 3: 1=The last row of the BASIC line.
		Bit 4: 1=Associated line number stored.
		Bit 5: Not used (always 0).
		Bit 6: Not used (always 0).
		Bit 7: Not used (always 0).
		Data Bytes 1-2: Line number of corresponding BASIC line (stored for the first row of the BASIC line only, holds \$0000).
\$EEF5	1	Flags used when listing the BASIC program: Bit 0 : 0=Not on the current line, 1=On the current line. Bit 1 : 0=Previously found the current line, 1=Not yet found the current line. Bit 2 : 0=Enable display file updates, 1=Disable display file updates. Bits 3-7: Not used (always 0).
\$EEF6	1	Store for temporarily saving the value of TVFLAG.
\$EEF7	1	Store for temporarily saving the value of COORDS.
\$EEF9	1	Store for temporarily saving the value of P_POSN.
\$EEFA	2	Store for temporarily saving the value of PR_CC.
\$EEFC	2	Store for temporarily saving the value of ECHO_E.
\$EEFE	2	Store for temporarily saving the value of DF_CC.
\$EF00	2	Store for temporarily saving the value of DF_CCL.
\$EF01	1	Store for temporarily saving the value of S_POSN.
\$EF03	2	Store for temporarily saving the value of SPOSNL.
\$EF05	1	Store for temporarily saving the value of SCR_CT.
\$EF06	1	Store for temporarily saving the value of ATTR_P.
\$EF07	1	Store for temporarily saving the value of MASK_P.
\$EF08	1	Store for temporarily saving the value of ATTR_T.
\$EF09	1512	Used to store screen area (12 rows of 14 columns) where menu will be shown. The rows are stored one after the other, with each row consisting of the following: - 8 lines of 14 display file bytes. - 14 attribute file bytes.
\$F4F1-\$F6E9		Not used. 505 bytes.
\$F6EA	2	The jump table address for the current menu.
\$F6EC	2	The text table address for the current menu.
\$F6EE	1	Cursor position info - Current row number.
\$F6EF	1	Cursor position info - Current column number.
\$F6F0	1	Cursor position info - Preferred column number. Holds the last user selected column position. The Editor will attempt to place the cursor on this column when the user moves up or down to a new line.
\$F6F1	1	Edit area info - Top row threshold for scrolling up.
\$F6F2	1	Edit area info - Bottom row threshold for scrolling down.
\$F6F3	1	Edit area info - Number of rows in the editing area.
\$F6F4	1	Flags used when deleting: Bit 0 : 1=Deleting on last row of the BASIC line, 0=Deleting on row other than the last row of the BASIC line. Bits 1-7: Not used (always 0).
\$F6F5	1	Number of rows held in the Below-Screen Line Edit Buffer.
\$F6F6	2	Intended to point to the next location to access within the Below-Screen Line Edit Buffer, but incorrectly initialised to \$0000 by the routine at \$30FC (ROM 0) and then never used.
\$F6F8	735	Below-Screen Line Edit Buffer. Holds the remainder of a BASIC line that has overflowed off the bottom of the Screen Line Edit Buffer. It can hold 21 rows, with each row consisting of 32 characters followed by 3 data bytes. Areas of white space that do not contain any editable characters (e.g. the indent that starts subsequent rows of a BASIC line) contain the value \$00. Data Byte 0: Bit 0: 1=The first row of the BASIC line. Bit 1: 1=Spans onto next row. Bit 2: Not used (always 0).

## SPECTRUM +2 ROM o DISASSEMBLY

		Bit 3: 1=The last row of the BASIC line.
		Bit 4: 1=Associated line number stored.
		Bit 5: Not used (always 0).
		Bit 6: Not used (always 0).
		Bit 7: Not used (always 0).
		Data Bytes 1-2: Line number of corresponding BASIC line (stored for the first row of the BASIC line only, holds \$0000).
\$F9D7	2	Line number of the BASIC line in the program area being edited (or \$0000 for no line).
\$F9DB	1	Number of rows held in the Above-Screen Line Edit Buffer.
\$F9DC	2	Points to the next location to access within the Above-Screen Line Edit Buffer.
\$F9DE	700	Above-Screen Line Edit Buffer. Holds the rows of a BASIC line that has overflowed off the top of the Screen Line Edit Buffer. It can hold 20 rows, with each row consisting of 32 characters followed by 3 data bytes. Areas of white space that do not contain any editable characters (e.g. the indent that starts subsequent rows of a BASIC line) contain the value \$00. Data Byte 0: Bit 0: 1=The first row of the BASIC line. Bit 1: 1=Spans onto next row. Bit 2: Not used (always 0). Bit 3: 1=The last row of the BASIC line. Bit 4: 1=Associated line number stored. Bit 5: Not used (always 0). Bit 6: Not used (always 0). Bit 7: Not used (always 0). Data Bytes 1-2: Line number of corresponding BASIC line (stored for the first row of the BASIC line only, holds \$0000).
\$FC9A	2	The line number at the top of the screen, or \$0000 for the first line.
\$FC9E	1	\$00=Print a leading space when constructing keyword.
\$FC9F	2	Address of the next character to fetch within the BASIC line in the program area, or \$0000 for no next character.
\$FCA1	2	Address of the next character to fetch from the Keyword Construction Buffer, or \$0000 for no next character.
\$FCA3	11	Keyword Construction Buffer. Holds either a line number or keyword string representation.
\$FCAE-\$FCFC		Construct a BASIC Line routine. « RAM routine - See end of file for description »
\$FCFD-\$FD2D		Copy String Into Keyword Construction Buffer routine. « RAM routine - See end of file for description »
\$FD2E-\$FD69		Identify Character Code of Token String routine. « RAM routine - See end of file for description »
\$FD6A	1	Flags used when shifting BASIC lines within edit buffer rows [Redundant]: Bit 0 : 1=Set to 1 but never reset or tested. Possibly intended to indicate the start of a new BASIC line and hence whether indentation required. Bit 1-7: Not used (always 0).
\$FD6B	1	The number of characters to indent subsequent rows of a BASIC line by.
\$FD6C	1	Cursor settings (indexed by IX+\$00) - initialised to \$00, but never used.
\$FD6D	1	Cursor settings (indexed by IX+\$01) - number of rows above the editing area.
\$FD6E	1	Cursor settings (indexed by IX+\$02) - initialised to \$00 (when using lower screen) or \$14 (when using main screen), but never subsequently used.
\$FD6F	1	Cursor settings (indexed by IX+\$03) - initialised to \$00, but never subsequently used.
\$FD70	1	Cursor settings (indexed by IX+\$04) - initialised to \$00, but never subsequently used.
\$FD71	1	Cursor settings (indexed by IX+\$05) - initialised to \$00, but never subsequently used.
\$FD72	1	Cursor settings (indexed by IX+\$06) - attribute colour.
\$FD73	1	Cursor settings (indexed by IX+\$07) - screen attribute where cursor is displayed.
\$FD74	9	The Keyword Conversion Buffer holding text to examine to see if it is a keyword.
\$FD7D	2	Address of next available location within the Keyword Conversion Buffer.
\$FD7F	2	Address of the space character between words in the Keyword Conversion Buffer.
\$FD81	1	Keyword Conversion Buffer flags, used when tokenizing a BASIC line: Bit 0 : 1=Buffer contains characters. Bit 1 : 1=Indicates within quotes.

## SPECTRUM +2 ROM 0 DISASSEMBLY

		Bit 2 : 1=Indicates within a REM.
		Bits 3-7: Not used (always reset to 0).
\$FD82	2	Address of the position to insert the next character within the BASIC line workspace. The BASIC line
		is created at the spare space pointed to by E_LINE.
\$FD84	1	BASIC line insertion flags, used when inserting a characters into the BASIC line workspace:
		Bit 0 : 1=The last character was a token.
		Bit 1 : 1=The last character was a space.
		Bits 2-7: Not used (always 0).
\$FD85	2	Count of the number of characters in the typed BASIC line being inserted.
\$FD87	2	Count of the number of characters in the tokenized version of the BASIC line being inserted.
\$FD89	1	Holds '<' or '>' if this was the previously examined character during tokenization of a BASIC line, else \$00.
\$FD8A	1	Locate Error Marker flag, holding \$01 is a syntax error was detected on the BASIC line being inserted and the equivalent position within the typed BASIC line needs to be found with, else it holds \$00 when tokenizing a BASIC line.
\$FD8B	2	Stores the stack pointer for restoration upon an insertion error into the BASIC line workspace.
\$FD8C-\$FF23		Not used. 408 bytes.
\$FF24	2	Never used. An attempt is made to set it to \$EC00. This is a remnant from the Spanish 128, which stored the address of the Screen Buffer here. The value is written to RAM bank 0 instead of RAM bank 7, and the value never subsequently accessed.
\$FF26	2	Not used.
\$FF28-\$FF60		Not used. On the Spanish 128 this memory holds a routine that copies a character into the display file. The code to copy to routine into RAM, and the routine itself are present in ROM 0 but are never executed. « RAM routine - See end of file for description »
\$FF61-\$FFFF		Not used. 159 bytes.

## Called ROM 1 Subroutines

```

ERROR_1      EQU $0008
PRINT_A_1    EQU $0010
GET_CHAR     EQU $0018
NEXT_CHAR    EQU $0020
BC_SPACES   EQU $0030
TOKENS       EQU $0095
BEEPER       EQU $03B5
BEEP         EQU $03F8
SA_ALL       EQU $075A
ME_CONTRL    EQU $08B6
SA_CONTROL   EQU $0970
PRINT_OUT    EQU $09F4
PO_T_UDG     EQU $0B52
PO_MSG       EQU $0C0A
TEMPS        EQU $0D4D
CLS          EQU $0D6B
CLS_LOWER    EQU $0D6E
CL_ALL       EQU $0DAF
CL_ATTR      EQU $0E88
CL_ADDR      EQU $0E9B
CLEAR_PRB    EQU $0EDF
ADD_CHAR     EQU $0F81
ED_ERROR     EQU $107F
CLEAR_SP     EQU $1097
KEY_INPUT    EQU $10A8
KEY_M_CL     EQU $10DB
MAIN_4       EQU $1303
ERROR_MSGS   EQU $1391
MESSAGES     EQU $1537

```

## SPECTRUM +2 ROM o DISASSEMBLY

```

REPORT_J      EQU $15C4
OUT_CODE      EQU $15EF
CHAN_OPEN     EQU $1601
CHAN_FLAG     EQU $1615
POINTERS      EQU $1664
CLOSE         EQU $16E5
MAKE_ROOM     EQU $1655
LINE_NO       EQU $1695
SET_MIN       EQU $16B0
SET_WORK      EQU $16BF
SET_STK       EQU $16C5
OPEN          EQU $1736
LIST_5        EQU $1822
NUMBER        EQU $18B6
LINE_ADDR     EQU $196E
EACH_STMT     EQU $198B
NEXT_ONE      EQU $19B8
RECLAIM       EQU $19E5
RECLAIM_2     EQU $19E8
E_LINE_NO     EQU $19FB
OUT_NUM_1     EQU $1A1B
CLASS_01      EQU $1C1F
VAL_FET_1     EQU $1C56
CLASS_04      EQU $1C6C
EXPT_2NUM     EQU $1C7A
EXPT_1NUM     EQU $1C82
EXPT_EXP      EQU $1C8C
CLASS_09      EQU $1CBE
FETCH_NUM     EQU $1CDE
USE_ZERO      EQU $1CE6
STOP          EQU $1CEE
F_REORDER     EQU $1D16
LOOK_PROG     EQU $1D86
NEXT          EQU $1DAB
PASS_BY       EQU $1E39
RESTORE       EQU $1E42
REST_RUN      EQU $1E45
RANDOMIZE      EQU $1E4F
CONTINUE      EQU $1E5F
GO_TO         EQU $1E67
COUT          EQU $1E7A
POKE          EQU $1E80
FIND_INT2     EQU $1E99
TEST_ROOM     EQU $1F05
PAUSE         EQU $1F3A
PRINT_2       EQU $1FDF
PR_ST_END     EQU $2048
STR_ALTER     EQU $2070
INPUT_1       EQU $2096
IN_ITEM_1     EQU $20C1
CO_TEMP_4     EQU $21FC
BORDER        EQU $2294
PIXEL_ADDR    EQU $22AA
PLOT          EQU $22DC
PLOT_SUB      EQU $22E5
CIRCLE        EQU $2320
DR_3_PRMS     EQU $238D
LINE_DRAW     EQU $2477
SCANNING      EQU $24FB
SYNTAX_Z      EQU $2530
LOOK_VARS     EQU $28B2
STK_VAR       EQU $2996
STK_FETCH     EQU $2BF1
D_RUN         EQU $2C15
ALPHA         EQU $2C8D
NUMERIC       EQU $2D1B
STACK_BC      EQU $2D2B

```

Should be OUT but renamed since some assemblers detect this as an instruction.

```

FP_TO_BC      EQU $2DA2
PRINT_FP      EQU $2DE3
HL_MULT_DE    EQU $30A9
STACK_NUM     EQU $33B4
TEST_ZERO     EQU $34E9
KP_SCAN       EQU $3C01
TEST_SCREEN   EQU $3C04
CHAR_SET      EQU $3D00

```

## RESTART ROUTINES — PART 1

RST \$10, \$18 and \$20 call the equivalent subroutines in ROM 1, via RST \$28.

RST \$00 - Reset the machine.

RST \$08 - Not used. Would have invoked the ZX Interface 1 if fitted.

RST \$10 - Print a character (equivalent to RST \$10 ROM 1).

RST \$18 - Collect a character (equivalent to RST \$18 ROM 1).

RST \$20 - Collect next character (equivalent to RST \$20 ROM 1).

RST \$28 - Call routine in ROM 1.

RST \$30 - Not used.

RST \$38 - Not used.

### RST \$00 — Reset Machine

L0000:	ORG \$0000 DI LD BC,\$692B DEC BC LD A,B OR C JR NZ,L0004  JP L00C7 DEFB \$00, \$00 DEFB \$00, \$00	Ensure interrupts are disabled.   Delay about 0.2s to allow screen switching mechanism to settle.   [There is no RST \$08. No instruction fetch at \$0008 hence ZX Interface 1 will not be paged in from this ROM. Credit: Paul Farrow]. to the main reset routine. [Spare bytes]
--------	---	---

### RST \$10 — Print A Character

L0010:	RST 28H DEFW PRINT_A_1 RET	Call corresponding routine in ROM 1. \$0010.
L0014:	DEFB \$00, \$00 DEFB \$00, \$00	[Spare bytes]

### RST \$18 — Collect A Character

L0018:	RST 28H DEFW GET_CHAR RET	Call corresponding routine in ROM 1. \$0018.
L001C:	DEFB \$00, \$00 DEFB \$00, \$00	[Spare bytes]

### RST \$20 — Collect Next Character

L0020:	RST 28H DEFW NEXT_CHAR RET	Call corresponding routine in ROM 1. \$0020.
L0024:	DEFB \$00, \$00	[Spare bytes]

DEFB \$00, \$00

## RST \$28 — Call Routine in ROM 1

RST 28 calls a routine in ROM 1 (or alternatively a routine in RAM while ROM 1 is paged in). Call as follows: RST 28 / DEFW address.

L0028:	EX (SP),HL	Get the address after the RST \$28 into HL, saving HL on the stack.
	PUSH AF	Save the AF registers.
	LD A,(HL)	Fetch the first address byte.
	INC HL	Point HL to the byte after
	INC HL	the required address.
	LD (RETADDR),HL	\$5B5A. Store this in RETADDR.
	DEC HL	(There is no RST \$30)
	LD H,(HL)	Fetch the second address byte.
	LD L,A	HL=Subroutine to call.
	POP AF	Restore AF.
	JP L005C	Jump ahead to continue.
L0037:	DEFB \$00	[Spare byte]

## MASKABLE INTERRUPT ROUTINE

This routine preserves the HL register pair. It then performs the following: - Execute the ROM switching code held in RAM to switch to ROM 1.

- Execute the maskable interrupt routine in ROM 1.
- Execute the ROM switching code held in RAM to return to ROM 0.
- Return to address \$0048 (ROM 0).

L0038:	PUSH HL	Save HL register pair.
	LD HL,L0048	Return address of \$0048 (ROM 0).
	PUSH HL	
	LD HL,SWAP	\$5B00. Address of swap ROM routine held in RAM at \$5B00.
	PUSH HL	
	LD HL,L0038	Maskable interrupt routine address \$0038 (ROM 0).
	PUSH HL	
	JP SWAP	\$5B00. Switch to other ROM (ROM 1) via routine held in RAM at \$5B00.
L0048:	POP HL	Restore the HL register pair.
	RET	End of interrupt routine.

## ERROR HANDLER ROUTINES — PART 1

### 128K Error Routine

L004A:	LD BC,\$7FFD	
	XOR A	ROM 0, Bank 0, Screen 0, 128K mode.
	DI	Ensure interrupts are disabled whilst paging.
	OUT (C),A	
	LD (BANK_M),A	\$5B5C. Note the new paging status.
	EI	Re-enable interrupts.
	DEC A	A=\$FF.
	LD (IY+\$00),A	Set ERR_NR to no error (\$FF).
	JP L0321	Jump ahead to continue.

## RESTART ROUTINES — PART 2

### Call ROM 1 Routine (RST \$28 Continuation)

Continuation from routine at \$0028 (ROM 0).

L005C:	LD (TARGET),HL	\$5B58. Save the address in ROM 0 to call.
--------	----------------	--

```
LD HL,YOUNGER
EX (SP),HL
PUSH HL
LD HL,(TARGET)
EX (SP),HL
JP SWAP
```

```
$5B14. HL='Return to ROM 0' routine held in RAM.
Stack HL.
Save previous stack address.
$5B58. HL=Retrieve address to call. [There is no NMI code. Credit: Andrew Owen].
Stack HL.
$5B00. Switch to other ROM (ROM 1) and return to address to call.
```

## RAM ROUTINES

The following code will be copied to locations \$5B00 to \$5B57, within the old ZX Printer buffer.

### Swap to Other ROM (copied to \$5B00)

Switch to the other ROM from that currently paged in.

[The switching between the two ROMs invariably enables interrupts, which may not always be desired (see the bug at \$09EC (ROM 0) in the PLAY command). To overcome this issue would require a rewrite of the SWAP routine as follows, but this is larger than the existing routine and so cannot simply be used in direct replacement of it. A work-around solution is to poke a JP instruction at the start of the SWAP routine in the ZX Printer buffer and direct control to the replacement routine held somewhere else in RAM. Credit: Toni Baker, ZX Computing Monthly] [However, the PLAY command bug may be fixed in another manner within the PLAY command itself, in which case there is no need to modify the SWAP routine.]

SWAP:

```
PUSH AF          Stack AF.
PUSH BC          Stack BC.
LD A,R           P/V flag=Interrupt status.
PUSH AF          Stack interrupt status.
LD BC,$7FFD      BC=Port number required for paging.
LD A,(BANK_M)    A=Current paging configuration.
XOR $10          Complement 'ROM' bit.
DI              Disable interrupts (in case an interrupt occurs between the next two instructions).
LD (BANK_M),A    Store revised paging configuration.
OUT (C),A        Page ROM.
POP AF           P/V flag=Former interrupt status.
JP PO,SWAP_EXIT  Jump if interrupts were previously disabled.
EI              Re-enable interrupts.
```

SWAP\_EXIT:

```
POP BC           Restore BC.
POP AF           Restore AF.
RET
```

L006B:

```
PUSH AF          Save AF and BC.
PUSH BC
LD BC,$7FFD
LD A,(BANK_M)    $5B5C.
XOR $10          Select other ROM.
DI              Disable interrupts whilst switching ROMs.
LD (BANK_M),A    $5B5C.
OUT (C),A        Switch to the other ROM.
EI
POP BC           Restore BC and AF.
POP AF
RET
```

### Return to Other ROM Routine (copied to \$5B14)

Switch to the other ROM from that currently paged in  
and then return to the address held in RETADDR.  
YOUNGER

```
L007F:          CALL SWAP          $5B00. Toggle to the other ROM.
                PUSH HL
                LD HL,(RETADDR)    $5B5A.
                EX (SP),HL
```

RET

Return to the address held in RETADDR.

**Error Handler Routine (copied to \$5B1D)**

This error handler routine switches back to ROM 0 and then executes the routine pointed to by system variable TARGET.

ONERR

L0088:	DI	Ensure interrupts are disabled whilst paging.
	LD A,(BANK_M)	\$5B5C. Fetch current paging configuration.
	AND \$EF	Select ROM 0.
	LD (BANK_M),A	\$5B5C. Save the new configuration
	LD BC,\$7FFD	
	OUT (C),A	Switch to ROM 0.
	EI	
	JP L00C3	Jump to \$00C3 (ROM 0) to continue.

**'P' Channel Input Routine (copied to \$5B2F)**

Called when data is read from channel 'P'.

It causes ROM 0 to be paged in so that the new RS232 routines can be accessed.

PIN

L009A:	LD HL,L06F7	RS232 input routine within ROM 0.
	JR L00A2	

**'P' Channel Output Routine (copied to \$5B34)**

Called when data is written to channel 'P'.

It causes ROM 0 to be paged in so that the new RS232 routines can be accessed.

Entry: A=Byte to send.

POUT

L009F:	LD HL,L07E9	RS232 output routine within ROM 0.
L00A2:	EX AF,AF'	Save AF registers.
	LD BC,\$7FFD	
	LD A,(BANK_M)	\$5B5C. Fetch the current paging configuration
	PUSH AF	and save it.
	AND \$EF	Select ROM 0.
	DI	Ensure interrupts are disabled whilst paging.
	LD (BANK_M),A	\$5B5C. Store the new paging configuration.
	OUT (C),A	Switch to ROM 0.
	JP L0605	Jump to the RS232 channel input/output handler routine.

**'P' Channel Exit Routine (copied to \$5B4A)**

Used when returning from a channel 'P' read or write operation.

It causes the original ROM to be paged back in and returns back to the calling routine.

POUT2

L00B5:	EX AF,AF'	Save AF registers. For a read, A holds the byte read and the flags the success status.
	POP AF	Retrieve original paging configuration.
	LD BC,\$7FFD	
	DI	Ensure interrupts are disabled whilst paging.
	LD (BANK_M),A	\$5B5C. Store original paging configuration.
	OUT (C),A	Switch back to original paging configuration.
	EI	
	EX AF,AF'	Restore AF registers. For a read, A holds the byte read and the flags the success status.
	RET	« End of RAM Routines »

## ERROR HANDLER ROUTINES — PART 2

### Call Subroutine

Called from ONERR (\$5B1D) to execute the routine pointed to by system variable SYNRET.

L00C3:	LD HL,(SYNRET) JP (HL)	\$5B8B. Fetch the address to call. and execute it.
--------	---------------------------	---

## INITIALISATION ROUTINES — PART 1

### Reset Routine (RST \$00 Continuation, Part 1)

Continuation from routine at \$0000 (ROM 0). It performs a test on all RAM banks. This test is crude and can fail to detect a variety of RAM errors.

L00C7:	LD B,\$08	Loop through all RAM banks.
L00C9:	LD A,B EXX DEC A LD BC,\$7FFD OUT (C),A LD HL,\$C000 LD DE,\$C001 LD BC,\$3FFF LD A,\$FF LD (HL),A CP (HL) JR NZ,L0131 XOR A LD (HL),A CP (HL) JR NZ,L0131 LDIR EXX DJNZ L00C9 LD (ROW01),A LD C,\$FD LD D,\$FF LD E,\$BF LD B,D LD A,\$0E OUT (C),A LD B,E LD A,\$FF OUT (C),A  JR L0137	Save B register. RAM bank number 0 to 7. 128K mode, ROM 0, Screen 0.  Switch RAM bank. Start of the current RAM bank.  All 16K of RAM bank.  Store \$FF into RAM location. Check RAM integrity. Jump if RAM error found.  Store \$00 into RAM location. Check RAM integrity. Jump if difference found. Clear the whole page Restore B registers. Repeat for other RAM banks. \$5B88. Signal no communications in progress to the keypad.    BC=\$FFFD, DE=\$FFBF.  Select AY register 14. BC=\$BFFD.  Set AY register 14 to \$FF. This will force a communications reset to the keypad if present. Jump ahead to continue.
L00FF:	DEFB \$00	[Spare byte]

## ROUTINE VECTOR TABLE

L0100:	JP L17CE	BASIC interpreter parser.
L0103:	JP L1857	'Line Run' entry point.
L0106:	JP L1EEE	Transfer bytes to logical RAM bank 4.
L0109:	JP L1F23	Transfer bytes from logical RAM bank 4.
L010C:	JP L004A	128K error routine.
L010F:	JP L03A2	Error routine. Called from patch at \$3B3B in ROM 1.

## SPECTRUM +2 ROM 0 DISASSEMBLY

L0112:	JP L1849	'Statement Return' routine. Called from patch at \$3B4D in ROM 1.
L0115:	JP L18C7	'Statement Next' routine. Called from patch at \$3B5D in ROM 1.
L0118:	JP L012D	Scan the keypad.
L011B:	JP L0A24	Play music strings.
L011E:	JP L11C2	MIDI byte output routine.
L0121:	JP L06F7	RS232 byte input routine.
L0124:	JP L07E9	RS232 text output routine.
L0127:	JP L08C2	RS232 byte output routine.
L012A:	JP L090F	COPY (screen dump) routine.
L012D:	RST 28H	Call keypad scan routine in ROM 1.
	DEFW KP_SCAN-\$0100	\$3B01. <b>[BUG]</b> - The address jumps into the middle of the keypad decode routine in ROM 1. It
	RET	looks like it is supposed to deal with the keypad and so the most likely addresses are \$3A42 (read keypad) or \$39A0 (scan keypad). At \$3C01 in ROM 1 is a vector jump command to \$39A0 to scan the keypad and this is similar enough to the \$3B01 to imply a simple error in one of the bytes. Credit: Paul Farrow]

## INITIALISATION ROUTINES — PART 2

### Fatal RAM Error

Set the border colour to indicate which RAM bank was found faulty: RAM bank 7 - Black.

RAM bank 6 - White.

RAM bank 5 - Yellow.

RAM bank 4 - Cyan.

RAM bank 3 - Green.

RAM bank 2 - Magenta.

RAM bank 1 - Red.

RAM bank 0 - Blue.

L0131:	EXX	Retrieve RAM bank number + 1 in B.
	LD A,B	Indicate which RAM bank failed by
	OUT (\$FE),A	setting the border colour.
L0135:	JR L0135	Infinite loop.

### Reset Routine (RST \$00 Continuation, Part 2)

Continuation from routine at \$00C7 (ROM 0).

L0137:	LD B,D	Complete setting up the sound chip registers.
	LD A,\$07	
	OUT (C),A	Select AY register 7.
	LD B,E	
	LD A,\$FF	Disable AY-3-8912 sound channels.
	OUT (C),A	
	LD DE,SWAP	\$5B00. Copy the various paging routines to the old printer buffer.
	LD HL,L006B	The source is in this ROM.
	LD BC,\$0058	There are eighty eight bytes to copy.
	LDIR	Copy the block of bytes.
	LD A,\$CF	Load A with the code for the Z80 instruction 'RST \$08'.
	LD (RAMRST),A	\$5B5D. Insert into new System Variable RAMRST.
	LD SP,TSTACK	\$5BFF. Set the stack pointer to last location of old buffer.
	LD A,\$04	
	CALL L1C83	Page in logical RAM bank 4 (physical RAM bank 7).
	LD IX,\$EBEC	First free entry in RAM disk.
	LD (SFNEXT),IX	\$5B83.
	LD (IX+\$0A),\$00	
	LD (IX+\$0B),\$C0	
	LD (IX+\$0C),\$00	
	LD HL,\$2BEC	
	LD A,\$01	AHL=Free space in RAM disk.
	LD (SFSPACE),HL	\$5B85. Current address.
	LD (SFSPACE+2),A	\$5B87. Current RAM bank.

## SPECTRUM +2 ROM 0 DISASSEMBLY

<p>LD A,\$05 CALL L1C83 LD HL,\$FFFF LD (\$5CB4),HL LD DE,CHAR_SET+\$01AF LD BC,\$00A8 EX DE,HL RST 28H DEFW MAKE_ROOM+\$000C  EX DE,HL INC HL LD (\$5C7B),HL DEC HL LD BC,\$0040 LD (\$5C38),BC LD (\$5CB2),HL</p>	<p>Page in logical RAM bank 5 (physical RAM bank 0). Load HL with known last working byte - 65535. P_RAMT. Set physical RAM top to 65535. \$3EAF. Set DE to address of the last bitmap of 'U' in ROM 1. There are 21 User Defined Graphics to copy. Swap so destination is \$FFFF.  Calling this address (LDDR/RET) in the main ROM cleverly copies the 21 characters to the end of RAM. Transfer DE to HL. Increment to address first byte of UDG 'A'. UDG. Update standard System Variable UDG.  Set values 0 for PIP and 64 for RASP. RASP. Update standard System Variables RASP and PIP. RAMTOP. Update standard System Variable RAMTOP - the last byte of the BASIC system area. Any machine code and graphics above this address are protected from NEW.</p>
---	---

Entry point for NEW with interrupts disabled and physical RAM bank 0 occupying the upper RAM region \$C000 - \$FFFF, i.e. the normal BASIC memory configuration.

<p>L019D: LD HL,CHAR_SET-\$0100 LD (\$5C36),HL LD HL,(\$5CB2) INC HL LD SP,HL IM 1 LD IY,\$5C3A  SET 4,(IY+\$01)  EI LD HL,\$000B LD (BAUD),HL XOR A LD (SERFL),A LD (COL),A LD (TVPARS),A LD HL,\$EC00  LD (\$FF24),HL    LD A,\$50 LD (WIDTH),A LD HL,\$000A LD (RNFIRST),HL LD (RNSTEP),HL LD HL,\$5CB6 LD (\$5C4F),HL LD DE,L05A8  LD BC,\$0015 EX DE,HL LDIR EX DE,HL DEC HL LD (\$5C57),HL</p>	<p>\$3C00. Set HL to where, in theory character zero would be. CHARS. Update standard System Variable CHARS. RAMTOP. Load HL with value of System Variable RAMTOP. Address next location. Set the Stack Pointer. Select Interrupt Mode 1. Set the IY register to address the standard System Variables and many of the new System Variables and even those of ZX Interface 1 in some cases. FLAGS. Signal 128K mode. [This bit was unused and therefore never set by 48K BASIC] With a stack and the IY register set, interrupts can be enabled. Set HL to eleven, timing constant for 9600 baud. \$5B5F. Select default RS232 baud rate of 9600 baud. Clear accumulator. \$5B61. Indicate no byte waiting in RS232 receive buffer. \$5B63. Set RS232 output column position to 0. \$5B65. Indicate no control code parameters expected. [BUG - Should write to RAM bank 7. Main RAM has now been corrupted. The value stored is subsequently never used. Credit: Geoff Wearmouth] This is a remnant from the Spanish 128, which used this workspace variable to hold the location of the Screen Buffer, but it also suffered from this bug. In fact there was never a need to write to the value at this point since it is written again later during the initialisation process. [The 1985 Sinclair Research ESPAGNOL source code says that this instruction will write to the (previously cleared) main BASIC RAM during initialization but that a different page of RAM will be present during NEW. Stuff and Nonsense! Assemblers and other utilities present above RAMTOP will be corrupted by the BASIC NEW command since \$FF24, and later \$EC13, will be written to even if they are above RAMTOP.] Default to a printer width of 80 columns. \$5B64. Set RS232 printer output width. Use 10 as the initial renumber line and increment. \$5B94. Store the initial line number when renumbering. \$5B96. Store the renumber line increment. Address after the System Variables. CHANS. Set the default location for the channel area. Point to Initial Channel Information in this ROM. This is similar to that in main ROM but channel 'P' has input and output addresses in the new \$5Bxx region. There are 21 bytes to copy. Switch pointer so destination is CHANS. Copy the block of bytes.  Decrement to point to channel information end-marker. DATADD. Set the default address of the terminator for the last DATA item.</p>
--	--

```

INC HL
LD ($5C53),HL
LD ($5C4B),HL
LD (HL),$80
INC HL
LD ($5C59),HL
LD (HL),$0D
INC HL
LD (HL),$80
INC HL
LD ($5C61),HL
LD ($5C63),HL
LD ($5C65),HL
LD A,$38
LD ($5C8D),A
LD ($5C8F),A
LD ($5C48),A
XOR A
LD ($EC13),A

LD A,$07
OUT ($FE),A
LD HL,$0523
LD ($5C09),HL
DEC (IY-$3A)
DEC (IY-$36)
LD HL,L05BD

LD DE,$5C10

LD BC,$000E
LDIR
RES 1,(IY+$01)
LD (IY+$00),$FF
LD (IY+$31),$02
RST 28H
DEFW CLS
RST 28H
DEFW TEST_SCREEN
LD DE,L0561
CALL L059C
LD (IY+$31),$02
SET 5,(IY+$02)
LD HL,TSTACK
LD (OLDSP),HL
CALL L1F64
LD A,$38
LD ($EC11),A
LD ($EC0F),A

```

PROG. Set the default address of the BASIC program area.  
 VARS. Set the default address of the BASIC variables area.  
 Insert the Variables end-marker.

E\_LINE. Set the default address of the editing line area.  
 Insert a carriage return.

Insert the editing line end-marker.

WORKSP. Set the address of the workspace.  
 STKBOT. Set the address of the start of the calculator stack.  
 STKEND. Set the address of the end of the calculator stack.  
 Attribute colour of black ink on white paper.  
 ATTR\_P. Set the permanent attribute colour.  
 MASK\_P. Set the permanent attribute mask.  
 BORDCR. Set the default border colour.

Temporary P\_FLAG. Clear the temporary store for P-FLAG. **[BUG** - Should write this to RAM bank 7. Main RAM has now been corrupted again. The effect of the bug can be seen by typing INVERSE 1: PRINT "Hello", followed by NEW, followed by PRINT "World", and will cause the second word to also be printed in inverse. Credit: Geoff Wearmouth]

Set the border white.  
 The values five and thirty five.  
 REPDEL. Set the default values for key delay and key repeat.  
 Set KSTATE+0 to \$FF.  
 Set KSTATE+4 to \$FF.  
 Address of the Initial Stream Data within this ROM (which is identical to that in main ROM).  
 STRMS. Address of the system variable holding the channels attached to streams data.

Initialise the streams system variables.  
 FLAGS. Signal printer not is use.  
 ERR\_NR. Signal no error.  
 DF\_SZ. Set the lower screen size to two rows.

\$0D6B. Clear the screen.  
 Attempt to display TV tuning test screen.  
 \$3C04. Will return if BREAK is not being pressed.  
 Address of the Sinclair copyright message.  
 Display the copyright message.  
 DF\_SZ. Set the lower screen size to two rows.  
 TV\_FLAG. Signal lower screen will require clearing.  
 \$5BFF.  
 \$5B81. Use the temporary stack as the previous stack.  
 Use Workspace RAM configuration (physical RAM bank 7).  
 Set colours to black ink on white paper.  
 Temporary ATTR\_T used by the 128 BASIC Editor.  
 Temporary ATTR\_P used by the 128 BASIC Editor.

[Note this is where \$EC13 (temporary P\_FLAG) and \$FF24 should be set]

```

CALL L25A3
CALL L1F3F
JP L25BE

```

Initialise mode and cursor settings. IX will point at editing settings information.  
 Use Normal RAM Configuration (physical RAM bank 0).  
 Jump to show the Main menu.

## COMMAND EXECUTION ROUTINES — PART 1

### Execute Command Line

A typed in command resides in the editing workspace. Execute it.

## SPECTRUM +2 ROM 0 DISASSEMBLY

The command could either be a new line to insert, or a line number to delete, or a numerical expression to evaluate.

L026B:	LD HL,FLAGS3 SET 0,(HL) LD (IY+\$00),\$FF LD (IY+\$31),\$02 LD HL,ONERR PUSH HL LD (\$5C3D),SP LD HL,L02BA LD (SYNRET),HL CALL L22AD CALL L22EA JP Z,L2217 CP '(' JP Z,L2217 CP '-' JP Z,L2217 CP '+' JP Z,L2217 CALL L22FF JP Z,L2217 CALL L1F64 LD A,(\$EC0E) CALL L1F3F CP \$04 JP NZ,L17CE	\$5B66. Select BASIC/Calculator mode. ERR_NR. Set to '0 OK' status. DF_SZ. Reset the number of rows in the lower screen. \$5B1D. Return address should an error occur. Stack it. Save the stack pointer in ERR_SP. Return address in ROM 0 after syntax checking. \$5B8B. Store it in SYNRET. Point to start of typed in BASIC command. Is the first character a function token, i.e. the start of a numerical expression? Jump if so to evaluate it. \$28. Is the first character the start of an expression? Jump if so to evaluate it. \$2D. Is the first character the start of an expression? Jump if so to evaluate it. \$2B. Is the first character the start of an expression? Jump if so to evaluate it. Is text just a number or a numerical expression? Jump if a numerical expression to evaluate it. Use Workspace RAM configuration (physical RAM bank 7). Fetch mode. Use Normal RAM Configuration (physical RAM bank 0). Calculator mode? Jump if not to parse and execute the BASIC command line, returning to \$02BA (ROM 0).
--------	--	---

Calculator mode

CALL L22B6 JP Z,L17CE	Is it a single LET command? Jump if so to parse and execute the BASIC command line, returning to \$02BA (ROM 0).
--------------------------	---

Otherwise ignore the command

POP HL RET	Drop ONERR return address.
---------------	----------------------------

## Return from BASIC Line Syntax Check

This routine is returned to when a BASIC line has been syntax checked.

L02BA:	BIT 7,(IY+\$00) JR NZ,L02C1 RET	Test ERR_NR. Jump ahead if no error. Simply return if an error.
--------	---------------------------------------	---

The syntax check was successful, so now proceed to parse the line for insertion or execution

L02C1:	LD HL,(\$5C59) LD (\$5C5D),HL RST 28H DEFW E_LINE_NO LD A,B OR C JP NZ,L03F7	ELINE. Point to start of editing area. Store in CH_ADD. \$19FB. Call E_LINE_NO in ROM 1 to read the line number into editing area.  Jump ahead if there was a line number.
--------	--	--

## Parse a BASIC Line with No Line Number

RST 18H CP \$0D	Get character. End of the line reached, i.e. no BASIC statement?
--------------------	---

	RET Z	Return if so.
	CALL L220E	Clear screen if it requires it.
	BIT 6,(IY+\$02)	TVFLAG. Clear lower screen?
	JR NZ,L02DF	Jump ahead if no need to clear lower screen.
	RST 28H	
	DEFW CLS_LOWER	\$0D6E. Clear the lower screen.
L02DF:	RES 6,(IY+\$02)	TVFLAG. Signal to clear lower screen.
	CALL L1F64	Use Workspace RAM configuration (physical RAM bank 7).
	LD HL,\$EC0D	Editor flags.
	BIT 6,(HL)	Using lower screen area for editing?
	JR NZ,L02F4	Jump ahead if so.
	INC HL	
	LD A,(HL)	Fetch the mode.
	CP \$00	In Edit Menu mode?
	CALL Z,L38A2	If so then clear lower editing area display.
L02F4:	CALL L1F3F	Use Normal RAM Configuration (physical RAM bank 0).
	LD HL,\$5C3C	TVFLAG.
	RES 3,(HL)	Signal mode has not changed.
	LD A,\$19	25.
	SUB (IY+\$4F)	S_POSN+1. Subtract the current print row position.
	LD (\$5C8C),A	SCR_CT. Set the number of scrolls.
	SET 7,(IY+\$01)	FLAGS. Not syntax checking.
	LD (IY+\$0A),\$01	NSPPC. Set line to be jumped to as line 1.

**[BUG** - Whenever a typed in command is executed directly from the editing workspace, a new GO SUB marker is set up on the stack. Any existing GO SUB calls that were on the stack are lost and as a result attempting to continue the program (without the use of CLEAR or RUN) will likely lead to a "7 RETURN without GOSUB" error report message being displayed. However, the stack marker will already have been lost due to the error handler routine at \$0321. The first action it does is to reset the stack pointer to point to the location of RAMTOP, i.e. after the GO SUB marker. This is why it is necessary for a new GO SUB marker needs to be set up. Credit: Michal Skrzypek]

	LD HL,\$3E00	The end of GO SUB stack marker.
	PUSH HL	Place it on the stack.
	LD HL,ONERR	\$5B1D. The return address should an error occur.
	PUSH HL	Place it on the stack.
	LD (\$5C3D),SP	ERR_SP. Store error routine address.
	LD HL,L0321	Address of error handler routine in ROM 0.
	LD (SYNRET),HL	\$5B8B. Store it in SYNRET.
	JP L1857	Jump ahead to the main parser routine to execute the line.

## ERROR HANDLER ROUTINES — PART 3

### Error Handler Routine

**[BUG** - Upon terminating a BASIC program, either via reaching the end of the program or due to an error occurring, execution is passed to this routine. The first action it does is to reset the stack pointer to point to the location of RAMTOP, i.e. after the GO SUB marker. However, this means that any existing GO SUB calls that were on the stack are lost and so attempting to continue the program (without the use of CLEAR or RUN) will likely lead to a "7 RETURN without GOSUB" error report message being displayed. When a new typed in command is executed, the code at \$030C sets up a new GO SUB marker on the stack. Credit: Michal Skrzypek]

L0321:	LD SP,\$5CB2)	RAMTOP.
	INC SP	Reset SP to top of memory map.
	LD HL,TSTACK	\$5BFF.
	LD (OLDSP),HL	\$5B81. Use the temporary stack as the previous stack.
	HALT	Trap error conditions where interrupts are disabled.
	RES 5,(IY+\$01)	FLAGS. Signal no new key.
	LD HL,FLAGS3	\$5B66.
	BIT 2,(HL)	Editing RAM disk catalogue?
	JR Z,L034A	Jump if not.
	CALL L1F64	Use Workspace RAM configuration (physical RAM bank 7).
	LD IX,(SFNEXT)	\$5B83.
	LD BC,\$0014	Catalogue entry size.
	ADD IX,BC	Remove last entry.
	CALL L1D75	Update catalogue entry (leaves logical RAM bank 4 paged in).
	CALL L1F3F	Use Normal RAM Configuration (physical RAM bank 0).

## SPECTRUM +2 ROM 0 DISASSEMBLY

Display error code held in ERR\_NR

L034A:	LD A,(\$5C3A)	Fetch error number from ERR_NR.
	INC A	Increment to give true error code.
L034E:	PUSH AF	Save the error code.
	LD HL,\$0000	
	LD (IY+\$37),H	FLAGX. Ensure not INPUT mode.
	LD (IY+\$26),H	X_PTR_hi. Clear to suppress error '?' marker.
	LD (\$5C0B),HL	DEFADD. Clear to signal no defined function is currently being evaluated.
	LD HL,\$0001	[Could have saved 2 bytes by using INC L].
	LD (\$5C16),HL	STRMS+\$0006. Ensure STRMS-00 specifies the keyboard.
	RST 28H	
	DEFW SET_MIN	\$16B0. Clears editing area and areas after it.
	RES 5,(IY+\$37)	FLAGX. Signal not INPUT mode. [Redundant since all flags were reset earlier]
	RST 28H	
	DEFW CLS_LOWER	\$0D6E. Clear lower editing screen.
	SET 5,(IY+\$02)	TVFLAG. Signal lower screen requires clearing.
	POP AF	Retrieve error code.
	LD B,A	Store error code in B.
	CP \$0A	Is it a numeric error code (1-9), i.e. suitable for immediate display?
	JR C,L037F	If so jump ahead to display it.
	CP \$1D	Is it one of the standard errors (A-R)?
	JR C,L037D	If so jump ahead to convert it into an upper case letter.
	ADD A,\$14	Otherwise convert it into a lower case letter.
	JR L037F	Jump ahead to display it. [Could have saved 2 bytes by using ADD A,\$0C instead of these two instructions]
L037D:	ADD A,\$07	Increase code to point to upper case letters.
L037F:	RST 28H	
	DEFW OUT_CODE	
	LD A,\$20	\$15EF. Display the character held in the A register.
	RST 10H	Display a space.
	LD A,B	
	CP \$1D	Retrieve the error code.
	JR C,L039C	Is it one of the standard errors (A-R)?
		Jump if an standard error message (A-R).

Display a new error message

[Note that there is no test to range check the error code value and therefore whether a message exists for it. Poking directly to system variable ERR\_NR with an invalid code (43 or above) will more than likely cause a crash]

SUB \$1D	A=Code \$00 - \$0E.
LD B,\$00	
LD C,A	Pass code to BC.
LD HL,L046C	Error message vector table.
ADD HL,BC	
ADD HL,BC	Find address in error message vector table.
LD E,(HL)	
INC HL	
LD D,(HL)	DE=Address of message to print.
CALL L059C	Print error message.
JR L03A2	Jump ahead.

Display a standard error message.

L039C:	LD DE,ERROR_MSGS	\$1391. Position of the error messages in ROM 1.
	RST 28H	A holds the error code.
	DEFW PO_MSG	\$0C0A. Call message printing routine.

Continue to display the line and statement number

L03A2:	XOR A	Select the first message " , " (a 'comma' and a 'space').
	LD DE,MESSAGES-1	\$1536. Message base address in ROM 1.
	RST 28H	
	DEFW PO_MSG	Print a comma followed by a space.
	LD BC,(\$5C45)	PPC. Fetch current line number.
	RST 28H	

## SPECTRUM +2 ROM o DISASSEMBLY

	DEFW OUT_NUM_1	\$1A1B. Print the line number.
	LD A,\$3A	Print ':'.
	RST 10H	
	LD C,(IY+\$0D)	SUBPPC. Fetch current statement number.
	LD B,\$00	
	RST 28H	
	DEFW OUT_NUM_1	\$1A1B. Print the statement number.
	RST 28H	
	DEFW CLEAR_SP	\$1097. Clear editing and workspace areas.
	LD A,(\$5C3A)	ERR_NR. Fetch the error code.
	INC A	
	JR Z,L03DF	Jump ahead for "0 OK".
	CP \$09	
	JR Z,L03CC	Jump for "A Invalid argument", thereby advancing to the next statement.
	CP \$15	
	JR NZ,L03CF	Jump unless "M Ramtop no good".
L03CC:	INC (IY+\$0D)	SUBPPC. Advance to the next statement.
L03CF:	LD BC,\$0003	
	LD DE,\$5C70	OSPPC. Continue statement number.
	LD HL,\$5C44	NSPPC. Next statement number.
	BIT 7,(HL)	Is there a statement number?
	JR Z,L03DD	Jump if so.
	ADD HL,BC	HL=SUBPPC. The current statement number.
L03DD:	LDDR	Copy SUBPPC and PPC to OSPPC and OLDPPC, for use by CONTINUE.
L03DF:	LD (IY+\$0A),\$FF	NSPPC. Signal no current statement number.
	RES 3,(IY+\$01)	FLAGS. Select K-Mode.
	LD HL,FLAGS3	\$5B66.
	RES 0,(HL)	Select 128 Editor mode.
	JP L25EA	Jump ahead to return control to the Editor.

## Error Handler Routine When Parsing BASIC Line

L03EF:	LD A,\$10	Error code 'G - No room for line'.
	LD BC,\$0000	
	JP L034E	Jump to print the error code.

## COMMAND EXECUTION ROUTINES — PART 2

### Parse a BASIC Line with a Line Number

This routine handles insertion of a BASIC line specified with a line number, or just a line number specified on its own, i.e. delete the line.

L03F7:	LD (\$5C49),BC	E_PPC. Store the line as the current line number with the program cursor.
	CALL L1F64	Use Workspace RAM configuration (physical RAM bank 7).
	LD A,B	[This test could have been performed before paging in bank 7 and hence could have benefited from a slight speed improvement.
	OR C	The test is redundant since BC holds a non-zero line number]
	JR Z,L040A	Jump if no line number.
	LD (\$5C49),BC	E_PPC. Current edit line number. [Redundant instruction - Line number has already been stored]
		Temporary E_PPC used by BASIC Editor.
L040A:	LD (\$EC08),BC	Use Normal RAM Configuration (physical RAM bank 0).
	CALL L1F3F	CH_ADD. Point to the next character in the BASIC line.
	LD HL,(\$5C5D)	
	EX DE,HL	
	LD HL,L03EF	Address of error handler routine should there be no room for the line.
	PUSH HL	Stack it.
	LD HL,(\$5C61)	WORKSP.
	SCF	
	SBC HL,DE	HL=Length of BASIC line.
	PUSH HL	Stack it.
	LD H,B	
	LD L,C	Transfer edit line number to HL.

## SPECTRUM +2 ROM 0 DISASSEMBLY

RST 28H  
DEFW LINE\_ADDR  
JR NZ,L0429

\$196E. Returns address of the line in HL.  
Jump if the line does not exist.

The line already exists so delete it

L0429: RST 28H  
DEFW NEXT\_ONE  
RST 28H  
DEFW RECLAIM\_2  
POP BC  
LD A,C  
DEC A  
OR B  
JR NZ,L0442

\$19B8. Find the address of the next line.  
  
\$19E8. Delete the line.  
BC=Length of the BASIC line.  
  
Is it 1, i.e. just an 'Enter' character, and hence only  
a line number was entered?  
Jump if there is a BASIC statement.

Just a line number entered. The requested line has already been deleted so move the program cursor to the next line

L0442: CALL L1F64  
PUSH HL  
LD HL,(\$5C49)  
CALL L3370  
LD (\$5C49),HL  
POP HL  
CALL L1F3F  
JR L046A  
PUSH BC  
INC BC  
INC BC  
INC BC  
INC BC  
DEC HL  
LD DE,(\$5C53)  
PUSH DE  
RST 28H  
DEFW MAKE\_ROOM  
POP HL  
LD (\$5C53),HL  
POP BC  
PUSH BC  
INC DE  
LD HL,(\$5C61)  
DEC HL  
DEC HL  
LDDR  
LD HL,(\$5C49)  
EX DE,HL  
POP BC  
LD (HL),B  
DEC HL  
LD (HL),C  
DEC HL  
LD (HL),E  
DEC HL  
LD (HL),D  
L046A: POP AF  
RET

Use Workspace RAM configuration (physical RAM bank 7).  
Save the address of the line.  
E\_PPC. Fetch current edit line number.  
Find closest line number (or \$0000 if no line).  
E\_PPC. Store current edit line number. Effectively refresh E\_PPC.  
HL=Address of the line.  
Use Normal RAM Configuration (physical RAM bank 0).  
Jump ahead to exit.  
BC=Length of the BASIC line. Stack it.  
  
BC=BC+4. Allow for line number and length bytes.  
Point to before the current line, i.e. the location to insert bytes at.  
PROG. Get start address of the BASIC program.  
Stack it.  
  
\$1655. Insert BC spaces at address HL.  
HL=Start address of BASIC program.  
PROG. Save start address of BASIC program.  
BC=Length of the BASIC line.  
  
Point to the first location of the newly created space.  
WORKSP. Address of end of the BASIC line in the workspace.  
  
Skip over the newline and terminator bytes.  
Copy the BASIC line from the workspace into the program area.  
E\_PPC. Current edit line number.  
  
BC=Length of BASIC line.  
Store the line length.  
  
DE=line number.  
  
Store the line number.  
Drop item (address of error handler routine).  
Exit with HL=Address of the line.

## ERROR HANDLER ROUTINES — PART 4

### New Error Message Vector Table

Pointers into the new error message table.

L046C:	DEFW L048C	Error report 'a'.
	DEFW L0497	Error report 'b'.
	DEFW L04A6	Error report 'c'.
	DEFW L04B0	Error report 'd'.
	DEFW L04C1	Error report 'e'.
	DEFW L04D4	Error report 'f'.
	DEFW L04E0	Error report 'g'.
	DEFW L04E0	Error report 'h'.
	DEFW L04F3	Error report 'i'.
	DEFW L0501	Error report 'j'.
	DEFW L0512	Error report 'k'.
	DEFW L0523	Error report 'l'.
	DEFW L0531	Error report 'm'.
	DEFW L0542	Error report 'n'.
	DEFW L054E	Error report 'o'.
	DEFW L0561	Error report 'p'.

## New Error Message Table

L048C:	DEFM "MERGE erro"	Report 'a'.
	DEFB 'r'+\$80	
L0497:	DEFM "Wrong file typ"	Report 'b'.
	DEFB 'e'+\$80	
L04A6:	DEFM "CODE erro"	Report 'c'.
	DEFB 'r'+\$80	
L04B0:	DEFM "Too many bracket"	Report 'd'.
	DEFB 's'+\$80	
L04C1:	DEFM "File already exist"	Report 'e'.
	DEFB 's'+\$80	
L04D4:	DEFM "Invalid nam"	Report 'f'.
	DEFB 'e'+\$80	
L04E0:	DEFM "File does not exis"	Report 'g' & 'h'.
	DEFB 't'+\$80	
L04F3:	DEFM "Invalid devic"	Report 'i'.
	DEFB 'e'+\$80	
L0501:	DEFM "Invalid baud rat"	Report 'j'.
	DEFB 'e'+\$80	
L0512:	DEFM "Invalid note nam"	Report 'k'.
	DEFB 'e'+\$80	
L0523:	DEFM "Number too bi"	Report 'l'.
	DEFB 'g'+\$80	
L0531:	DEFM "Note out of rang"	Report 'm'.
	DEFB 'e'+\$80	
L0542:	DEFM "Out of rang"	Report 'n'.
	DEFB 'e'+\$80	
L054E:	DEFM "Too many tied note"	Report 'o'.
	DEFB 's'+\$80	
L0561:	DEFB \$7F	(c)
	DEFM "1986, "	
	DEFB \$7F	(c)
	DEFM "1982 Amstrad Consumer" Copyright / Report 'p'.	
	DEFB \$0D	
	DEFM " Electronics pl"	
	DEFB 'c'+\$80	

## Print Message

Print a message which is terminated by having bit 7 set, pointed at by DE.

L059C:	LD A,(DE)	Fetch next byte.
	AND \$7F	Mask off top bit.
	PUSH DE	Save address of current message byte.
	RST 10H	Print character.

POP DE	Restore message byte pointer.
LD A,(DE)	
INC DE	
ADD A,A	Carry flag will be set if byte is \$FF.
JR NC,L059C	Else print next character.
RET	

## INITIALISATION ROUTINES — PART 3

### The 'Initial Channel Information'

Initially there are four channels ('K', 'S', 'R', & 'P') for communicating with the 'keyboard', 'screen', 'work space' and 'printer'.

For each channel the output routine address comes before the input routine address and the channel's code.

This table is almost identical to that in ROM 1 at \$15AF but with changes to the channel P routines to use the RS232 port instead of the ZX Printer. Used at \$01DD (ROM 0).

L05A8:	DEFW PRINT_OUT	\$09F4 - K channel output routine.
	DEFW KEY_INPUT	\$10A8 - K channel input routine.
	DEFB 'K'	\$4B - Channel identifier 'K'.
	DEFW PRINT_OUT	\$09F4 - S channel output routine.
	DEFW REPORT_J	\$15C4 - S channel input routine.
	DEFB 'S'	\$53 - Channel identifier 'S'.
	DEFW ADD_CHAR	\$0F81 - R channel output routine.
	DEFW REPORT_J	\$15C4 - R channel input routine.
	DEFB 'R'	\$52 - Channel identifier 'R'.
	DEFW POUT	\$5B34 - P Channel output routine.
	DEFW PIN	\$5B2F - P Channel input routine.
	DEFB 'P'	\$50 - Channel identifier 'P'.
	DEFB \$80	End marker.

### The 'Initial Stream Data'

Initially there are seven streams - \$FD to \$03.

This table is identical to that in ROM 1 at \$15C6.

Used at \$0226 (ROM 0).

L05BD:	DEFB \$01, \$00	Stream \$FD leads to channel 'K'.
	DEFB \$06, \$00	Stream \$FE leads to channel 'S'.
	DEFB \$0B, \$00	Stream \$FF leads to channel 'R'.
	DEFB \$01, \$00	Stream \$00 leads to channel 'K'.
	DEFB \$01, \$00	Stream \$01 leads to channel 'K'.
	DEFB \$06, \$00	Stream \$02 leads to channel 'S'.
	DEFB \$10, \$00	Stream \$03 leads to channel 'P'.

## ERROR HANDLER ROUTINES — PART 5

### Produce Error Report

L05CB:	POP HL	Point to the error byte.
	LD BC,\$7FFD	
	XOR A	ROM 0, Screen 0, Bank 0, 128 mode.
	DI	Ensure interrupts disable whilst paging.
	LD (BANK_M),A	\$5B5C. Store new state in BANK_M.
	OUT (C),A	Switch to ROM 0.
	EI	
	LD SP,(\$5C3D)	Restore SP from ERR_SP.
	LD A,(HL)	Fetch the error number.
	LD (RAMERR),A	\$5B5E. Store the error number.
	INC A	

## SPECTRUM +2 ROM 0 DISASSEMBLY

CP \$1E  
JR NC,L05E7

[**BUG** - This should be \$1D. As such, error code 'a' will be diverted to ROM 1 for handling. Credit: Paul Farrow]  
Jump if not a standard error code.

Handle a standard error code

RST 28H  
DEFW RAMRST

\$5B5D. Call the error handler routine in ROM 1.

Handle a new error code

L05E7: DEC A  
LD (IY+\$00),A  
LD HL,(\$5C5D)  
LD (\$5C5F),HL  
RST 28H  
DEFW SET\_STK  
RET

Store in ERR\_NR.  
CH\_ADD.  
X\_PTR. Set up the address of the character after the '?' marker.  
\$16C5. Set the calculator stack.  
Return to the error routine.

## Check for BREAK into Program

L05F5: LD A,\$7F  
IN A,(\$FE)  
RRA  
RET C  
LD A,\$FE  
IN A,(\$FE)  
RRA  
RET C  
CALL L05CB  
DEFB \$14

Read keyboard row B - SPACE.  
  
Extract the SPACE key.  
Return if SPACE not pressed.  
Read keyboard row CAPS SHIFT - V.  
  
Extract the CAPS SHIFT key.  
Return if CAPS SHIFT not pressed.  
Produce an error.  
"L Break into program"

## RS232 PRINTER ROUTINES

### RS232 Channel Handler Routines

This routine handles input and output RS232 requested. It is similar to the routine in the ZX Interface 1 ROM at \$0D5A, but in that ROM the routine is only used for input.

L0605: EI  
EX AF,AF'  
LD DE,POUT2  
PUSH DE  
RES 3,(IY+\$02)  
PUSH HL  
LD HL,(\$5C3D)  
LD E,(HL)  
INC HL  
LD D,(HL)  
AND A  
LD HL,ED\_ERROR  
SBC HL,DE  
JR NZ,L0656

Enabled interrupts.  
Save AF registers.  
\$5B4A. Address of the RS232 exit routine held in RAM.  
Stack it.  
TVFLAG. Indicate not automatic listing.  
Save the input/output routine address.  
Fetch location of error handler routine from ERR\_SP.  
  
DE=Address of error handler routine.  
  
\$107F in ROM 1.  
  
Jump if error handler address is different, i.e. due to INKEY\$# or PRINT#.

Handle INPUT#

POP HL  
LD SP,(\$5C3D)  
POP DE  
POP DE

Retrieve the input/output routine address.  
ERR\_SP.  
Discard the error handler routine address.  
Fetch the original address of ERR\_SP (this was stacked at the beginning of the INPUT routine in ROM 1).

## SPECTRUM +2 ROM 0 DISASSEMBLY

L0629:	LD (\$5C3D),DE PUSH HL LD DE,L062F PUSH DE JP (HL)	ERR_SP. Save the input/output routine address. Address to return to. Stack the address. Jump to the RS232 input/output routine.
--------	--	---

Return here from the input/output routine

L062F:	JR C,L063A	Jump if a character was received.
	JR Z,L0637	Jump if a character was not received.
L0633:	CALL L05CB DEFB \$07	Produce an error "8 End of file".

A character was not received

L0637:	POP HL JR L0629	Retrieve the input routine address. Jump back to await another character.
--------	--------------------	--

A character was received

L063A:	CP \$0D JR Z,L064C LD HL,(RETADDR) PUSH HL RST 28H DEFW ADD_CHAR+4 POP HL LD (RETADDR),HL POP HL JR L0629	Is it a carriage return? Jump ahead if so. \$5B5A. Fetch the return address.  \$0F85. Insert the character into the INPUT line.  \$5B5A. Restore the return address. Retrieve the input routine address. Jump back to await another character.
--------	--	--

Enter was received so end reading the stream

L064C:	POP HL LD A,(BANK_M) OR \$10 PUSH AF JP POUT2	Discard the input routine address. \$5B5C. Fetch current paging configuration. Select ROM 1. Stack the required paging configuration. \$5B4A. Exit.
--------	---	---

Handle INKEY\$# and PRINT#

L0656:	POP HL LD DE,L065C PUSH DE JP (HL)	Retrieve the input/output routine address.  Stack the return address. Jump to input or output routine.
--------	---	---

Return here from the input/output routine. When returning from the output routine, either the carry or zero flags should always be set to avoid the false generation of error report "8 End of file" [though this is not always the case - see bugs starting at \$088B (ROM 0)].

L065C:	RET C RET Z JR L0633	Return if a character was received. Return if a character was not received or was written. Produce error report "8 End of file".
--------	----------------------------	--

## FORMAT Routine

The format command sets the RS232 baud rate, e.g. FORMAT "P"; 9600.

It attempts to match against one of the supported baud rates, or uses the next higher baud rate if a non-standard value is requested. The maximum baud rate supported is 9600, and this is used for any rates specified that are higher than this.

L0660:	RST 28H DEFW GET_CHAR RST 28H DEFW EXPT_EXP	[Could just do RST \$18] \$0018. Get an expression. \$1C8C.
--------	--	--

## SPECTRUM +2 ROM o DISASSEMBLY

	BIT 7,(IY+\$01)	FLAGS.
	JR Z,L0680	Jump ahead if syntax checking.
	RST 28H	
	DEFW STK_FETCH	\$2BF1. Fetch the expression.
	LD A,C	
	DEC A	
	OR B	
	JR Z,L0678	Jump ahead if string is 1 character long.
	CALL L05CB	Produce error report.
	DEFB \$24	"i Invalid device".
L0678:	LD A,(DE)	Get character.
	AND \$DF	Convert to upper case.
	CP 'P'	\$50. Is it channel 'P'?
L0680:	JP NZ,L1931	Jump if not to produce error report "C Nonsense in BASIC".
	LD HL,(\$5C5D)	CH_ADD. Next character to be interpreted.
	LD A,(HL)	
	CP \$3B	Next character must be ';'. \$1C82.
	JP NZ,L1931	Jump if not to produce error report "C Nonsense in BASIC".
	RST 28H	Skip past the ';' character.
	DEFW NEXT_CHAR	\$0020. [Could just do RST \$20]
	RST 28H	Get a numeric expression from the line.
	DEFW EXPT_1NUM	\$1C82.
	BIT 7,(IY+\$01)	FLAGS. Checking syntax mode?
	JR Z,L069C	Jump ahead if so.
	RST 28H	Get the result as an integer.
	DEFW FIND_INT2	\$1E99.
L069C:	LD (HD_00),BC	\$5B71. Store the result temporarily for use later.
	RST 28H	[Could just do RST \$18]
	DEFW GET_CHAR	\$0018. Get the next character in the BASIC line.
	CP \$0D	It should be ENTER.
	JR Z,L06A8	Jump ahead if it is.
	CP ':'	\$3A. Or the character is allowed to be ':'.
L06A8:	JP NZ,L1931	Jump if not to produce error report "C Nonsense in BASIC".
	CALL L18C0	Check for end of line.
	LD BC,(HD_00)	\$5B71. Get the baud rate saved earlier.
	LD A,B	Is it zero?
	OR C	
	JR NZ,L06B7	Jump if not, i.e. a numeric value was specified.
	CALL L05CB	Produce error report.
	DEFB \$25	"j invalid baud rate"

Lookup the timing constant to use for the specified baud rate

L06B7:	LD HL,L06D7	Table of supported baud rates.
L06BA:	LD E,(HL)	
	INC HL	
	LD D,(HL)	
	INC HL	
	EX DE,HL	HL=Supported baud rate value.
	LD A,H	
	CP \$25	Reached the last baud rate value in the table?
	JR NC,L06CE	Jump is so to use a default baud rate of 9600.
	AND A	
	SBC HL,BC	Table entry matches or is higher than requested baud rate?
	JR NC,L06CE	Jump ahead if so to use this baud rate.
	EX DE,HL	
	INC HL	Skip past the timing constant value
	INC HL	for this baud rate entry.
	JR L06BA	

The baud rate has been matched

L06CE:	EX DE,HL	HL points to timing value for the baud rate.
	LD E,(HL)	
	INC HL	
	LD D,(HL)	DE=Timing value for the baud rate.

LD (BAUD),DE  
RET

\$5B71. Store new value in system variable BAUD.

## Baud Rate Table

Consists of entries of baud rate value followed by timing constant to use in the RS232 routines.

L06D7:	DEFW \$0032, \$0AA5	Baud=50.
	DEFW \$006E, \$04D4	Baud=110.
	DEFW \$012C, \$01C3	Baud=300.
	DEFW \$0258, \$00E0	Baud=600.
	DEFW \$04B0, \$006E	Baud=1200.
	DEFW \$0960, \$0036	Baud=2400.
	DEFW \$12C0, \$0019	Baud=4800.
	DEFW \$2580, \$000B	Baud=9600.

## RS232 Input Routine

Exit: Carry flag set if a byte was read with the byte in A. Carry flag reset upon error.

L06F7:	LD HL,SERFL	\$5B61. SERFL holds second char that can be received
	LD A,(HL)	Is the second-character received flag set?
	AND A	i.e. have we already received data?
	JR Z,L0704	Jump ahead if not.
	LD (HL),\$00	Otherwise clear the flag
	INC HL	
	LD A,(HL)	and return the data which we received earlier.
	SCF	Set carry flag to indicate success
	RET	

## Read Byte from RS232 Port

The timing of the routine is achieved using the timing constant held in system variable BAUD.

Exit: Carry flag set if a byte was read, or reset upon error.

A=Byte read in.

L0704:	CALL L05F5	Check the BREAK key, and produce error message if it is being pressed.
	DI	Ensure interrupts are disabled to achieve accurate timing.
	EXX	
	LD DE,(BAUD)	\$5B71. Fetch the baud rate timing constant.
	LD HL,(BAUD)	\$5B71.
	SRL H	
	RR L	HL=BAUD/2. So that will sync to half way point in each bit.
	OR A	[Redundant byte]
	LD B,\$FA	Waiting time for start bit.
	EXX	Save B.
	LD C,\$FD	
	LD D,\$FF	
	LD E,\$BF	
	LD B,D	
	LD A,\$0E	
	OUT (C),A	Selects register 14, port I/O of AY-3-8912.
	IN A,(C)	Read the current state of the I/O lines.
	OR \$F0	%11110000. Default all input lines to 1.
	AND \$FB	%11111011. Force CTS line to 0.
	LD B,E	B=\$BF.
	OUT (C),A	Make CTS (Clear To Send) low to indicate ready to receive.
	LD H,A	Store status of other I/O lines.

Look for the start bit

L072D: LD B,D

## SPECTRUM +2 ROM 0 DISASSEMBLY

	IN A,(C)	Read the input line.
	AND \$80	%10000000. Test TXD (input) line.
L0734:	JR Z,L073D	Jump if START BIT found.
	EXX	Fetch timeout counter
	DEC B	and decrement it.
	EXX	Store it.
	JR NZ,L072D	Continue to wait for start bit if not timed out.
	XOR A	Reset carry flag to indicate no byte read.
	PUSH AF	Save the failure flag.
L073D:	JR L0776	Timed out waiting for START BIT.
	IN A,(C)	Second test of START BIT - it should still be 0.
	AND \$80	Test TXD (input) line.
	JR NZ,L0734	Jump back if it is no longer 0.
	IN A,(C)	Third test of START BIT - it should still be 0.
	AND \$80	Test TXD (input) line.
	JR NZ,L0734	Jump back if it is no longer 0.

A start bit has been found, so the 8 data bits are now read in.

As each bit is read in, it is shifted into the msb of A. Bit 7 of A is preloaded with a 1 to represent the start bit and when this is shifted into the carry flag it signifies that 8 data bits have been read in.

	EXX	
	LD BC,\$FFFD	
	LD A,\$80	
L0750:	EX AF,AF'	Preload A with the START BIT. It forms a shift counter used to count
	ADD HL,DE	the number of bits to read in.
	NOP	HL=1.5*(BAUD).
	NOP	(4) Fine tune the following delay.
	NOP	
	NOP	

### BD-DELAY

L0755:	DEC HL	(6) Delay for 26*BAUD.
	LD A,H	(4)
	OR L	(4)
	JR NZ,L0755	(12) Jump back to until delay completed.
	IN A,(C)	Read a bit.
	AND \$80	Test TXD (input) line.
	JP Z,L076A	Jump if a 0 received.

Received one 1

	EX AF,AF'	Fetch the bit counter.
	SCF	Set carry flag to indicate received a 1.
	RRA	Shift received bit into the byte (C->76543210->C).
	JR C,L0773	Jump if START BIT has been shifted out indicating all data bits have been received.
	EX AF,AF'	Save the bit counter.
	JP L0750	Jump back to read the next bit.

Received one 0

L076A:	EX AF,AF'	Fetch the bit counter.
	OR A	Clear carry flag to indicate received a 0.
	RRA	Shift received bit into the byte (C->76543210->C).
	JR C,L0773	Jump if START BIT has been shifted out indicating all data bits have been received.
	EX AF,AF'	Save the bit counter.
	JP L0750	Jump back to read next bit.

After looping 8 times to read the 8 data bits, the start bit in the bit counter will be shifted out and hence A will contain a received byte.

L0773:	SCF	Signal success.
	PUSH AF	Push success flag.
	EXX	

## SPECTRUM +2 ROM o DISASSEMBLY

The success and failure paths converge here

L0776:	LD A,H OR \$04 LD B,E OUT (C),A EXX LD H,D LD L,E LD BC,\$0007 OR A SBC HL,BC	A=%1111x1xx. Force CTS line to 1. B=\$BF. Make CTS (Clear To Send) high to indicate not ready to receive.  HL=(BAUD).  HL=(BAUD)-7.
L0785:	DEC HL LD A,H OR L JR NZ,L0785 LD BC,\$FFFD ADD HL,DE ADD HL,DE ADD HL,DE	Delay for the stop bit.  Jump back until delay completed. HL will be \$0000. DE=(BAUD).  HL=3*(BAUD). This is how long to wait for the next start bit.

The device at the other end of the cable may send a second byte even though CTS is low. So repeat the procedure to read another byte.

L0790:	IN A,(C) AND \$80 JR Z,L079E DEC HL LD A,H OR L JR NZ,L0790	Read the input line. %10000000. Test TXD (input) line. Jump if START BIT found. Decrement timeout counter.  Jump back looping for a start bit until a timeout occurs.
--------	---	--

No second byte incoming so return status of the first byte read attempt

	POP AF EI RET	Return status of first byte read attempt - carry flag reset for no byte received or carry flag set and A holds the received byte.
L079E:	IN A,(C) AND \$80 JR NZ,L0790 IN A,(C) AND \$80 JR NZ,L0790	Second test of START BIT - it should still be 0. Test TXD (input) line. Jump back if it is no longer 0. Third test of START BIT - it should still be 0. Test TXD (input) line. Jump back if it is no longer 0.

A second byte is on its way and is received exactly as before

	LD H,D LD L,E LD BC,\$0002 SRL H RR L OR A SBC HL,BC LD BC,\$FFFD LD A,\$80 EX AF,AF'	HL=(BAUD).  HL=(BAUD)/2.  HL=(BAUD)/2 - 2.  Preload A with the START BIT. It forms a shift counter used to count the number of bits to read in.
L07BC:	NOP NOP NOP NOP	Fine tune the following delay.
L07C1:	ADD HL,DE DEC HL LD A,H OR L JR NZ,L07C1 IN A,(C)	HL=1.5*(BAUD). Delay for 26*(BAUD).  Jump back to until delay completed. Read a bit.

## SPECTRUM +2 ROM 0 DISASSEMBLY

AND \$80  
JP Z,L07D6

Test TXD (input) line.  
Jump if a 0 received.

Received one 1

EX AF,AF'  
SCF  
RRA  
JR C,L07DF  
EX AF,AF'  
JP L07BC

Fetch the bit counter.  
Set carry flag to indicate received a 1.  
Shift received bit into the byte (C->76543210->C).  
Jump if START BIT has been shifted out indicating all data bits have been received.  
Save the bit counter.  
Jump back to read the next bit.

Received one 0

L07D6: EX AF,AF'  
OR A  
RRA  
JR C,L07DF  
EX AF,AF'  
JP L07BC

Fetch the bit counter.  
Clear carry flag to indicate received a 0.  
Shift received bit into the byte (C->76543210->C).  
Jump if START BIT has been shifted out indicating all data bits have been received.  
Save the bit counter.  
Jump back to read next bit.

Exit with the byte that was read in

L07DF: LD HL,SERFL  
LD (HL),\$01  
INC HL  
LD (HL),A  
POP AF  
EI  
RET

\$5B61.  
Set the flag indicating a second byte is in the buffer.  
  
Store the second byte read in the buffer.  
Return the first byte.  
Re-enable interrupts.

## RS232 Output Routine

This routine handles control codes, token expansion, graphics and UDGs. It therefore cannot send binary data and hence cannot support EPSON format ESC control codes [Credit: Andrew Owen].

The routine suffers from a number of bugs as described in the comments below. It also suffers from a minor flaw in the design, which prevents interlacing screen and printer control codes and their parameters. For example, the following will not work correctly: 10 LPRINT CHR\$ 16

20 PRINT AT 0,0

30 LPRINT CHR\$ 0;"ABC"

The control byte 16 gets stored in TVDATA so that the system knows how to interpret its parameter byte. However, the AT control code 22 in line 20 will overwrite it. When line 30 is executed, TVDATA still holds the control code for 'AT' and so this line is interpreted as PRINT AT instead of PRINT INK. [Credit: Ian Collier (+3)]

Entry: A=character to output.

Exit : Carry flag reset indicates success.

L07E9: PUSH AF  
LD A,(TVPARS)  
OR A  
JR Z,L07FF  
DEC A  
LD (TVPARS),A  
JR NZ,L07FA

Save the character to print.  
\$5B65. Number of parameters expected.  
  
Jump if no parameters.  
Ignore the parameter.  
\$5B65.  
Jump ahead if we have not processed all parameters.

All parameters processed

L07FA: POP AF  
JP L0891  
POP AF  
LD (\$5C0F),A  
RET

Retrieve character to print.  
Jump ahead to continue.  
Retrieve character to print.  
TVDATA+1. Store it for use later.

L07FF: POP AF  
CP \$A3  
JR C,L0811

Retrieve character to print.  
Test against code for 'SPECTRUM'.  
Jump ahead if not a token.

## SPECTRUM +2 ROM o DISASSEMBLY

### Process tokens

	LD HL,(RETADDR)	\$5B5A. Save RETADDR temporarily.
	PUSH HL	
	RST 28H	
	DEFW PO_T_UDG	\$0B52. Print tokens via call to ROM 1 routine PO-T&UDG.
	POP HL	
	LD (RETADDR),HL	\$5B5A. Restore the original contents of RETADDR.
	SCF	
	RET	
L0811:	LD HL,\$5C3B	FLAGS.
	RES 0,(HL)	Suppress printing a leading space.
	CP ''	\$20. Is character to output a space?
	JR NZ,L081C	Jump ahead if not a space.
	SET 0,(HL)	Signal leading space required.
L081C:	CP \$7F	Compare against copyright symbol.
	JR C,L0822	Jump ahead if not a graphic or UDG character.
	LD A,'?	\$3F. Print a '?' for all graphic and UDG characters.
L0822:	CP \$20	Is it a control character?
	JR C,L083D	Jump ahead if so.

### Printable character

L0826:	PUSH AF	Save the character to print.
	LD HL,COL	\$5B63. Point to the column number.
	INC (HL)	Increment the column number.
	LD A,(WIDTH)	\$5B64. Fetch the number of columns.
	CP (HL)	
	JR NC,L0839	Jump if end of row not reached.
	CALL L0841	Print a carriage return and line feed.
	LD A,\$01	
	LD (COL),A	\$5B63. Set the print position to column 1.
L0839:	POP AF	Retrieve character to print.
	JP L08C2	Jump ahead to print the character.

### Process control codes

L083D:	CP \$0D	Is it a carriage return?
	JR NZ,L084F	Jump ahead if not.

### Handle a carriage return

L0841:	XOR A	
	LD (COL),A	\$5B63. Set the print position back to column 0.
	LD A,\$0D	
	CALL L08C2	Print a carriage return.
	LD A,\$0A	
	JP L08C2	Print a line feed.
L084F:	CP \$06	Is it a comma?
	JR NZ,L0872	Jump ahead if not.

### Handle a comma

	LD BC,(COL)	\$5B63. Fetch the column position.
	LD E,\$00	Will count number of columns to move across to reach next comma position.
L0859:	INC E	Increment column counter.
	INC C	Increment column position.
	LD A,C	
	CP B	End of row reached?
	JR Z,L0867	Jump if so.
L085F:	SUB \$08	
	JR Z,L0867	Jump if column 8, 16 or 32 reached.
	JR NC,L085F	Column position greater so subtract another 8.
	JR L0859	Jump back and increment column position again.

## SPECTRUM +2 ROM o DISASSEMBLY

Column 8, 16 or 32 reached. Output multiple spaces until the desired column position is reached.

L0867:	PUSH DE LD A,\$20 CALL L07E9 POP DE DEC E RET Z JR L0867	Save column counter in E.  Output a space via a recursive call. Retrieve column counter to E. More spaces to output? Return if no more to output. Repeat for the next space to output.
L0872:	CP \$16 JR Z,L087F CP \$17 JR Z,L087F CP \$10 RET C JR L0888	Is it AT? Jump ahead to handle AT. Is it TAB? Jump ahead to handle TAB. Check for INK, PAPER, FLASH, BRIGHT, INVERSE, OVER. Ignore if not one of these. Jump ahead to handle INK, PAPER, FLASH, BRIGHT, INVERSE, OVER.

Handle AT and TAB

L087F:	LD (\$5C0E),A LD A,\$02 LD (TVPARS),A RET	TV_DATA. Store the control code for use later, \$16 (AT) or \$17 (TAB). Two parameters expected (even for TAB). \$5B65. Return with zero flag set.
--------	--	---

Handle INK, PAPER, FLASH, BRIGHT, INVERSE, OVER

L0888:	LD (\$5C0E),A LD A,\$02   LD (TVPARS),A RET	TV_DATA. Store the control code for use later. Two parameters expected. <b>[BUG</b> - Should be 1 parameter. 'LPRINT INK 4' will produce error report 'C Nonsense in BASIC'. Credit: Toni Baker, ZX Computing Monthly]. \$5B65. <b>[BUG</b> - Should return with the carry flag reset and the zero flag set. It causes a statement such as 'LPRINT INK 1;' to produce error report '8 End of file'. It is due to the main RS232 processing loop using the state of the flags to determine the success/failure response of the RS232 output routine. Credit: Ian Collier (+3), Andrew Owen (128)] [The bug can be fixed by inserting a XOR A instruction before the RET instruction. Credit: Paul Farrow]
--------	--	---

All parameters processed

L0891:	LD D,A LD A,(\$5C0E) CP \$16 JR Z,L08A1 CP \$17 CCF   RET NZ	D=Character to print. TV_DATA. Fetch the control code. Is it AT? Jump ahead to handle AT parameter. Is it TAB? <b>[BUG</b> - Should return with the carry flag reset and the zero flag set. It causes a statement such as 'LPRINT INK 1;' to produce error report '8 End of file'. It is due to the main RS232 processing loop using the state of the flags to determine the success/failure response of the RS232 output routine. Credit: Toni Baker, ZX Computing Monthly] Ignore if not TAB.
--------	--	---

[The bug can be fixed by replacing the instructions CCF and RET NZ with the following. Credit: Paul Farrow.

```

        JR Z,NOT_TAB
        XOR A
        RET
NOT_TAB

```

Handle TAB parameter

LD A,(\$5C0F) LD D,A	TV_DATA+1. Fetch the saved parameter. Fetch parameter to D.
-------------------------	--

## Process AT and TAB

L08A1:	LD A,(WIDTH)	\$5B64.
	CP D	Reached end of row?
	JR Z,L08A9	Jump ahead if so.
	JR NC,L08AF	Jump ahead if before end of row.

Column position equal or greater than length of row requested

L08A9:	LD B,A	(WIDTH).
	LD A,D	TAB/AT column position.
	SUB B	TAB/AT position - WIDTH.
	LD D,A	The new required column position.
	JR L08A1	Handle the new TAB/AT position.
L08AF:	LD A,D	Fetch the desired column number.
	OR A	
	JP Z,L0841	Jump to output a carriage return if column 0 required.
L08B4:	LD A,(COL)	\$5B63. Fetch the current column position.
	CP D	Compare against desired column position.
	RET Z	Done if reached requested column.
	PUSH DE	Save the number of spaces to output.
	LD A,\$20	
	CALL L07E9	Output a space via a recursive call.
	POP DE	Retrieve number of spaces to output.
	JR L08B4	Keep outputting spaces until desired column reached.

**Write Byte to RS232 Port**

The timing of the routine is achieved using the timing constant held in system variable BAUD.

Entry: A holds character to send.

Exit: Carry and zero flags reset.

L08C2:	PUSH AF	Save the byte to send.
	LD C,\$FD	
	LD D,\$FF	
	LD E,\$BF	
	LD B,D	
	LD A,\$0E	
	OUT (C),A	Select AY register 14 to control the RS232 port.
L08CE:	CALL L05F5	Check the BREAK key, and produce error message if it is being pressed.
	IN A,(C)	Read status of data register.
	AND \$40	%01000000. Test the DTR line.
	JR NZ,L08CE	Jump back until device is ready for data.
	LD HL,(BAUD)	\$5B5F. HL=Baud rate timing constant.
	LD DE,\$0002	
	OR A	
	SBC HL,DE	
	EX DE,HL	DE=(BAUD)-2.
	POP AF	Retrieve the byte to send.
	CPL	Invert the bits of the byte (RS232 logic is inverted).
	SCF	Carry is used to send START BIT.
	LD B,\$0B	B=Number of bits to send (1 start + 8 data + 2 stop).
	DI	Disable interrupts to ensure accurate timing.

## Transmit each bit

L08E7:	PUSH BC	Save the number of bits to send.
	PUSH AF	Save the data bits.
	LD A,\$FE	
	LD H,D	
	LD L,E	HL=(BAUD)-2.
	LD BC,\$BFFD	AY-3-8912 data register.
	JP NC,L08F9	Branch to transmit a 1 or a 0 (initially sending a 0 for the start bit).

## SPECTRUM +2 ROM o DISASSEMBLY

Transmit a 0

AND \$F7	Clear the RXD (out) line.
OUT (C),A	Send out a 0 (high level).
JR L08FF	Jump ahead to continue with next bit.

Transmit a 1

L08F9:	OR \$08	Set the RXD (out) line.
	OUT (C),A	Send out a 1 (low level).
	JR L08FF	Jump ahead to continue with next bit.

Delay the length of a bit

L08FF:	DEC HL	(6) Delay 26*BAUD cycles.
	LD A,H	(4)
	OR L	(4)
	JR NZ,L08FF	(12) Jump back until delay is completed.
	NOP	(4) Fine tune the timing.
	NOP	(4)
	NOP	(4)
	POP AF	Retrieve the data bits to send.
	POP BC	Retrieve the number of bits left to send.
	OR A	Clear carry flag.
	RRA	Shift the next bit to send into the carry flag.
	DJNZ L08E7	Jump back to send next bit until all bits sent.
	EI	Re-enable interrupts.
	RET	Return with carry and zero flags reset.

## COPY Command Routine

This routine copies 22 rows of the screen, outputting them to the printer a half row at a time. It is designed for EPSON compatible printers supporting double density bit graphics and 7/72 inch line spacing.

Only the pixel information is processed; the attributes are ignored.

L090F:	LD HL,HD_0B	Half row counter.
	LD (HL),\$2B	Set the half row counter to 43 half rows (will output 44 half rows in total).
L0914:	LD HL,L0998	Point to printer configuration data (7/72 inch line spacing, double density bit graphics).
	CALL L097E	Send the configuration data to printer.
	CALL L0934	Output a half row, at double height.
	LD HL,L099F	Table holds a line feed only.
	CALL L097E	Send a line feed to printer.
	LD HL,HD_0B	\$5B72. The half row counter is tested to see if it is zero
	XOR A	and if so then the line spacing is reset to its
	CP (HL)	original value.
	JR Z,L092D	Jump if done, resetting printer line spacing.
	DEC (HL)	Decrement half row counter.
	JR L0914	Repeat for the next half row.

Copy done so reset printer line spacing before exiting

L092D:	LD HL,L09A1	Point to printer configuration data (1/6 inch line spacing).
	CALL L097E	Send the configuration data to printer.
	RET	[Could have saved 1 byte by using JP \$097E (ROM 0)]

## Output Half Row

L0934:	LD HL,HD_00	\$5B71. Pixel column counter.
	LD (HL),\$FF	Set pixel column counter to 255 pixels.
L0939:	CALL L0945	Output a column of pixels, at double height.
	LD HL,HD_00	\$5B71. Pixel column counter.

## SPECTRUM +2 ROM o DISASSEMBLY

XOR A	
CP (HL)	Check if all pixels in this row have been output.
RET Z	Return if so.
DEC (HL)	Decrement pixel column counter.
JR L0939	Repeat for all pixels in this row.

Output a column of pixels (at double height)

L0945:	LD DE,\$C000	D=%11000000. Used to hold the double height pixel.
	LD BC,(HD_00)	\$5B71. C=Pixel column counter, B=Half row counter.
	SCF	
	RL B	B=2xB+1
	SCF	
	RL B	B=4xB+3. The pixel row coordinate.
	LD A,C	Pixel column counter.
	CPL	
	LD C,A	C=255-C. The pixel column coordinate.
	XOR A	Clear A. Used to generate double height nibble of pixels to output.
	PUSH AF	
	PUSH DE	
	PUSH BC	Save registers.
L0959:	CALL L098C	Test whether pixel (B,C) is set
	POP BC	
	POP DE	Restore registers.
	LD E,\$00	Set double height pixel = 0.
	JR Z,L0963	Jump if pixel is reset.
	LD E,D	The double height pixel to output (%11000000, %00110000, %00001100 or %00000011).
L0963:	POP AF	
	OR E	Add the double height pixel value to the byte to output.
	PUSH AF	
	DEC B	Decrement half row coordinate.
	SRL D	
	SRL D	Create next double height pixel value (%00110000, %00001100 or %00000011).
	PUSH DE	
	PUSH BC	
	JR NC,L0959	Repeat for all four pixels in the half row.
	POP BC	
	POP DE	Unload the stack.
	POP AF	
	LD B,\$03	Send double height nibble of pixels output 3 times.

## Output Nibble of Pixels

Send each nibble of pixels (i.e. column of 4 pixels) output 3 times so that the width of a pixel is the same size as its height.

L0974:	PUSH BC	
	PUSH AF	
	CALL L08C2	Send byte to RS232 port.
	POP AF	
	POP BC	
	DJNZ L0974	
	RET	

## Output Characters from Table

This routine is used to send a sequence of EPSON printer control codes out to the RS232 port. It sends (HL) characters starting from HL+1.

L097E:	LD B,(HL)	Get number of bytes to send.
	INC HL	Point to the data to send.
L0980:	LD A,(HL)	Retrieve value.
	PUSH HL	
	PUSH BC	

CALL L08C2	Send byte to RS232 port.
POP BC	
POP HL	
INC HL	Point to next data byte to send.
DJNZ L0980	Repeat for all bytes.
RET	

## Test Whether Pixel (B,C) is Set

L098C:	RST 28H	Get address of (B,C) pixel into HL and pixel position within byte into A.
	DEFW PIXEL_ADDR	\$22AA.
	LD B,A	B=Pixel position within byte (0-7).
	INC B	
	XOR A	Pixel mask.
	SCF	Carry flag holds bit to be rotated into the mask.
L0993:	RRA	Shift the mask bit into the required bit position.
	DJNZ L0993	
	AND (HL)	Isolate this pixel from A.
	RET	

## EPSON Printer Control Code Tables

L0998:	DEFB \$06	6 characters follow.
	DEFB \$1B, \$31	ESC '1' - 7/72 inch line spacing.
	DEFB \$1B, \$4C, \$00, \$03	ESC 'L' 0 3 - Double density (768 bytes per row).
L099F:	DEFB \$01	1 character follows.
	DEFB \$0A	Line feed.
L09A1:	DEFB \$02	2 characters follow.
	DEFB \$1B, \$32	ESC '2' - 1/6 inch line spacing.

## PLAY COMMAND ROUTINES

Up to 3 channels of music/noise are supported by the AY-3-8912 sound generator.

Up to 8 channels of music can be sent to support synthesisers, drum machines or sequencers via the MIDI interface, with the first 3 channels also played by the AY-3-8912 sound generator. For each channel of music, a MIDI channel can be assigned to it using the 'Y' command.

The PLAY command reserves and initialises space for the PLAY command. This comprises a block of \$003C bytes used to manage the PLAY command (IY points to this command data block) and a block of \$0037 bytes for each channel string (IX is used to point to the channel data block for the current channel). [Note that the command data block is \$04 bytes larger than it needs to be, and each channel data block is \$11 bytes larger than it needs to be]

Entry: B=The number of strings in the PLAY command (1..8).

## Command Data Block Format

IY+\$00 / IY+\$01	Channel 0 data block pointer. Points to the data for channel 0 (string 1).
IY+\$02 / IY+\$03	Channel 1 data block pointer. Points to the data for channel 1 (string 2).
IY+\$04 / IY+\$05	Channel 2 data block pointer. Points to the data for channel 2 (string 3).
IY+\$06 / IY+\$07	Channel 3 data block pointer. Points to the data for channel 3 (string 4).
IY+\$08 / IY+\$09	Channel 4 data block pointer. Points to the data for channel 4 (string 5).
IY+\$0A / IY+\$0B	Channel 5 data block pointer. Points to the data for channel 5 (string 6).
IY+\$0C / IY+\$0D	Channel 6 data block pointer. Points to the data for channel 6 (string 7).
IY+\$0E / IY+\$0F	Channel 7 data block pointer. Points to the data for channel 7 (string 8).
IY+\$10	Channel bitmap. Initialised to \$FF and a 0 rotated in to the left for each string parameters of the PLAY command, thereby indicating the channels in use.
IY+\$11 / IY+\$12	Channel data block duration pointer. Points to duration length store in channel 0 data block (string 1).
IY+\$13 / IY+\$14	Channel data block duration pointer. Points to duration length store in channel 1 data block (string 2).
IY+\$15 / IY+\$16	Channel data block duration pointer. Points to duration length store in channel 2 data block (string 3).
IY+\$17 / IY+\$18	Channel data block duration pointer. Points to duration length store in channel 3 data block (string 4).
IY+\$19 / IY+\$1A	Channel data block duration pointer. Points to duration length store in channel 4 data block (string 5).
IY+\$1B / IY+\$1C	Channel data block duration pointer. Points to duration length store in channel 5 data block (string 6).
IY+\$1D / IY+\$1E	Channel data block duration pointer. Points to duration length store in channel 6 data block (string 7).

## SPECTRUM +2 ROM 0 DISASSEMBLY

IY+\$1F / IY+\$20	Channel data block duration pointer. Points to duration length store in channel 7 data block (string 8).
IY+\$21	Channel selector. It is used as a shift register with bit 0 initially set and then shift to the left until a carry occurs, thereby indicating all 8 possible channels have been processed.
IY+\$22	Temporary channel bitmap, used to hold a working copy of the channel bitmap at IY+\$10.
IY+\$23 / IY+\$24	Address of the channel data block pointers, or address of the channel data block duration pointers (allows the routine at \$0A8D (ROM 0) to be used with both set of pointers).
IY+\$25 / IY+\$26	Stores the smallest duration length of all currently playing channel notes.
IY+\$27 / IY+\$28	The current tempo timing value (derived from the tempo parameter 60..240 beats per second).
IY+\$29	The current effect waveform value.
IY+\$2A	Temporary string counter selector.
IY+\$2B..IY+\$37	Holds a floating point calculator routine.
IY+\$38..IY+\$3B	Not used.

## Channel Data Block Format

IX+\$00	The note number being played on this channel (equivalent to index offset into the note table).
IX+\$01	MIDI channel assigned to this string (range 0 to 15).
IX+\$02	Channel number (range 0 to 7), i.e. index position of the string within the PLAY command.
IX+\$03	12*Octave number (0, 12, 24, 36, 48, 60, 72, 84 or 96).
IX+\$04	Current volume (range 0 to 15, or if bit 4 set then using envelope).
IX+\$05	Last note duration value as specified in the string (range 1 to 9).
IX+\$06 / IX+\$07	Address of current position in the string.
IX+\$08 / IX+\$09	Address of byte after the end of the string.
IX+\$0A	Flags: Bit 0 : 1=Single closing bracket found (repeat string indefinitely). Bits 1-7: Not used (always 0).
IX+\$0B	Open bracket nesting level (range \$00 to \$04).
IX+\$0C / IX+\$0D	Return address for opening bracket nesting level 0 (points to character after the bracket).
IX+\$0E / IX+\$0F	Return address for opening bracket nesting level 1 (points to character after the bracket).
IX+\$10 / IX+\$11	Return address for opening bracket nesting level 2 (points to character after the bracket).
IX+\$12 / IX+\$13	Return address for opening bracket nesting level 3 (points to character after the bracket).
IX+\$14 / IX+\$15	Return address for opening bracket nesting level 4 (points to character after the bracket).
IX+\$16	Closing bracket nesting level (range \$FF to \$04).
IX+\$17...IX+\$18	Return address for closing bracket nesting level 0 (points to character after the bracket).
IX+\$19...IX+\$1A	Return address for closing bracket nesting level 1 (points to character after the bracket).
IX+\$1B...IX+\$1C	Return address for closing bracket nesting level 2 (points to character after the bracket).
IX+\$1D...IX+\$1E	Return address for closing bracket nesting level 3 (points to character after the bracket).
IX+\$1F...IX+\$20	Return address for closing bracket nesting level 4 (points to character after the bracket).
IX+\$21	Tied notes counter (for a single note the value is 1).
IX+\$22 / IX+\$23	Duration length, specified in 96ths of a note.
IX+\$24...IX+\$25	Subsequent note duration length (used only with triplets), specified in 96ths of a note.
IX+\$26...IX+\$36	Not used.

L09A4:	DI	Disable interrupts to ensure accurate timing.
--------	----	---

## Create a workspace for the play channel command strings

	PUSH BC	B=Number of channel string (range 1 to 8). Also used as string index number in the following loop.
	LD DE,\$0037	
	LD HL,\$003C	
L09AC:	ADD HL,DE	Calculate HL=\$003C + (\$0037 * B).
	DJNZ L09AC	
	LD C,L	
	LD B,H	BC=Space required (maximum = \$01F4).
	RST 28H	
	DEFW BC_SPACES	\$0030. Make BC bytes of space in the workspace.
	DI	Interrupts get re-enabled by the call mechanism to ROM 1 so disable them again.
	PUSH DE	
	POP IY	IY=Points at first new byte - the command data block.
	PUSH HL	
	POP IX	IX=Points at last new byte - byte after all channel information blocks.

## SPECTRUM +2 ROM 0 DISASSEMBLY

LD (IY+\$10),\$FF

Initial channel bitmap with value meaning 'zero strings'

Loop over each string to be played

L09BF: LD BC,\$FFC9  
 ADD IX,BC  
 LD (IX+\$03),\$3C  
 LD (IX+\$01),\$FF  
 LD (IX+\$04),\$0F  
 LD (IX+\$05),\$05  
 LD (IX+\$21),\$00  
 LD (IX+\$0A),\$00  
 LD (IX+\$0B),\$00  
 LD (IX+\$16),\$FF  
 LD (IX+\$17),\$00  
 LD (IX+\$18),\$00

\$-37 (\$37 bytes is the size of a play channel string information block).  
 IX points to start of space for the last channel.  
 Default octave is 5.  
 No MIDI channel assigned.  
 Default volume is 15.  
 Default note duration.  
 Count of the number of tied notes.  
 Signal not to repeat the string indefinitely.  
 No opening bracket nesting level.  
 No closing bracket nesting level.  
 Return address for closing bracket nesting level 0.  
 [No need to initialise this since it is written to before it is ever tested]

**[BUG** - At this point interrupts are disabled and IY is now being used as a pointer to the master PLAY information block. Unfortunately, interrupts are enabled during the STK\_FETCH call and IY is left containing the wrong value. This means that if an interrupt were to occur during execution of the subroutine then there would be a one in 65536 chance that (IY+\$40) will be corrupted - this corresponds to the volume setting for music channel A. Rewriting the SWAP routine to only re-enable interrupts if they were originally enabled would cure this bug (see end of file for description of her suggested fix). Credit: Toni Baker, ZX Computing Monthly] [An alternative and simpler solution to the fix Toni Baker describes would be to stack IY, set IY to point to the system variables at \$5C3A, call STK\_FETCH, disable interrupts, then pop the stacked value back to IY. Credit: Paul Farrow]

RST 28H  
 DEFW STK\_FETCH  
 DI  
 LD (IX+\$06),E  
 LD (IX+\$07),D  
 LD (IX+\$0C),E  
 LD (IX+\$0D),D  
 EX DE,HL  
 ADD HL,BC  
 LD (IX+\$08),L  
 LD (IX+\$09),H  
 POP BC  
 PUSH BC  
 DEC B  
 LD C,B  
 LD B,\$00  
 SLA C  
 PUSH IY  
 POP HL  
 ADD HL,BC  
 PUSH IX  
 POP BC  
 LD (HL),C  
 INC HL  
 LD (HL),B  
 OR A  
 RL (IY+\$10)

Get the details of the string from the stack.  
 \$2BF1.  
 Interrupts get re-enabled by the call mechanism to ROM 1 so disable them again.  
 Store the current position within in the string, i.e. the beginning of it.

Store the return position within the string for a closing bracket, which is initially the start of the string in case a single closing bracket is found.  
 HL=Points to start of string. BC=Length of string.  
 HL=Points to address of byte after the string.  
 Store the address of the character just after the string.  
 B=String index number (range 1 to 8).  
 Save it on the stack again.  
 Reduce the index so it ranges from 0 to 7.

BC=String index\*2.

HL=Address of the command data block.  
 Skip 8 channel data pointer words.

BC=Address of current channel information block.  
 Store the pointer to the channel information block.

Clear the carry flag.  
 Rotate one zero-bit into the least significant bit of the channel bitmap. This initially holds \$FF but once this loop is over, this byte has a zero bit for each string parameter of the PLAY command.  
 B=Current string index.  
 Decrement string index so it ranges from 0 to 7.  
 Save it for future use on the next iteration.  
 Store the channel number.  
 Jump back while more channel strings to process.  
 Drop item left on the stack.

Entry point here from the vector table at \$011B

L0A24: LD (IY+\$27),\$1A  
 LD (IY+\$28),\$0B  
 PUSH IY  
 POP HL

Set the initial tempo timing value.  
 Corresponds to a 'T' command value of 120, and gives two crotchets per second.  
 HL=Points to the command data block.

LD BC,\$002B	
ADD HL,BC	
EX DE,HL	DE=Address to store RAM routine.
LD HL,LOA50	HL=Address of the RAM routine bytes.
LD BC,\$000D	
LDIR	Copy the calculator routine to RAM.
LD D,\$07	Register 7 - Mixer.
LD E,\$F8	I/O ports are inputs, noise output off, tone output on.
CALL LOE9B	Write to sound generator register.
LD D,\$0B	Register 11 - Envelope Period (Fine).
LD E,\$FF	Set period to maximum.
CALL LOE9B	Write to sound generator register.
INC D	Register 12 - Envelope Period (Coarse).
CALL LOE9B	Write to sound generator register.
JR LOA9C	Jump ahead to continue. [Could have saved these 2 bytes by having the code at \$0A9C (ROM 0) immediately follow]

## Calculate Timing Loop Counter « RAM Routine »

This routine is copied into the command data block (offset \$2B..\$37) by the routine at \$0A24 (ROM 0).

It uses the floating point calculator found in ROM 1, which is usually invoked via a RST \$28 instruction. Since ROM 0 uses RST \$28 to call a routine in ROM 1, it is unable to invoke the floating point calculator this way. It therefore copies the following routine to RAM and calls it with ROM 1 paged in.

The routine calculates  $(10/x)/7.33e-6$ , where x is the tempo 'T' parameter value multiplied by 4. The result is used as an inner loop counter in the wait routine at \$0F95 (ROM 0).

Each iteration of this loop takes 26 T-states. The time taken by 26 T-states is 7.33e-6 seconds. So the total time for the loop to execute is 2.5/TEMPO seconds.

Entry: The value 4\*TEMPO exists on the calculator stack (where TEMPO is in the range 60..240).

Exit : The calculator stack holds the result.

LOA50:	RST 28H	Invoke the floating point calculator.
	DEFB \$A4	stk-ten. = x, 10
	DEFB \$01	exchange. = 10, x
	DEFB \$05	division. = 10/x
	DEFB \$34	stk-data. = 10/x, 7.33e-6
	DEFB \$DF	- exponent \$6F (floating point number 7.33e-6).
	DEFB \$75	- mantissa byte 1
	DEFB \$F4	- mantissa byte 2
	DEFB \$38	- mantissa byte 3
	DEFB \$75	- mantissa byte 4
	DEFB \$05	division. = (10/x)/7.33e-6
	DEFB \$38	end-calc.
	RET	

## Test BREAK Key

Test for BREAK being pressed.

Exit: Carry flag reset if BREAK is being pressed.

LOA5D:	LD A,\$7F	
	IN A,(\$FE)	
	RRA	
	RET C	Return with carry flag set if SPACE not pressed.
	LD A,\$FE	
	IN A,(\$FE)	
	RRA	
	RET	Return with carry flag set if CAPS not pressed.

## Select Channel Data Block Duration Pointers

Point to the start of the channel data block duration pointers within the command data block.

Entry: IY=Address of the command data block.

Exit : HL=Address of current channel pointer.

L0A69: LD BC,\$0011 Offset to the channel data block duration pointers table.  
JR L0A71 Jump ahead to continue.

## Select Channel Data Block Pointers

Point to the start of the channel data block pointers within the command data block.

Entry: IY=Address of the command data block.

Exit: HL=Address of current channel pointer.

L0A6E: LD BC,\$0000 Offset to the channel data block pointers table.  
L0A71: PUSH IY  
POP HL HL=Point to the command data block.  
ADD HL,BC Point to the desired channel pointers table.  
LD (IY+\$23),L  
LD (IY+\$24),H Store the start address of channels pointer table.  
LD A,(IY+\$10) Fetch the channel bitmap.  
LD (IY+\$22),A Initialise the working copy.  
LD (IY+\$21),\$01 Channel selector. Set the shift register to indicate the first channel.  
RET

## Get Channel Data Block Address for Current String

L0A86: LD E,(HL)  
INC HL  
LD D,(HL) Fetch the address of the current channel data block.  
PUSH DE  
POP IX Return it in IX.  
RET

## Next Channel Data Pointer

L0A8D: LD L,(IY+\$23) The address of current channel data pointer.  
LD H,(IY+\$24)  
INC HL  
INC HL Advance to the next channel data pointer.  
LD (IY+\$23),L  
LD (IY+\$24),H The address of new channel data pointer.  
RET

## PLAY Command (Continuation)

This section is responsible for processing the PLAY command and is a continuation of the routine at \$09A4 (ROM 0). It begins by determining the first note to play on each channel and then enters a loop to play these notes, fetching the subsequent notes to play at the appropriate times.

L0A9C: CALL L0A6E Select channel data block pointers.  
L0A9F: RR (IY+\$22) Working copy of channel bitmap. Test if next string present.  
JR C,L0AAB Jump ahead if there is no string for this channel.

HL=Address of channel data pointer.

L0AAB: CALL L0A86 Get address of channel data block for the current string into IX.  
CALL L0B7B Find the first note to play for this channel from its play string.  
SLA (IY+\$21) Have all channels been processed?  
JR C,L0AB6 Jump ahead if so.  
CALL L0A8D Advance to the next channel data block pointer.  
JR L0A9F Jump back to process the next channel.

The first notes to play for each channel have now been determined. A loop is entered that coordinates playing the notes and fetching subsequent notes when required. Notes across channels may be of different lengths and so the shortest one is determined, the tones for all channels set and then a waiting

delay entered for the shortest note delay. This delay length is then subtracted from all channel note lengths to leave the remaining lengths that each note needs to be played for. For the channel with the smallest note length, this will now have completely played and so a new note is fetched for it. The smallest length of the current notes is then determined again and the process described above repeated. A test is made on each iteration to see if all channels have run out of data to play, and if so this ends the PLAY command.

L0AB6:	CALL L0FB0	Find smallest duration length of the current notes across all channels.
	PUSH DE	Save the smallest duration length.
	CALL L0F61	Play a note on each channel.
	POP DE	DE=The smallest duration length.
L0ABE:	LD A,(IY+\$10)	Channel bitmap.
	CP \$FF	Is there anything to play?
	JR NZ,L0ACA	Jump if there is.
	CALL L0EB2	Turn off all sound and restore IY.
	EI	Re-enable interrupts.
	RET	End of play command.
L0ACA:	DEC DE	DE=Smallest channel duration length, i.e. duration until the next channel state change.
	CALL L0F95	Perform a wait.
	CALL L0FE0	Play a note on each channel and update the channel duration lengths.
	CALL L0FB0	Find smallest duration length of the current notes across all channels.
	JR L0ABE	Jump back to see if there is more to process.

## PLAY Command Character Table

Recognised characters in PLAY commands.

L0AD6:        DEFM "HZYXWUVM)(NO!"

## Get Play Character

Get the current character from the PLAY string and then increment the character pointer within the string.

Exit: Carry flag set if string has been fully processed.

Carry flag reset if character is available.

A=Character available.

L0AE4:	CALL L0F02	Get the current character from the play string for this channel.
	RET C	Return if no more characters.
	INC (IX+\$06)	Increment the low byte of the string pointer.
	RET NZ	Return if it has not overflowed.
	INC (IX+\$07)	Else increment the high byte of the string pointer.
	RET	Returns with carry flag reset.

## Get Next Note in Semitones

Finds the number of semitones above C for the next note in the string,

Entry:        IX=Address of the channel data block.

Exit :        A=Number of semitones above C, or \$80 for a rest.

L0AF0:	PUSH HL	Save HL.
	LD C,\$00	Default is for a 'natural' note, i.e. no adjustment.
L0AF3:	CALL L0AE4	Get the current character from the PLAY string, and advance the position pointer.
	JR C,L0B00	Jump if at the end of the string.
	CP '&'	\$26. Is it a rest?
	JR NZ,L0B0B	Jump ahead if not.
	LD A,\$80	Signal that it is a rest.
L0AFE:	POP HL	Restore HL.
	RET	
L0B00:	LD A,(IY+\$21)	Fetch the channel selector.
	OR (IY+\$10)	Clear the channel flag for this string.
	LD (IY+\$10),A	Store the new channel bitmap.
	JR L0AFE	Jump back to return.
L0B0B:	CP '#'	\$23. Is it a sharpen?

	JR NZ,L0B12	Jump ahead if not.
	INC C	Increment by a semitone.
	JR L0AF3	Jump back to get the next character.
L0B12:	CP '\$'	\$24. Is it a flatten?
	JR NZ,L0B19	Jump ahead if not.
	DEC C	Decrement by a semitone.
	JR L0AF3	Jump back to get the next character.
L0B19:	BIT 5,A	Is it a lower case letter?
	JR NZ,L0B23	Jump ahead if lower case.
	PUSH AF	It is an upper case letter so
	LD A,\$0C	increase an octave
	ADD A,C	by adding 12 semitones.
	LD C,A	
	POP AF	
L0B23:	AND \$DF	Convert to upper case.
	SUB \$41	Reduce to range 'A'->0 .. 'G'->6.
	JP C,L0F41	Jump if below 'A' to produce error report "k Invalid note name".
	CP \$07	Is it 7 or above?
	JP NC,L0F41	Jump if so to produce error report "k Invalid note name".
	PUSH BC	C=Number of semitones.
	LD B,\$00	
	LD C,A	BC holds 0..6 for 'a'..'g'.
	LD HL,L0E18	Look up the number of semitones above note C for the note.
	ADD HL,BC	
	LD A,(HL)	A=Number of semitones above note C.
	POP BC	C=Number of semitones due to sharpen/flatten characters.
	ADD A,C	Adjust number of semitones above note C for the sharpen/flatten characters.
	POP HL	Restore HL.
	RET	

## Get Numeric Value from Play String

Get a numeric value from a PLAY string, returning 0 if no numeric value present.

Entry: IX=Address of the channel data block.

Exit : BC=Numeric value, or 0 if no numeric value found.

L0B3C:	PUSH HL	Save registers.
	PUSH DE	
	LD L,(IX+\$06)	Get the pointer into the PLAY string.
	LD H,(IX+\$07)	
	LD DE,\$0000	Initialise result to 0.
L0B47:	LD A,(HL)	
	CP '0'	\$30. Is character numeric?
	JR C,L0B64	Jump ahead if not.
	CP ':'	\$3A. Is character numeric?
	JR NC,L0B64	Jump ahead if not.
	INC HL	Advance to the next character.
	PUSH HL	Save the pointer into the string.
	CALL L0B6F	Multiply result so far by 10.
	SUB '0'	\$30. Convert ASCII digit to numeric value.
	LD H,\$00	
	LD L,A	HL=Numeric digit value.
	ADD HL,DE	Add the numeric value to the result so far.
	JR C,L0B61	Jump ahead if an overflow to produce error report "l number too big".
	EX DE,HL	Transfer the result into DE.
	POP HL	Retrieve the pointer into the string.
	JR L0B47	Loop back to handle any further numeric digits.
L0B61:	JP L0F39	Jump to produce error report "l number too big". [Could have saved 1 byte by directly using JP C,\$0F39 (ROM 0) instead of using this JP and the two JR C,\$0B61 (ROM 0) instructions that come here]

The end of the numeric value was reached

L0B64:	LD (IX+\$06),L	Store the new pointer position into the string.
	LD (IX+\$07),H	

```

PUSH DE
POP BC
POP DE
POP HL
RET

```

Return the result in BC.  
Restore registers.

## Multiply DE by 10

```

L0B6F:    LD HL,$0000
          LD B,$0A
L0B74:    ADD HL,DE
          JR C,L0B61
          DJNZ L0B74
          EX DE,HL
          RET

```

Add DE to HL ten times.

Jump ahead if an overflow to produce error report "I number too big".

Transfer the result into DE.

## Find Next Note from Channel String

```

L0B7B:    CALL L0A5D
          JR C,L0B88
          CALL L0EB2
          EI
          CALL L05CB
          DEFB $14
L0B88:    CALL L0AE4
          JP C,L0DC1
          CALL L0E0F
          LD B,$00
          SLA C
          LD HL,L0DE9
          ADD HL,BC
          LD E,(HL)
          INC HL
          LD D,(HL)
          EX DE,HL
          CALL L0BA3
          JR L0B7B

```

Test for BREAK being pressed.  
Jump ahead if not pressed.  
Turn off all sound and restore IY.  
Re-enable interrupts.  
Produce error report. [Could have saved 1 byte by using JP \$05F5 (ROM 0)]  
"L Break into program"

Get the current character from the PLAY string, and advance the position pointer.  
Jump if at the end of the string.  
Find the handler routine for the PLAY command character.

Generate the offset into the  
command vector table.  
HL points to handler routine for this command character.

Fetch the handler routine address.  
HL=Handler routine address for this command character.  
Make an indirect call to the handler routine.  
Jump back to handle the next character in the string.

Comes here after processing a non-numeric digit that does not have a specific command routine handler Hence the next note to play has been determined and so a return is made to process the other channels.

```

L0BA2:    RET
L0BA3:    JP (HL)

```

Just make a return.  
Jump to the command handler routine.

## Play Command '!' (Comment)

A comment is enclosed within exclamation marks, e.g. "! A comment !".

Entry: IX=Address of the channel data block.

```

L0BA4:    CALL L0AE4
          JP C,L0DC0
          CP '!'
          RET Z
          JR L0BA4

```

Get the current character from the PLAY string, and advance the position pointer.  
Jump if at the end of the string.  
\$21. Is it the end-of-comment character?  
Return if it is.  
Jump back to test the next character.

## Play Command 'O' (Octave)

The 'O' command is followed by a numeric value within the range 0 to 8, although due to loose range checking the value MOD 256 only needs to be within 0 to 8. Hence O256 operates the same as O0.

Entry: IX=Address of the channel data block.

L0BAF:	CALL L0B3C	Get following numeric value from the string into BC.
	LD A,C	Is it between 0 and 8?
	CP \$09	
	JP NC,L0F31	Jump if above 8 to produce error report "n Out of range".
	SLA A	Multiply A by 12.
	SLA A	
	LD B,A	
	SLA A	
	ADD A,B	
	LD (IX+\$03),A	Store the octave value.
	RET	

## Play Command 'N' (Separator)

The 'N' command is simply a separator marker and so is ignored.

Entry: IX=Address of the channel data block.

L0BC4:	RET	Nothing to do so make an immediate return.
--------	-----	--

## Play Command '(' (Start of Repeat)

A phrase can be enclosed within brackets causing it to be repeated, i.e. played twice.

Entry: IX=Address of the channel data block.

L0BC5:	LD A,(IX+\$0B)	A=Current level of open bracket nesting.
	INC A	Increment the count.
	CP \$05	Only 4 levels supported.
	JP Z,L0F49	Jump if this is the fifth to produce error report "d Too many brackets".
	LD (IX+\$0B),A	Store the new open bracket nesting level.
	LD DE,\$000C	Offset to the bracket level return position stores.
	CALL L0C46	HL=Address of the pointer in which to store the return location of the bracket.
	LD A,(IX+\$06)	Store the current string position as the return address of the open bracket.
	LD (HL),A	
	INC HL	
	LD A,(IX+\$07)	
	LD (HL),A	
	RET	

## Play Command ')' (End of Repeat)

A phrase can be enclosed within brackets causing it to be repeated, i.e. played twice.

Brackets can also be nested within each other, to 4 levels deep.

If a closing bracket is used without a matching opening bracket then the whole string up until that point is repeated indefinitely.

Entry: IX=Address of the channel data block.

L0BE1:	LD A,(IX+\$16)	Fetch the nesting level of closing brackets.
	LD DE,\$0017	Offset to the closing bracket return address store.
	OR A	Is there any bracket nesting so far?
	JP M,L0C0F	Jump if none. [Could have been faster by jumping to \$0C12 (ROM 0)]

Has the bracket level been repeated, i.e. re-reached the same position in the string as the closing bracket return address?

CALL L0C46	HL=Address of the pointer to the corresponding closing bracket return address store.
LD A,(IX+\$06)	Fetch the low byte of the current address.
CP (HL)	Re-reached the closing bracket?
JR NZ,L0C0F	Jump ahead if not.
INC HL	Point to the high byte.
LD A,(IX+\$07)	Fetch the high byte address of the current address.
CP (HL)	Re-reached the closing bracket?

## SPECTRUM +2 ROM o DISASSEMBLY

JR NZ,LOC0F

Jump ahead if not.

The bracket level has been repeated. Now check whether this was the outer bracket level.

DEC (IX+\$16)  
LD A,(IX+\$16)

Decrement the closing bracket nesting level since this level has been repeated.  
[There is no need for the LD A,(IX+\$16) and OR A instructions since the DEC (IX+\$16) already set the flags]

OR A  
RET P

Reached the outer bracket nesting level?  
Return if not the outer bracket nesting level such that the character after the closing bracket is processed next.

The outer bracket level has been repeated

BIT 0,(IX+\$0A)  
RET Z

Was this a single closing bracket?  
Return if it was not.

The repeat was caused by a single closing bracket so re-initialise the repeat

LD (IX+\$16),\$00  
XOR A  
JR LOC2A

Restore one level of closing bracket nesting.  
Select closing bracket nesting level 0.  
Jump ahead to continue.

A new level of closing bracket nesting

LOC0F: LD A,(IX+\$16)  
INC A  
CP \$05

Fetch the nesting level of closing brackets.  
Increment the count.  
Only 5 levels supported (4 to match up with opening brackets and a 5th to repeat indefinitely).

JP Z,LOC49  
LD (IX+\$16),A  
CALL LOC46  
LD A,(IX+\$06)  
LD (HL),A  
INC HL

Jump if this is the fifth to produce error report "d Too many brackets".  
Store the new closing bracket nesting level.  
HL=Address of the pointer to the appropriate closing bracket return address store.  
Store the current string position as the return address for the closing bracket.

LD A,(IX+\$07)  
LD (HL),A

LOC2A: LD A,(IX+\$0B)  
LD DE,\$000C

Fetch the nesting level of opening brackets.

CALL LOC46  
LD A,(HL)  
LD (IX+\$06),A  
INC HL  
LD A,(HL)  
LD (IX+\$07),A  
DEC (IX+\$0B)  
RET P

HL=Address of the pointer to the opening bracket nesting level return address store.  
Set the return address of the nesting level's opening bracket as new current position within the string.

For a single closing bracket only, this will be the start address of the string.

Decrement level of open bracket nesting.  
Return if the closing bracket matched an open bracket.

There is one more closing bracket then opening brackets, i.e. repeat string indefinitely

LD (IX+\$0B),\$00  
SET 0,(IX+\$0A)  
RET

Set the opening brackets nesting level to 0.  
Signal a single closing bracket only, i.e. to repeat the string indefinitely.

### Get Address of Bracket Pointer Store

LOC46: PUSH IX  
POP HL  
ADD HL,DE  
LD B,\$00  
LD C,A  
SLA C  
ADD HL,BC

HL=IX.  
HL=IX+DE.

HL=IX+DE+2\*A.

RET

## Play Command 'T' (Tempo)

A temp command must be specified in the first play string and is followed by a numeric value in the range 60 to 240 representing the number of beats (crotchets) per minute.

Entry: IX=Address of the channel data block.

L0C51:	CALL L0B3C	Get following numeric value from the string into BC.
	LD A,B	
	OR A	
	JP NZ,L0F31	Jump if 256 or above to produce error report "n Out of range".
	LD A,C	
	CP \$3C	
	JP C,L0F31	Jump if 59 or below to produce error report "n Out of range".
	CP \$F1	
	JP NC,L0F31	Jump if 241 or above to produce error report "n Out of range".
A holds a value in the range 60 to 240		
	LD A,(IX+\$02)	Fetch the channel number.
	OR A	Tempo 'T' commands have to be specified in the first string.
	RET NZ	If it is in a later string then ignore it.
	LD B,\$00	[Redundant instruction - B is already zero]
	PUSH BC	C=Tempo value.
	POP HL	
	ADD HL,HL	
	ADD HL,HL	HL=Tempo*4.
	PUSH HL	
	POP BC	BC=Tempo*4. [Would have been quicker to use the combination LD B,H and LD C,L]
	PUSH IY	Save the pointer to the play command data block.
	RST 28H	
	DEFW STACK_BC	\$2D2B. Place the contents of BC onto the stack. The call restores IY to \$5C3A.
	DI	Interrupts get re-enabled by the call mechanism to ROM 1 so disable them again.
	POP IY	Restore IY to point at the play command data block.
	PUSH IY	Save the pointer to the play command data block.
	PUSH IY	
	POP HL	HL=pointer to the play command data block.
	LD BC,\$002B	
	ADD HL,BC	HL =IY+\$002B.
	LD IY,\$5C3A	Reset IY to \$5C3A since this is required by the floating point calculator.
	PUSH HL	HL=Points to the calculator RAM routine.
	LD HL,L0C95	
	LD (RETADDR),HL	\$5B5A. Set up the return address.
	LD HL,YOUNGER	
	EX (SP),HL	Stack the address of the swap routine used when returning to this ROM.
	PUSH HL	Re-stack the address of the calculator RAM routine.
	JP SWAP	\$5B00. Toggle to other ROM and make a return to the calculator RAM routine.

## Tempo Command Return

The calculator stack now holds the value  $(10/(\text{Tempo} \times 4))/7.33\text{e-}6$  and this is stored as the tempo value.

The result is used an inner loop counter in the wait routine at \$0F95 (ROM 0). Each iteration of this loop takes 26 T-states. The time taken by 26 T-states is  $7.33\text{e-}6$  seconds. So the total time for the loop to execute is  $2.5/\text{TEMPO}$  seconds.

L0C95:	DI	Interrupts get re-enabled by the call mechanism to ROM 1 so disable them again.
	RST 28H	
	DEFW FP_TO_BC	\$2DA2. Fetch the value on the top of the calculator stack.
	DI	Interrupts get re-enabled by the call mechanism to ROM 1 so disable them again.
	POP IY	Restore IY to point at the play command data block.
	LD (IY+\$27),C	Store tempo timing value.
	LD (IY+\$28),B	

RET

## Play Command 'M' (Mixer)

This command is used to select whether to use tone and/or noise on each of the 3 channels.

It is followed by a numeric value in the range 1 to 63, although due to loose range checking the value MOD 256 only needs to be within 0 to 63. Hence M256 operates the same as M0.

Entry: IX=Address of the channel data block.

L0CA3:	CALL L0B3C LD A,C CP \$40 JP NC,L0F31	Get following numeric value from the string into BC. A=Mixer value. Is it 64 or above? Jump if so to produce error report "n Out of range".
--------	--	--

Bit 0: 1=Enable channel A tone.  
 Bit 1: 1=Enable channel B tone.  
 Bit 2: 1=Enable channel C tone.  
 Bit 3: 1=Enable channel A noise.  
 Bit 4: 1=Enable channel B noise.  
 Bit 5: 1=Enable channel C noise.

CPL  LD E,A LD D,\$07 CALL L0E9B RET	Invert the bits since the sound generator's mixer register uses active low enable. This also sets bit 6 1, which selects the I/O port as an output. E=Mixer value. D=Register 7 - Mixer. Write to sound generator register to set the mixer. [Could have saved 1 byte by using JP \$0E9B (ROM 0)]
---	--

## Play Command 'V' (Volume)

This sets the volume of a channel and is followed by a numeric value in the range 0 (minimum) to 15 (maximum), although due to loose range checking the value MOD 256 only needs to be within 0 to 15. Hence V256 operates the same as V0.

Entry: IX=Address of the channel data block.

L0CB4:	CALL L0B3C LD A,C CP \$10 JP NC,L0F31 LD (IX+\$04),A	Get following numeric value from the string into BC.  Is it 16 or above? Jump if so to produce error report "n Out of range". Store the volume level.
--------	--	---

**[BUG** - An attempt to set the volume for a sound chip channel is now made. However, this routine fails to take into account that it is also called to set the volume for a MIDI only channel, i.e. play strings 4 to 8. As a result, corruption occurs to various sound generator registers, causing spurious sound output. There is in fact no need for this routine to set the volume for any channels since this is done every time a new note is played - see routine at \$0AB6 (ROM 0). the bug fix is to simply to make a return at this point. This routine therefore contains 11 surplus bytes. Credit: Ian Collier (+3), Paul Farrow (128)]

LD E,(IX+\$02) LD A,\$08 ADD A,E LD D,A LD E,C CALL L0E9B RET	E=Channel number. Offset by 8. A=8+index. D=Sound generator register number for the channel. E=Volume level. Write to sound generator register to set the volume for the channel. [Could have saved 1 byte by using JP \$0E9B (ROM 0)]
---	--

## Play Command 'U' (Use Volume Effect)

This command turns on envelope waveform effects for a particular sound chip channel. The volume level is now controlled by the selected envelope waveform for the channel, as defined by the 'W' command. MIDI channels do not support envelope waveforms and so the routine has the effect of setting the volume of a MIDI channel to maximum, i.e. 15. It might seem odd that the volume for MIDI channels is set to 15 rather than just filtered out. However, the three sound chip channels can also drive three MIDI channels and so it would be inconsistent for these MIDI channels to have their volume set to 15 but have the other MIDI channels behave differently. However, it could be argued that all MIDI channels should be unaffected by the 'U' command.

There are no parameters to this command.

Entry: IX=Address of the channel data block.

L0CCC:	LD E,(IX+\$02)	Get the channel number.
	LD A,\$08	Offset by 8.
	ADD A,E	A=8+index.
	LD D,A	D=Sound generator register number for the channel. [This is not used and so there is no need to generate it. It was probably a left over from copying and modifying the 'V' command routine. Deleting it would save 7 bytes. Credit: Ian Collier (+3), Paul Farrow (128)]
	LD E,\$1F	E=Select envelope defined by register 13, and reset volume bits to maximum (though these are not used with the envelope).
	LD (IX+\$04),E	Store that the envelope is being used (along with the reset volume level).
	RET	

## Play command 'W' (Volume Effect Specifier)

This command selects the envelope waveform to use and is followed by a numeric value in the range 0 to 7, although due to loose range checking the value MOD 256 only needs to be within 0 to 7.

Hence W256 operates the same as W0.

Entry: IX=Address of the channel data block.

L0CD9:	CALL L0B3C	Get following numeric value from the string into BC.
	LD A,C	
	CP \$08	Is it 8 or above?
	JP NC,L0F31	Jump if so to produce error report "n Out of range".
	LD B,\$00	
	LD HL,L0E07	Envelope waveform lookup table.
	ADD HL,BC	HL points to the corresponding value in the table.
	LD A,(HL)	
	LD (IY+\$29),A	Store new effect waveform value.
	RET	

## Play Command 'X' (Volume Effect Duration)

This command allows the duration of a waveform effect to be specified, and is followed by a numeric value in the range 0 to 65535. A value of 1 corresponds to the minimum duration, increasing up to 65535 and then maximum duration for a value of 0. If no numeric value is specified then the maximum duration is used.

Entry: IX=Address of the channel data block.

L0CED:	CALL L0B3C	Get following numeric value from the string into BC.
	LD D,\$0B	Register 11 - Envelope Period Fine.
	LD E,C	
	CALL L0E9B	Write to sound generator register to set the envelope period (low byte).
	INC D	Register 12 - Envelope Period Coarse.
	LD E,B	
	CALL L0E9B	Write to sound generator register to set the envelope period (high byte).
	RET	[Could have saved 1 byte by using JP \$0E9B (ROM 0)]

## Play Command 'Y' (MIDI Channel)

This command sets the MIDI channel number that the string is assigned to and is followed by a numeric value in the range 1 to 16, although due to loose range checking the value MOD 256 only needs to be within 1 to 16.

Hence Y257 operates the same as Y1.

Entry: IX=Address of the channel data block.

L0CFC:	CALL L0B3C	Get following numeric value from the string into BC.
	LD A,C	
	DEC A	Is it 0?
	JP M,L0F31	Jump if so to produce error report "n Out of range".
	CP \$10	Is it 10 or above?
	JP NC,L0F31	Jump if so to produce error report "n Out of range".
	LD (IX+\$01),A	Store MIDI channel number that this string is assigned to.
	RET	

## Play Command 'Z' (MIDI Programming Code)

This command is used to send a programming code to the MIDI port. It is followed by a numeric value in the range 0 to 255, although due to loose range checking the value MOD 256 only needs to be within 0 to 255. Hence Z256 operates the same as Z0.

Entry: IX=Address of the channel data block.

L0D0D:	CALL L0B3C LD A,C CALL L11C2 RET	Get following numeric value from the string into BC. A=(low byte of) the value. Write byte to MIDI device. [Could have saved 1 byte by using JP \$0E9B (ROM 0)]
--------	---	--

## Play Command 'H' (Stop)

This command stops further processing of a play command. It has no parameters.

Entry: IX=Address of the channel data block.

L0D15:	LD (IY+\$10),\$FF RET	Indicate no channels to play, thereby causing the play command to terminate.
--------	--------------------------	---

## Play Commands 'a'..'g', 'A'..'G', '1'.."12", '&' and '\_'

This handler routine processes commands 'a'..'g', 'A'..'G', '1'.."12", '&' and '\_', and determines the length of the next note to play. It provides the handling of triplet and tied notes.

It stores the note duration in the channel data block's duration length entry, and sets a pointer in the command data block's duration lengths pointer table to point at it. A single note letter is deemed to be a tied note count of 1. Triplets are deemed a tied note count of at least 2.

Entry: IX=Address of the channel data block.

A=Current character from play string.

L0D1A:	CALL L0E38 JP C,L0DA0	Is the current character a number? Jump if not number digit.
--------	--------------------------	---

The character is a number digit

CALL L0DCB CALL L0DD3	HL=Address of the duration length within the channel data block. Store address of duration length in command data block's channel duration length pointer table.
XOR A LD (IX+\$21),A CALL L0EE7 CALL L0B3C LD A,C OR A	Set no tied notes. Get the previous character in the string, the note duration. Get following numeric value from the string into BC.
JP Z,L0F31 CP \$0D JP NC,L0F31 CP \$0A JR C,L0D51	Is the value 0? Jump if so to produce error report "n Out of range". Is it 13 or above? Jump if so to produce error report "n Out of range". Is it below 10? Jump if so.

It is a triplet semi-quaver (10), triplet quaver (11) or triplet crotchet (12)

CALL L0E1F CALL L0D93 LD (HL),E INC HL LD (HL),D CALL L0D93 INC HL LD (HL),E INC HL LD (HL),D INC HL	DE=Note duration length for the duration value. Increment the tied notes counter. HL=Address of the duration length within the channel data block.  Store the duration length. Increment the counter of tied notes.  Store the subsequent note duration length in the channel data block.
--	--

## SPECTRUM +2 ROM o DISASSEMBLY

JR L0D57

Jump ahead to continue.

The note duration was in the range 1 to 9

<p>L0D51: LD (IX+\$05),C CALL L0E1F L0D57: CALL L0D93 L0D5A: CALL L0F02 CP '_' JR NZ,L0D8D CALL L0AE4 CALL L0B3C LD A,C CP \$0A JR C,L0D7E</p>	<p>C=Note duration value (1..9). DE=Duration length for this duration value. Increment the tied notes counter. Get the current character from the play string for this channel. \$5F. Is it a tied note? Jump ahead if not. Get the current character from the PLAY string, and advance the position pointer. Get following numeric value from the string into BC. Place the value into A. Is it below 10? Jump ahead for 1 to 9 (semiquaver ... semibreve).</p>
--	--

A triplet note was found as part of a tied note

<p>PUSH HL PUSH DE CALL L0E1F POP HL ADD HL,DE LD C,E LD B,D EX DE,HL POP HL LD (HL),E INC HL LD (HL),D LD E,C LD D,B JR L0D47</p>	<p>HL=Address of the duration length within the channel data block. DE=First tied note duration length. DE=Note duration length for this new duration value. HL=Current tied note duration length. HL=Current+new tied note duration lengths.  BC=Note duration length for the duration value. DE=Current+new tied note duration lengths. HL=Address of the duration length within the channel data block.  Store the combined note duration length in the channel data block.  DE=Note duration length for the second duration value. Jump back.</p>
--	---

A non-triplet tied note

<p>L0D7E: LD (IX+\$05),C PUSH HL PUSH DE CALL L0E1F POP HL ADD HL,DE EX DE,HL POP HL JP L0D5A</p>	<p>Store the note duration value. HL=Address of the duration length within the channel data block. DE=First tied note duration length. DE=Note duration length for this new duration value. HL=Current tied note duration length. HL=Current+new tied not duration lengths. DE=Current+new tied not duration lengths. HL=Address of the duration length within the channel data block. Jump back to process the next character in case it is also part of a tied note.</p>
---	--

The number found was not part of a tied note, so store the duration value

<p>L0D8D: LD (HL),E INC HL LD (HL),D JP L0DBB</p>	<p>HL=Address of the duration length within the channel data block. (For triplet notes this could be the address of the subsequent note duration length) Store the duration length. Jump forward to make a return.</p>
---	--

This subroutine is called to increment the tied notes counter

<p>L0D93: LD A,(IX+\$21) INC A CP \$0B JP Z,L0F59 LD (IX+\$21),A RET</p>	<p>Increment counter of tied notes.  Has it reached 11? Jump if so to produce to error report "o too many tied notes". Store the new tied notes counter.</p>
--	--

The character is not a number digit so is 'A'..'G', '&' or '\_'

## SPECTRUM +2 ROM 0 DISASSEMBLY

L0DA0:	CALL L0EE7 LD (IX+\$21),\$01	Get the previous character from the string. Set the number of tied notes to 1.
Store a pointer to the channel data block's duration length into the command data block		
	CALL L0DCB CALL L0DD3	HL=Address of the duration length within the channel data block. Store address of duration length in command data block's channel duration length pointer table.
	LD C,(IX+\$05) PUSH HL CALL L0E1F POP HL LD (HL),E INC HL LD (HL),D JP L0DBB	C=The duration value of the note (1 to 9). [Not necessary] Find the duration length for the note duration value. [Not necessary] Store it in the channel data block.
L0DBB:	POP HL INC HL INC HL PUSH HL RET	Jump to the instruction below. [Redundant instruction]  Modify the return address to point to the RET instruction at \$0BA2 (ROM 0).  [Over elaborate when a simple POP followed by RET would have sufficed, saving 3 bytes]

## End of String Found

This routine is called when the end of string is found within a comment. It marks the string as having been processed and then returns to the main loop to process the next string.

L0DC0:	POP HL	Drop the return address of the call to the comment command.
--------	--------	---

Enter here if the end of the string is found whilst processing a string.

L0DC1:	LD A,(IY+\$21) OR (IY+\$10) LD (IY+\$10),A RET	Fetch the channel selector. Clear the channel flag for this string. Store the new channel bitmap.
--------	---	---

## Point to Duration Length within Channel Data Block

L0DCB:	PUSH IX POP HL LD BC,\$0022 ADD HL,BC RET	HL=Address of the channel data block.  HL=Address of the store for the duration length.
--------	---	---

## Store Entry in Command Data Block's Channel Duration Length Pointer Table

L0DD3:	PUSH HL PUSH IY POP HL LD BC,\$0011 ADD HL,BC  LD B,\$00 LD C,(IX+\$02) SLA C ADD HL,BC  POP DE	Save the address of the duration length within the channel data block.  HL=Address of the command data block.  HL=Address within the command data block of the channel duration length pointer table.  BC=Channel number. BC=2*Index number. HL=Address within the command data block of the pointer to the current channel's data block duration length. DE=Address of the duration length within the channel data block.
--------	--	---

```
LD (HL),E
INC HL
LD (HL),D
EX DE,HL
RET
```

Store the pointer to the channel duration length in the command data block's channel duration pointer table.

## PLAY Command Jump Table

Handler routine jump table for all PLAY commands.

L0DE9:	DEFW L0D1A	Command handler routine for all other characters.
	DEFW L0BA4	'!' command handler routine.
	DEFW L0BAF	'O' command handler routine.
	DEFW L0BC4	'N' command handler routine.
	DEFW L0BC5	('' command handler routine.
	DEFW L0BE1	'J' command handler routine.
	DEFW L0C51	'T' command handler routine.
	DEFW L0CA3	'M' command handler routine.
	DEFW L0CB4	'V' command handler routine.
	DEFW L0CCC	'U' command handler routine.
	DEFW L0CD9	'W' command handler routine.
	DEFW L0CED	'X' command handler routine.
	DEFW L0CFC	'Y' command handler routine.
	DEFW L0D0D	'Z' command handler routine.
	DEFW L0D15	'H' command handler routine.

## Envelope Waveform Lookup Table

Table used by the play 'W' command to find the corresponding envelope value to write to the sound generator envelope shape register (register 13). This filters out the two duplicate waveforms possible from the sound generator and allows the order of the waveforms to be arranged in a more logical fashion.

L0E07:	DEFB \$00	W0 - Single decay then off. (Continue off, attack off, alternate off, hold off)
	DEFB \$04	W1 - Single attack then off. (Continue off, attack on, alternate off, hold off)
	DEFB \$0B	W2 - Single decay then hold. (Continue on, attack off, alternate on, hold on)
	DEFB \$0D	W3 - Single attack then hold. (Continue on, attack on, alternate off, hold on)
	DEFB \$08	W4 - Repeated decay. (Continue on, attack off, alternate off, hold off)
	DEFB \$0C	W5 - Repeated attack. (Continue on, attack on, alternate off, hold off)
	DEFB \$0E	W6 - Repeated attack-decay. (Continue on, attack on, alternate on, hold off)
	DEFB \$0A	W7 - Repeated decay-attack. (Continue on, attack off, alternate on, hold off)

## Identify Command Character

This routines attempts to match the command character to those in a table.

The index position of the match indicates which command handler routine is required to process the character. Note that commands are case sensitive.

Entry: A=Command character.

Exit : Zero flag set if a match was found.

BC=Identifying the character matched, 1 to 15 for match and 0 for no match.

L0E0F:	LD BC,\$000F	Number of characters + 1 in command table.
	LD HL,L0AD6	Start of command table.
	CPIR	Search for a match.
	RET	

## Semitones Table

This table contains an entry for each note of the scale, A to G, and is the number of semitones above the note C.

L0E18:	DEFB \$09	'A'
	DEFB \$0B	'B'
	DEFB \$00	'C'

DEFB \$02	'D'
DEFB \$04	'E'
DEFB \$05	'F'
DEFB \$07	'G'

## Find Note Duration Length

L0E1F:	PUSH HL	Save HL.
	LD B,\$00	
	LD HL,L0E2B	Note duration table.
	ADD HL,BC	Index into the table.
	LD D,\$00	
	LD E,(HL)	Fetch the length from the table.
	POP HL	Restore HL.
	RET	

## Note Duration Table

A whole note is given by a value of 96d and other notes defined in relation to this.  
The value of 96d is the lowest common denominator from which all note durations can be defined.

L0E2B:	DEFB \$80	Rest [Not used since table is always indexed into with a value of 1 or more]
	DEFB \$06	Semi-quaver (sixteenth note).
	DEFB \$09	Dotted semi-quaver (3/32th note).
	DEFB \$0C	Quaver (eighth note).
	DEFB \$12	Dotted quaver (3/16th note).
	DEFB \$18	Crotchet (quarter note).
	DEFB \$24	Dotted crotchet (3/8th note).
	DEFB \$30	Minim (half note).
	DEFB \$48	Dotted minim (3/4th note).
	DEFB \$60	Semi-breve (whole note).
	DEFB \$04	Triplet semi-quaver (1/24th note).
	DEFB \$08	Triplet quaver (1/12th note).
	DEFB \$10	Triplet crotchet (1/6th note).

## Is Numeric Digit?

Tests whether a character is a number digit.

Entry: A=Character.

Exit : Carry flag reset if a number digit.

L0E38:	CP '0'	\$30. Is it '0' or less?
	RET C	Return with carry flag set if so.
	CP ':'	\$3A. Is it more than '9'?
	CCF	
	RET	Return with carry flag set if so.

## Play a Note On a Sound Chip Channel

This routine plays the note at the current octave and current volume on a sound chip channel. For play strings 4 to 8, it simply stores the note number and this is subsequently played later.

Entry: IX=Address of the channel data block.

A=Note value as number of semitones above C (0..11).

L0E3F:	LD C,A	C=The note value.
	LD A,(IX+\$03)	Octave number * 12.
	ADD A,C	Add the octave number and the note value to form the note number.
	CP \$80	Is note within range?
	JP NC,L0F51	Jump if not to produce error report "m Note out of range".
	LD C,A	C=Note number.

## SPECTRUM +2 ROM 0 DISASSEMBLY

```
LD A,(IX+$02)    Get the channel number.
OR A             Is it the first channel?
JR NZ,L0E5E      Jump ahead if not.
```

Only set the noise generator frequency on the first channel

```
L0E5E: LD A,C          A=Note number (0..107), in ascending audio frequency.
        CPL          Invert since noise register value is in descending audio frequency.
        AND $7F      Mask off bit 7.
        SRL A
        SRL A
        LD D,$06     Divide by 4 to reduce range to 0..31.
        LD E,A       Register 6 - Noise pitch.
        CALL L0E9B   Write to sound generator register.
        LD (IX+$00),C Store the note number.
        LD A,(IX+$02) Get the channel number.
        CP $03      Is it channel 0, 1 or 2, i.e. a sound chip channel?
        RET NC      Do not output anything for play strings 4 to 8.
```

Channel 0, 1 or 2

```
LD HL,L10B5      Start of note lookup table.
LD B,$00        BC=Note number.
LD A,C          A=Note number.
SUB $15         A=Note number - 21.
JR NC,L0E76     Jump if note number was 21 or above.
LD DE,$0FBF     Note numbers $00 to $14 use the lowest note value.
JR L0E7D        [Could have saved 4 bytes by using XOR A and dropping through to $0E76 (ROM 0)]
```

Note number 21 to 107 (range 0 to 86)

```
L0E76: LD C,A          Generate offset into the table.
        SLA C        Point to the entry in the table.
        ADD HL,BC
        LD E,(HL)
        INC HL
        LD D,(HL)
L0E7D: EX DE,HL      DE=Word to write to the sound chip registers to produce this note.
        LD D,(IX+$02) HL=Register word value to produce the note.
        SLA D        Get the channel number.
                     D=2*Channel number, to give the tone channel register (fine control) number 0, 2, or 4.
        LD E,L       E=The low value byte.
        CALL L0E9B   Write to sound generator register.
        INC D        D=Tone channel register (coarse control) number 1, 3, or 5.
        LD E,H       E=The high value byte.
        CALL L0E9B   Write to sound generator register.
        BIT 4,(IX+$04) Is the envelope waveform being used?
        RET Z        Return if it is not.
        LD D,$0D     Register 13 - Envelope Shape.
        LD A,(IY+$29) Get the effect waveform value.
        LD E,A
        CALL L0E9B   Write to sound generator register.
        RET          [Could have saved 4 bytes by dropping down into the routine below.]
```

## Set Sound Generator Register

```
L0E9B: PUSH BC
        LD BC,$FFFD
        OUT (C),D    Select the register.
        LD BC,$BFFD
        OUT (C),E    Write out the value.
        POP BC
```

RET

## Read Sound Generator Register

L0EA8:	PUSH BC LD BC,\$FFFD OUT (C),A IN A,(C) POP BC RET	Select the register. Read the register's value.
--------	---	--

## Turn Off All Sound

L0EB2:	LD D,\$07 LD E,\$FF CALL L0E9B	Register 7 - Mixer. I/O ports are inputs, noise output off, tone output off. Write to sound generator register.
--------	--------------------------------------	---

Turn off the sound from the AY-3-8912

LD D,\$08 LD E,\$00 CALL L0E9B INC D CALL L0E9B INC D CALL L0E9B CALL L0A6E	Register 8 - Channel A volume. Volume of 0. Write to sound generator register to set the volume to 0. Register 9 - Channel B volume. Write to sound generator register to set the volume to 0. Register 10 - Channel C volume. Write to sound generator register to set the volume to 0. Select channel data block pointers.
--	---

Now reset all MIDI channels in use

L0ECB:	RR (IY+\$22) JR C,L0ED7 CALL L0A86 CALL L11AC	Working copy of channel bitmap. Test if next string present. Jump ahead if there is no string for this channel. Get address of channel data block for the current string into IX. Turn off the MIDI channel sound assigned to this play string.
L0ED7:	SLA (IY+\$21) JR C,L0EE2 CALL L0A8D JR L0ECB	Have all channels been processed? Jump ahead if so. Advance to the next channel data block pointer. Jump back to process the next channel.
L0EE2:	LD IY,\$5C3A RET	Restore IY.

## Get Previous Character from Play String

Get the previous character from the PLAY string, skipping over spaces and 'Enter' characters.

Entry: IX=Address of the channel data block.

L0EE7:	PUSH HL PUSH DE LD L,(IX+\$06) LD H,(IX+\$07)	Save registers.  Get the current pointer into the PLAY string.
L0EEF:	DEC HL LD A,(HL) CP ' ' JR Z,L0EEF CP \$0D JR Z,L0EEF LD (IX+\$06),L LD (IX+\$07),H POP DE POP HL	Point to previous character. Fetch the character. \$20. Is it a space? Jump back if a space. Is it an 'Enter'? Jump back if an 'Enter'. Store this as the new current pointer into the PLAY string.  Restore registers.

RET

## Get Current Character from Play String

Get the current character from the PLAY string, skipping over spaces and 'Enter' characters.

Exit: Carry flag set if string has been fully processed.

Carry flag reset if character is available.

A=Character available.

L0F02:	PUSH HL	Save registers.
	PUSH DE	
	PUSH BC	
	LD L,(IX+\$06)	HL=Pointer to next character to process within the PLAY string.
	LD H,(IX+\$07)	
L0F0B:	LD A,H	
	CP (IX+\$09)	Reached end-of-string address high byte?
	JR NZ,L0F1A	Jump forward if not.
	LD A,L	
	CP (IX+\$08)	Reached end-of-string address low byte?
	JR NZ,L0F1A	Jump forward if not.
	SCF	Indicate string all processed.
	JR L0F24	Jump forward to return.
L0F1A:	LD A,(HL)	Get the next play character.
	CP ' '	\$20. Is it a space?
	JR Z,L0F28	Ignore the space by jumping ahead to process the next character.
	CP \$0D	Is it 'Enter'?
	JR Z,L0F28	Ignore the 'Enter' by jumping ahead to process the next character.
	OR A	Clear the carry flag to indicate a new character has been returned.
L0F24:	POP BC	Restore registers.
	POP DE	
	POP HL	
	RET	
L0F28:	INC HL	Point to the next character.
	LD (IX+\$06),L	
	LD (IX+\$07),H	Update the pointer to the next character to process with the PLAY string.
	JR L0F0B	Jump back to get the next character.

## Produce Play Error Reports

L0F31:	CALL L0EB2	Turn off all sound and restore IY.
	EI	
	CALL L05CB	Produce error report.
	DEFB \$29	"n Out of range"
L0F39:	CALL L0EB2	Turn off all sound and restore IY.
	EI	
	CALL L05CB	Produce error report.
	DEFB \$27	"l Number too big"
L0F41:	CALL L0EB2	Turn off all sound and restore IY.
	EI	
	CALL L05CB	Produce error report.
	DEFB \$26	"k Invalid note name"
L0F49:	CALL L0EB2	Turn off all sound and restore IY.
	EI	
	CALL L05CB	Produce error report.
	DEFB \$1F	"d Too many brackets"
L0F51:	CALL L0EB2	Turn off all sound and restore IY.
	EI	
	CALL L05CB	Produce error report.
	DEFB \$28	"m Note out of range"
L0F59:	CALL L0EB2	Turn off all sound and restore IY.
	EI	
	CALL L05CB	Produce error report.
	DEFB \$2A	"o Too many tied notes"

## Play Note on Each Channel

Play a note and set the volume on each channel for which a play string exists.

L0F61:	CALL L0A6E	Select channel data block pointers.
L0F64:	RR (IY+\$22)	Working copy of channel bitmap. Test if next string present.
	JR C,L0F8B	Jump ahead if there is no string for this channel.
	CALL L0A86	Get address of channel data block for the current string into IX.
	CALL L0AF0	Get the next note in the string as number of semitones above note C.
	CP \$80	Is it a rest?
	JR Z,L0F8B	Jump ahead if so and do nothing to the channel.
	CALL L0E3F	Play the note if a sound chip channel.
	LD A,(IX+\$02)	Get channel number.
	CP \$03	Is it channel 0, 1 or 2, i.e. a sound chip channel?
	JR NC,L0F88	Jump if not to skip setting the volume.

One of the 3 sound chip generator channels so set the channel's volume for the new note

	LD D,\$08	
	ADD A,D	A=0 to 2.
	LD D,A	D=Register (8 + string index), i.e. channel A, B or C volume register.
	LD E,(IX+\$04)	E=Volume for the current channel.
	CALL L0E9B	Write to sound generator register to set the output volume.
L0F88:	CALL L118D	Play a note and set the volume on the assigned MIDI channel.
L0F8B:	SLA (IY+\$21)	Have all channels been processed?
	RET C	Return if so.
	CALL L0A8D	Advance to the next channel data block pointer.
	JR L0F64	Jump back to process the next channel.

## Wait Note Duration

This routine is the main timing control of the PLAY command.

It waits for the specified length of time, which will be the lowest note duration of all active channels.

The actual duration of the wait is dictated by the current tempo.

Entry: DE=Note duration, where 96d represents a whole note.

Enter a loop waiting for  $(135 + ((26 * (\text{tempo} - 100)) - 5)) * \text{DE} + 5$  T-states

L0F95:	PUSH HL	(11) Save HL.
	LD L,(IY+\$27)	(19) Get the tempo timing value.
	LD H,(IY+\$28)	(19)
	LD BC,\$0064	(10) BC=100
	OR A	(4)
	SBC HL,BC	(15) HL=tempo timing value - 100.
	PUSH HL	(11)
	POP BC	(10) BC=tempo timing value - 100.
	POP HL	(10) Restore HL.

Tempo timing value =  $(10 / (\text{TEMPO} * 4)) / 7.33\text{e-}6$ , where  $7.33\text{e-}6$  is the time for 26 T-states.

The loop below takes 26 T-states per iteration, where the number of iterations is given by the tempo timing value.

So the time for the loop to execute is  $2.5 / \text{TEMPO}$  seconds.

For a TEMPO of 60 beats (crotchets) per second, the time per crotchet is 1/24 second.

The duration of a crotchet is defined as 24 from the table at \$0E0C, therefore the loop will get executed 24 times and hence the total time taken will be 1 second.

The tempo timing value above has 100 subtracted from it, presumably to approximately compensate for the overhead time previously taken to prepare the notes for playing. This reduces the total time by 2600 T-states, or 733us.

L0FA5:	DEC BC	(6) Wait for tempo-100 loops.
	LD A,B	(4)
	OR C	(4)
	JR NZ,L0FA5	(12/7)
	DEC DE	(6) Repeat DE times
	LD A,D	(4)
	OR E	(4)

JR NZ,L0F95 (12/7)  
RET (10)

## Find Smallest Duration Length

This routine finds the smallest duration length for all current notes being played across all channels.

Exit: DE=Smallest duration length.

L0FB0: LD DE,\$FFFF Set smallest duration length to 'maximum'.  
CALL L0A69 Select channel data block duration pointers.  
L0FB6: RR (IY+\$22) Working copy of channel bitmap. Test if next string present.  
JR C,L0FCE Jump ahead if there is no string for this channel.

HL=Address of channel data pointer. DE holds the smallest duration length found so far.

PUSH DE Save the smallest duration length.  
LD E,(HL)  
INC HL  
LD D,(HL)  
EX DE,HL DE=Channel data block duration length.  
LD E,(HL)  
INC HL  
LD D,(HL) DE=Channel duration length.  
PUSH DE  
POP HL HL=Channel duration length.  
POP BC Last channel duration length.  
OR A  
SBC HL,BC Is current channel's duration length smaller than the smallest so far?  
JR C,L0FCE Jump ahead if so, with the new smallest value in DE.

The current channel's duration was not smaller so restore the last smallest into DE.

PUSH BC  
POP DE DE=Smallest duration length.  
L0FCE: SLA (IY+\$21) Have all channel strings been processed?  
JR C,L0FD9 Jump ahead if so.  
CALL L0A8D Advance to the next channel data block duration pointer.  
JR L0FB6 Jump back to process the next channel.  
L0FD9: LD (IY+\$25),E  
LD (IY+\$26),D Store the smallest channel duration length.  
RET

## Play a Note on Each Channel and Update Channel Duration Lengths

This routine is used to play a note and set the volume on all channels.

It subtracts an amount of time from the duration lengths of all currently playing channel note durations. The amount subtracted is equivalent to the smallest note duration length currently being played, and as determined earlier.

Hence one channel's duration will go to 0 on each call of this routine, and the others will show the remaining lengths of their corresponding notes.

Entry: IY=Address of the command data block.

L0FE0: XOR A  
LD (IY+\$2A),A Holds a temporary channel bitmap.  
CALL L0A6E Select channel data block pointers.  
L0FE7: RR (IY+\$22) Working copy of channel bitmap. Test if next string present.  
JP C,L1079 Jump ahead if there is no string for this channel.  
CALL L0A86 Get address of channel data block for the current string into IX.  
PUSH IY  
POP HL HL=Address of the command data block.  
LD BC,\$0011  
ADD HL,BC HL=Address of channel data block duration pointers.  
LD B,\$00  
LD C,(IX+\$02) BC=Channel number.  
SLA C BC=2\*Channel number.

## SPECTRUM +2 ROM 0 DISASSEMBLY

ADD HL,BC	HL=Address of channel data block duration pointer for this channel.
LD E,(HL)	
INC HL	
LD D,(HL)	DE=Address of duration length within the channel data block.
EX DE,HL	HL=Address of duration length within the channel data block.
PUSH HL	Save it.
LD E,(HL)	
INC HL	
LD D,(HL)	DE=Duration length for this channel.
EX DE,HL	HL=Duration length for this channel.
LD E,(Y+\$25)	
LD D,(Y+\$26)	DE=Smallest duration length of all current channel notes.
OR A	
SBC HL,DE	HL=Duration length - smallest duration length.
EX DE,HL	DE=Duration length - smallest duration length.
POP HL	HL=Address of duration length within the channel data block.
JR Z,L101B	Jump if this channel uses the smallest found duration length.
LD (HL),E	
INC HL	Update the duration length for this channel with the remaining length.
LD (HL),D	
JR L1079	Jump ahead to update the next channel.

The current channel uses the smallest found duration length

[A note has been completed and so the channel volume is set to 0 prior to the next note being played. This occurs on both sound chip channels and MIDI channels. When a MIDI channel is assigned to more than one play string and a rest is used in one of those strings. As soon as the end of the rest period is encountered, the channel's volume is set to off even though one of the other play strings controlling the MIDI channel may still be playing. This can be seen using the command PLAY "Y1a&", "Y1N9a". Here, string 1 starts playing 'a' for the period of a crotchet (1/4 of a note), where as string 2 starts playing 'a' for nine periods of a crotchet (9/4 of a note). When string 1 completes its crotchet, it requests to play a period of silence via the rest '&'. This turns the volume of the MIDI channel off even though string 2 is still timing its way through its nine crotchets. The play command will therefore continue for a further seven crotchets but in silence. This is because the volume for note is set only at its start and no coordination occurs between strings to turn the volume back on for the second string. It is arguably what the correct behaviour should be in such a circumstance where the strings are providing conflicting instructions, but having the latest command or note take precedence seems a logical approach. Credit: Ian Collier (+3), Paul Farrow (128)]

L101B:	LD A,(IX+\$02)	Get the channel number.
	CP \$03	Is it channel 0, 1 or 2, i.e. a sound chip channel?
	JR NC,L102B	Jump ahead if not a sound generator channel.
	LD D,\$08	
	ADD A,D	
	LD D,A	D=Register (8+channel number) - Channel volume.
	LD E,\$00	E=Volume level of 0.
	CALL L0E9B	Write to sound generator register to turn the volume off.
L102B:	CALL L11AC	Turn off the assigned MIDI channel sound.
	PUSH IX	
	POP HL	HL=Address of channel data block.
	LD BC,\$0021	
	ADD HL,BC	HL=Points to the tied notes counter.
	DEC (HL)	Decrement the tied notes counter. [This contains a value of 1 for a single note]
	JR NZ,L1045	Jump ahead if there are more tied notes.
	CALL L0B7B	Find the next note to play for this channel from its play string.
	LD A,(Y+\$21)	Fetch the channel selector.
	AND (Y+\$10)	Test whether this channel has further data in its play string.
	JR NZ,L1079	Jump to process the next channel if this channel does not have a play string.
	JR L105C	The channel has more data in its play string so jump ahead.

The channel has more tied notes

L1045:	PUSH IY	
	POP HL	HL=Address of the command data block.
	LD BC,\$0011	
	ADD HL,BC	HL=Address of channel data block duration pointers.
	LD B,\$00	
	LD C,(IX+\$02)	BC=Channel number.
	SLA C	BC=2*Channel number.
	ADD HL,BC	HL=Address of channel data block duration pointer for this channel.
	LD E,(HL)	
	INC HL	

## SPECTRUM +2 ROM 0 DISASSEMBLY

	LD D,(HL)	DE=Address of duration length within the channel data block.
	INC DE	
	INC DE	
	LD (HL),D	Point to the subsequent note duration length.
	DEC HL	
	LD (HL),E	Store the new duration length.
L105C:	CALL L0AF0	Get next note in the string as number of semitones above note C.
	LD C,A	C=Number of semitones.
	LD A,(IY+\$21)	Fetch the channel selector.
	AND (IY+\$10)	Test whether this channel has a play string.
	JR NZ,L1079	Jump to process the next channel if this channel does not have a play string.
	LD A,C	A=Number of semitones.
	CP \$80	Is it a rest?
	JR Z,L1079	Jump to process the next channel if it is.
	CALL L0E3F	Play the new note on this channel at the current volume if a sound chip channel, or simply store the note for play strings 4 to 8.
	LD A,(IY+\$21)	Fetch the channel selector.
	OR (IY+\$2A)	Insert a bit in the temporary channel bitmap to indicate this channel has more to play.
	LD (IY+\$2A),A	Store it.

Check whether another channel needs its duration length updated

L1079:	SLA (IY+\$21)	Have all channel strings been processed?
	JR C,L1085	Jump ahead if so.
	CALL L0A8D	Advance to the next channel data pointer.
	JP L0FE7	Jump back to update the duration length for the next channel.

**[BUG -** By this point, the volume for both sound chip and MIDI channels has been set to 0, i.e. off. So although the new notes have been set playing on the sound chip channels, no sound is audible. For MIDI channels, no new notes have yet been output and hence these are also silent. If the time from turning the volume off for the current note to the time to turn the volume on for the next note is short enough, then it will not be noticeable. However, the code at \$1085 (ROM 0) introduces a 1/96th of a note delay and as a result a 1/96th of a note period of silence between notes. The bug can be resolved by simply deleting the two instructions below that introduce the delay. A positive side effect of the bug in the 'V' volume command at \$0CB4 (ROM 0) is that it can be used to overcome the gaps of silence between notes for sound chip channels. By interspersing volume commands between notes, a new volume level is immediately set before the 1/96th of a note delay is introduced for the new note. Therefore, the delay occurs when the new note is audible instead of when it is silent. For example, PLAY "cV15cV15c" instead of PLAY "ccc". The note durations are still 1/96th of a note longer than they should be though. This technique will only work on the sound chip channels and not for any MIDI channels. Credit: Ian Collier (+3), Paul Farrow (128)]

L1085:	LD DE,\$0001	Delay for 1/96th of a note.
	CALL L0F95	
	CALL L0A6E	Select channel data block pointers.

All channel durations have been updated. Update the volume on each sound chip channel, and the volume and note on each MIDI channel

L108E:	RR (IY+\$2A)	Temporary channel bitmap. Test if next string present.
	JR NC,L10AB	Jump ahead if there is no string for this channel.
	CALL L0A86	Get address of channel data block for the current string into IX.
	LD A,(IX+\$02)	Get the channel number.
	CP \$03	Is it channel 0, 1 or 2, i.e. a sound chip channel?
	JR NC,L10A8	Jump ahead if so to process the next channel.
	LD D,\$08	
	ADD A,D	
	LD D,A	D=Register (8+channel number) - Channel volume.
	LD E,(IX+\$04)	Get the current volume.
	CALL L0E9B	Write to sound generator register to set the volume of the channel.
L10A8:	CALL L118D	Play a note and set the volume on the assigned MIDI channel.
L10AB:	SLA (IY+\$21)	Have all channels been processed?
	RET C	Return if so.
	CALL L0A8D	Advance to the next channel data pointer.
	JR L108E	Jump back to process the next channel.

## Note Lookup Table

Each word gives the value of the sound generator tone registers for a given note.

# SPECTRUM +2 ROM o DISASSEMBLY

There are 9 octaves, containing a total of 108 notes. These represent notes 21 to 128. Notes 0 to 20 cannot be reproduced on the sound chip and so note 21 will be used for all of these (they will however be sent to a MIDI device if one is assigned to a channel). [Note that both the sound chip and the MIDI port can not play note 128 and so its inclusion in the table is a waste of 2 bytes]. The PLAY command does not allow octaves higher than 8 to be selected directly. Using PLAY "O8G" will select note 115. To select higher notes, sharps must be included, e.g. PLAY "O8#G" for note 116, PLAY "O8##G" for note 117, etc, up to PLAY "O8#####G" for note 127. Attempting to access note 128 using PLAY "O8#####G" will lead to error report "m Note out of range".

L10B5:	DEFW \$0FBF	Octave 1, Note 21 - A (27.50 Hz, Ideal=27.50 Hz, Error=-0.01%) C0
	DEFW \$0EDC	Octave 1, Note 22 - A# (29.14 Hz, Ideal=29.16 Hz, Error=-0.08%)
	DEFW \$0E07	Octave 1, Note 23 - B (30.87 Hz, Ideal=30.87 Hz, Error=-0.00%)
	DEFW \$0D3D	Octave 2, Note 24 - C (32.71 Hz, Ideal=32.70 Hz, Error=+0.01%) C1
	DEFW \$0C7F	Octave 2, Note 25 - C# (34.65 Hz, Ideal=34.65 Hz, Error=-0.00%)
	DEFW \$0BCC	Octave 2, Note 26 - D (36.70 Hz, Ideal=36.71 Hz, Error=-0.01%)
	DEFW \$0B22	Octave 2, Note 27 - D# (38.89 Hz, Ideal=38.89 Hz, Error=+0.01%)
	DEFW \$0A82	Octave 2, Note 28 - E (41.20 Hz, Ideal=41.20 Hz, Error=+0.00%)
	DEFW \$09EB	Octave 2, Note 29 - F (43.66 Hz, Ideal=43.65 Hz, Error=+0.00%)
	DEFW \$095D	Octave 2, Note 30 - F# (46.24 Hz, Ideal=46.25 Hz, Error=-0.02%)
	DEFW \$08D6	Octave 2, Note 31 - G (49.00 Hz, Ideal=49.00 Hz, Error=+0.00%)
	DEFW \$0857	Octave 2, Note 32 - G# (51.92 Hz, Ideal=51.91 Hz, Error=+0.01%)
	DEFW \$07DF	Octave 2, Note 33 - A (55.01 Hz, Ideal=55.00 Hz, Error=+0.01%)
	DEFW \$076E	Octave 2, Note 34 - A# (58.28 Hz, Ideal=58.33 Hz, Error=-0.08%)
	DEFW \$0703	Octave 2, Note 35 - B (61.75 Hz, Ideal=61.74 Hz, Error=+0.02%)
	DEFW \$069F	Octave 3, Note 36 - C (65.39 Hz, Ideal=65.41 Hz, Error=-0.02%) C2
	DEFW \$0640	Octave 3, Note 37 - C# (69.28 Hz, Ideal=69.30 Hz, Error=-0.04%)
	DEFW \$05E6	Octave 3, Note 38 - D (73.40 Hz, Ideal=73.42 Hz, Error=-0.01%)
	DEFW \$0591	Octave 3, Note 39 - D# (77.78 Hz, Ideal=77.78 Hz, Error=+0.01%)
	DEFW \$0541	Octave 3, Note 40 - E (82.41 Hz, Ideal=82.41 Hz, Error=+0.00%)
	DEFW \$04F6	Octave 3, Note 41 - F (87.28 Hz, Ideal=87.31 Hz, Error=-0.04%)
	DEFW \$04AE	Octave 3, Note 42 - F# (92.52 Hz, Ideal=92.50 Hz, Error=+0.02%)
	DEFW \$046B	Octave 3, Note 43 - G (98.00 Hz, Ideal=98.00 Hz, Error=+0.00%)
	DEFW \$042C	Octave 3, Note 44 - G# (103.78 Hz, Ideal=103.83 Hz, Error=-0.04%)
	DEFW \$03F0	Octave 3, Note 45 - A (109.96 Hz, Ideal=110.00 Hz, Error=-0.04%)
	DEFW \$03B7	Octave 3, Note 46 - A# (116.55 Hz, Ideal=116.65 Hz, Error=-0.08%)
	DEFW \$0382	Octave 3, Note 47 - B (123.43 Hz, Ideal=123.47 Hz, Error=-0.03%)
	DEFW \$034F	Octave 4, Note 48 - C (130.86 Hz, Ideal=130.82 Hz, Error=+0.04%) C3
	DEFW \$0320	Octave 4, Note 49 - C# (138.55 Hz, Ideal=138.60 Hz, Error=-0.04%)
	DEFW \$02F3	Octave 4, Note 50 - D (146.81 Hz, Ideal=146.83 Hz, Error=-0.01%)
	DEFW \$02C8	Octave 4, Note 51 - D# (155.68 Hz, Ideal=155.55 Hz, Error=+0.08%)
	DEFW \$02A1	Octave 4, Note 52 - E (164.70 Hz, Ideal=164.82 Hz, Error=-0.07%)
	DEFW \$027B	Octave 4, Note 53 - F (174.55 Hz, Ideal=174.62 Hz, Error=-0.04%)
	DEFW \$0257	Octave 4, Note 54 - F# (185.04 Hz, Ideal=185.00 Hz, Error=+0.02%)
	DEFW \$0236	Octave 4, Note 55 - G (195.83 Hz, Ideal=196.00 Hz, Error=-0.09%)
	DEFW \$0216	Octave 4, Note 56 - G# (207.57 Hz, Ideal=207.65 Hz, Error=-0.04%)
	DEFW \$01F8	Octave 4, Note 57 - A (219.92 Hz, Ideal=220.00 Hz, Error=-0.04%)
	DEFW \$01DC	Octave 4, Note 58 - A# (232.86 Hz, Ideal=233.30 Hz, Error=-0.19%)
	DEFW \$01C1	Octave 4, Note 59 - B (246.86 Hz, Ideal=246.94 Hz, Error=-0.03%)
	DEFW \$01A8	Octave 5, Note 60 - C (261.42 Hz, Ideal=261.63 Hz, Error=-0.08%) C4 Middle C
	DEFW \$0190	Octave 5, Note 61 - C# (277.10 Hz, Ideal=277.20 Hz, Error=-0.04%)
	DEFW \$0179	Octave 5, Note 62 - D (294.01 Hz, Ideal=293.66 Hz, Error=+0.12%)
	DEFW \$0164	Octave 5, Note 63 - D# (311.35 Hz, Ideal=311.10 Hz, Error=+0.08%)
	DEFW \$0150	Octave 5, Note 64 - E (329.88 Hz, Ideal=329.63 Hz, Error=+0.08%)
	DEFW \$013D	Octave 5, Note 65 - F (349.65 Hz, Ideal=349.23 Hz, Error=+0.12%)
	DEFW \$012C	Octave 5, Note 66 - F# (369.47 Hz, Ideal=370.00 Hz, Error=-0.14%)
	DEFW \$011B	Octave 5, Note 67 - G (391.66 Hz, Ideal=392.00 Hz, Error=-0.09%)
	DEFW \$010B	Octave 5, Note 68 - G# (415.13 Hz, Ideal=415.30 Hz, Error=-0.04%)
	DEFW \$00FC	Octave 5, Note 69 - A (439.84 Hz, Ideal=440.00 Hz, Error=-0.04%)
	DEFW \$00EE	Octave 5, Note 70 - A# (465.72 Hz, Ideal=466.60 Hz, Error=-0.19%)
	DEFW \$00E0	Octave 5, Note 71 - B (494.82 Hz, Ideal=493.88 Hz, Error=+0.19%)
	DEFW \$00D4	Octave 6, Note 72 - C (522.83 Hz, Ideal=523.26 Hz, Error=-0.08%) C5
	DEFW \$00C8	Octave 6, Note 73 - C# (554.20 Hz, Ideal=554.40 Hz, Error=-0.04%)
	DEFW \$00BD	Octave 6, Note 74 - D (586.46 Hz, Ideal=587.32 Hz, Error=-0.15%)
	DEFW \$00B2	Octave 6, Note 75 - D# (622.70 Hz, Ideal=622.20 Hz, Error=+0.08%)
	DEFW \$00A8	Octave 6, Note 76 - E (659.77 Hz, Ideal=659.26 Hz, Error=+0.08%)
	DEFW \$009F	Octave 6, Note 77 - F (697.11 Hz, Ideal=698.46 Hz, Error=-0.19%)
	DEFW \$0096	Octave 6, Note 78 - F# (738.94 Hz, Ideal=740.00 Hz, Error=-0.14%)
	DEFW \$008D	Octave 6, Note 79 - G (786.10 Hz, Ideal=784.00 Hz, Error=+0.27%)
	DEFW \$0085	Octave 6, Note 80 - G# (833.39 Hz, Ideal=830.60 Hz, Error=+0.34%)

DEFW \$007E	Octave 6, Note 81 - A (879.69 Hz, Ideal=880.00 Hz, Error=-0.04%)
DEFW \$0077	Octave 6, Note 82 - A# (931.43 Hz, Ideal=933.20 Hz, Error=-0.19%)
DEFW \$0070	Octave 6, Note 83 - B (989.65 Hz, Ideal=987.76 Hz, Error=+0.19%)
DEFW \$006A	Octave 7, Note 84 - C (1045.67 Hz, Ideal=1046.52 Hz, Error=-0.08%) C6
DEFW \$0064	Octave 7, Note 85 - C# (1108.41 Hz, Ideal=1108.80 Hz, Error=-0.04%)
DEFW \$005E	Octave 7, Note 86 - D (1179.16 Hz, Ideal=1174.64 Hz, Error=+0.38%)
DEFW \$0059	Octave 7, Note 87 - D# (1245.40 Hz, Ideal=1244.40 Hz, Error=+0.08%)
DEFW \$0054	Octave 7, Note 88 - E (1319.53 Hz, Ideal=1318.52 Hz, Error=+0.08%)
DEFW \$004F	Octave 7, Note 89 - F (1403.05 Hz, Ideal=1396.92 Hz, Error=+0.44%)
DEFW \$004B	Octave 7, Note 90 - F# (1477.88 Hz, Ideal=1480.00 Hz, Error=-0.14%)
DEFW \$0047	Octave 7, Note 91 - G (1561.14 Hz, Ideal=1568.00 Hz, Error=-0.44%)
DEFW \$0043	Octave 7, Note 92 - G# (1654.34 Hz, Ideal=1661.20 Hz, Error=-0.41%)
DEFW \$003F	Octave 7, Note 93 - A (1759.38 Hz, Ideal=1760.00 Hz, Error=-0.04%)
DEFW \$003B	Octave 7, Note 94 - A# (1878.65 Hz, Ideal=1866.40 Hz, Error=+0.66%)
DEFW \$0038	Octave 7, Note 95 - B (1979.30 Hz, Ideal=1975.52 Hz, Error=+0.19%)
DEFW \$0035	Octave 8, Note 96 - C (2091.33 Hz, Ideal=2093.04 Hz, Error=-0.08%) C7
DEFW \$0032	Octave 8, Note 97 - C# (2216.81 Hz, Ideal=2217.60 Hz, Error=-0.04%)
DEFW \$002F	Octave 8, Note 98 - D (2358.31 Hz, Ideal=2349.28 Hz, Error=+0.38%)
DEFW \$002D	Octave 8, Note 99 - D# (2463.13 Hz, Ideal=2488.80 Hz, Error=-1.03%)
DEFW \$002A	Octave 8, Note 100 - E (2639.06 Hz, Ideal=2637.04 Hz, Error=+0.08%)
DEFW \$0028	Octave 8, Note 101 - F (2771.02 Hz, Ideal=2793.84 Hz, Error=-0.82%)
DEFW \$0025	Octave 8, Note 102 - F# (2995.69 Hz, Ideal=2960.00 Hz, Error=+1.21%)
DEFW \$0023	Octave 8, Note 103 - G (3166.88 Hz, Ideal=3136.00 Hz, Error=+0.98%)
DEFW \$0021	Octave 8, Note 104 - G# (3358.81 Hz, Ideal=3322.40 Hz, Error=+1.10%)
DEFW \$001F	Octave 8, Note 105 - A (3575.50 Hz, Ideal=3520.00 Hz, Error=+1.58%)
DEFW \$001E	Octave 8, Note 106 - A# (3694.69 Hz, Ideal=3732.80 Hz, Error=-1.02%)
DEFW \$001C	Octave 8, Note 107 - B (3958.59 Hz, Ideal=3951.04 Hz, Error=+0.19%)
DEFW \$001A	Octave 9, Note 108 - C (4263.10 Hz, Ideal=4186.08 Hz, Error=+1.84%) C8
DEFW \$0019	Octave 9, Note 109 - C# (4433.63 Hz, Ideal=4435.20 Hz, Error=-0.04%)
DEFW \$0018	Octave 9, Note 110 - D (4618.36 Hz, Ideal=4698.56 Hz, Error=-1.71%)
DEFW \$0016	Octave 9, Note 111 - D# (5038.21 Hz, Ideal=4977.60 Hz, Error=+1.22%)
DEFW \$0015	Octave 9, Note 112 - E (5278.13 Hz, Ideal=5274.08 Hz, Error=+0.08%)
DEFW \$0014	Octave 9, Note 113 - F (5542.03 Hz, Ideal=5587.68 Hz, Error=-0.82%)
DEFW \$0013	Octave 9, Note 114 - F# (5833.72 Hz, Ideal=5920.00 Hz, Error=-1.46%)
DEFW \$0012	Octave 9, Note 115 - G (6157.81 Hz, Ideal=6272.00 Hz, Error=-1.82%)
DEFW \$0011	Octave 9, Note 116 - G# (6520.04 Hz, Ideal=6644.80 Hz, Error=-1.88%)
DEFW \$0010	Octave 9, Note 117 - A (6927.54 Hz, Ideal=7040.00 Hz, Error=-1.60%)
DEFW \$000F	Octave 9, Note 118 - A# (7389.38 Hz, Ideal=7465.60 Hz, Error=-1.02%)
DEFW \$000E	Octave 9, Note 119 - B (7917.19 Hz, Ideal=7902.08 Hz, Error=+0.19%)
DEFW \$000D	Octave 10, Note 120 - C ( 8526.20 Hz, Ideal= 8372.16 Hz, Error=+1.84%) C9
DEFW \$000C	Octave 10, Note 121 - C# ( 9236.72 Hz, Ideal= 8870.40 Hz, Error=+4.13%)
DEFW \$000C	Octave 10, Note 122 - D ( 9236.72 Hz, Ideal= 9397.12 Hz, Error=-1.71%)
DEFW \$000B	Octave 10, Note 123 - D# (10076.42 Hz, Ideal= 9955.20 Hz, Error=+1.22%)
DEFW \$000B	Octave 10, Note 124 - E (10076.42 Hz, Ideal=10548.16 Hz, Error=-4.47%)
DEFW \$000A	Octave 10, Note 125 - F (11084.06 Hz, Ideal=11175.36 Hz, Error=-0.82%)
DEFW \$0009	Octave 10, Note 126 - F# (12315.63 Hz, Ideal=11840.00 Hz, Error=+4.02%)
DEFW \$0009	Octave 10, Note 127 - G (12315.63 Hz, Ideal=12544.00 Hz, Error=-1.82%)
DEFW \$0008	Octave 10, Note 128 - G# (13855.08 Hz, Ideal=13289.60 Hz, Error=+4.26%)

## Play Note on MIDI Channel

This routine turns on a note on the MIDI channel and sets its volume, if MIDI channel is assigned to the current string.

Three bytes are sent, and have the following meaning:

Byte 1: Channel number \$00..\$0F, with bits 4 and 7 set.

Byte 2: Note number \$00..\$7F.

Byte 3: Note velocity \$00..\$78.

Entry: IX=Address of the channel data block.

L118D:	LD A,(IX+\$01)	Is a MIDI channel assigned to this string?
	OR A	
	RET M	Return if not.

A holds the assigned channel number (\$00..\$0F)

OR \$90	Set bits 4 and 7 of the channel number. A=\$90..\$9F.
---------	---

CALL L11C2	Write byte to MIDI device.
LD A,(IX+\$00)	The note number.
CALL L11C2	Write byte to MIDI device.
LD A,(IX+\$04)	Fetch the channel's volume.
RES 4,A	Ensure the 'using envelope' bit is reset so
SLA A	that A holds a value between \$00 and \$0F.
SLA A	Multiply by 8 to increase the range to \$00..\$78.
SLA A	A=Note velocity.
CALL L11C2	Write byte to MIDI device.
RET	[Could have saved 1 byte by using JP \$11C2 (ROM 0)]

## Turn MIDI Channel Off

This routine turns off a note on the MIDI channel, if a MIDI channel is assigned to the current string.

Three bytes are sent, and have the following meaning:

Byte 1: Channel number \$00..\$0F, with bit 7 set.

Byte 2: Note number \$00..\$7F.

Byte 3: Note velocity \$40.

Entry: IX=Address of the channel data block.

L11AC:	LD A,(IX+\$01)	Is a MIDI channel assigned to this string?
	OR A	
	RET M	Return if not.

A holds the assigned channel number (\$00..\$0F)

OR \$80	Set bit 7 of the channel number. A=\$80..\$8F.
CALL L11C2	Write byte to MIDI device.
LD A,(IX+\$00)	The note number.
CALL L11C2	Write byte to MIDI device.
LD A,\$40	The note velocity.
CALL L11C2	Write byte to MIDI device.
RET	[Could have saved 1 byte by using JP \$11C2 (ROM 0)]

## Send Byte to MIDI Device

This routine sends a byte to the MIDI port. MIDI devices communicate at 31250 baud, although this routine actually generates a baud rate of 31388, which is within the 1% tolerance supported by MIDI devices.

Entry: A=Byte to send.

L11C2:	LD L,A	Store the byte to send.
	LD BC,\$FFFD	
	LD A,\$0E	
	OUT (C),A	Select register 14 - I/O port.
	LD BC,\$BFFD	
	LD A,\$FA	Set RS232 'RXD' transmit line to 0. (Keep KEYPAD 'CTS' output line low to prevent the keypad resetting)
	OUT (C),A	Send out the START bit.
L11D3:	LD E,\$03	(7) Introduce delays such that the next bit is output 113 T-states from now.
	DEC E	(4)
	JR NZ,L11D3	(12/7)
	NOP	(4)
	NOP	(4)
	NOP	(4)
	NOP	(4)
	LD A,L	(4) Retrieve the byte to send.
	LD D,\$08	(7) There are 8 bits to send.
L11DD:	RRA	(4) Rotate the next bit to send into the carry.
	LD L,A	(4) Store the remaining bits.
	JP NC,L11E8	(10) Jump if it is a 0 bit.
	LD A,\$FE	(7) Set RS232 'RXD' transmit line to 1. (Keep KEYPAD 'CTS' output line low to prevent the keypad resetting)
	OUT (C),A	(11)

L11E8:	JR L11EE LD A,\$FA	(12) Jump forward to process the next bit. (7) Set RS232 'RXD' transmit line to 0. (Keep KEYPAD 'CTS' output line low to prevent the keypad resetting)
	OUT (C),A	(11)
L11EE:	JR L11EE	(12) Jump forward to process the next bit.
L11F0:	LD E,\$02	(7) Introduce delays such that the next data bit is output 113 T-states from now.
	DEC E	(4)
	JR NZ,L11F0	(12/7)
	NOP	(4)
	ADD A,\$00	(7)
	LD A,L	(4) Retrieve the remaining bits to send.
	DEC D	(4) Decrement the bit counter.
	JR NZ,L11DD	(12/7) Jump back if there are further bits to send.
	NOP	(4) Introduce delays such that the stop bit is output 113 T-states from now.
	NOP	(4)
	ADD A,\$00	(7)
	NOP	(4)
	NOP	(4)
	LD A,\$FE	(7) Set RS232 'RXD' transmit line to 0. (Keep KEYPAD 'CTS' output line low to prevent the keypad resetting)
	OUT (C),A	(11) Send out the STOP bit.
L1206:	LD E,\$06	(7) Delay for 101 T-states (28.5us).
	DEC E	(4)
	JR NZ,L1206	(12/7)
	RET	(10)

## CASSETTE / RAM DISK COMMAND ROUTINES — PART 1

### SAVE Routine

L120A:	LD HL,FLAGS3 SET 5,(HL) JR L1224	\$5B66. Indicate SAVE.
--------	--	---------------------------

### LOAD Routine

L1211:	LD HL,FLAGS3 SET 4,(HL) JR L1224	\$5B66. Indicate LOAD.
--------	--	---------------------------

### VERIFY Routine

L1218:	LD HL,FLAGS3 SET 7,(HL) JR L1224	\$5B66. Indicate VERIFY.
--------	--	-----------------------------

### MERGE Routine

L121F:	LD HL,FLAGS3 SET 6,(HL)	\$5B66. Indicate MERGE.
L1224:	LD HL,FLAGS3 RES 3,(HL) RST 18H CP 'I' JP NZ,L13DD	\$5B66. Indicate using cassette. Get current character. \$21. 'I' Jump ahead to handle cassette command.

## RAM disk operation

	LD HL,FLAGS3	\$5B66.
	SET 3,(HL)	Indicate using RAM disk.
	RST 20H	Move on to next character.
	JP L13DD	Jump ahead to handle RAM disk command.
L1238:	CALL L05CB	Produce error report.
	DEFB \$0B	"C Nonsense in BASIC"

**RAM Disk Command Handling**

The information relating to the file is copied into memory in \$5B66 (FLAGS3) to ensure that it is available once other RAM banks are switched in. This code is very similar to that in the ZX Interface 1 ROM at \$08F6.

Entry: HL=Start address.  
IX=File header descriptor.

L123C:	LD (HD_0D),HL	\$5B74. Save start address.
	LD A,(IX+\$00)	Transfer header file information
	LD (HD_00),A	\$5B71. from IX to HD_00 onwards.
	LD L,(IX+\$0B)	
	LD H,(IX+\$0C)	
	LD (HD_0B),HL	\$5B72.
	LD L,(IX+\$0D)	
	LD H,(IX+\$0E)	
	LD (HD_11),HL	\$5B78.
	LD L,(IX+\$0F)	
	LD H,(IX+\$10)	
	LD (HD_0F),HL	\$5B76.

A copy of the header information has now been copied from IX+\$00 onwards to HD\_00 onwards

OR A	Test file type.
JR Z,L126D	Jump ahead for a program file.
CP \$03	
JR Z,L126D	Jump ahead for a CODE/SCREEN\$ file.

## An array type

	LD A,(IX+\$0E)	
	LD (HD_0F),A	\$5B76. Store array name.
L126D:	PUSH IX	IX points to file header.
	POP HL	Retrieve into HL.
	INC HL	HL points to filename.
	LD DE,N_STR1	\$5B67.
	LD BC,\$000A	
	LDIR	Copy the filename.
	LD HL,FLAGS3	\$5B66.
	BIT 5,(HL)	SAVE operation?
	JP NZ,L1BCC	Jump ahead if SAVE.

## Load / Verify or Merge

	LD HL,HD_00	\$5B71.
	LD DE,SC_00	\$5B7A.
	LD BC,\$0007	
	LDIR	Transfer requested details from HD_00 onwards into SC_00 onwards.
	CALL L1C4D	Find and load requested file header into HD_00 (\$5B71).

The file exists else the call above would have produced an error "h file does not exist"

LD A,(SC_00)	\$5B7A. Requested file type.
LD B,A	
LD A,(HD_00)	\$5B71. Loaded file type.

## SPECTRUM +2 ROM o DISASSEMBLY

	CP B	
	JR NZ,L129F	Error 'b' if file types do not match.
	CP \$03	Is it a CODE file type?
	JR Z,L12AF	Jump ahead to avoid MERGE program/array check.
	JR C,L12A3	Only file types 0, 1 and 2 are OK.
L129F:	CALL L05CB	Produce error report.
	DEFB \$1D	"b Wrong file type"
L12A3:	LD A,(FLAGS3)	\$5B66.
	BIT 6,A	Is it a MERGE program/array operation?
	JR NZ,L12E4	Jump ahead if so.
	BIT 7,A	Is it a VERIFY program/array operation?
	JP Z,L12FA	Jump ahead if LOAD.

Either a verify program/array or a load/verify CODE/SCREEN\$ type file

L12AF:	LD A,(FLAGS3)	\$5B66.
	BIT 6,A	MERGE operation?
	JR Z,L12BA	Jump ahead if VERIFY.

Cannot merge CODE/SCREEN\$

CALL L05CB	Produce error report.
DEFB \$1C	"a MERGE error"

## RAM Disk VERIFY! Routine

L12BA:	LD HL,(SC_0B)	\$5B7B. Length requested.
	LD DE,(HD_0B)	\$5B72. File length.
	LD A,H	
	OR L	
	JR Z,L12CD	Jump ahead if requested length is 0, i.e. not specified.
	SBC HL,DE	Is file length <= requested length?
	JR NC,L12CD	Jump ahead if so; requested length is OK.

File was smaller than requested

	CALL L05CB	Produce error report.
	DEFB \$1E	"c CODE error"
L12CD:	LD HL,(SC_0D)	\$5B7D. Fetch start address.
	LD A,H	
	OR L	Is length 0, i.e. not provided?
	JR NZ,L12D7	Jump ahead if start address was provided.
	LD HL,(HD_0D)	\$5B74. Not provided so use file's start address.
L12D7:	LD A,(HD_00)	\$5B71. File type.
	AND A	Is it a program?
	JR NZ,L12E0	Jump ahead if not.
	LD HL,(\$5C53)	PROG. Set start address as start of program area.
L12E0:	CALL L139D	Load DE bytes at address pointed to by HL. [The Spectrum 128 manual states that the VERIFY keyword is not used with the RAM disk yet it clearly is, although verifying a RAM disk file simply loads it in just as LOAD would do. To support verifying, the routine at \$1E56 (ROM 0) which loads blocks of data would need to be able to load or verify a block. The success status would then need to be propagated back to here via routines at \$139D (ROM 0), \$1C6A (ROM 0) and \$1E56 (ROM 0)]
	RET	[Could have saved 1 byte by using JP \$139D (ROM 0), although could have saved a lot more by not supporting the VERIFY keyword at all]

## RAM Disk MERGE! Routine

L12E4:	LD BC,(HD_0B)	\$5B72. File length.
	PUSH BC	Save the length.
	INC BC	Increment for terminator \$80 (added later).

## SPECTRUM +2 ROM 0 DISASSEMBLY

RST 28H DEFW BC_SPACES LD (HL), \$80 EX DE, HL POP DE PUSH HL CALL L139D POP HL RST 28H DEFW ME_CONTRL+\$0018 RET	\$0030. Create room in the workspace for the file. Insert terminator. HL=Start address. DE=File length. Save start address. Load DE bytes to address pointed to by HL. Retrieve start address.  \$08CE. Delegate actual merge handling to ROM 1.
---	--

### RAM Disk LOAD! Routine

L12FA: LD DE, (HD_0B) LD HL, (SC_0D) PUSH HL LD A, H OR L JR NZ, L130C	\$5B72. File length. \$5B7D. Requested start address. Save requested start address.  Was start address specified? (0 if not). Jump ahead if start address specified.
---	---

Start address was not specified

INC DE INC DE INC DE EX DE, HL JR L1315	Allow for variable overhead.  HL=File Length+3. Jump ahead to test if there is room.
---	---

A start address was specified

L130C: LD HL, (SC_0B) EX DE, HL SCF SBC HL, DE JR C, L131E	\$5B7B. Requested length. DE=Requested length. HL=File length.  File length-Requested Length-1 Jump if file is smaller than requested.
--	--

Test if there is room since file is bigger than requested

L1315: LD DE, \$0005 ADD HL, DE LD B, H LD C, L RST 28H DEFW TEST_ROOM	 Space required in BC.  \$1F05. Will automatically produce error '4' if out of memory.
---	---

Test file type

L131E: POP HL LD A, (HD_00) L1322: AND A JR Z, L1354	Requested start address. \$5B71. Get requested file type. Test file type. Jump if program file type.
---	---

Array type

LD A, H OR L JR Z, L1334	Was start address of existing array specified? Jump ahead if not.
--------------------------------	--

Start address of existing array was specified

DEC HL

## SPECTRUM +2 ROM o DISASSEMBLY

```
LD B,(HL)
DEC HL
LD C,(HL)
DEC HL
INC BC
INC BC
INC BC
RST 28H
DEFW RECLAIM_2
```

Fetch array length.

Allow for variable header.

\$19E8. Delete old array.

Insert new array entry into variables area

```
L1334:    LD HL,($5C59)
          DEC HL
          LD BC,(HD_0B)
          PUSH BC
          INC BC
          INC BC
          INC BC
          LD A,(SC_0F)
          PUSH AF
          RST 28H
          DEFW MAKE_ROOM
          INC HL
          POP AF
          LD (HL),A
          POP DE
          INC HL
          LD (HL),E
          INC HL
          LD (HL),D
          INC HL
L1350:    CALL L139D
          RET
```

E\_LINE.

Point to end

\$5B72. Array length.

Save array length.

Allow for variable header.

\$5B7F. Get array name.

Save array name.

\$1655. Create room for new array.

Store array name.

Store array length.

Load DE bytes to address pointed to by HL.

[Could have saved 1 byte by using JP \$139D (ROM 0)]

Program type

```
L1354:    LD HL,FLAGS3
          RES 1,(HL)
          LD DE,($5C53)
          LD HL,($5C59)
          DEC HL
          RST 28H
          DEFW RECLAIM
          LD BC,(HD_0B)
          LD HL,($5C53)
          RST 28H
          DEFW MAKE_ROOM
          INC HL
          LD BC,(HD_0F)
          ADD HL,BC
          LD ($5C4B),HL
          LD A,(HD_11+1)
          LD H,A
          AND $C0
          JR NZ,L138F
          LD A,(HD_11)
          LD L,A
          LD ($5C42),HL
          LD (IY+$0A),$00
          LD HL,FLAGS3
          SET 1,(HL)
L138F:    LD HL,($5C53)
          LD DE,(HD_0B)
          DEC HL
          LD ($5C57),HL
```

\$5B66.

Signal do not auto-run BASIC program.

PROG. Address of start of BASIC program.

E\_LINE. Address of end of program area.

Point before terminator.

\$19E5. Delete current BASIC program.

\$5B72. Fetch file length.

PROG. Address of start of BASIC program.

\$1655. Create room for the file.

Allow for terminator.

\$5B76. Length of variables.

Determine new address of variables.

VARs.

\$5B79. Fetch high byte of auto-run line number.

If holds \$80 then no auto-run line number specified.

\$5B78. Low byte of auto-run line number.

NEWPPC. Set line number to run.

NSPPC. Statement 0.

\$5B66.

Signal auto-run BASIC program.

PROG. Address of start of BASIC program.

\$5B72. Program length.

NXTLIN. Set the address of next line to the end of the program.

INC HL  
JR L1350

Jump back to load program bytes.

## RAM Disk Load Bytes

Make a check that the requested length is not zero before proceeding to perform the LOAD, MERGE or VERIFY. Note that VERIFY simply performs a LOAD.

Entry: HL=Destination address.  
DE=Length.  
IX=Address of catalogue entry.  
HD\_00-HD\_11 holds file header information.

L139D:	LD A,D OR E RET Z CALL L1C6A RET	Return if length is zero. Load bytes [Could have used JP \$1C6A (ROM 0) to save 1 byte]
--------	--	---

## Get Expression from BASIC Line

Returns in BC.

L13A4:	RST 28H DEFW EXPT_EXP BIT 7,(IY+\$01) RET Z PUSH AF RST 28H DEFW STK_FETCH POP AF RET	Expect an expression on the BASIC line. \$1C8C. Return early if syntax checking.  Get the item off the calculator stack  \$2BF1.
--------	---	--

## Check Filename and Copy

Called to check a filename for validity and to copy it into N\_STR1 (\$5B67).

L13B2:	RST 20H CALL L13A4 RET Z PUSH AF LD A,C OR B JR Z,L13D9 LD HL,\$000A SBC HL,BC JR C,L13D9 PUSH DE PUSH BC LD HL,N_STR1 LD B,\$0A LD A,\$20	Advance the pointer into the BASIC line. Get expression from BASIC line. Return if syntax checking. [No need to save AF - see comment below] Check for zero length.  Jump if so to produce error report "f Invalid name". Check for length greater than 10.  Jump if so to produce error report "f Invalid name". Save the filename start address. Save the filename length. \$5B67. HL points to filename buffer.
L13CC:	LD (HL),A INC HL DJNZ L13CC POP BC POP HL LD DE,N_STR1 LDIR POP AF RET	Fill it with 10 spaces.   Restore filename length. Restore filename start address. \$5B67. DE points to where to store the filename. Perform the copy. [No need to have saved AF as not subsequently used]
L13D9:	CALL L05CB DEFB \$21	Produce error report. "f Invalid name"

## Cassette / RAM Disk Command Handling

Handle SAVE, LOAD, MERGE, VERIFY commands.

Bit 3 of FLAGS3 indicates whether a cassette or RAM disk command.

This code is very similar to that in ROM 1 at \$0605.

L13DD:	RST 28H DEFW EXPT_EXP BIT 7,(IY+\$01) JR Z,L1426 LD BC,\$0011 LD A,(\$5C74) AND A JR Z,L13F1 LD C,\$22	\$1C8C. Pass the parameters of the 'name' to the calculator stack.  Jump ahead if checking syntax. Size of save header, 17 bytes. T_ADDR. Indicates which BASIC command. Is it SAVE? Jump ahead if so. Otherwise need 34d bytes for LOAD, MERGE and VERIFY commands. 17 bytes for the header of the requested file, and 17 bytes for the files tested from tape.
L13F1:	RST 28H DEFW BC_SPACES PUSH DE POP IX LD B,\$0B LD A,\$20	\$0030. Create space in workspace. Get start of the created space into IX.  Clear the filename.
L13FB:	LD (DE),A INC DE DJNZ L13FB LD (IX+\$01),\$FF RST 28H DEFW STK_FETCH LD HL,\$FFF6 DEC BC ADD HL,BC INC BC JR NC,L141F LD A,(\$5C74) AND A JR NZ,L1418 CALL L05CB DEFB \$0E	Set all characters to spaces.   Indicate a null name. The parameters of the name are fetched. \$2BF1. = -10.  Jump ahead if filename length within 10 characters. T_ADDR. Indicates which BASIC command. Is it SAVE? Jump ahead if not since LOAD, MERGE and VERIFY can have null filenames. Produce error report. "F Invalid file name"

Continue to handle the name of the program.

L1418:	LD A,B OR C JR Z,L1426 LD BC,\$000A	Jump forward if the name has a null length. Truncate longer filenames.
--------	--	---

The name is now transferred to the work space (second location onwards)

L141F:	PUSH IX POP HL INC HL EX DE,HL LDIR	Transfer address of the workspace to HL. Step to the second location.  Copy the filename.
--------	---	--

The many different parameters, if any, that follow the command are now considered.

Start by handling 'xxx "name" DATA'.

L1426:	RST 18H CP \$E4 JR NZ,L147E	Get character from BASIC line. Is it 'DATA'? Jump if not DATA.
--------	-----------------------------------	--

'xxx "name" DATA'

LD A,(\$5C74)	T_ADDR. Check the BASIC command.
---------------	----------------------------------

## SPECTRUM +2 ROM o DISASSEMBLY

	CP \$03	Is it MERGE?
	JP Z,L1238	"C Nonsense in BASIC" if so.
	RST 20H	Get next character from BASIC line.
	RST 28H	
	DEFW LOOK_VARS	\$28B2. Look in the variables area for the array.
	JR NC,L144E	Jump if handling an existing array.
	LD HL,\$0000	Signal 'using a new array'.
	BIT 6,(IY+\$01)	FLAGS. Is it a string Variable?
	JR Z,L1444	Jump forward if so.
	SET 7,C	Set bit 7 of the array's name.
L1444:	LD A,(\$5C74)	T_ADDR.
	DEC A	Give an error if trying to
	JR Z,L1463	SAVE or VERIFY a new array.
	CALL L05CB	Produce error report.
	DEFB \$01	"2 Variable not found"

Continue with the handling of an existing array

L144E:	JP NZ,L1238	Jump if not an array to produce "C Nonsense in BASIC".
	BIT 7,(IY+\$01)	FLAGS.
	JR Z,L1470	Jump forward if checking syntax.
	LD C,(HL)	
	INC HL	Point to the 'low length' of the variable.
	LD A,(HL)	The low length byte goes into
	LD (IX+\$0B),A	the work space.
	INC HL	
	LD A,(HL)	The high length byte goes into
	LD (IX+\$0C),A	the work space.
	INC HL	Step past the length bytes.

The next part is common to both 'old' and 'new' arrays

L1463:	LD (IX+\$0E),C	Copy the array's name.
	LD A,\$01	Assume an array of numbers - Code \$01.
	BIT 6,C	
	JR Z,L146D	Jump if it is so.
	INC A	Indicate it is an array of characters - Code \$02.
L146D:	LD (IX+\$00),A	Save the 'type' in the first location of the header area.

The last part of the statement is examined before joining the other pathways

L1470:	EX DE,HL	Save the pointer in DE.
	RST 20H	
	CP ')'	\$29. Is the next character a ')'?
	JR NZ,L144E	Give report C if it is not.
	RST 20H	Advance to next character.
	CALL L18C0	Move on to the next statement if checking syntax.
	EX DE,HL	Return the pointer to the HL. (The pointer indicates the start of an existing array's contents).
	JP L1538	Jump forward.

Now Consider 'SCREEN\$'

L147E:	CP \$AA	Is the present code the token 'SCREEN\$'?
	JR NZ,L14A1	Jump ahead if not.

'xxx "name" SCREEN\$'

	LD A,(\$5C74)	T_ADDR_lo. Check the BASIC command.
	CP \$03	Is it MERGE?
	JP Z,L1238	Jump to "C Nonsense in BASIC" if so since it is not possible to have 'MERGE name SCREEN\$'.
	RST 20H	Advance pointer into BASIC line.
	CALL L18C0	Move on to the next statement if checking syntax.
	LD (IX+\$0B),\$00	Length of the block.

## SPECTRUM +2 ROM o DISASSEMBLY

```
LD (IX+$0C),$1B
LD HL,$4000
LD (IX+$0D),L
LD (IX+$0E),H
JR L14EE
```

The display area and the attribute area occupy \$1800 locations.  
Start of the block, beginning of the display file \$4000.

Store in the workspace.  
Jump forward.

Now consider 'CODE'

```
L14A1:      CP $AF
           JR NZ,L14F4
```

Is the present code the token 'CODE'?  
Jump ahead if not.

'xxx "name" CODE'

```
LD A,($5C74)
CP $03
JP Z,L1238

RST 20H
RST 28H
DEFW PR_ST_END
JR NZ,L14BF
LD A,($5C74)
AND A
JP Z,L1238
RST 28H
DEFW USE_ZERO
JR L14CE
```

T\_ADDR\_lo. Check the BASIC command.  
Is it MERGE?  
Jump to "C Nonsense in BASIC" if so since it is not possible to have 'MERGE name CODE'.  
Advance pointer into BASIC line.  
\$2048.  
Jump forward if the statement has not finished  
T\_ADDR\_lo.  
It is not possible to have 'SAVE name CODE' by itself.  
Jump if so to produce "C Nonsense in BASIC".  
\$1CE6. Put a zero on the calculator stack - for the 'start'.  
Jump forward.

Look for a 'starting address'

```
L14BF:      RST 28H
           DEFW EXPT_1NUM
           RST 18H
           CP ','
           JR Z,L14D3
           LD A,($5C74)
           AND A
           JP Z,L1238
L14CE:      RST 28H
           DEFW USE_ZERO
           JR L14D7
```

\$1C82. Fetch the first number.  
\$2C. Is the present character a ','?  
Jump if it is - the number was a 'starting address'  
T\_ADDR\_lo.  
Refuse 'SAVE name CODE' that does not have a 'start' and a 'length'.  
Jump if so to produce "C Nonsense in BASIC".  
\$1CE6. Put a zero on the calculator stack - for the 'length'.  
Jump forward.

Fetch the 'length' as it was specified

```
L14D3:      RST 20H
           RST 28H
           DEFW EXPT_1NUM
```

Advance to next character.  
\$1C82. Fetch the 'length'.

The parameters are now stored in the header area of the work space

```
L14D7:      CALL L18C0
           RST 28H
           DEFW FIND_INT2
           LD (IX+$0B),C
           LD (IX+$0C),B
           RST 28H
           DEFW FIND_INT2
           LD (IX+$0D),C
           LD (IX+$0E),B
           LD H,B
           LD L,C
```

But move on to the next statement now if checking syntax.  
\$1E99. Compress the 'length' into BC.  
Store the length of the CODE block.  
\$1E99. Compress the 'starting address' into BC.  
Store the start address of the CODE block.  
Transfer start address pointer to HL.

'SCREEN\$' and 'CODE' are both of type 3

## SPECTRUM +2 ROM 0 DISASSEMBLY

L14EE:	LD (IX+\$00),\$03 JR L1538	Store file type = \$03 (CODE). Rejoin the other pathways.
<p>'xxx "name" / 'SAVE "name" LINE' Now consider 'LINE' and 'no further parameters'</p>		
L14F4:	CP \$CA JR Z,L1501 CALL L18C0 LD (IX+\$0E),\$80 JR L1518	Is the present code the token 'LINE'? Jump ahead if so. Move on to the next statement if checking syntax. Indicate no LINE number. Jump forward.

Fetch the 'line number' that must follow 'LINE'

L1501:	LD A,(\$5C74) AND A JP NZ,L1238 RST 20H RST 28H DEFW EXPT_1NUM CALL L18C0 RST 28H DEFW FIND_INT2 LD (IX+\$0D),C LD (IX+\$0E),B	T_ADDR_lo. Only allow 'SAVE name LINE number'. Is it SAVE? Produce "C Nonsense in BASIC" if not. Advance pointer into BASIC line. Get LINE number onto calculator stack \$1C82. Pass the number to the calculator stack. Move on to the next statement if checking syntax. Retrieve LINE number from calculator stack \$1E99. Compress the 'line number' into BC. Store the LINE number.
--------	--	---

'LINE' and 'no further parameters' are both of type 0

L1518:	LD (IX+\$00),\$00 LD HL,(\$5C59) LD DE,(\$5C53) SCF SBC HL,DE LD (IX+\$0B),L LD (IX+\$0C),H LD HL,(\$5C4B) SBC HL,DE LD (IX+\$0F),L LD (IX+\$10),H EX DE,HL	Store file type = \$00 (program). E_LINE. The pointer to the end of the variables area. PROG. The pointer to the start of the BASIC program.  Perform the subtraction to find the length of the 'program + variables'.  Store the length. VARS. Repeat the operation but this time storing the length of the 'program' only.  Transfer pointer to HL.
--------	--	--

In all cases the header information has now been prepared:

- The location 'IX+00' holds the type number.
  - Locations 'IX+01 to IX+0A' holds the name (\$FF in 'IX+01' if null).
  - Locations 'IX+0B & IX+0C' hold the number of bytes that are to be found in the 'data block'.
  - Locations 'IX+0D to IX+10' hold a variety of parameters whose exact interpretation depends on the 'type'.
- The routine continues with the first task being to separate SAVE from LOAD, VERIFY and MERGE.

L1538:	LD A,(FLAGS3) BIT 3,A JP NZ,L123C LD A,(\$5C74) AND A JR NZ,L154A RST 28H DEFW SA_CONTROL RET	\$5B66. Using RAM disk? Jump if the operation is on the RAM disk. T_ADDR_lo. Get the BASIC command. Is it SAVE? Jump ahead if not.  \$0970. Run the save routine in ROM 1.
--------	---	---

In the case of a LOAD, VERIFY or MERGE command the first seventeen bytes of the 'header area' in the work space hold the prepared information, as detailed above and it is now time to fetch a 'header' from the tape.

L154A:	RST 28H DEFW SA_ALL+\$0007 RET	\$0761. Run the load/merge/verify routine in ROM 1.
--------	--------------------------------------	---

## EDITOR ROUTINES — PART 1

### Relist the BASIC Program from the Current Line

This routine lists the BASIC program from the current line number. It initially shows the last line displayed but rows may subsequently be scrolled up until the required BASIC line has been found. The structure of the ROM program only supports listing BASIC lines that are 20 rows or less; larger lines are shown truncated to 20 rows.

L154E:	LD HL,\$EEF5 RES 0,(HL) SET 1,(HL)	Flags. Signal this is not the current line. Signal not yet located the current line.
--------	--	--

A loop is entered to display a screenful of program listing. If the current line number is not found in the lines displayed then all lines are scrolled up and the listing reproduced. This procedure repeats until the current line number has been found and displayed.

L1555:	LD HL,(\$5C49) LD A,H OR L JR NZ,L155F LD (\$EC06),HL	E_PPC. Fetch current line number.  Is there a currently selected line? Jump ahead if so. Set to \$0000 to indicate no editable characters before the cursor.
L155F:	LD A,(\$F9DB)  PUSH AF LD HL,(\$FC9A) CALL L3370 LD (\$F9D7),HL CALL L3248 CALL L30FC POP AF	Fetch the number of rows of the BASIC line that are in the Above-Screen Line Edit Buffer, i.e. that are off the top of the screen. Line number of the BASIC line at the top of the screen (or 0 for the first line). Find closest line number (or \$0000 if no subsequent line exists). Store the line number of the BASIC line being edited in the buffer. Set default Above-Screen Line Edit Buffer settings. Set default Below-Screen Line Edit Buffer settings. A=Number of rows of the BASIC line that are in the Above-Screen Line Edit Buffer.
L1573:	OR A JR Z,L1582	Are there any rows off the top of the screen? Jump ahead if not.

The current settings indicate that the top BASIC line straggles into the Above-Screen Line Edit Buffer. It is therefore necessary to insert the current BASIC line into the Below-Screen Line Edit Buffer and then shift the appropriate number of rows into the Above-Screen Line Edit Buffer.

PUSH AF CALL L3105 EX DE,HL CALL L3290 POP AF DEC A JR L1573	Save the number of rows off the top of the screen. Copy a BASIC line from the program area into the Below-Screen Line Edit Buffer. DE=Address of the Below-Screen Line Edit Buffer. Shift up a row into the Above-Screen Line Edit Buffer. Retrieve the number of rows off the top of the screen. Decrement the number of rows. Jump back to shift up another row if required.
--	--

Either there the top BASIC line does not straggle off the top of the the screen or the appropriate number of rows have been copied into the Above-Screen Line Edit Buffer. In the latter case, the Below-Screen Line Edit Buffer contains the remaining rows of the BASIC line and which be copied into the top of the Screen Line Edit Buffer.

L1582:	LD C,\$00 CALL L30DA LD B,C LD A,(\$EC15) LD C,A PUSH BC PUSH DE	C=Row 0. DE=Start address in Screen Line Edit Buffer of the first row, as specified in C. B=Row 0. The number of editing rows on screen. C=Number of editing rows on screen. B=Row number, C=Number of editing rows on screen. DE=Start address in Screen Line Edit Buffer of the first row.
--------	--	--

Enter a loop to copy BASIC line rows into the Screen Line Edit Buffer. The Below-Screen Line Edit Buffer is used as a temporary store for holding each BASIC line as it is copied into the Screen Line Edit Buffer. If the top BASIC line straggles above the screen then this loop is entered with the remains of the line already in the Below-Screen Line Edit Buffer.

L158E:	CALL L3105  LD A,(\$EEF5)	Shift up all rows of the BASIC line in the Below-Screen Line Edit Buffer, or if empty then copy a BASIC line from the program area into it. If no BASIC line available then empty the first row of the Below-Screen Line Edit Buffer. Listing flags.
--------	---------------------------------	---

## SPECTRUM +2 ROM 0 DISASSEMBLY

BIT 1,A	Has the current line been previously found?
JR Z,L15B5	Jump if so.

The current line has not yet been found so examine the current row in case it is the current line

PUSH DE	DE=Start address in Screen Line Edit Buffer of the current row.
PUSH HL	HL=Address of the first row in the Below-Screen Line Edit Buffer.
LD DE,\$0020	
ADD HL,DE	Point to the flag byte for the first row.
BIT 0,(HL)	Is it the first row of a BASIC line?
JR Z,L15B3	Jump if not.

The Below-Screen Line Edit Buffer contains a complete BASIC line so determine whether this is the current line

INC HL	
LD D,(HL)	Get line number into DE.
INC HL	
LD E,(HL)	
OR A	
LD HL,(\$5C49)	E_PPC. Current line number.
SBC HL,DE	
JR NZ,L15B3	Jump ahead unless this is the current line.
LD HL,\$EEF5	
SET 0,(HL)	Signal this is the current line.
L15B3: POP HL	HL=Address of the current row in the Below-Screen Line Edit Buffer.
POP DE	DE=Start address in Screen Line Edit Buffer of the current row.

Copy the row of the BASIC line from the Below-Screen Line Edit Buffer into the Screen Line Edit Buffer

L15B5: PUSH BC	B=Row number, C=Number of editing rows on screen.
PUSH HL	HL=Address of the current row in the Below-Screen Line Edit Buffer.
LD BC,\$0023	
LDIR	Copy the first row of the BASIC line in the Below-Screen Line Edit Buffer into the next row of the Screen Line Edit Buffer.
POP HL	HL=Address of the current row in the Below-Screen Line Edit Buffer.
POP BC	B=Row number, C=Number of editing rows on screen.
PUSH DE	DE=Start address in Screen Line Edit Buffer of the next row.
PUSH BC	B=Row number, C=Number of editing rows on screen.
EX DE,HL	DE=Address of the current row in the Below-Screen Line Edit Buffer.
LD HL,\$EEF5	Flags.
BIT 0,(HL)	Is this the current line?
JR Z,L15F2	Jump if not.

This is the current line so scan across the BASIC line to locate the cursor column position

LD B,\$00	Column 0.
L15CA: LD HL,(\$EC06)	HL=Count of the number of editable characters in the BASIC line up to the cursor within the Screen Line Edit Buffer.
LD A,H	
OR L	Are there any editable characters in this row prior to the cursor?
JR Z,L15DF	Jump if there are none, i.e. cursor at start of the row.

There are editable characters on this row prior to the cursor **[BUG - Entering ' 10 REM' or '0010 REM' will insert the line into the program area but instead of placing the cursor on the following row it is placed after the following BASIC line, or if the line inserted was the last in the program then the cursor is placed on row 20. The bug occurs due to the leading spaces or zeros, and hence will apply to every BASIC command. When the line is inserted into the Screen Line Edit Buffer, the leading spaces are discarded and hence the line length is shorter than that typed in. However, it is the typed in line length that is used when parsing the BASIC line in the Screen Line Edit Buffer and as a result this causes an attempt to find the remaining characters on the following row of the Screen Line Edit Buffer. If another BASIC line is on the following Screen Line Edit Buffer row then the search completes and the cursor is placed on the row after this BASIC line. If there is not a BASIC line on the following row then the search continues on the next row. Since this will also be empty, the search advances onto the next row, and then the next, and so on until row 20 is reached. To fix the bug, the typed in character count until the cursor (held in \$EC06) ideally needs to be adjusted to match the actual number of characters stored in the Screen Line Edit Buffer. However, this is not a trivial change to implement. A simpler solution to fix the bug is to intercept when a move to the next row is made and to determine whether the BASIC line actually continues on this row. Credit: Paul Farrow] [To fix the bug, the POP HL and JR NC,\$15EA (ROM 0) instructions following the call to \$2E67 (ROM 0) should be replaced with the following. Credit: Paul Farrow.**

## SPECTRUM +2 ROM 0 DISASSEMBLY

	<i>PUSH DE</i>	<i>DE=Address of the start of the row of the BASIC line in the Screen Line Edit Buffer.</i>
	<i>PUSH AF</i>	<i>Save the flags.</i>
	<i>LD HL,\$0020</i>	
	<i>ADD HL,DE</i>	
	<i>EX DE,HL</i>	<i>DE=Address of the flag byte for the row in the Screen Line Edit Buffer.</i>
	<i>POP AF</i>	<i>Restore the flags.</i>
	<i>JR C,CHAR_FOUND</i>	<i>Jump if editable column found.</i>
	<i>LD A,(DE)</i>	<i>Fetch the flag byte.</i>
	<i>BIT 1,A</i>	<i>Does the BASIC line span onto the next row?</i>
	<i>JR NZ,SPANS_ROW</i>	<i>Jump if it does.</i>
	<i>POP DE</i>	<i>DE=Address of the start of the BASIC row in the Screen Line Edit Buffer.</i>
	<i>POP HL</i>	
	<i>LD HL,\$0000</i>	<i>Signal no editable characters left on the row.</i>
	<i>LD (\$EC06),HL</i>	
	<i>JP \$15DF (ROM 0)</i>	<i>Jump since all characters on the row have been scanned through.</i>
<i>SPANS_ROW</i>		
	<i>POP DE</i>	<i>DE=Address of the start of the BASIC row in the Screen Line Edit Buffer.</i>
	<i>POP HL</i>	
	<i>JP \$15EA (ROM 0)</i>	<i>Jump if no editable columns left on the row.</i>
<i>CHAR_FOUND</i>		
	<i>POP DE</i>	<i>DE=Address of the start of the BASIC row in the Screen Line Edit Buffer.</i>
	<i>POP HL</i>	<i>]</i>

<i>PUSH HL</i>	
<i>CALL L2E67</i>	Find editable position on this row from the previous column to the right, returning column number in B.
<i>POP HL</i>	
<i>JR NC,L15EA</i>	Jump if no editable character found on this row, i.e. there must be more characters on the next row.

An editable character was found to the right on the current row

<i>DEC HL</i>	Decrement the count of characters prior to the cursor.
<i>INC B</i>	Advance to next column.
<i>LD (\$EC06),HL</i>	Update the count of the number of editable characters up to the cursor.
<i>JR L15CA</i>	Jump back to test next column.

Column position of cursor located, find the closest editable character

<i>L15DF:</i>	<i>CALL L2E67</i>	Find editable position on this row from the previous column to the right, returning column number in B.
	<i>CALL NC,L2E89</i>	If no editable character found then find editable position to the left, returning column number in B.
	<i>LD HL,\$EEF5</i>	Flags.
	<i>LD (HL),\$00</i>	Signal 'not the current line', 'current line has previously been found' and 'update display file enabled'.

Store the current cursor position

<i>L15EA:</i>	<i>LD A,B</i>	A=Column number. This will be the preferred column number.
	<i>POP BC</i>	B=Row number, C=Number of editing rows on screen.
	<i>PUSH BC</i>	
	<i>LD C,B</i>	C=Row number.
	<i>LD B,A</i>	B=Column number.
	<i>CALL L2A37</i>	Store this as the current cursor editing position.

Move to next row

<i>L15F2:</i>	<i>POP BC</i>	B=Row number, C=Number of editing rows on screen.
	<i>POP DE</i>	DE=Start address in Screen Line Edit Buffer of the next row.
	<i>LD A,C</i>	A=Number of editing rows on screen.
	<i>INC B</i>	Next row.
	<i>CP B</i>	Reached the bottom screen row?
	<i>JR NC,L158E</i>	Jump back if not to display the next row.

The bottom screen row has been exceeded

```
LD A,($EEF5)
BIT 1,A
JR Z,L1621
```

Listing flags.  
Has the current line been previously found?  
Jump if so.

Current line has not yet been found

```
BIT 0,A
JR NZ,L1621
```

Is this the current line?  
Jump if so.

This is not the current line

```
LD HL,($5C49)
LD A,H
OR L
JR Z,L1613
LD ($FC9A),HL
CALL L3248

JR L161C
```

E\_PPC. Current line number.  
  
Jump if there is no current line number.  
Store it as the line number at top of the screen.  
Set default Above-Screen Line Edit Buffer settings to clear the count of the number of rows it contains.  
Jump forward.

There is no current line number

```
L1613:    LD ($FC9A),HL
          CALL L3378
```

Set the line number at top of the screen to \$0000, i.e. first available.  
Create line number representation in the Keyword Construction Buffer of the next BASIC line.

```
L161C:    LD ($5C49),HL
          POP DE
          POP BC
          JP L1555
```

E\_PPC. Current line number is the first in the BASIC program.  
DE=Start address in Screen Line Edit Buffer of the first row.  
B=Row number, C=Number of editing rows on screen.  
Jump back to continue listing the program until the current line is found.

The bottom line is the current line

```
L1621:    POP DE
          POP BC
          CP A
```

DE=Start address in Screen Line Edit Buffer of the first row.  
B=Row number, C=Number of editing rows on screen.  
Set the zero flag if current line has yet to be found, hence signal do not update cursor position settings.

## Print All Screen Line Edit Buffer Rows to the Display File

Print all rows of the edit buffer to the display file, and updating the cursor position settings if required.

Entry: Zero flag reset if update of cursor position settings required.

B=Row number.

C=Number of editing rows on screen.

```
L1624:    PUSH AF
          LD A,C
          LD C,B
          CALL L30DA
          EX DE,HL
L162B:    PUSH AF
          CALL L362A
          POP AF
          LD DE,$0023
          ADD HL,DE
L1634:    INC C
          CP C
          JR NC,L162B
```

Save the zero flag.  
Save the number of editing rows on screen.  
C=Row number.  
DE=Start address in Screen Line Edit Buffer of row held in C and transfer into HL.  
A=Number of editing rows on screen.  
Print a row of the edit buffer to the screen.

Point to the start of the next row.  
Advance to the next row.  
All rows printed?  
Jump back if not to print next row.

All rows printed

```
POP AF
```

Retrieve the zero flag.

RET Z

Return if 'not the current line' and 'current line has previously been found'.

Find the new cursor column position

L163D:	CALL L2A2D	Get current cursor position (C=row, B=column, A=preferred column).
	CALL L2B9E	Find next Screen Line Edit Buffer editable position to right, moving to next row if necessary. Returns column number in B.
	LD HL,(\$EC06)	Fetch the number of editable characters on this row prior to the cursor.
	DEC HL	Decrement the count.
	LD A,H	Are there any characters?
	OR L	
	LD (\$EC06),HL	Store the new count.
	JR NZ,L163D	Jump if there are some characters prior to the cursor.
	JP L2A37	Store cursor editing position, with preferred column of 0.
	RET	[Redundant byte]

## Clear Editing Display

L164F:	LD B,\$00	Top row of editing area.
	LD A,(\$EC15)	The number of editing rows on screen.
	LD D,A	D=Number of rows in editing area.
	JP L3B7F	Clear specified display rows.

## Shift All Edit Buffer Rows Up and Update Display File if Required

This routine shifts all edit buffer rows up, updating the display file if required.

Entry: HL=Address of the 'Bottom Row Scroll Threshold' within the editing area information.

Exit : Carry flag set if edit buffer rows were shifted.

L1658:	LD B,\$00	Row number to start shifting from.
	PUSH HL	Save the address of the 'Bottom Row Scroll Threshold' within the editing area information.

Attempt to shift a row into the Above-Screen Line Edit Buffer

LD C,B	Find the address of row 0.
CALL L30DA	DE=Start address in Screen Line Edit Buffer of the row specified in C.
CALL L3290	Attempt to shift the top row of the Screen Line Edit Buffer into the Above-Screen Line Edit Buffer.
POP HL	Retrieve the address of the 'Bottom Row Scroll Threshold' within the editing area information.
RET NC	Return if the Above-Screen Line Edit Buffer is full, i.e. no edit buffer rows shifted.

A change to the number of rows in the Above-Screen Line Edit Buffer occurred

CALL L3105	Shift up rows of the BASIC line in Below-Screen Line Edit Buffer, inserting the next line BASIC line if the buffer becomes empty. Returns with HL holding the address of the first row in the Below-Screen Line Edit Buffer.
------------	--

Shift All Screen Line Edit Buffer Rows Up and Update Display File if Required

L1667:	PUSH BC	B=Row counter.
	PUSH HL	HL=Address of first row in the Below-Screen Line Edit Buffer.
	LD HL,\$0023	DE=Address of the current row in the Screen Line Edit Buffer.
	ADD HL,DE	HL=Address of the next row in the Screen Line Edit Buffer.
	LD A,(\$EC15)	
	LD C,A	C=Number of editing rows on screen.
	CP B	Any rows to shift?
	JR Z,L1682	Jump if not.

Shift all Screen Line Edit Buffer rows up

## SPECTRUM +2 ROM 0 DISASSEMBLY

L1675:	PUSH BC PUSH BC LD BC,\$0023 LDIR POP BC LD A,C INC B CP B JR NZ,L1675	C=Number of editing rows on screen. C=Number of editing rows on screen. DE=Current Screen Line Edit Buffer row, HL=Next Screen Line Edit Buffer row. Shift one row of the Screen Line Edit Buffer up. C=Number of editing rows on screen. Fetch the number of editing rows on screen. Next row. All rows shifted? Repeat for all edit buffer rows to shift.
--------	--	---

All Screen Line Edit Buffer rows have been shifted up

L1682:	POP BC	C=Number of editing rows on screen, B=Row number, i.e. 0.
L1683:	POP HL	HL=Address of the first row in the Below-Screen Line Edit Buffer.
	CALL L363E	Shift up all edit rows in the display file if updating required.
	LD BC,\$0023	HL=Address of the first row in the Below-Screen Line Edit Buffer, DE=Address of last row in Screen Line Edit Buffer.
	LDIR	Copy the first row of the Below-Screen Line Edit Buffer into the last row of the Screen Line Edit Buffer.
	SCF	Signal that edit buffer rows were shifted.
	POP BC	B=Row counter.
	RET	

### Shift All Edit Buffer Rows Down and Update Display File if Required

This routine shifts all edit buffer rows down, updating the display file if required.

Exit : Carry flag set if edit buffer rows were shifted.

B=Last row number to shift.

Shift all rows in the Above-Screen Line Edit Buffer, shifting in a new BASIC line if applicable

L168E:	LD B,\$00	Last row number to shift.
	CALL L3251	Attempt to shift down the Above-Screen Line Edit Buffer, loading in a new BASIC line if it is empty.
	RET NC	Return if Above-Screen Line Edit Buffer is empty, i.e. no edit buffer rows were shifted.

Entry point from routine at \$2EF9 (ROM 0) to insert a blank row

L1694:	PUSH BC	B=Last row number to shift.
	PUSH HL	HL=Address of next row to use within the Above-Screen Line Edit Buffer.

Shift all rows in the Below-Screen Line Edit Buffer down, shifting in a new BASIC line if applicable

	LD A,(\$EC15)	A=Number of editing rows on screen.
	LD C,A	C=Number of editing rows on screen.
	CALL L30DA	DE=Start address in Screen Line Edit Buffer of the last editing row.
	CALL L3144	Shift down all rows in the Below-Screen Line Edit Buffer, or empty the buffer a row does not straggle off the bottom of the screen.
	JR NC,L16C8	Jump if the Below-Screen Line Edit Buffer is full.
	DEC DE	DE=Address of the last flag byte of the penultimate editing row in the Screen Line Edit Buffer.
	LD HL,\$0023	Length of an edit buffer row.
	ADD HL,DE	HL=Address of the last flag byte of the last editing row in the Screen Line Edit Buffer.
	EX DE,HL	DE=Address of last flag byte of last editing row in Screen Line Edit Buffer,
	PUSH BC	HL=Address of last flag byte of penultimate editing row in Screen Line Edit Buffer.
	LD A,B	C=Number of editing rows on screen, B=Last row number to shift.
	CP C	Any rows to shift?
	JR Z,L16B9	Jump if not.
L16AD:	PUSH BC	C=Row number to shift, B=Last row number to shift.
	LD BC,\$0023	
	LDDR	Copy one row of the Screen Line Edit Buffer down.
	POP BC	C=Number of editing rows on screen, B=Row shift counter.
L16B4:	LD A,B	A=Row shift counter.

```

DEC C
CP C
JR C,L16AD

```

Repeat for all edit buffer rows to shift.

All Screen Line Edit Buffer rows have been shifted down

L16B9:	<pre> EX DE,HL INC DE POP BC POP HL CALL L3652 LD BC,\$0023 LDIR SCF POP BC RET </pre>	<p>HL=Address of last flag byte of first editing row in Screen Line Edit Buffer, DE=Address of byte before start of first editing row in Screen Line Edit Buffer. DE=Start of first row in Screen Line Edit Buffer. C=Number of editing rows on screen, B=Last row number to shift. HL=Address of next row to use within the Above-Screen Line Edit Buffer. Shift down all edit rows in the display file if updating required.</p> <p>Copy the next row of the Above-Screen Line Edit Buffer into the first row of the Screen Line Edit Buffer. Signal Below-Screen Line Edit Buffer is not full. B=Last row number to shift.</p>
--------	--	---

The Below-Screen Line Edit Buffer is full

L16C8:	<pre> POP HL POP BC RET </pre>	<p>Restore registers. B=Last row number to shift.</p>
--------	--------------------------------	---

## Insert Character into Edit Buffer Row, Shifting Row Right

This routine shifts a byte into an edit buffer row, shifting all existing characters right until either the end of the row is reached or the specified end column is reached.

Entry: DE=Start address of an edit buffer row.  
A=Character to shift into left of row.  
B=Column to start shifting at.

Exit : A=Byte shifted out from last column.  
HL=Points byte after row (i.e. flag byte).  
Zero flag set if the character shifted out was a null (\$00).

L16CB:	<pre> PUSH DE LD H,\$00 LD L,B </pre>	<p>Save DE. HL=Start column number.</p>
L16CF:	<pre> ADD HL,DE LD D,A LD A,B </pre>	<p>HL=Address of the starting column. Store the character to shift in. A=Start column number.</p>

Shift all bytes in the row to the right.

L16D2:	<pre> LD E,(HL) LD (HL),D LD D,E INC HL INC A CP \$20 JR C,L16D2 LD A,E CP \$00 POP DE RET </pre>	<p>Fetch a character from the row. Replace it with the character to shift in. Store the old character for use next time. Point to the next column.</p> <p>End of row reached? Jump if not to shift the next character. A=Character that was shifted out. Return with zero flag set if the character was \$00. Restore DE</p>
--------	---	--

## Insert Character into Edit Buffer Row, Shifting Row Left

This routine shifts a byte into an edit buffer row, shifting all existing characters left until either the beginning of the row is reached or the specified end column is reached.

Entry: DE=Start address of an edit buffer row.

A=Character to shift into right of row.

B=Column to stop shifting at.

Exit : A=Byte shifted out.

HL=Points byte before row.

Zero flag set if the character shifted out was a null (\$00).

L16E0:	PUSH DE	Save DE.
	LD HL,\$0020	32 columns.
L16E4:	ADD HL,DE	Point to the flag byte for this row.
	PUSH HL	Save it.
	LD D,A	Store the character to shift in.
	LD A,\$1F	Maximum of 31 shifts.
	JR L16F2	Jump ahead to start shifting.
L16EB:	LD E,(HL)	Fetch a character from the row.
	LD (HL),D	Replace it with the character to shift in.
	LD D,E	Store the old character for use next time.
	CP B	End column reached?
	JR Z,L16F5	Jump if so to exit.
	DEC A	Decrement column counter.
L16F2:	DEC HL	Point back a column.
	JR L16EB	Loop back to shift the next character.
L16F5:	LD A,E	A=Character that was shifted out.
	CP \$00	Return with zero flag set if the character was \$00.
	POP HL	Fetch address of next flag byte for the row.
	POP DE	Restore DE.
	RET	

## BASIC LINE AND COMMAND INTERPRETATION ROUTINES — PART 1

### The Syntax Offset Table

Similar in construction to the table in ROM 1 at \$1A48.

[No instruction fetch at \$1708 hence ZX Interface 1 will not be paged in by this ROM. Credit: Paul Farrow].

L16FB:	DEFB \$B1	DEF FN -> \$17AC (ROM 0)
	DEFB \$C9	CAT -> \$17C5 (ROM 0)
	DEFB \$BC	FORMAT -> \$17B9 (ROM 0)
	DEFB \$BE	MOVE -> \$17BC (ROM 0)
	DEFB \$C3	ERASE -> \$17C2 (ROM 0)
	DEFB \$AF	OPEN # -> \$17AF (ROM 0)
	DEFB \$B4	CLOSE # -> \$17B5 (ROM 0)
	DEFB \$93	MERGE -> \$1795 (ROM 0)
	DEFB \$91	VERIFY -> \$1794 (ROM 0)
	DEFB \$92	BEEP -> \$1796 (ROM 0)
	DEFB \$95	CIRCLE -> \$179A (ROM 0)
	DEFB \$98	INK -> \$179E (ROM 0)
	DEFB \$98	PAPER -> \$179F (ROM 0)
	DEFB \$98	FLASH -> \$17A0 (ROM 0)
	DEFB \$98	BRIGHT -> \$17A1 (ROM 0)
	DEFB \$98	INVERSE -> \$17A2 (ROM 0)
	DEFB \$98	OVER -> \$17A3 (ROM 0)
	DEFB \$98	OUT -> \$17A4 (ROM 0)
	DEFB \$7F	LPRINT -> \$178C (ROM 0)
	DEFB \$81	LLIST -> \$178F (ROM 0)
	DEFB \$2E	STOP -> \$173D (ROM 0)
	DEFB \$6C	READ -> \$177C (ROM 0)
	DEFB \$6E	DATA -> \$177F (ROM 0)
	DEFB \$70	RESTORE -> \$1782 (ROM 0)
	DEFB \$48	NEW -> \$175B (ROM 0)
	DEFB \$94	BORDER -> \$17A8 (ROM 0)
	DEFB \$56	CONTINUE -> \$176B (ROM 0)
	DEFB \$3F	DIM -> \$1755 (ROM 0)
	DEFB \$41	REM -> \$1758 (ROM 0)

DEFB \$2B	FOR -> \$1743 (ROM 0)
DEFB \$17	GO TO -> \$1730 (ROM 0)
DEFB \$1F	GO SUB -> \$1739 (ROM 0)
DEFB \$37	INPUT -> \$1752 (ROM 0)
DEFB \$77	LOAD -> \$1793 (ROM 0)
DEFB \$44	LIST -> \$1761 (ROM 0)
DEFB \$0F	LET -> \$172D (ROM 0)
DEFB \$59	PAUSE -> \$1778 (ROM 0)
DEFB \$2B	NEXT -> \$174B (ROM 0)
DEFB \$43	POKE -> \$1764 (ROM 0)
DEFB \$2D	PRINT -> \$174F (ROM 0)
DEFB \$51	PLOT -> \$1774 (ROM 0)
DEFB \$3A	RUN -> \$175E (ROM 0)
DEFB \$6D	SAVE -> \$1792 (ROM 0)
DEFB \$42	RANDOMIZE -> \$1768 (ROM 0)
DEFB \$0D	IF -> \$1734 (ROM 0)
DEFB \$49	CLS -> \$1771 (ROM 0)
DEFB \$5C	DRAW -> \$1785 (ROM 0)
DEFB \$44	CLEAR -> \$176E (ROM 0)
DEFB \$15	RETURN -> \$1740 (ROM 0)
DEFB \$5D	COPY -> \$1789 (ROM 0)

## The Syntax Parameter Table

Similar to the parameter table in ROM 1 at \$1A7A.

L172D:	DEFB \$01	CLASS-01 LET
	DEFB '='	\$3D. '='
	DEFB \$02	CLASS-02
L1730:	DEFB \$06	CLASS-06 GO TO
	DEFB \$00	CLASS-00
	DEFW GO_TO	\$1E67. GO TO routine in ROM 1.
L1734:	DEFB \$06	CLASS-06 IF
	DEFB \$CB	'THEN'
	DEFB \$0E	CLASS-0E
	DEFW L1986	New IF routine in ROM 0.
L1739:	DEFB \$06	CLASS-06 GO SUB
	DEFB \$0C	CLASS-0C
	DEFW L1A72	New GO SUB routine in ROM 0.
L173D:	DEFB \$00	CLASS-00 STOP
	DEFW STOP	\$1CEE. STOP routine in ROM 1.
L1740:	DEFB \$0C	CLASS-0C RETURN
	DEFW L1A8E	New RETURN routine in ROM 0.
L1743:	DEFB \$04	CLASS-04 FOR
	DEFB '='	\$3D. '='
	DEFB \$06	CLASS-06
	DEFB \$CC	'TO'
	DEFB \$06	CLASS-06
	DEFB \$0E	CLASS-0E
	DEFW L19A0	New FOR routine in ROM 0.
L174B:	DEFB \$04	CLASS-04 NEXT
	DEFB \$00	CLASS-00
	DEFW NEXT	\$1DAB. NEXT routine in ROM 1.
L174F:	DEFB \$0E	CLASS-0E PRINT
	DEFW L2197	New PRINT routine in ROM 0.
L1752:	DEFB \$0E	CLASS-0E INPUT
	DEFW L21AB	New INPUT routine in ROM 0.
L1755:	DEFB \$0E	CLASS-0E DIM
	DEFW L21F4	New DIM routine in ROM 0.
L1758:	DEFB \$0E	CLASS-0E REM
	DEFW L1881	New REM routine in ROM 0.
L175B:	DEFB \$0C	CLASS-0C NEW
	DEFW L21C9	New NEW routine in ROM 0.
L175E:	DEFB \$0D	CLASS-0D RUN
	DEFW L1A21	New RUN routine in ROM 0.

## SPECTRUM +2 ROM o DISASSEMBLY

L1761:	DEFB \$0E	CLASS-0E LIST
	DEFW L1B94	New LIST routine in ROM 0.
L1764:	DEFB \$08	CLASS-08 POKE
	DEFB \$00	CLASS-00
	DEFW POKE	\$1E80. POKE routine in ROM 1.
L1768:	DEFB \$03	CLASS-03 RANDOMIZE
	DEFW RANDOMIZE	\$1E4F. RANDOMIZE routine in ROM 1.
L176B:	DEFB \$00	CLASS-00 CONTINUE
	DEFW CONTINUE	\$1E5F. CONTINUE routine in ROM 1.
L176E:	DEFB \$0D	CLASS-0D CLEAR
	DEFW L1A2C	New CLEAR routine in ROM 0.
L1771:	DEFB \$00	CLASS-00 CLS
	DEFW CLS	\$0D6B. CLS routine in ROM 1.
L1774:	DEFB \$09	CLASS-09 PLOT
	DEFB \$00	CLASS-00
	DEFW PLOT	\$22DC. PLOT routine in ROM 1
L1778:	DEFB \$06	CLASS-06 PAUSE
	DEFB \$00	CLASS-00
	DEFW PAUSE	\$1F3A. PAUSE routine in ROM 1.
L177C:	DEFB \$0E	CLASS-0E READ
	DEFW L19CA	New READ routine in ROM 0.
L177F:	DEFB \$0E	CLASS-0E DATA
	DEFW L1A0A	New DATA routine in ROM 0.
L1782:	DEFB \$03	CLASS-03 RESTORE
	DEFW RESTORE	\$1E42. RESTORE routine in ROM 1.
L1785:	DEFB \$09	CLASS-09 DRAW
	DEFB \$0E	CLASS-0E
	DEFW L21DD	New DRAW routine in ROM 0.
L1789:	DEFB \$0C	CLASS-0C COPY
	DEFW L21C6	New COPY routine in ROM 0.
L178C:	DEFB \$0E	CLASS-0E LPRINT
	DEFW L2193	New LPRINT routine in ROM 0.
L178F:	DEFB \$0E	CLASS-0E LLIST
	DEFW L1B90	New LLIST routine in ROM 0.
L1792:	DEFB \$0B	CLASS-0B SAVE
L1793:	DEFB \$0B	CLASS-0B LOAD
L1794:	DEFB \$0B	CLASS-0B VERIFY
L1795:	DEFB \$0B	CLASS-0B MERGE
L1796:	DEFB \$08	CLASS-08 BEEP
	DEFB \$00	CLASS-00
	DEFW BEEP	\$03F8. BEEP routine in ROM 1.
L179A:	DEFB \$09	CLASS-09 CIRCLE
	DEFB \$0E	CLASS-0E
	DEFW L21CD	New CIRCLE routine in ROM 0.
L179E:	DEFB \$07	CLASS-07 INK
L179F:	DEFB \$07	CLASS-07 PAPER
L17A0:	DEFB \$07	CLASS-07 FLASH
L17A1:	DEFB \$07	CLASS-07 BRIGHT
L17A2:	DEFB \$07	CLASS-07 INVERSE
L17A3:	DEFB \$07	CLASS-07 OVER
L17A4:	DEFB \$08	CLASS-08 OUT
	DEFB \$00	CLASS-00
	DEFW COUT	\$1E7A. OUT routine in ROM 1.
L17A8:	DEFB \$06	CLASS-06 BORDER
	DEFB \$00	CLASS-00
	DEFW BORDER	\$2294. BORDER routine in ROM 1.
L17AC:	DEFB \$0E	CLASS-0E DEF FN
	DEFW L1AAB	New DEF FN routine in ROM 0.
L17AF:	DEFB \$06	CLASS-06 OPEN #
	DEFB ','	\$2C. ','
	DEFB \$0A	CLASS-0A
	DEFB \$00	CLASS-00
	DEFW OPEN	\$1736. OPEN # routine in ROM 1.
L17B5:	DEFB \$06	CLASS-06 CLOSE #
	DEFB \$00	CLASS-00
	DEFW CLOSE	\$16E5. CLOSE # routine in ROM 1.
L17B9:	DEFB \$0E	CLASS-0E FORMAT

## SPECTRUM +2 ROM 0 DISASSEMBLY

	DEFW L0660	FORMAT routine in ROM 0.
L17BC:	DEFB \$0A	CLASS-0A MOVE
	DEFB ','	\$2C. ','
	DEFB \$0A	CLASS-0A
	DEFB \$0C	CLASS-0C
	DEFW L1B0F	Just execute a RET.
L17C2:	DEFB \$0E	CLASS-0E ERASE
	DEFW L1C2B	New ERASE routine in ROM 0.
L17C5:	DEFB \$0E	CLASS-0E CAT
	DEFW L1C04	New CAT routine in ROM 0.
L17C8:	DEFB \$0C	CLASS-0C SPECTRUM
	DEFW L1B4A	SPECTRUM routine in ROM 0.
L17CB:	DEFB \$0E	CLASS-0E PLAY
	DEFW L2336	PLAY routine in ROM 0.

(From Logan & O'Hara's 48K ROM disassembly):

The requirements for the different command classes are as follows: CLASS-00 - No further operands.

CLASS-01 - Used in LET. A variable is required.

CLASS-02 - Used in LET. An expression, numeric or string, must follow.

CLASS-03 - A numeric expression may follow. Zero to be used in case of default.

CLASS-04 - A single character variable must follow.

CLASS-05 - A set of items may be given.

CLASS-06 - A numeric expression must follow.

CLASS-07 - Handles colour items.

CLASS-08 - Two numeric expressions, separated by a comma, must follow.

CLASS-09 - As for CLASS-08 but colour items may precede the expressions.

CLASS-0A - A string expression must follow.

CLASS-0B - Handles cassette/RAM disk routines.

In addition the 128 adds the following classes:

CLASS-0C - Like class 00 but calling ROM 0. (Used by SPECTRUM, MOVE, COPY, NEW, GO SUB, RETURN)

CLASS-0D - Like class 06 but calling ROM 0. (Used by CLEAR, RUN)

CLASS-0E - Handled in ROM 0. (Used by PLAY, ERASE, CAT, FORMAT, CIRCLE, LPRINT, LLIST, DRAW, DATA, READ, LIST, DIM, INPUT, PRINT, FOR, IF)

## The 'Main Parser' Of the BASIC Interpreter

The parsing routine of the BASIC interpreter is entered at \$17CE (ROM 0) when syntax is being checked, and at \$1857 (ROM 0) when a BASIC program of one or more statements is to be executed.

This code is similar to that in ROM 1 at \$1B17.

L17CE:	RES 7,(IY+\$01)	FLAGS. Signal 'syntax checking'.
	RST 28H	
	DEFW E_LINE_NO	\$19FB. CH-ADD is made to point to the first code after any line number
	XOR A	
	LD (\$5C47),A	SUBPPC. Set to \$00.
	DEC A	
	LD (\$5C3A),A	ERR_NR. Set to \$FF.
	JR L17E0	Jump forward to consider the first statement of the line.

## The Statement Loop

Each statement is considered in turn until the end of the line is reached.

L17DF:	RST 20H	Advance CH-ADD along the line.
L17E0:	RST 28H	
	DEFW SET_WORK	\$16BF. The work space is cleared.
	INC (IY+\$0D)	SUBPPC. Increase SUBPPC on each passage around the loop.
	JP M,L1931	Only '127' statements are allowed in a single line. Jump to report "C Nonsense in BASIC".
	RST 18H	Fetch a character.
	LD B,\$00	Clear the register for later.
	CP \$0D	Is the character a 'carriage return'?
	JP Z,L1882	jump if it is.
	CP ':'	\$3A. Go around the loop again if it is a ':'.
	JR Z,L17DF	

## SPECTRUM +2 ROM 0 DISASSEMBLY

A statement has been identified so, first, its initial command is considered

	LD HL,L1840	Pre-load the machine stack with the return address.
	PUSH HL	
	LD C,A	Save the command temporarily
	RST 20H	in the C register whilst CH-ADD is advanced again.
	LD A,C	
	SUB \$CE	Reduce the command's code by \$CE giving the range indexed from \$00.
	JR NC,L1813	Jump for DEF FN and above.
	ADD A,\$CE	
	LD HL,L17C8	
	CP \$A3	Is it 'SPECTRUM'?
	JR Z,L181F	Jump if so into the scanning loop with this address.
	LD HL,L17CB	
	CP \$A4	Is it 'PLAY'?
	JR Z,L181F	Jump if so into the scanning loop with this address.
	JP L1931	Produce error report "C Nonsense in BASIC".
L1813:	LD C,A	Move the command code to BC (B holds \$00).
	LD HL,L16FB	The base address of the syntax offset table.
	ADD HL,BC	
	LD C,(HL)	
	ADD HL,BC	Find address for the command's entries in the parameter table.
	JR L181F	Jump forward into the scanning loop with this address.

Each of the command class routines applicable to the present command are executed in turn.

Any required separators are also considered.

L181C:	LD HL,(\$5C74)	T_ADDR. The temporary pointer to the entries in the parameter table.
L181F:	LD A,(HL)	Fetch each entry in turn.
	INC HL	Update the pointer to the entries for the next pass.
	LD (\$5C74),HL	T_ADDR.
	LD BC,L181C	Pre-load the machine stack with the return address.
	PUSH BC	
	LD C,A	Copy the entry to the C register for later.
	CP \$20	
	JR NC,L1839	Jump forward if the entry is a 'separator'.
	LD HL,L18D4	The base address of the 'command class' table.
	LD B,\$00	
	ADD HL,BC	Index into the table.
	LD C,(HL)	
	ADD HL,BC	HL=base + code + (base + code).
	PUSH HL	HL=The starting address of the required command class routine.
	RST 18H	Before making an indirect jump to the command class routine pass the command code
	DEC B	to the A register and set the B register to \$FF.
	RET	Return to the stacked address.

## The 'Separator' Subroutine

The report 'Nonsense in BASIC' is given if the required separator is not present.

But note that when syntax is being checked the actual report does not appear on the screen - only the 'error marker'.

This code is similar to that in ROM 1 at \$1B6F.

L1839:	RST 18H	The current character is
	CP C	fetches and compared to the entry in the parameter table.
	JP NZ,L1931	Give the error report if there is not a match.
	RST 20H	Step past a correct character
	RET	and return.

## The 'Statement Return' Subroutine

After the correct interpretation of a statement, a return is made to this entry point.

This code is similar to that in ROM 1 at \$1B76.

L1840:	CALL L05F5 JR C,L1849 CALL L05CB DEFB \$14	Check for BREAK Jump if pressed. Produce error report. "L Break into program"
L1849:	BIT 7,(IY+\$0A) JP NZ,L18C7 LD HL,(\$5C42) BIT 7,H JR Z,L186B	NSPPC - statement number in line to be jumped to Jump forward if there is not a 'jump' to be made. NEWPPC, line number to be jumped to.  Jump forward unless dealing with a further statement in the editing area.

## The 'Line Run' Entry Point

This entry point is used wherever a line in the editing area is to be 'run'.

In such a case the syntax/run flag (bit 7 of FLAGS) will be set.

The entry point is also used in the syntax checking of a line in the editing area that has more than one statement (bit 7 of FLAGS will be reset).

This code is similar to that in ROM 1 at \$1B8A.

L1857:	LD HL,\$FFFE LD (\$5C45),HL LD HL,(\$5C61) DEC HL LD DE,(\$5C59) DEC DE LD A,(\$5C44) JR L18A1	A line in the editing area is considered as line '-2'. PPC. WORKSP. Make HL point to the end marker of the editing area.  E_LINE. Make DE point to the location before the end marker of the editing area.  NSPPC. Fetch the number of the next statement to be handled. Jump forward.
--------	---	---

## The 'Line New' Subroutine

There has been a jump in the program and the starting address of the new line has to be found.

This code is similar to that in ROM 1 at 1B9E.

L186B:	RST 28H DEFW LINE_ADDR LD A,(\$5C44) JR Z,L188F AND A JR NZ,L18BC LD B,A LD A,(HL) AND \$C0 LD A,B JR Z,L188F CALL L05CB DEFB \$FF	\$196E. The starting address of the line, or the 'first line after' is found. NSPPC. Collect the statement number. Jump forward if the required line was found. Check the validity of the statement number - must be zero. Jump if not to produce error report "N Statement lost". Also check that the 'first line after' is not after the actual 'end of program'.  Jump forward with valid addresses; otherwise signal the error 'OK'. Produce error report. "0 OK"
--------	--	--

## REM Routine

The return address to STMT-RET is dropped which has the effect of forcing the rest of the line to be ignored.

This code is similar to that in ROM 1 at \$1BB2.

L1881:	POP BC	Drop the statement return address.
--------	--------	------------------------------------

## The 'Line End' Routine

If checking syntax a simple return is made but when 'running' the address held by NXTLIN has to be checked before it can be used.

This code is similar to that in ROM 1 at \$1BB3.

L1882:	BIT 7,(IY+\$01) RET Z LD HL,(\$5C55) LD A,\$C0 AND (HL)	Return if syntax is being checked. NXTLIN. Return if the address is after the end of the program - the 'run' is finished.
--------	---	---

RET NZ  
XOR A

Signal 'statement zero' before proceeding.

## The 'Line Use' Routine

This routine has three functions:

- Change statement zero to statement '1'.
- Find the number of the new line and enter it into PPC.
- Form the address of the start of the line after.

This code is similar to that in ROM 1 at \$1BBF.

L188F:	CP \$01	Statement zero becomes statement 1.
	ADC A,\$00	
	LD D,(HL)	The line number of the line to be used is collected and
	INC HL	passed to PPC.
	LD E,(HL)	
	LD (\$5C45),DE	PPC.
	INC HL	
	LD E,(HL)	Now find the 'length' of the line.
	INC HL	
	LD D,(HL)	
	EX DE,HL	Switch over the values.
	ADD HL,DE	Form the address of the start of the line after in HL and the
	INC HL	location before the 'next' line's first character in DE.

## The 'Next Line' Routine

On entry the HL register pair points to the location after the end of the 'next' line to be handled and the DE register pair to the location before the first character of the line.

This applies to lines in the program area and also to a line in the editing area - where the next line will be the same line again whilst there are still statements to be interpreted.

This code is similar to that in ROM 1 at \$1BD1.

L18A1:	LD (\$5C55),HL	NXTLIN. Set NXTLIN for use once the current line has been completed.
	EX DE,HL	
	LD (\$5C5D),HL	CH_ADD. CH_ADD points to the location before the first character to be considered.
	LD D,A	The statement number is fetched.
	LD E,\$00	The E register is cleared in case the 'Each Statement' routine is used.
	LD (IY+\$0A),\$FF	NSPPC. Signal 'no jump'.
	DEC D	
	LD (IY+\$0D),D	SUB_PPC. Statement number-1.
	JP Z,L17DF	Jump if the first statement.
	INC D	For later statements the 'starting address' has to be found.
	RST 28H	
	DEFW EACH_STMT	\$198B.
	JR Z,L18C7	Jump forward unless the statement does not exist.
L18BC:	CALL L05CB	Produce error report.
	DEFB \$16	"N Statement lost"

## The 'CHECK-END' Subroutine

This is called when the syntax of the edit-line is being checked. The purpose of the routine is to give an error report if the end of a statement has not been reached and to move on to the next statement if the syntax is correct.

The routine is the equivalent of routine CHECK\_END in ROM 1 at \$1BEE.

L18C0:	BIT 7,(IY+\$01)	Very like CHECK-END at 1BEE in ROM 1
	RET NZ	Return unless checking syntax.
	POP BC	Drop scan loop and statement return addresses.
	POP BC	

## The 'STMT-NEXT' Routine

If the present character is a 'carriage return' then the 'next statement' is on the 'next line', if '.' it is on the same line; but if any other character is found then there is an error in syntax.

The routine is the equivalent of routine STMT\_NEXT in ROM 1 at \$1BF4.

L18C7:	RST 18H	Fetch the present character.
	CP \$0D	Consider the 'next line' if
	JR Z,L1882	it is a 'carriage return'.
	CP '.'	\$3A. Consider the 'next statement'
	JP Z,L17DF	if it is a '.'.
	JP L1931	Otherwise there has been a syntax error so produce "C Nonsense in BASIC".

## The 'Command Class' Table

L18D4:	DEFB L18F8-\$	CLASS-00 -> L18D9 = \$24
	DEFB L1918-\$	CLASS-01 -> L18F9 = \$43
	DEFB L191C-\$	CLASS-02 -> L18FD = \$46
	DEFB L18F5-\$	CLASS-03 -> L18D6 = \$1E
	DEFB L1924-\$	CLASS-04 -> L1905 = \$4C
	DEFB L18F9-\$	CLASS-05 -> L18DA = \$20
	DEFB L192D-\$	CLASS-06 -> L190E = \$53
	DEFB L1939-\$	CLASS-07 -> L191A = \$5E
	DEFB L1929-\$	CLASS-08 -> L190A = \$4D
	DEFB L1963-\$	CLASS-09 -> L1944 = \$86
	DEFB L1935-\$	CLASS-0A -> L1916 = \$57
	DEFB L1967-\$	CLASS-0B -> L1948 = \$88
	DEFB L18E6-\$	CLASS-0C -> L18C7 = \$06
	DEFB L18E3-\$	CLASS-0D -> L18C4 = \$02
	DEFB L18E7-\$	CLASS-0E -> L18C8 = \$05

## The 'Command Classes — 0C, 0D & 0E'

For commands of class-0D a numeric expression must follow.

L18E3:	RST 28H	Code 0D enters here.
	DEFW FETCH_NUM	\$1CDE.

The commands of class-0C must not have any operands. e.g. SPECTRUM.

L18E6:	CP A	Code 0C enters here. Set zero flag.
--------	------	-------------------------------------

The commands of class-0E may be followed by a set of items. e.g. PLAY.

L18E7:	POP BC	Code 0E enters here. Retrieve return address.
	CALL Z,L18C0	If handling commands of classes 0C & 0D and syntax is being checked move on
		now to consider the next statement.
	EX DE,HL	Save the line pointer in DE.

After the command class entries and the separator entries in the parameter table have been considered the jump to the appropriate command routine is made.

The routine is similar to JUMP-C-R in ROM 1 at \$1C16.

LD HL,(\$5C74)	T_ADDR.
LD C,(HL)	Fetch the pointer to the entries in the parameter table
INC HL	and fetch the address of the
LD B,(HL)	required command routine.
EX DE,HL	Exchange the pointers back.
PUSH BC	Make an indirect jump to the command routine.
RET	

## The 'Command Classes — 00, 03 & 05'

These routines are the equivalent of the routines in ROM 1 starting at \$1C0D.

The commands of class-03 may, or may not, be followed by a number. e.g. RUN & RUN 200.

L18F5:            RST 28H                    Code 03 enters here.  
                   DEFW FETCH\_NUM           \$1CDE. A number is fetched but zero is used in cases of default.

The commands of class-00 must not have any operands. e.g. COPY & CONTINUE.

L18F8:            CP A                        Code 00 enters here. Set the zero flag.

The commands of class-05 may be followed by a set of items. e.g. PRINT & PRINT "222".

L18F9:            POP BC                    Code 05 enters here. Drop return address.  
                   CALL Z,L18C0            If handling commands of classes 00 & 03 and syntax is being checked move on now  
    to consider the next statement.  
                   EX DE,HL                Save the line pointer in DE.  
                   LD HL,(\$C74)            T\_ADDR. Fetch the pointer to the entries in the parameter table.  
                   LD C,(HL)  
                   INC HL  
                   LD B,(HL)                Fetch the address of the required command routine.  
                   EX DE,HL                Exchange the pointers back.  
                   PUSH HL                Save command routine address.  
                   LD HL,L1917            The address to return to (the RET below).  
                   LD (RETADDR),HL       \$5B5A. Store the return address.  
                   LD HL,YOUNGER        \$5B14. Paging subroutine.  
                   EX (SP),HL            Replace the return address with the address of the YOUNGER routine.  
                   PUSH HL                Save the original top stack item.  
                   LD H,B  
                   LD L,C                HL=Address of command routine.  
                   EX (SP),HL            Put onto the stack so that an indirect jump will be made to it.  
                   JP SWAP                \$5B00. Switch to other ROM and 'return' to the command routine.

Comes here after ROM 1 has been paged in, the command routine called, ROM 0 paged back in.

L1917:            RET                        Simply make a return.

## The 'Command Class — 01'

Command class 01 is concerned with the identification of the variable in a LET, READ or INPUT statement.

L1918:            RST 28H                    Delegate handling to ROM 1.  
                   DEFW CLASS\_01        \$1C1F.  
                   RET

## The 'Command Class — 02'

Command class 02 is concerned with the actual calculation of the value to be assigned in a LET statement.

L191C:            POP BC                    Code 02 enters here. Delegate handling to ROM 1.  
                   RST 28H  
                   DEFW VAL\_FET\_1        \$1C56. "... used by LET, READ and INPUT statements to first evaluate and then  
    assign values to the previously designated variable" (Logan/O'Hara)  
                   CALL L18C0            Move on to the next statement if checking syntax  
                   RET                    else return here.

## The 'Command Class — 04'

The command class 04 entry point is used by FOR & NEXT statements.

L1924:	RST 28H DEFW CLASS_04 RET	Code 04 enters here. Delegate handling to ROM 1. \$1C6C.
--------	---------------------------------	---

## The 'Command Class — 08'

Command class 08 allows for two numeric expressions, separated by a comma, to be evaluated.

L1928:	RST 20H	[Redundant byte]
L1929:	RST 28H DEFW EXPT_2NUM RET	Delegate handling to ROM 1. \$1C7A.

## The 'Command Class — 06'

Command class 06 allows for a single numeric expression to be evaluated.

L192D:	RST 28H DEFW EXPT_1NUM RET	Code 06 enters here. Delegate handling to ROM 1. \$1C82.
--------	----------------------------------	---

## Report C — Nonsense in BASIC

L1931:	CALL L05CB  DEFB \$0B	Produce error report. [Could have saved 4 bytes by using the identical routine at \$1238 (ROM 0) instead] "C Nonsense in BASIC"
--------	-----------------------------	---

## The 'Command Class — 0A'

Command class 0A allows for a single string expression to be evaluated.

L1935:	RST 28H DEFW EXPT_EXP RET	Code 0A enters here. Delegate handling to ROM 1. \$1C8C.
--------	---------------------------------	---

## The 'Command Class — 07'

Command class 07 is the command routine for the six colour item commands.  
Makes the current temporary colours permanent.

L1939:	BIT 7,(IY+\$01) RES 0,(IY+\$02) JR Z,L1946 RST 28H DEFW TEMPS	The syntax/run flag is read. TV_FLAG. Signal 'main screen'. Jump ahead if syntax checking. Only during a 'run' call TEMPS to ensure the temporary \$0D4D. colours are the main screen colours.
L1946:	POP AF LD A,(\$5C74) SUB (L179E & \$00FF)+\$28 RST 28H DEFW CO_TEMP_4 CALL L18C0 LD HL,(\$5C8F) LD (\$5C8D),HL LD HL,\$5C91 LD A,(HL)	Drop the return address. T_ADDR. Reduce to range \$D9-\$DE which are the token codes for INK to OVER.  \$21FC. Change the temporary colours as directed by the BASIC statement. Move on to the next statement if checking syntax. ATTR_T. Now the temporary colour ATTR_P. values are made permanent P_FLAG. Value of P_FLAG also has to be considered.

The following instructions cleverly copy the even bits of the supplied byte to the odd bits.  
In effect making the permanent bits the same as the temporary ones.

RLCA	Move the mask leftwards.
XOR (HL)	Impress onto the mask
AND \$AA	only the even bits of the
XOR (HL)	other byte.
LD (HL),A	Restore the result.
RET	

## The 'Command Class — 09'

This routine is used by PLOT, DRAW & CIRCLE statements in order to specify the default conditions of 'FLASH 8; BRIGHT 8; PAPER 8;' that are set up before any embedded colour items are considered.

L1963:	RST 28H	Code 09 enters here. Delegate handling to ROM 1.
	DEFW CLASS_09	\$1CBE.
	RET	

## The 'Command Class — 0B'

This routine is used by SAVE, LOAD, VERIFY & MERGE statements.

L1967:	POP AF	Drop the return address.
	LD A,(FLAGS3)	\$5B66.
	AND \$0F	Clear LOAD/SAVE/VERIFY/MERGE indication bits.
	LD (FLAGS3),A	\$5B66.
	LD A,(\$5C74)	T_ADDR-lo.
	SUB 1+(L1792 & \$00FF)	Correct by \$74 so that SAVE = \$00, LOAD = \$01, VERIFY = \$02, MERGE = \$03.
	LD (\$5C74),A	T_ADDR-lo.
	JP Z,L120A	Jump to handle SAVE.
	DEC A	
	JP Z,L1211	Jump to handle LOAD.
	DEC A	
	JP Z,L1218	Jump to handle VERIFY.
	JP L121F	Jump to handle MERGE.

## IF Routine

On entry the value of the expression between the IF and the THEN is the 'last value' on the calculator stack. If this is logically true then the next statement is considered; otherwise the line is considered to have been finished.

L1986:	POP BC	Drop the return address.
	BIT 7,(IY+\$01)	
	JR Z,L199D	Jump forward if checking syntax.

Now 'delete' the last value on the calculator stack

L198D:	LD HL,(\$5C65)	STKEND.
	LD DE,\$FFFB	-5
	ADD HL,DE	The present 'last value' is deleted.
	LD (\$5C65),HL	STKEND. HL point to the first byte of the value.
	RST 28H	
	DEFW TEST_ZERO	\$34E9. Is the value zero?
	JP C,L1882	If the value was 'FALSE' jump to the next line.
L199D:	JP L17E0	But if 'TRUE' jump to the next statement (after the THEN).

## FOR Routine

This command routine is entered with the VALUE and the LIMIT of the FOR statement already on the top of the calculator stack.

L19A0:	CP \$CD	Jump forward unless a 'STEP' is given.
	JR NZ,L19AD	
	RST 20H	Advance pointer

## SPECTRUM +2 ROM o DISASSEMBLY

CALL L192D	Indirectly call EXPT_1NUM in ROM 1 to get the value of the STEP.
CALL L18C0	Move on to the next statement if checking syntax.
JR L19C5	Otherwise jump forward.

There has not been a STEP supplied so the value '1' is to be used.

L19AD:	CALL L18C0	Move on to the next statement if checking syntax.
	LD HL,(\$5C65)	STKEND.
	LD (HL),\$00	
	INC HL	
	LD (HL),\$00	
	INC HL	
	LD (HL),\$01	
	INC HL	
	LD (HL),\$00	
	INC HL	
	LD (HL),\$00	Place a value of 1 on the calculator stack.
	INC HL	
	LD (\$5C65),HL	STKEND.

The three values on the calculator stack are the VALUE (v), the LIMIT (l) and the STEP (s).  
These values now have to be manipulated. Delegate handling to ROM 1.

L19C5:	RST 28H	
	DEFW F_REORDER	\$1D16.
	RET	

### READ Routine

L19C9:	RST 20H	Come here on each pass, after the first, to move along the READ statement.
L19CA:	CALL L1918	Indirectly call CLASS_01 in ROM 1 to consider whether the variable has been used before, and find the existing entry if it has.
	BIT 7,(IY+\$01)	
	JR Z,L1A01	Jump forward if checking syntax.
	RST 18H	Save the current pointer CH_ADD in X_PTR.
	LD (\$5C5F),HL	X_PTR.
	LD HL,(\$5C57)	DATADD.
	LD A,(HL)	Fetch the current DATA list pointer
	CP \$2C	and jump forward unless a new
	JR Z,L19EA	DATA statement has to be found.
	LD E,\$E4	The search is for 'DATA'.
	RST 28H	
	DEFW LOOK_PROG	\$1D86.
	JR NC,L19EA	Jump forward if the search is successful.
	CALL L05CB	Produce error report.
	DEFB \$0D	"E Out of Data"

Pick up a value from the DATA list.

L19EA:	INC HL	Advance the pointer along the DATA list.
	LD (\$5C5D),HL	CH_ADD.
	LD A,(HL)	
	RST 28H	
	DEFW VAL_FET_1	\$1C56. Fetch the value and assign it to the variable.
	RST 18H	
	LD (\$5C57),HL	DATADD.
	LD HL,(\$5C5F)	X_PTR. Fetch the current value of CH_ADD and store it in DATADD.
	LD (IY+\$26),\$00	X_PTR_hi. Clear the address of the character after the '?' marker.
	LD (\$5C5D),HL	CH_ADD. Make CH-ADD once again point to the READ statement.
	LD A,(HL)	
L1A01:	RST 18H	GET the present character
	CP ','	\$2C. Check if it is a ','.
L1A04:	JR Z,L19C9	If it is then jump back as there are further items.

CALL L18C0  
RET

Return if checking syntax  
or here if not checking syntax.

## DATA Routine

During syntax checking a DATA statement is checked to ensure that it contains a series of valid expressions, separated by commas. But in 'run-time' the statement is passed by.

L1A0A:       BIT 7,(IY+\$01)               Jump forward unless checking syntax.  
              JR NZ,L1A1B

A loop is now entered to deal with each expression in the DATA statement.

L1A10:       RST 28H                       \$24FB. Scan the next expression.  
              DEFW SCANNING               \$2C. Check for the correct separator ','.  
              CP ','                       but move on to the next statement if not matched.  
              CALL NZ,L18C0               Whilst there are still expressions to be checked  
              RST 20H                       go around again.  
              JR L1A10

The DATA statement has to be passed-by in 'run-time'.

L1A1B:       LD A,\$E4                     It is a 'DATA' statement that is to be passed-by.

On entry the A register will hold either the token 'DATA' or the token 'DEF FN' depending on the type of statement that is being 'passed-by'.

L1A1D:       RST 28H                       \$1E39. Delegate handling to ROM 1.  
              DEFW PASS\_BY  
              RET

## RUN Routine

The parameter of the RUN command is passed to NEWPPC by calling the GO TO command routine. The operations of 'RESTORE 0' and 'CLEAR 0' are then performed before a return is made.

L1A21:       RST 28H                       \$1E67.  
              DEFW GO\_TO                  Now perform a 'RESTORE 0'.  
              LD BC,\$0000  
              RST 28H                       \$1E45.  
              DEFW REST\_RUN               Exit via the CLEAR command routine.  
              JR L1A2F

## CLEAR Routine

This routine allows for the variables area to be cleared, the display area cleared and RAMTOP moved. In consequence of the last operation the machine stack is rebuilt thereby having the effect of also clearing the GO SUB stack.

L1A2C:       RST 28H                       \$1E99. Fetch the operand - using zero by default.  
              DEFW FIND\_INT2  
L1A2F:       LD A,B                       Jump forward if the operand is  
              OR C                       other than zero. When called  
              JR NZ,L1A37               from RUN there is no jump.  
              LD BC,(\$5CB2)               RAMTOP. Use RAMTOP if the parameter is 0.  
L1A37:       PUSH BC                       BC = Address to clear to. Save it.  
              LD DE,(\$5C4B)               VARS.  
              LD HL,(\$5C59)               E LINE.  
              DEC HL  
              RST 28H                       Delete the variables area.  
              DEFW RECLAIM               \$19E5.  
              RST 28H                       Clear the screen  
              DEFW CLS                    \$0D6B.

## SPECTRUM +2 ROM o DISASSEMBLY

The value in the BC register pair which will be used as RAMTOP is tested to ensure it is neither too low nor too high.

	LD HL,(\$5C65)	STKEND. The current value of STKEND
	LD DE,\$0032	is increased by 50 before
	ADD HL,DE	being tested. This forms the
	POP DE	ADE = address to clear to lower limit.
	SBC HL,DE	
	JR NC,L1A5A	Ramtop no good.
	LD HL,(\$5CB4)	P_RAMT. For the upper test the value
	AND A	for RAMTOP is tested against P_RAMT.
	SBC HL,DE	
	JR NC,L1A5E	Jump forward if acceptable.
L1A5A:	CALL L05CB	Produce error report.
	DEFB \$15	"M Ramtop no good"
L1A5E:	LD (\$5CB2),DE	RAMTOP.
	POP DE	Retrieve interpreter return address from stack
	POP HL	Retrieve 'error address' from stack
	POP BC	Retrieve the GO SUB stack end marker. <b>[BUG</b> - It is assumed that the top of the
		GO SUB stack will be empty and hence only contain the end marker. This will not
		be the case if CLEAR is used within a subroutine, in which case BC will now hold
		the calling line number and this will be stacked in place of the end marker. When a
		RETURN command is encountered, the GO SUB stack appears to contain an entry
		since the end marker was not the top item. An attempt to return is therefore made.
		The CLEAR command handler within the 48K Spectrum ROM does not make any
		assumption about the contents of the GO SUB stack and instead always re-inserts
		the end marker. The bug could be fixed by inserting the line LD BC,\$3E00 after the
		POP BC. Credit: Ian Collier (+3), Paul Farrow (128)]
		RAMTOP.
	LD SP,(\$5CB2)	
	INC SP	
	PUSH BC	Stack the GO SUB stack end marker.
	PUSH HL	Stack 'error address'.
	LD (\$5C3D),SP	ERR_SP.
	PUSH DE	Stack the interpreter return address.
	RET	

## GO SUB Routine

The present value of PPC and the incremented value of SUBPPC are stored on the GO SUB stack.

L1A72:	POP DE	Save the return address.
	LD H,(IY+\$0D)	SUBPPC. Fetch the statement number and increment it.
	INC H	
	EX (SP),HL	Exchange the 'error address' with the statement number.
	INC SP	Reclaim the use of a location.
	LD BC,(\$5C45)	PPC.
	PUSH BC	Next save the present line number.
	PUSH HL	Return the 'error address' to the machine stack
	LD (\$5C3D),SP	ERR-SP. and reset ERR-SP to point to it.
	PUSH DE	Stack the return address.
	RST 28H	
	DEFW GO_TO	\$1E67. Now set NEWPPC & NSPPC to the required values.
	LD BC,\$0014	But before making the jump make a test for room.
	RST 28H	
	DEFW TEST_ROOM	\$1F05. Will automatically produce error '4' if out of memory.
	RET	

## RETURN Routine

The line number and the statement number that are to be made the object of a 'return' are fetched from the GO SUB stack.

L1A8E:	POP BC	Fetch the return address.
	POP HL	Fetch the 'error address'.
	POP DE	Fetch the last entry on the GO SUB stack.
	LD A,D	The entry is tested to see if

	CP \$3E	it is the GO SUB stack end marker.
	JR Z,L1AA5	Jump if it is.
	DEC SP	The full entry uses three locations only.
	EX (SP),HL	Exchange the statement number with the 'error address'.
	EX DE,HL	Move the statement number.
	LD (\$5C3D),SP	ERR_SP. Reset the error pointer.
	PUSH BC	Replace the return address.
	LD (\$5C42),HL	NEWPPC. Enter the line number.
	LD (IY+\$0A),D	NSPPC. Enter the statement number.
	RET	
L1AA5:	PUSH DE	Replace the end marker and
	PUSH HL	the 'error address'.
	CALL L05CB	Produce error report.
	DEFB \$06	"7 RETURN without GO SUB"

## DEF FN Routine

During syntax checking a DEF FN statement is checked to ensure that it has the correct form.  
Space is also made available for the result of evaluating the function.  
But in 'run-time' a DEF FN statement is passed-by.

L1AAB:	BIT 7,(IY+\$01)	
	JR Z,L1AB6	Jump forward if checking syntax.
	LD A,\$CE	Otherwise pass-by the
	JP L1A1D	'DEF FN' statement.

First consider the variable of the function.

L1AB6:	SET 6,(IY+\$01)	Signal 'a numeric variable'.
	RST 28H	
	DEFW ALPHA	\$2C8D. Check that the present code is a letter.
	JR NC,L1AD5	Jump forward if not.
	RST 20H	Fetch the next character.
	CP '\$'	\$24.
	JR NZ,L1AC9	Jump forward unless it is a '\$'.
	RES 6,(IY+\$01)	Change bit 6 as it is a string variable.
	RST 20H	Fetch the next character.
L1AC9:	CP '('	\$28. A '(' must follow the variable's name.
	JR NZ,L1B09	Jump forward if not.
	RST 20H	Fetch the next character
	CP ')'	\$29. Jump forward if it is a ')' as there are no parameters of the function.
	JR Z,L1AF2	

A loop is now entered to deal with each parameter in turn.

L1AD2:	RST 28H	
	DEFW ALPHA	\$2C8D.
L1AD5:	JP NC,L1931	The present code must be a letter.
	EX DE,HL	Save the pointer in DE.
	RST 20H	Fetch the next character.
	CP '\$'	\$24.
	JR NZ,L1AE0	Jump forward unless it is a '\$'.
	EX DE,HL	Otherwise save the new pointer in DE instead.
	RST 20H	Fetch the next character.
L1AE0:	EX DE,HL	Move the pointer to the last character of the name to HL.
	LD BC,\$0006	Now make six locations after that last character.
	RST 28H	
	DEFW MAKE_ROOM	\$1655.
	INC HL	
	INC HL	
	LD (HL),\$0E	Enter a 'number marker' into the first of the new locations.
	CP ','	\$2C. If the present character is a ',' then jump back as
	JR NZ,L1AF2	there should be a further parameter.
	RST 20H	
	JR L1AD2	Otherwise jump out of the loop.

Next the definition of the function is considered.

L1AF2:	CP ')'	\$29. Check that the ')' does exist.
	JR NZ,L1B09	Jump if not.
	RST 20H	The next character is fetched.
	CP '='	\$3D. It must be an '='.
	JR NZ,L1B09	Jump if not.
	RST 20H	Fetch the next character.
	LD A,(\$5C3B)	FLAGS.
	PUSH AF	Save the nature (numeric or string) of the variable
	RST 28H	
	DEFW SCANNING	\$24FB. Now consider the definition as an expression.
	POP AF	Fetch the nature of the variable.
	XOR (IY+\$01)	FLAGS. Check that it is of the same type
	AND \$40	as found for the definition.
L1B09:	JP NZ,L1931	Give an error report if required.
	CALL L18C0	Move on to consider the next statement in the line.

## MOVE Routine

L1B0F:	RET	Simply return.
--------	-----	----------------

## MENU ROUTINES — PART 1

### Run Tape Loader

Used by Main Menu - Tape Loader option.

L1B10:	LD HL,\$EC0E	Fetch mode.
	LD (HL),\$FF	Set Tape Loader mode.
	CALL L1F3F	Use Normal RAM Configuration (physical RAM bank 0).
	RST 28H	
	DEFW SET_MIN	\$16B0. Clear out editing area.
	LD HL,(\$5C59)	E_LINE.
	LD BC,\$0003	Create 3 bytes of space for the LOAD "" command.
	RST 28H	
	DEFW MAKE_ROOM	\$1655.
	LD HL,L1B8D	Address of command bytes for LOAD "".
	LD DE,(\$5C59)	E_LINE.
	LD BC,\$0003	
	LDIR	Copy LOAD "" into the line editing area.
	CALL L026B	Parse and execute the BASIC line. [Will not return here but will exit via the error handler routine]

### List Program to Printer

Used by Edit Menu - Print option.

L1B33:	CALL L1F3F	Use Normal RAM Configuration (physical RAM bank 0).
	RST 28H	
	DEFW SET_MIN	\$16B0. Clear out editing area.
	LD HL,(\$5C59)	E_LINE.
	LD BC,\$0001	Create 1 byte of space.
	RST 28H	
	DEFW MAKE_ROOM	\$1655.
	LD HL,(\$5C59)	E_LINE.
	LD (HL),\$E1	Copy LLIST into the line editing area.
	CALL L026B	Parse and execute the BASIC line. [Will not return here but will exit via the error handler routine]

## BASIC LINE AND COMMAND INTERPRETATION ROUTINES — PART 2

### SPECTRUM Routine

Return to 48K BASIC Mode. This routine will force caps lock is off.

L1B4A:	CALL L1B72	Overwrite 'P' channel data to use the ZX Printer.
	LD SP,(\$5C3D)	ERR_SP. Purge the stack.
	POP HL	Remove error handler address.
	LD HL,MAIN_4	\$1303. The main execution loop within ROM 1.
	PUSH HL	
	LD HL,PRINT_A_1+\$0003	\$0013. Address of a \$FF byte within ROM 1, used to generate error report "0 OK".
	PUSH HL	
	LD HL,ERROR_1	\$0008. The address of the error handler within ROM 1.
	PUSH HL	

**[BUG -** Although the channel 'P' information has been reconfigured to use the ZX Printer, the ZX printer buffer and associated system variables still need to be cleared. Failure to do so means that the first use of the ZX Printer will cause garbage to be printed, i.e. the paging routines and new system variables still present in the ZX Printer buffer. Subsequently printer output will then be ok since the ZX Printer buffer and system variables will be cleared. Worse still, there is the possibility that new data to be printed will be inserted beyond the ZX Printer buffer since ROM 1 does not trap whether the ZX Printer system variable PR\_POSN and PR\_CC hold invalid values. The bug can be fixed by inserting the following instructions, which cause the ZX Printer buffer to be cleared immediately after switching to ROM 1 and before the error report "0 OK" is produced. Credit: Paul Farrow and Andrew Owen.]

LD HL,CLEAR_PRB	Address of the routine in ROM 1 to clear the ZX Printer buffer and associated system variables.
PUSH HL	
SET 1,(IY+\$01)	FLAGS. Signal the printer is in use.]
LD A,\$20	Force 48K mode.
LD (BANK_M),A	\$5B5C.
JP SWAP	\$5B00. Swap to ROM 1 and return via a RST \$08 / DEFB \$FF.

## MENU ROUTINES — PART 2

### Main Menu — 48 BASIC Option

L1B66:	LD HL,\$0000	Stack a \$0000 address to return to.
	PUSH HL	
	LD A,\$20	Force 48 mode.
	LD (BANK_M),A	\$5B5C
	JP SWAP	\$5B00. Swap to ROM 1, return to \$0000.

### Set 'P' Channel Data

This routine overwrites the 'P' channel data with the 'S' channel data, i.e. the default values when using the ZX Printer.

L1B72:	LD HL,(\$5C4F)	CHANS.
	LD DE,\$0005	
	ADD HL,DE	HL=Address 'S' channel data.
	LD DE,\$000A	
	EX DE,HL	HL=\$000A, DE=Address 'S' channel data.
	ADD HL,DE	HL=Address 'P' channel data.
	EX DE,HL	DE=Address 'P' channel data, HL=Address 'S' channel data.
	LD BC,\$0004	
	LDIR	Copy the 'S' channel data over the 'P' channel data.
	RES 3,(IY+\$30)	FLAGS2. Signal caps lock unset. [Not really necessary for switching back to 48 BASIC mode]
	RES 4,(IY+\$01)	FLAGS. Signal not 128K mode.
	RET	

## LOAD "" Command Bytes

Used by the Tape Loader routine.

L1B8D:	DEFB \$EF, \$22, \$22	LOAD ""
--------	-----------------------	---------

## BASIC LINE AND COMMAND INTERPRETATION ROUTINES — PART 3

### LLIST Routine

L1B90:	LD A,\$03	Printer channel.
	JR L1B96	Jump ahead to join LIST.

### LIST Routine

L1B94:	LD A,\$02	Main screen channel.
L1B96:	LD (IY+\$02),\$00	TV_FLAG. Signal 'an ordinary listing in the main part of the screen'.
	RST 28H	
	DEFW SYNTAX_Z	\$2530.
	JR Z,L1BA2	Do not open the channel if checking syntax.
	RST 28H	
	DEFW CHAN_OPEN	\$1601. Open the channel.
L1BA2:	RST 28H	
	DEFW GET_CHAR	\$0018. [Could just do RST \$18]
	RST 28H	
	DEFW STR_ALTER	\$2070. See if the stream is to be changed.
	JR C,L1BC2	Jump forward if unchanged.
	RST 28H	
	DEFW GET_CHAR	\$0018. Get current character.
	CP \$3B	Is it a ','?
	JR Z,L1BB5	Jump if it is.
	CP ','	\$2C. Is it a ','?
	JR NZ,L1BBD	Jump if it is not.
L1BB5:	RST 28H	
	DEFW NEXT_CHAR	\$0020. Get the next character.
	CALL L192D	Indirectly call EXPT-1NUM in ROM 1 to check that a numeric expression follows, e.g. LIST #5,20.
	JR L1BC5	Jump forward with it.
L1BBD:	RST 28H	
	DEFW USE_ZERO	\$1CE6. Otherwise use zero and
	JR L1BC5	jump forward.

Come here if the stream was unaltered.

L1BC2:	RST 28H	
	DEFW FETCH_NUM	\$1CDE. Fetch any line or use zero if none supplied.
L1BC5:	CALL L18C0	If checking the syntax of the edit-line move on to the next statement.
	RST 28H	
	DEFW LIST_5+3	\$1825. Delegate handling to ROM 1.
	RET	

### RAM Disk SAVE! Routine

L1BCC:	LD (OLDSP),SP	\$5B81. Save SP.
	LD SP,TSTACK	\$5BFF. Use temporary stack.
	CALL L1CB6	Create new catalogue entry.
	LD BC,(HD_0B)	\$5B72. get the length of the file.

```
LD HL,$FFF7
OR $FF
SBC HL,BC
CALL L1D12
LD BC,$0009
LD HL,HD_00
CALL L1DCB
LD HL,(HD_0D)
LD BC,(HD_0B)
CALL L1DCB
CALL L1D75
LD A,$05
CALL L1C83
LD SP,(OLDSP)
RET
```

-9 (9 is the length of the file header).  
 Extend the negative number into the high byte.  
 AHL=-(length of file + 9).  
 Check for space in RAM disk (produce "4 Out of memory" if no room).  
 File header length.  
 \$5B71. Address of file header.  
 Store file header to RAM disk.  
 \$5B74. Start address of file data.  
 \$5B72. Length of file data.  
 Store bytes to RAM disk.  
 Update catalogue entry (leaves logical RAM bank 4 paged in).  
 Page in logical RAM bank 5 (physical RAM bank 0).  
 \$5B81. Use original stack.

## CAT! Routine

```
L1C04:  RST 28H
        DEFW GET_CHAR
        CP '!'
        JP NZ,L1931
        RST 28H
        DEFW NEXT_CHAR
        CALL L18C0
        LD A,$02
        RST 28H
        DEFW CHAN_OPEN
        LD (OLDSP),SP
        LD SP,TSTACK
        CALL L20F1
        LD A,$05
        CALL L1C83
        LD SP,(OLDSP)
        RET
```

Get the current character.  
 \$0018. [Could just do RST \$18 here]  
 \$21. Is it '!'?  
 Jump to "C Nonsense in BASIC" if not.  
 Get the next character.  
 \$0020. [Could just do RST \$20 here]  
 Check for end of statement.  
 Select main screen.  
 \$1601.  
 \$5B81. Store SP.  
 \$5BFF. Use temporary stack.  
 Print out the catalogue.  
 Page in logical RAM bank 5 (physical RAM bank 0).  
 \$5B81. Use original stack.

## ERASE! Routine

```
L1C2B:  RST 28H
        DEFW GET_CHAR
        CP '!'
        JP NZ,L1931
        CALL L13B2
        CALL L18C0
        LD (OLDSP),SP
        LD SP,TSTACK
        CALL L1F7E
        LD A,$05
        CALL L1C83
        LD SP,(OLDSP)
        RET
```

Get character from BASIC line.  
 \$0018.  
 \$21. Is it '!'?  
 Jump to "C Nonsense in BASIC" if not.  
 Get the filename into N\_STR1.  
 Make sure we've reached the end of the BASIC statement.  
 \$5B81. Store SP.  
 \$5BFF. Use temporary stack.  
 Do the actual erasing (leaves logical RAM bank 4 paged in).  
 Restore RAM configuration.  
 Page in logical RAM bank 5 (physical RAM bank 0).  
 \$5B81. Use original stack.

## RAM DISK COMMAND ROUTINES — PART 2

### Load Header from RAM Disk

```
L1C4D:  LD (OLDSP),SP
        LD SP,TSTACK
        CALL L1D54
```

\$5B81. Store SP.  
 \$5BFF. Use temporary stack.  
 Find file (return details pointed to by IX). Leaves logical RAM bank 4 paged in.

The file exists else the call above would have produced an error "h file does not exist"

LD HL,HD_00	\$5B71. Load 9 header bytes.
LD BC,\$0009	
CALL L1E56	Load bytes from RAM disk.
LD A,\$05	Restore RAM configuration.
CALL L1C83	Page in logical RAM bank 5 (physical RAM bank 0).
LD SP,(OLDSP)	\$5B81. Use original stack.
RET	

## Load from RAM Disk

Used by LOAD, VERIFY and MERGE. Note that VERIFY will simply perform a LOAD.

Entry: HL=Destination address.  
 DE=Length (will be greater than zero).  
 IX=File descriptor.  
 IX=Address of catalogue entry (IX+\$10-IX+\$12 points to the address of the file's data, past its header).  
 HD\_00-HD\_11 holds file header information.

L1C6A:	LD (OLDSP),SP	\$5B81. Store SP
	LD SP,TSTACK	\$5BFF. Use temporary stack.
	LD B,D	
	LD C,E	BC=Length.
	CALL L1E56	Load bytes from RAM disk.
	CALL L1D75	Update catalogue entry (leaves logical RAM bank 4 paged in).
	LD A,\$05	Restore RAM configuration.
	CALL L1C83	Page in logical RAM bank 5 (physical RAM bank 0).
	LD SP,(OLDSP)	\$5B81. Use original stack.
	RET	

## PAGING ROUTINES — PART 1

### Page Logical RAM Bank

This routine converts between logical and physical RAM banks and pages the selected bank in.

Entry: A=Logical RAM bank.

L1C83:	PUSH HL	Save BC and HL.
	PUSH BC	
	LD HL,L1CA0	Physical banks used by RAM disk.
	LD B,\$00	
	LD C,A	BC=Logical RAM bank.
	ADD HL,BC	Point to table entry.
	LD C,(HL)	Look up physical page.
	DI	Disable interrupts whilst paging.
	LD A,(BANK_M)	\$5B5C. Fetch the current configuration.
	AND \$F8	Mask off current RAM bank.
	OR C	Include new RAM bank.
	LD (BANK_M),A	\$5B5C. Store the new configuration.
	LD BC,\$7FFD	
	OUT (C),A	Perform the page.
	EI	Re-enable interrupts.
	POP BC	Restore BC and HL.
	POP HL	
	RET	

### Physical RAM Bank Mapping Table

L1CA0:	DEFB \$01	Logical bank \$00.
--------	-----------	--------------------

DEFB \$03  
 DEFB \$04  
 DEFB \$06  
 DEFB \$07  
 DEFB \$00

Logical bank \$01.  
 Logical bank \$02.  
 Logical bank \$03.  
 Logical bank \$04.  
 Logical bank \$05.

## RAM DISK COMMAND ROUTINES — PART 3

### Compare Filenames

Compare filenames at N\_STR1 and IX.

Exit: Zero flag set if filenames match.

Carry flag set if filename at DE is alphabetically lower than filename at IX.

L1CA6: LD DE,N\_STR1 \$5B67.

Compare filenames at DE and IX

L1CA9: PUSH IX  
 POP HL  
 LD B,\$0A Maximum of 10 characters.  
 L1CAE: LD A,(DE)  
 INC DE  
 CP (HL) compare each character.  
 INC HL  
 RET NZ Return if characters are different.  
 DJNZ L1CAE Repeat for all characters of the filename.  
 RET

### Create New Catalogue Entry

Add a catalogue entry with filename contained in N\_STR1.

Exit: HL=Address of next free catalogue entry.

IX=Address of newly created catalogue entry.

L1CB6: CALL L1D31 Find entry in RAM disk area, returning IX pointing to catalogue entry (leaves logical RAM bank 4 paged in).  
 JR Z,L1CBF Jump ahead if does not exist.  
 CALL L05CB Produce error report.  
 DEFB \$20 "e File already exists"  
 L1CBF: PUSH IX  
 LD BC,\$3FEC 16384-20 (maximum size of RAM disk catalogue).  
 ADD IX,BC IX grows downwards as new RAM disk catalogue entries added. If adding the maximum size to IX does not result in the carry flag being set then the catalogue is full, so issue an error report "4 Out of Memory".  
 POP IX  
 JR NC,L1D2D Jump if out of memory.  
 LD HL,\$FFEC -20 (20 bytes is the size of a RAM disk catalogue entry).  
 LD A,\$FF Extend the negative number into the high byte.  
 CALL L1D12 Ensure space in RAM disk area.  
 LD HL,FLAGS3 \$5B66.  
 SET 2,(HL) Signal editing RAM disk catalogue.  
 PUSH IX  
 POP DE DE=Address of new catalogue entry.  
 LD HL,N\_STR1 \$5B67. Filename.  
 L1CDD: LD BC,\$000A 10 characters in the filename.  
 LDIR Copy the filename.  
 SET 0,(IX+\$13) Indicate catalogue entry requires updating.  
 LD A,(IX+\$0A) Set the file access address to be the  
 LD (IX+\$10),A start address of the file.  
 LD A,(IX+\$0B)  
 LD (IX+\$11),A

LD A,(IX+\$0C)	
LD (IX+\$12),A	
XOR A	Set the fill length to zero.
LD (IX+\$0D),A	
LD (IX+\$0E),A	
LD (IX+\$0F),A	
LD A,\$05	
CALL L1C83	Logical RAM bank 5 (physical RAM bank 0).
PUSH IX	
POP HL	HL=Address of new catalogue entry.
LD BC,\$FFEC	-20 (20 bytes is the size of a catalogue entry).
ADD HL,BC	
LD (SFNEXT),HL	\$5B83. Store address of next free catalogue entry.
RET	

## Adjust RAM Disk Free Space

Adjust the count of free bytes within the RAM disk.

The routine can produce "4 Out of memory" when adding.

Entry: AHL=Size adjustment (negative when a file added, positive when a file deleted).

A=Bit 7 set for adding data, else deleting data.

L1D12:	LD DE,(SFSPACE)	\$5B85.
	EX AF,AF'	A'HL=Requested space.
	LD A,(SFSPACE+2)	\$5B87. ADE=Free space on RAM disk.
	LD C,A	CDE=Free space.
	EX AF,AF'	AHL=Requested space.
	BIT 7,A	A negative adjustment, i.e. adding data?
	JR NZ,L1D29	Jump ahead if so.

Deleting data

	ADD HL,DE	
	ADC A,C	AHL=Free space left.
L1D22:	LD (SFSPACE),HL	\$5B85. Store free space.
	LD (SFSPACE+2),A	\$5B87.
	RET	

Adding data

L1D29:	ADD HL,DE	
	ADC A,C	
	JR C,L1D22	Jump back to store free space if space left.
L1D2D:	CALL L05CB	Produce error report.
	DEFB 03	"4 Out of memory"

## Find Catalogue Entry for Filename

L1D31:	LD A,\$04	Page in logical RAM bank 4 (physical RAM bank 7).
	CALL L1C83	
	LD IX,\$EBEC	Point to first catalogue entry.
L1D3A:	LD DE,(SFNEXT)	\$5B83. Pointer to last catalogue entry.
	OR A	Clear carry flag.
	PUSH IX	
	POP HL	HL=First catalogue entry.
	SBC HL,DE	
	RET Z	Return with zero flag set if end of catalogue reached and hence filename not found.
	CALL L1CA6	Test filename match with N_STR1 (\$5B67).
	JR NZ,L1D4D	Jump ahead if names did not match.
	OR \$FF	Reset zero flag to indicate filename exists.
	RET	
L1D4D:	LD BC,\$FFEC	-20 bytes (20 bytes is the size of a catalogue entry).

ADD IX,BC  
JR L1D3A

Point to the next directory entry.  
Test the next name.

## Find RAM Disk File

Find a file in the RAM disk matching name held in N\_STR1,  
and return with IX pointing to the catalogue entry.

L1D54:	CALL L1D31	Find entry in RAM disk area, returning IX pointing to catalogue entry (leaves logical RAM bank 4 paged in).
	JR NZ,L1D5D	Jump ahead if it exists.
	CALL L05CB	Produce error report.
	DEFB \$23	"h File does not exist"
L1D5D:	LD A,(IX+\$0A)	Take the current start address (bank + location)
	LD (IX+\$10),A	and store it as the current working address.
	LD A,(IX+\$0B)	
	LD (IX+\$11),A	
	LD A,(IX+\$0C)	
	LD (IX+\$12),A	
	LD A,\$05	Page in logical RAM bank 5 (physical RAM bank 0).
	CALL L1C83	
	RET	[Could have saved 1 byte by using JP \$1C83 (ROM 0)]

## Update Catalogue Entry

L1D75:	LD A,\$04	Page in logical RAM bank 4 (physical RAM bank 7).
	CALL L1C83	
	BIT 0,(IX+\$13)	
	RET Z	Ignore if catalogue entry does not require updating.
	RES 0,(IX+\$13)	Indicate catalogue entry updated.
	LD HL,FLAGS3	\$5B66.
	RES 2,(HL)	Signal not editing RAM disk catalogue.
	LD L,(IX+\$10)	Points to end address within logical RAM bank.
	LD H,(IX+\$11)	
	LD A,(IX+\$12)	Points to end logical RAM bank.
	LD E,(IX+\$0A)	Start address within logical RAM bank.
	LD D,(IX+\$0B)	
	LD B,(IX+\$0C)	Start logical RAM bank.
	OR A	Clear carry flag.
	SBC HL,DE	HL=End address-Start address. Maximum difference fits within 14 bits.
	SBC A,B	A=End logical RAM bank-Start logical RAM bank - 1 if addresses overlap.
	RL H	
	RL H	Work out how many full banks of 16K are being used.
	SRA A	Place this in the upper two bits of H.
	RR H	
	SRA A	
	RR H	HL=Total length.
	LD (IX+\$0D),L	Length within logical RAM bank.
	LD (IX+\$0E),H	
	LD (IX+\$0F),A	

Copy the end address of the previous entry into the new entry

LD L,(IX+\$10)	End address within logical RAM bank.
LD H,(IX+\$11)	
LD A,(IX+\$12)	End logical RAM bank.
LD BC,\$FFEC	-20 bytes (20 bytes is the size of a catalogue entry).
ADD IX,BC	Address of next catalogue entry.
LD (IX+\$0A),L	Start address within logical RAM bank.
LD (IX+\$0B),H	
LD (IX+\$0C),A	Start logical RAM bank.
RET	

## Save Bytes to RAM Disk

L1DCB:	LD A,B OR C RET Z PUSH HL LD DE,\$C000 EX DE,HL SBC HL,DE JR Z,L1DF4 JR C,L1DF4	Check whether a data length of zero was requested.  Ignore if so since all bytes already saved. Save the source address. DE=The start of the upper RAM bank. HL=The start of the RAM bank. DE=Source address. HL=RAM bank start - Source address. Jump ahead if saving bytes from \$C000. Jump ahead if saving bytes from an address above \$C000.
--------	---	--

Source is below \$C000

PUSH HL SBC HL,BC JR NC,L1DEB	HL=Distance below \$C000 (RAM bank start - Source address).  Jump if requested bytes are all below \$C000.
-------------------------------------	--

Source spans across \$C000

LD H,B LD L,C POP BC OR A SBC HL,BC EX (SP),HL LD DE,\$C000 PUSH DE JR L1E13	HL=Requested length. BC=Distance below \$C000.  HL=Bytes occupying upper RAM bank. Stack it. HL=Source address. Start of upper RAM bank.  Jump forward.
--	--

Source fits completely below upper RAM bank (less than \$C000)

L1DEB:	POP HL POP HL LD DE,\$0000 PUSH DE PUSH DE JR L1E13	Forget the 'distance below \$C000' count. HL=Source address. Remaining bytes to transfer.  Stack dummy Start of upper RAM bank. Jump forward.
--------	--	--

Source fits completely within upper RAM bank (greater than or equal \$C000)

L1DF4:	LD H,B LD L,C LD DE,\$0020 OR A SBC HL,DE JR C,L1E03	HL=Requested length. DE=Length of buffer.  HL=Requested length-Length of buffer = Buffer overspill. Jump if requested length will fit within the buffer.
--------	---	--

Source spans transfer buffer

EX (SP),HL LD B,D LD C,E JR L1E08	Stack buffer overspill. HL=\$0000.  BC=Buffer length. Jump forward.
--	--

Source fits completely within transfer buffer

L1E03:	POP HL LD DE,\$0000 PUSH DE	HL=Destination address. Remaining bytes to transfer. Stack 'transfer buffer in use' flag.
--------	-----------------------------------	---

Transfer a block

## SPECTRUM +2 ROM 0 DISASSEMBLY

L1E08:	PUSH BC LD DE,STRIP1 LDIR POP BC PUSH HL LD HL,STRIP1	Stack the length. \$5B98. Transfer buffer. Transfer bytes. BC=Length. HL=New source address. \$5B98. Transfer buffer.
L1E13:	LD A,\$04 CALL L1C83 LD E,(IX+\$10) LD D,(IX+\$11) LD A,(IX+\$12) CALL L1C83	Page in logical RAM bank 4 (physical RAM bank 7).  Fetch the address from the current logical RAM bank. Logical RAM bank. Page in appropriate logical RAM bank.
L1E24:	LDI LD A,D OR E JR Z,L1E43	Transfer a byte from the file to the required RAM disk location or transfer buffer.  Has DE been incremented to \$0000? Jump if end of RAM bank reached.
L1E2A:	LD A,B OR C JP NZ,L1E24 LD A,\$04 CALL L1C83 LD (IX+\$10),E LD (IX+\$11),D LD A,\$05 CALL L1C83 POP HL POP BC JR L1DCB	Repeat until all bytes transferred. Page in logical RAM bank 4 (physical RAM bank 7).  Store the next RAM bank source address. Page in logical RAM bank 5 (physical RAM bank 0).  HL=Source address. BC=Length. Re-enter this routine to transfer another block.

The end of a RAM bank has been reached so switch to the next bank

L1E43:	LD A,\$04 CALL L1C83 INC (IX+\$12) LD A,(IX+\$12) LD DE,\$C000 CALL L1C83 JR L1E2A	Page in logical RAM bank 4 (physical RAM bank 7).  Increment to the new logical RAM bank. Fetch the new logical RAM bank. The start of the RAM disk Page in next RAM bank. Jump back to transfer another block.
--------	--	---

## Load Bytes from RAM Disk

Used for loading file header and data.

Entry: IX=RAM disk catalogue entry address. IX+\$10-IX+\$12 points to the next address to fetch from the file.  
HL=Destination address.  
BC=Requested length.

L1E56:	LD A,B OR C RET Z PUSH HL LD DE,\$C000 EX DE,HL SBC HL,DE JR Z,L1E86 JR C,L1E86	Check whether a data length of zero was requested.  Ignore if so since all bytes already loaded. Save the destination address. DE=The start of the upper RAM bank. HL=The start of the RAM bank. DE=Destination address. HL=RAM bank start - Destination address. Jump if destination is \$C000. Jump if destination is above \$C000.
--------	---	---

Destination is below \$C000

L1E64:	PUSH HL SBC HL,BC JR NC,L1E7B	HL=Distance below \$C000 (RAM bank start - Destination address).  Jump if requested bytes all fit below \$C000.
--------	-------------------------------------	---

Code will span across \$C000

## SPECTRUM +2 ROM o DISASSEMBLY

LD H,B	
LD L,C	HL=Requested length.
POP BC	BC=Distance below \$C000.
OR A	
SBC HL,BC	HL=Bytes destined for upper RAM bank.
EX (SP),HL	Stack it. HL=Destination address.
LD DE,\$0000	Remaining bytes to transfer.
PUSH DE	
LD DE,\$C000	Start of upper RAM bank.
PUSH DE	
EX DE,HL	HL=Start of upper RAM bank.
JR L1E9F	Jump forward.

Code fits completely below upper RAM bank (less than \$C000)

L1E7B:	POP HL	Forget the 'distance below \$C000' count.
	POP HL	HL=Destination address.
	LD DE,\$0000	Remaining bytes to transfer.
	PUSH DE	
	PUSH DE	Stack dummy Start of upper RAM bank.
	PUSH DE	
	EX DE,HL	HL=\$0000, DE=Destination address.
	JR L1E9F	Jump forward.

Code destined for upper RAM bank (greater than or equal to \$C000)

L1E86:	LD H,B	
	LD L,C	HL=Requested length.
	LD DE,\$0020	DE=Length of buffer.
	OR A	
	SBC HL,DE	HL=Requested length-Length of buffer = Buffer overspill.
	JR C,L1E95	Jump if requested length will fit within the buffer.

Code will span transfer buffer

EX (SP),HL	Stack buffer overspill. HL=\$0000.
LD B,D	
LD C,E	BC=Buffer length.
JR L1E9A	Jump forward.

Code will all fit within transfer buffer

L1E95:	POP HL	HL=Destination address.
	LD DE,\$0000	Remaining bytes to transfer.
	PUSH DE	Stack 'transfer buffer in use' flag.
L1E9A:	PUSH BC	Stack the length.
	PUSH HL	Stack destination address.
	LD DE,STRIP1	\$5B98. Transfer buffer.

Transfer a block

L1E9F:	LD A,\$04	Page in logical RAM bank 4 (physical RAM bank 7).
	CALL L1C83	
	LD L,(IX+\$10)	RAM bank address.
	LD H,(IX+\$11)	
	LD A,(IX+\$12)	Logical RAM bank.
	CALL L1C83	Page in appropriate logical RAM bank.

Enter a loop to transfer BC bytes, either to required destination or to the transfer buffer

L1EB0:	LDI	Transfer a byte from the file to the required location or transfer buffer.
	LD A,H	
	OR L	Has HL been incremented to \$0000?
	JR Z,L1EDB	Jump if end of RAM bank reached.

L1EB6:	LD A,B OR C JP NZ,L1EB0 LD A,\$04 CALL L1C83 LD (IX+\$10),L LD (IX+\$11),H LD A,\$05 CALL L1C83 POP DE POP BC LD HL,STRIP1 LD A,B OR C JR Z,L1ED6 LDIR	Repeat until all bytes transferred. Page in logical RAM bank 4 (physical RAM bank 7).  Store the next RAM bank destination address. Page in logical RAM bank 5 (physical RAM bank 0).  DE=Destination address. BC=Length. \$5B98. Transfer buffer.
L1ED6:	EX DE,HL POP BC JP L1E56	All bytes transferred? Jump forward if so. Transfer code in buffer to the required address. HL=New destination address. BC=Remaining bytes to transfer. Re-enter this routine to transfer another block.

The end of a RAM bank has been reached so switch to the next bank

L1EDB:	LD A,\$04 CALL L1C83 INC (IX+\$12) LD A,(IX+\$12) LD HL,\$C000 CALL L1C83 JR L1EB6	Page in logical RAM bank 4 (physical RAM bank 7).  Increment to the new logical RAM bank. Fetch the new logical RAM bank. The start of the RAM disk. Page in next logical RAM bank. Jump back to transfer another block.
--------	--	--

## Transfer Bytes to RAM Bank 4 — Vector Table Entry

This routine can be used to transfer bytes from the current RAM bank into logical RAM bank 4.

It is not used in this ROM and is a remnant of the original Spanish Spectrum 128 ROM 0.

Entry: HL=Source address in conventional RAM.  
DE=Destination address in logical RAM bank 4 (physical RAM bank 7).  
BC=Number of bytes to save.

L1EEE:	PUSH AF LD A,(BANK_M) PUSH AF PUSH HL PUSH DE PUSH BC LD IX,N_STR1+3 LD (IX+\$10),E LD (IX+\$11),D LD (IX+\$12),\$04 CALL L1DCB	Save AF. \$5B5C. Fetch current physical RAM bank configuration. Save it. Save source address. Save destination address. Save length. \$5B6A. Store destination address as the current address pointer.  Destination is in logical RAM bank 4 (physical RAM bank 7). Store bytes to RAM disk.
--------	---	--

Entered here by load vector routine

L1F07:	LD A,\$05 CALL L1C83 POP BC POP DE POP HL ADD HL,BC EX DE,HL ADD HL,BC EX DE,HL POP AF LD BC,\$7FFD DI	Page in logical RAM bank 5 (physical RAM bank 0).  Get length. Get destination address. Get source address. HL=Address after end of source. DE=Address after end of source. HL=Destination address. HL=Address after end of destination. HL=Address after end of source. DE=Address after end of destination. Get original RAM bank configuration.  Disable interrupts whilst paging.
--------	---	--

OUT (C),A	
LD (BANK_M),A	\$5B5C.
EI	Re-enable interrupts.
LD BC,\$0000	Signal all bytes loaded/saved.
POP AF	Restore AF.
RET	

## Transfer Bytes from RAM Bank 4 — Vector Table Entry

This routine can be used to transfer bytes from logical RAM bank 4 into the current RAM bank.

It is not used in this ROM and is a remnant of the original Spanish Spectrum 128 ROM 0.

Entry: HL=Source address in logical RAM bank 4 (physical RAM bank 7).  
 DE=Destination address in current RAM bank.  
 BC=Number of bytes to load.

L1F23:	PUSH AF	Save AF.
	LD A,(BANK_M)	\$5B5C. Fetch current physical RAM bank configuration.
	PUSH AF	Save it.
	PUSH HL	Save source address.
	PUSH DE	Save destination address.
	PUSH BC	Save length.
	LD IX,N_STR1+3	\$5B6A.
	LD (IX+\$10),L	Store source address as the current address pointer.
	LD (IX+\$11),H	
	LD (IX+\$12),\$04	Source is in logical RAM bank 4 (physical RAM bank 7).
	EX DE,HL	HL=Destination address.
	CALL L1E56	Load bytes from RAM disk.
	JR L1F07	Join the save vector routine above.

## PAGING ROUTINES — PART 2

### Use Normal RAM Configuration

Page in physical RAM bank 0, use normal stack and stack TARGET address.

Entry: HL=TARGET address.

L1F3F:	EX AF,AF'	Save AF.
	LD A,\$00	Physical RAM bank 0.
	DI	Disable interrupts whilst paging.
	CALL L1F59	Page in physical RAM bank 0.
	POP AF	AF=Address on stack when CALLEd.
	LD (TARGET),HL	\$5B58. Store HL.
	LD HL,(OLDSP)	\$5B81. Fetch the old stack.
	LD (OLDSP),SP	\$5B81. Save the current stack.
	LD SP,HL	Use the old stack.
	EI	Re-enable interrupts.
	LD HL,(TARGET)	\$5B58. Restore HL.
	PUSH AF	Re-stack the return address.
	EX AF,AF'	Get AF back.
	RET	

### Select RAM Bank

Used twice by the ROM to select either physical RAM bank 0 or physical RAM bank 7.

However, it could in theory also be used to set other paging settings.

Entry: A=RAM bank number.

L1F59:	PUSH BC	Save BC
	LD BC,\$7FFD	
	OUT (C),A	Perform requested paging.

LD (BANK\_M),A  
POP BC  
RET

\$5B5C.  
Restore BC.

## Use Workspace RAM Configuration

Page in physical RAM bank 7, use workspace stack and stack TARGET address.

Entry: HL=TARGET address.

L1F64:	EX AF,AF'	Save A.
	DI	Disable interrupts whilst paging.
	POP AF	Fetch return address.
	LD (TARGET),HL	\$5B58. Store HL.
	LD HL,(OLDSP)	\$5B81. Fetch the old stack.
	LD (OLDSP),SP	\$5B81. Save the current stack.
	LD SP,HL	Use the old stack.
	LD HL,(TARGET)	\$5B58. Restore HL.
	PUSH AF	Stack return address.
	LD A,\$07	RAM bank 7.
	CALL L1F59	Page in RAM bank 7.
	EI	Re-enable interrupts.
	EX AF,AF'	Restore A.
	RET	

## RAM DISK COMMAND ROUTINES — PART 4

### Erase a RAM Disk File

N\_STR1 contains the name of the file to erase.

L1F7E:	CALL L1D31	Find entry in RAM disk area, returning IX pointing to catalogue entry (leaves logical RAM bank 4 paged in).
	JR NZ,L1F87	Jump ahead if it was found. [Could have saved 3 bytes by using JP Z,\$1D5D (ROM 0)]
	CALL L05CB	Produce error report.
	DEFB \$23	"h File does not exist"
L1F87:	LD L,(IX+\$0D)	AHL=Length of file.
	LD H,(IX+\$0E)	
	LD A,(IX+\$0F)	Bit 7 of A will be 0 indicating to delete rather than add.
	CALL L1D12	Free up this amount of space.
	PUSH IY	Preserve current value of IY.
	LD IY,(SFNEXT)	\$5B83. IY points to next free catalogue entry.
	LD BC,\$FFEC	BC=-20 (20 bytes is the size of a catalogue entry).
	ADD IX,BC	IX points to the next catalogue entry
	LD L,(IY+\$0A)	AHL=First spare byte in RAM disk file area.
	LD H,(IY+\$0B)	
	LD A,(IY+\$0C)	
	POP IY	Restore IY to normal value.
	LD E,(IX+\$0A)	BDE=Start of address of next RAM disk file entry.
	LD D,(IX+\$0B)	
	LD B,(IX+\$0C)	
	OR A	
	SBC HL,DE	
	SBC A,B	
	RL H	
	RL H	
	SRA A	
	RR H	
	SRA A	
	RR H	HL=Length of all files to be moved.
	LD BC,\$0014	20 bytes is the size of a catalogue entry.
	ADD IX,BC	IX=Catalogue entry to delete.
	LD (IX+\$10),L	Store file length in the 'deleted' catalogue entry.

## SPECTRUM +2 ROM 0 DISASSEMBLY

LD (IX+\$11),H LD (IX+\$12),A LD BC,\$FFEC ADD IX,BC LD L,(IX+\$0A) LD H,(IX+\$0B) LD D,(IX+\$0C) LD BC,\$0014 ADD IX,BC LD A,D CALL L1C83 LD A,(BANK_M) LD E,A LD BC,\$7FFD LD A,\$07 DI OUT (C),A EXX LD L,(IX+\$0A) LD H,(IX+\$0B) LD D,(IX+\$0C) LD A,D CALL L1C83 LD A,(BANK_M) LD E,A LD BC,\$7FFD EXX	-20 (20 bytes is the size of a catalogue entry). IX=Next catalogue entry. DHL=Start address of next RAM disk file entry.  20 bytes is the size of a catalogue entry. IX points to catalogue entry to delete. Page in logical RAM bank for start address of entry to delete.  \$5B5C. Save current RAM bank configuration in E. Select physical RAM bank 7.  Disable interrupts whilst performing paging operations. Page in selected RAM bank. DHL'=Start address of next RAM disk file entry. DHL=Start of address of RAM disk file entry to delete.  Page in logical RAM bank for file entry (will update BANK_M). \$5B5C. Get RAM bank configuration for the file in E.  DHL=Start address of next RAM disk file entry.
--	---

At this point we have the registers and alternate registers pointing to the actual bytes in the RAM disk for the file to be deleted and the next file, with length bytes of the catalogue entry for the file to be deleted containing the length of bytes for all subsequent files that need to be moved down in memory. A loop is entered to move all of these bytes where the delete file began.

DHL holds the address of the byte to be moved.

E contains the value which should be OUTed to \$5B5C to page in the relevant RAM page.

L2009:	LD A,\$07 DI OUT (C),A LD A,(IX+\$10) SUB \$01 LD (IX+\$10),A JR NC,L202C LD A,(IX+\$11) SUB \$01 LD (IX+\$11),A JR NC,L202C LD A,(IX+\$12) SUB \$01 LD (IX+\$12),A JR C,L205D	Select physical RAM bank 7. Disable interrupts whilst performing paging operations. Page in selected RAM bank. Decrement end address.  If no carry then the decrement is finished. Otherwise decrement the middle byte.  If no carry then the decrement is finished. Otherwise decrement the highest byte.
L202C:	OUT (C),E LD A,(HL) INC L JR NZ,L2043 INC H JR NZ,L2043 EX AF,AF' INC D LD A,D CALL L1C83 LD A,(BANK_M) LD E,A LD HL,\$C000 EX AF,AF'	Jump forward if finished moving the file. Page in RAM bank containing the next file. Get the byte from the next file. Increment DHL. If not zero then the increment is finished. Otherwise increment the middle byte. If not zero then the increment is finished. Save the byte read from the next file. Advance to next logical RAM bank for the next file.  Page in next logical RAM bank for next file entry (will update BANK_M). \$5B5C. Get RAM bank configuration for the next file in E. The next file continues at the beginning of the next RAM bank. Retrieve the byte read from the next file.
L2043:	EXX DI OUT (C),E	DHL=Address of file being deleted. Disable interrupts whilst performing paging operations. Page in next RAM bank containing the next file.

# SPECTRUM +2 ROM o DISASSEMBLY

	LD (HL),A INC L JR NZ,L205A INC H JR NZ,L205A INC D LD A,D CALL L1C83 LD A,(BANK_M) LD E,A LD HL,\$C000 L205A: EXX JR L2009	Store the byte taken from the next file. Increment DHL. If not zero then the increment is finished. Otherwise increment the middle byte. If not zero then the increment is finished. Advance to next logical RAM bank for the file being deleted.  Page in next logical RAM bank for file being deleted entry (will update BANK_M). \$5B5C. Get RAM bank configuration for the file being deleted in E. The file being deleted continues at the beginning of the next RAM bank. DHL=Address of byte in next file. DHL'=Address of byte in file being deleted.
The file has been moved		
L205D:	LD A,\$04 CALL L1C83 LD A,\$00 LD HL,\$0014	Page in logical RAM bank 4 (physical RAM bank 7).
L2067:	CALL L1D12 LD E,(IX+\$0D) LD D,(IX+\$0E) LD C,(IX+\$0F) LD A,D RLCA RL C RLCA RL C LD A,D AND \$3F LD D,A PUSH IX	AHL=20 bytes is the size of a catalogue entry. Delete a catalogue entry.  CDE=File length of file entry to delete.
L2080:	PUSH DE LD DE,\$FFEC ADD IX,DE POP DE LD L,(IX+\$0A) LD H,(IX+\$0B) LD A,(IX+\$0C) OR A SBC HL,DE SUB C BIT 6,H JR NZ,L209B SET 6,H DEC A	C=RAM bank.  Mask off upper bits to leave length in this bank (range 0-16383). DE=Length in this bank. Save address of catalogue entry to delete.  -20 (20 bytes is the size of a catalogue entry). Point to next catalogue entry. DE=Length in this bank.
L209B:	LD (IX+\$0A),L LD (IX+\$0B),H LD (IX+\$0C),A LD L,(IX+\$10) LD H,(IX+\$11) LD A,(IX+\$12) OR A SBC HL,DE SUB C BIT 6,H JR NZ,L20B8 SET 6,H DEC A	AHL=File start address.  Will move into next RAM bank?  Jump if same RAM bank. New address in next RAM bank. Next RAM bank.
L20B8:	LD (IX+\$10),L LD (IX+\$11),H LD (IX+\$12),A PUSH IX POP HL PUSH DE	Save new start address of file.  Fetch end address of file.  Will move into next RAM bank?  Jump if same RAM bank. New address in next RAM bank. Next RAM bank.  Save new end address of file.  HL=Address of next catalogue entry.

LD DE,(SFNEXT)	\$5B83.
OR A	
SBC HL,DE	End of catalogue reached?
POP DE	DE=Length in this bank.
JR NZ,L2080	Jump if not to move next entry.
LD DE,(SFNEXT)	\$5B83. Start address of the next available catalogue entry.
POP HL	
PUSH HL	HL=Start address of catalogue entry to delete.
OR A	
SBC HL,DE	
LD B,H	
LD C,L	BC=Length of catalogue entries to move.
POP HL	
PUSH HL	HL=Start address of catalogue entry to delete.
LD DE,\$0014	20 bytes is the size of a catalogue entry.
ADD HL,DE	HL=Start address of previous catalogue entry.
EX DE,HL	DE=Start address of previous catalogue entry.
POP HL	HL=Start address of catalogue entry to delete.
DEC DE	DE=End address of catalogue entry to delete.
DEC HL	HL=End address of next catalogue entry.
LDDR	Move all catalogue entries.
LD HL,(SFNEXT)	\$5B83. Start address of the next available catalogue entry.
LD DE,\$0014	20 bytes is the size of a catalogue entry.
ADD HL,DE	
LD (SFNEXT),HL	\$5B83. Store the new location of the next available catalogue entry.
RET	

## Print RAM Disk Catalogue

This routine prints catalogue filenames in alphabetically order.

It does this by repeatedly looping through the catalogue to find the next 'highest' name.

L20F1:	LD A,\$04	Page in logical RAM bank 4
	CALL L1C83	(physical RAM bank 7)
	LD HL,L2140	HL points to ten \$00 bytes, the initial comparison filename.
L20F9:	LD BC,L214A	BC point to ten \$FF bytes.
	LD IX,\$EBEC	IX points to first catalogue entry.
L2100:	CALL L05F5	Check for BREAK.
	PUSH IX	Save address of catalogue entry.
	EX (SP),HL	HL points to current catalogue entry. Top of stack points to ten \$00 data.
	LD DE,(SFNEXT)	\$5B83. Find address of next free catalogue entry.
	OR A	
	SBC HL,DE	Have we reached end of catalogue?
	POP HL	Fetch address of catalogue entry.
	JR Z,L2130	Jump ahead if end of catalogue reached.
	LD D,H	
	LD E,L	DE=Current catalogue entry.
	PUSH HL	
	PUSH BC	
	CALL L1CA9	Compare current filename (initially ten \$00 bytes).
	POP BC	
	POP HL	
	JR NC,L2129	Jump if current catalogue name is 'above' the previous.
	LD D,B	
	LD E,C	DE=Last filename
	PUSH HL	
	PUSH BC	
	CALL L1CA9	Compare current filename (initially ten \$FF bytes).
	POP BC	
	POP HL	
	JR C,L2129	Jump if current catalogue name is 'below' the previous.
	PUSH IX	
	POP BC	BC=Address of current catalogue entry name.
L2129:	LD DE,\$FFEC	-20 (20 bytes is the size of a catalogue entry).
	ADD IX,DE	Point to next catalogue entry.

## SPECTRUM +2 ROM 0 DISASSEMBLY

L2130:	JR L2100 PUSH HL LD HL,L214A OR A SBC HL,BC POP HL RET Z LD H,B LD L,C CALL L2154 JR L20F9	Check next filename. HL points to current catalogue entry. Address of highest theoretical filename data.  Was a new filename to print found?  Return if all filenames printed.  HL=Address of current catalogue entry name. Print the catalogue entry. Repeat for next filename.
--------	--	--

### Print Catalogue Filename Data

L2140:	DEFB \$00, \$00, \$00, \$00, \$00 DEFB \$00, \$00, \$00, \$00, \$00	Lowest theoretical filename.
L214A:	DEFB \$FF, \$FF, \$FF, \$FF, \$FF DEFB \$FF, \$FF, \$FF, \$FF, \$FF	Highest theoretical filename.

### Print Single Catalogue Entry

L2154:	PUSH HL PUSH BC POP HL LD DE,N_STR1 LD BC,\$000A LDIR LD A,\$05 CALL L1C83 LD HL,(OLDSP) LD (OLDSP),SP LD SP,HL LD HL,N_STR1 LD B,\$0A	Save address of filename.  [No need to transfer BC to HL since they already have the same value]. \$5B67. Copy the filename to N_STR1 so that it is visible when this RAM bank is paged out.  Page in logical RAM bank 5 (physical RAM bank 0).  \$5B81. \$5B81. Save temporary stack. Use original stack. \$5B67. HL points to filename. 10 characters to print.
L2171:	LD A,(HL) PUSH HL PUSH BC RST 28H DEFW PRINT_A_1 POP BC POP HL INC HL DJNZ L2171 LD A,\$0D RST 28H DEFW PRINT_A_1 RST 28H DEFW TEMPS LD HL,(OLDSP) LD (OLDSP),SP LD SP,HL LD A,\$04 CALL L1C83 POP HL RET	Print each character of the filename.  \$0010.  Print a newline character. \$0010. \$0D4D. Copy permanent colours to temporary colours. \$5B81. \$5B81. Save original stack. Switch back to temporary stack. Page in logical RAM bank 4 (physical RAM bank 7).  HL=Address of filename.

## BASIC LINE AND COMMAND INTERPRETATION ROUTINES — PART 4

### LPRINT Routine

L2193:	LD A,\$03	Printer channel.
	JR L2199	Jump ahead.

### PRINT Routine

L2197:	LD A,\$02	Main screen channel.
L2199:	RST 28H	
	DEFW SYNTAX_Z	\$2530.
	JR Z,L21A1	Jump forward if syntax is being checked.
	RST 28H	
	DEFW CHAN_OPEN	\$1601.
L21A1:	RST 28H	
	DEFW TEMPS	\$0D4D.
	RST 28H	
	DEFW PRINT_2	\$1FDF. Delegate handling to ROM 1.
	CALL L18C0	"C Nonsense in BASIC" during syntax checking if not at end of line or statement.
	RET	

### INPUT Routine

This routine allows for values entered from the keyboard to be assigned to variables. It is also possible to have print items embedded in the INPUT statement and these items are printed in the lower part of the display.

L21AB:	RST 28H	
	DEFW SYNTAX_Z	\$2530.
	JR Z,L21B8	Jump forward if syntax is being checked.
	LD A,\$01	Open channel 'K'.
	RST 28H	
	DEFW CHAN_OPEN	\$1601.
	RST 28H	Clear the lower part of the display.
	DEFW CLS_LOWER	\$0D6E. <b>[BUG]</b> - This call will re-select channel 'S' and so should have been called prior to opening channel 'K'. It is a direct copy of the code that appears in the standard Spectrum ROM (and ROM 1). It is debatable whether it is better to reproduce the bug so as to ensure that the INPUT routine operates the same in 128K mode as it does in 48K mode. Credit: Geoff Wearmouth]
L21B8:	LD (IY+\$02),\$01	TV_FLAG. Signal that the lower screen is being handled. [Not a bug as has been reported elsewhere. The confusion seems to have arisen due to the incorrect system variable being originally mentioned in the Spectrum ROM Disassembly by Logan and O'Hara]
	RST 28H	
	DEFW IN_ITEM_1	\$20C1. Call the subroutine to deal with the INPUT items.
	CALL L18C0	Move on to the next statement if checking syntax.
	RST 28H	
	DEFW INPUT_1+\$000A	\$20A0. Delegate handling to ROM 1.
	RET	

### COPY Routine

L21C6:	JP L090F	Jump to new COPY routine.
--------	----------	---------------------------

## NEW Routine

L21C9:           DI  
                  JP L019D                   Re-initialise the machine.

## CIRCLE Routine

This routine draws an approximation to the circle with centre co-ordinates X and Y and radius Z. These numbers are rounded to the nearest integer before use.

Thus Z must be less than 87.5, even when (X,Y) is in the centre of the screen.

The method used is to draw a series of arcs approximated by straight lines.

L21CD:           RST 18H                   Get character from BASIC line.  
                  CP ','                   \$2C. Check for second parameter.  
                  JR NZ,L220A             Jump ahead (for error C) if not.  
                  RST 20H                  Advance pointer into BASIC line.  
                  RST 28H                  Get parameter.  
                  DEFW EXPT\_1NUM          \$1C82. Radius to calculator stack.  
                  CALL L18C0              Move to consider next statement if checking syntax.  
                  RST 28H  
                  DEFW CIRCLE+\$000D       \$232D. Delegate handling to ROM 1.  
                  RET

## DRAW Routine

This routine is entered with the co-ordinates of a point X0, Y0, say, in COORDS. If only two parameters X, Y are given with the DRAW command, it draws an approximation to a straight line from the point X0, Y0 to X0+X, Y0+Y.

If a third parameter G is given, it draws an approximation to a circular arc from X0, Y0 to X0+X, Y0+Y turning anti-clockwise through an angle G radians.

L21DD:           RST 18H                   Get current character.  
                  CP ','                   \$2C.  
                  JR Z,L21E9              Jump if there is a third parameter.  
                  CALL L18C0              Error C during syntax checking if not at end of line/statement.  
                  RST 28H  
                  DEFW LINE\_DRAW          \$2477. Delegate handling to ROM 1.  
                  RET  
L21E9:           RST 20H                   Get the next character.  
                  RST 28H  
                  DEFW EXPT\_1NUM          \$1C82. Angle to calculator stack.  
                  CALL L18C0              Error C during syntax checking if not at end of line/statement.  
                  RST 28H  
                  DEFW DR\_3\_PRMS+\$0007   \$2394. Delegate handling to ROM 1.  
                  RET

## DIM Routine

This routine establishes new arrays in the variables area. The routine starts by searching the existing variables area to determine whether there is an existing array with the same name. If such an array is found then it is 'reclaimed' before the new array is established. A new array will have all its elements set to zero if it is a numeric array, or to 'spaces' if it is an array of strings.

L21F4:           RST 28H                   Search to see if the array already exists.  
                  DEFW LOOK\_VARS          \$28B2.  
                  JR NZ,L220A             Jump if array variable not found.  
                  RST 28H  
                  DEFW SYNTAX\_Z          \$2530.  
                  JR NZ,L2206             Jump ahead during syntax checking.  
                  RES 6,C                  Test the syntax for string arrays as if they were numeric.  
                  RST 28H  
                  DEFW STK\_VAR            \$2996. Check the syntax of the parenthesised expression.  
                  CALL L18C0              Error when checking syntax unless at end of line/statement.

An 'existing array' is reclaimed.

L2206:	RST 28H DEFW D_RUN RET	\$2C15. Delegate handling to ROM 1.
--------	------------------------------	-------------------------------------

## Error Report C — Nonsense in BASIC

L220A:	CALL L05CB DEFB \$0B	Produce error report. "C Nonsense in BASIC"
--------	-------------------------	--

## Clear Screen Routine

Clear screen if it is not already clear.

L220E:	BIT 0,(IY+\$30) RET Z RST 28H DEFW CL_ALL RET	FLAGS2. Is the screen clear? Return if it is.  \$0DAF. Otherwise clear the whole display.
--------	---	--

## Evaluate Numeric Expression

This routine is called when a numerical expression is typed directly into the editor or calculator.

A numeric expression is any that begins with '(', '-' or '+', or is one of the function keywords, e.g. ABS, SIN, etc, or is the name of a numeric variable.

L2217:	LD HL,\$FFFE LD (\$5C45),HL	A line in the editing area is considered as line '-2'. PPC. Signal no current line number.
--------	--------------------------------	---

Check the syntax of the BASIC line

RES 7,(IY+\$01) CALL L22AD RST 28H DEFW SCANNING BIT 6,(IY+\$01) JR Z,L2259 RST 18H CP \$0D JR NZ,L2259	Indicate 'syntax checking' mode. Point to start of the BASIC command line.  \$24FB. Evaluate the command line. Is it a numeric value? Jump to produce an error if a string result. Get current character. Is it the end of the line? Jump if not to produce an error if not.
---	--

The BASIC line has passed syntax checking so now execute it

L2259:	SET 7,(IY+\$01) CALL L22AD LD HL,L0321 LD (SYNRET),HL RST 28H DEFW SCANNING BIT 6,(IY+\$01) JR Z,L2259 LD DE,LASTV LD HL,(\$5C65) LD BC,\$0005 OR A SBC HL,BC LDIR JP L225D	If so, indicate 'execution' mode. Point to start of the BASIC command line. Set up the error handler routine address. \$5B8B.  \$24FB. Evaluate the command line. Is it a numeric value? Jump to produce an error if a string result. \$5B8D. DE points to last calculator value. STKEND. The length of the floating point value.  HL points to value on top of calculator stack. Copy the value in the workspace to the top of the calculator stack. [Could have saved 1 byte by using a JR instruction]
L225D:	CALL L05CB DEFB \$19 LD A,\$0D	Produce error report. "Q Parameter error" Make it appear that 'Enter' has been pressed.

	CALL L228E	Process key press.
	LD BC,\$0001	
	RST 28H	
	DEFW BC_SPACES	\$0030. Create a byte in the workspace.
	LD (\$5C5B),HL	K_CUR. Address of the cursor.
	PUSH HL	Save it.
	LD HL,(\$5C51)	CURCHL. Current channel information.
	PUSH HL	Save it.
	LD A,\$FF	Channel 'R', the workspace.
	RST 28H	
	DEFW CHAN_OPEN	\$1601.
	RST 28H	
	DEFW PRINT_FP	\$2DE3. Print a floating point number to the workspace.
	POP HL	Get the current channel information address.
	RST 28H	
	DEFW CHAN_FLAG	\$1615. Set appropriate flags back for the old channel.
	POP DE	DE=Address of the old cursor position.
	LD HL,(\$5C5B)	K_CUR. Address of the cursor.
	AND A	
	SBC HL,DE	HL=Length of floating point number.
L2283:	LD A,(DE)	Fetch the character and make it appear to have been typed.
	CALL L228E	Process the key press.
	INC DE	
	DEC HL	Decrement floating point number character count.
	LD A,H	
	OR L	
	JR NZ,L2283	Repeat for all characters.
	RET	

## Process Key Press

L228E:	PUSH HL	Save registers.
	PUSH DE	
	CALL L1F64	Use Workspace RAM configuration (physical RAM bank 7).
	LD HL,\$EC0D	Editor flags.
	RES 3,(HL)	Reset 'line altered' flag
	PUSH AF	
	LD A,\$02	Main screen
	RST 28H	
	DEFW CHAN_OPEN	\$1601.
	POP AF	
	CALL L2688	Process key press.
	LD HL,\$EC0D	Editor flags.
	RES 3,(HL)	Reset 'line altered' flag
	CALL L1F3F	Use Normal RAM Configuration (physical RAM bank 0).
	POP DE	Restore registers.
	POP HL	
	RET	

## Find Start of BASIC Command

Point to the start of a typed in BASIC command  
and return first character in A.

L22AD:	LD HL,(\$5C59)	E_LINE. Get the address of command being typed in.
	DEC HL	
	LD (\$5C5D),HL	CH_ADD. Store it as the address of next character to be interpreted.
	RST 20H	Get the next character.
	RET	

## Is LET Command?

A typed in command resides in the editing workspace.

This function tests whether the text is a single LET command.

Exit: Zero flag set if a single LET command.

L22B6:	CALL L22AD	Point to start of typed in command.
	CP \$F1	Is it 'LET'?
	RET NZ	Return if not with zero flag reset.
	LD HL,(\$5C5D)	CH_ADD. HL points to next character.
L22BF:	LD A,(HL)	Fetch next character.
	INC HL	
	CP \$0D	Has end of line been found?
	RET Z	Return if so with zero flag set.
	CP ':'	\$3A. Has start of new statement been found?
	JR NZ,L22BF	Loop back if not.
	OR A	Return zero flag reset indicating a multi-statement
	RET	LET command.

## Is Operator Character?

Exit: Zero flag set if character is an operator.

L22CA:	LD B,A	Save B.
	LD HL,L22DC	Start of operator token table.
L22CE:	LD A,(HL)	Fetch character from the table.
	INC HL	Advance to next entry.
	OR A	End of table?
	JR Z,L22D8	Jump if end of table reached.
	CP B	Found required character?
	JR NZ,L22CE	Jump if not to try next character in table.

Found

LD A,B	Restore character to A.
RET	Return with zero flag set to indicate an operator.

Not found

L22D8:	OR \$FF	Reset zero flag to indicate not an operator.
	LD A,B	Restore character to A.
	RET	

## Operator Tokens Table

L22DC:	DEFB \$2B, \$2D, \$2A	'+', '-', '**'
	DEFB \$2F, \$5E, \$3D	'/', '^', '='
	DEFB \$3E, \$3C, \$C7	'>', '<', '<='
	DEFB \$C8, \$C9, \$C5	'>=', '<>', 'OR'
	DEFB \$C6	'AND'
	DEFB \$00	End marker.

## Is Function Character?

Exit: Zero set if a function token.

L22EA:	CP \$A5	'RND'. (first 48K token)
	JR C,L22FC	Jump ahead if not a token with zero flag reset.
	CP \$C4	'BIN'.
	JR NC,L22FC	Jump ahead if not a function token.

	CP \$AC	'AT'.
	JR Z,L22FC	Jump ahead if not a function token.
	CP \$AD	'TAB'.
	JR Z,L22FC	Jump ahead if not a function token.
	CP A	Return zero flag set if a function token.
	RET	
L22FC:	CP \$A5	Return zero flag set if a function token.
	RET	

## Is Numeric or Function Expression?

Exit: Zero flag set if a numeric or function expression.

L22FF:	LD B,A	Fetch character code.
	OR \$20	Make lowercase.
	CP 'a'	\$61. Is it 'a' or above?
	JR C,L230C	Jump ahead if not a letter.
	CP '{'	\$7B. Is it below '{'?
L2308:	JR NC,L230C	Jump ahead if not.
	CP A	Character is a letter so return
	RET	with zero flag set.
L230C:	LD A,B	Fetch character code.
	CP ','	\$2E. Is it ','?
	RET Z	Return zero flag set indicating numeric.
	CALL L2329	Is character a number?
	JR NZ,L2326	Jump ahead if not a number.
L2315:	RST 20H	Get next character.
	CALL L2329	Is character a number?
	JR Z,L2315	Repeat for next character if numeric.
	CP ','	\$2E. Is it ','?
	RET Z	Return zero flag set indicating numeric.
	CP 'E'	\$45. Is it 'E'?
	RET Z	Return zero flag set indicating numeric.
	CP 'e'	\$65. Is it 'e'?
	RET Z	Return zero flag set indicating numeric.
	JR L22CA	Jump to test for operator tokens.
L2326:	OR \$FF	Reset the zero flag to indicate non-alphanumeric.
	RET	

## Is Numeric Character?

Exit: Zero flag set if numeric character.

L2329:	CP '0'	\$30. Is it below '0'?
	JR C,L2333	Jump below '0'.
	CP ':'	\$3A. Is it below ':'?
	JR NC,L2333	Jump above '9'
	CP A	
	RET	Set zero flag if numeric.
L2333:	CP '0'	\$30. This will cause zero flag to be reset.
	RET	

## PLAY Routine

L2336:	LD B,\$00	String index.
	RST 18H	
L2339:	PUSH BC	
	RST 28H	Get string expression.
	DEFW EXPT_EXP	
	POP BC	
	INC B	
	CP ','	\$2C. A ',' indicates another string.

	JR NZ,L2346	Jump ahead if no more.
	RST 20H	Advance to the next character.
	JR L2339	Loop back.
L2346:	LD A,B	Check the index.
	CP \$09	Maximum of 8 strings (to support synthesisers, drum machines or sequencers).
	JR C,L234F	
	CALL L05CB	Produce error report.
	DEFB \$2B	"p (c) 1986 Sinclair Research Ltd" <b>[BUG]</b> - This should be "Parameter error". The Spanish 128 produces "p Bad parameter" but to save memory perhaps the UK 128 was intended to use the existing "Q Parameter error" and the change of the error code byte here was overlooked. In that case it would have had a value of \$19. Note that generation of this error when using the main screen editor will result in a crash. Credit: Andrew Owen]
L234F:	CALL L18C0	Ensure end-of-statement or end-of-line.
	JP L09A4	Continue with PLAY code.

## UNUSED ROUTINES — PART 1

There now follows 513 bytes of routines that are not used by the ROM, from \$2355 (ROM 0) to \$2555 (ROM 0). They are remnants of the original Spanish 128's ROM code, although surprisingly they appear in a different order within that ROM.

### Return to Editor

[Never called by this ROM]

L2355:	LD HL,TSTACK	\$5BFF.
	LD (OLDSP),HL	\$5B81.
	CALL L1F64	Use Workspace RAM configuration (physical RAM bank 7).
	JP L25EA	Jump ahead to the Editor.

### BC=HL-DE, Swap HL and DE

Exit: BC=HL-DE.

DE=HL, HL=DE.

[Never called by this ROM]

L2361:	AND A	
	SBC HL,DE	
	LD B,H	
	LD C,L	BC=HL-DE.
	ADD HL,DE	
	EX DE,HL	HL=DE, DE=HL.
	RET	

### Create Room for 1 Byte

Creates a single byte in the workspace, or automatically produces an error '4' if not.

[Never called by this ROM]

L2369:	LD BC,\$0001	Request 1 byte.
	PUSH HL	
	PUSH DE	
	CALL L2377	Test whether there is space. If it fails this will cause the error handler in ROM 0 to be called. If MAKE_ROOM were called directly and
	POP DE	and out of memory condition detected then the ROM 1 error handler would
	POP HL	be called instead.
	RST 28H	\$1655. The memory check passed so safely make the room.
	DEFW MAKE_ROOM	
	RET	

## Room for BC Bytes?

Test whether there is room for the specified number of bytes in the spare memory, producing error "4 Out of memory" if not. This routine is very similar to that at \$3F66 with the exception that this routine assumes IY points at the system variables.

Entry: BC=Number of bytes required.

Exit : Returns if the room requested is available else an error '4' is produced.

[Called by the routine at \$2369 (ROM 0), which is itself never called by this ROM]

L2377:	LD HL,(\$5C65)	STKEND.
	ADD HL,BC	Would adding the specified number of bytes overflow the RAM area?
	JR C,L2387	Jump to produce an error if so.
	EX DE,HL	DE=New end address.
	LD HL,\$0082	Would there be at least 130 bytes at the top of RAM?
	ADD HL,DE	
	JR C,L2387	Jump to produce an error if not.
	SBC HL,SP	If the stack is lower in memory, would there still be enough room?
	RET C	Return if there would.
L2387:	LD (IY+\$00),\$03	Signal error "4 Out of Memory".
	JP L0321	Jump to error handler routine.

## HL = A\*32

[Called by routines at \$23A2 (ROM 0) and \$23D7 (ROM 0), which are themselves never called by this ROM]

L238E:	ADD A,A	A*2.
	ADD A,A	A*4. Then multiply by 8 in following routine.

## HL = A\*8

[Called by the routine at \$2400 (ROM 0), which ultimately is itself never called by this ROM]

L2390:	LD L,A	
	LD H,\$00	
	ADD HL,HL	A*2.
	ADD HL,HL	A*4.
	ADD HL,HL	A*8.
	RET	Return HL=A*8.

## Find Amount of Free Space

Exit: Carry flag set if no more space, else HL holds the amount of free space.

[Never called by this ROM]

L2397:	LD HL,\$0000	
	ADD HL,SP	HL=SP.
	LD DE,(\$5C65)	STKEND.
	OR A	
	SBC HL,DE	Effectively SP-STKEND, i.e. the amount of available space.
	RET	

## Print Screen Buffer Row

Prints row from the screen buffer to the screen.

Entry: A=Row number.

[Never called by this ROM]

L23A3:	RES 0,(IY-\$39)	KSTATE+1. Signal do not invert attribute value. [IY+\$3B on the Spanish 128]
	CALL L238E	HL=A*32. Number of bytes prior to the requested row.
	PUSH HL	Save offset to requested row to print.
	LD DE,(\$FF24)	Fetch address of screen buffer.

ADD HL,DE	Point to row entry.
LD D,H	
LD E,L	DE=Address of row entry.
EX (SP),HL	Stack address of row entry. HL=Offset to requested row to print.
PUSH HL	Save offset to requested row to print.
PUSH DE	Save address of row entry.
LD DE,\$5800	Attributes file.
ADD HL,DE	Point to start of corresponding row in attributes file.
EX DE,HL	DE=Start address of corresponding row in attributes file.
POP HL	HL=Address of row entry.
LD BC,\$0020	32 columns.
LD A,(\$5C8F)	ATTR_T. Fetch the temporary colours.
CALL L24BA	Set the colours for the 32 columns in this row, processing any colour control codes from the print string.
	HL=Offset to requested row to print.
POP HL	
LD A,H	
LD H,\$00	Calculate corresponding display file address.
ADD A,A	
ADD A,A	
ADD A,A	
ADD A,\$40	
LD D,A	
LD E,H	
ADD HL,DE	
EX DE,HL	DE=Display file address.
POP HL	HL=Offset to requested row to print.
LD B,\$20	32 columns.
JP L2400	Print one row to the display file.

## Blank Screen Buffer Content

Sets the specified number of screen buffer positions from the specified row to \$FF.

Entry: A=Row number.

BC=Number of bytes to set.

[Never called by this ROM]

L23D7:	LD D,\$FF	The character to set the screen buffer contents to.
	CALL L238E	HL=A*32. Offset to the specified row.
	LD A,D	
	LD DE,(\$FF24)	Fetch the address of the screen buffer.
	ADD HL,DE	HL=Address of first column in the requested row.
	LD E,L	
	LD D,H	
	INC DE	DE=Address of second column in the requested row.
	LD (HL),A	Store the character.
	DEC BC	
	LDIR	Repeat for all remaining bytes required.
	RET	

## Print Screen Buffer to Display File

[Never called by this ROM]

L23EA:	CALL L24A7	Set attributes file from screen buffer.
	LD DE,\$4000	DE=First third of display file.
	LD HL,(\$FF24)	Fetch address of screen buffer.
	LD B,E	Display 256 characters.
	CALL L2400	Display string.
	LD D,\$48	Middle third of display file.
	CALL L2400	Display string.
	LD D,\$50	Last third of display file.
	LD B,\$C0	Display 192 characters.

## Print Screen Buffer Characters to Display File

Displays ASCII characters, UDGs, graphic characters or two special symbols in the display file, but does not alter the attributes file. Character code \$FE is used to represent the error marker bug symbol and the character code \$FF is used to represent a null, which is displayed as a space.

Entry: DE=Display file address.  
HL=Points to string to print.  
B=Number of characters to print.

[Used by routine at \$23EA (ROM 0) and called by the routine at \$23A2 (ROM 0), both of which are themselves never called by this ROM]

L2400:	LD A,(HL)	Fetch the character.
	PUSH HL	Save string pointer.
	PUSH DE	Save display file address.
	CP \$FE	Was it \$FE (bug) or \$FF (null)?
	JR C,L240B	Jump ahead if not.
	SUB \$FE	Reduce range to \$00-\$01.
	JR L2441	Jump ahead to show symbol.

Comes here if character code if below \$FE

L240B:	CP \$20	Is it a control character?
	JR NC,L2416	Jump ahead if not.

Comes here if a control character

	LD HL,L2546	Graphic for a 'G' (not a normal G though). Used to indicate embedded colour control codes.
	AND A	Clear the carry flag to indicate no need to switch back to RAM bank 7.
	EX AF,AF'	Save the flag.
	JR L244A	Jump ahead to display the symbol.
L2416:	CP \$80	Is it a graphic character or UDG?
	JR NC,L2428	Jump ahead if so.

Comes here if an ASCII character

	CALL L2390	HL=A*8.
	LD DE,(\$5C36)	CHARS.
	ADD HL,DE	Point to the character bit pattern.
	POP DE	Fetch the display file address.
	CALL \$FF28	Copy character into display file (via RAM Routine). Can't use routine at \$244B (ROM 0) since it does not perform a simple return.
	JR L246F	Continue with next character.

Comes here if a graphic character or UDG

L2428:	CP \$90	Is it a graphic character?
	JR NC,L2430	Jump ahead if not.

Comes here if a graphic character

	SUB \$7F	Reduce range to \$01-\$10.
	JR L2441	Jump ahead to display the symbol.

Comes here if a UDG

L2430:	SUB \$90	Reduce range to \$00-\$6D.
	CALL L2390	HL=A*8.
	POP DE	Fetch display file address.
	CALL L1F3F	Use Normal RAM Configuration (RAM bank 0) to allow access to character bit patterns.
	PUSH DE	Save display file address.
	LD DE,(\$5C7B)	UDG. Fetch address of UDGs.
	SCF	Set carry flag to indicate need to switch back to RAM bank 7.
	JR L2448	Jump ahead to locate character bit pattern and display the symbol.

## SPECTRUM +2 ROM 0 DISASSEMBLY

Come here if (HL) was \$FE or \$FF, or with a graphic character.

At this point A=\$00 if (HL) was \$FE indicating a bug symbol, or \$01 if (HL) was \$FF indicating a null, or A=\$01-\$10 if a graphic character.

L2441:	LD DE,L254E	Start address of the graphic character bitmap table.
	CALL L2390	HL=A*8 -> \$0000 or \$0008.
	AND A	Clear carry flag to indicate no need to switch back to RAM bank 7.
L2448:	EX AF,AF'	Save switch bank indication flag.
	ADD HL,DE	Point to the symbol bit pattern data.
L244A:	POP DE	Fetch display file address. Drop through into routine below.

## Copy A Character « RAM Routine »

Routine copied to RAM at \$FF36-\$FF55 by subroutine at \$248E (ROM 0).

Also used in ROM from above routine.

This routine copies 8 bytes from HL to DE. It increments HL and D after each byte, restoring D afterwards.

It is used to copy a character into the display file.

Entry: HL=Character data.

DE=Display file address.

[Called by a routine that is itself never called by this ROM]

L244B:	LD C,D	Save D.
	LD A,(HL)	
	LD (DE),A	Copy byte 1.
	INC HL	
	INC D	
	LD A,(HL)	
	LD (DE),A	Copy byte 2.
	INC HL	
	INC D	
	LD A,(HL)	
	LD (DE),A	Copy byte 3.
	INC HL	
	INC D	
	LD A,(HL)	
	LD (DE),A	Copy byte 4.
	INC HL	
	INC D	
	LD A,(HL)	
	LD (DE),A	Copy byte 5.
	INC HL	
	INC D	
	LD A,(HL)	
	LD (DE),A	Copy byte 6.
	INC HL	
	INC D	
	LD A,(HL)	
	LD (DE),A	Copy byte 7.
	INC HL	
	INC D	
	LD A,(HL)	
	LD (DE),A	Copy byte 8.
	LD D,C	Restore D. « Last byte copied to RAM »

When the above routine is used in ROM, it drops through to here.

L246B:	EX AF,AF'	Need to switch back to RAM bank 7?
	CALL C,L1F64	If so then switch to use Workspace RAM configuration (physical RAM bank 7).
L246F:	POP HL	Fetch address of string data.
	INC HL	Move to next character.
	INC DE	Advance to next display file column.
	DJNZ L2400	Repeat for all requested characters.
	RET	

## Toggle ROMs 1 « RAM Routine »

Routine copied to RAM at \$FF28-\$FF35 by subroutine at \$248E (ROM 0).

This routine toggles to the other ROM than the one held in BANK\_M.

Entry: A'= Current paging configuration.

[Called by a routine that is itself never called by this ROM]

L2475:	PUSH BC	Save BC
	DI	Disable interrupts whilst paging.
	LD BC,\$7FFD	
	LD A,(BANK_M)	\$5B5C. Fetch current paging configuration.
	XOR \$10	Toggle ROMs.
	OUT (C),A	Perform paging.
	EI	Re-enable interrupts.
	EX AF,AF'	Save the new configuration in A'. « Last byte copied to RAM »

## Toggle ROMs 2 « RAM Routine »

Routine copied to RAM at \$FF56-\$FF60 by subroutine at \$248E (ROM 0).

This routine toggles to the other ROM than the one specified.

It is used to page back to the original configuration.

Entry: A'= Current paging configuration.

[Called by a routine that is itself never called by this ROM]

L2483:	EX AF,AF'	Retrieve current paging configuration.
	DI	Disable interrupts whilst paging.
	LD C,\$FD	Restore Paging I/O port number.
	XOR \$10	Toggle ROMs.
	OUT (C),A	Perform paging.
	EI	Re-enable interrupts.
	POP BC	Restore BC.
	RET	« Last byte copied to RAM »

## Construct 'Copy Character' Routine in RAM

This routine copies 3 sections of code into RAM to construct a single routine that can be used to copy the bit pattern for a character into the display file.

Copy \$2475-\$2482 (ROM 0) to \$FF28-\$FF35 (14 bytes).

Copy \$244B-\$246A (ROM 0) to \$FF36-\$FF55 (32 bytes).

Copy \$2483-\$248D (ROM 0) to \$FF56-\$FF60 (11 bytes).

[Never called by this ROM]

L248E:	LD HL,L2475	Point to the 'page in other ROM' routine.
	LD DE,\$FF28	Destination RAM address.
	LD BC,\$000E	
	LDIR	Copy the routine.
	PUSH HL	
	LD HL,L244B	Copy a character routine.
	LD C,\$20	
	LDIR	Copy the routine.
	POP HL	HL=\$2483 (ROM 0), which is the address of the 'page back to original ROM' routine.
	LD C,\$0B	
	LDIR	Copy the routine.
	RET	

## Set Attributes File from Screen Buffer

This routine parses the screen buffer string contents looking for colour control codes and changing the attributes file contents correspondingly.

[Called by the routine at \$23EA (ROM 0), which is itself never called by this ROM]

L24A7:	RES 0,(IY-\$39)	KSTATE+1. Signal do not invert attribute value. [Spanish 128 uses IY-\$3B]
	LD DE,\$5800	The start of the attributes file.

LD BC,\$02C0  
LD HL,(\$FF24)  
LD A,(\$5C8D)  
LD (\$5C8F),A

22 rows of 32 columns.  
The address of the string to print.  
ATTR\_P.  
ATTR\_T. Use the permanent colours.

## Set Attributes for a Screen Buffer Row

L24BA:           EX AF,AF'           Save the colour byte.

The main loop returns here on each iteration

L24BB:	PUSH BC LD A,(HL) CP \$FF JR NZ,L24C9 LD A,(\$5C8D) LD (DE),A INC HL INC DE JR L2526	Save the number of characters. Fetch a character from the buffer. Is it blank? Jump ahead if not. ATTR_P. Get the default colour byte. Store it in the attributes file. Point to next screen buffer position. Point to next attributes file position. Jump ahead to handle the next character.
--------	--	--

Not a blank character

L24C9:	EX AF,AF' LD (DE),A INC DE EX AF,AF' INC HL CP \$15 JR NC,L2526 CP \$10 JR C,L2526	Get the colour byte. Store it in the attributes file. Point to the next attributes file position. Save the colour byte. Point to the next screen buffer position. Is the string character OVER or above? Jump if it is to handle the next character. Is the string character below INK? Jump if it is to handle the next character.
--------	--	---

Screen buffer character is INK, PAPER, FLASH, BRIGHT or INVERSE.

DEC HL JR NZ,L24E1	Point back to the previous screen buffer position. Jump if not INK.
-----------------------	--

Screen character was INK so insert the new ink into the attribute byte.

L24E1:	INC HL LD A,(HL) LD C,A EX AF,AF' AND \$F8 JR L2524 CP \$11 JR NZ,L24F0	Point to the next screen buffer position. Fetch the ink colour from the next screen buffer position. and store it in C. Get the colour byte. Mask off the ink bits. Jump ahead to store the new attribute value and then to handle the next character. Is the string character PAPER? Jump ahead if not.
--------	--	---

Screen character was PAPER so insert the new paper into the attribute byte.

L24F0:	INC HL LD A,(HL) ADD A,A ADD A,A ADD A,A LD C,A EX AF,AF' AND \$C7 JR L2524 CP \$12 JR NZ,L24FD	Point to the next screen buffer position. Fetch the paper colour from the next screen buffer position.  Multiple by 8 so that ink colour become paper colour.  Get the colour byte. Mask off the paper bits. Jump ahead to store the new attribute value and then to handle the next character. Is the string character FLASH? Jump ahead if not.
--------	---	--

Screen character was FLASH

	INC HL	Point to the next screen buffer position.
	LD A,(HL)	Fetch the flash status from the next screen buffer position.
	RRCA	Shift the flash bit into bit 0.
	LD C,A	
	EX AF,AF'	Get the colour byte.
	AND \$7F	Mask off the flash bit.
	JR L2524	Jump ahead to store the new attribute value and then to handle the next character.
L24FD:	CP \$13	Is the string character BRIGHT?
	JR NZ,L250B	Jump ahead if not.

Screen character was BRIGHT

	INC HL	Point to the next screen buffer position.
	LD A,(HL)	Fetch the bright status from the next screen buffer position.
	RRCA	
	RRCA	Shift the bright bit into bit 0.
	LD C,A	
	EX AF,AF'	Get the colour byte.
	AND \$BF	Mask off the bright bit.
	JR L2524	Jump ahead to store the new attribute value and then to handle the next character.
L250B:	CP \$14	Is the string character INVERSE?
	INC HL	Point to the next screen buffer position.
	JR NZ,L2526	Jump ahead if not to handle the next character.

Screen character was INVERSE

	LD C,(HL)	Fetch the inverse status from the next screen buffer position.
	LD A,(\$5C01)	KSTATE+1. Fetch inverting status (Bit 0 is 0 for non-inverting, 1 for inverting).
	XOR C	Invert status.
	RRA	Shift status into the carry flag.
	JR NC,L2526	Jump if not inverting to handle the next character.
	LD A,\$01	Signal inverting is active.
	XOR (IY-\$39)	KSTATE+1. Toggle the status.
	LD (\$5C01),A	KSTATE+1. Store the new status.
	EX AF,AF'	Get the colour byte.
	CALL L2532	Swap ink and paper in the colour byte.
L2524:	OR C	Combine the old and new colour values.
	EX AF,AF'	Save the new colour byte.
L2526:	POP BC	Fetch the number of characters.
	DEC BC	
	LD A,B	
	OR C	
	JP NZ,L24BB	Repeat for all characters.
	EX AF,AF'	Get colour byte.
	LD (\$5C8F),A	ATTR_T. Make it the new temporary colour.
	RET	

## Swap Ink and Paper Attribute Bits

Entry: A=Attribute byte value.

Exit : A=Attribute byte value with paper and ink bits swapped.

[Called by the routine at \$24A7 (ROM 0), which is itself never called by this ROM]

L2532:	LD B,A	Save the original colour byte.
	AND \$C0	Keep only the flash and bright bits.
	LD C,A	
	LD A,B	
	ADD A,A	Shift ink bits into paper bits.
	ADD A,A	
	ADD A,A	
	AND \$38	Keep only the paper bits.
	OR C	Combine with the flash and bright bits.

```
LD C,A
LD A,B
RRA
RRA
RRA
AND $07
OR C
RET
```

Get the original colour byte.

Shift the paper bits into the ink bits.

Keep only the ink bits.

Add with the paper, flash and bright bits.

## Character Data

Graphic control code indicator

L2546:	DEFB \$00	0 0 0 0 0 0 0 0	
	DEFB \$3C	0 0 1 1 1 1 0 0	XXXX
	DEFB \$62	0 1 1 0 0 0 1 0	XX X
	DEFB \$60	0 1 1 0 0 0 0 0	XX
	DEFB \$6E	0 1 1 0 1 1 1 0	XX XXX
	DEFB \$62	0 1 1 0 0 0 1 0	XX X
	DEFB \$3E	0 0 1 1 1 1 1 0	XXXX
	DEFB \$00	0 0 0 0 0 0 0 0	

Error marker

L254E:	DEFB \$00	0 0 0 0 0 0 0 0	
	DEFB \$6C	0 1 1 0 1 1 0 0	XX XX
	DEFB \$10	0 0 0 1 0 0 0 0	X
	DEFB \$54	0 1 0 1 0 1 0 0	X X X
	DEFB \$BA	1 0 1 1 1 0 1 0	X XXX X
	DEFB \$38	0 0 1 1 1 0 0 0	XXX
	DEFB \$54	0 1 0 1 0 1 0 0	X X X
	DEFB \$82	1 0 0 0 0 0 1 0	X X

« End of Unused ROM Routines »

## KEY ACTION TABLES

### Editing Keys Action Table

Each editing key code maps to the appropriate handling routine.

This includes those keys which mirror the functionality of the add-on keypad; these are found by trapping the keyword produced by the keystrokes in 48K mode.

[Surprisingly there is no attempt to produce an intelligible layout instead the first 16 keywords have been used. Additionally the entries for DELETE and ENTER should probably come in the first six entries for efficiency reasons.]

L2556:	DEFB \$15	Number of table entries.
	DEFB \$0B	Key code: Cursor up.
	DEFW L2ABA	CURSOR-UP handler routine.
	DEFB \$0A	Key code: Cursor Down.
	DEFW L2ADB	CURSOR-DOWN handler routine.
	DEFB \$08	Key code: Cursor Left.
	DEFW L2AFD	CURSOR-LEFT handler routine.
	DEFB \$09	Key code: Cursor Right.
	DEFW L2B09	CURSOR-RIGHT handler routine.
	DEFB \$AD	Key code: Extend Mode + P.
	DEFW L2A75	TEN-ROWS-UP handler routine.
	DEFB \$AC	Key code: Symbol Shift + I.
	DEFW L2A4B	TEN-ROWS-DOWN handler routine.
	DEFB \$AF	Key code: Extend Mode + I.
	DEFW L29FA	WORD-LEFT handler routine.
	DEFB \$AE	Key code: Extend Mode + Shift + J.
	DEFW L2A07	WORD-RIGHT handler routine.
	DEFB \$A6	Key code: Extend Mode + N, or Graph + W.

DEFW L29A9	TOP-OF-PROGRAM handler routine.
DEFB \$A5	Key code: Extend Mode + T, or Graph + V.
DEFW L29D1	END-OF-PROGRAM handler routine.
DEFB \$A8	Key code: Extend Mode Symbol Shift + 2, or Graph Y.
DEFW L2AAD	START-OF-LINE handler routine.
DEFB \$A7	Key code: Extend Mode + M, or Graph + X.
DEFW L2AA0	END-OF-LINE handler routine.
DEFB \$AA	Key code: Extend Mode + Shift + K.
DEFW L2941	DELETE-RIGHT handler routine.
DEFB \$0C	Key code: Delete.
DEFW L2951	DELETE handler routine.
DEFB \$B3	Key code: Extend Mode + W.
DEFW L303D	DELETE-WORD-RIGHT handler routine.
DEFB \$B4	Key code: Extend Mode + E.
DEFW L2FE2	DELETE-WORD-LEFT handler routine.
DEFB \$B0	Key code: Extend Mode + J.
DEFW L3098	DELETE-TO-END-OF-LINE handler routine.
DEFB \$B1	Key code: Extend Mode + K.
DEFW L3064	DELETE-TO-START-OF-LINE handler routine.
DEFB \$0D	Key code: Enter.
DEFW L296A	ENTER handler routine.
DEFB \$A9	Key code: Extend Mode + Symbol Shift + 8, or Graph + Z.
DEFW L26BA	TOGGLE handler routine.
DEFB \$07	Key code: Edit.
DEFW L2723	MENU handler routine.

## Menu Keys Action Table

Each menu key code maps to the appropriate handling routine.

L2596:	DEFB \$04	Number of entries.
	DEFB \$0B	Key code: Cursor up.
	DEFW L274D	MENU-UP handler routine.
	DEFB \$0A	Key code: Cursor down.
	DEFW L2750	MENU-DOWN handler routine.
	DEFB \$07	Key code: Edit.
	DEFW L2736	MENU-SELECT handler routine.
	DEFB \$0D	Key code: Enter.
	DEFW L2736	MENU-SELECT handler routine.

## MENU ROUTINES — PART 3

### Initialise Mode Settings

Called before Main menu displayed.

L25A3:	CALL L28E4	Reset Cursor Position.
	LD HL,\$0000	No top line.
	LD (\$FC9A),HL	Line number at top of screen.
	LD A,\$82	Signal waiting for key press, and menu is displayed.
	LD (\$EC0D),A	Store the Editor flags.
	LD HL,\$0000	No current line number.
	LD (\$5C49),HL	E_PPC. Current line number.
	CALL L35E2	Reset indentation settings.
	CALL L3684	Reset to 'L' Mode
	RET	[Could have saved one byte by using JP \$3684 (ROM 0)]

### Show Main Menu

L25BE:	LD HL,TSTACK	\$5BFF.
	LD (OLDSP),HL	\$5B81.

	CALL L1F64	Use Workspace RAM configuration (physical RAM bank 7).
	LD A,\$02	Select main screen.
	RST 28H	
	DEFW CHAN_OPEN	
L25CC:	LD HL,L2763	\$1601.
	LD (\$F6EA),HL	Jump table for Main Menu.
	LD HL,L2770	Store current menu jump table address.
	LD (\$F6EC),HL	The Main Menu text.
	PUSH HL	Store current menu text table address.
	LD HL,\$EC0D	Store address of menu on stack.
	SET 1,(HL)	Editor flags.
	RES 4,(HL)	Indicate 'menu displayed'.
	DEC HL	Signal return to main menu.
	LD (HL),\$00	Current menu index.
	POP HL	Select top entry.
	CALL L36CE	Retrieve address of menu.
	JP L2672	Display menu and highlight first item.
		Jump ahead to enter the main key waiting and processing loop.

## EDITOR ROUTINES — PART 2

### Return to Editor / Calculator / Menu from Error

L25EA:	LD IX,\$FD6C	Point IX at editing settings information.
	LD HL,TSTACK	\$5BFF.
	LD (OLDSP),HL	\$5B81.
	CALL L1F64	Use Workspace RAM configuration (physical RAM bank 7).
	LD A,\$02	
	RST 28H	
	DEFW CHAN_OPEN	\$1601. Select main screen.
	CALL L368E	Reset 'L' mode.
	LD HL,\$5C3B	FLAGS.
L2602:	BIT 5,(HL)	Has a key been pressed?
	JR Z,L2602	Wait for a key press.
	LD HL,\$EC0D	Editor flags.
	RES 3,(HL)	Signal line has not been altered.
	BIT 6,(HL)	Is editing area the lower screen?
	JR NZ,L2623	If so then skip printing a banner and jump ahead to return to the Editor.
	LD A,(\$EC0E)	Fetch mode.
	CP \$04	Calculator mode?
	JR Z,L2620	Jump ahead if so.
	CP \$00	Edit Menu mode?
	JP NZ,L28ED	Jump if not to re-display Main menu.

Edit menu Print mode

CALL L386E	Clear screen and print "128 BASIC" in the banner line.
JR L2623	Jump ahead to return to the Editor.

Calculator mode

L2620:	CALL L3873	Clear screen and print "Calculator" in the banner line.
--------	------------	---

### Return to the Editor

Either as the result of a re-listing, an error or from completing the Edit Menu Print option.

**[BUG]** - Occurs only with ZX Interface 1 attached and a BASIC line such as 1000 OPEN #4, "X" (the line number must be greater than 999). This produces the error message "Invalid device expression, 1000:1" but the message is too long to fit on a single line. When using the lower screen for editing, spurious effects happen to the bottom lines. When using the full screen editor, a crash occurs. Credit: Toni Baker, ZX Computing Monthly] [The bug is caused by system variable DF\_SZ being increased to 3 as a result of the error message spilling onto an extra line. The error can be resolved by inserting a LD (IY+\$31),\$02 instruction at \$2623 (ROM 0). Credit: Paul Farrow]

## SPECTRUM +2 ROM 0 DISASSEMBLY

L2623:	CALL L30FC CALL L3248 LD A,(\$EC0E) CP \$04 JR Z,L2672	Reset Below-Screen Line Edit Buffer settings to their default values. Reset Above-Screen Line Edit Buffer settings to their default values. Fetch the mode. Calculator mode? Jump ahead if not to wait for a key press.
--------	--	---

### Calculator mode

LD HL,(\$5C49) LD A,H OR L JR NZ,L264C LD HL,(\$5C53) LD BC,(\$5C4B) AND A SBC HL,BC JR NZ,L2649	E_PPC. Fetch current line number.  Is there a current line number? Jump ahead if so. PROG. Address of start of BASIC program. VARS. Address of start of variables area.  HL=Length of program. Jump if a program exists.
--	--

### No program exists

L2649: L264C:	LD HL,\$0000 LD (\$EC08),HL LD HL,(\$EC08) CALL L1F3F RST 28H DEFW LINE_ADDR RST 28H DEFW LINE_NO CALL L1F64 LD (\$5C49),DE LD HL,\$EC0D BIT 5,(HL) JR NZ,L2672 LD HL,\$0000 LD (\$EC06),HL CALL L154E CALL L2A18 CALL L296A	Set no line number last edited. Fetch line number of last edited line. Use Normal RAM Configuration (physical RAM bank 0). Find address of line number held in HL, or the next line if it does not exist. \$196E. Return address in HL. Find line number for specified address, and return in DE. \$1695. Fetch the line number for the line found. Use Workspace RAM configuration (physical RAM bank 7). E_PPC. Save the current line number. Editor flags. Process the BASIC line? Jump ahead if calculator mode.  Signal no editable characters in the line prior to the cursor. Relist the BASIC program. Set attribute at editing position so as to show the cursor. Call the ENTER handler routine.
------------------	---	--

## Main Waiting Loop

Enter a loop to wait for a key press. Handles key presses for menus, the Calculator and the Editor.

L2672:	LD SP,TSTACK CALL L368E CALL L36A5  PUSH AF LD A,(\$5C39) CALL L270B POP AF CALL L2688 JR L2672	\$5BFF. Use temporary stack. Reset 'L' mode. Wait for a key. [Note that it is possible to change CAPS LOCK mode whilst on a menu] Save key code. PIP. Tone of keyboard click. Produce a key click noise. Retrieve key code. Process the key press. Wait for another key.
--------	--	--

## Process Key Press

Handle key presses for the menus and the Editor.

Entry: A=Key code.  
Zero flag set if a menu is being displayed.

L2688:	LD HL,\$EC0D BIT 1,(HL) PUSH AF	Editor flags. Is a menu is displayed? Save key code and flags.
--------	---------------------------------------	--

## SPECTRUM +2 ROM 0 DISASSEMBLY

	LD HL,L2596	Use menu keys lookup table.
	JR NZ,L2696	Jump if menu is being displayed.
	LD HL,L2556	Use editing keys lookup table.
L2696:	CALL L3F75	Find and call the action handler for this key press.
	JR NZ,L26A0	Jump ahead if no match found.
	CALL NC,L2706	If required then produce error beep.
	POP AF	Restore key code.
	RET	

No action defined for key code

L26A0:	POP AF	Restore key code and flags.
	JR Z,L26A8	Jump if menu is not being displayed.

A menu is being displayed, so just ignore key press

	XOR A	Select 'L' mode.
	LD (\$5C41),A	MODE.
	RET	

A menu is not being displayed

L26A8:	LD HL,\$EC0D	Editor flags.
	BIT 0,(HL)	Is the Screen Line Edit Buffer is full?
	JR Z,L26B3	Jump if not to process the key code.

The buffer is full so ignore the key press

	CALL L2706	Produce error beep.
	RET	[Could have save a byte by using JP \$2706 (ROM 0)]
L26B3:	CP \$A3	Was it a supported function key code?
	JR NC,L2672	Ignore by jumping back to wait for another key. <b>[BUG]</b> - This should be RET NC since it was called from the loop at \$2672 (ROM 0). Repeatedly pressing an unsupported key will result in a stack memory leak and eventual overflow. Credit: John Steven (+3), Paul Farrow (128)]
	JP L2917	Jump forward to handle the character key press.

## TOGGLE Key Handler Routine

Toggle between editing in the lower and upper screen areas.  
Also used by the editing menu SCREEN option.

L26BA:	LD A,(\$EC0E)	Fetch mode.
	CP \$04	Calculator mode?
	RET Z	Return if so (TOGGLE has no effect in Calculator mode).
	CALL L164F	Clear Editing Display.
	LD HL,\$EC0D	Editor flags.
	RES 3,(HL)	Reset 'line altered' flag.
	LD A,(HL)	
	XOR \$40	
	LD (HL),A	Toggle screen editing area flag.
	AND \$40	
	JR Z,L26D5	Jump forward if the editing area is now the upper area.
	CALL L26DA	Set the lower area as the current editing area.
	JR L26D8	Jump forward.
L26D5:	CALL L26ED	Set the upper area as the current editing area.
L26D8:	SCF	Signal do not produce an error beep.
	RET	

## Select Lower Screen

Set the lower screen as the editing area.

L26DA:	CALL L38A2 LD HL,\$EC0D SET 6,(HL) CALL L2E53 CALL L3AA9 CALL L2905 JR L26F8	Clear lower editing area display. Editor flags. Signal using lower screen. Reset to lower screen. Set default lower screen editing cursor settings. Set default lower screen editing settings. Jump ahead to continue.
--------	--	--

## Select Upper Screen

Set the upper screen as the editing area.

L26ED:	LD HL,\$EC0D RES 6,(HL) CALL L28E4 CALL L386E	Editor flags. Signal using main screen. Reset Cursor Position. Clear screen and print the "128 BASIC" banner line.
L26F8:	LD HL,(\$FC9A) LD A,H OR L CALL NZ,L3370 CALL L154E JP L2A18	Line number at top of screen.  Is there a line? If there is then get the address of BASIC line for this line number. Relist the BASIC program. Set attribute at editing position so as to show the cursor, and return.

## Produce Error Beep

This is the entry point to produce the error beep, e.g. when trying to cursor up or down past the BASIC program.

It produces a different tone and duration from the error beep of 48K mode. The change in pitch is due to the SRL A instruction at \$2709 (ROM 0), and the change in duration is due to the instruction at \$2710 (ROM 0) which loads HL with \$0C80 as opposed to \$1A90 which is used when in 48K mode. The key click and key repeat sounds are produced by entering at \$270B (ROM 0) but with A holding the value of system variable PIP. This produces the same tone as 48K mode but is of a much longer duration due to HL being loaded with \$0C80 as opposed to the value of \$00C8 used in 48K mode. The Spanish 128 uses the same key click tone and duration in 128K mode as it does in 48K mode, leading to speculation that the Spectrum 128 (and subsequent models) should have done the same and hence suffer from a bug. However, there is no reason why this should be the case, and it can easily be imagined that the error beep note duration of 48K mode would quickly become very irritating when in 128K mode where it is likely to occur far more often. Hence the reason for its shorter duration. The reason for the longer key click is less clear, unless it was to save memory by using a single routine. However, it would only have required an additional 3 bytes to set HL independently for key clicks, which is not a great deal considering there is 1/2K of unused routines at \$2355 (ROM 0). Since the INPUT command is handled by ROM 1, it produces key clicks at the 48K mode duration even when executed from 128 BASIC mode.

L2706:	LD A,(\$5C38) SRL A	RASP. Divide by 2.
--------	------------------------	-----------------------

This entry point is called to produce the key click tone. In 48K mode, the key click sound uses an HL value of \$00C8 and so is 16 times shorter than in 128K mode.

L270B:	PUSH IX LD D,\$00 LD E,A LD HL,\$0C80	Pitch.  Duration.
L2713:	RST 28H DEFW BEEPER POP IX RET	\$03B5. Produce a tone.

## Produce Success Beep

L2719:	PUSH IX LD DE,\$0030 LD HL,\$0300 JR L2713	Frequency*Time. Duration. Jump to produce the tone.
--------	---	---

## MENU ROUTINES — PART 4

### Menu Key Press Handler Routines

#### Menu Key Press Handler — MENU

This is executed when the EDIT key is pressed, either from within a menu or from the BASIC editor.

L2723:	CALL L2A12	Remove cursor, restoring old attribute.
	LD HL,\$EC0D	HL points to Editor flags.
	SET 1,(HL)	Signal 'menu is being displayed'.
	DEC HL	HL=\$EC0C.
	LD (HL),\$00	Set 'current menu item' as the top item.
L272E:	LD HL,(\$F6EC)	Address of text for current menu.
	CALL L36CE	Display menu and highlight first item.
	SCF	Signal do not produce an error beep.
	RET	

#### Menu Key Press Handler — SELECT

L2736:	LD HL,\$EC0D	HL points to Editor flags.
	RES 1,(HL)	Clear 'displaying menu' flag.
	DEC HL	HL=\$EC0C.
	LD A,(HL)	A=Current menu option index.
	LD HL,(\$F6EA)	HL points to jump table for current menu.
	PUSH HL	
	PUSH AF	
	CALL L3764	Restore menu screen area.
	POP AF	
	POP HL	
	CALL L3F75	Call the item in the jump table corresponding to the currently selected menu item.
	JP L2A18	Set attribute at editing position so as to show the cursor, and return.

#### Menu Key Press Handler — CURSOR UP

L274D:	SCF	Signal move up.
	JR L2751	Jump ahead to continue.

#### Menu Key Press Handler — CURSOR DOWN

L2750:	AND A	Signal moving down.
L2751:	LD HL,\$EC0C	
	LD A,(HL)	Fetch current menu index.
	PUSH HL	Save it.
	LD HL,(\$F6EC)	Address of text for current menu.
	CALL C,L37CD	Call if moving up.
	CALL NC,L37DC	Call if moving down.
	POP HL	HL=Address of current menu index store.
	LD (HL),A	Store the new menu index.

Comes here to complete handling of Menu cursor up and down. Also as the handler routines for Edit Menu return to 128 BASIC option and Calculator menu return to Calculator option, which simply make a return.

L2761:	SCF
	RET

## Menu Tables

### Main Menu

Jump table for the main 128K menu, referenced at \$25CC (ROM 0).

L2763:	DEFB \$04	Number of entries.
	DEFB \$00	
	DEFW L2857	Tape Loader option handler.
	DEFB \$01	
	DEFW L2892	128 BASIC option handler.
	DEFB \$02	
	DEFW L28AB	Calculator option handler.
	DEFB \$03	
	DEFW L1B66	48 BASIC option handler.

Text for the main 128K menu

L2770:	DEFB \$05	Number of entries.
	DEFM "128 "	Menu title.
	DEFB \$FF	
L277A:	DEFM "Tape Loade"	
	DEFB 'r'+\$80	
L2785:	DEFM "128 BASI"	
	DEFB 'C'+\$80	
L278E:	DEFM "Calculato"	
	DEFB 'r'+\$80	
	DEFM "48 BASI"	
	DEFB 'C'+\$80	
	DEFB ''+\$80	\$A0. End marker.

### Edit Menu

Jump table for the Edit menu

L27A1:	DEFB \$05	Number of entries.
	DEFB \$00	
	DEFW L2761	(Return to) 128 BASIC option handler.
	DEFB \$01	
	DEFW L2877	Renumber option handler.
	DEFB \$02	
	DEFW L283D	Screen option handler.
	DEFB \$03	
	DEFW L2888	Print option handler.
	DEFB \$04	
	DEFW L2842	Exit option handler.

Text for the Edit menu

L27B1:	DEFB \$06	Number of entries.
	DEFM "Options "	
	DEFB \$FF	
	DEFM "128 BASI"	
	DEFB 'C'+\$80	
	DEFM "Renumbe"	
	DEFB 'r'+\$80	
	DEFM "Scree"	
	DEFB 'n'+\$80	
	DEFM "Prin"	
	DEFB 't'+\$80	
	DEFM "Exi"	
	DEFB 't'+\$80	

DEFB '\$'+\$80

\$A0. End marker.

## Calculator Menu

Jump table for the Calculator menu

L27DC:	DEFB \$02	Number of entries.
	DEFB \$00	
	DEFW L2761	(Return to) Calculator option handler.
	DEFB \$01	
	DEFW L2842	Exit option handler.

Text for the Calculator menu

L27E3:	DEFB 03	Number of entries.
	DEFM "Options "	
	DEFB \$FF	
	DEFM "Calculato"	
	DEFB '\$'+\$80	
	DEFM "Exi"	
	DEFB '\$'+\$80	
	DEFB '\$'+\$80	\$A0. End marker.

## Tape Loader Text

L27FC:	DEFB \$16,\$00,\$00	AT 0,0
	DEFB \$10, \$00	INK 0
	DEFB \$11, \$07	PAPER 7
	DEFB \$13, \$00	BRIGHT 1
	DEFM "Insert tape and press PLAY"	
	DEFB \$0D	
	DEFM "To cancel - press BREAK twic"	
	DEFB '\$'+\$80	

## Menu Handler Routines

### Edit Menu — Screen Option

L283D:	CALL L26BA	Toggle between editing in the lower and upper screen areas.
	JR L289A	Jump ahead.

### Edit Menu / Calculator Menu — Exit Option

L2842:	LD HL,\$EC0D	Editor flags.
	RES 6,(HL)	Indicate main screen editing.
	CALL L28E4	Reset Cursor Position.
	LD B,\$00	Top row to clear.
	LD D,\$17	Bottom row to clear.
	CALL L3B7F	Clear specified display rows.
	CALL L1F3F	Use Normal RAM Configuration (physical RAM bank 0).
	JP L25BE	Jump back to show the menu.

## Main Menu — Tape Loader Option

L2857:	CALL L3878 LD HL,\$5C3C SET 0,(HL) LD DE,L27FC CALL L059C RES 0,(HL) SET 6,(HL)	Clear screen and print "Tape Loader" in the banner line. TVFLAG. Signal using lower screen area. Point to message "Insert tape and press PLAY. To cancel - press BREAK twice". Print the text. Signal using main screen area. [This bit is unused in the 48K Spectrum and only ever set in 128K mode via the Tape Loader option. It is never subsequently tested or reset. It may have been the intention to use this to indicate that the screen requires clearing after loading to remove the "Tape Loader" banner and the lower screen message "Insert tape and press PLAY. To cancel - press BREAK twice"] Tape Loader mode. [Redundant since call to \$1B10 (ROM 0) will set it to \$FF]
	LD A,\$07 LD (\$EC0E),A LD BC,\$0000 CALL L3751 JP L1B10	Perform 'Print AT 0,0;'. Run the tape loader.

## Edit Menu — Renumber Option

L2877:	CALL L38A9 CALL NC,L2706 LD HL,\$0000 LD (\$5C49),HL LD (\$EC08),HL JR L288B	Run the renumber routine. If not successful then produce error beep if required. There is no current line number. E_PPC. Current line number. Temporary E_PPC used by BASIC Editor. Jump ahead to display the "128 BASIC" banner if required, set the menu mode and return.
--------	---	--

## Edit Menu — Print Option

L2888:	CALL L1B33	Perform an LLIST.
Edit Menu - Renumber option joins here		
L288B:	LD HL,\$EC0D BIT 6,(HL) JR NZ,L289A	Editor flags. Using lower editing screen? Jump ahead if so.
L2892:	LD HL,\$5C3C RES 0,(HL) CALL L386E	TVFLAG. Allow leading space. Clear screen and print the "128 BASIC" banner line.

Edit Menu - Screen option joins here

L289A:	LD HL,\$EC0D RES 5,(HL) RES 4,(HL) LD A,\$00 LD HL,L27A1 LD DE,L27B1 JR L28D7	Editor flags. Signal not to process the BASIC line. Signal return to main menu. Select Edit menu mode. [Could have saved 1 byte by using XOR A] Edit Menu jump table. Edit Menu text table. Store the new mode and menu details.
--------	---	--

## Main Menu — Calculator Option

L28AB:	LD HL,\$EC0D SET 5,(HL) SET 4,(HL)	Editor flags. Signal to process the BASIC line. Signal return to calculator.
--------	--	--

```

RES 6,(HL)
CALL L28E4
CALL L3873
LD A,$04
LD ($EC0E),A
LD HL,$0000
LD ($5C49),HL
CALL L154E
LD BC,$0000
LD A,B
CALL L2A1E
LD A,$04
LD HL,L27DC
LD DE,L27E3

```

Signal editing are is the main screen.  
 Reset cursor position.  
 Clear screen and print "Calculator" in the banner line.  
 Set calculator mode.  
 Store mode.  
 No current line number.  
 E\_PPC. Store current line number.  
 Relist the BASIC program.  
 B=Row. C=Column. Top left of screen.  
 Preferred column.  
 Store editing position and print cursor.  
 Select calculator mode.  
 Calculator Menu jump table  
 Calculator Menu text table

Edit Menu - Print option joins here

```

L28D7:      LD ($EC0E),A
            LD ($F6EA),HL
            LD ($F6EC),DE
            JP L2623

```

Store mode.  
 Store address of current menu jump table.  
 Store address of current menu text.  
 Return to the Editor.

## EDITOR ROUTINES — PART 3

### Reset Cursor Position

```

L28E4:      CALL L2E45
            CALL L3AA0
            JP L290E

```

Reset to main screen.  
 Set default main screen editing cursor details.  
 Set default main screen editing settings.

### Return to Main Menu

```

L28ED:      LD B,$00
            LD D,$17
            CALL L3B7F
            JP L25CC

```

Top row of editing area.  
 Bottom row of editing area.  
 Clear specified display rows.  
 Jump to show Main menu.

### Main Screen Error Cursor Settings

Main screen editing cursor settings.  
 Gets copied to \$F6EE.

```

L28F7:      DEFB $06
            DEFB $00
            DEFB $00
            DEFB $00
            DEFB $04
            DEFB $10
            DEFB $14

```

Number of bytes in table.  
 \$F6EE = Cursor position - row 0.  
 \$F6EF = Cursor position - column 0.  
 \$F6F0 = Cursor position - column 0 preferred.  
 \$F6F1 = Top row before scrolling up.  
 \$F6F2 = Bottom row before scrolling down.  
 \$F6F3 = Number of rows in the editing area.

### Lower Screen Good Cursor Settings

Lower screen editing cursor settings.  
 Gets copied to \$F6EE.

```

L28FE:      DEFB $06
            DEFB $00
            DEFB $00

```

Number of bytes in table.  
 \$F6EE = Cursor position - row 0.  
 \$F6EF = Cursor position - column 0.

DEFB \$00  
 DEFB \$00  
 DEFB \$01  
 DEFB \$01

\$F6F0 = Cursor position - column 0 preferred.  
 \$F6F1 = Top row before scrolling up.  
 \$F6F2 = Bottom row before scrolling down.  
 \$F6F3 = Number of rows in the editing area.

## Initialise Lower Screen Editing Settings

Used when selecting lower screen. Copies 6 bytes from \$28FF (ROM 0) to \$F6EE.

L2905:	LD HL,L28FE LD DE,\$F6EE JP L3F61	Default lower screen editing information. Editing information stores. Copy bytes.
--------	---	---

## Initialise Main Screen Editing Settings

Used when selecting main screen. Copies 6 bytes from \$28F8 (ROM 0) to \$F6EE.

L290E:	LD HL,L28F7 LD DE,\$F6EE JP L3F61	Default main screen editing information. Editing information stores. Copy bytes.
--------	---	--

## Handle Key Press Character Code

This routine handles a character typed at the keyboard, inserting it into the Screen Line Edit Buffer as appropriate.

Entry: A=Key press character code.

L2917:	LD HL,\$EC0D OR A OR A BIT 0,(HL) JP NZ,L2A18 RES 7,(HL) SET 3,(HL) PUSH HL PUSH AF CALL L2A12 POP AF PUSH AF CALL L2EA7 POP AF LD A,B CALL L2B9E  POP HL SET 7,(HL) JP NC,L2A18  LD A,B JP C,L2A1E  JP L2A18	Editor flags. Clear carry flag. [Redundant instruction since carry flag return state never checked] [Redundant instruction] Is the Screen Line Edit Buffer is full? Jump if it is to set attribute at editing position so as to show the cursor, and return. Signal got a key press. Signal current line has been altered. Save address of the flags. Save key code. Remove cursor, restoring old attribute.  Get and save key code. Insert the character into the Screen Line Edit Buffer. Get key code. B=Current cursor column position. Find next Screen Line Edit Buffer editable position to right, moving to next row if necessary. Get address of the flags. Signal wait for a key. Jump if new position not available to set cursor attribute at existing editing position, and return. A=New cursor column position. Jump if new position is editable to store editing position and print cursor. [This only needs to be JP \$2A1E (ROM 0), thereby saving 3 bytes, since a branch to \$2A18 (ROM 0) would have been taken above if the carry flag was reset] Set attribute at editing position so as to show the cursor, and return.
--------	---	--

## DELETE-RIGHT Key Handler Routine

Delete a character to the right. An error beep is not produced if there is nothing to delete.

Symbol:

DEL  
→

Exit: Carry flag set to indicate not to produce an error beep.

L2941:	LD HL,\$EC0D SET 3,(HL) CALL L2A12 CALL L2F38 SCF LD A,B JP L2A1E	HL points to Editor flags. Indicate 'line altered'. Remove cursor, restoring old attribute. Exit with C=row, B=column. Delete character to the right, shifting subsequent rows as required. Signal do not produce an error beep. A=The new cursor editing position. Store editing position and print cursor, and then return.
--------	---	---

## DELETE Key Handler Routine

Delete a character to the left. An error beep is not produced if there is nothing to delete.

Symbol:

DEL  
←

Exit: Carry flag set to indicate not to produce an error beep.

L2951:	LD HL,\$EC0D RES 0,(HL) SET 3,(HL) CALL L2A12 CALL L2B81 CCF JP C,L2A18  CALL L2F38 SCF LD A,B JP L2A1E	HL points to Editor flags. Signal that the Screen Line Edit Buffer is not full. Indicate 'line altered'. Remove cursor, restoring old attribute. Exit with C=row, B=column. Select previous column position (Returns carry flag set if editable). Signal do not produce an error beep if not editable. Jump if not editable to set attribute at editing position so as to show the cursor, and return. Delete character to the right, shifting subsequent rows as required. Signal do not produce an error beep. A=The new cursor editing position. Store editing position and print cursor, and then return.
--------	--	---

## ENTER Key Handler Routine

This routine handles ENTER being pressed. If not on a BASIC line then it does nothing. If on an unaltered BASIC line then insert a blank row after it and move the cursor to it. If on an altered BASIC line then attempt to enter it into the BASIC program, otherwise return to produce an error beep.

Exit: Carry flag reset to indicate to produce an error beep.

L296A:	CALL L2A12 PUSH AF CALL L30DA PUSH BC LD B,\$00 CALL L2E67  POP BC JR C,L2984	Remove cursor, restoring old attribute. Save preferred column number. DE=Start address in Screen Line Edit Buffer of the row specified in C. Stack current editing position. Column 0. Is this a blank row? i.e. Find editable position on this row to the right, returning column number in B. Retrieve current editing position. Jump ahead if editable position found, i.e. not a blank row.
--------	---	--

No editable characters on the row, i.e. a blank row

LD HL,\$0020 ADD HL,DE LD A,(HL) CPL AND \$09 JR Z,L29A0	Point to the flag byte for the row. Fetch the flag byte. Invert it. Keep the 'first row' and 'last row' flags. Jump if both flags were set indicating not on a BASIC line.
---	--

On a BASIC line

L2984:	LD A,(\$EC0D) BIT 3,A JR Z,L2990	Editor flags. Has the current line been altered? Jump ahead if not.
--------	--	---

The current BASIC line has been altered

CALL L2CB4	Enter line into program.
------------	--------------------------

L2990:	JR NC,L29A5 CALL L2C72	Jump if syntax error to produce an error beep. Find end of the current BASIC line in the Screen Line Edit Buffer, scrolling up rows as required. Returns column number into B.
	CALL L2B9E CALL L2EF4	Find address of end position in current BASIC line. Returns address into HL. Insert a blank line in the Screen Line Edit Buffer, shifting subsequent rows down.

Display the cursor on the first column of the next row

LD B,\$00	First column.
POP AF	A=Preferred column number.
SCF	Signal do not produce an error beep.
JP L2A1E	Store editing position and print cursor, and then return.

Cursor is on a blank row, which is not part of a BASIC line

L29A0:	POP AF SCF JP L2A18	Discard stacked item. Signal do not produce an error beep. Set attribute at current editing position so as to show the cursor, and return.
--------	---------------------------	--

A syntax error occurred so return signalling to produce an error beep

L29A5:	POP AF JP L2A18	Discard stacked item. Set attribute at current editing position so as to show the cursor, and return.
--------	--------------------	--

## TOP-OF-PROGRAM Key Handler Routine

Move to the first row of the first line of the BASIC program. An error beep is not produced if there is no program.

Symbol:



Exit: Carry flag set to indicate not to produce an error beep.

L29A9:	LD A,(\$EC0E) CP \$04 RET Z	Fetch mode. Calculator mode? Exit if so.
--------	-----------------------------------	--

Editor mode

CALL L2A12	Remove cursor, restoring old attribute.
LD HL,\$0000	The first possible line number.
CALL L1F3F	Use Normal RAM Configuration (physical RAM bank 0).
RST 28H	Find address of line number 0, or the next line if it does not exist.
DEFW LINE_ADDR	\$196E. Return address in HL.
RST 28H	Find line number for specified address, and return in DE.
DEFW LINE_NO	\$1695. DE=Address of first line in the BASIC program.
CALL L1F64	Use Workspace RAM configuration (physical RAM bank 7).
LD (\$5C49),DE	E_PPC. Store the current line number.
LD A,\$0F	Paper 1, Ink 7 - Blue.
CALL L3AB7	Set the cursor colour.
CALL L154E	Relist the BASIC program.
SCF	Signal do not produce an error beep.
JP L2A18	Set attribute at editing position so as to show the cursor, and return.

## END-OF-PROGRAM Key Handler Routine

Move to the last row of the bottom line of the BASIC program. An error beep is not produced if there is no program.

Symbol:



Exit: Carry flag set to indicate not to produce an error beep.

L29D1:	LD A,(\$EC0E)	Fetch mode.
--------	---------------	-------------

CP \$04  
RET Z

Calculator mode?  
Exit if so.

Editor mode

CALL L2A12  
LD HL,\$270F  
CALL L1F3F  
RST 28H  
DEFW LINE\_ADDR  
EX DE,HL  
RST 28H  
DEFW LINE\_NO  
CALL L1F64  
LD (\$5C49),DE  
LD A,\$0F  
CALL L3AB7  
CALL L154E  
SCF  
JP L2A18

Remove cursor, restoring old attribute.  
The last possible line number, 9999.  
Use Normal RAM Configuration (physical RAM bank 0).  
Find address of line number 9999, or the previous line if it does not exist.  
\$196E. Return address in HL.  
DE=Address of last line number.  
Find line number for specified address, and return in DE.  
\$1695. DE=Address of last line in the BASIC program.  
Use Workspace RAM configuration (physical RAM bank 7).  
E\_PPC. Store the current line number.  
Paper 1, Ink 7 - Blue.  
Set the cursor colour.  
Relist the BASIC program.  
Signal do not produce an error beep.  
Set attribute at editing position so as to show the cursor, and return.

## WORD-LEFT Key Handler Routine

This routine moves to the start of the current word that the cursor is on, or if it is on the first character of a word then it moves to the start of the previous word. If there is no word to move to then signal to produce an error beep.

Symbol:



Exit: Carry flag reset to indicate to produce an error beep.

L29FA: CALL L2A12  
CALL L2C10  
JP NC,L2A18  
  
LD A,B  
  
JP L2A1E

Remove cursor, restoring old attribute.  
Find start of the current word to the left.  
Jump if no word to the left to restore cursor attribute at current editing position, and return. [Could have saved 4 bytes by joining the routine below, i.e. JR \$29E7]  
A=New cursor column number. Carry flag is set indicating not to produce an error beep.  
Store editing position and print cursor, and then return.

## WORD-RIGHT Key Handler Routine

This routine moves to the start of the next word. If there is no word to move to then signal to produce an error beep.

Symbol:



Exit: Carry flag reset to indicate to produce an error beep.

L2A07: CALL L2A12  
CALL L2C2F  
JR NC,L2A18  
  
LD A,B  
  
JR L2A1E

Remove cursor, restoring old attribute.  
Find start of the current word to the right.  
Jump if no word to the right to restore cursor attribute at current editing position, and return.  
A=The new cursor editing column number. Carry is set indicating not to produce an error beep.  
Store editing position and print cursor, and then return.

## Remove Cursor

Remove editing cursor colour from current position.

Exit: C=row number.

B=Column number.

L2A12: CALL L2A2D  
JP L3675

Get current cursor position (C=row, B=column, A=preferred column).  
Restore previous colour to character square

## Show Cursor

Set editing cursor colour at current position.

Exit: C=row number.

B=Column number.

L2A18:       CALL L2A2D  
              JP L3666

Get current cursor position (C=row, B=column, A=preferred column).

Set editing position character square to cursor colour to show it. [Could have saved 1 byte by using a JR instruction to join the end of the routine below]

## Display Cursor

Set editing cursor position and colour and then show it.

Entry:       C=Row number.

              B=Column number.

              A=Preferred column number.

L2A1E:       CALL L2A37  
              PUSH AF  
              PUSH BC  
              LD A,\$0F  
              CALL L3AB7  
              POP BC  
              POP AF  
              JP L3666

Store new editing position.

Paper 1, Ink 7 - Blue.

Store new cursor colour.

Set editing position character square to cursor colour to show it.

## Fetch Cursor Position

Returns the three bytes of the cursor position.

Exit : C=Row number.

B=Column number

A=Preferred column number.

L2A2D:       LD HL,\$F6EE  
              LD C,(HL)  
              INC HL  
              LD B,(HL)  
              INC HL  
              LD A,(HL)  
              INC HL  
              RET

Editing info.

Row number.

Column number.

Preferred column number.

## Store Cursor Position

Store new editing cursor position.

Entry:       C=Row number.

              B=Column number.

              A=Preferred column number.

L2A37:       LD HL,\$F6EE  
              LD (HL),C  
              INC HL  
              LD (HL),B  
              INC HL  
              LD (HL),A  
              RET

Editing information.

Row number.

Column number.

Preferred column number.

## Get Current Character from Screen Line Edit Buffer

L2A40:	PUSH HL CALL L30DA LD H,\$00 LD L,B ADD HL,DE LD A,(HL) POP HL RET	DE=Start address in Screen Line Edit Buffer of the row specified in C. [Could have saved 2 bytes by calling the unused routine at \$2EA1 (ROM 0)]  Point to the column position within the row. Get character at this position.
--------	---	---

## TEN-ROWS-DOWN Key Handler Routine

Move down 10 rows within the BASIC program, attempting to place the cursor as close to the preferred column number as possible. An error beep is produced if there is not 10 rows below.

Symbol:



Exit: Carry flag reset to indicate to produce an error beep.

L2A4B:	CALL L2A12 LD E,A LD D,\$0A	Remove cursor, restoring old attribute. E=Preferred column. The ten lines to move down.
L2A51:	PUSH DE CALL L2B56  POP DE  JR NC,L2A18  LD A,E CALL L2A37 LD B,E CALL L2B1F  JR NC,L2A68 DEC D JR NZ,L2A51 LD A,E JR C,L2A1E	Move down to the next row, shifting rows up as appropriate. If moving onto a new BASIC line then insert the previous BASIC line into the BASIC program if it has been altered. Returns new row number in C. Jump if there was no row below to set attribute at editing position so as to show the cursor, and return. A=Preferred column. Store cursor editing position. B=Preferred column. Find closest Screen Line Edit Buffer editable position to the right else to the left, returning column number in B. Jump if no editable position found on the row, i.e. a blank row. Decrement row counter. Repeat to move down to the next row. A=Preferred column. Jump if editable row exists to store editing position and print cursor, and then return. [Redundant check of the carry flag, should just be JR \$2A1E (ROM 0)]

A blank row was found below, must be at the end of the BASIC program

L2A68:	PUSH DE CALL L2B31 POP DE LD B,E CALL L2B1F  LD A,E OR A JR L2A1E	Move back up to the previous row.  B=Preferred column. Find closest Screen Line Edit Buffer editable position to the right else to the left, returning column number in B. A=Preferred column. Carry will be reset indicating to produce an error beep. Store editing position and print cursor, and then return.
--------	---	---

## TEN-ROWS-UP Key Handler Routine

Move up 10 rows within the BASIC program, attempting to place the cursor as close to the preferred column number as possible. An error beep is produced if there is not 10 rows above.

Symbol:



Exit: Carry flag reset to indicate to produce an error beep.

L2A75:	CALL L2A12	Remove cursor, restoring old attribute.
	LD E,A	E=Preferred column.
	LD D,\$0A	The ten lines to move up.
L2A7B:	PUSH DE	
	CALL L2B31	Move up to the previous row, shifting rows down as appropriate. If moving onto a new BASIC line then
	POP DE	insert the previous BASIC line into the BASIC program if it has been altered.
	JR NC,L2A18	Jump if there was no row above to set cursor attribute colour at existing editing position, and return.
	LD A,E	A=Preferred column.
	CALL L2A37	Store cursor editing position.
	LD B,E	B=Preferred column.
	CALL L2B28	Find closest Screen Line Edit Buffer editable position to the left else right, return column number in B.
	JR NC,L2A93	Jump if no editable positions were found in the row, i.e. it is a blank row.
	DEC D	Decrement row counter.
	JR NZ,L2A7B	Repeat to move up to the previous row.
	LD A,E	A=Preferred column.
	JP C,L2A1E	Jump if editable row exists to store editing position and print cursor, and then return.
		[Redundant check of the carry flag, should just be JP \$2A1E (ROM 0)]

A blank row was found above, must be at the start of the BASIC program [???? Can this ever be the case?]

L2A93:	PUSH AF	Save the preferred column number and the flags.
	CALL L2B56	Move back down to the next row. Returns new row number in C.
	LD B,\$00	Column 0.
	CALL L2BFA	Find editable position in the Screen Line Edit Buffer row to the right, return column position in B.
	POP AF	A=Preferred column. Carry will be reset indicating to produce an error beep.
	JP L2A1E	Store editing position and print cursor, and then return.

## END-OF-LINE Key Handler Routine

Move to the end of the current BASIC line. An error beep is produced if there is no characters in the current BASIC line.

Symbol:



Exit: Carry flag reset to indicate to produce an error beep and set not to produce an error beep.

L2AA0:	CALL L2A12	Remove cursor, restoring old attribute.
	CALL L2C72	Find the end of the current BASIC line in the Screen Line Edit Buffer.
	JP NC,L2A18	Jump if a blank row to set attribute at existing editing position so as to show the cursor, and return.
	LD A,B	A=The new cursor editing column number. Carry is set indicating not to produce an error beep.
	JP L2A1E	Store editing position and print cursor, and then return.

## START-OF-LINE Key Handler Routine

Move to the start of the current BASIC line. An error beep is produced if there is no characters in the current BASIC line.

Symbol:



Exit: Carry flag reset to indicate to produce an error beep.

L2AAD:	CALL L2A12	Remove cursor, restoring old attribute.
	CALL L2C57	Find the start of the current BASIC line in the Screen Line Edit Buffer.
	JP NC,L2A18	Jump if a blank row to set attribute at existing editing position so as to show the cursor, and return.
	LD A,B	A=The new cursor editing position. Carry is set indicating not to produce an error beep.

JP L2A1E

Store editing position and print cursor, and then return.

## CURSOR-UP Key Handler Routine

Move up 1 row, attempting to place the cursor as close to the preferred column number as possible.

An error beep is produced if there is no row above.

Exit: Carry flag reset to indicate to produce an error beep.

L2ABA:	CALL L2A12	Remove cursor, restoring old attribute.
	LD E,A	E=Preferred column.
	PUSH DE	
	CALL L2B31	Move up to the previous row, shifting rows down as appropriate. If moving onto a new BASIC line then
	POP DE	insert the previous BASIC line into the BASIC program if it has been altered.
	JP NC,L2A18	Jump if there was no row above to set cursor attribute colour at existing editing position, and return.
	LD B,E	B=Preferred column.
	CALL L2B28	Find closest Screen Line Edit Buffer editable position to the left else right, return column number in B.
	LD A,E	A=Preferred column.
	JP C,L2A1E	Jump if an editable position was found to store editing position and print cursor, and then return.

A blank row was found above, must be at the start of the BASIC program [???? Can this ever be the case?]

PUSH AF	Save the preferred column number and the flags.
CALL L2B56	Move down to the next row, shifting rows up as appropriate. Returns new row number in C.
LD B,\$00	Column 0.
CALL L2B1F	Find closest Screen Line Edit Buffer editable position to the right.
POP AF	A=Preferred column. Carry flag is reset indicating to produce an error beep.
JP L2A1E	Store editing position and print cursor, and then return.

## CURSOR-DOWN Key Handler Routine

Move down 1 row, attempting to place the cursor as close to the preferred column number as possible.

An error beep is produced if there is no row below.

Exit: Carry flag reset to indicate to produce an error beep.

L2ADB:	CALL L2A12	Remove cursor, restoring old attribute.
	LD E,A	E=Preferred column.
	PUSH DE	
	CALL L2B56	Move down to the next row, shifting rows up as appropriate. If moving onto a new BASIC line then
	POP DE	insert the previous BASIC line into the BASIC program if it has been altered. Returns new row number in C.
	JP NC,L2A18	Jump if there was no row below to set attribute at editing position so as to show the cursor, and return.
	LD B,E	B=Preferred column.
	CALL L2B28	Find closest Screen Line Edit Buffer editable position to the left else right, return column number in B.
	LD A,E	A=Preferred column.
	JP C,L2A1E	Jump if an editable position was found to store editing position and print cursor, and then return.

A blank row was found above, must be at the start of the BASIC program [???? Can this ever be the case?]

PUSH DE	Save the preferred column.
CALL L2B31	Move up to the previous row, shifting rows down as appropriate.
POP DE	
LD B,E	B=Preferred column.
CALL L2B1F	Find closest Screen Line Edit Buffer editable position to the right else to the left, returning column number in B.

LD A,E	A=Preferred column.
OR A	Reset carry flag to indicate to produce an error beep.
JP L2A1E	Store editing position and print cursor, and then return.

## CURSOR-LEFT Key Handler Routine

Move left 1 character, stopping if the start of the first row of the first BASIC line is reached.  
 An error beep is produced if there is no character to the left or no previous BASIC line to move to.  
 Exit: Carry flag reset to indicate to produce an error beep.

L2AFD:	CALL L2A12	Remove cursor, restoring old attribute. Returns with C=row, B=column.
	CALL L2B81	Find next Screen Line Edit Buffer editable position to left, wrapping to previous row as necessary.
	JP C,L2A1E	Jump if editable position found to store editing position and print cursor, and then return.

A blank row was found above, must be at the start of the BASIC program

JP L2A18	Set cursor attribute at existing editing position, and return. Carry flag is reset indicating to produce an error beep.
----------	---

## CURSOR-RIGHT Key Handler Routine

Move right 1 character, stopping if the end of the last row of the last BASIC line is reached.  
 An error beep is produced if there is no character to the right or no next BASIC line to move to.  
 Exit: Carry flag reset to indicate to produce an error beep.

L2B09:	CALL L2A12	Remove cursor, restoring old attribute.
	CALL L2B9E	Find next Screen Line Edit Buffer editable position to right, wrapping to next row if necessary.
	JP C,L2A1E	Jump if editable position found to store editing position and print cursor, and then return.

A blank row was found below, must be at the end of the BASIC program

PUSH AF	Save the carry flag and preferred column number.
CALL L2B31	Move up to the previous row, shifting rows down as appropriate.
LD B,\$1F	Column 31.
CALL L2C05	Find the last editable column position searching to the left, returning the column number in B. (Returns carry flag set if there is one)
POP AF	Carry flag is reset indicating to produce an error beep.
JP L2A1E	Store editing position and print cursor, and then return.

## Edit Buffer Routines — Part 1

### Find Closest Screen Line Edit Buffer Editable Position to the Right else Left

This routine searches the specified Screen Line Edit Buffer row from the specified column to the right looking for the first editable position. If one cannot be found then a search is made to the left.

Entry: B=Column number.  
 Exit : Carry flag set if character at specified column is editable.  
 B=Number of closest editable column.  
 HL=Address of closest editable position.

L2B1F:	PUSH DE	
	CALL L2BFA	Find Screen Line Edit Buffer editable position from previous column (or current column if the previous column does not exist) to the right, return column position in B.
	CALL NC,L2C05	If no editable character found then search to the left for an editable character, return column position in B.

POP DE  
RET

## Find Closest Screen Line Edit Buffer Editable Position to the Left else Right

This routine searches the specified Screen Line Edit Buffer row from the specified column to the left looking for the first editable position. If one cannot be found then a search is made to the right.

Entry: B=Column number.  
Exit : Carry flag set if character at specified column is editable.  
B=Number of closest editable column.  
HL=Address of closest editable position.

L2B28:	PUSH DE CALL L2C05  CALL NC,L2BFA  POP DE RET	Find Screen Line Edit Buffer editable position to the left, returning column position in B. If no editable character found then search from previous column (or current column if the previous column does not exist) to the right, return column position in B.
--------	---	---

## Insert BASIC Line, Shift Edit Buffer Rows Down If Required and Update Display File If Required

Called from the cursor up and down related key handlers. For example, when cursor up key is pressed the current BASIC line may need to be inserted into the BASIC program if it has been altered. It may also be necessary to shift all rows down should the upper scroll threshold be reached. If the cursor was on a blank row between BASIC lines then it is necessary to shift all BASIC lines below it up, i.e. remove the blank row.

Entry: C=Current cursor row number in the Screen Line Edit Buffer.  
Exit : C=New cursor row number in the Screen Line Edit Buffer.  
Carry flag set if a new row was moved to.

L2B31:	CALL L2CA2  JR NC,L2B55 PUSH BC CALL L30DA LD B,\$00 CALL L2E67  CALL NC,L2FA6  POP BC LD HL,\$F6F1 LD A,(HL) CP C JR C,L2B53	If current BASIC line has been altered and moved off of then insert it into the program. Jump if BASIC line was not inserted. [Could have saved 1 byte by using RET NC] Save the new cursor row and column numbers. DE=Start address in Screen Line Edit Buffer of the row specified in C. Column 0. Is this a blank row? i.e. Find editable position on this row to the right, returning column number in B. If no editable position found then the cursor is on a blank row so shift all BASIC lines below it up to close the gap. Retrieve the new cursor row and column numbers. Point to the editing area information. Fetch the upper scroll threshold. Is it on the threshold? Jump if on a row below the threshold.
--------	---	--

The upper row threshold for triggering scrolling the screen has been reached so proceed to scroll down one row

PUSH BC CALL L168E POP BC RET C	Save the new cursor row and column numbers. Shift all edit buffer rows down, and update display file if required.  Return if edit buffer rows were shifted.
--	--

The edit buffer rows were not shifted down

L2B53:	LD A,C OR A RET Z DEC C SCF RET	On the top row of the editing area?  Return with carry flag reset if on the top row. Move onto the previous row. Signal a new row was moved to.
--------	--	---

## Insert BASIC Line, Shift Edit Buffer Rows Up If Required and Update Display File If Required

Called from the cursor up and down related key handlers. For example, when cursor down key is pressed the current BASIC line may need to be inserted into the BASIC program if it has been altered. It may also be necessary to shift all rows up should the lower scroll threshold be reached. If the cursor was on a blank row between BASIC lines then it is necessary to shift all BASIC lines below it up, i.e. remove the blank row.

Entry: C=Current cursor row number in the Screen Line Edit Buffer.

Exit : C=New cursor row number in the Screen Line Edit Buffer.

Carry flag set if a new row was moved to.

L2B56:	PUSH BC	Save row number.
	CALL L30DA	DE=Start address in Screen Line Edit Buffer of row held in C, i.e. the new cursor row.
	LD B,\$00	Column 0.
	CALL L2E67	Is this a blank row? i.e. Find editable position on this row to the right, returning column number in B.
	POP BC	Get row number.
	JR C,L2B65	Jump if editable position found, i.e. the row exists. [Could have saved 2 bytes by using JP NC,\$2FA6 (ROM 0)]
	JP L2FA6	Cursor is on a blank row so shift all BASIC lines below it up to close the gap.
L2B65:	CALL L2C8E	Insert the BASIC Line into the BASIC program if the line has been altered.
	JR NC,L2B80	Jump if the line was inserted into the program. [Could have saved 1 byte by using RET NC]

The BASIC line was not inserted into the program. C=New cursor row number, B=New cursor column number, A=New cursor preferred column number

LD HL,\$F6F1	Point to the editing area information.
INC HL	Point to the 'Bottom Row Scroll Threshold' value. [Could have saved 1 byte by using LD HL,\$F6F2]
LD A,C	Fetch the new cursor row number.
CP (HL)	Is it on the lower scroll threshold?
JR C,L2B7E	Jump if on a row above the threshold.

The lower row threshold for triggering scrolling the screen has been reached so proceed to scroll up one row

PUSH BC	Save the new cursor row and column numbers.
PUSH HL	Save the editing area information address.
CALL L1658	Shift all edit buffer rows up, and update display file if required.
POP HL	
POP BC	
RET C	Return if edit buffer rows were shifted.

The edit buffer rows were not shifted up

INC HL	Point to the 'Number of Rows in the Editing Area' value.
LD A,(HL)	A=Number of rows in the editing area.
CP C	On the last row of the editing area?
RET Z	Return with carry flag reset if on the bottom row.
L2B7E:	INC C
	SCF
	Signal a new row was moved to.
L2B80:	RET

## Find Next Screen Line Edit Buffer Editable Position to Left, Wrapping Above if Required

This routine searches to the left to see if an editable position exists. If there is no editable position available to the left on the current row then the previous row is examined from the last column position.

Entry: B=Column number.

Carry flag reset.

Exit : Carry flag set if a position to the 'left' exists.

B=Number of new editable position.

HL=Address of new editable position.

L2B81:	LD D,A	Save the key code character.
--------	--------	------------------------------

## SPECTRUM +2 ROM o DISASSEMBLY

DEC B	Back one column position.
JP M,L2B8C	Jump if already at beginning of row.
LD E,B	E=Column number.
CALL L2C05	Find Screen Line Edit Buffer editable position to the left, returning column position in B.
LD A,E	A=Column number.
RET C	Return if the new column is editable, i.e. the cursor can be moved within this row.

Wrap above to the previous row

L2B8C:	PUSH DE	E=Store the column number.
	CALL L2B31	Move up to the previous row, shifting rows down as appropriate. If moving onto a new BASIC line then
	POP DE	insert the previous BASIC line into the BASIC program if it has been altered.
	LD A,E	A=Column number.
	RET NC	Return if there was no row above.

A row above exists

LD B,\$1F	Column 31.
CALL L2C05	Find the last editable column position searching to the left, returning the column number in B. (Returns carry flag set if there is one)
LD A,B	A=Column number of the closest editable position.
RET C	Return if an editable position was found, i.e. the cursor can be moved.

Return column 0

LD A,D	Restore the key code character.
LD B,\$00	Set column position 0.
RET	<b>[BUG]</b> - This should really ensure the carry flag is reset to signal that no editable position to the left exists, e.g. by using OR A. Fortunately, the carry flag is always reset when this routine is called and so the bug is harmless. Credit: Paul Farrow]

## Find Next Screen Line Edit Buffer Editable Position to Right, Wrapping Below if Required

This routine searches to the right to see if an editable position exists. If there is no editable position available to the right on the current row then the next row is examined from the first column position.

The routine is also called when a character key has been pressed and in this case if the cursor moves to the next row then a blank row is inserted and all affected rows are shifted down.

Entry: B=Column number.  
C=Row number.

Exit : Carry flag set if a position to the 'right' exists.  
B=Number of closest editable column, i.e. new column number.  
A=New column position, i.e. preferred column number or indentation column number.  
HL=Address of the new editable position.

L2B9E:	LD D,A	Save the key code character.
	INC B	Advance to the next column position.
	LD A,\$1F	Column 31.
	CP B	
	JR C,L2BAB	Jump if reached end of row.

New position is within the row

LD E,B	E=New column number.
CALL L2BFA	Find Screen Line Edit Buffer editable position from previous column to the right, returning column position in B.
LD A,E	A=New column number.
RET C	Return if the new column is editable, i.e. the cursor can be moved within this row.

Need to wrap below to the next row

L2BAB:	DEC B	B=Original column position.
--------	-------	-----------------------------

## SPECTRUM +2 ROM 0 DISASSEMBLY

PUSH BC PUSH HL LD HL,\$EC0D BIT 7,(HL) JR NZ,L2BE6	Save original column and row numbers. HL=Address of the new editable position. Editor flags. Got a key press? Jump if not.
---	--

A key is being pressed so need to insert a new row

CALL L30DA LD HL,\$0020 ADD HL,DE LD A,(HL) BIT 1,A JR NZ,L2BE6	DE=Start address in Screen Line Edit Buffer of the row specified in C.  Point to the flag byte for the current row.  Does the BASIC line row span onto another row? Jump if so to test the next row (it could just be the cursor).
--	---

The BASIC line row does not span onto another row, i.e. cursor at end of line

SET 1,(HL)  RES 3,(HL) LD HL,\$0023 ADD HL,DE EX DE,HL  POP HL POP BC PUSH AF CALL L2B56  POP AF CALL L30DA LD HL,\$0023 ADD HL,DE EX DE,HL RES 0,A SET 3,A CALL L2EF9	Signal that the row spans onto another row, i.e. a new blank row containing the cursor. Signal that the row is not the last row of the BASIC line. Point to the next row.  DE=Address of the next row. [Redundant calculation as never used. Could have saved 5 bytes] HL=Address of the new editable position. B=Original column number. C=Row number. Save flag byte for the previous row. Move down to the next row, shifting rows up as appropriate. Returns new row number in C. Retrieve flag byte for the previous row. DE=Start address in Screen Line Edit Buffer of the new row, as specified in C.  HL=Address of the row after the new row. DE=Address of the row after the new row. HL=Address of the new row. Signal 'not the start row of the BASIC line'. Signal 'end row of the BASIC line'. Insert a blank row into the Screen Edit Buffer at row specified by C, shifting rows down.
---	---

**[BUG -** When typing a line that spills over onto a new row, the new row needs to be indented. However, instead of the newly inserted row being indented, it is the row after it that gets indented. The indentation occurs within the Screen Line Edit Buffer and is not immediately reflected in the display file. When the newly typed line is executed or inserted into the program area, the Screen Line Edit Buffer gets refreshed and hence the effect of the bug is never normally seen. The bug can be fixed by inserting the following instructions. Credit: Paul Farrow.

LD HL,\$FFDD ADD HL,DE EX DE,HL  CALL L361A  LD A,B SCF RET	-35.  DE=Points to the start of the previous row.]  Indent the row by setting the appropriate number of null characters in the current Screen Line Edit Buffer row. A=First column after indentation. Signal not to produce an error beep.
---	--

Wrap below to the next row. Either a key was not being pressed, or a key was being pressed and the BASIC line spans onto a row below (which could contain the cursor only)

L2BE6:	POP HL POP BC PUSH DE CALL L2B56  POP DE  LD A,B	HL=Address of the new editable position. B=Original column position. E=New column number. Move down to the next row, shifting rows up as appropriate. If moving onto a new BASIC line then insert the previous BASIC line into the BASIC program if it has been altered. Returns new row number in C. A=Original column position.
--------	---	--

RET NC                      Return if there was no row below.

A row below exists

LD B,\$00                      Column 0.  
CALL L2BFA                      Find Screen Line Edit Buffer editable position to the right, returning column position in B.  
  
LD A,B                      A=New column position.  
RET C                      Return if an editable position was found, i.e. the cursor can be moved.

Return column 0

LD A,E                      A=Preferred column number.  
LD B,\$00                      Column 0.  
RET                      Return with carry flag reset.

## Find Screen Line Edit Buffer Editable Position from Previous Column to the Right

This routine finds the first editable character position in the specified Screen Line Edit Buffer row from the previous column to the right. It first checks the current column, then the previous column and then the columns to the right. The column containing the first non-null character encountered is returned.

Entry:        B=Column number to start searching from.  
              C=Row number.  
  
Exit :        Carry flag set if an editable character was found.  
              B=Number of closest editable column.

L2BFA:        PUSH DE                      Save registers.  
              PUSH HL  
              CALL L30DA                      DE=Start address in Screen Line Edit Buffer of the row specified in C.  
              CALL L2E67                      Find editable position on this row from the previous column to the right, returning column number in B.  
  
              JP L2C8B                      Restore registers and return. [Could have saved a byte by using JR \$2C2D (ROM 0)]

## Find Screen Line Edit Buffer Editable Position to the Left

This routine finds the first editable character position in the Screen Line Edit Buffer row from the current column to the left. It first checks the current column and returns this if it contains an editable character. Otherwise it searches the columns to the left and if an editable character is found then it returns the column to the right of it.

Entry:        B=Column number to start searching from.  
              C=Row number.  
  
Exit :        Carry flag set if an editable character was found.  
              B=Number of the column after the editable position.

L2C05:        PUSH DE                      Save registers.  
              PUSH HL  
              CALL L30DA                      DE=Start address in Screen Line Edit Buffer of the row specified in C.  
              CALL L2E89                      Find editable position from current column to the left, returning the column number in B.  
  
              JP L2C8B                      Restore registers and return. [Could have saved a byte by using JR \$2C2D (ROM 0)]

## Find Start of Word to Left in Screen Line Edit Buffer

This routine searches for the start of the current word to the left within the current Screen Line Edit Buffer. It is called from the WORD-LEFT key handler routine.

Entry:        C=Row number.  
  
Exit :        Carry flag set if word to the left is found.  
              B=Column position of the found word.

# SPECTRUM +2 ROM 0 DISASSEMBLY

L2C10:	PUSH DE	Save registers.
	PUSH HL	

Search towards the left of this row until a space or start of line is found

L2C12:	CALL L2B81	Find next Screen Line Edit Buffer editable position to left, moving to next row if necessary.
	JR NC,L2C2D	Jump if not editable, i.e. at start of line.
L2C17:	CALL L2A40	Get character at new position.
	CP ''	\$20. Is it a space?
	JR Z,L2C12	Jump back if it is, until a non-space or start of line is found.

Search towards the left of this row until the start of the word or start of the line is found

L2C1E:	CALL L2B81	Find next Screen Line Edit Buffer editable position to left, moving to next row if necessary.
	JR NC,L2C2D	Jump if not editable, i.e. at start of line.
	CALL L2A40	Get character at new position.
	CP ''	\$20. Is it a space?
	JR NZ,L2C1E	Jump back if it is not, until a space or start of line is found.

A space prior to the word was found

	CALL L2B9E	Find next Screen Line Edit Buffer editable position to right to start of the word, moving to next row if necessary. [Returns carry flag set since the character will exist]
L2C2D:	JR L2C8B	Jump forward to restore registers and return.

## Find Start of Word to Right in Screen Line Edit Buffer

This routine searches for the start of the current word to the right within the current Screen Line Edit Buffer. It is called from the WORD-RIGHT key handler routine.

Entry: C=Row number.  
Exit : Carry flag set if word to the right is found.  
B=Column position of the found word.

L2C2F:      PUSH DE                      Save registers.  
              PUSH HL

Search towards the right of this row until a space or end of line is found

L2C31:	CALL L2B9E	Find next Screen Line Edit Buffer editable position to right, moving to next row if necessary.
	JR NC,L2C51	Jump if none editable, i.e. at end of line.
	CALL L2A40	Get character at new position.
	CP ' '	\$20. Is it a space?
	JR NZ,L2C31	Jump back if it is not, until a space or end of line is found.

Search towards the right of this row until the start of a new word or end of the line is found

L2C3D:	CALL L2B9E	Find next Screen Line Edit Buffer editable position to right, moving to next row if necessary.
	JR NC,L2C51	Jump if none editable, i.e. at end of line.
	CALL L2E67	Find editable position on this row from the previous column to the right, returning column number in B.
	JR NC,L2C51	Jump if none editable, i.e. at start of next line.
	CALL L2A40	Get character at new position.
	CP ''	\$20. Is it a space?
	JR Z,L2C3D	Loop back until a non-space is found, i.e. start of a word.

Start of new word found

SCF	Indicate cursor position can be moved.
JR L2C8B	Jump forward to restore registers and return.

End of line or start of next line was found

L2C51:	CALL NC,L2B81	If no word on this row then find next Screen Line Edit Buffer editable position to left, moving to previous row if necessary thereby restoring the row number to its original value. [Carry flag is always reset by here so the test on the flag is unnecessary]
	OR A	Clear carry flag to indicate cursor position can not be moved.
	JR L2C8B	Jump forward to restore registers and return.

## Find Start of Current BASIC Line in Screen Line Edit Buffer

This routine searches for the start of the BASIC line, wrapping to the previous rows as necessary.  
It is called from the START-OF-LINE key handler routine.

Entry: C=Row number.  
Exit : Carry flag set if row is not blank.  
B=New cursor column.

L2C57:	PUSH DE	Save registers.
	PUSH HL	
L2C59:	CALL L30DA	DE=Start address in Screen Line Edit Buffer of the row specified in C.
	LD HL,\$0020	
	ADD HL,DE	Point to flag byte of next row.
	BIT 0,(HL)	On first row of the BASIC line?
	JR NZ,L2C6B	Jump if on the first row of the BASIC line.

Not on the first row of the BASIC line

CALL L2B31	Move up to the previous row, shifting rows down as appropriate. If moving onto a new BASIC line then insert the previous BASIC line into the BASIC program if it has been altered.
JR C,L2C59	Jump back if still on the same BASIC line, i.e. was not on first row of the BASIC line.
JR L2C8B	Jump forward to restore registers and return.

On the first row of the BASIC line, so find the starting column

L2C6B:	LD B,\$00	Column 0.
	CALL L2BFA	Find Screen Line Edit Buffer editable position to the right, return column position in B. (Returns carry flag reset if blank row)
	JR L2C8B	Jump forward to restore registers and return.

## Find End of Current BASIC Line in Screen Line Edit Buffer

This routine searches for the end of the BASIC line, wrapping to the next rows as necessary.  
It is called from the END-OF-LINE key handler routine.

Entry: C=Row number.  
Exit : Carry flag set if row is not blank.  
B=New cursor column.

L2C72:	PUSH DE	Save registers.
	PUSH HL	
L2C74:	CALL L30DA	DE=Start address in Screen Line Edit Buffer of the row specified in C.
	LD HL,\$0020	
	ADD HL,DE	Point to flag byte of next row.
	BIT 3,(HL)	On last row of the BASIC line?
	JR NZ,L2C86	Jump if on the last row of the BASIC line.

Not on the last row of the BASIC line

CALL L2B56	Move down to the next row, shifting rows up as appropriate. If moving onto a new BASIC line then insert the previous BASIC line into the BASIC program if it has been altered. Returns new row number in C.
JR C,L2C74	Jump back if still on the same BASIC line, i.e. was not on last row of the BASIC line.
JR L2C8B	Jump forward to restore registers and return.

On the last row of the BASIC line, so find the last column

L2C86:	LD B,\$1F CALL L2C05	Column 31. Find the last editable column position searching to the left, returning the column number in B. (Returns carry flag reset if blank row)
L2C8B:	POP HL POP DE RET	Restore registers.

## Insert BASIC Line into Program if Altered

L2C8E:	LD A,(\$EC0D) BIT 3,A SCF RET Z CALL L30DA LD HL,\$0020 ADD HL,DE BIT 3,(HL) SCF RET Z JR L2CB4	Editor flags. Has the current line been altered? Signal line not inserted into BASIC program. Return if it has not. DE=Start address in Screen Line Edit Buffer of the row specified in C.  HL points to the flag byte for the row. Is this the end of the BASIC line? Signal line not inserted into BASIC program. Return if it is not. Insert line into BASIC program.
--------	---	--

## Insert Line into BASIC Program If Altered and the First Row of the Line

L2CA2:	LD A,(\$EC0D) BIT 3,A SCF RET Z CALL L30DA LD HL,\$0020 ADD HL,DE BIT 0,(HL) SCF RET Z	Editor flags. Has current line been altered? Signal success. Return if it has not. DE=Start address in Screen Line Edit Buffer of the row specified in C.  Point to the flag byte for the row. Is this the first row of the BASIC line? Signal success. Return if it is not.
--------	---	---

## Insert Line into BASIC Program

This routine parses a line and if valid will insert it into the BASIC program. If in calculator mode then the line is not inserted into the BASIC program. If a syntax error is found then the location to show the error marker is determined.

Entry: C=Row number.

Exit : Carry flag reset if a syntax error.

Carry flag set if the BASIC line was inserted successfully, and C=Cursor row number, B=Cursor column number, A=Preferred cursor column number.

L2CB4:	LD A,\$02	Signal on first row of BASIC line.
--------	-----------	------------------------------------

Find the start address of the row in the Screen Line Edit Buffer

L2CB6:	CALL L30DA LD HL,\$0020 ADD HL,DE BIT 0,(HL) JR NZ,L2CC9 DEC C JP P,L2CB6	DE=Start address in Screen Line Edit Buffer of the row specified in C.  Point to the flag byte for the row. First row of the BASIC line? Jump ahead if so. Move to previous row. Jump back until found the first row of the BASIC line or the top of the screen.
--------	---	--

First row of the BASIC line is above the screen

## SPECTRUM +2 ROM 0 DISASSEMBLY

```
LD C,$00
LD A,$01
```

Row 0.  
Signal first row of BASIC line above screen.

DE=Start address of the first row of the BASIC line

HL=Address of the flag byte for the first row of the BASIC line

L2CC9:	LD HL,\$EC00	BASIC line insertion flags.
	LD DE,\$EC03	BASIC line insertion error flags.
	OR \$80	Signal location of cursor not yet found.
	LD (HL),A	
	LD (DE),A	
	INC HL	
	INC DE	
	LD A,\$00	[Could have saved 1 byte by using XOR A]
	LD (HL),A	Starting column number of the first visible row of the BASIC line being entered.
	LD (DE),A	
	INC HL	
	INC DE	
	LD A,C	Fetch the row number of the first visible row of the BASIC line being entered.
	LD (HL),A	Store the start row number of the first visible row of the BASIC line being entered.
	LD (DE),A	
	LD HL,\$0000	
	LD (\$EC06),HL	No editable characters in the line prior to the cursor.
	CALL L3385	Copy 'Insert Keyword Representation Into Keyword Construction Buffer' routine to RAM.
	CALL L3C0E	Tokenize the typed BASIC line.
	PUSH IX	IX=Address of cursor settings.
	CALL L1F3F	Use Normal RAM Configuration (physical RAM bank 0).
	CALL L026B	Syntax check/execute the command line.
	CALL L1F64	Use Workspace RAM configuration (physical RAM bank 7).
	POP IX	IX=Address of cursor settings.
	LD A,(\$5C3A)	ERR_NR. Fetch error code.
	INC A	Was an error code set?
	JR NZ,L2D15	Jump ahead if so.
	LD HL,\$EC0D	Editor flags.
	RES 3,(HL)	Signal line has not been altered.
	CALL L3684	Reset to 'L' Mode.
	LD A,(\$EC0E)	Fetch mode.
	CP \$04	Calculator mode?
	CALL NZ,L154E	If not calculator mode then relist the BASIC program.
	CALL L2719	Produce success beep.
	CALL L2A2D	Get current cursor position (C=Row, B=Column, A=Preferred column).
	SCF	Set the carry flag to signal that that BASIC line was inserted successfully.
	RET	

A syntax error occurred

L2D15:	LD HL,\$EC00 LD DE,\$EC03 LD A,(DE) RES 7,A LD (HL),A INC HL INC DE LD A,(DE) LD (HL),A INC HL INC DE LD A,(DE) LD (HL),A  CALL L3C0A JR C,L2D30	BASIC line insertion flags. BASIC line insertion error flags. Fetch the BASIC line insertion error flags. Signal location of cursor found. Update the BASIC line insertion flags with the error flags.             Restore the initial column number, i.e. column 0.             Restore the initial row number, i.e. row number of the first visible row of the BASIC line being entered. Locate the position to insert the error marker into the typed BASIC line. Jump if the error marker was found.
--------	---	--

Assume the error maker is at the same position as the cursor

## SPECTRUM +2 ROM 0 DISASSEMBLY

LD BC,(\$EC06)	Fetch the number of editable characters in the line prior to the cursor within the Screen Line Edit Buffer.
----------------	---

The position of the error marker within the typed BASIC line has been determined. Now shift the cursor to the corresponding position on the screen.

L2D30: LD HL,(\$EC06)	Fetch the number of editable characters in the line prior to the cursor within the Screen Line Edit Buffer.
OR A	
SBC HL,BC	HL=Difference between the cursor and the error marker positions (negative if the error marker is after the cursor).
PUSH AF	Save the flags.
PUSH HL	HL=Difference between the cursor and error marker.
CALL L2A2D	Get current cursor position, returning C=row number, B=column number, A=preferred column number.
POP HL	HL=Difference between the cursor and error marker.
POP AF	Restore the flags.
JR C,L2D50	Jump if error marker is after the cursor position.
JR Z,L2D6B	Jump if cursor is at the same location as the error marker.

The error marker is before the cursor position. Move the cursor back until it is at the same position as the error marker.

L2D41: PUSH HL	Save the number of positions to move.
LD A,B	B=Cursor column number.
CALL L2B81	Find previous editable position to the left in the Screen Line Edit Buffer, moving to previous row if necessary.
POP HL	Retrieve the number of positions to move.
JR NC,L2D6B	Jump if no previous editable position exists.
DEC HL	Decrement the number of positions to move.
LD A,H	
OR L	
JR NZ,L2D41	Jump back if the cursor position requires further moving.
JR L2D6B	Jump ahead to continue.

The error marker is after the cursor position. Move the cursor back until it is at the same position as the error marker.

L2D50: PUSH HL	Save the number of positions that the error marker is before the cursor. This will be a negative number if the cursor is after the error marker.
L2D51: LD HL,\$EC0D	Editor flags.
RES 7,(HL)	Signal 'got a key press'. Used in routine at \$2B9E (ROM 0) to indicate that a new character has caused the need to shift the cursor position.
POP HL	Retrieve the negative difference in the cursor and error marker positions.
EX DE,HL	DE=Negative difference in the cursor and error marker positions.
LD HL,\$0000	Make the negative difference a positive number by subtracting it from 0.
OR A	
SBC HL,DE	HL=Positive difference in the cursor and error marker positions.
L2D5E: PUSH HL	Save the number of positions to move.
LD A,B	B=Cursor column number.
CALL L2B9E	Find next editable position to the right in the Screen Line Edit Buffer, moving to next row if necessary.
POP HL	Retrieve the number of positions to move.
JR NC,L2D6B	Jump if no next editable position exists.
DEC HL	Decrement the number of positions to move.
LD A,H	
OR L	
JR NZ,L2D5E	Jump back if the cursor position requires further moving.

The cursor position is at the location of the error marker position

L2D6B: LD HL,\$EC0D	Editor flags.
SET 7,(HL)	Set 'waiting for key press' flag.

**[BUG]** - When moving the cursor up or down, an attempt is made to place the cursor at the same column position that it had on the previous row (the preferred column). If this is not possible then the cursor is placed at the end of the row. However, it is the intention that the preferred column is still remembered and hence an attempt is made to place the cursor at this column whenever it is subsequently moved. However, a bug at this point in the ROM causes the preferred column position for the cursor to be overwritten with random data. If the cursor was moved from its original position into its

error position then the preferred column gets set to zero and the next up or down cursor movement will cause the cursor marker to jump to the left-hand side of the screen. However, if the cursor remained in the same position then the preferred column gets set to a random value and so on the next up or down cursor movement the cursor marker can jump to a random position on the screen. The bug can be reproduced by typing a line that is just longer than one row, pressing enter twice and then cursor down. The cursor marker will probably jump somewhere in the middle of the screen. Press an arrow again and the computer may even crash. Credit: Ian Collier (+3), Andrew Owen (128)] [The bug can be fixed by pre-loading the A register with the current preferred column number. Credit: Paul Farrow.

LD A,(\$F6F0)	Fetch the preferred column position.]
CALL L2A37	Store cursor editing position.
LD A,\$17	Paper 2, Ink 7 - Red.
CALL L3AB7	Set the cursor colour to show the position of the error.
OR A	Reset the carry flag to signal that a syntax error occurred.
RET	

## Fetch Next Character from BASIC Line to Insert

This routine fetches a character from the BASIC line being inserted. The line may span above or below the screen, and so the character is retrieved from the appropriate buffer.

Exit : A=Character fetched from the current position, or 'Enter' if end of line found.

L2D7A:	LD HL,\$EC00	Point to the 'insert BASIC line' details.
	BIT 7,(HL)	Has the column with the cursor been found?
	JR Z,L2D88	Jump if it has been found.
	LD HL,\$EC06)	
	INC HL	Increment the count of the number of editable characters in the BASIC line up to the cursor.
L2D88:	LD (\$EC06),HL	
	LD HL,\$EC00	Point to the 'insert BASIC line' details.
	LD A,(HL)	Fetch flags.
	INC HL	
	LD B,(HL)	Fetch the column number of the character being examined.
	INC HL	
	LD C,(HL)	Fetch the row number of the character being examined.
	PUSH HL	
	AND \$0F	Extract the status code.

Register A:

Bit 0: 1=First row of the BASIC line off top of screen.

Bit 1: 1=On first row of the BASIC line.

Bit 2: 1=Using lower screen and only first row of the BASIC line visible.

Bit 3: 1=At end of last row of the BASIC line (always 0 at this point).

LD HL,L2DAB	Jump table to select appropriate handling routine.
CALL L3F75	Call handler routine.

Register L:

\$01 - A character was returned from the Above-Screen Line Edit Buffer row.

\$02 - A character was returned from the Screen Line Edit Buffer row.

\$04 - A character was returned from the Below-Screen Line Edit Buffer row.

\$08 - At the end of the last row of the BASIC line.

Register A holds the character fetched or 'Enter' if at the end of the BASIC line.

	LD E,L	E=Return status.
	POP HL	
	JR Z,L2D9F	Jump if no match found.
	LD A,\$0D	A='Enter' character.
L2D9F:	LD (HL),C	Save the next character position row to examine.
	DEC HL	
	LD (HL),B	Save the next character position column to examine.
	DEC HL	
	PUSH AF	Save the character.
	LD A,(HL)	Fetch the current status flags.
	AND \$F0	Keep the upper nibble.

OR E  
LD (HL),A  
POP AF  
RET

Update the location flags that indicate where to obtain the next character from.  
Store the status flags.  
Retrieve the character.

## Fetch Next Character Jump Table

Jump to one of three handling routines when fetching the next character from the BASIC line to insert.

L2DAB:	DEFB \$03	Number of table entries.
	DEFB \$02	On first row of the BASIC line.
	DEFW L2DD2	
	DEFB \$04	Using lower screen and only first row of the BASIC line visible.
	DEFW L2E0F	
	DEFB \$01	First row of the BASIC line off top of screen.
	DEFW L2DB5	

## Fetch Character from the Current Row of the BASIC Line in the Screen Line Edit Buffer

Fetch character from the current row of the BASIC line in the Screen Line Edit Buffer, skipping nulls until the end of the BASIC line is found.

Entry: C=Row number.  
Exit : L=\$01 - A character was returned from the Above-Screen Line Edit Buffer row, with A holding the character.  
\$02 - A character was returned from the Screen Line Edit Buffer row, with A holding the character.  
\$04 - A character was returned from the Below-Screen Line Edit Buffer row, with A holding the character.  
\$08 - At the end of the last row of the BASIC line, with A holding an 'Enter' character.  
Zero flag set to indicate a match from the handler table was found.

Table entry point - First row of BASIC line off top of screen

L2DB5:	CALL L32DD	Find row address in Above-Screen Line Edit Buffer, return in DE.
L2DB8:	CALL L2E34	Fetch character from Above-Screen Line Edit Buffer row.
	JR NC,L2DC4	Jump if end of row reached.
	CP \$00	Is it a null character, i.e. not editable?
	JR Z,L2DB8	Jump back if so until character found or end of row reached.
	LD L,\$01	Signal a character was returned from the Above-Screen Line Edit Buffer row, with A holding the character.
	RET	Return with zero flag reset to indicate match found.

End of row reached - no more editable characters in Above-Screen Line Edit Buffer row

L2DC4:	INC C	Next row.
	LD B,\$00	Column 0.
	LD HL,(\$F9DB)	<b>[BUG -</b> This should be LD HL,\$F9DB. The bug manifests itself when Enter is pressed on an edited BASIC line that goes off the top of the screen and causes corruption to that line. The bug at \$30F6 (ROM 0) that sets default data for the Below-Screen Line Edit Buffer implies that originally there was the intention to have a pointer into the next location to use within that buffer, and so it seems to reasonable to assume the same arrangement would have been intended for the Above-Screen Line Edit Buffer. If that were the case then the instruction here was intended to fetch the next address within the Above-Screen Line Edit Buffer. Credit: Ian Collier (+3), Andrew Owen (128)]
	LD A,C	Fetch the row number.
	CP (HL)	Exceeded last row of Above-Screen Line Edit Buffer?
	JR C,L2DB5	Jump back if not exceeded last row the Above-Screen Line Edit Buffer.

All characters from rows off top of screen fetched so continue onto the rows on screen [Note it is not possible to have more than 20 rows off the top of the screen]

LD B,\$00	Column 0.
LD C,\$00	Row 0. This is the first visible row of the BASIC line on screen.

Table entry point - On visible row of BASIC line

## SPECTRUM +2 ROM 0 DISASSEMBLY

C=Row number of the first visible row of the BASIC line in the Screen Line Edit Buffer  
 B=Starting column number of the first visible row of the BASIC line in the Screen Line Edit Buffer

L2DD2:	PUSH HL LD HL,\$F6EE LD A,(HL) CP C JR NZ,L2DE4	Save address of the table entry. Point to the cursor position details. Fetch the row number of the cursor. Is cursor on the first visible row of the BASIC line? Jump if not.
--------	---	---

Cursor on first visible row of the BASIC line in the Screen Line Edit Buffer.

L2DE4: L2DE5:	INC HL LD A,(HL) CP B JR NZ,L2DE4 LD HL,\$EC00 RES 7,(HL) POP HL CALL L30DA CALL L2E34  JR NC,L2DF4 CP \$00 JR Z,L2DD2	Fetch the column number of the cursor. Reached the column with the cursor in the first visible row of the BASIC line? Jump if not. BASIC line insertion flags. Indicate that the column with the cursor has been found. Retrieve address of the table entry. DE=Start address in Screen Line Edit Buffer of the row specified in C. Fetch character from Screen Line Edit Buffer row at column held in B, then increment B. Jump if end of row reached. Is the character a null, i.e. not editable? Jump back if null to keep fetching characters until a character is found or the end of the row is reached.
------------------	--	--

A character in the current row of the BASIC line was found

LD L,\$02  RET	L=Signal a character was returned from the Screen Line Edit Buffer row, with A holding the character. Return with zero flag reset to indicate match found.
----------------------	---

End of row reached - no editable characters in the Screen Line Edit Buffer row

L2DF4:	LD HL,\$0020 ADD HL,DE BIT 3,(HL) JR Z,L2E01	Point to the flag byte for the row. Is it the last row of the BASIC line? Jump if not.
--------	---	--

On last row of the BASIC line and finished fetching characters from the line

LD L,\$08 LD A,\$0D RET	L=Signal at the end of the last row of the BASIC line. A='Enter' character. Return with zero flag reset to indicate match found.
-------------------------------	--

Not on the last row of the BASIC line so move to the beginning of the next, if it is on screen.

L2E01:	LD HL,\$F6F3 INC C LD A,(HL) CP C LD B,\$00 JR NC,L2DE5	Point to the 'top row scroll threshold' value. Next row of the BASIC line in the Screen Line Edit Buffer. Fetch the number of the last row in the Screen Line Edit Buffer. Exceeded the upper scroll threshold? Column 0. Jump back if not to retrieve the character from the next row.
--------	--	--

The upper row threshold for triggering scrolling the screen has been reached so proceed to scroll up one line

LD B,\$00 LD C,\$01	Column 0. [Redundant byte] Row 1. (Row 0 holds a copy of the last row visible on screen)
------------------------	---

Table entry point - Using lower screen and only top row of a multi-row BASIC line is visible

L2E0F: L2E12:	CALL L31E9 CALL L2E34	Find the address of the row specified by C in Below-Screen Line Edit Buffer, into DE. Fetch character from Below-Screen Line Edit Buffer row, incrementing the column number.
------------------	--------------------------	--

## SPECTRUM +2 ROM o DISASSEMBLY

JR NC,L2E1E	Jump if end of row reached.
CP \$00	Is the character a null, i.e. not editable?
JR Z,L2E12	Jump back if null to keep fetching characters until a character is found or the end of the row is reached.
LD L,\$04	L=Signal a character was returned from the Below-Screen Line Edit Buffer row, with A holding the character.
RET	Return with zero flag reset to indicate match found.

End of row reached - no editable characters in the (below screen) Below-Screen Line Edit Buffer row

L2E1E:	LD HL,\$0020	
	ADD HL,DE	Point to the flag byte for the row.
	BIT 3,(HL)	Is it the last row of the BASIC line?
	JR NZ,L2E2F	Jump if so.
	INC C	Next row.
	LD B,\$00	Column 0.
	LD A,(\$F6F5)	Fetch number of rows in the Below-Screen Line Edit Buffer.
	CP C	Exceeded last line in Below-Screen Line Edit Buffer?
	JR NC,L2E0F	Jump back if not to retrieve the character from the next row.

All characters from rows off bottom of screen fetched so return an 'Enter' [Note it is not possible to have more than 20 rows off the bottom of the screen]

L2E2F:	LD L,\$08	L=Signal at the end of the last row of the BASIC line.
	LD A,\$0D	A='Enter' character.
	RET	Return with zero flag reset to indicate match found.

### Fetch Character from Edit Buffer Row

L2E34:	LD A,\$1F	Column 31.
	CP B	Is column
	CCF	
	RET NC	Return if B is greater than 31.
	LD L,B	
	LD H,\$00	HL=Column number.
	ADD HL,DE	
	LD A,(HL)	Fetch the character at the specified column.
	INC B	Increment the column number.
	SCF	Signal character fetched.
	RET	

### Upper Screen Rows Table

Copied to \$EC15-\$EC16.

L2E41:	DEFB \$01	Number of bytes to copy.
	DEFB \$14	Number of editing rows (20 for upper screen).

### Lower Screen Rows Table

Copied to \$EC15-\$EC16.

L2E43:	DEFB \$01	Number of bytes to copy.
	DEFB \$01	Number of editing rows (1 for lower screen).

### Reset to Main Screen

L2E45:	LD HL,\$5C3C	TVFLAG.
	RES 0,(HL)	Signal using main screen.
	LD HL,L2E41	Upper screen lines table.

LD DE,\$EC15  
JP L3F61

Destination workspace variable. The number of editing rows on screen.  
Copy one byte from \$2E42 (ROM 0) to \$EC15

## Reset to Lower Screen

L2E53: LD HL,\$5C3C  
SET 0,(HL)  
LD BC,\$0000  
CALL L3751  
LD HL,L2E43  
LD DE,\$EC15  
JP L3F61

TVFLAG.  
Signal using lower screen.  
  
Perform 'PRINT AT 0,0;'.  
Lower screen lines table.  
Destination workspace variable. The number of editing rows on screen.  
Copy one byte from \$2E44 (ROM 0) to \$EC15

## Find Edit Buffer Editable Position from Previous Column to the Right

This routine finds the first editable character position in the specified edit buffer row from the previous column to the right. It first checks the current column, then the previous column and then the columns to the right. The column containing the first non-null character encountered is returned.

Entry: B =Column number to start searching from.  
DE=Start of row in edit buffer.  
Exit : Carry flag set if an editable character was found.  
HL=Address of closest editable position.  
B =Number of closest editable column.

L2E67: LD H,\$00  
LD L,B  
ADD HL,DE  
LD A,(HL)  
CP \$00  
SCF  
RET NZ  
LD A,B  
OR A  
JR Z,L2E81  
PUSH HL  
DEC HL  
LD A,(HL)  
CP \$00  
SCF  
POP HL  
RET NZ

[Could have saved 1 byte by calling routine at \$2EA1 (ROM 0)]  
HL=Column number.  
HL=Address in edit buffer of the specified column.  
Fetch the contents.  
Is it a null character, i.e. end-of-line or past the end-of-line?  
  
Return if this character is part of the edited line.

L2E7C: LD A,(HL)  
CP \$00  
SCF  
RET NZ

Get the current character.  
Is it a null (i.e. end-of-line)?  
Signal position is editable.  
Return if this character is part of the edited line.

L2E81: INC HL  
INC B  
LD A,B  
CP \$1F  
JR C,L2E7C  
RET

Advance to the next position.  
Increment the column number.  
  
Reached the end of the row?  
Jump back if more columns to check.  
Return with carry flag reset if specified column position does not exist.

## Find Edit Buffer Editable Position to the Left

This routine finds the first editable character position in the specified edit buffer row from the current column to the left. It first checks the current column and returns this if it contains an editable character. Otherwise it searches the columns to the left and if an editable character is found then it returns the column to the right of it.

Entry: B =Column number to start searching from.  
DE=Start of row in edit buffer.  
Exit : Carry flag set if an editable character was found.  
HL=Address of closest editable position.

B =Number of the column after the editable position.

L2E89:	LD H,\$00	[Could have saved 1 byte by calling routine at \$2EA1 (ROM 0)]
	LD L,B	HL=Column number.
	ADD HL,DE	HL=Address in edit buffer of the specified column.
	LD A,(HL)	Fetch the contents.
	CP \$00	Is it a null character, i.e. end-of-line or past the end-of-line?
	SCF	Signal position is editable.
	RET NZ	Return if an editable character was found.
L2E92:	LD A,(HL)	Get the current character.
	CP \$00	Is it a null, i.e. non-editable?
	JR NZ,L2E9E	Jump if not.
	LD A,B	At column 0?
	OR A	
	RET Z	Return if so.
	DEC HL	Next column position to test.
	DEC B	Decrement column index number.
	JR L2E92	Repeat test on previous column.
L2E9E:	INC B	Advance to the column after the editable position.
	SCF	Signal position is editable.
	RET	

## Fetch Edit Buffer Row Character

Entry: DE=Add of edit buffer row.  
B =Column number.

Exit : A =Character at specified column.

[Not used by the ROM]

L2EA1:	LD H,\$00	
	LD L,B	HL=Column number.
	ADD HL,DE	HL=Address in edit buffer of the specified column.
	LD A,(HL)	Get the current character.
	RET	

## Insert Character into Screen Line Edit Buffer

Called when a non-action key is pressed. It inserts a character into the Screen Line Edit Buffer if there is room.

Entry: A=Character code.  
B=Cursor column position.  
C=Cursor row position.

L2EA7:	LD HL,\$EC0D	Editor flags.
	OR A	Clear carry flag. [Redundant since carry flag return state never checked]
	BIT 0,(HL)	Is the Screen Line Edit Buffer is full?
	RET NZ	Return if it is.
	PUSH BC	Save cursor position.
	PUSH AF	Save key code. [Redundant since \$30DA (ROM 0) preserves AF]
	CALL L30DA	DE=Start address in Screen Line Edit Buffer of the row specified in C.
	POP AF	Get key code. [Redundant since \$30DA (ROM 0) preserves AF]

Insert the character into the current row. If a spill from this row occurs then insert that character into the start of the following row and shift all existing characters right by one. Repeat this process until all rows have been shifted.

L2EB4:	CALL L16CB	Insert character into edit buffer row at current cursor position, shifting the row right. Returns carry flag reset. Zero flag will be set if byte shift out of last column position was \$00.
	PUSH AF	Save key code and flags.
	EX DE,HL	HL=Address of edit buffer row. DE=Address of flags.
	CALL L362A	Print a row of the edit buffer to the screen.
	EX DE,HL	DE=Address of edit buffer row. HL=Address of flags.
	POP AF	Get key code and flags.

## SPECTRUM +2 ROM o DISASSEMBLY

CCF	Sets the carry flag since it was reset via the call to \$16CB (ROM 0). [Redundant since never tested]
JR Z,L2EF2	Jump ahead to make a return if there was no spill out from column 31, with the carry flag set.

There was a spill out from the current row, and so this character will need to be inserted as the first character of the following row.  
If this is the last row of the BASIC line then a new row will need to be inserted.

PUSH AF	Save key code.
LD B,\$00	First column in the next row.
INC C	Next row.
LD A,(\$EC15)	The number of editing rows on screen.
CP C	Has the bottom of the Screen Line Edit Buffer been reached?
JR C,L2EEE	Jump ahead if so.

The editing screen is not full

LD A,(HL)	Fetch contents of flag byte for the row (byte after the 32 columns).
LD E,A	E=Old flags.
AND \$D7	Mask off 'last row of BASIC line' flag. [Other bits not used, could have used AND \$F7]
CP (HL)	Has the status changed?
LD (HL),A	Store the new flags, marking it as not the last BASIC row.
LD A,E	A=Original flags byte for the row.
SET 1,(HL)	Signal that the row spans onto another row.
PUSH AF	Save the flags.
CALL L30DA	DE=Start address in Screen Line Edit Buffer of the following row, as specified in C.
POP AF	Fetch the flags.
JR Z,L2EE8	Jump if the character was not inserted into the last row of the BASIC line.

The character was inserted into the last row of the BASIC line causing a spill of an existing character into a new row, and therefore a new 'last' row needs to be inserted.

RES 0,A	Signal not the first row of the BASIC line.
CALL L2EF9	Insert a blank line into the Screen Edit Buffer.
JR NC,L2EF2	Jump if the buffer is full to exit.
CALL L361A	Indent the row by setting the appropriate number of null characters in the current Screen Line Edit Buffer row.
POP AF	Get key code.
JR L2EB4	Jump back to insert the character in the newly inserted row. [Could have saved 2 bytes by using JR \$2EEB (ROM 0)]

The character was not inserted into the last row of the BASIC line, so find the first editable position on the following row, i.e. skip over any indentation.

L2EE8:	CALL L2E67	Find editable position on this row from the previous column to the right, returning column number in B.
	POP AF	Get key code.
	JR L2EB4	Jump back to insert the character into the first editable position of next the row.

The Screen Edit Line Buffer is full and the character insertion requires shifting of all rows that are off screen in the Below-Screen Line Edit Buffer.

L2EEE:	POP AF	Get key code.
	CALL L3194	Insert the character at the start of the Below-Screen Line Edit Buffer, shifting all existing characters to the right.

All paths join here

L2EF2:	POP BC	Retrieve cursor position.
	RET	

## Insert Blank Row into Screen Edit Buffer, Shifting Rows Down

This routine inserts a blank row at the specified row, shifting affected rows down.

Entry: C=Row number to insert the row at.

Exit : Carry flag set to indicate edit buffer rows were shifted.

L2EF4:           CALL L30DA  
                  LD A,\$09

DE=Start address in Screen Line Edit Buffer of the row specified in C.  
Signal 'first row' and 'last row', indicating a new blank row.

DE=Address of row within Screen Line Edit Buffer.

C=Row number to insert the row at.

A=Screen Line Edit Buffer row flags value.

```
L2EF9:    PUSH BC
           PUSH DE
           LD B,C
           LD HL,L2F15
           LD C,A
           PUSH BC
           CALL L1694
```

Save registers.

B=Row number.

The empty row data.

C=Flags for the row.

Shift all Screen Line Edit Buffer rows down and insert a new blank row, updating the display file if required.

POP BC

LD A,C

JR NC,L2F12

A=Flags for the row.

Jump if no edit buffer rows were shifted.

Rows were shifted down

```
LD C,B
CALL L30DA
LD HL,$0020
ADD HL,DE
LD (HL),A
SCF
```

B=Row number, where the new blank row now is.

DE=Start address in Screen Line Edit Buffer of the row specified in C.

Point to the flag byte for the row.

Store the flag byte value for the row.

Signal edit buffer rows were shifted.

```
L2F12: POP DE
        POP BC
        RET
```

Restore registers.

### Empty Edit Buffer Row Data

[illegible]

32 null column markers, i.e. none of the columns are editable.

```

DEFB $00
DEFB $00
DEFB $00
DEFB $00
DEFB $00
DEFB $00
DEFB $00
DEFB $00
DEFB $00
DEFB $09

```

Flags: Bit 0: 1=The first row of the BASIC line. Bit 1: 0=Does not span onto another row. Bit 2: 0=Not used (always 0). Bit 3: 1=The last row of the BASIC line. Bit 4: 0=No associated line number. Bit 5: 0=Not used (always 0). Bit 6: 0=Not used (always 0). Bit 7: 0=Not used (always 0).

```

DEFW $0000

```

There is no BASIC line number associated with this edit row.

## Delete a Character from a BASIC Line in the Screen Line Edit Buffer

Delete a character at the specified position, shifting subsequent characters left as applicable.

Entry: B=Column number.

C=Row number.

```

L2F38:    PUSH BC          Save initial cursor row and column numbers.
          CALL L30DA       DE=Start address in Screen Line Edit Buffer of the row specified in C.
          PUSH BC          Stack initial cursor row and column numbers again.

```

Enter a loop to find the last row of the BASIC line or the end of the visible screen, whichever comes first

```

L2F3D:    LD HL,$0020      Point to the flag byte for this row.
          ADD HL,DE         Does the row span onto another row?
          BIT 1,(HL)        A null character will be inserted. [Could have saved 1 byte by using XOR A and
          LD A,$00          placing it above the BIT 1,(HL) instruction]
          JR Z,L2F57        Jump ahead if the row does not span onto another row, i.e. the last row.

```

The row spans onto another

```

          INC C             C=Advance to the next row.
          LD HL,$0023
          ADD HL,DE
          EX DE,HL          DE points to the first character of the next row. HL points to the first character of the
                           current row.
          LD A,($EC15)       A=Number of editing lines.
          CP C              Has the end of the screen been reached?
          JR NC,L2F3D        Jump back if within screen range to find the last row of the BASIC line.

```

The end of the screen has been reached without the end of the BASIC line having been reached

```

          DEC C             Point to last row on screen.
          CALL L31EF        Shift all characters of the BASIC Line held within the Below-Screen Line Edit Buffer.

```

A loop is entered to shift all characters to the left, beginning with the last row of the BASIC line in the Screen Line Edit Buffer and until the row that matches the current cursor position is reached.

```

L2F57:    POP HL           Fetch the initial cursor row and column numbers.
L2F58:    PUSH HL          Stack initial cursor row and column numbers.
          CALL L30DA       DE=Start address in Screen Line Edit Buffer of the last row, as specified in C.
          POP HL           HL=Initial cursor row and column numbers.
          LD B,A           B=Character to insert.
          LD A,C           A=Row number to delete from.
          CP L             Deleting from the same row as the cursor is on within the BASIC line?
          LD A,B           A=Character to insert.
          PUSH AF          Save the flags status.
          JR NZ,L2F67      Jump if not deleting from the row containing the cursor.

```

## SPECTRUM +2 ROM o DISASSEMBLY

Deleting from the row matching the cursor position within the BASIC line, therefore only shift those bytes after the cursor position

LD B,H	B=Initial column number.
JR L2F70	Jump ahead to continue, with zero flag set to indicate deleting from the row contain the cursor.

Deleting on row after that matching the cursor position, therefore shift all editable characters within the row

L2F67:	PUSH AF	Save the character to insert.
	PUSH HL	Save initial cursor row and column numbers.
	LD B,\$00	
	CALL L2E67	Find first editable position on this row searching to the right, returning column number in B.
	POP HL	HL=Initial cursor row and column numbers.
	POP AF	A=Character to insert, and zero flag reset to indicate not deleting from the row contain the cursor.

DE=Start address of Screen Line Edit Buffer row.

A=Character to shift into right of row.

B=The column to start shifting at.

C=Row number to start shifting from.

Zero flag is set if deleting from the row matching the cursor position.

L2F70:	PUSH HL	HL=Initial cursor row and column numbers.
	LD HL,\$F6F4	Deleting flags.
	SET 0,(HL)	Signal deleting on the row matching the cursor position.
	JR Z,L2F7A	Jump if deleting from the row matching the cursor position.
	RES 0,(HL)	Signal not deleting on the row matching the cursor position.
L2F7A:	CALL L16E0	Insert the character into the end of the edit buffer row, shifting all columns left until the cursor position is reached.
	PUSH AF	A=Character shifted out, and therefore to be potentially shifted into the end of the previous row.
	PUSH BC	B=New column number. C=Row number.
	PUSH DE	DE=Start address of row to delete from.
	LD HL,\$F6F4	Deleting flags.
	BIT 0,(HL)	Deleting from the row matching the cursor position?
	JR NZ,L2F95	Jump ahead if so.

Deleting from a row after the cursor position

LD B,\$00	Column 0.
CALL L2BFA	Is there an editable character on the row?
JR C,L2F95	Jump if there is.

Shifting the characters on this row has resulted in a blank row, so shift all rows below screen up to remove this blank row

CALL L2FA6	Shift up all BASIC line rows below to close the gap.
POP DE	DE=Start address of row to delete from.
POP BC	B=New column number. C=Row number.
JR L2F9A	Jump ahead.

There are characters remaining on the row following the shift so display this to the screen and then continue to shift the remaining rows

L2F95:	POP HL	HL=Start address of the row.
	POP BC	B=New column number. C=Row number.
	CALL L362A	Print the row of the edit buffer to the screen, if required.
L2F9A:	POP AF	A=Character to insert.
	DEC C	Previous row.
	LD B,A	B=Character to insert.
	POP HL	HL=Initial cursor row and column numbers.
	POP AF	Retrieve the flags status (zero flag set if deleting from the row matching the cursor position).
	LD A,B	A=Character to insert.
	JP NZ,L2F58	Jump back if not deleting from the row matching the cursor position, i.e. all rows after the cursor have not yet been shifted.

**[BUG** - The 'line altered' flag is not cleared when an 'edited' null line is entered. To reproduce the bug, insert a couple of BASIC lines, type a character, delete it, and then cursor up or down onto a program line. The line is considered to have been changed and so is processed as if it consists of characters. Further, when cursor down is pressed to move to a BASIC line below, that line is deemed to have changed and hence moving off from it causing that line to be re-inserted into the BASIC program. Credit: Ian Collier (+3), Paul Farrow (128)] [The fix for the bug is to check whether all characters have been deleted from the line and if so to reset the 'line altered' flag. This would require the following code to be inserted at this point. Credit: Paul Farrow. PUSH DE LD HL,\$0020 ADD HL,DE ; Point to the flag byte for this row. POP DE BIT 0,(HL) ; First row of BASIC line in addition to the last? JR Z,SKIP\_CLEAR ; Jump ahead if not. LD B,\$00 CALL \$2E67 (ROM 0) ; Is this a blank row? i.e. Find editable position on this row to the right, returning column number in B. JR C,SKIP\_CLEAR ; Jump if a character exists on the line. LD HL,\$EC0D RES 3,(HL) ; Signal that the current line has not been altered. SKIP\_CLEAR: XOR A ; Set the preferred column to 0.]

SCF	[Redundant since never subsequently checked]
POP BC	Retrieve initial cursor row and column numbers.
RET	

## Shift Rows Up to Close Blank Row in Screen Line Edit Buffer

The cursor is on a blank row but has been moved off of it. Therefore shift all BASIC lines below it up so as to remove the blank row.

Entry: DE=Address of the row in the Screen Line Edit Buffer containing the cursor.

C =Row number in the Screen Line Edit Buffer containing the cursor.

Carry flag set if rows were shifted up, i.e. a row below existed.

L2FA6:	LD HL,\$0020	
	ADD HL,DE	Point to the flag byte for the row.
	LD A,(HL)	
	BIT 0,(HL)	Is the cursor on a blank row (which is flagged as the first row of a BASIC line)?
	JR NZ,L2FD8	Jump ahead if it is. [Could have improved speed by jumping to \$2FDC (ROM 0) since DE already holds the start address of the row]

Cursor not on a blank row but is on its own row at the end of a multi-row BASIC line

PUSH AF	Save the cursor row flag byte.
PUSH BC	Save the cursor row number in C.
LD A,C	Is the cursor on row 0?
OR A	
JR NZ,L2FCA	Jump ahead if it is not, i.e. there is at least one row above.

Cursor on row 0, hence a BASIC line must be off the top of the screen [???? Can this ever be the case?]

PUSH BC	Save the cursor row number.
LD HL,(\$FC9A)	Line number at top of screen.
CALL L3370	Find closest line number (or \$0000 if no line).
LD (\$FC9A),HL	Line number at top of screen.
LD A,(\$F9DB)	Fetch the number of rows of the BASIC line that are in the Above-Screen Line Edit Buffer,
LD C,A	i.e. that are off the top of the screen.
DEC C	Decrement the row count, i.e. one less row off the top of the screen.
CALL L32DD	DE=Address of row in Above-Screen Line Edit Buffer.
POP BC	Retrieve the cursor row number.
JR L2FCE	Jump ahead.

There is a row above so set this as the last row of the BASIC line

L2FCA:	DEC C	Previous row, i.e. the last row of the BASIC line that contains editable characters.
	CALL L30DA	DE=Start address in Screen Line Edit Buffer of the previous row.
L2FCE:	POP BC	Retrieve the cursor row number.
	POP AF	Retrieve the cursor row flag byte, which indicates last row of BASIC line.
	LD HL,\$0020	Point to the flag byte for the previous row.
	ADD HL,DE	
	RES 1,(HL)	Signal that the previous row does not span onto another row.
	OR (HL)	Keep the previous row's first BASIC row flag.
	LD (HL),A	Update the flag byte for the previous row.

Shift up all rows below the old cursor position within the Screen Line Edit Buffer and including the Below-Screen Line Edit Buffer, and update the display file if required

L2FD8:	LD B,C CALL L30DA CALL L3105  JP L1667	B=Row number in the Screen Line Edit Buffer. DE=Start address in Screen Line Edit Buffer of the row specified in C. Shift up rows of the BASIC line in the Below-Screen Line Edit Buffer, or insert the next line BASIC line if buffer empty. Shift Screen Line Edit Buffer rows up from row specified by B and update the display file if required. [Could have saved 3 bytes by replacing the instructions CALL \$3105 (ROM 0) / JP \$1667 (ROM 0) with JP \$1664 (ROM 0)]
--------	--	---

## DELETE-WORD-LEFT Key Handler Routine

This routine deletes to the start of the current word that the cursor is on, or if it is on the first character of a word then it deletes to the start of the previous word. Since the function works by deleting one character at a time, display file updates are disabled whilst the function is executing to prevent screen flicker. If there is no word to delete then an error beep is requested.

Symbol:

← DEL  
←

Exit: Carry flag reset to indicate to produce an error beep and set not to produce an error beep.

L2FE2:	CALL L30AA	Remove cursor attribute, disable display file updates and get current cursor position. Exits with HL pointing to the editing area information.
L2FE5:	PUSH HL CALL L30BB JR Z,L301D CALL L2B81 POP HL JR NC,L301E	Save address of the editing area information. Does a previous character exist in the current Screen Line Edit Buffer row? Jump if at the start of the BASIC line to print all rows. Is previous column position editable? (Returns carry flag set if editable) Retrieve address of the editing area information. Jump if not editable to print all rows.

A previous character exists and is editable

CALL L2A40 PUSH AF PUSH HL CALL L2F38 POP HL POP AF CP \$20 JR Z,L2FE5	Get character from current cursor position. Save current character. Save address of the editing area information. Delete character to the right, shifting subsequent rows as required. Retrieve address of the editing area information. Retrieve current character. Is it a space? Jump back if so to find the end of the last word.
---	--

The end of the word to delete has been found, so enter a loop to search for the start of the word

L2FFF:	PUSH HL CALL L30BB JR Z,L301D CALL L2B81 POP HL JR NC,L301E CALL L2A40 CP \$20 JR Z,L3019	Save address of the editing area information. Does a previous character exist in the current Screen Line Edit Buffer row? Jump if at the start of a BASIC line to print all rows. Is previous column position editable? (Returns carry flag set if editable) Retrieve address of the editing area information. Jump if not editable to print all rows. Get character from current cursor position Is it a space? Jump if so.
--------	---	--

Character is not a space

PUSH HL CALL L2F38 POP HL JR L2FFF	Save address of the editing area information. Delete character to the right, shifting subsequent rows as required. Retrieve address of the editing area information. Jump back to delete next character until start of the word found.
---	---

A space prior to a word has been found

L3019:	PUSH HL	Save address of the editing area information.
--------	---------	---

	CALL L2B9E	Find next Screen Line Edit Buffer editable position to right, moving to next row if necessary.
L301D:	POP HL	Retrieve address of the editing area information.
Print all rows to the screen		
L301E:	LD A,B	Fetch the new end column number.
	PUSH AF	Save the flags status.
	PUSH HL	Save address of the editing area information.
	LD HL,\$EEF5	
	RES 2,(HL)	Re-enable display file updates.
	LD A,(\$EC15)	The number of editing rows on screen. [This will end up being used as the alternate cursor column]
	PUSH BC	Save the row and new column numbers.
	LD B,\$00	B=Print from row 0.
	LD C,A	C=Number of editing rows on screen.
	CP A	Set the zero flag to signal not to change cursor position settings.
	CALL L1624	Print all Screen Line Edit Buffer rows to the display file.
	POP BC	Retrieve the row and new column numbers.
	LD HL,\$EC0D	Editor flags.
	SET 3,(HL)	Indicate current line has been altered.
	POP HL	Retrieve address of the editing area information.

**[BUG]** - The preferred cursor column field gets corrupted with the number of editing rows on screen. Credit: Ian Collier (+3), Andrew Owen (128)] [The bug can be fixed by pre-loading the A register with the current preferred column number. Credit: Paul Farrow.

LD A,(\$F6F0)	Fetch the preferred column position.]
---------------	---------------------------------------

CALL L2A1E	Store editing position and print cursor.
POP AF	Retrieve the flags status.
RET	

## DELETE-WORD-RIGHT Key Handler Routine

This routine deletes to the start of the next word. Since the function works by deleting one character at a time, display file updates are disabled whilst the function is executing to prevent screen flicker.

If there is no word to delete then an error beep is requested.

Symbol:

DEL →

Exit: Carry flag set to indicate not to produce an error beep.

L303D:	CALL L30AA	Remove cursor attribute, disable display file updates and get current cursor position.
		Exits with HL pointing to the editing area information.
L3040:	PUSH HL	Save address of the editing area information.
	CALL L2A40	Get character from current cursor position.
	POP HL	Retrieve address of the editing area information.
	CP \$00	Is it a null character, i.e. end of BASIC line?
	SCF	Signal do not produce an error beep.
	JR Z,L301E	Jump if end of the BASIC line to print all rows.
	PUSH AF	Save the character.
	PUSH HL	Save address of the editing area information.
	CALL L2F38	Delete character to the right, shifting subsequent rows as required.
	POP HL	Retrieve address of the editing area information.
	POP AF	Retrieve the character.
	CP \$20	Was the character a space?
	JR NZ,L3040	Jump back if not to delete the next character until the end of the word is found.
L3055:	CALL L2A40	Get character from current cursor position.
	CP \$20	Is it a space?
	SCF	Signal do not produce an error beep.
	JR NZ,L301E	Jump if not to print all rows.
	PUSH HL	Save address of the editing area information.
	CALL L2F38	Delete character to the right, shifting subsequent rows as required.
	POP HL	Retrieve address of the editing area information.

JR L3055

Jump back to delete all subsequent spaces until the start of the next word or the end of the line is found.

## DELETE-TO-START-OF-LINE Key Handler Routine

Delete to the start of the current BASIC line. Since the function works by deleting one character at a time, display file updates are disabled whilst the function is executing to prevent screen flicker.

An error beep is not produced if there is no characters in the current BASIC line.

Symbol:



Exit: Carry flag set to indicate not to produce an error beep.

L3064:	CALL L30AA	Remove cursor attribute, disable display file updates and get current cursor position. Exits with HL pointing to the editing area information.
L3067:	PUSH HL CALL L30DA LD HL,\$0020 ADD HL,DE BIT 0,(HL) JR NZ,L307F	Save address of the editing area information. DE=Start address in Screen Line Edit Buffer of the row specified in C.  Point to the flag byte for the row. Is it the first row of the BASIC line? Jump if so.

Not in the first row of a BASIC line

CALL L2B81	Is previous column position editable? (Returns carry flag set if editable)
JR NC,L3093	Jump if not editable since nothing to delete.
CALL L2F38	Delete character to the right, shifting subsequent rows as required.
POP HL	Retrieve address of the editing area information.
JR L3067	Jump back to delete next character until first row of the BASIC line is found.
PUSH HL	[Redundant byte]

In the first row of the BASIC line

L307F:	LD A,B CP \$00 JR Z,L3093 DEC B CALL L2A40 INC B CP \$00 JR Z,L3093 DEC B CALL L2F38 JR L307F	Fetch the new end column number. Is it at the start of the row? Jump if so since nothing to delete. Point to previous column. Get character from current cursor position. Point back to the new end column. Is it a null character, i.e. not editable? Jump if so since nothing to delete. Point to previous column. Delete character to the right, shifting subsequent rows as required. Jump back to delete the next character until the start of the BASIC line is found.
L3093:	POP HL	Retrieve address of the editing area information.
L3094:	SCF JP L301E	Signal not to produce error beep. Jump back to print all rows.

## DELETE-TO-END-OF-LINE Key Handler Routine

Delete to the end of the current BASIC line. Since the function works by deleting one character at a time, display file updates are disabled whilst the function is executing to prevent screen flicker.

An error beep is not produced if there is no characters in the current BASIC line.

Symbol:



Exit: Carry flag set to indicate not to produce an error beep.

L3098:	CALL L30AA	Remove cursor attribute, disable display file updates and get current cursor position. Exits with HL pointing to the editing area information.
L309B:	CALL L2A40 CP \$00	Get character from current cursor position. Is it a null character, i.e. at end of BASIC line?

SCF	Signal not to produce an error beep.
JR Z,L3094	Jump if end of BASIC line to print all rows.
PUSH HL	Save address of the editing area information.
CALL L2F38	Delete character to the right, shifting subsequent rows as required.
POP HL	Retrieve address of the editing area information.
JR L309B	Jump back to delete the next character until the end of the BASIC line is found.

## Remove Cursor Attribute and Disable Updating Display File

This routine is called by the DELETE key handler routines. Aside from removing the cursor from the display, it prevents display file updates occurring whilst the delete functions are executing.

Exit: HL=Address of the editing area information.

A=Cursor column number preferred.

B=Cursor column number.

C=Cursor row number.

L30AA:	LD HL,\$EC0D	Editor flags.
	RES 0,(HL)	Signal that the Screen Line Edit Buffer is not full.
	CALL L2A12	Remove cursor, restoring old attribute.
	LD HL,\$EEF5	
	SET 2,(HL)	Indicate not to print edit buffer rows, therefore preventing intermediate screen updates.
	LD HL,\$F6F1	Point to the editing area information.
	RET	

## Previous Character Exists in Screen Line Edit Buffer?

This routine tests the whether a previous character exists in the current BASIC line within the Screen Line Edit Buffer.

Entry: C=Row number.

B=Column number.

Exit : Zero flag set if at start of the BASIC line (first column or leading null).

L30BB:	CALL L30DA	DE=Start address in Screen Line Edit Buffer of the row specified in C.
	LD HL,\$0020	
	ADD HL,DE	HL=Address of the flag byte for this row.
	BIT 0,(HL)	Is this the first row of a BASIC line?
	JR Z,L30D4	Jump if not.

On first row of a BASIC line

	LD A,B	Fetch the column number.
	CP \$00	At the start of the row?
	JR Z,L30D8	Jump ahead if so.
	DEC B	Move to the previous column.
	CALL L2A40	Get current character from Screen Line Edit Buffer.
	INC B	Move back to the original column.
	CP \$00	Does the position contain a null?
	JR Z,L30D8	Jump if not.
L30D4:	LD A,\$01	
	OR A	Reset the zero flag.
	RET	
L30D8:	XOR A	Set the zero flag.
	RET	

## Find Row Address in Screen Line Edit Buffer

Find address in Screen Line Edit Buffer of specified row.

This routine calculates  $DE = \$EC16 + \$0023 * C$ .

Entry: C=Row number.

Exit : DE=Address of edit row.

L30DA:	LD HL,\$EC16	Point to the Screen Line Edit Buffer.
--------	--------------	---------------------------------------

L30DD:	PUSH AF	Save A.
	LD A,C	A=Edit row number.
	LD DE,\$0023	35 bytes per row.
L30E2:	OR A	Row requested found?
	JR Z,L30E9	Jump to exit if so.
	ADD HL,DE	Advance to next row.
	DEC A	
	JR L30E2	Jump to test if requested row found.
L30E9:	EX DE,HL	Transfer address to DE.
	POP AF	Restore A.
	RET	

## Find Position within Screen Line Edit Buffer

Find the address of a specified row and column in the Screen Line Edit Buffer.

The routine calculates  $DE = \$EC16 + \$0023 * C + B$ .

Entry: B=Column number.

C=Row number.

Exit : HL=Address of specified position.

[Not used by the ROM]

L30EC:	PUSH DE	
	CALL L30DA	DE=Start address in Screen Line Edit Buffer of the row specified in C.
	LD H,\$00	
	LD L,B	
	ADD HL,DE	$DE = \$EC16 + \$0023 * C + B$ .
	POP DE	
	RET	

## Below-Screen Line Edit Buffer Settings

This table holds the default values for the Below-Screen Line Edit Buffer settings starting at \$F6F5. It should only contain a table of 3 bytes to tie up with the space allocated within the Editor workspace variables at \$F6F5. As a result, the last 2 bytes will get copied into the Below-Screen Line Edit Buffer itself. It appears that the word at \$F6F6 is supposed to be a pointer to the next available or accessed location within the buffer but this facility is never used. Therefore the table need only be 1 byte long, in which case it would be more efficient for the routine at \$30FC (ROM 0) to simply set the byte at \$F6F5 directly.

L30F6:	DEFB \$05	Number of bytes in table.
	DEFB \$00	\$F6F5 = Number of rows held in the Below-Screen Line Edit Buffer.
	DEFW \$0000	\$F6F6/7. <b>[BUG]</b> - These two bytes should not be here and the table should only contain 3 bytes. Credit: Paul Farrow]
	DEFW \$F6F8	\$F6F8/9 = Points to next location within the Below-Screen Line Edit Buffer.

## Set Below-Screen Line Edit Buffer Settings

Sets the default values for the Below-Screen Line Edit Buffer settings.

Copy 5 bytes from \$30F7-\$30FB (ROM 0) to \$F6F5-\$F6F9.

L30FC:	LD HL,L30F6	Default Below-Screen Line Edit Buffer settings.
	LD DE,\$F6F5	Destination address.
	JP L3F61	Copy bytes.

## Shift Up Rows in Below-Screen Line Edit Buffer

Shifts up all rows in the Below-Screen Line Edit Buffer, or if empty then copies a BASIC line from the program area into the Below-Screen Line Edit Buffer.

Exit: HL=Address of the Below-Screen Line Edit Buffer.

L3105:	PUSH BC	Save BC.
	PUSH DE	Save DE.
	LD HL,\$F6F5	Point to the Below-Screen Line Edit Buffer details.
	PUSH HL	Save it.

## SPECTRUM +2 ROM o DISASSEMBLY

LD A,(HL)  
OR A  
JR NZ,L3127

A=Number of rows held in Below-Screen Line Edit Buffer.  
Are there any rows below screen?  
Jump if so.

There are no rows in the Below-Screen Line Edit Buffer

PUSH HL  
CALL L3385  
  
LD HL,(\$F9D7)  
CALL L3378  
  
JR NC,L311E  
LD (\$F9D7),HL  
L311E: LD B,H  
LD C,L  
POP HL  
CALL L32FC  
  
DEC A  
  
JR L313C

Save the address of the Below-Screen Line Edit Buffer details.  
Copy 'Insert Keyword Representation Into Keyword Construction Buffer' routine into RAM.  
HL=Line number of the BASIC line in the program area being edited.  
Create line number representation in the Keyword Construction Buffer of the next BASIC line.  
Jump if next line does not exist, with HL holding \$0000.  
Store the new line number.  
  
BC=Line number of the next BASIC line, or last BASIC line in the program.  
Retrieve the address of the Below-Screen Line Edit Buffer details.  
Copy the BASIC line into the Below-Screen Line Edit Buffer, or empty the first buffer row if the BASIC line does not exist.  
Decrement the count of the number of rows held in the Below-Screen Line Edit Buffer, i.e. assume the rows have been shifted.  
Jump forward.

There are rows in the Below-Screen Line Edit Buffer so shift all rows up

L3127: LD HL,\$EC0D  
RES 0,(HL)  
LD HL,\$F6F8  
LD D,H  
LD E,L  
LD BC,\$0023  
ADD HL,BC  
LD BC,\$02BC  
LDIR  
DEC A  
  
SCF  
L313C: POP DE  
LD (DE),A  
LD HL,\$F6F8  
POP DE  
POP BC  
RET

Editor flags.  
Signal that the Screen Line Edit Buffer is not full.  
Below-Screen Line Edit Buffer, the temporary copy of line being edited.  
  
Move all rows in the Below-Screen Line Edit Buffer up by one row.  
20 rows.  
  
Decrement the count of the number of rows held in the Below-Screen Line Edit Buffer.  
[Redundant since never subsequently checked]  
DE=Points to number of rows held in the Below-Screen Line Edit Buffer.  
Update the number of rows held in the Below-Screen Line Edit Buffer  
HL=Address of first row in the Below-Screen Line Edit Buffer.  
Restore DE.  
Restore BC.

### Shift Down Rows in Below-Screen Line Edit Buffer

Shifts down all rows in the Below-Screen Line Edit Buffer, or the last Screen Line Edit Buffer row contains a complete BASIC line then it empties the Below-Screen Line Edit Buffer.

Entry: DE=Start address in Screen Line Edit Buffer of the last editing row.

Exit : Carry flag reset to indicate Below-Screen Line Edit Buffer full.

A =Number of rows held in the Below-Screen Line Edit Buffer.

HL=Address of first row in the Below-Screen Line Edit Buffer.

L3144: PUSH BC  
PUSH DE  
LD HL,\$0020  
ADD HL,DE  
LD A,(HL)  
CPL  
AND \$11  
JR NZ,L3165

Save BC.  
DE=Start address in Screen Line Edit Buffer of the last editing row.  
  
Point to the flag byte for the edit buffer row.  
Fetch flag byte.  
Invert bits.  
  
Jump if not the first row of the BASIC line or no associated line number stored.

First row of the BASIC line or an associated line number stored

## SPECTRUM +2 ROM o DISASSEMBLY

	PUSH HL	HL=Points at flag byte of the last Screen Line Edit Buffer row.
	PUSH DE	DE=Address of the last Screen Line Edit Buffer row.
	INC HL	
	LD D,(HL)	
	INC HL	
	LD E,(HL)	DE=Corresponding BASIC line number.
	PUSH DE	Save it.
	CALL L3385	Copy 'Insert Keyword Representation Into Keyword Construction Buffer' routine to RAM.
	POP HL	HL=Corresponding line number for last editing row.
	CALL L3370	Find the closest line number.
	JR NC,L3163	Jump if line does not exist.
	LD (\$F9D7),HL	Store as the line number of the BASIC line being edited.
L3163:	POP DE	DE=Address of the last Screen Line Edit Buffer row.
	POP HL	HL=Points at flag byte of edit buffer row.
L3165:	BIT 0,(HL)	Is it the first row of the BASIC line?
	LD HL,\$F6F5	Point to the Below-Screen Line Edit Buffer details.
	PUSH HL	Save the address of the Below-Screen Line Edit Buffer details.
	JR Z,L3172	Jump if not the first row of the BASIC line.

The first row of the BASIC line, hence after the shift there will not be a row straggling off the bottom of the screen

	LD A,\$00	Signal no rows held in the Below-Screen Line Edit Buffer. [Could have saved 1 byte by using XOR A]
	SCF	Signal Below-Screen Line Edit Buffer is not full.
	JR L313C	Store new flag.

Not the first row the BASIC line

L3172:	LD A,(HL)	Fetch the number of rows held in the Below-Screen Line Edit Buffer.
	CP \$14	Has the bottom of the buffer been reached?
	JR Z,L313C	Jump if so, with the carry flag reset to indicate the buffer is full.

The Below-Screen Line Edit Buffer is not full so copy the last Screen Line Edit Buffer row into the top 'visible' Below-Screen Line Edit Buffer row

	LD BC,\$0023	Length of an edit buffer row.
	LD HL,\$F6F8	Address of the first row in the Below-Screen Line Edit Buffer.
	EX DE,HL	HL=Address of the last row in the Screen Line Edit Buffer, DE=Address of the first row in the Below-Screen Line Edit Buffer.
	LDIR	Copy the last Screen Line Edit Buffer row into the first Below-Screen Line Edit Buffer row, i.e. the 'visible' edit buffer row.

Copy all Below-Screen Line Edit Buffer rows down

	LD HL,\$F9D6	
	LD D,H	
	LD E,L	DE=End of the last row in the Below-Screen Line Edit Buffer.
	LD BC,\$0023	Length of an edit buffer row.
	OR A	
	SBC HL,BC	HL=End of penultimate row in the Below-Screen Line Edit Buffer.
	LD BC,\$02BC	Length of the Below-Screen Line Edit Buffer minus one row.
	LDDR	Shift all the rows down by one.
	INC A	Increment the number of rows held in the Below-Screen Line Edit Buffer.
	SCF	Signal Below-Screen Line Edit Buffer is not full.
	JR L313C	Jump to store the number of rows held in the Below-Screen Line Edit Buffer.

### Insert Character into Below-Screen Line Edit Buffer

Called when a non-action key is pressed and rows of the BASIC line spans into the Below-Screen Line Edit Buffer and therefore require shifting.

Entry: HL=Current row's flag byte.

A=Character code to insert at the start of the first row of the Below-Screen Line Edit Buffer.

L3194:	PUSH BC	Save registers.
--------	---------	-----------------

## SPECTRUM +2 ROM o DISASSEMBLY

PUSH DE	
PUSH AF	Save the character to insert.
LD B,\$00	Column 0.
LD C,\$01	Row 1.
PUSH HL	Save address of the row's flag byte.
CALL L31E9	Find row address specified by C in the Below-Screen Line Edit Buffer, into DE.
POP HL	Retrieve address of the row's flag byte.
BIT 3,(HL)	Is this the end row of the BASIC line?
RES 3,(HL)	Indicate that it is no longer the end row of the BASIC line.
JR NZ,L31C6	Jump if it was the end row of the BASIC line.

The row in the Below-Screen Line Edit Buffer is not the last row of the BASIC line.

Insert the character into the current row. If a spill from this row occurs then insert that character into the start of the following row and shift all existing characters right by one. Repeat this process until all rows have been shifted.

L31A6:	CALL L2E67	Find first editable position on this row from the previous column to the right, returning column number in B.
	POP AF	A=Character to insert.
L31AA:	CALL L16CB	Insert character into the start of the edit buffer row, shifting the row right. Returns carry flag reset.
	JR Z,L31E0	Jump if the byte shifted out of the last column position was \$00, hence no more shifting required.

The end character of the row has spilled out so it must be inserted as the first editable character of the following row

PUSH AF	Stack the character which needs to be inserted into the next row.
LD B,\$00	B=First column in the next row.
INC C	C=Next row.
LD A,C	
CP \$15	Has the bottom row of the Below-Screen Line Edit Buffer been reached, i.e. row 21?
JR C,L31C6	Jump ahead if not.

The bottom row of the Below-Screen Line Edit Buffer has been reached

DEC HL	Point to last character of the current row.
LD A,(HL)	Get the character.
INC HL	Point back to the flag byte of this row.
CP \$00	Is the character a null character? [Could have saved 1 byte by using AND A]
JR Z,L31C6	Jump ahead if it is.

The Below-Screen Line Edit Buffer is completely full

PUSH HL	Save address of the flag byte.
LD HL,\$EC0D	Editor flags.
SET 0,(HL)	Signal that the Screen Line Edit Buffer (including Below-Screen Line Edit Buffer) is full.
POP HL	HL=Address of the flag byte.

Check whether there is another row to shift

L31C6:	BIT 1,(HL)	Does the row span onto another row?
	SET 1,(HL)	Signal that the row spans onto another row.
	RES 3,(HL)	Signal not the last row of the BASIC line.
	CALL L31E9	Find the address of the row specified by C in Below-Screen Line Edit Buffer, into DE.
	JR NZ,L31A6	Jump back if spans onto another row to shift it also.

All existing rows have now been shifted but a new row needs to be inserted

PUSH BC	B=Column number. C=Row number.
PUSH DE	DE=Start address of the row in the edit buffer.
CALL L360C	Null all column positions in the edit buffer row.
LD (HL),\$08	Set the flag byte for the row to indicate it is the last row of the BASIC line.
POP DE	DE=Start address of the row in the edit buffer.
POP BC	B=Column number. C=Row number.

CALL L361A  
POP AF  
JR L31AA

Indent the row by setting the appropriate number of null characters.  
Get character to insert.  
Jump back to insert it.

The shifting of all rows has completed

L31E0: LD A,C  
LD (\$F6F5),A  
SET 3,(HL)  
POP DE  
POP BC  
RET

Get the row number.  
Store as the number of rows held within the Below-Screen Line Edit Buffer.  
Mark this row as the last row of the BASIC line.  
Restore registers.

## Find Row Address in Below-Screen Line Edit Buffer

Find address in the Below-Screen Line Edit Buffer of specified row.

This routine calculates  $DE = \$F6F8 + \$0023 * C$ .

Entry: C=Row number.

Exit : Address of edit row in DE.

L31E9: LD HL,\$F6F8  
JP L30DD

Address of the Below-Screen Line Edit Buffer.  
Jump to find the row address and return.

## Delete a Character from a BASIC Line in the Below-Screen Line Edit Buffer

Delete a character at the specified position, shifting subsequent characters left as applicable.

Exit: A=Character shifted out of the top row of the Below-Screen Line Edit Buffer.

L31EF: PUSH BC  
PUSH DE  
LD HL,\$EC0D  
RES 0,(HL)  
  
LD A,(\$F6F5)  
LD C,A  
OR A  
LD A,\$00  
JR Z,L3241

Save registers.  
  
Editor flags.  
Signal that the Screen Line Edit Buffer (including Below-Screen Line Edit Buffer) is not full.  
A=Number of rows held in the Below-Screen Line Edit Buffer.  
C=Number of rows held in the Below-Screen Line Edit Buffer.  
Are there any rows in the Below-Screen Line Edit Buffer?  
A null character.  
Jump if there are no rows. [Redundant check since this routine should never be called if there are no rows in this buffer]

There is at least one row in the Below-Screen Line Edit Buffer

L31FF: CALL L31E9  
PUSH AF  
LD B,\$00  
CALL L2E67  
JR NC,L3218

Find the address of the last used row within Below-Screen Line Edit Buffer, into DE.  
Save the character to insert.  
Start searching from column 0.  
Find editable position on this row to the right, returning column number in B.  
Jump if no editable position found, i.e. a blank row.

The row is not blank

POP AF

A=Character to insert.

DE=Address within a row of edit buffer.

A=Character to shift into right of row.

B=The column to start shifting at.

CALL L16E0  
  
PUSH AF  
PUSH BC  
LD B,\$00

Insert the character into the end of the edit buffer row, shifting all columns left until the cursor position is reached.  
A=Character shifted out, zero flag set if the shifted out character was a null (\$00).  
Save the row number.  
Start searching from column 0.

## SPECTRUM +2 ROM 0 DISASSEMBLY

CALL L2E67	Is this now a blank row? i.e. Find editable position on this row to the right, returning column number in B.
POP BC	C=Row number.
JR C,L323C	Jump if editable position found.

The row is already blank or the result of the shift has caused it to become blank.  
HL points to the last blank character in the row.

L3218:	INC HL	Point to the flag byte for the blank row.
	LD A,(HL)	Fetch the flag byte.
	PUSH AF	Save the flag byte for the blank row.
	PUSH BC	Save the row number.
	LD A,C	Fetch the row number of this blank row.
	CP \$01	Is this the first row in the Below-Screen Line Edit Buffer?
	JR NZ,L322A	Jump if not.

The first row in the Below-Screen Line Edit Buffer is empty and hence the BASIC line now fits completely on screen, i.e. within the Screen Line Edit Buffer

LD A,(\$EC15)	The number of editing rows on screen.
LD C,A	C=Bottom row number in the Screen Line Edit Buffer.
CALL L30DA	DE=Start address in Screen Line Edit Buffer of the bottom row, as specified in C.
JR L322E	Jump ahead to continue.

The blank row is not the first row in the Below-Screen Line Edit Buffer, and hence there are further rows above to be shifted

L322A:	DEC C	Previous row within the Below-Screen Line Edit Buffer.
	CALL L31E9	Find the address of the row specified by C in Below-Screen Line Edit Buffer, into DE.
L322E:	POP BC	Retrieve the row number.
	POP AF	A=Flag byte value for the blank row.
	LD HL,\$0020	
	ADD HL,DE	Point to the flag byte for the row above.
	RES 1,(HL)	Signal that the row above does not span onto another row.
	OR (HL)	Or in the flag bits from the blank row, essentially this will retain the 'last row' bit.
	LD (HL),A	Update the flag byte for the row above.
	LD HL,\$F6F5	Point to the number of rows held in the Below-Screen Line Edit Buffer.
	DEC (HL)	Decrement the row count.

Continue with the next row

L323C:	POP AF	Fetch the character shifted out from the current row, ready for insertion into the row above.
	DEC C	Previous row.
	JR NZ,L31FF	Jump back if the character shifted out was not null, i.e. more rows above to shift.

All rows in the Below-Screen Line Edit Buffer have been shifted

L3241:	SCF	[Redundant since never subsequently checked]
	POP DE	Restore registers.
	POP BC	
	RET	

## Above-Screen Line Edit Buffer Settings

This table holds the default values for the Below-Screen Line Edit Buffer settings starting at \$F9DB.  
It appears that the word at \$F9DC is supposed to be a pointer to the next available or accessed location within the buffer but this facility is never used.  
Therefore the table need only be 1 byte long, in which case it would be more efficient for the routine at \$3248 (ROM 0) to simply set the byte at \$F9DB directly.

L3244:	DEFB \$03	Number of bytes in table.
	DEFB \$00	\$F9DB = Number of rows held in the Above-Screen Line Edit Buffer.
	DEFW \$F9DE	\$F9DC/D = Points to next available location within the Above-Screen Line Edit Buffer.

## Set Above-Screen Line Edit Buffer Settings

Sets the default values for the Above-Screen Line Edit Buffer settings.

Copy 3 bytes from \$3245-\$3247 (ROM 0) to \$F9DB-\$F9DD.

L3248:	LD HL,L3244 LD DE,\$F9DB JP L3F61	Default Above-Screen Line Edit Buffer settings. Destination address. Copy bytes.
--------	---	--

## Shift Rows Down in the Above-Screen Line Edit Buffer

If Above-Screen Line Edit Buffer contains row then decrement the count, i.e. less rows off screen.

If the Above-Screen Line Edit Buffer is empty then load in the new BASIC line at the top of the screen.

Exit : HL=Address of next row to use within the Above-Screen Line Edit Buffer.

Carry flag reset if Above-Screen Line Edit Buffer is empty, i.e. no edit buffer rows were shifted.

L3251:	PUSH BC PUSH DE LD HL,\$F9DB PUSH HL LD A,(HL) OR A JR NZ,L3279	Save registers.  Point to the Above-Screen Line Edit Buffer settings. Save address of the Above-Screen Line Edit Buffer settings. Fetch number of rows of the BASIC line that are off the top of the screen. Are there any rows off the top of the screen? Jump if there are.
--------	---	---

There are no rows of the BASIC line off the top of the screen so use the top line that is visible on screen

L326A:	PUSH HL CALL L3385  LD HL,(\$FC9A) CALL L3370 JR NC,L326A LD (\$FC9A),HL LD B,H LD C,L POP HL INC HL INC HL INC HL JR NC,L3283	Save address of the Above-Screen Line Edit Buffer settings. Copy 'Insert Keyword Representation Into Keyword Construction Buffer' routine to RAM. HL=New line number at top of screen. Verify the line number exists, or fetch the next line number if not. Jump if the line does not exist. Store the line number found as the one at the top of the screen.  BC=New line number at top of screen. HL=Address of the Above-Screen Line Edit Buffer settings.   Point to the first row of the Above-Screen Line Edit Buffer. Jump if the line did not exist.
--------	---	--

The line specified as the one at the top of the screen does exist **[BUG]** - HL points to the start of the first row of the Above-Screen Line Edit Buffer but it should point to the settings fields 3 bytes earlier since the call to \$32FC (ROM 0) will advance HL by 3 bytes. The bug manifests itself when modifying a BASIC line that spans off the top of the screen. It causes corruption to the line number, causing a new BASIC line to be inserted rather than updating the line being edited. When editing lines with a high line number, the corrupted line number can end up larger 9999 and hence the line is deemed invalid when Enter is pressed to insert the line into the BASIC program. The effects of the bug are often masked by the bug at \$2DC7 (ROM 0) which performs LD HL,(\$F9DB) instead of LD HL,\$F9DB and thereby fails to detect when the end of the Above-Screen Line Edit Buffer has been reached. The bug can be fixed by inserted three DEC HL instructions before the call to \$32FC (ROM 0). Credit: Paul Farrow]

CALL L32FC DEC A  EX DE,HL JR L3283	Copy the new BASIC line into the Above-Screen Line Edit Buffer. Decrement the count of the number of rows held in the Above-Screen Line Edit Buffer. HL=Start of the next row in the Above-Screen Line Edit Buffer. Jump ahead to continue.
---	--

There are rows of the BASIC line off the top of the screen

L3279:	LD HL,(\$F9DC) LD BC,\$0023 SBC HL,BC SCF DEC A	HL=Address of the next location within the Above-Screen Line Edit Buffer to use.  Point to the previous row location within the Above-Screen Line Edit Buffer. Signal to update the number of rows held in the Above-Screen Line Edit Buffer. Decrement the count of the number of rows held in the Above-Screen Line Edit Buffer.
--------	---	--

A=New number of rows held in the Above-Screen Line Edit Buffer.

HL=Address of a next row to use within the Above-Screen Line Edit Buffer.

Carry flag reset if no need to update the count of the number of rows in the Above-Screen Line Edit Buffer.

L3283:	EX DE,HL POP HL JR NC,L3288	DE=Address of next row to use within the Above-Screen Line Edit Buffer. HL=Address of the Above-Screen Line Edit Buffer settings. Jump if no need to update the count of the number of rows in the Above-Screen Line Edit Buffer.
L3288:	LD (HL),A INC HL LD (HL),E INC HL LD (HL),D EX DE,HL POP DE POP BC RET	Store the number of rows held in the Above-Screen Line Edit Buffer.  Store the address of the next row to use within the Above-Screen Line Edit Buffer. HL=Address of next row to use within the Above-Screen Line Edit Buffer. Restore registers.

## Shift Row Up into the Above-Screen Line Edit Buffer if Required

This routine is used to shift up a Screen Line Edit Buffer or a Below-Screen Line Edit Buffer row into the Above-Screen Line Edit Buffer.

If shifting the top row of the Screen Line Edit Buffer would result in a straggle into the Above-Screen Line Edit Buffer then the top row is shifted into the next available location within the Above-Screen Line Edit Buffer. If the shift would place the start of a BASIC line on the top row then the Above-Screen Line Edit Buffer is set as empty.

The routine is also called when relisting the BASIC program. The first BASIC line may straggle above the screen and so it is necessary to load the BASIC line into the Above-Screen Line Edit Buffer. This is achieved by using the Below-Screen Line Edit Buffer as a temporary line workspace. This routine is called to shift each row into the Above-Screen Line Edit Buffer as appropriate.

Entry: DE=Start address of the first row in the Screen Line Edit Buffer, or start address of a Below-Screen Line Edit Buffer row.

Exit : HL=Address of next row to use within the Below-Screen or Screen Line Edit Buffer.

Carry flag set if the Line Edit Buffer if not full.

L3290:	PUSH BC PUSH DE LD HL,\$0020 ADD HL,DE LD A,(HL) CPL AND \$11 JR NZ,L32A8	Save registers.   Point to the flag byte for this row within the Below-Screen or Screen Line Edit Buffer. Fetch the flag byte.  Jump if not the first row of the BASIC line or no associated line number stored.
--------	--	--

First row of the BASIC line and associated line number stored

L32A8:	PUSH DE PUSH HL INC HL LD D,(HL) INC HL LD E,(HL) LD (\$FC9A),DE POP HL  POP DE BIT 3,(HL) LD HL,\$F9DB PUSH HL JR Z,L32C6	DE=Start address of the row. HL=Address of the flag byte for the row in the Line Edit Buffer.   DE=Line number of the corresponding BASIC line. Store this as the line number that is at the top of the screen. HL=Address of the flag byte for the row in the Below-Screen or Screen Line Edit Buffer. DE=Start address of the row. Is this the last row of the BASIC line? Point to the Above-Screen Line Edit Buffer settings. Stack the address of the Above-Screen Line Edit Buffer settings. Jump if not the last row of the BASIC line.
--------	---	---

The last row of the BASIC line

PUSH HL CALL L3385  LD HL,(\$FC9A)	Stack the address of the Above-Screen Line Edit Buffer settings. Copy 'Insert Keyword Representation Into Keyword Construction Buffer' routine to RAM. Line number at top of screen.
---	--

## SPECTRUM +2 ROM o DISASSEMBLY

CALL L3378	Create line number representation in the Keyword Construction Buffer of the next BASIC line.
LD (\$FC9A),HL	Update the line number at top of screen.
POP HL	HL=Address of the Above-Screen Line Edit Buffer settings.
INC HL	
INC HL	
INC HL	Point to the start of the Above-Screen Line Edit Buffer.
LD A,\$00	No rows held in the Above-Screen Line Edit Buffer. [Could have saved 1 byte by using XOR A]
SCF	Signal to update the number of rows count.
JR L3283	Jump back to store the new Above-Screen Line Edit Buffer settings.

Not the last row of the BASIC line

L32C6:	LD A,(HL)	Fetch the number of rows held in the Above-Screen or Screen Line Edit Buffer.
	CP \$14	Are there 20 rows, i.e. the buffer is full?
	JR Z,L32D9	Jump if the buffer is full, with the carry flag reset.

Shift the top row of the Screen Line Edit Buffer into the Above-Screen Line Edit Buffer

INC A	Increment the count of the number of rows in the Above-Screen Line Edit Buffer.
LD HL,(\$F9DC)	Fetch the address of the next row to use within the Above-Screen Line Edit Buffer.
LD BC,\$0023	The length of one row in the edit buffer, including the 3 data bytes.
EX DE,HL	DE=Address of next location within the Above-Screen Line Edit Buffer, HL=Address of the row in the Below-Screen or Screen Line Edit Buffer to store.
LDIR	Copy the row of the BASIC line into the Above-Screen Line Edit Buffer.
EX DE,HL	HL=Address of next row to use within the Above-Screen Line Edit Buffer.
SCF	Signal to update the count of the number of rows.
JR L3283	Jump back to store the new Above-Screen Line Edit Buffer settings.

Above-Screen Line Edit Buffer is full

L32D9:	POP HL	HL=Address of the Above-Screen Line Edit Buffer settings.
	POP DE	Restore registers.
	POP BC	
	RET	

## Find Row Address in Above-Screen Line Edit Buffer

Find the address in the Above-Screen Line Edit Buffer of the specified row.

This routine calculates  $DE = \$F9DE + \$0023 * C$ .

Entry: C=Row number.

Exit : DE=Address of edit row.

L32DD:	LD HL,\$F9DE	Point to the start of the Above-Screen Line Edit Buffer.
	JP L30DD	Find the row address.

## BASIC Line Character Action Handler Jump Table

L32E3:	DEFB \$08	Number of table entries.
	DEFB \$0D	Code: Enter.
	DEFW L35F2	Address of the 'Enter' action handler routine.
	DEFB \$01	Code: NULL.
	DEFW L3600	Null remaining columns of an edit buffer row.
	DEFB \$12	Code: FLASH.
	DEFW L3380	Fetch next de-tokenized character from the BASIC line within the program area.
	DEFB \$13	Code: BRIGHT.
	DEFW L3380	Fetch next de-tokenized character from the BASIC line within the program area.
	DEFB \$14	Code: INVERSE.
	DEFW L3380	Fetch next de-tokenized character from the BASIC line within the program area.
	DEFB \$15	Code: OVER.
	DEFW L3380	Fetch next de-tokenized character from the BASIC line within the program area.

DEFB \$10  
 DEFW L3380  
 DEFB \$11  
 DEFW L3380

Code: INK.  
 Fetch next de-tokenized character from the BASIC line within the program area.  
 Code: PAPER.  
 Fetch next de-tokenized character from the BASIC line within the program area.

## Copy a BASIC Line into the Above-Screen or Below-Screen Line Edit Buffer

Copy a BASIC line into the Above-Screen or Below-Screen Line Edit Buffer, handling indentation.

Entry: HL=Address of the previous row's flag byte in Above-Screen or Below-Screen Line Edit Buffer.

BC=Line number corresponding to the row being edited.

Exit : A=Number of rows in the Above-Screen Line Edit Buffer.

HL=Address of the first row of the BASIC line being edited in the Above-Screen Line Edit Buffer.

DE=Address of the last row of the BASIC line being edited in the Above-Screen Line Edit Buffer.

L32FC:	LD D,H	HL=Address of the previous row's flag byte in the Above-Screen/Below-Screen Line Edit Buffer.
	LD E,L	DE=Address of the previous row's flag byte in the Above-Screen/Below-Screen Line Edit Buffer.
	INC DE	
	INC DE	
	INC DE	Advance to the start of the row in the edit buffer.
	PUSH DE	DE=Address of the start of the BASIC line in the Above-Screen/Below-Screen Line Edit Buffer.
	LD HL,\$0020	
	ADD HL,DE	Point to the flag byte for the row.
	LD (HL),\$01	Signal the first row of the BASIC line.
	INC HL	
	LD (HL),B	
	INC HL	
	LD (HL),C	Store the corresponding BASIC line number.
	LD C,\$01	Row 1.
	LD B,\$00	Column 0.

Enter a loop to process each character from the current BASIC line

L3310:	PUSH BC	Save the column and row numbers.
	PUSH DE	Save the Above-Screen/Below-Screen Line Edit Buffer address.
	LD A,(\$EC0E)	Fetch mode.
	CP \$04	Calculator mode?
	CALL NZ,L353D	If not then fetch the next de-tokenized character from the BASIC line within the program area.
	POP DE	Retrieve the Above-Screen/Below-Screen Line Edit Buffer address.
	POP BC	Retrieve the column and row numbers.
	JR C,L332D	Jump if Editor mode and a character was available (if calculator mode then carry flag was reset by test above).

Calculator mode, or Editor mode and a character was not available

LD A,C	A=Row number.
CP \$01	Is it row 1?
LD A,\$0D	A='Enter' character.
JR NZ,L332D	Jump if not.

Row 1

	LD A,B	A=Column number.
	OR A	Is it column 0?
	LD A,\$01	A='Null' character, the code used to indicate to null edit positions.
	JR Z,L332D	Jump if so.
	LD A,\$0D	A='Enter' character.
L332D:	LD HL,L32E3	The action handler table.
	CALL L3F75	Call the action handler routine to process the character.
	JR C,L3352	Jump if no more characters are available.
	JR Z,L3310	Jump back if an action handler was found so as to process the next character.

## SPECTRUM +2 ROM o DISASSEMBLY

A character was available but there was no action handler routine to process it

PUSH AF LD A,\$1F CP B JR NC,L334C	A=Character.  Exceeded column 31? Jump ahead if not.
---	---

Exceeded last column

LD A,\$12  CALL L3357 JR C,L3349	New flag byte value indicating the row spans onto another row and there is an associated line number. Mark this row as spanning onto the next and clear the following row's flags. Jump ahead if not at bottom of the line edit buffer.
---	---

At the bottom of the edit buffer so process the line as if an 'Enter' character had been encountered

POP AF LD A,\$0D JR L332D	Discard the stacked item. A='Enter' character. Jump back to process the 'Enter' code.
---------------------------------	---

The edit buffer has room for another character

L3349: CALL L361A	Indent the row by setting the appropriate number of null characters in the current Above-Screen Line Edit Buffer row.
L334C: POP AF CALL L35EB JR L3310	A=Character. Store the character in the current row/column in the Above-Screen Line Edit Buffer. Jump back to handle the next character.

No more characters are available

L3352: POP HL LD A,C RET Z	HL=Address of the BASIC line being edited in the Above-Screen Line Edit Buffer. A=Number of rows in the Above-Screen Line Edit Buffer. [Redundant since carry flag is always set by here, and zero flag never subsequently checked]
SCF RET	[Redundant since never subsequently checked]

### Set 'Continuation' Row in Line Edit Buffer

This routine is used when the insertion of a BASIC line needs to span onto a another row.

It marks the current row as 'not the last row of the BASIC line' and clears the following row's flags

Entry: DE=Address of start of line edit buffer row.  
B=Column number (will be \$20).  
C=Row number.  
A=New flag byte value (will be \$12).

Exit : Carry flag reset if bottom of line edit buffer reached.  
HL=Address of the flag byte for the new row.

L3357: PUSH AF CALL L360C POP AF XOR (HL) LD (HL),A LD A,C CP \$14 RET NC INC C LD HL,\$0023 ADD HL,DE EX DE,HL LD HL,\$0020 ADD HL,DE LD (HL),\$00	Save the new flag byte value. HL=Address of flag byte for the row. Retrieve the new flag byte value. Toggle to set 'associated line number' and 'row spans onto another row' flags. Store the new flag byte value. A=Row number. At bottom of line edit buffer? Return if so. Advance the row number.  Point to the start of the next row.   Point to the flag byte for the next row. Clear the flags to indicate no BASIC line on this row.
---	--

SCF  
RET

Signal still on a row within the edit buffer.

## BASIC Line Handling Routines

### Find Address of BASIC Line with Specified Line Number

This routine finds the address of the BASIC line in the program area with the specified line number, or the next line is the specified one does not exist.

Entry: HL=Line number.

Exit : Carry flag set if line exists.

DE=Points to the command of the BASIC line within the program area.

HL=Line number (\$0000 for no line number).

L3370:	CALL L34DC  RET C LD HL,\$0000 RET	Find the address of the BASIC line in the program area with the specified line number.  Return if the line exists. No line number.
--------	--	---

### Create Next Line Number Representation in Keyword Construction Buffer

This routine is used to create a string representation of the line number for the next line after the specified line, and store it in the Keyword Construction Buffer.

Entry: HL=Line number.

A=Print leading space flag (\$00=Print leading space).

Exit : Carry flag set to indicate specified line exists.

DE=Points to the command field of the BASIC line.

HL=Line number, or \$0000 if line does not exist.

L3378:	CALL L3456 RET C LD HL,\$0000 RET	Create next line number representation in the Keyword Construction Buffer. Return if line exists. Line not found.
--------	--	---

### Fetch Next De-tokenized Character from Selected BASIC Line in Program Area

Exit: Carry flag reset if a character was available.

A=Character fetched.

L3380:	CALL L353D CCF RET NC	Fetch the next de-tokenized character from the BASIC line within the program area.  Return if a character was available. <b>[BUG - This should just be a RET. Its effect is harmless since the routine below has previously been called and hence simply overwrites the data already copied to RAM. Credit: Ian Collier (+3), Andrew Owen (128)]</b>
--------	-----------------------------	--

### Copy 'Insert Keyword Representation into Keyword Construction Buffer' Routine into RAM

Copies Insert Keyword Representation Into Keyword Construction Buffer routine into physical RAM bank 7, and resets pointers to indicate that there is no BASIC line currently being de-tokenized.

L3385:	LD HL,\$0000 LD (\$FC9F),HL LD (\$FCA1),HL LD HL,L339A LD DE,\$FCAE  LD BC,\$00BC LDIR	Signal no line number of command. Signal no further character to fetch from the BASIC line within the program area. Signal no further character to fetch from the Keyword Construction Buffer. Source for Insert Keyword Representation Into Keyword Construction Buffer routine. Destination for Insert Keyword Representation Into Keyword Construction Buffer routine.  Copy the routine to RAM bank 7 at address \$FCAE.
--------	---	--

RET

**Insert Keyword Representation into Keyword Construction Buffer « RAM Routine »**

This routine copies a keyword string from ROM 1 into the Keyword Construction Buffer, terminating it with an 'end of BASIC line' marker (code ' ' + \$80). Only standard Spectrum keywords are handled by this routine (SPECTRUM and PLAY are processed elsewhere).

The routine is run from RAM bank 7 at \$FCAE so that access to both ROMs is available.

Depending on the value of A (which should be the ASCII code less \$A5, e.g. 'RND', the first (48K) keyword, has A=0), a different index into the token table is taken. This is to allow speedier lookup since there are never more than 15 keywords to advance through.

Entry: A=Keyword character code-\$A5 (range \$00-\$5A).

DE=Insertion address within Keyword Construction Buffer.

Copied to physical RAM bank 7 at \$FCAE-\$FCFC by subroutine at \$3385 (ROM 0).

L339A:	DI	Disable interrupts whilst paging.
	LD BC,\$7FFD	
	LD D,\$17	Page in ROM 1, SCREEN 0, no locking, RAM bank 7.
	OUT (C),D	
	CP \$50	Was the token \$F5 or above?
	JR NC,L33D7	
	CP \$40	Was the token \$E5 or above?
	JR NC,L33D0	
	CP \$30	Was the token \$D5 or above?
	JR NC,L33C9	
	CP \$20	Was the token \$C5 or above?
	JR NC,L33C2	
	CP \$10	Was the token \$B5 or above?
	JR NC,L33BB	

Used for token range \$A5-\$B4 (\$00 <= A <= \$0F)

LD HL,TOKENS+\$0001	\$0096. Token table entry "RND" in ROM 1.
JR L33DC	

Used for token range \$B5-\$C4 (\$10 <= A <= \$1F)

L33BB:	SUB \$10	
	LD HL,TOKENS+\$003A	\$00CF. Token table entry "ASN" in ROM 1.
	JR L33DC	

Used for token range \$C5-\$D4 (\$20 <= A <= \$2F)

L33C2:	SUB \$20	
	LD HL,TOKENS+\$006B	\$0100. Token table entry "OR" in ROM 1.
	JR L33DC	

Used for token range \$D5-\$E4 (\$30 <= A <= \$3F)

L33C9:	SUB \$30	
	LD HL,TOKENS+\$00A9	\$013E. Token table entry "MERGE" in ROM 1.
	JR L33DC	

Used for token range \$E5-\$F4 (\$40 <= A <= \$4F)

L33D0:	SUB \$40	
	LD HL,TOKENS+\$00F6	\$018B. Token table entry "RESTORE" in ROM 1.
	JR L33DC	

Used for token range \$F5-\$FF (A >= \$50)

L33D7:	SUB \$50	
	LD HL,TOKENS+\$013F	\$01D4. Token table entry "PRINT" in ROM 1.
L33DC:	LD B,A	Take a copy of the index value.
	OR A	If A=0 then already have the entry address.

L33DE:	JR Z,L33E9	If indexed item found then jump ahead to copy the characters of the token.
L33E0:	LD A,(HL)	Fetch a character.
	INC HL	Point to next character.
	AND \$80	Has end of token marker been found?
	JR Z,L33E0	Loop back for next character if not.
	DEC B	Count down the index of the required token.
	JR L33DE	Jump back to test whether the required token has been reached.

## Copy Keyword Characters « RAM Routine »

This routine copies a keyword string from ROM 1 into the Keyword Construction Buffer, terminating it with an 'end of BASIC line' marker (code ' '+\$80). A leading space will be inserted if required and a trailing space is always inserted.

The routine is run from physical RAM bank 7 so that access to both ROMs is available.

Entry: HL=Address of keyword string in ROM 1.

DE=Insertion address within Keyword Construction Buffer.

Copied to physical RAM bank 7 at \$FCFD-\$FD2D by subroutine at \$3385 (ROM 0).

L33E9:	LD DE,\$FCA3	DE=Keyword Construction Buffer.
	LD (\$FCA1),DE	Store the start address of the constructed keyword.
	LD A,(\$FC9E)	Print a leading space?
	OR A	
	LD A,\$00	
	LD (\$FC9E),A	Signal leading space not required.
	JR NZ,L33FF	Jump if leading space not required.
	LD A,\$20	Print a leading space.
	LD (DE),A	Insert a leading space.
	INC DE	Advance to next buffer position.
L33FF:	LD A,(HL)	Fetch a character of the keyword.
	LD B,A	Store it.
	INC HL	Advance to next keyword character.
	LD (DE),A	Store the keyword character in the BASIC line buffer.
	INC DE	Advance to the next buffer position.
	AND \$80	Test if the end of the keyword string.
	JR Z,L33FF	Jump back if not to repeat for all characters of the keyword.
	LD A,B	Get keyword character back.
	AND \$7F	Mask off bit 7 which indicates the end of string marker.
	DEC DE	Point back at the last character of the keyword copied into the buffer
	LD (DE),A	and store it.
	INC DE	Advance to the position in the buffer after the last character of the keyword.
	LD A,' '+\$80	\$A0. Space + end marker.
	LD (DE),A	Store an 'end of BASIC line so far' marker.
	LD A,\$07	
	LD BC,\$7FFD	
	OUT (C),A	Page in ROM 0, SCREEN 0, no locking, RAM bank 7.
	EI	Re-enable interrupts.
	RET	

## Identify Token from Table

This routine identifies the string within the Keyword Conversion Buffer and returns the character code. The last character of the string to identify has bit 7 set.

Only 48K mode tokens are identified.

Exit: Carry flag set if token identified.

A=Character code.

Copied to RAM at \$FD2E-\$FD69 by routine at \$3385 (ROM 0).

L341A:	DI	Disable interrupts whilst paging.
	LD BC,\$7FFD	
	LD D,\$17	Select ROM 1, SCREEN 0, RAM bank 7.
	OUT (C),D	
	LD HL,TOKENS+1	\$0096. Address of token table in ROM 1.
	LD B,\$A5	Character code of the first token - 'RND'.

Entry point here used to match 128K mode tokens and mis-spelled tokens

## SPECTRUM +2 ROM o DISASSEMBLY

L3427:	LD DE,\$FD74	Keyword Conversion Buffer holds the text to match against.
L342A:	LD A,(DE)	Fetch a character from the buffer.
	AND \$7F	Mask off terminator bit.
	CP \$61	Is it lowercase?
	LD A,(DE)	Fetch the character again from the buffer.
	JR C,L3434	Jump if uppercase.
	AND \$DF	Make the character uppercase.
L3434:	CP (HL)	Does the character match the current item in the token table?
	JR NZ,L3440	Jump if it does not.
	INC HL	Point to the next character in the buffer.
	INC DE	Point to the next character in the token table.
	AND \$80	Has the terminator been reached?
	JR Z,L342A	Jump back if not to test the next character in the token.

A match was found

	SCF	Signal a match was found.
	JR L344C	Jump ahead to continue.
L3440:	INC B	The next character code to test against.
	JR Z,L344B	Jump if all character codes tested.

The token does not match so skip to the next entry in the token table

L3443:	LD A,(HL)	Fetch the character from the token table.
	AND \$80	Has the end terminator been found?
	INC HL	Point to the next character.
	JR Z,L3443	Jump back if no terminator found.
	JR L3427	Jump back to test against the next token.

All character codes tested and no match found

L344B:	OR A	Clear the carry flag to indicate no match found.
--------	------	--

The common exit point

L344C:	LD A,B	Fetch the character code of the matching token (\$00 for no match).
	LD D,\$07	Select ROM 0, SCREEN 0, RAM bank 7.
	LD BC,\$7FFD	
	OUT (C),D	
	EI	Re-enable interrupts.
	RET	« Last byte copied to RAM »

## Create Next Line Number Representation in Keyword Construction Buffer

This routine is used to create a string representation of the line number for the next line after the specified line, and store it in the Keyword Construction Buffer.

Entry: HL=Line number.  
A=Print leading space flag (\$00=Print leading space).  
Exit : Carry flag set to indicate specified line available.  
DE=Points to the command field of the BASIC line.  
HL=Line number.

L3456:	CALL L3510	Clear BASIC line construction pointers (address of next character in the Keyword Construction Buffer and the address of the next character in the BASIC line within the program area being de-tokenized).
	OR A	<b>[BUG]</b> - Supposed to be XOR A to ensure that a leading space is shown before a command keyword is printed. However, most of the time the A register will enter the routine holding \$00 and so the bug is probably harmless. Credit: Paul Farrow
	LD (\$FC9E),A	Print a leading space flag.
	CALL L1F3F	Use Normal RAM Configuration (physical RAM bank 0).
	CALL L351C	Find address of the specified BASIC line, into HL.
	JR NC,L34B7	Jump if suitable line number not found, i.e. end of program reached.
	JR NZ,L3473	Jump if line number did not match, i.e. is higher than the line requested.

## SPECTRUM +2 ROM o DISASSEMBLY

The line number requested exists

```
LD A,B
OR C
JR Z,L3473
```

BC=Line number.

Jump if the first program line requested (line number of 0).

Fetch the next line

```
CALL L34F5
CALL L34FF
JR NC,L34B7
```

Move to the start of the next BASIC line.

Check whether at the end of the BASIC program.

Jump if at the end of the BASIC program.

Insert line number into the BASIC Line Construction Buffer

```
L3473: LD D,(HL)
      INC HL
      LD E,(HL)
      CALL L1F64
      PUSH DE
      PUSH HL
      PUSH IX
      LD IX,$FCA3
      LD ($FCA1),IX
      EX DE,HL
      LD B,$00
      LD DE,$FC18
      CALL L34BB
      LD DE,$FF9C
      CALL L34BB
      LD DE,$FFF6
      CALL L34BB
      LD DE,$FFFF
      CALL L34BB
```

HL=Address of the BASIC line.

DE=Line number.

Use Workspace RAM configuration (physical RAM bank 7).

Save the line number.

Save the address of the BASIC line+1.

Save IX.

IX=Keyword Construction Buffer, the location where the line number will be created.

Store the start of the buffer as the next location to store a character in.

HL=Line number.

Signal no digit printed yet.

-1000.

Insert the thousand digit.

-100.

Insert the hundred digit.

-10.

Insert the ten digit.

-1.

Insert the units digits. [Note that this is not designed to handle line number 0, which technically is not supported by Sinclair BASIC. The call would need to be preceded by a LD B,\$01 instruction to make this function support a line number of 0. Credit:

Ian Collier (+3), Andrew Owen (128)]

IX points to previous ASCII digit.

```
DEC IX
LD A,(IX+$00)
OR $80
LD (IX+$00),A
POP IX
POP HL
POP DE
INC HL
INC HL
INC HL
LD ($FC9F),HL
EX DE,HL
SCF
RET
```

Set bit 7 to mark it as the end of the line number representation.

Restore registers.

HL=Address of the BASIC line+1.

DE=Line number.

HL=Points to length field of the BASIC line.

HL=Points to the command field of the BASIC line.

Store it as the next character to fetch when parsing the BASIC line to de-tokenize it.

DE=Points to the command field of the BASIC line, HL=Line number.

Signal line exists.

End of program reached, no line number available

```
L34B7: CALL L1F64
      RET
```

Use Workspace RAM configuration (physical RAM bank 7).

Return with carry flag reset to signal line does not exist.

## Insert ASCII Line Number Digit

Insert text representation of a line number digit in a buffer.

Insert a \$00 character for every leading zero.

Entry: DE=Subtraction amount (-1000, -100, -10, -1).

HL=Line number.

IX=Address of the buffer to write the ASCII line number to.

B=Indicates if digit printed yet (\$00=not printed).

Exit : IX points to next buffer location.  
 B=\$01 if digit printed.  
 HL=Line number remainder.

L34BB:	XOR A	A=Counter.
L34BC:	ADD HL,DE	Keep adding DE
	INC A	and incrementing the counter
	JR C,L34BC	until there is no carry.
	SBC HL,DE	Adjust for the last addition and.
	DEC A	counter value that caused the overflow.

A=Number of multiples of DE in the line number

	ADD A,\$30	Convert to an ASCII digit.
	LD (IX+\$00),A	Store in the buffer.
	CP '0'	\$30. Is it a zero?
	JR NZ,L34D7	Jump ahead if not.
	LD A,B	Get the 'digit printed' flag.
	OR A	
	JR NZ,L34D9	Jump ahead if already printed a digit.
	LD A,\$00	Otherwise this is a leading zero, so
	LD (IX+\$00),A	store a zero byte to indicate 'nothing to print'.
	JR L34D9	and jump ahead to point to the next buffer location.
L34D7:	LD B,\$01	Indicate 'digit printed'.
L34D9:	INC IX	Point to the next buffer location.
	RET	

## Find Address of BASIC Line with Specified Line Number

This routine finds the address of the BASIC line in the program area with the specified line number, or the next line is the specified one does not exist.

Entry: HL=Line number.  
 A=\$00 to print a leading space.  
 Exit : Carry flag set if line exists.  
 DE=Points to the command of the BASIC line within the program area.  
 HL=Line number.

L34DC:	CALL L3510	Clear BASIC line construction pointers (address of next character in the Keyword Construction Buffer and the address of the next character in the BASIC line within the program area being de-tokenized).
	OR A	<b>[BUG]</b> - Supposed to be XOR A to ensure that a leading space is shown before a command keyword is printed. However, most of the time the A register will enter the routine holding \$00 and so the bug is probably harmless. Credit: Paul Farrow]
	LD (\$FC9E),A	Store 'print a leading space' flag.
	CALL L1F3F	Use Normal RAM Configuration (physical RAM bank 0).
	CALL L351C	Find the address of the BASIC line with this line number, or the next line otherwise.
	JR NC,L34B7	Jump if does not exist.
	EX DE,HL	HL=Address of BASIC line.
	LD A,L	
	OR H	Address of \$0000, i.e. no line exists?
	SCF	Assume line number found.
	JP NZ,L3473	Jump if a line was found.
	CCF	Reset carry flag to indicate line number does not exist
	JR L34B7	and jump to make a return.

## Move to Next BASIC Line

L34F5:	PUSH HL	Save the address of the original line.
	INC HL	Skip past the line number.
	INC HL	
	LD E,(HL)	Retrieve the line length into DE.
	INC HL	

LD D,(HL)	
INC HL	
ADD HL,DE	Point to the start of the next line.
POP DE	DE=Address of original line.
RET	

## Check if at End of BASIC Program

Check whether at the end of the BASIC program.

Entry: HL=Address of BASIC line.

Exit : Carry flag reset if end of BASIC program reached.

L34FF:	LD A,(HL)	
	AND \$C0	
	SCF	Signal not at end of BASIC.
	RET Z	Return if not at end of program.
	CCF	Signal at end of BASIC.
	RET	

## Compare Line Numbers

Compare line number at (HL) has line number held in BC.

Entry: HL=Address of first line number.

BC=Second line number.

Exit : Carry flag and zero flag set if the line number matches.

Zero flag reset if no match, with carry flag set if line number held in BC is lower than the line number pointed to by HL.

L3506:	LD A,B	Test the first byte.
	CP (HL)	
	RET NZ	Return if not the same.
	LD A,C	Test the second byte.
	INC HL	
	CP (HL)	
	DEC HL	
	RET NZ	Return if not the same.
	SCF	Signal line number matches.
	RET	

## Clear BASIC Line Construction Pointers

L3510:	PUSH HL	
	LD HL,\$0000	
	LD (\$FCA1),HL	Signal no next character to fetch from the Keyword Construction Buffer.
	LD (\$FC9F),HL	Signal no next character to fetch within the BASIC line in the program area.
	POP HL	
	RET	

## Find Address of BASIC Line

This routine finds the address of the BASIC line within the program area with the specified line number.

Entry: HL=Line number to find (\$0000 for first program line).

Exit : Carry flag set if requested or next line exists.

Zero flag reset if no match, with carry flag set if line number is lower than the first program line number.

HL=Address of the BASIC line number, or \$0000 if line does not exist.

DE=Address of previous BASIC line number, or \$0000 if line does not exist.

BC=Line number.

L351C:	PUSH HL
--------	---------

## SPECTRUM +2 ROM o DISASSEMBLY

	POP BC	BC=Line number. [Quicker to have used the instructions LD B,H / LD C,L]
	LD DE,\$0000	
	LD HL,(\$5C53)	PROG. Address of the start of BASIC program.
	CALL L34FF	Test for end of BASIC program.
	RET NC	Return if at end of program.
	CALL L3506	Compare line number at (HL) with BC.
	RET C	Return if line number matches or is lower than the first program line number.
	LD A,B	
	OR C	
	SCF	
	RET Z	Return with carry and zero flags set if first program line was requested (line number 0).
L3530:	CALL L34F5	Get address of next BASIC line.
	CALL L34FF	Test for end of BASIC program.
	RET NC	Return if at end of program.
	CALL L3506	Compare line number at (HL) with BC.
	JR NC,L3530	If line number not the same or greater then back to test next line.
	RET	Exit with carry flag set if line found.

### Fetch Next De-tokenized Character from BASIC Line in Program Area

This routine translates a tokenized BASIC line within the program area into the equivalent 'typed' line, i.e. non-tokenized.

The line number has been previously converted into a string representation and is held within the Keyword Construction Buffer at \$FCA3. On each call of this routine, the next character of the BASIC line representation is fetched. Initially this is the line number characters from the Keyword Construction Buffer, and then the characters from the program line itself. As a token character is encountered, it is converted into its string representation and stored in the Keyword Construction Buffer. Then each character of this string is fetched in turn. Once all of these characters have been fetched, the next character will be from the last position accessed within the BASIC line in the program area.

Exit: Carry flag set to indicate that a character was available.

A=Character fetched.

L353D:	LD HL,(\$FCA1)	Fetch the address of the character within the Keyword Construction Buffer.
	LD A,L	
	OR H	Is there an address defined, i.e. characters still within the buffer to fetch?
	JR Z,L3562	Jump ahead if not.

There is a character within the Keyword Construction Buffer

	LD A,(HL)	Fetch a character from the buffer.
	INC HL	Point to the next character.
	CP ' '+\$80	\$A0. Was it a trailing space, i.e. the last character?
	LD B,A	Save the character.
	LD A,\$00	Signal 'print a leading space'.
	JR NZ,L354F	Jump ahead if not.
	LD A,\$FF	Signal 'do not print a leading space'.
L354F:	LD (\$FC9E),A	Store the 'print a leading space' flag value.
	LD A,B	Get the character back.
	BIT 7,A	Is it the last character in the buffer, i.e. the terminator bit is set?
	JR Z,L355A	Jump ahead if not.
	LD HL,\$0000	Signal no more characters within the Keyword Construction Buffer to fetch.
L355A:	LD (\$FCA1),HL	Store the address of the next line number/keyword character within the construction buffer, or \$0000 if no more characters.
	AND \$7F	Mask off the terminator bit.
	JP L35B5	Jump ahead to continue. [Could have saved 1 byte by using JR \$35B5 (ROM 0)]

There is no line number/keyword defined within the buffer so fetch the next tokenized character from the BASIC line in the program area

L3562:	LD HL,(\$FC9F)	Fetch the address of the next character within the BASIC line construction workspace.
	LD A,L	
	OR H	Is there a character defined, i.e. end of line not yet reached?
	JP Z,L35B7	Jump ahead if not. [Could have saved 1 byte by using JR \$35B7 (ROM 0)]
	CALL L1F3F	Use Normal RAM Configuration (physical RAM bank 0).
L356D:	LD A,(HL)	Fetch a character from the buffer.
	CP \$0E	Is it the hidden number marker indicating a floating-point representation?
	JR NZ,L357A	Jump ahead if it is not.

## SPECTRUM +2 ROM 0 DISASSEMBLY

	INC HL	Skip over it the floating-point representation.
	INC HL	
	INC HL	
	INC HL	
	INC HL	
	INC HL	
	JR L356D	Jump back to fetch the next character.
L357A:	CALL L1F64	Use Workspace RAM configuration (physical RAM bank 7).
	INC HL	Point to the next character.
	LD (\$FC9F),HL	Store the address of the next command within the BASIC line to fetch.
	CP \$A5	'RND'. Is the current character a standard '48K' keyword? ('RND' = first 48K keyword)
	JR C,L358D	Jump ahead if not.
	SUB \$A5	Reduce command code range to \$00-\$5A.

**[BUG** - The routine assumes all tokens require a leading and trailing space. However, this is not true for tokens '<=', '>=' and '<>'. Credit: Ian Collier (+3), Paul Farrow (128)]

<i>[To fix the bug, the call to \$FCAE would need to be replaced with code such as the following. Credit: Paul Farrow.]</i>		
	PUSH AF	
	CALL \$FCAE	Construct a string representation of the keyword in the Keyword Construction Buffer.
	POP AF	DE=Address of last character copied.
	CP \$22	Was it '<=' or above?
	JR C,\$353D (ROM 0)	Jump back if not to fetch and return the first character of the keyword string.
	CP \$25	Was it '<>' or below?
	JR NC,\$353D (ROM 0)	Jump back if not to fetch and return the first character of the keyword string.
	LD HL,(\$FCA1)	Is there a leading space?
	LD A,(HL)	
	CP ''	
	JR NZ,NOT_LEADING	Jump if there is not.
	INC HL	
	LD (\$FCA1),HL	Skip past the leading space.
NOT_LEADING	LD A,\$FF	Signal 'do not print a leading space'.
	LD (\$FC9E),A	
	LD A,(DE)	Is there a trailing space?
	CP '+'\$80	
	JR NZ,NOT_TRAILING	Jump if there is not.
	DEC DE	
	EX DE,HL	
	SET 7,(HL)	Set the terminator bit on the preceding character.
NOT_TRAILING		

CALL \$FCAE	Construct a string representation of the keyword in the Keyword Construction Buffer.
JP L353D	Jump back to fetch and return the first character of the keyword string. [Could have saved 1 byte by using JR \$353D (ROM 0)]

It is not a standard 48K keyword

L358D:	CP \$A3	Is it a '128K' keyword, i.e. 'SPECTRUM' or 'PLAY'?
	JR C,L35A1	Jump if not.

It is a 128K keyword

JR NZ,L3598	Jump if it is 'PLAY'.
-------------	-----------------------

Handle 'SPECTRUM'

	LD HL,L35BA	Keyword string "SPECTRUM".
	JR L359B	Jump forward.
L3598:	LD HL,L35C2	Keyword string "PLAY".
L359B:	CALL \$FCFD	Copy the keyword string characters into the Keyword Construction Buffer.

## SPECTRUM +2 ROM o DISASSEMBLY

JP L353D

Jump back to fetch and return the first character of the keyword string. [Could have saved 1 byte by using JR \$353D (ROM 0)]

Not a keyword

L35A1:	PUSH AF	Save the character.
	LD A,\$00	
	LD (\$FC9E),A	Signal to print a trailing space.
	POP AF	Get the character back.
	CP \$0D	Is it an 'Enter' character?
	JR NZ,L35B5	Jump if not to exit.

The end of the line was found so signal no further characters to fetch

	LD HL,\$0000	
	LD (\$FCA1),HL	Signal no further character to fetch from the Keyword Construction Buffer.
	LD (\$FC9F),HL	Signal no further character to fetch from the BASIC line within the program area.
L35B5:	SCF	Set the carry flag to indicate that a character was available.
	RET	

There was no character within the buffer

L35B7:	SCF	
	CCF	Reset the carry flag to indicate that a character was not available.
	RET	

## Edit Buffer Routines — Part 2

### Keywords String Table

The following strings are terminated by having bit 7 set, referenced at \$3593 (ROM 0) and \$3F2E (ROM 0).  
The table consists of the new 128K mode keywords and mis-spelled keywords.

L35BA:	DEFM "SPECTRU"
	DEFB 'M'+\$80
L35C2:	DEFM "PLA"
	DEFB 'Y'+\$80
	DEFM "GOT"
	DEFB 'O'+\$80
	DEFM "GOSU"
	DEFB 'B'+\$80
	DEFM "DEFF"
	DEFB 'N'+\$80
	DEFM "OPEN"
	DEFB '#' + \$80
	DEFM "CLOSE"
	DEFB '#' + \$80

### Indentation Settings

Copied to \$FD6A-\$FD6B.

L35DF:	DEFB \$02	Number of bytes in table.
	DEFB \$01	Flag never subsequently used. Possibly intended to indicate the start of a new BASIC line and hence whether indentation required.
	DEFB \$05	Number of characters to indent by.

## Set Indentation Settings

L35E2:	LD HL,L35DF LD DE,\$FD6A JP L3F61	HL=Address of the indentation settings data table. Destination address. Copy two bytes from \$35DF-\$35E0 (ROM 0) to \$FD6A-\$FD6B.
--------	---	---

## Store Character in Column of Edit Buffer Row

Store character in the specified column of the current edit buffer row.

Entry:	B=Column number. DE=Start address of row. A=Character to insert.
Exit :	B=Next column number.

L35EB:	LD L,B LD H,\$00 ADD HL,DE LD (HL),A INC B RET	Point to the required column. Store the character. Advance to the next column.
--------	---	--

## 'Enter' Action Handler Routine

L35F2:	CALL L360C LD A,(HL) OR \$18 LD (HL),A LD HL,\$FD6A  SET 0,(HL)  SCF RET	Null remaining column positions in the edit buffer row. Fetch the flag byte. Signal associated line number and last row in the BASIC line. Update the flag byte. [Redundant since flag never subsequently tested. Deleting these instructions would have saved 5 bytes] Flag possibly intended to indicate the start of a new BASIC line and hence whether indentation required. Signal no more characters are available, i.e. end of line.
--------	---	---

## 'Null Columns' Action Handler Routine

L3600:	CALL L360C SET 3,(HL) LD HL,\$FD6A  SET 0,(HL)  SCF RET	Null remaining column positions in the edit buffer row. Signal last row of the BASIC line in the row flag byte. [Redundant since flag never subsequently tested. Deleting these instructions would have saved 5 bytes] Flag possibly intended to indicate the start of a new BASIC line and hence whether indentation required. Signal no more characters are available, i.e. end of line.
--------	--	--

## Null Column Positions

This routine inserts null characters into the remainder of a line edit buffer row.

Entry:	B=Initial column to null. DE=Address of start of edit row.
Exit :	HL=Address of the row's flag byte.

L360C:	LD L,B LD H,\$00 ADD HL,DE LD A,\$20	HL=Number of columns. Point to column position in line edit buffer row. 32 columns.
--------	---	---

L3612:	CP B	Found specified column?
	RET Z	Return if so.
	LD (HL), \$00	Store a null in the location.
	INC HL	Next buffer position.
	INC B	Increment column position counter.
	JR L3612	Repeat for next column.

## Indent Edit Buffer Row

Indent a row by setting the appropriate number of characters in an edit buffer row to nulls, i.e. character \$00.

Entry: DE=Address of row within edit buffer.

Exit : B=First usable column number in the row.

L361A:	LD A, (\$FD6B)	Get the number of indentation columns.
	LD B, \$00	Start at first column.
L361F:	LD H, \$00	
	LD L, B	HL=Column position.
	ADD HL, DE	
	LD (HL), \$00	Put a null in the column position.
	INC B	Next position.
	DEC A	
	JR NZ, L361F	Repeat for all remaining columns.
	RET	

## Print Edit Buffer Row to Display File if Required

Print a row of the edit buffer to the display file if required.

Entry: HL=Address of edit buffer row.

L362A:	PUSH BC	Save registers.
	PUSH DE	
	PUSH HL	
	PUSH HL	Save edit buffer row address.
	LD HL, \$EEF5	
	BIT 2, (HL)	Is printing of the edit buffer row required?
	POP HL	Retrieve edit buffer row address.
	JR NZ, L363A	Jump if printing is not required.
	LD B, C	B=Cursor row position.
	CALL L3B3F	Print the edit buffer row to the screen. Returns with the carry flag set.
L363A:	POP HL	Restore registers.
	POP DE	
	POP BC	
	RET	

## Shift Up Edit Rows in Display File if Required

This routine shifts edit rows in the display file up if required, replacing the bottom row with the top entry from the Below-Screen Line Edit Buffer.

Entry: HL=Address of first row within the Below-Screen Line Edit Buffer.

C =Number of editing rows on screen.

B =Row number to shift from.

L363E:	PUSH BC	Save registers.
	PUSH DE	
	PUSH HL	
	PUSH HL	Save edit buffer row address.
	LD HL, \$EEF5	
	BIT 2, (HL)	Is updating of the display file required?
	POP HL	Retrieve edit buffer row address.
	JR NZ, L364E	Jump if updating is not required.
	LD E, C	E=Cursor row position, i.e. row to shift from.
	CALL L3AE0	Shift up edit rows in the display file, replacing the bottom row with the top entry from the Below-Screen Line Edit Buffer.

L364E:	POP HL POP DE POP BC RET	Restore registers.
--------	-----------------------------------	--------------------

## Shift Down Edit Rows in Display File if Required

This routine shifts edit rows in the display file down if required, replacing the top row with the bottom entry from the Above-Screen Line Edit Buffer.

Entry: HL=Address of next row to use within the Above-Screen Line Edit Buffer.

C =Number of editing rows on screen.

B =Row number to shift from.

L3652:	PUSH BC PUSH DE PUSH HL PUSH HL LD HL,\$EEF5 BIT 2,(HL) POP HL JR NZ,L3662 LD E,C CALL L3AE7	Save registers.  Save edit buffer row address.  Is updating of the display file required? Retrieve edit buffer row address. Jump if updating is not required. E=Cursor row position, i.e. row to shift from. Shift down edit rows in the display file, replacing the top row with the bottom entry from the Above-Screen Line Edit Buffer.
L3662:	POP HL POP DE POP BC RET	Restore registers.

## Set Cursor Attribute Colour

L3666:	PUSH AF PUSH BC PUSH DE PUSH HL LD A,B LD B,C LD C,A CALL L3ABE POP HL POP DE POP BC POP AF RET	Save registers.    Swap B with C.  Set cursor position attribute. Restore registers.
--------	---	---

## Restore Cursor Position Previous Attribute

L3675:	PUSH AF PUSH BC PUSH DE PUSH HL LD A,B LD B,C LD C,A CALL L3AD3 POP HL POP DE POP BC POP AF RET	Save registers    Column. Row. Column. Restore cursor position attribute. Restore registers.
--------	---	--

## Reset 'L' Mode

L3684:	LD A,\$00 LD (\$5C41),A LD A,\$02 LD (\$5C0A),A	Select 'L' mode. MODE. Reset repeat key duration. REPPER
L368E:	LD HL,\$5C3B LD A,(HL) OR \$0C LD (HL),A LD HL,\$EC0D BIT 4,(HL) LD HL,FLAGS3 JR NZ,L36A2 RES 0,(HL) RET	FLAGS.  Select L-Mode and Print in L-Mode.  Editor flags. Return to the calculator? \$5B66. Jump ahead if so. Select Editor/Menu mode.
L36A2:	SET 0,(HL) RET	Select BASIC/Calculator mode.

## Wait for a Key Press

Exit: A holds key code.

L36A5:	PUSH HL	Preserve contents of HL.
L36A6:	LD HL,\$5C3B	FLAGS.
L36A9:	BIT 5,(HL) JR Z,L36A9 RES 5,(HL) LD A,(\$5C08) LD HL,\$5C41 RES 0,(HL) CP \$20 JR NC,L36C8 CP \$10 JR NC,L36A6 CP \$06 JR C,L36A6	Wait for a key press. Clear the new key indicator flag. Fetch the key pressed from LAST_K. MODE. Remove extended mode. Is it a control code? Jump if not to accept all characters and token codes (used for the keypad). Is it a cursor key? Jump back if not to wait for another key. Is it a cursor key? Jump back if not to wait for another key.

Control code or cursor key

	CALL L36CA JR NC,L36A6	Handle CAPS LOCK code and 'mode' codes. Jump back if mode might have changed.
L36C8:	POP HL RET	Restore contents of HL.
L36CA:	RST 28H DEFW KEY_M_CL RET	\$10DB. Handle CAPS LOCK code and 'mode' codes via ROM 1.

## MENU ROUTINES — PART 5

### Display Menu

HL=Address of menu text.

L36CE:	PUSH HL CALL L3761 LD HL,\$5C3C RES 0,(HL) POP HL LD E,(HL)	Save address of menu text. Store copy of menu screen area and system variables. TVFLAG. Signal using main screen. HL=Address of menu text. Fetch number of table entries.
--------	--	--

## SPECTRUM +2 ROM 0 DISASSEMBLY

	INC HL	Point to first entry.
	PUSH HL	
	LD HL,L3812	Set title colours.
	CALL L3759	Print them.
	POP HL	
	CALL L3759	Print menu title pointed to by HL.
	PUSH HL	
	CALL L3848	Print Sinclair stripes.
	LD HL,L3820	Black ' '.
	CALL L3759	Print it.
	POP HL	HL=Address of first menu item text.
	PUSH DE	Save number of menu items left to print.
	LD BC,\$0807	
L36F7:	CALL L3751	Perform 'Print AT 8,7;' (this is the top left position of the menu).
	PUSH BC	Save row print coordinates.
	LD B,\$0C	Number of columns in a row of the menu.
	LD A,\$20	Print ' '.
	RST 10H	
L36FD:	LD A,(HL)	Fetch menu item character.
	INC HL	
	CP \$80	End marker found?
	JR NC,L3706	Jump if end of text found.
	RST 10H	Print menu item character
	DJNZ L36FD	Repeat for all characters in menu item text.
L3706:	AND \$7F	Clear bit 7 to yield a final text character.
	RST 10H	Print it.
L3709:	LD A,\$20	
	RST 10H	Print trailing spaces
	DJNZ L3709	Until all columns filled.
	POP BC	Fetch row print coordinates.
	INC B	Next row.
	CALL L3751	Print AT.
	DEC E	
	JR NZ,L36F7	Repeat for all menu items.
	LD HL,\$6F38	Coordinates, pixel (111, 56) = end row 13, column 7.
	POP DE	Fetch number of menu items to E.
	SLA E	
	SLA E	
	SLA E	
	LD D,E	Determine number of pixels to span all menu items.
	DEC D	
	LD E,\$6F	D=8*Number of menu items - 1.
	LD BC,\$FF00	Number of pixels in width of menu.
	LD A,D	B=-1, C=0. Plot a vertical line going up.
	CALL L373F	A=Number of vertical pixels to plot.
	LD BC,\$0001	Plot line.
	LD A,E	B=0, C=1. Plot a horizontal line going to the right.
	CALL L373F	A=Number of horizontal pixels to plot.
	LD BC,\$0100	Plot line.
	LD A,D	B=1, C=0. Plot a vertical line going down.
	INC A	A=Number of vertical pixels to plot.
	CALL L373F	Include end pixel.
	XOR A	Plot line.
	CALL L37F0	A=Index of menu option to highlight.
	RET	Toggle menu option selection so that it is highlight.
		[Could have saved one byte by using JP \$37F0 (ROM 0)]

## Plot a Line

L373F:	PUSH AF	Save registers.
	PUSH HL	
	PUSH DE	
	PUSH BC	
	LD B,H	Coordinates to BC.
	LD C,L	

RST 28H	
DEFW PLOT_SUB+4	\$22E9. Plot pixel
POP BC	Restore registers.
POP DE	
POP HL	
POP AF	
ADD HL,BC	Determine coordinates of next pixel.
DEC A	
JR NZ,L373F	Repeat for all pixels.
RET	

## Print "AT B,C" Characters

L3751:	LD A,\$16	'AT'.
	RST 10H	Print.
	LD A,B	B=Row number.
	RST 10H	Print.
	LD A,C	C=Column number.
	RST 10H	Print.
	RET	

## Print String

Print characters pointed to by HL until \$FF found.

L3759:	LD A,(HL)	Fetch a character.
	INC HL	Advance to next character.
	CP \$FF	Reach end of string?
	RET Z	Return if so.
	RST 10H	Print the character.
	JR L3759	Back for the next character.

## Store Menu Screen Area

Store copy of menu screen area and system variables.

L3761:	SCF	Set carry flag to signal to save screen area.
	JR L3765	Jump ahead to continue.

## Restore Menu Screen Area

Restore menu screen area and system variables from copy.

Entry: IX=Address of the cursor settings information.

L3764:	AND A	Reset carry flag to signal restore screen area.
L3765:	LD DE,\$EEF6	Store for TVFLAG.
	LD HL,\$5C3C	TVFLAG.
	JR C,L376E	Jump if storing copies.
	EX DE,HL	Exchange source and destination pointers.
L376E:	LDI	Transfer the byte.
	JR C,L3773	Jump if storing copies.
	EX DE,HL	Restore source and destination pointers.
L3773:	LD HL,\$5C7D	COORDS. DE=\$EEF7 by now.
	JR C,L3779	Jump if storing copies.
	EX DE,HL	Exchange source and destination pointers.
L3779:	LD BC,\$0014	Copy 20 bytes.
	LDIR	Copy COORDS until ATTR_T.
	JR C,L3781	Jump if storing copies.
	EX DE,HL	Restore source and destination pointers.
L3781:	EX AF,AF'	Save copy direction flag.

## SPECTRUM +2 ROM o DISASSEMBLY

	LD BC,\$0707	Menu will be at row 7, column 7.
	CALL L3BB5	B=Number of rows to end row of screen. C=Number of columns to the end column of the screen.
	LD A,(IX+\$01)	A=Rows above the editing area (\$16 when using the lower screen, \$00 when using the main screen).
	ADD A,B	B=Row number within editing area.
	LD B,A	B=Bottom screen row to store.
	LD A,\$0C	A=Number of rows to store. [Could have been just \$07 freeing up 630 bytes of workspace]
L378F:	PUSH BC	B holds number of row to store.
	PUSH AF	A holds number of rows left to store.
	PUSH DE	DE=End of destination address.
	RST 28H	
	DEFW CL_ADDR	\$0E9B. HL=Display file address of row B.
	LD BC,\$0007	Menu always starts at column 7.
	ADD HL,BC	HL=Address of attribute byte at column 7.
	POP DE	
	CALL L37A4	Store / restore menu screen row.
	POP AF	
	POP BC	
	DEC B	Next row.
	DEC A	More rows to store / restore?
	JR NZ,L378F	Repeat for next row
	RET	

### Store / Restore Menu Screen Row

Entry: HL=Start address of menu row in display file.  
DE=Screen location/Workspace store for screen row.  
AF'=Carry flag set for store to workspace, reset for restore to screen.

Exit : DE=Screen location/workspace store for next screen row.

Save the display file bytes

L37A4:	LD BC,\$080E	B=Menu row is 8 lines deep. C=Menu is 14 columns wide.
L37A7:	PUSH BC	Save number of row lines.
	LD B,\$00	Just keep the column count in BC.
	PUSH HL	Save display file starting address.
	EX AF,AF'	Retrieve copy direction flag.
	JR C,L37AF	Jump if storing copies of display file bytes.
	EX DE,HL	Exchange source and destination pointers.
L37AF:	LDIR	Copy the row of menu display file bytes.
	JR C,L37B4	Jump if storing copies of display file bytes.
	EX DE,HL	Restore source and destination pointers.
L37B4:	EX AF,AF'	Save copy direction flag.
	POP HL	Fetch display file starting address.
	INC H	Advance to next line
	POP BC	Fetch number of lines.
	DJNZ L37A7	Repeat for next line.

Now save the attributes

	PUSH BC	B=0. C=Number of columns.
	PUSH DE	DE=Destination address.
	RST 28H	
	DEFW CL_ATTR	\$0E88. HL=Address of attribute byte.
	EX DE,HL	DE=Address of attribute byte.
	POP DE	
	POP BC	
	EX AF,AF'	Retrieve copy direction flag.
	JR C,L37C6	Jump if storing copies of attribute bytes.
	EX DE,HL	Restore source and destination pointers.
L37C6:	LDIR	Copy the row of menu attribute bytes.
	JR C,L37CB	Jump if storing copies of attribute bytes.
	EX DE,HL	Restore source and destination pointers.

L37CB:        EX AF,AF'  
              RET                                Save copy direction flag.

## Move Up Menu

L37CD:        CALL L37F0                                Toggle old menu item selection to de-highlight it.  
              DEC A                                        Decrement menu index.  
              JP P,L37D7                                Jump if not exceeded top of menu.  
              LD A,(HL)                                Fetch number of menu items.  
              DEC A                                        Ignore the title.  
              DEC A                                        Make it indexed from 0.  
L37D7:        CALL L37F0                                Toggle new menu item selection to highlight it.  
              SCF                                        Ensure carry flag is set to prevent immediately  
              RET                                        calling menu down routine upon return.

## Move Down Menu

L37DC:        PUSH DE                                        Save DE.  
              CALL L37F0                                Toggle old menu item selection to de-highlight it.  
              INC A                                        Increment menu index.  
              LD D,A                                        Save menu index.  
              LD A,(HL)                                fetch number of menu items.  
              DEC A                                        Ignore the title.  
              DEC A                                        Make it indexed from 0.  
              CP D                                        Has bottom of menu been exceeded?  
              LD A,D                                        Fetch menu index.  
              JP P,L37EB                                Jump if bottom menu not exceeded.  
              XOR A                                        Select top menu item.  
L37EB:        CALL L37F0                                Toggle new menu item selection to highlight it.  
              POP DE                                        Restore DE.  
              RET

## Toggle Menu Option Selection Highlight

L37F0:        PUSH AF                                        Save registers.  
              PUSH HL  
              PUSH DE  
              LD HL,\$5907                                First attribute byte at position (9,7).  
              LD DE,\$0020                                The increment for each row.  
              AND A  
              JR Z,L3800                                Jump ahead if highlighting the first entry.  
L37FC:        ADD HL,DE                                        Otherwise increase HL  
              DEC A                                        for each row.  
              JR NZ,L37FC  
L3800:        LD A,\$78                                        Flash 0, Bright 1, Paper 7, Ink 0 = Bright white.  
              CP (HL)                                        Is the entry already highlighted?  
              JR NZ,L3807                                Jump ahead if not.  
              LD A,\$68                                        Flash 0, Bright 1, Paper 5, Ink 0 = Bright cyan.  
L3807:        LD D,\$0E                                        There are 14 columns to set.  
L3809:        LD (HL),A                                        Set the attributes for all columns.  
              INC HL  
              DEC D  
              JR NZ,L3809  
              POP DE                                        Restore registers.  
              POP HL  
              POP AF  
              RET

## Menu Title Colours Table

L3812:	DEFB \$16, \$07, \$07	AT 7,7
	DEFB \$15, \$00	OVER 0
	DEFB \$14, \$00	INVERSE 0
	DEFB \$10, \$07	INK 7
	DEFB \$11, 00	PAPER 0
	DEFB \$13, \$01	BRIGHT 1
	DEFB \$FF	

## Menu Title Space Table

L3820:	DEFB \$11, \$00	PAPER 0
	DEFB ''	
	DEFB \$11, \$07	PAPER 7
	DEFB \$10, \$00	INK 0
	DEFB \$FF	

## Menu Sinclair Stripes Bitmaps

Bit-patterns for the Sinclair stripes used on the menus.

L3828:	DEFB \$01	0 0 0 0 0 0 0 1	X
	DEFB \$03	0 0 0 0 0 0 1 1	XX
	DEFB \$07	0 0 0 0 0 1 1 1	XXX
	DEFB \$0F	0 0 0 0 1 1 1 1	XXXX
	DEFB \$1F	0 0 0 1 1 1 1 1	XXXXXX
	DEFB \$3F	0 0 1 1 1 1 1 1	XXXXXXXX
	DEFB \$7F	0 1 1 1 1 1 1 1	XXXXXXXXX
	DEFB \$FF	1 1 1 1 1 1 1 1	XXXXXXXXXX
	DEFB \$FE	1 1 1 1 1 1 1 0	XXXXXXXXX
	DEFB \$FC	1 1 1 1 1 1 0 0	XXXXXXX
	DEFB \$F8	1 1 1 1 1 0 0 0	XXXXXX
	DEFB \$F0	1 1 1 1 0 0 0 0	XXXXX
	DEFB \$E0	1 1 1 0 0 0 0 0	XXX
	DEFB \$C0	1 1 0 0 0 0 0 0	XX
	DEFB \$80	1 0 0 0 0 0 0 0	X
	DEFB \$00	0 0 0 0 0 0 0 0	

## Sinclair Strip 'Text'

CHARS points to RAM at \$5A98, and characters '' and '!' redefined as the Sinclair strips using the bit patterns above.

L3838:	DEFB \$10, \$02, ''	INK 2
	DEFB \$11, \$06, '!'	PAPER 6
	DEFB \$10, \$04, ''	INK 4
	DEFB \$11, \$05, '!'	PAPER 5
	DEFB \$10, \$00, ''	INK 0
	DEFB \$FF	

## Print the Sinclair stripes on the menu

L3848:	PUSH BC	Save registers.
	PUSH DE	
	PUSH HL	
	LD HL,L3828	Graphics bit-patterns
	LD DE,STRIP1	\$5B98.

```
LD BC,$0010
LDIR
LD HL,($5C36)
PUSH HL
LD HL,STRIP1-$0100
LD ($5C36),HL
LD HL,L3838
CALL L3759
POP HL
LD ($5C36),HL
POP HL
POP DE
POP BC
RET
```

Copy two characters.

Save CHARS.

\$5A98.

Set CHARS to point to new graphics.

Point to the strip string.

Print it.

Restore CHARS.

Restore registers.

## Print '128 BASIC' Banner

```
L386E:    LD HL,L2785
          JR L387B
```

"128 BASIC" text from main menu.

Jump ahead to print banner.

## Print 'Calculator' Banner

```
L3873:    LD HL,L278E
          JR L387B
```

"Calculator" text from main menu.

Jump ahead to print banner.

## Print 'Tape Loader' Banner

```
L3878:    LD HL,L277A
```

"Tape Loader" text from main menu.

## Print Banner

```
L387B:    PUSH HL
          CALL L38A2
          LD HL,$5AA0
          LD B,$20
          LD A,$40
L3886:    LD (HL),A
          INC HL
          DJNZ L3886
          LD HL,L3812
          CALL L3759
          LD BC,$1500
          CALL L3751
          POP DE
          CALL L059C
          LD C,$1A
          CALL L3751
          JP L3848
```

Address in memory of the text of the selected menu item.

Clear lower editing area display.

Address of banner row in attributes.

32 columns.

FLASH 0, BRIGHT 1, PAPER 0, INK 0.

Set a black row.

Menu title colours table.

Print the colours as a string.

Perform 'Print AT 21,0;'.

Address in memory of the text of the selected menu item.

Print the text.

B has not changed and still holds 21.

Perform 'Print AT 21,26;'.

Print Sinclair stripes and return to calling routine.

## Clear Lower Editing Display

```
L38A2:    LD B,$15
          LD D,$17
          JP L3B7F
```

Top row of editing area.

Bottom row of editing area.

Reset Display.

## RENUMBER ROUTINE

Exit: Carry flag reset if required to produce an error beep.

L38A9:	CALL L1F3F CALL L3A26 LD A,D OR E JP Z,L39E1 LD HL,(RNSTEP) RST 28H DEFW HL_MULT_DE	Use Normal RAM Configuration (physical RAM bank 0). DE=Count of the number of BASIC lines.  Were there any BASIC lines? Jump if not to return since there is nothing to renumber. \$5B96. Fetch the line number increment for Renumber.  \$30A9. HL=HL*DE in ROM 1. HL=Number of lines * Line increment = New last line number. <b>[BUG]</b> - If there are more than 6553 lines then an arithmetic overflow will occur and hence the test below to check if line 9999 would be exceeded will fail. The carry flag will be set upon such an overflow and simply needs to be tested. The bug can be resolved by following the call to HL_MULT_DE with a JP C,\$39E1 (ROM 0) instruction. Credit: Ian Collier (+3), Andrew Owen (128)] DE=Offset of new last line number from the first line number. \$5B94. Starting line number for Renumber. HL=New last line number. 10000.
	EX DE,HL LD HL,(RNFIRST) ADD HL,DE LD DE,\$2710 OR A SBC HL,DE JP NC,L39E1	Would the last line number above 9999? Jump if so to return since Renumber cannot proceed.

There is a program that can be renumbered

L38CB:	LD HL,(\$5C53) RST 28H DEFW NEXT_ONE INC HL INC HL LD (RNLIN),HL INC HL INC HL LD (N_STR1+4),DE LD A,(HL) RST 28H DEFW NUMBER CP \$0D JR Z,L38E6 CALL L392F JR L38D9	PROG. HL=Address of first BASIC line. Find the address of the next BASIC line from the \$19B8. location pointed to by HL, returning it in DE. Advance past the line number bytes to point at the line length bytes. \$5B92. Store the address of the BASIC line's length bytes. Advance past the line length bytes to point at the command. \$5B6B. Store the address of the next BASIC line. Get a character from the BASIC line. Advance past a floating point number, if present. \$18B6. Is the character an 'ENTER'? Jump if so to examine the next line. Parse the line, renumbering any tokens that may be followed by a line number. Repeat for all remaining character until end of the line.
L38D9:	LD DE,(N_STR1+4) LD HL,(\$5C4B) AND A SBC HL,DE EX DE,HL JR NZ,L38CB	\$5B6B. DE=Address of the next BASIC line. VARS. Fetch the address of the end of the BASIC program.  Has the end of the BASIC program been reached? HL=Address of start of the current BASIC line. Jump back if not to examine the next line.
L38E6:		

The end of the BASIC program has been reached so now it is time to update the line numbers and line lengths.

L38FE:	CALL L3A26 LD B,D LD C,E LD DE,\$0000 LD HL,(\$5C53) PUSH BC PUSH DE PUSH HL LD HL,(RNSTEP) RST 28H DEFW HL_MULT_DE LD DE,(RNFIRST)	DE=Count of the number of BASIC lines.  BC=Count of the number of BASIC lines.  PROG. HL=Address of first BASIC line. BC=Count of number of lines left to update. DE=Index of the current line. HL=Address of current BASIC line. \$5B96. HL=Renumber line increment. Calculate new line number offset, i.e. Line increment * Line index. \$30A9. HL=HL*DE in ROM 1. \$5B94. The initial line number when renumbering.
--------	--	---

## SPECTRUM +2 ROM 0 DISASSEMBLY

ADD HL,DE	HL=The new line number for the current line.
EX DE,HL	DE=The new line number for the current line.
POP HL	HL=Address of current BASIC line.
LD (HL),D	Store the new line number for this line.
INC HL	
LD (HL),E	
INC HL	
LD C,(HL)	Fetch the line length.
INC HL	
LD B,(HL)	
INC HL	
ADD HL,BC	Point to the next line.
POP DE	DE=Index of the current line.
INC DE	Increment the line index.
POP BC	BC=Count of number of lines left to update.
DEC BC	Decrement counter.
LD A,B	
OR C	
JR NZ,L38FE	Jump back while more lines to update.
CALL L1F64	Use Workspace RAM configuration (physical RAM bank 7).
LD (RNLINE),BC	\$5B92. Clear the address of line length bytes of the 'current line being renumbered'.
	[No need to clear this]
SCF	Signal not to produce an error beep.
RET	

## Tokens Using Line Numbers

A list of all tokens that maybe followed by a line number and hence require consideration.

L3928:	DEFB \$CA	'LINE'.
	DEFB \$F0	'LIST'.
	DEFB \$E1	'LLIST'.
	DEFB \$EC	'GO TO'.
	DEFB \$ED	'GO SUB'.
	DEFB \$E5	'RESTORE'.
	DEFB \$F7	'RUN'.

## Parse a Line Renumbering Line Number References

This routine examines a BASIC line for any tokens that may be followed by a line number reference and if one is found then the new line number is calculated and substituted for the old line number reference. Although checks are made to ensure an out of memory error does not occur, the routine simply returns silently in such scenarios and the renumber routine will continue onto the next BASIC line.

Entry: HL=Address of current character in the current BASIC line.  
A=Current character.

L392F:	INC HL	Point to the next character.
	LD (HD_11+1),HL	\$5B79. Store it.
	EX DE,HL	DE=Address of next character.
	LD BC,\$0007	There are 7 tokens that may be followed by a line number, and these are listed in the table at \$3928 (ROM 0).
	LD HL,L3928	
	CPIR	Search for a match for the current character.
	EX DE,HL	HL=Address of next character.
	RET NZ	Return if no match found.

A token that might be followed by a line number was found. If it is followed by a line number then proceed to renumber the line number reference. Note that the statements such as GO TO VAL "100" will not be renumbered. The line numbers of each BASIC line will be renumbered as the last stage of the renumber process at \$38F3 (ROM 0).

	LD C,\$00	Counts the number of digits in the current line number representation. B will be \$00 from above.
L3940:	LD A,(HL)	Fetch the next character.
	CP ''	\$20. Is it a space?
	JR Z,L3960	Jump ahead if so to parse the next character.

## SPECTRUM +2 ROM 0 DISASSEMBLY

```
RST 28H
DEFW NUMERIC
JR NC,L3960
CP ':'
JR Z,L3960
CP $0E
JR Z,L3964
OR $20
CP 'e'
JR NZ,L395C
LD A,B
OR C
JR NZ,L3960
```

\$2D1B. Is the character a numeric digit?  
 Jump if a numeric digit to parse the next character.  
 \$2E. Is it a decimal point?  
 Jump ahead if so to parse the next character.  
 Does it indicate a hidden number?  
 Jump ahead if so to process it.  
 Convert to lower case.  
 \$65. Is it an exponent 'e'?  
 Jump if not to parse the next character.  
 Have any digits been found?  
  
 Jump ahead to parse the next character.

A line number reference was not found

```
L395C:    LD HL,(HD_11+1)
          RET
L3960:    INC BC
          INC HL
          JR L3940
```

\$5B79. Retrieve the address of the next character.  
  
 Increment the number digit counter.  
 Point to the next character.  
 Jump back to parse the character at this new address.

An embedded number was found

```
L3964:    LD (HD_00),BC
          PUSH HL
          RST 28H
          DEFW NUMBER
          CALL L3A57
          LD A,(HL)
          POP HL
          CP ':'
          JR Z,L3978
          CP $0D
          RET NZ
```

\$5B71. Note the number of digits in the old line number reference.  
 Save the address of the current character.  
  
 \$18B6. Advance past internal floating point representation, if present.  
 Skip over any spaces.  
 Fetch the new character.  
 HL=Address of the current character.  
 \$3A. Is it ':'?  
 Jump if so.  
 Is it 'ENTER'?  
 Return if not.

End of statement/line found

```
L3978:    INC HL
          RST 28H
          DEFW STACK_NUM
          RST 28H
          DEFW FP_TO_BC

          LD H,B
          LD L,C
          RST 28H
          DEFW LINE_ADDR
          JR Z,L3990
          LD A,(HL)
          CP $80

          JR NZ,L3990
          LD HL,$270F
          JR L39A1
```

Point to the next character.  
  
 \$33B4. Move floating point number to the calculator stack.  
  
 \$2DA2. Fetch the number line to BC. [**BUG** - This should test the carry flag to check whether the number was too large to be transferred to BC. If so then the line number should be set to 9999, as per the instructions at \$398B (ROM 0). As a result, the call the LINE\_ADDR below can result in a crash. The bug can be resolved using a JR C, \$398B (ROM 0) instruction. Credit: Ian Collier (+3), Andrew Owen (128)]  
  
 Transfer the number line to HL.  
 Find the address of the line number specified by HL.  
 \$196E. HL=Address of the BASIC line, or the next one if it does not exist.  
 Jump if the line exists.  
 Has the end of the BASIC program been reached?  
 [**BUG** - This tests for the end of the variables area and not the end of the BASIC program area. Therefore, the renumber routine will not terminate properly if variables exist in memory when it is called. Executing CLEAR prior to renumbering will overcome this bug. It can be fixed by replacing CP \$80 with the instructions AND \$C0 / JR Z,\$3990 (ROM 0). Credit: Ian Collier (+3), Andrew Owen (128)]  
 Jump ahead if not.  
 Make the reference point to line 9999.  
 Jump ahead to update the reference to use the new line number.

The reference line exists

```
L3990:    LD (HD_0F+1),HL
```

\$5B77. Store the address of the referenced line.

## SPECTRUM +2 ROM 0 DISASSEMBLY

CALL L3A2C  
LD HL,(RNSTEP)  
RST 28H  
DEFW HL\_MULT\_DE

LD DE,(RNFIRST)  
ADD HL,DE

DE=Count of the number of BASIC lines up to the referenced line.  
\$5B96. Fetch the line number increment.

\$30A9. HL=HL\*DE in ROM 1. HL=Number of lines \* Line increment = New referenced line number. [An overflow could occur here and would not be detected. The code at \$38B9 (ROM 0) should have trapped that such an overflow would occur and hence there would have been no possibility of it occurring here.]  
\$5B94. Starting line number for Renumber.  
HL=New referenced line number.

HL=New line number being referenced

L39A1: LD DE,HD\_0B+1  
PUSH HL  
CALL L3A5D  
LD E,B  
INC E  
LD D,\$00  
PUSH DE  
PUSH HL  
LD L,E  
LD H,\$00  
LD BC,(HD\_00)  
OR A  
SBC HL,BC  
LD (HD\_00),HL  
  
JR Z,L39F0  
JR C,L39E6

\$5B73. Temporary buffer to generate ASCII representation of the new line number.  
Save the new line number being referenced.  
Create the ASCII representation of the line number in the buffer.

DE=Number of digits in the new line number.  
DE=Number of digits in the new line number.  
HL=Address of the first non-'0' character in the buffer.

HL=Number of digits in the new line number.  
\$5B71. Fetch the number of digits in the old line number reference.

Has the number of digits changed?  
\$5B71. Store the difference between the number of digits in the old and new line numbers.  
Jump if they are the same length.  
Jump if the new line number contains less digits than the old.

The new line number contains more digits than the old line number

LD B,H  
LD C,L  
LD HL,(HD\_11+1)  
  
PUSH HL  
PUSH DE  
LD HL,(\$5C65)  
ADD HL,BC  
JR C,L39DF  
EX DE,HL  
LD HL,\$0082  
ADD HL,DE  
JR C,L39DF  
SBC HL,SP  
CCF  
JR C,L39DF  
POP DE  
POP HL  
RST 28H  
DEFW MAKE\_ROOM  
JR L39F0

BC=Length of extra space required for the new line number.  
\$5B79. Fetch the start address of the old line number representation within the BASIC line.  
Save start address of the line number reference.  
DE=Number of non-'0' characters in the line number string.  
STKEND. Fetch the start of the spare memory.  
Would a memory overflow occur if the space were created?  
Jump if not to return without changing the line number reference.  
DE=New STKEND address.  
Would there be at least 130 bytes at the top of RAM?  
  
Jump if not to return without changing the line number reference.  
Is the new STKEND address below the stack?  
  
Jump if not to return without changing the line number reference.  
DE=Number of non-'0' characters in the line number string.  
HL=Start address of line number reference.  
  
\$1655. Create the space for the extra line number digits.  
Jump ahead to update the number digits.

No room available to insert extra line number digits

L39DF: POP DE  
POP HL

Discard stacked items.

[At this point the stack contains 3 surplus items. These are not explicitly popped off the stack since the call to \$1F64 (ROM 0) will restore the stack to the state it was in at \$38A9 (ROM 0) when the call to \$1F3F (ROM 0) saved it.] Exit if no BASIC program, renumbering would cause a line number overflow or renumbering would cause an out of memory condition

L39E1: CALL L1F64  
AND A

Use Workspace RAM configuration (physical RAM bank 7).  
Reset the carry flag so that an error beep will be produced.

RET

The new line number contains less digits than the old line number

L39E6:	DEC BC	BC=Number of digits in the old line number reference.
	DEC E	Decrement number of digits in the new line number.
	JR NZ,L39E6	Repeat until BC has been decremented by the number of digits in the new line number, thereby leaving BC holding the number of digits in the BASIC line to be discarded.
	LD HL,(HD_11+1)	\$5B79. Fetch the start address of the old line number representation within the BASIC line.
	RST 28H	
	DEFW RECLAIM_2	\$19E8. Discard the redundant bytes.

The appropriate amount of space now exists in the BASIC line so update the line number value

L39F0:	LD DE,(HD_11+1)	\$5B79. Fetch the start address of the old line number representation within the BASIC line.
	POP HL	HL=Address of the first non-'0' character in the buffer.
	POP BC	BC=Number of digits in the new line number.
	LDIR	Copy the new line number into place.
	EX DE,HL	HL=Address after the line number text in the BASIC line.
	LD (HL),\$0E	Store the hidden number marker.
	POP BC	Retrieve the new line number being referenced.
	INC HL	HL=Address of the next position within the BASIC line.
	PUSH HL	
	RST 28H	
	DEFW STACK_BC	\$2D2B. Put the line number on the calculator stack, returning HL pointing to it. <b>[BUG</b>
		- This stacks the new line number so that the floating point representation can be copied. However, the number is not actually removed from the calculator stack. Therefore the amount of free memory reduces by 5 bytes as each line with a line number reference is renumbered. A call to FP_TO_BC (at \$2DA2 within ROM 1) after the floating point form has been copied would fix the bug. Note that all leaked memory is finally reclaimed when control is returned to the Editor but the bug could prevent large programs from being renumbered. Credit: Paul Farrow]
	POP DE	DE=Address of the next position within the BASIC line.
	LD BC,\$0005	
	LDIR	Copy the floating point form into the BASIC line.
	EX DE,HL	HL=Address of character after the newly inserted floating point number bytes.
	PUSH HL	
	LD HL,(RNLIN)	\$5B92. HL=Address of the current line's length bytes.
	PUSH HL	
	LD E,(HL)	
	INC HL	
	LD D,(HL)	DE=Existing length of the current line.
	LD HL,(HD_00)	\$5B71. HL=Change in length of the line.
	ADD HL,DE	
	EX DE,HL	DE=New length of the current line.
	POP HL	HL=Address of the current line's length bytes.
	LD (HL),E	
	INC HL	
	LD (HL),D	Store the new length.
	LD HL,(N_STR1+4)	\$5B6B. HL=Address of the next BASIC line.
	LD DE,(HD_00)	\$5B71. DE=Change in length of the current line.
	ADD HL,DE	
	LD (N_STR1+4),HL	\$5B6B. Store the new address of the next BASIC line.
	POP HL	HL=Address of character after the newly inserted floating point number bytes.
	RET	

## Count the Number of BASIC Lines

This routine counts the number of lines in the BASIC program, or if entered at \$3A2C (ROM 0) counts the number of lines up in the BASIC program to the address specified in HD\_0F+1.

Exit: DE=Number of lines.

## SPECTRUM +2 ROM o DISASSEMBLY

L3A26:	LD HL,(\$5C4B)	VARs. Fetch the address of the variables
	LD (HD_0F+1),HL	\$5B77. and store it.
L3A2C:	LD HL,(\$5C53)	PROG. Fetch the start of the BASIC program
	LD DE,(HD_0F+1)	\$5B77. and compare against the address of
	OR A	the end address to check whether there is
	SBC HL,DE	a BASIC program.
	JR Z,L3A52	Jump if there is no BASIC program.
	LD HL,(\$5C53)	PROG. Fetch the start address of the BASIC program.
	LD BC,\$0000	A count of the number of lines.
L3A3E:	PUSH BC	Save the line number count.
	RST 28H	Find the address of the next BASIC line from the
	DEFW NEXT_ONE	\$19B8. location pointed to by HL, returning it in DE.
	LD HL,(HD_0F+1)	\$5B77. Fetch the start of the variables area,
	AND A	i.e. end of the BASIC program.
	SBC HL,DE	
	JR Z,L3A4F	Jump if end of BASIC program reached.
	EX DE,HL	HL=Address of current line.
	POP BC	Retrieve the line number count.
	INC BC	Increment line number count.
	JR L3A3E	Jump back to look for the next line.
L3A4F:	POP DE	Retrieve the number of BASIC lines and
	INC DE	increment since originally started on a line.
	RET	

No BASIC program

L3A52:	LD DE,\$0000	There are no BASIC lines.
	RET	

## Skip Spaces

L3A56:	INC HL	Point to the next character.
L3A57:	LD A,(HL)	Fetch the next character.
	CP ' '	\$20. Is it a space?
	JR Z,L3A56	Jump if so to skip to next character.
	RET	

## Create ASCII Line Number Representation

Creates an ASCII representation of a line number, replacing leading zeros with spaces.

Entry:	HL=The line number to convert.
	DE=Address of the buffer to build ASCII representation in.
	B=Number of non-'0' characters minus 1 in the ASCII representation.
Exit :	HL=Address of the first non-'0' character in the buffer.

L3A5D:	PUSH DE	Store the buffer address.
	LD BC,\$FC18	BC=-1000.
	CALL L3A81	Insert how many 1000s there are.
	LD BC,\$FF9C	BC=-100.
	CALL L3A81	Insert how many 100s there are.
	LD C,\$F6	BC=-10.
	CALL L3A81	Insert how many 10s there are.
	LD A,L	A=Remainder.
	ADD A,'0'	\$30. Convert into an ASCII character ('0'..'9').
	LD (DE),A	Store it in the buffer.
	INC DE	Point to the next buffer position.

Now skip over leading zeros

	LD B,\$03	Skip over 3 leading zeros at most.
	POP HL	Retrieve the buffer start address.
L3A77:	LD A,(HL)	Fetch a character.

CP '0'	\$30. Is it a leading zero?
RET NZ	Return as soon as a non-'0' character is found.
LD (HL),' '	\$20. Replace it with a space.
INC HL	Point to the next buffer location.
DJNZ L3A77	Repeat until all leading zeros removed.
RET	

## Insert Line Number Digit

This routine effectively works out the result of HL divided by BC. It does this by repeatedly adding a negative value until no overflow occurs.

Entry: HL=Number to test.  
 BC=Negative amount to add.  
 DE=Address of buffer to insert ASCII representation of the number of divisions.

Exit : HL=Remainder.  
 DE=Next address in the buffer.

L3A81:	XOR A	Assume a count of 0 additions.
L3A82:	ADD HL,BC	Add the negative value.
	INC A	Increment the counter.
	JR C,L3A82	If no overflow then jump back to add again.
	SBC HL,BC	Undo the last step
	DEC A	and the last counter increment.
	ADD A,'0'	\$30. Convert to an ASCII character ('0'..'9').
	LD (DE),A	Store it in the buffer.
	INC DE	Point to the next buffer position.
	RET	

## EDITOR ROUTINES — PART 4

### Initial Lower Screen Cursor Settings

Copied to \$FD6C-\$FD73.

L3A8E:	DEFB \$08	Number of bytes in table.
	DEFB \$00	\$FD6C. [Setting never used]
	DEFB \$00	\$FD6D = Rows above the editing area.
	DEFB \$14	\$FD6E. [Setting never used]
	DEFB \$00	\$FD6F. [Setting never used]
	DEFB \$00	\$FD70. [Setting never used]
	DEFB \$00	\$FD71. [Setting never used]
	DEFB \$0F	\$FD72 = Cursor attribute colour (blue paper, white ink).
	DEFB \$00	\$FD73 = Stored cursor position screen attribute colour (None = black paper, black ink).

### Initial Main Screen Cursor Settings

Copied to \$FD6C-\$FD73.

L3A97:	DEFB \$08	Number of bytes in table.
	DEFB \$00	\$FD6C. [Setting never used]
	DEFB \$16	\$FD6D = Rows above the editing area.
	DEFB \$01	\$FD6E. [Setting never used]
	DEFB \$00	\$FD6F. [Setting never used]
	DEFB \$00	\$FD70. [Setting never used]
	DEFB \$00	\$FD71. [Setting never used]
	DEFB \$0F	\$FD72 = Cursor attribute colour (blue paper, white ink).
	DEFB \$00	\$FD73 = Stored cursor position screen attribute colour (None = black paper, black ink).

## Set Main Screen Editing Cursor Details

Set initial cursor editing settings when using the main screen.  
Copies 8 bytes from \$3A8F-\$3A96 (ROM 0) to \$FD6C-\$FD73.

L3AA0:	LD IX,\$FD6C LD HL,L3A8E JR L3AAC	Point IX at cursor settings in workspace. Initial values table for the lower screen cursor settings. Jump ahead.
--------	---	--

## Set Lower Screen Editing Cursor Details

Set initial cursor editing settings when using the lower screen.  
Copies 8 bytes from \$3A98-\$3A9F (ROM 0) to \$FD6C-\$FD73.

L3AA9:	LD HL,L3A97	Initial values table for the main screen cursor settings.
L3AAC:	LD DE,\$FD6C JP L3F61	DE=Cursor settings in workspace. Jump to copy the settings.

## UNUSED ROUTINES — PART 2

### Print 'AD'

This routine prints to the current channel the contents of register A and then the contents of register D.  
[Never called by ROM].

L3AB2:	RST 10H LD A,D RST 10H SCF RET	Print character held in A.  Print character held in D.
--------	--	--

## EDITOR ROUTINES — PART 5

### Store Cursor Colour

L3AB7:	AND \$3F LD (IX+\$06),A SCF RET	Mask off flash and bright bits. Store it as the new cursor attribute value.
--------	--	--

### Set Cursor Position Attribute

L3ABE:	LD A,(IX+\$01)  ADD A,B LD B,A CALL L3BC1 LD A,(HL) LD (IX+\$07),A CPL AND \$C0 OR (IX+\$06) LD (HL),A SCF RET	A=Rows above the editing area (\$16 when using the lower screen, \$00 when using the main screen). B=Row number within editing area. B=Screen row number. Get address of attribute byte into HL. Fetch current attribute byte. Store the current attribute byte. Invert colours. Mask off flash and bright bits. Get cursor colour. Store new attribute value to screen. [Redundant since calling routine preserves AF]
--------	--	---

## Restore Cursor Position Attribute

L3AD3:	LD A,(IX+\$01)	A=Rows above the editing area (\$16 when using the lower screen, \$00 when using the main screen).
	ADD A,B	B=Row number within editing area.
	LD B,A	B=Screen row number.
	CALL L3BC1	Get address of attribute byte into HL.
	LD A,(IX+\$07)	Get previous attribute value.
	LD (HL),A	Set colour.
	RET	

## Shift Up Edit Rows in Display File

This routine shifts edit rows in the display file up, replacing the bottom row with the top entry from the Below-Screen Line Edit Buffer.

Entry: HL=Address of first row in the Below-Screen Line Edit Buffer.  
 E =Number of editing rows on screen.  
 B =Row number to shift from.

L3AE0:	PUSH HL	Save the address of the Below-Screen Line Edit Buffer row.
	LD H,\$00	Indicate to shift rows up.
	LD A,E	A=Number of editing rows on screen.
	SUB B	A=Number of rows to shift, i.e. from current row to end of edit screen.
	JR L3AEE	Jump ahead.

## Shift Down Edit Rows in Display File

This routine shifts edit rows in the display file down, replacing the top row with the bottom entry from the Above-Screen Line Edit Buffer.

Entry: HL=Address of next row to use within the Above-Screen Line Edit Buffer.  
 E =Number of editing rows on screen.  
 B =Row number to shift from.

L3AE7:	PUSH HL	Save the address of the first row in Below-Screen Line Edit Buffer.
	LD A,E	A=Number of editing rows on screen.
	LD E,B	E=Row number to shift from.
	LD B,A	B=Number of editing rows on screen.
	SUB E	A=Number of rows to shift, i.e. from current row to end of edit screen.
	LD H,\$FF	Indicate to shift rows down.

Shift Rows

L3AEE:	LD C,A	C=Number of rows to shift.
	LD A,B	A=Row number to shift from.
	CP E	Is it the final row of the editing screen?
	JR Z,L3B3E	Jump if so to simply display the row.

Shift all display file and attributes rows up

	PUSH DE	Save number of editing rows on screen, in E.
	CALL L3BB9	B=Inverted row number, i.e. 24-row number.
L3AF7:	PUSH BC	B=Inverted row number, C=Number of rows left to shift.
	LD C,H	Store the direction flag.
	RST 28H	
	DEFW CL_ADDR	\$0E9B. HL=Destination display file address, for the row number specified by 24-B.
	EX DE,HL	DE=Destination display file address.
	XOR A	
	OR C	Fetch the direction flag.
	JR Z,L3B04	Jump if moving up to the previous row.
	INC B	Move to the previous row (note that B is inverted, i.e. 24-row number).
	JR L3B05	Jump ahead.
L3B04:	DEC B	Move to the next row (note that B is inverted, i.e. 24-row number).

## SPECTRUM +2 ROM 0 DISASSEMBLY

L3B05:	PUSH DE RST 28H DEFW CL_ADDR POP DE	DE=Destination display file address.  \$0E9B. HL=Source display file address, for the row number held in B. DE=Destination display file address.
Copy one row of the display file		
L3B0F:	LD A,C LD C,\$20 LD B,\$08 PUSH BC PUSH HL PUSH DE LD B,\$00 LDIR POP DE POP HL POP BC INC H INC D DJNZ L3B0F	Fetch the direction flag. 32 columns. 8 lines.  Copy one line in the display file.  Next source line in the display file. Next destination line in the display file. Repeat for all lines in the row.
Copy one row of display attributes		
	PUSH AF PUSH DE RST 28H DEFW CL_ATTR EX DE,HL EX (SP),HL  RST 28H DEFW CL_ATTR EX DE,HL EX (SP),HL POP DE LD BC,\$0020 LDIR	Save the duration flag. DE=Address of next destination row in the display file. HL=Address of next source row in the display file. \$0E88. DE=Address of corresponding attribute cell. HL=Address of corresponding source attribute cell. Store source attribute cell on the stack, and fetch the next destination row in the display file in HL. HL=Address of next destination row in the display file. \$0E88. DE=Address of corresponding destination attribute cell. HL=Address of corresponding destination attribute cell. Store destination attribute cell on the stack, and fetch the source attribute cell in HL. DE=Destination attribute cell.  Copy one row of the attributes file.
Repeat to shift the next row		
	POP AF POP BC AND A JR Z,L3B37 INC B  JR L3B38 DEC B	Retrieve the direction flag. B=Inverted row number, C=Number of rows left to shift. Shifting up or down? Jump if shifting rows up. Move to the previous row, i.e. the row to copy (note that B is inverted, i.e. 24-row number). Jump ahead. Move to the next row, i.e. the row to copy (note that B is inverted, i.e. 24-row number).
L3B37:	DEC B	Decrement the row counter.
L3B38:	DEC C LD H,A JR NZ,L3AF7 POP DE LD B,E	H=Direction flag. Jump if back more rows to shift. E=Number of editing rows on screen. B=Number of editing rows on screen.
L3B3E:	POP HL	HL=Address of the Line Edit Buffer row to print (either in the Above-Screen Line Edit Buffer or in the Below-Screen Line Edit Buffer).

## Print a Row of the Edit Buffer to the Screen

This routine prints all 32 characters of a row in the edit buffer to the display file.

When shifting all rows up, this routine prints the top entry of the Below-Screen Line Edit Buffer to the first row of the display file.

When shifting all rows down, this routine prints the bottom entry of the Above-Screen Line Edit Buffer to the last editing row of the display file.

Entry: B =Row number to print at.

HL=Address of edit buffer row to print.

L3B3F:	CALL L3BD9 EX DE,HL LD A,(\$5C3C) PUSH AF LD HL,\$EC0D BIT 6,(HL) RES 0,A JR Z,L3B52 SET 0,A	Exchange colour items. Transfer address of edit buffer row to DE. TVFLAG.  Editor flags. Test the editing area flag. Allow leading space. Jump if editing area is the main screen. Suppress leading space.
L3B52:	LD (\$5C3C),A LD C,\$00 CALL L3751 EX DE,HL LD B,\$20	TVFLAG. The first column position of the edit row. Print AT. HL=Address of edit buffer row. 32 columns.
L3B5D:	LD A,(HL) AND A JR NZ,L3B63 LD A,\$20	Character present in this position?  Jump if character found. Display a space for a null character.
L3B63:	CP \$90 JR NC,L3B76 RST 28H DEFW PRINT_A_1	Is it a single character or UDG? Jump if it is a UDG. Print the character. \$0010.
L3B6A:	INC HL DJNZ L3B5D POP AF LD (\$5C3C),A CALL L3BD9 SCF RET	Repeat for all column positions. Restore original suppress leading space status. TVFLAG. Exchange colour items. [Redundant since never subsequently checked]
L3B76:	CALL L1F3F RST 10H CALL L1F64 JR L3B6A	Use Normal RAM Configuration (physical RAM bank 0). Print it (need to page in RAM bank 0 to allow access to UDGs). Use Workspace RAM configuration (physical RAM bank 7). Jump back for next character.

## Clear Display Rows

L3B7F:	CALL L3BD9 LD A,D SUB B INC A LD C,A CALL L3BB9	Exchange 48 and 128 editing colour items. Bottom row to clear.  A=Number of rows to clear. C=Number of rows to clear. B=Number of rows to end of screen.
--------	--	---

Clear display file row

L3B89:	PUSH BC RST 28H DEFW CL_ADDR LD C,\$08	B=Row number. C=Row to clear.  \$0E9B. Find display file address. 8 lines in the row.
L3B8F:	PUSH HL LD B,\$20 XOR A	Save start of row address. 32 columns.
L3B93:	LD (HL),A INC HL DJNZ L3B93 POP HL INC H DEC C JR NZ,L3B8F LD B,\$20 PUSH BC RST 28H	Blank the row.   Get start of row address. Next line.  Repeat for all rows. 32 columns.

## SPECTRUM +2 ROM o DISASSEMBLY

```
DEFW CL_ATTR
EX DE,HL
POP BC
```

\$0E88. Find attribute address.

BC=32 columns.

Reset display file attributes

```
L3BA7: LD A,($5C8D)
LD (HL),A
INC HL
DJNZ L3BA7
```

ATTR\_P.  
Set display file position attribute.

Repeat for all attributes in the row.

Repeat for next row

```
POP BC
DEC B
DEC C
JR NZ,L3B89
CALL L3BD9
SCF
RET
```

B=Row number. C=Number of rows to clear.

Repeat for all rows.  
Exchange 48 and 128 editing colour items.  
[Redundant since never subsequently checked]

## Find Rows and Columns to End of Screen

This routine calculates the number of rows to the end row of the screen and the number of columns to the end column of the screen. It takes into account the number of rows above the editing area.

Entry: B=Row number.  
C=Column number.

Exit : B=Number of rows to end row of screen.  
C=Number of columns to the end column of the screen.

```
L3BB5: LD A,$21
SUB C
LD C,A
```

Reverse column number.

C=33-C. Columns to end of screen.

## Find Rows to End of Screen

This routine calculates the number of rows to the end row of the screen. It takes into account the number of rows above the editing area.

Entry: B=Row number.  
Exit : B=Number of rows to end of screen.  
IX=Address of the cursor settings information.

```
L3BB9: LD A,$18
SUB B
SUB (IX+$01)
LD B,A
RET
```

Row 24.  
A=24-B.  
Subtract the number of rows above the editing area.  
B=Rows to end of screen.

## Get Attribute Address

Get the address of the attribute byte for the character position (B,C).

Entry: B=Row number.  
C=Column number.  
Exit : HL=Address of attribute byte.

```
L3BC1: PUSH BC
XOR A
LD D,B
LD E,A
RR D
RR E
```

Save BC.  
A=0.

DE=B\*256.

RR D	
RR E	
RR D	
RR E	DE=B*32.
LD HL,\$5800	Start of attributes file.
LD B,A	B=0.
ADD HL,BC	Add column offset.
ADD HL,DE	Add row offset.
POP BC	Restore BC.
RET	

## Exchange Colour Items

Exchange 128 Editor and main colour items.

L3BD9:	PUSH AF	Save registers.
	PUSH HL	
	PUSH DE	
	LD HL,(\$5C8D)	ATTR_P, MASK_P. Fetch main colour items.
	LD DE,(\$5C8F)	ATTR_T, MASK_T.
	EXX	Store them.
	LD HL,(\$EC0F)	Alternate Editor ATTR_P, MASK_P. Fetch alternate Editor colour items.
	LD DE,(\$EC11)	Alternate Editor ATTR_T, MASK_T.
	LD (\$5C8D),HL	ATTR_P, MASK_P. Store alternate Editor colour items as main colour items.
	LD (\$5C8F),DE	ATTR_T, MASK_T.
	EXX	Retrieve main colour items ATTR_T and MASK_T.
	LD (\$EC0F),HL	Alternate Editor ATTR_P, MASK_P.
	LD (\$EC11),DE	Alternate Editor ATTR_T, MASK_T. Store alternate Editor colour items as main colour items.
	LD HL,\$EC13	Alternate P_FLAG. Temporary Editor store for P_FLAG.
	LD A,(\$5C91)	P_FLAG.
	LD D,(HL)	Fetch alternate Editor version.
	LD (HL),A	Store main version in alternate Editor store.
	LD A,D	A=Alternate Editor version.
	LD (\$5C91),A	P_FLAG. Store it as main version.
	POP DE	Restore registers.
	POP HL	
	POP AF	
	RET	

## EDITOR ROUTINES — PART 5

### Tokenize BASIC Line

This routine serves two purposes. The first is to tokenize a typed BASIC line into a tokenized version. The second is when a syntax error is subsequently detected within the tokenized line, and it is then used to search for the position within the typed line where the error marker should be shown.

This routine parses the BASIC line entered by the user and generates a tokenized version in the workspace area as pointed to by system variable E\_LINE. It suffers from a number of bugs related to the handling of '>' and '<' characters. The keywords '<>', '>=' and '<=' are the only keywords that do not commence with letters and the routine traps these in a different manner to all other keywords. If a '<' or '>' is encountered then it is not immediately copied to the BASIC line workspace since the subsequent character must be examined as it could be a '>' or '<' character and therefore might form the keywords '<>', '>=' or '<='. A problem occurs if the subsequent character is a letter since the parser now expects the start of a possible keyword. It should at this point insert the '<' or '>' into the BASIC line workspace but neglects to do this. It is only when the next non-letter character is encountered that the '<' or '>' gets inserted, but this is now after the previously found string has been inserted. This results the following types of errors:

'PRINT varA>varB' is seen by the parser as 'PRINT varAvarB>' and hence a syntax error occurs.

'PRINT varA>varB1' is seen by the parser as 'PRINT varAvarB>1' and hence is accepted as a valid statement.

A work-around is to follow the '<' or '>' with a space since this forces the '<' or '>' to be inserted before the next potential keyword is examined.

A consequence of shifting a '<' or '>' is that a line such as 'PRINT a\$b\$b\$' is seen by the parser as 'PRINT a\$b\$b\$>' and so it throws a syntax error.

The parser saved the '>' character for consideration when the next character was examined to see if it was part of the keywords '<>', '>=' or '<=', but fails to discard it if the end of the statement is immediately encountered. Modifying the statement to a form that will be accepted will still cause a syntax error since the parser mistakenly believes the '>' character applies to this statement.

The parser identifies string literals contained within quotes and will not tokenize any keywords that appear inside them, except for the keywords "<>", "<=" and ">=" which it neglects to check for. Keywords are also not tokenized following a REM statement, except again for "<>", "<=" and ">=", until the end of the line is reached. This differs slightly to 48K BASIC mode. In 48K BASIC mode, typing a ':' following a REM statement will cause a change from 'L'

## SPECTRUM +2 ROM 0 DISASSEMBLY

cursor mode to 'K' cursor mode and hence the next key press results in a keyword token being inserted. In 128K BASIC mode, typing a ':' will not change to 'K' cursor mode and hence the next key press will just be the letter, number or symbol. This does not affect the running of the program since 48K BASIC mode will ignore all characters after a REM command until the end of the line. However, creating such a REM statement in 128K BASIC mode that appears similar to one created in 48K BASIC mode will result in more memory being used since the 'keyword' must be spelled out letter by letter. When being used to locate the error marker position, the same process is performed as when tokenizing but no characters are actually inserted into the workspace (they are still there from when the line was originally tokenized). Instead, a check is made after each character is processed to see if the error marker address held in system variable X\_PTR has been reached. If it does match then the routine returns with BC holding the character position where the error marker should be displayed at.

Entry point - A syntax error was detected so the error marker must be located

L3C0A:	LD A,\$01	Signal to locate the error marker.
	JR L3C10	Jump forward.

Entry point - Tokenize the BASIC line

L3C0E:	LD A,\$00	Signal to tokenize the BASIC line. [Could have saved 1 byte by using XOR A]
L3C10:	LD (\$FD8A),A	Store the 'locate error marker' flag.
	LD HL,\$0000	
	LD (\$FD85),HL	Reset count of the number of characters in the typed BASIC line being inserted.
	LD (\$FD87),HL	Reset count of the number of characters in the tokenized version of the BASIC line being inserted.
	ADD HL,SP	
	LD (\$FD8B),HL	Store the stack pointer.
	CALL L3510	Clear BASIC line construction pointers (address of next character in the Keyword Construction Buffer and the address of the next character in the BASIC line within the program area being de-tokenized).
	LD A,\$00	[Could have saved 1 byte by using XOR A]
	LD (\$FD84),A	Signal last character was not a keyword and was not a space. <b>[BUG - Should reset the '&lt;' and '&gt;' store at \$FD89 to \$00 here. Attempting to insert a BASIC line such as 'PRINT VAL a\$&gt;b' will fail since the parser does not like '&gt;' immediately after 'a\$', due to the bug at \$3C5F (ROM 0). The parser stores the '&gt;' in \$FD89 since it will check the following character in case it should replace the two characters with the token '&lt;&gt;', '&gt;=' or '&lt;='.</b> After the parser throws the syntax error, it does not clear \$FD89 and so even if the line is modified such that it should be accepted, e.g. 'PRINT VAL a\$=b', the parser believes the line is really '>PRINT VAL n\$=b' and so throws another syntax error. Since a letter follows the '>', the contents of \$FD89 will get cleared and hence a second attempt to insert the line will now succeed. Credit: Paul Farrow]
	LD HL,\$FD74	HL=Start address of the Keyword Conversion Buffer.
	LD (\$FD7D),HL	Store as the next available location.
	CALL L1F3F	Use Normal RAM Configuration (physical RAM bank 0).
	RST 28H	
	DEFW SET_MIN	\$16B0. Clear the editing areas.
	CALL L1F64	Use Workspace RAM configuration (physical RAM bank 7).
	LD A,\$00	[Could have saved 1 byte by using XOR A, or 2 bytes by clearing this above]
	LD (\$FD81),A	Clear Keyword Conversion Buffer flags - not within REM, not with Quotes, no characters in the buffer.
	LD HL,(\$5C59)	E_LINE.
	LD (\$FD82),HL	Store the address of the workspace for the tokenized BASIC line.
	LD HL,\$0000	[Could have saved 1 byte by using LD H,A followed by LD L,A]
	LD (\$FD7F),HL	Signal no space character between words in the Keyword Conversion Buffer.

Enter a loop to fetch each character from the BASIC line and insert it into the workspace, tokenizing along the way

L3C48:	LD HL,(\$FD85)	
	INC HL	Increment count of the number of characters in the typed BASIC line.
	LD (\$FD85),HL	
	CALL L3D44	Fetch the next character from BASIC line being inserted, return in B.
	LD C,A	Save the character status value.

C=\$01 if not a space, not a letter, not a '#' and not a '\$'.

\$02 if a '#' or '\$'.

\$03 if a space.

\$06 if a letter.

B=Character fetched.

## SPECTRUM +2 ROM 0 DISASSEMBLY

LD A,(\$FD81)	Have any Keyword Conversion Buffer flags been set?
CP \$00	Has anything be put into the buffer yet?
JR NZ,L3C9B	Jump if so.

The first character to potentially put into the Keyword Conversion Buffer

L3C5A: LD A,C	Retrieve the character status value.
AND \$04	Is the character a letter?
JR Z,L3C94	Jump if not.

Insert the character

L3C5F:

**[BUG** - At this point a '>' or '<' that was previously stored should be inserted into the BASIC line workspace. However, the routine proceeds with the new potential keyword and this is entered into the BASIC line workspace next. The '>' or '<' will only be inserted when the next non-letter character is encountered. This causes an expression such as 'a>b1' to be translated into 'ab>1'. Credit: Ian Collier (+3), Paul Farrow (128)] [The bug can be fixed by testing if whether a '<' or '>' character is stored. Credit: Paul Farrow.

LD A,(\$FD89)	<i>Was the last character a '&gt;' or '&lt;'?</i>
AND A	<i>Jump if not.</i>
JR Z,INSERT	<i>Save the new character.</i>
PUSH BC	
LD B,A	
CALL \$3E0B (ROM 0)	<i>Insert the '&gt;' or '&lt;' into the BASIC line workspace.</i>
POP BC	<i>Retrieve the new character.</i>
XOR A	
LD (\$FD89),A	<i>Clear the '&gt;' or '&lt;'.</i>
<b>INSERT</b>	

CALL L3D90	Insert the character into the Keyword Conversion Buffer.
JR NC,L3C6B	Jump if no more room within the buffer, hence string is too large to be a token.
LD A,\$01	Signal Keyword Conversion Buffer contains characters.
LD (\$FD81),A	
JR L3C48	Jump back to fetch and process the next character.

No room to insert the character into the Keyword Conversion Buffer hence string is too large to be a valid token

L3C6B: LD HL,(\$FD7F)	Fetch the address of the space character between words within the Keyword Conversion Buffer.
LD A,L	
OR H	Is there an address set?
JP NZ,L3CC5	Jump if so to copy the first word into the BASIC line workspace and the move the second word to the start of the Keyword Conversion Buffer. Further characters can then be appended and the contents re-evaluated in case a complete keyword is then available.

Copy the Keyword Conversion Buffer into the BASIC line workspace

L3C73: PUSH BC	Save the character to insert.
CALL L3D74	Copy Keyword Conversion Buffer contents into BASIC line workspace.
POP BC	Retrieve the character to insert.
LD A,\$00	
LD (\$FD81),A	Signal the Keyword Conversion Buffer is empty.

C=\$01 if not a space, not a letter, not a '#' and not a '\$'.

\$02 if a '#' or '\$'.

\$03 if a space.

\$06 if a letter.

B=Character fetched.

L3C7D: LD A,C	Retrieve the character status value.
AND \$01	Is it a space, or not a letter and not a '#' and not a '\$'?

## SPECTRUM +2 ROM o DISASSEMBLY

JR NZ,L3C5A	Jump back if so to insert the character either into the Keyword Conversion Buffer or the BASIC line workspace.
-------------	--

The string was too long to be a keyword and was followed by a space, a '#' or a '\$'. Enter a loop to insert each character of the string into the BASIC line workspace.

LD A,B	Retrieve the character to insert.
CALL L3DBD	Insert character into BASIC line workspace.
RET NC	Return if tokenizing is complete.
LD HL,(\$FD85)	
INC HL	Increment the count of the number of characters in the typed BASIC line being inserted.
LD (\$FD85),HL	
CALL L3D44	Fetch the next character from BASIC line being inserted.
LD C,A	Save the flags.
JR L3C7D	Jump back to insert the character of the non-keyword string into the BASIC line workspace.

The character is not a letter so insert directly into the BASIC line workspace

L3C94:	LD A,B	Retrieve the character to insert.
	CALL L3DBD	Insert character into BASIC line workspace, tokenizing '<>', '<=' and '>=' if encountered.
	RET NC	Return if tokenizing is complete.
	JR L3C48	Jump back to fetch and process the next character.

Keyword Conversion buffer flags are set - either the buffer already contains characters, or within quotes or within a REM statement

L3C9B:	CP \$01	Is the Keyword Conversion Buffer empty or the contents marked as being within quotes or within a REM?
	JR NZ,L3C94	Jump back if so to insert the character since this is either the first character of a new word or is within quotes or within a REM.

C=\$01 if not a space, not a letter, not a '#' and not a '\$'.  
 \$02 if a '#' or '\$'.  
 \$03 if a space.  
 \$06 if a letter.

LD A,C	Retrieve the character status value.
AND \$01	Is it a letter or a '#' or a '\$'?
JR Z,L3C5F	Jump if so to simply insert the character.

The character is a space, or is not a letter and not a '#' and not a '\$', i.e. the last character was the end of a potential keyword

L3CA5:	PUSH BC	Save the next character to insert and the character status value.
	CALL L3F25	Attempt to identify the string in Keyword Conversion Buffer.
	POP BC	Retrieve the next character to insert and the character status value.
	JR C,L3D24	Jump if keyword identified.

The string in the Keyword Conversion Buffer was not identified as a keyword

LD HL,(\$FD7F)	Fetch the address of the space character between words within the Keyword Conversion Buffer.
LD A,H	
OR L	Is there an address set, i.e. a space between words?
JR NZ,L3CC5	Jump if there is a space character.
LD A,C	Retrieve the character status value.
AND \$02	Is it a space?
JR Z,L3C73	Jump if not to copy Keyword Conversion Buffer into the workspace since it is not a keyword.

Character is a space. Allow this as the keyword could be DEF FN, GO TO, GO SUB, etc.

CALL L3D90	Insert the character into the Keyword Conversion Buffer.
------------	--

## SPECTRUM +2 ROM o DISASSEMBLY

JR NC,L3C6B	Jump back if no room to insert the character, i.e. not a keyword since too large.
LD HL,(\$FD7D)	Fetch the next location address.
DEC HL	Point back to the last character.
LD (\$FD7F),HL	Store as the address of the space character. This is used for double keywords such as DEF FN.
JR L3C48	Jump back to fetch and process the next character.

The string in the Keyword Conversion Buffer contains two words separated by a space that do not form a valid double keyword (such as DEF FN, GO SUB, GO TO, etc).

For a BASIC line such as 'IF FLAG THEN' the Keyword Conversion Buffer holds the characters 'FLAG THEN'.

The 'FLAG' characters get moved to the workspace and the 'THEN' characters are shifted to the start of the Keyword Conversion Buffer before being re-evaluated to see if they form a keyword.

L3CC5:	PUSH BC	Save the character to insert and the character status value.
	LD HL,\$FD74	Point to the start address of the Keyword Conversion Buffer.
	LD DE,(\$FD7F)	Fetch the address of the space character between words within the Keyword Conversion Buffer.
	LD A,D	
	CP H	Is the space possibly at the start of the buffer?
	JR NZ,L3CD6	Jump if not.
	LD A,E	
	CP L	Is the space at the start of the buffer?
	JR NZ,L3CD6	Jump if not.
	INC DE	Point to the next location within the buffer, counter-acting the following decrement.
L3CD6:	DEC DE	Point to the previous location within the buffer.
	JR L3CDA	Jump ahead to copy all characters to the BASIC line workspace.

Copy all characters from the Keyword Conversion Buffer prior to the space into the BASIC line workspace

L3CD9:	INC HL	Point to the next location within the Keyword Conversion Buffer.
L3CDA:	LD A,(HL)	Fetch a character from the Keyword Conversion Buffer.
	AND \$7F	Mask off the terminator bit.
	PUSH HL	HL=Location within Keyword Conversion Buffer.
	PUSH DE	DE=Location of last character within the Keyword conversion Buffer.
	CALL L3DBD	Insert character into BASIC line workspace, including a stored '<' or '>' character.
	POP DE	
	POP HL	
	LD A,H	
	CP D	Possibly reached the character prior to the space?
	JR NZ,L3CD9	Jump back if not to copy the next character.
	LD A,L	
	CP E	Reached the character prior to the space?
	JR NZ,L3CD9	Jump back if not to copy the next character.

Now proceed to handle the next word

LD DE,(\$FD7F)	DE=Address of the space character between words.
LD HL,\$FD74	
LD (\$FD7F),HL	Set the address of the space character to be the start of the buffer.
LD BC,(\$FD7D)	BC=Next location within the Keyword Conversion Buffer.
DEC BC	Point to the last used location.
LD A,D	
CP H	Is the space possibly at the start of the buffer?
JR NZ,L3D17	Jump if not.
LD A,E	
CP L	Is the space at the start of the buffer?
JR NZ,L3D17	Jump if not.

The space character is at the start of the Keyword Conversion Buffer

INC DE	DE=Address after the space character within the Keyword Conversion Buffer.
PUSH HL	HL=Start address of the Keyword Conversion Buffer.
LD HL,\$0000	
LD (\$FD7F),HL	Signal no space character between words.
POP HL	HL=Start address of the Keyword Conversion Buffer.

## SPECTRUM +2 ROM o DISASSEMBLY

LD A,B	
CP H	Is the space possibly the last character in the buffer?
JR NZ,L3D17	Jump if not.
LD A,C	
CP L	Is the space the last character in the buffer?
JR NZ,L3D17	Jump if not.
POP BC	Retrieve the character to insert and the character status value.
JR L3D36	Jump ahead to continue.

The space is not at the start of the Keyword Conversion Buffer, i.e. the buffer contains another word after the space.

The first word has already been copied to the BASIC line workspace so now copy the second word to the start of the Keyword Conversion Buffer and then see if it is a valid keyword. [It is not recommended to name a variable as per a keyword since statements such as 'PRINT then' will fail the syntax check since the variable 'then' is interpreted as the keyword 'THEN' and so the statement is seen as 'PRINT THEN', which in this case is invalid.] HL points to the start of the Keyword Conversion Buffer. DE points to the space between the two words.

L3D17:	LD A,(DE)	Fetch a character from the second word.
	LD (HL),A	Store it at the beginning of the buffer.
	INC HL	
	INC DE	
	AND \$80	Reached the last character in the buffer, i.e. the terminator bit set?
	JR Z,L3D17	Jump if not to copy the next character.
	LD (\$FD7D),HL	Store the new address of the next free location.
	JR L3CA5	Jump back to attempt identification of the 'second' word as a keyword.

The string in the Keyword Conversion Buffer was identified as a keyword, so insert the token character code of the keyword into the BASIC line workspace.  
A=Character code of identified token.

L3D24:	PUSH BC	Save the next character to insert and the character status value.
	CALL L3DBD	Insert character held in A into BASIC line workspace.
	POP BC	Retrieve the next character to insert and the character status value.

The token has been inserted into the BASIC line workspace so reset the Keyword Conversion Buffer

	LD HL,\$0000	
	LD (\$FD7F),HL	Indicate no space character between words in the Keyword Conversion Buffer.
	LD A,(\$FD81)	Fetch the flag bits.
	CP \$04	Within a REM statement?
	JR Z,L3D3B	Jump if so to retain the 'within a REM' flag bit.
L3D36:	LD A,\$00	
	LD (\$FD81),A	Signal no characters within the Keyword Conversion Buffer.
L3D3B:	LD HL,\$FD74	Start address of the Keyword Conversion Buffer.
	LD (\$FD7D),HL	Store this as the next location within the buffer.
	JP L3C5A	Jump back to insert the next character either into the Keyword Conversion Buffer or the BASIC line workspace.

## Fetch Next Character and Character Status from BASIC Line to Insert

Fetch the next character from the BASIC line being inserted and check whether a letter, a space, a '#' or a '\$'.

Exit: B=Character.

A=\$01 if not a space, not a letter, not a '#' and not a '\$'.

\$02 if a '#' or '\$'.

\$03 if a space.

\$06 if a letter.

L3D44:	CALL L2D7A	Fetch the next character from the BASIC line being inserted.
	LD B,A	Save the character.
	CP '?'	\$3F. Is it below '?' (the error marker)?
	JR C,L3D56	Jump if so.
	OR \$20	Make lowercase.
	CALL L3D6D	Is it a letter?
	JR C,L3D6A	Jump if so.
L3D53:	LD A,\$01	Indicate not space, not letter, not '#' and not '\$'.
	RET	
L3D56:	CP \$20	Is it a space?

	JR Z,L3D67	Jump if so.
	CP '#'	\$23. Is it '#'?
	JR Z,L3D64	Jump if so.
	JR C,L3D53	Jump if below '#'.
	CP '\$'	\$24. Is it '\$'?
	JR NZ,L3D53	Jump if not.
L3D64:	LD A,\$02	Indicate a '#' or '\$'.
	RET	
L3D67:	LD A,\$03	Indicate a space.
	RET	
L3D6A:	LD A,\$06	Indicate a letter.
	RET	

## Is Lowercase Letter?

L3D6D:	CP \$7B	Is the character above 'z'?
	RET NC	Return with carry flag reset if above 'z'.
	CP \$61	Is the character below 'a'?
	CCF	Return with carry flag reset if below 'a'.
	RET	

## Copy Keyword Conversion Buffer Contents into BASIC Line Workspace

L3D74:

[To fix the error marker bug at \$3EA2 (ROM 0), the code below up until the instruction at \$3D81 (ROM 0) should have been as follows]

<i>LD HL,\$FD74</i>	<i>Start address of the Keyword Conversion Buffer.</i>
<i>CALL \$3D81 (ROM 0)</i>	<i>Copy all characters into the BASIC line workspace.</i>
<i>LD HL,\$FD74</i>	<i>Start address of the Keyword Conversion Buffer.</i>
<i>LD (\$FD7D),HL</i>	<i>Store the next available location.</i>
<i>SUB A</i>	<i>A=0.</i>
<i>LD (\$FD7F),A</i>	
<i>LD (\$FD80),A</i>	<i>Signal no space character between words in the Keyword Conversion Buffer.</i>
<i>RET</i>	

	LD HL,\$FD74	Start address of the Keyword Conversion Buffer.
	LD (\$FD7D),HL	Store the next available location.
	SUB A	A=0.
	LD (\$FD7F),A	
	LD (\$FD80),A	Signal no space character between words in the Keyword Conversion Buffer.
L3D81:	LD A,(HL)	Fetch a character from the buffer.
	AND \$7F	Mask off the terminator bit.
	PUSH HL	Save buffer location.
	CALL L3E43	Insert the character into the BASIC line workspace, suppressing spaces as required.
	POP HL	Retrieve buffer location.
	LD A,(HL)	Re-fetch the character from the buffer.
	AND \$80	Is it the terminator character?
	RET NZ	Return if so.
	INC HL	Point to the next character in the buffer.
	JR L3D81	Jump back to handle next buffer character.

## Insert Character into Keyword Conversion Buffer

Entry; B=Character to insert.

Exit : Carry flag reset if no room to insert the character within the buffer.

L3D90:	LD HL,(\$FD7D)	Fetch address within Keyword Conversion Buffer.
	LD DE,\$FD7D	Address after Keyword Conversion Buffer.
	LD A,D	

## SPECTRUM +2 ROM o DISASSEMBLY

CP H JR NZ,L3D9F LD A,E CP L JP Z,L3DBA	Has end of buffer possibly been reached? Jump if not.  Has end of buffer been reached? Jump if so. [Could have saved a byte by using JR instead of JP]
---	--

End of buffer not reached

L3D9F:      LD DE,\$FD74 LD A,D CP H JR NZ,L3DAA LD A,E CP L JR Z,L3DB0	Start address of Keyword Conversion Buffer.  Possibly at the start of the buffer? Jump if not.  At the start of the buffer? Jump if so to simply store the character.
---	---

Not at the start of the buffer so need to remove terminator bit from the previous character

L3DAA:      DEC HL LD A,(HL) AND \$7F LD (HL),A INC HL L3DB0:      LD A,B OR \$80 LD (HL),A INC HL LD (\$FD7D),HL SCF RET	Point to the last character.  Clear the terminator bit from the last character.  Point back at the current location. Retrieve the new character. Set the terminator bit. Store the character in the buffer. Point to the next location. Store the address of the next location. Signal character inserted.
--	--

End of buffer reached

L3DBA:      SCF CCF RET	Clear the carry flag to indicate no room to insert the character within the buffer.
-------------------------------	---

### Insert Character into BASIC Line Workspace, Handling '>' and '<'

This routine inserts a character into the BASIC line workspace, tokenizing '>=', '<=' and '<>'.

Entry:      A=Character to insert.

Exit :      If tokenizing a BASIC line then returns with carry flag reset if tokenizing is complete.  
           If searching for the error marker location then returns with the carry flag set if the error marker has not been found,  
           otherwise a return is made to the main calling routine with BC holding the number of characters in the typed BASIC line,  
           i.e. the error marker location is at the end of the line.

L3DBD:      PUSH AF	Save the character to insert.
---------------------	-------------------------------

**[BUG** - The string characters "<>", "<=" and ">=" get tokenized to a single character '<>', '<=' and '>=' respectively even within quotes or a REM statement. Credit: Paul Collins (+3), Paul Farrow (128)] **[BUG** - 128 BASIC mode handles a colon character found following a REM statement differently to 48K mode. In 48K mode, typing a colon returns the cursor into 'K' mode and hence the next key press inserts a keyword token. In 128K mode, typing a colon does not cause the characters following it to be interpreted as a possible keyword. There is no noticeable difference when executing the REM statement since subsequent statements are ignored following a REM command. However, for consistency the 128K mode editor ought to generate identical BASIC lines to those that would be created from 48K mode. Credit: Paul Farrow] [The following instructions would be required fix the two bugs described above. Credit: Paul Farrow.

LD A,(\$FD81) BIT 1,A JR NZ,WITHIN BIT 2,A JR Z,NOT_WITHIN POP AF PUSH AF	Within quotes? Jump forward if within quotes. Within a REM statement? Jump forward if not within a REM statement.
---	--

WITHIN	CP ':'	
	JR NZ,WITHIN	Jump if not a colon.
	LD A,(\$FD81)	
	AND \$FB	Signal not within a REM statement.
	LD (\$FD81),A	
	POP AF	Retrieve the character to insert.
NOT_WITHIN	JP \$3E0B (ROM 0)	Simply insert the character into the BASIC line workspace.

L3DCD:	LD A,(\$FD89)	Was the previous character '<' or '>'?
	OR A	
	JR NZ,L3DD6	Jump if so.
	POP AF	Retrieve the character to insert.
	CP '>'	\$3E. Is it '>'?
	JR Z,L3DD1	Jump if so to store for special treatment later.
	CP '<'	\$3C. Is it '<'?
	JR Z,L3DD1	Jump if so to store for special treatment later.
	CALL L3E0B	Insert the character into the BASIC line workspace.
	RET	[Could have saved 1 byte by using JP \$3E0B (ROM 0)]

The character was '<' or '>'

L3DD1:	LD (\$FD89),A	Store '<' or '>'.
	SCF	Signal tokenizing not complete or error marker not found.
	RET	

The previous character was '<' or '>'

L3DD6:	CP '<'	\$3C. Was the previous character '<'?
	LD A,\$00	Reset the indicator that the previous
	LD (\$FD89),A	character was '<' or '>'.
	JR NZ,L3DF9	Jump ahead if not '<'.

Previous character was '<'

L3DE8:	POP AF	Retrieve the character to insert.
	CP '>'	\$3E. Is it '>'?
	JR NZ,L3DE8	Jump ahead if not.
	LD A,\$C9	Tokenize to the single character '<>'.
	JR L3DCD	Jump back to insert the character and return.
	CP '='	\$3D. Is it '='?
	JR NZ,L3DF0	Jump ahead if not.
	LD A,\$C7	Tokenize to '<='.
	JR L3DCD	Jump back to insert the character and return.

Previous character was '<' and new character is '<'

L3DF0:	PUSH AF	Save the current character to insert.
	LD A,'<'	\$3C.
	CALL L3E0B	Put the preceding '<' character into the line.
	POP AF	Retrieve the character to insert.
	JR L3DCD	Jump back to insert the character and return.

Previous character was '>'

L3DF9:	POP AF	Retrieve the character to insert.
	CP '='	\$3D. Is it '='?
	JR NZ,L3E02	Jump ahead if not.
	LD A,\$C8	Tokenize to '>='.
	JR L3DCD	Jump back to insert the character and return.

Previous character was '>' and new character is '>'

L3E02:	PUSH AF LD A,'> CALL L3E0B POP AF JR L3DCD	Save the current character to insert. \$3E. Put the preceding '>' character into the line. Retrieve the character to insert. Jump back to insert the character and return.
--------	--	--

## Insert Character into BASIC Line Workspace, Handling 'REM' and Quotes

This routine inserts a character into the BASIC line workspace, with special handling of a 'REM' command and strings contained within quotes.

Entry: A=Character to insert.

Exit : If tokenizing a BASIC line then returns with carry flag reset if tokenizing is complete.

If searching for the error marker location then returns with the carry flag set if the error marker has not been found, otherwise a return is made directly to the main calling routine with BC holding the number of characters in the typed BASIC line, i.e. the error marker location is at the end of the line.

L3E0B:	CP \$0D JR Z,L3E2F CP \$EA LD B,A JR NZ,L3E1B	Is it 'ENTER'? Jump ahead if so. Is it 'REM'? Save the character. Jump ahead if not REM.
--------	---	--

It is a 'REM' character

L3E1B:	LD A,\$04 LD (\$FD81),A JR L3E29 CP \$22 JR NZ,L3E29	Indicate that within a REM statement.  Jump ahead to insert the character into the BASIC line workspace. Is it a quote? Jump ahead if not.
--------	--	--

It is a quote character

L3E29:	LD A,(\$FD81) AND \$FE XOR \$02  LD (\$FD81),A LD A,B CALL L3E43 SCF RET	Signal last character was not a keyword. Toggle the 'within quotes' flag. Will be 1 for an opening quote, then 0 for a closing quote.  Retrieve the character. Insert the character into the BASIC line workspace, suppressing spaces as required. Indicate BASIC line tokenization not complete.
--------	--	--

It is an 'ENTER' character

**[BUG]** - At this point a check should be made to see whether the last character was a space. If it was then it will not have been inserted but instead the flag in \$FD84 (ROM 0) will have been set. The purpose of the flag is to filter out double spaces caused by the leading/trailing spaces of tokens. Only if the following character is not a space will the previous character, the space, be inserted. When the end of the line is found, there is no attempt to insert this space. The bug can be fixed by the two modifications shown below. Credit: Paul Farrow]

L3E2F:	LD A,(\$FD8A) CP \$00  JR Z,L3E40	Fetch the 'locate error marker' flag. Searching for the error marker following a syntax error? [Could have saved 1 byte by using AND A] Jump if tokenizing the BASIC line.
--------	--	--

The end of the line was reached and no error marker was found so assume the error marker exists at the end of the typed line

LD BC,(\$FD85) LD HL,(\$FD8B)	BC=Count of number of the characters in the typed BASIC line being inserted.
----------------------------------	--

*[The first part of the fix for the trailing space bug is as follows:*

*LD A,(\$FD84)  
AND \$02  
JR Z,GOT\_COUNT  
INC BC*

*Fetch the BASIC line insertion flags.  
Was the last character a space?  
Jump if not.  
Increment to account for the final space.*

## GOT\_COUNT

LD SP,HL	Restore the stack pointer.
SCF	Indicate the error marker was not found within the tokenized BASIC line.
RET	Return back to the top level calling routine, to \$2D2A (ROM 0).

Tokenizing the BASIC line

L3E40:

<i>[The second part of the fix for the trailing space bug is as follows:</i>	
LD A,(\$FD84)	<i>Fetch the BASIC line insertion flags.</i>
AND \$02	<i>Was the last character a space?</i>
LD A,\$20	<i>Insert a space into the line.</i>
CALL NZ,\$3EA2 (ROM 0)	<i>If so then insert the character into the BASIC line workspace.]</i>

SCF	
CCF	Carry flag reset to indicate tokenizing complete.
RET	

## Insert Character into BASIC Line Workspace With Space Suppression

This routine is called to insert a character into the BASIC line workspace, suppressing both leading and trailing spaces around tokens, e.g. 'PRINT 10' does not require a space stored between 'PRINT' and '10' within the BASIC line.

The routine maintains two flags which indicate whether the last character was a space or was a token. Whenever a space is encountered, it is noted but not inserted straight away. It is only after the subsequent character is examined that the routine can determine whether the space should or should not be inserted.

Entry: A=Character to insert.

Exit : A=Updated BASIC line insertion flags.

L3E43:	LD E,A	Save the character to insert in E.
	LD A,(\$FD84)	
	LD D,A	D=BASIC line insertion flags.
	LD A,E	Restore character to insert back to A.
	CP \$20	Is it a space?
	JR NZ,L3E6D	Jump ahead if not.

Character to insert is a space

LD A,D	A=BASIC line insertion flags.
AND \$01	Was the last character a token?
JR NZ,L3E66	Jump ahead if so.
LD A,D	A=BASIC line insertion flags.
AND \$02	Was the last character a space?
JR NZ,L3E5E	Jump ahead if so.

Character to insert is a space and the last character was not a space/token. This could be the start of a new keyword so note the space but do not insert it now.

LD A,D	A=BASIC line insertion flags.
OR \$02	Signal the last character was a space.
LD (\$FD84),A	Store the updated BASIC line insertion flags.
RET	

Character to insert is a space and the last character was a space. The new space could be the start of a new keyword so keep the 'last character was a space' flag set but insert a space for the previous space that was noted.

L3E5E:	LD A,E	Retrieve the character to insert.
	CALL L3EA2	Insert the character into the BASIC line workspace.
	LD A,(\$FD84)	A=BASIC line insertion flags.
	RET	

## SPECTRUM +2 ROM 0 DISASSEMBLY

Character to insert is a space and the last character was a token. Do not insert trailing spaces for tokens.

L3E66:	LD A,D	A=BASIC line insertion flags.
	AND \$FE	Signal last character was not a token.
	LD (\$FD84),A	Store the updated BASIC line insertion flags.
	RET	[Could have saved 2 bytes by using JR \$3E5A (ROM 0)]

Character to insert is not a space

L3E6D:	CP \$A3	Compare against the token 'SPECTRUM' (the first 128K keyword).
	JR NC,L3E95	Jump ahead if a token.

Character to insert is not a space and not a token

LD A,D	A=BASIC line insertion flags.
AND \$02	Was the last character a space?
JR NZ,L3E81	Jump ahead if it was.

Character to insert is not a space and not a token and the last character inserted was not a space, so just insert the character

LD A,D	A=BASIC line insertion flags.
AND \$FE	Signal last character was not a keyword.
LD (\$FD84),A	Store the new flags.
LD A,E	Retrieve the character to insert.
CALL L3EA2	Insert the character into the BASIC line workspace.
RET	[Could have saved one byte by using JP \$3EA2 (ROM 0)]

Character to insert is not a space and not a token and the last character was a space. Since the new character is not a token, the previous space was not the start of a new keyword so insert a space and then the new character.

L3E81:	PUSH DE	Save the BASIC line insertion flags.
	LD A,\$20	Insert a space into the line.
	CALL L3EA2	Insert the character into the BASIC line workspace.
	POP DE	Retrieve the flags.
	LD A,D	A=BASIC line insertion flags.
	AND \$FE	Signal last character was not a keyword.
	AND \$FD	Signal last character was not a space.
	LD (\$FD84),A	Store the updated BASIC line insertion flags. [Could have saved 6 bytes by using JR \$3E79 (ROM 0)]
	LD A,E	Retrieve the character to insert.
	CALL L3EA2	Insert the character into the BASIC line workspace.
	RET	

Character to insert is a token. Clear any previously noted space since leading spaces are not required for tokens.

L3E95:	LD A,D	A=BASIC line insertion flags.
	AND \$FD	Signal last character was not a space.
	OR \$01	Signal last character was a keyword.
	LD (\$FD84),A	Store the updated BASIC line insertion flags. [Could have saved 6 bytes by using JR \$3E79 (ROM 0)]
	LD A,E	Retrieve the character to insert.
	CALL L3EA2	Insert the character into the BASIC line workspace.
	RET	

## Insert a Character into BASIC Line Workspace

This routine is called for two purposes. The first use is for inserting a character or token into the BASIC line workspace (situated at E\_LINE).

The second use is after a syntax error has been identified within the tokenized BASIC line in the workspace and the location of the error marker needs to be established. For the second case, the system variable X\_PTR holds the address of where the error occurred within the tokenized BASIC line in the workspace.

The Editor needs to identify how many characters there are before the equivalent error position is reached within the typed BASIC line. To locate it, the typed BASIC line is re-parsed but this time without inserting any characters into the BASIC line workspace, since this still contains the tokenized line from before. This tokenized line will now also include embedded floating point numbers for any numeric literals contained within the BASIC line. As the

## SPECTRUM +2 ROM 0 DISASSEMBLY

typed line is re-parsed, a count of the characters examined so far is kept and instead of inserting tokenized characters within the BASIC line workspace, a check is made to see whether the insertion location has reached the address of the error marker. If it has then the parsing of the BASIC line terminates and the count of the typed line characters indicates the equivalent position within it of the error. However, should the last character have been a token then the typed line count will also include the number of characters that form the keyword, and so this must be subtracted from the count.

Entry:     A=Character to insert.

          DE=Address of insertion position within the BASIC line workspace.

Exit :     If searching for the error marker position and it is found then a return is made directly to the top level calling routine with BC holding the number of characters in

the typed BASIC line prior to the equivalent error marker position.

L3EA2:	LD HL,(\$FD87) INC HL LD (\$FD87),HL LD HL,(\$FD82) LD B,A LD A,(\$FD8A) CP \$00  LD A,B JR Z,L3EDA	Increment the count of the number of characters in the tokenized BASIC line.  HL=Address of next insertion position in the BASIC line workspace. Save the character to insert. Fetch the 'locate error marker' flag. Searching for the error marker following a syntax error? [Could have saved 1 byte by using AND A] A=Character to insert. Jump if tokenizing the BASIC line.
--------	--	---

Locating the error marker

LD DE,(\$5C5F) LD A,H CP D JR NZ,L3ED7 LD A,L CP E JR NZ,L3ED7	X_PTR. Fetch the address of the character after the error marker.  Has the error marker position possibly been reached? Jump ahead if not.  Has the error marker position been reached? Jump ahead if not.
--	--

The error marker has been reached

**[BUG]** - The desired character count until the error marker is held at address \$FD85 and needs the length of the last character to be removed from it, which for a token would be several bytes. However, the routine simply returns the lower of the tokenized and typed counts, and this yields very unhelpful error marker positions shown within the typed BASIC line. Credit: Ian Collier (+3), Andrew Owen (128)] [The code below up until the instruction at \$3ED1 (ROM 0) should have been as follows. Changes to the code at \$3D74 (ROM 0) are also required. Credit: Paul Farrow.

LD HL,(\$FD7D) LD DE,\$FD74 AND A SBC HL,DE EX DE,HL LD HL,(\$FD85)  SBC HL,DE LD B,H LD C,L	Fetch the next address within the Keyword Conversion Buffer. Fetch the start address of the Keyword Conversion Buffer.  HL=Length of the keyword (excluding leading or trailing spaces). DE=Length of the keyword (excluding leading or trailing spaces). BC=Count of the number of characters in the typed BASIC line until the error marker location was found. Subtract the number of characters in the keyword text.  Transfer the result to BC, and then return via the instructions at \$3ED1 (ROM 0) onwards.]
---	---

LD BC,(\$FD85)  LD HL,(\$FD87)  AND A SBC HL,BC JR NC,L3ED1 LD BC,(\$FD87)	Count of the number of characters in the typed BASIC line until the error marker location was found.  Count of the number of characters in the tokenized BASIC line until the error marker location.  Jump if the tokenized version is longer than the typed version. Count of the number of characters in the tokenized version of the BASIC line until the error marker location.
---	--

L3ED1:	LD HL,(\$FD8B) LD SP,HL SCF RET	Fetch the saved stack pointer. Restore the stack pointer. Set the carry flag to indicate the error marker has been located. Return back to the top level calling routine, to \$2D2A (ROM 0).
--------	--	---

The error marker has not yet been reached

## SPECTRUM +2 ROM o DISASSEMBLY

L3ED7: SCF Set the carry flag to indicate error marker locating mode.  
JR L3EDC Jump ahead to continue.

Tokenizing the BASIC line

L3EDA: SCF Reset carry flag to signal BASIC line tokenizing mode.  
CCF Use Normal RAM Configuration (physical RAM bank 0).  
L3EDC: CALL L1F3F Jump if tokenizing the BASIC line.  
JR NC,L3EEE

Searching for the error marker so need to consider embedded floating point numbers

**[BUG]** - This should fetch the next character from the tokenized BASIC line and not the current character. This routine is called to process every visible character in the BASIC line, but is not called for embedded floating point numbers. It must therefore test whether the current character is followed by an embedded floating point number and if so to skip over it. The routine does make an attempt to detect embedded floating point numbers but incorrectly performs the test on the visible character and not the character that follows it. The bug can be fixed as replacing the LD A,(HL) instruction with the following instructions. Credit: Paul Farrow.

INC HL	Advance to the next character in the tokenized BASIC line.
LD A,(HL)	Fetch the next character in the tokenized BASIC line.
DEC HL	Point back to the current character in the tokenized BASIC line.]
LD A,(HL)	Fetch the current character in the tokenized BASIC line.
EX DE,HL	DE=Insert position within the tokenized BASIC line.
CP \$0E	Is it the 'number' marker?
JR NZ,L3F04	Jump ahead if not.
INC DE	Skip over the 5 byte hidden number representation.
INC DE	<b>[BUG]</b> - There should be another INC DE instruction here to take into account the character that the tokenizer would
INC DE	have inserted. As a result, the attempt to locate the error marker location will drift off
INC DE	by one byte for every numeric
INC DE	literal within the BASIC statement, and if there are many numeric literals in the
INC DE	statement then the error marker location
JR L3F04	may never be found before the end of the statement is parsed. Credit: Ian Collier (+3), Andrew Owen (128)]
	Jump ahead to continue.

Come here if tokenizing the BASIC line

L3EEE:	PUSH AF	Save the character to insert and the carry flag reset.
	LD BC,\$0001	Request to insert 1 byte.
	PUSH HL	
	PUSH DE	
	CALL L3F0D	Check that there is memory available for 1 byte,
	POP DE	automatically producing error '4' if not.
	POP HL	
	RST 28H	BC=Number of bytes. HL=Address location before the position.
	DEFW POINTERS	\$1664. Update all system variables due to the insertion. Exit with DE pointing to old
	LD HL,(\$5C65)	STKEND position, BC with number of bytes 'shifted'.
	EX DE,HL	STKEND. Fetch the start of the spare memory.
	LDDR	DE=Address of spare memory. HL=Address of character in the BASIC line.
	POP AF	Shift up all affected bytes to make the room for the new character.
		Retrieve the character to insert and the flags. The carry flag will be reset and hence
		will indicate that tokenizing the BASIC line is not complete.
	LD (DE),A	Store the character in the BASIC line workspace.
L3F04:	INC DE	Advance to the next character in the BASIC line.
	CALL L1F64	Use Workspace RAM configuration (physical RAM bank 7).
	LD (\$FD82),DE	Store the address of the next insertion position within the BASIC line workspace.
	RET	

## Room for BC Bytes?

Test whether there is room for the specified number of bytes in the spare memory, producing error "4 Out of memory" if not.

Entry: BC=Number of bytes required.

Exit : Returns if the room requested room is available else an error '4' is produced.

## SPECTRUM +2 ROM o DISASSEMBLY

L3F0D:	LD HL,(\$5C65) ADD HL,BC JR C,L3F1D EX DE,HL LD HL,\$0082 ADD HL,DE JR C,L3F1D SBC HL,SP RET C	STKEND. Would adding the specified number of bytes overflow the RAM area? Jump to produce an error if so. DE=New end address. Would there be at least 130 bytes at the top of RAM?  Jump to produce an error if not. If the stack is lower in memory, would there still be enough room? Return if there would.
L3F1D:	LD A,\$03 LD (\$5C3A),A JP L0321	ERR_NR. Signal error "4 Out of Memory". Jump to error handler routine.

## Identify Keyword

This routine identifies the string within the Keyword Conversion Buffer and returns the token character code. The last character of the string has bit 7 set. The routine attempts to identify 48K mode keywords, 128K mode keywords and a number of mis-spelled keywords (those that require a space within them). Exit: Carry flag set if a keyword was identified.  
A=Token character code.

L3F25:	CALL \$FD2E RET C	Attempt to identify 48K mode keyword. Return if keyword identified.
--------	----------------------	--

Attempt to identify 128K mode keywords and mis-spelled keywords.

LD B,\$F9 LD DE,\$FD74 LD HL,L35BA CALL \$FD3B RET NC	Base character code (results in codes \$F9-\$FF). DE=Address of Keyword Conversion Buffer. HL=Keywords string table. Attempt to identify 128K mode/mis-spelled keyword. Return if no keyword identified.
---	--

Attempt to convert mis-spelled keywords

	CP \$FF JR NZ,L3F3D LD A,\$D4 JR L3F5F	Was it "CLOSE#"?  Use character code for 'CLOSE #'. Jump ahead to continue.
L3F3D:	CP \$FE JR NZ,L3F45 LD A,\$D3 JR L3F5F	Was it "OPEN#"? Jump if not. Use character code for 'OPEN #'. Jump ahead to continue.
L3F45:	CP \$FD JR NZ,L3F4D LD A,\$CE JR L3F5F	Was it "DEFFN"? Jump if not. Use character code for 'DEF FN'. Jump ahead to continue.
L3F4D:	CP \$FC JR NZ,L3F55 LD A,\$ED JR L3F5F	Was it "GOSUB"? Jump if not. Use character code for 'GO SUB'. Jump ahead to continue.
L3F55:	CP \$FB JR NZ,L3F5D LD A,\$EC JR L3F5F	Was it "GOTO"? Jump if not. Use character code for 'GO TO'. Jump ahead to continue.
L3F5D:	SUB \$56	Reduce to \$A3 for 'SPECTRUM' and \$A4 for 'PLAY'.
L3F5F:	SCF RET	Signal keyword identified.

## Copy Data Block

This routine is used on 8 occasions to copy a block of default data.

Entry: DE=Destination address.  
HL=Address of source data table, which starts with the number of bytes to copy followed by the bytes themselves.

L3F61:	LD B,(HL)	Get number of bytes to copy.
	INC HL	Point to the first byte to copy.
L3F63:	LD A,(HL)	Fetch the byte from the source
	LD (DE),A	and copy it to the destination.
	INC DE	Increment destination address.
	INC HL	Increment source address.
	DJNZ L3F63	Repeat for all bytes.
	RET	

## Get Numeric Value for ASCII Character

Exit: Carry flag set if character was numeric and A holding value.  
 [Never called by this ROM]

L3F6A:	CP '0'	\$30. Test against '0'.
	CCF	
	RET NC	Return with carry flag reset if not numeric character.
	CP ':'	\$3A. Test against ':'.
	RET NC	Return with carry flag reset if not numeric character.
	SUB '0'	\$30. Get numeric value.
	SCF	Return with carry flag set to indicate a numeric character.
	RET	

## Call Action Handler Routine

If the code in A matches an entry in the table pointed to by HL then execute the action specified by the entry's routine address.

Entry: A=Code.  
 HL=Address of action table.

Exit : Zero flag reset if no match found.  
 Carry flag reset if an error beep is required, or to signal no suitable action handler found.  
 HL=Address of next table entry if a match was found.

L3F75:	PUSH BC	Save registers.
	PUSH DE	
	LD B,(HL)	Fetch number of table entries.
	INC HL	Point to first entry.
L3F79:	CP (HL)	Possible match for A?
	INC HL	
	LD E,(HL)	
	INC HL	
	LD D,(HL)	DE=Address to call if a match.
	JR Z,L3F88	Jump if a match.
	INC HL	Next table entry.
	DJNZ L3F79	Repeat for next table entry.

No match found

	SCF	Return with carry flag reset to signal an error beep is required
	CCF	and with the zero flag reset to signal a match was not found.
	POP DE	Restore registers.
	POP BC	
	RET	

Found a match

L3F88:	EX DE,HL	HL=Action routine to call.
	POP DE	
	POP BC	
	CALL L3F95	Indirectly call the action handler routine.
	JR C,L3F92	Jump if no error beep is required.
	CP A	Set zero flag to indicate a match was found.
	RET	Exit with carry flag reset to indicate error beep required.
L3F92:	CP A	Set zero flag to indicate a match was found.

## PROGRAMMERS' INITIALS

L3F96:	DEFB \$00	
	DEFM "MB"	Martin Brennan.
	DEFB \$00	
	DEFM "SB"	Steve Berry.
	DEFB \$00	
	DEFM "AC"	Andrew Cummins.
	DEFB \$00	
	DEFM "RG"	Rupert Goodwins.
	DEFB \$00	
	DEFM "KM"	Kevin Males.
	DEFB \$00	

## UNUSED SPACE

[illegible]

## END OF ROM MARKER

```
L3FFF:      DEFB $01
            END
```

## REFERENCE INFORMATION — PART 2

### Routines Copied/Constructed in RAM

#### Construct Keyword Representation

This routine copies a keyword string from ROM 1 into the BASIC Line Construction Buffer, terminating it with an 'end of BASIC line' marker (code '+'+\$80). Only standard Spectrum keywords are handled by this routine (SPECTRUM and PLAY are processed elsewhere).

The routine is run from RAM bank 7 at \$FCAE so that access to both ROMs is available.

Depending on the value of A (which should be the ASCII code less \$A5, e.g. 'RND', the first (48K) keyword, has A=0), a different index into the token table is taken. This is to allow speedier lookup since there are never more than 15 keywords to advance through.

Entry: A=Keyword character code-\$A5 (range \$00-\$5A).

DE=Insertion address within BASIC Line Construction Buffer.

Copied to physical RAM bank 7 at \$FCAE-\$FCFC by routine at \$3385 (ROM 0).

\$FCAE	DI	Disable interrupts whilst paging.
	LD BC,\$7FFD	
	LD D,\$17	Page in ROM 1, SCREEN 0, no locking, RAM bank 7.
	OUT (C),D	
	CP \$50	Was the token \$F5 or above?
	JR NC,\$FCEB	
	CP \$40	Was the token \$E5 or above?
	JR NC,\$FCE4	
	CP \$30	Was the token \$D5 or above?
	JR NC,\$FCDD	
	CP \$20	Was the token \$C5 or above?
	JR NC,\$FCD6	
	CP \$10	Was the token \$B5 or above?
	JR NC,\$FCCF	

Used for token range \$A5-\$B4 (\$00 <= A <= \$0F)

LD HL,\$0096	Token table entry 'RND' in ROM 1.
JR \$FCF0	

Used for token range \$B5-\$C4 (\$10 <= A <= \$1F)

\$FCCF	SUB \$10	
	LD HL,\$00CF	Token table entry 'ASN' in ROM 1.
	JR \$FCF0	

Used for token range \$C5-\$D4 (\$20 <= A <= \$2F)

\$FCD6	SUB \$20	
	LD HL,\$0100	Token table entry 'OR' in ROM 1.
	JR \$FCF0	

Used for token range \$D5-\$E4 (\$30 <= A <= \$3F)

\$FCDD	SUB \$30	
	LD HL,\$013E	Token table entry 'MERGE' in ROM 1.
	JR \$FCF0	

Used for token range \$E5-\$F4 (\$40 <= A <= \$4F)

\$FCE4	SUB \$40	
	LD HL,\$018B	Token table entry 'RESTORE' in ROM 1.
	JR \$FCF0	

Used for token range \$F5-\$FF (A >= \$50)

\$FCEB	SUB \$50	
	LD HL,\$01D4	Token table entry 'PRINT' in ROM 1.
\$FCF0	LD B,A	Take a copy of the index value.
	OR A	If A=0 then already have the entry address.
\$FCF2	JR Z,\$FCFD	If indexed item found then jump ahead to copy the characters of the token.
\$FCF4	LD A,(HL)	Fetch a character.
	INC HL	Point to next character.
	AND \$80	Has end of token marker been found?
	JR Z,\$FCF4	Loop back for next character if not.
	DEC B	Count down the index of the required token.

## Copy Keyword Characters

This routine copies a keyword string from ROM 1 into the BASIC Line Construction Buffer, terminating it with an 'end of BASIC line' marker (code ' '+\$80). The routine is run from RAM bank 7 so that access to both ROMs is available.

Entry: HL=Address of keyword string in ROM 1.

DE=Insertion address within BASIC Line Construction Buffer.

Copied to physical RAM bank 7 at \$FCFD-\$FD2D by subroutine at \$3385 (ROM 0).

\$FCFD	LD DE,\$FCA3	DE=Keyword Construction Buffer.
	LD (\$FCA1),DE	Store the start address of the constructed keyword.
	LD A,\$FC9E)	Print a leading space?
	OR A	
	LD A,\$00	
	LD (\$FC9E),A	Signal leading space not required.
	JR NZ,\$FD13	Jump if leading space not required.
	LD A,\$20	Print a leading space.
	LD (DE),A	Insert a leading space.
	INC DE	Advance to next buffer position.
\$FD13	LD A,(HL)	Fetch a character of the keyword.
	LD B,A	Store it.
	INC HL	Advance to next keyword character.
	LD (DE),A	Store the keyword character in the BASIC line buffer.
	INC DE	Advance to the next buffer position.
	AND \$80	Test if the end of the keyword string.
	JR Z,\$FD13	Jump back if not to repeat for all characters of the keyword.
	LD A,B	Get keyword character back.
	AND \$7F	Mask of bit 7 which indicates the end of string marker.
	DEC DE	Point back at the last character of the keyword copied into the buffer
	LD (DE),A	and store it.
	INC DE	Advance to the position in the buffer after the last character of the keyword.
	LD A,' '+\$80	\$A0. ' ' + end marker
	LD (DE),A	Store an 'end of BASIC line so far' marker.
	LD A,\$07	
	LD BC,\$7FFD	
	OUT (C),A	Page in ROM 0, SCREEN 0, no locking, RAM bank 7.
	EI	Re-enable interrupts.

## Identify Token

This routine identifies the string within the Keyword Conversion Buffer and returns the character code. The last character of the string to identify has bit 7 set.

Exit: Carry flag set if token identified.

B=Character code.

Copied to physical RAM bank 7 at \$FD2E-\$FD69 by subroutine at \$3385 (ROM 0).

\$FD2E	DI	Disable interrupts whilst paging.
	LD BC,\$7FFD	
	LD D,\$17	Select ROM 1, SCREEN 0, RAM bank 7.
	OUT (C),D	
	LD HL,\$0096	Address of token table in ROM 1.
	LD B,\$A5	Character code of the first token - 'RND'.

Entry point here used to match 128K mode tokens and mis-spelled tokens

## SPECTRUM +2 ROM 0 DISASSEMBLY

\$FD3B	LD DE,\$FD74	Keyword Conversion Buffer holds the text to match against.
\$FD3E	LD A,(DE)	Fetch a character from the buffer.
	AND \$7F	Mask off terminator bit.
	CP \$61	Is it lowercase?
	LD A,(DE)	Fetch the character again from the buffer.
	JR C,\$FD48	Jump if uppercase.
	AND \$DF	Make the character uppercase.
\$FD48	CP (HL)	Does the character match the current item in the token table?
	JR NZ,\$FD54	Jump if it does not.
	INC HL	Point to the next character in the buffer.
	INC DE	Point to the next character in the token table.
	AND \$80	Has the terminator been reached?
	JR Z,\$FD3E	Jump back if not to test the next character in the token.

A match was found

	SCF	Signal a match was found.
	JR \$FD60	Jump ahead to continue.
\$FD54	INC B	The next character code to test against.
	JR Z,\$FD5F	Jump if all character codes tested.

The token does not match so skip to the next entry in the token table

\$FD57	LD A,(HL)	Fetch the character from the token table.
	AND \$80	Has the end terminator been found?
	INC HL	Point to the next character.
	JR Z,\$FD57	Jump back if no terminator found.
	JR \$FD3B	Jump back to test against the next token.

All character codes tested and no match found

\$FD5F	OR A	Clear the carry flag to indicate no match found.
--------	------	--

The common exit point

\$FD60	LD A,B	Fetch the character code of the matching token (\$00 for no match).
	LD D,\$07	Select ROM 0, SCREEN 0, RAM bank 7.
	LD BC,\$7FFD	
	OUT (C),D	
	EI	Re-enable interrupts.

## Insert Character into Display File

Copy a character into the display file.

Entry: HL=Character data.  
DE=Display file address.

This routine is constructed from three segments and stitched together in physical RAM bank 7 to form a single routine.

Created in physical RAM Bank 7 at \$FF28-\$FF60 by routine at \$248E (ROM 0). [Construction routine never actually called by the ROM]

\$FF28	PUSH BC	Save BC
	DI	Disable interrupts whilst paging.
	LD BC,\$7FFD	
	LD A,(BANK_M)	\$5B5C. Fetch current paging configuration.
	XOR \$10	Toggle ROMs.
	OUT (C),A	Perform paging.
	EI	Re-enable interrupts.
	EX AF,AF'	Save the new configuration in A'.
	LD C,D	Save D.
	LD A,(HL)	
	LD (DE),A	Copy byte 1.
	INC HL	
	INC D	
	LD A,(HL)	

LD (DE),A	Copy byte 2.
INC HL	
INC D	
LD A,(HL)	
LD (DE),A	Copy byte 3.
INC HL	
INC D	
LD A,(HL)	
LD (DE),A	Copy byte 4.
INC HL	
INC D	
LD A,(HL)	
LD (DE),A	Copy byte 5.
INC HL	
INC D	
LD A,(HL)	
LD (DE),A	Copy byte 6.
INC HL	
INC D	
LD A,(HL)	
LD (DE),A	Copy byte 7.
INC HL	
INC D	
LD A,(HL)	
LD (DE),A	Copy byte 8.
LD D,C	Restore D.
EX AF,AF'	Retrieve current paging configuration.
DI	Disable interrupts whilst paging.
LD C,\$FD	Restore Paging I/O port number.
XOR \$10	Toggle ROMs.
OUT (C),A	Perform paging.
EI	Re-enable interrupts.
POP BC	Restore BC.

## Standard Error Report Codes

0 — OK	Successful completion, or jump to a line number bigger than any existing.
1 — NEXT without FOR	The control variable does not exist (it has not been set up by a FOR statement), but there is an ordinary variable with the same name.
2 — Variable not found	For a simple variable, this will happen if the variable is used before it has been assigned to by a LET, READ or INPUT statement, loaded from disk (or tape), or set up in a FOR statement. For a subscripted variable, it will happen if the variable is used before it has been dimensioned in a DIM statement, or loaded from disk (or tape).
3 — Subscript wrong	A subscript is beyond the dimension of the array or there are the wrong number of subscripts.
4 — Out of memory	There is not enough room in the computer for what you are trying to do.
5 — Out of screen	An INPUT statement has tried to generate more than 23 lines in the lower half of the screen. Also occurs with 'PRINT AT 22,xx'.
6 — Number too big	Calculations have yielded a number greater than approximately 10 <sup>38</sup> .
7 — RETURN without GO SUB	There has been one more RETURN than there were GO SUBs.
8 — End of file	Input returned unacceptable character code.
9 — STOP statement	After this, CONTINUE will not repeat the STOP but carries on with the statement after.
A — Invalid argument	The argument for a function is unsuitable.
B — Integer out of range	When an integer is required, the floating point argument is rounded to the nearest integer. If this is outside a suitable range, then this error results.
C — Nonsense in BASIC	The text of the (string) argument does not form a valid expression.
D — BREAK - CONT repeats	BREAK was pressed during some peripheral operation.
E — Out of DATA	You have tried to READ past the end of the DATA list.
F — Invalid file name	SAVE with filename empty or longer than 10 characters.
G — No room for line	There is not enough room left in memory to accommodate the new program line.
H — STOP in INPUT	Some INPUT data started with STOP.
I — FOR without NEXT	A FOR loop was to be executed no times (e.g. FOR n=1 TO 0) and corresponding NEXT statement could not be found.
J — Invalid I/O device	Attempting to input characters from or output characters to a device that doesn't support it.
K — Invalid colour	The number specified is not an appropriate value.
L — BREAK into program	BREAK pressed. This is detected between two statements.

M — RAMTOP no good	The number specified for RAMTOP is either too big or too small.
N — Statement lost	Jump to a statement that no longer exists.
O — Invalid Stream	Trying to input from or output to a stream that isn't open or that is out of range (0...15), or trying to open a stream that is out of range.
P — FN without DEF	User-defined function used without a corresponding DEF in the program.
Q — Parameter error	Wrong number of arguments, or one of them is the wrong type.
R — Tape loading error	A file on tape was found but for some reason could not be read in, or would not verify.

## Standard System Variables

These occupy addresses \$5C00-\$5CB5.

KSTATE	\$5C00	8	IY-\$3A	Used in reading the keyboard.
LASTK	\$5C08	1	IY-\$32	Stores newly pressed key.
REPDEL	\$5C09	1	IY-\$31	Time (in 50ths of a second) that a key must be held down before it repeats. This starts off at 35.
REPPER	\$5C0A	1	IY-\$30	Delay (in 50ths of a second) between successive repeats of a key held down - initially 5.
DEFADD	\$5C0B	2	IY-\$2F	Address of arguments of user defined function (if one is being evaluated), otherwise 0.
K_DATA	\$5C0D	1	IY-\$2D	Stores second byte of colour controls entered from keyboard.
TVDATA	\$5C0E	2	IY-\$2C	Stores bytes of colour, AT and TAB controls going to TV.
STRMS	\$5C10	38	IY-\$2A	Addresses of channels attached to streams.
CHARS	\$5C36	2	IY-\$04	256 less than address of character set, which starts with ' ' and carries on to '©'.
RASP	\$5C38	1	IY-\$02	Length of warning buzz.
PIP	\$5C39	1	IY-\$01	Length of keyboard click.
ERR_NR	\$5C3A	1	IY+\$00	1 less than the report code. Starts off at 255 (for -1) so 'PEEK 23610' gives 255.
FLAGS	\$5C3B	1	IY+\$01	Various flags to control the BASIC system: Bit 0: 1=Suppress leading space. Bit 1: 1=Using printer, 0=Using screen. Bit 2: 1=Print in L-Mode, 0=Print in K-Mode. Bit 3: 1=L-Mode, 0=K-Mode. Bit 4: 1=128K Mode, 0=48K Mode. [Always 0 on 48K Spectrum] Bit 5: 1=New key press code available in LAST_K. Bit 6: 1=Numeric variable, 0=String variable. Bit 7: 1=Line execution, 0=Syntax checking.
TVFLAG	\$5C3C	1	IY+\$02	Flags associated with the TV: Bit 0 : 1=Using lower editing area, 0=Using main screen. Bit 1-2: Not used (always 0). Bit 3 : 1=Mode might have changed. Bit 4 : 1=Automatic listing in main screen, 0=Ordinary listing in main screen. Bit 5 : 1=Lower screen requires clearing after a key press. Bit 6 : 1=Tape Loader option selected (set but never tested). [Always 0 on 48K Spectrum] Bit 7 : Not used (always 0).
ERR_SP	\$5C3D	2	IY+\$03	Address of item on machine stack to be used as error return.
LISTSP	\$5C3F	2	IY+\$05	Address of return address from automatic listing.
MODE	\$5C41	1	IY+\$07	Specifies cursor type: \$00='L' or 'C'. \$01='E'. \$02='G'. \$04='K'.
NEWPPC	\$5C42	2	IY+\$08	Line to be jumped to.
NSPPC	\$5C44	1	IY+\$0A	Statement number in line to be jumped to.
PPC	\$5C45	2	IY+\$0B	Line number of statement currently being executed.
SUBPPC	\$5C47	1	IY+\$0D	Number within line of statement currently being executed.
BORDCR	\$5C48	1	IY+\$0E	Border colour multiplied by 8; also contains the attributes normally used for the lower half of the screen.
E_PPC	\$5C49	2	IY+\$0F	Number of current line (with program cursor).
VARS	\$5C4B	2	IY+\$11	Address of variables.
DEST	\$5C4D	2	IY+\$13	Address of variable in assignment.
CHANS	\$5C4F	2	IY+\$15	Address of channel data.

# SPECTRUM +2 ROM o DISASSEMBLY

CURCHL	\$5C51	2	IY+\$17	Address of information currently being used for input and output.
PROG	\$5C53	2	IY+\$19	Address of BASIC program.
NXTLIN	\$5C55	2	IY+\$1B	Address of next line in program.
DATADD	\$5C57	2	IY+\$1D	Address of terminator of last DATA item.
E_LINE	\$5C59	2	IY+\$1F	Address of command being typed in.
K_CUR	\$5C5B	2	IY+\$21	Address of cursor.
CH_ADD	\$5C5D	2	IY+\$23	Address of the next character to be interpreted - the character after the argument of PEEK, or the NEWLINE at the end of a POKE statement.
X_PTR	\$5C5F	2	IY+\$25	Address of the character after the '?' marker.
WORKSP	\$5C61	2	IY+\$27	Address of temporary work space.
STKBOT	\$5C63	2	IY+\$29	Address of bottom of calculator stack.
STKEND	\$5C65	2	IY+\$2B	Address of start of spare space.
BREG	\$5C67	1	IY+\$2D	Calculator's B register.
MEM	\$5C68	2	IY+\$2E	Address of area used for calculator's memory (usually MEMBOT, but not always).
FLAGS2	\$5C6A	1	IY+\$30	Flags: Bit 0 : 1=Screen requires clearing. Bit 1 : 1=Printer buffer contains data. Bit 2 : 1=In quotes. Bit 3 : 1=CAPS LOCK on. Bit 4 : 1=Using channel 'K'. Bit 5-7: Not used (always 0).
DF_SZ	\$5C6B	1	IY+\$31	The number of lines (including one blank line) in the lower part of the screen.
S_TOP	\$5C6C	2	IY+\$32	The number of the top program line in automatic listings.
OLDPPC	\$5C6E	2	IY+\$34	Line number to which CONTINUE jumps.
OSPPC	\$5C70	1	IY+\$36	Number within line of statement to which CONTINUE jumps.
FLAGX	\$5C71	1	IY+\$37	Flags: Bit 0 : 1=Simple string complete so delete old copy. Bit 1 : 1=Indicates new variable, 0=Variable exists. Bit 2-4: Not used (always 0). Bit 5 : 1=INPUT mode. Bit 6 : 1=Numeric variable, 0=String variable. Holds nature of existing variable. Bit 7 : 1=Using INPUT LINE.
STRLEN	\$5C72	2	IY+\$38	Length of string type destination in assignment.
T_ADDR	\$5C74	2	IY+\$3A	Address of next item in syntax table.
SEED	\$5C76	2	IY+\$3C	The seed for RND. Set by RANDOMIZE.
FRAMES	\$5C78	3	IY+\$3E	3 byte (least significant byte first), frame counter incremented every 20ms.
UDG	\$5C7B	2	IY+\$41	Address of first user-defined graphic. Can be changed to save space by having fewer user-defined characters.
COORDS	\$5C7D	1	IY+\$43	X-coordinate of last point plotted.
	\$5C7E	1	IY+\$44	Y-coordinate of last point plotted.
P_POSN	\$5C7F	1	IY+\$45	33-column number of printer position.
PR_CC	\$5C80	2	IY+\$46	Full address of next position for LPRINT to print at (in ZX Printer buffer). Legal values \$5B00 - \$5B1F. [Not used in 128K mode]
ECHO_E	\$5C82	2	IY+\$48	33-column number and 24-line number (in lower half) of end of input buffer.
DF_CC	\$5C84	2	IY+\$4A	Address in display file of PRINT position.
DF_CCL	\$5C86	2	IY+\$4C	Like DF CC for lower part of screen.
S_POSN	\$5C88	1	IY+\$4E	33-column number for PRINT position.
	\$5C89	1	IY+\$4F	24-line number for PRINT position.
SPOSNL	\$5C8A	2	IY+\$50	Like S_POSN for lower part.
SCR_CT	\$5C8C	1	IY+\$52	Counts scrolls - it is always 1 more than the number of scrolls that will be done before stopping with 'scroll?'. Permanent current colours, etc, as set up by colour statements.
ATTR_P	\$5C8D	1	IY+\$53	
MASK_P	\$5C8E	1	IY+\$54	Used for transparent colours, etc. Any bit that is 1 shows that the corresponding attribute bit is taken not from ATTR_P, but from what is already on the screen.
ATTR_T	\$5C8F	1	IY+\$55	Temporary current colours (as set up by colour items).
MASK_T	\$5C90	1	IY+\$56	Like MASK_P, but temporary.
P_FLAG	\$5C91	1	IY+\$57	Flags:

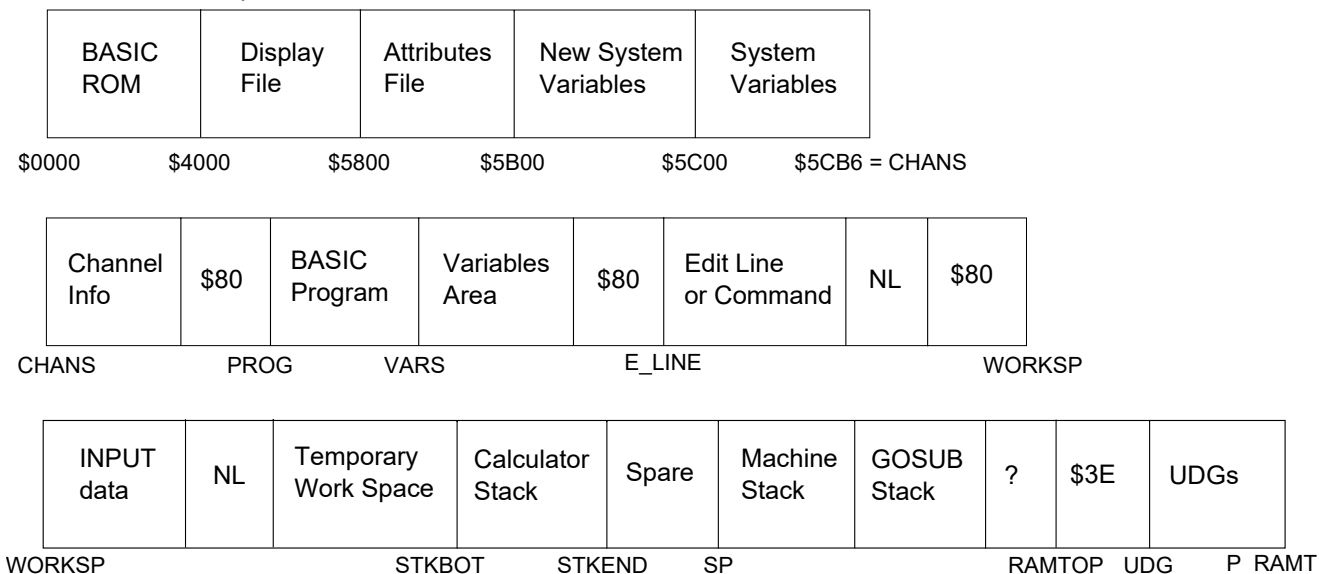
## SPECTRUM +2 ROM o DISASSEMBLY

Bit 0: 1=OVER 1, 0=OVER 0.  
 Bit 1: Not used (always 0).  
 Bit 2: 1=INVERSE 1, 0=INVERSE 0.  
 Bit 3: Not used (always 0).  
 Bit 4: 1=Using INK 9.  
 Bit 5: Not used (always 0).  
 Bit 6: 1=Using PAPER 9.  
 Bit 7: Not used (always 0).

MEMBOT	\$5C92	30	IY+\$58	Calculator's memory area - used to store numbers that cannot conveniently be put on the calculator stack.
	\$5CB0	2	IY+\$76	Not used on standard Spectrum. [Used by ZX Interface 1 Edition 2 for printer WIDTH]
RAMTOP	\$5CB2	2	IY+\$78	Address of last byte of BASIC system area.
P_RAMT	\$5CB4	2	IY+\$7A	Address of last byte of physical RAM.

## Memory Map

The conventional memory is used as follows:



## I Register

The I register is used along with the R register by the Z80 for automatic memory refreshing. Setting the I register to a value between \$40 and \$7F causes memory refreshes to occur to the lower 16K RAM. This RAM is contended with the ULA which uses it for the generation of the video display. The memory refreshes get interpreted by the ULA as the CPU requesting to access the lower 16K RAM bank very rapidly and very often. The ULA is not able to handle reads at such a high frequency, with the consequence that it fails to fetch and output the next screen byte. Instead it uses re-uses the byte previously read. This causes a visible corruption to the video display output, often referred to a 'snow', although no actual corruption occurs to the video display RAM. This also happens when the I register is set to a value between \$C0 and \$FF when a contended RAM bank is paged in and, unlike the Spectrum 16K/48K, can lead to a machine crash.

## Screen File Formats

The two screens available on the Spectrum +2, the normal screen in RAM bank 5 (\$4000-\$5AFF) and the shadow screen in RAM bank 7 (\$C000-\$FFFF), both use the same file format.

## Display File

The display file consists of 3 areas, each consisting of 8 characters rows, with each row consisting of 8 pixel lines.

Each pixel line consists of 32 cell columns, with each cell consisting of a byte that represents 8 pixels.

The address of a particular cell is formed as follows:

s	1	0	a	a	l	l	l	r	r	r	c	c	c	c	c
Bit: 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

where: s = Screen (0-1: 0=Normal screen, 1=Shadow Screen)  
 aa = Area (0-2)  
 rrr = Row (0-7)  
 lll = Line (0-7)  
 ccccc = Column (0-31)

An area value of 3 denotes the attributes file, which consists of a different format.

## Attributes File

The attributes file consists of 24 characters rows, with each row consisting of 32 cell columns.  
 Each cell consisting of a byte that holds the colour information.

The address of a particular cell is formed as follows:

s	1	0	1	1	0	r	r	r	r	r	c	c	c	c	c
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

where: s = Screen (0-1: 0=Normal screen, 1=Shadow Screen)  
 rrrrr = Row (0-23)  
 ccccc = Column (0-31)

Each cell holds a byte of colour information:

f	b	p	p	p	i	i	i
7	6	5	4	3	2	1	0

where: f = Flash (0-1: 0=Off, 1=On)  
 b = Bright (0-1: 0=Off, 1=On)  
 ppp = Paper (0-7: 0=Black, 1=Blue, 2=Red, 3=Magenta, 4=Green, 5=Cyan, 6=Yellow, 7=White)  
 iii = Ink (0-7: 0=Black, 1=Blue, 2=Red, 3=Magenta, 4=Green, 5=Cyan, 6=Yellow, 7=White)

## Address Conversion Between Display File and Attributes File

The address of the attribute cell corresponding to an address in the display file can be constructed by moving bits 11 to 12 (the area value) to bit positions 8 to 9, setting bit 10 to 0 and setting bits 11 to 12 to 1.

The address of the display file character cell corresponding to an address in the attributes file can be constructed by moving bits 8 to 9 (the row value) to bit positions 11 to 12, and then setting bits 8 to 9 to 0.

## Standard I/O Ports

### Port \$FE

This controls the cassette interface, the speaker, the border colour and is used to read the keyboard.

Since it is the ULA that controls these facilities, it will introduce a delay when accessing the port if it is busy at the time, and hence I/O port \$FE is subject to contention.

OUTPUT:

Bit 0-2: Border colour (0=Black, 1=Blue, 2=Red, 3=Magenta, 4=Green, 5=Cyan, 6=Yellow, 7=White).

Bit 3 : MIC output (1=Off, 0=On).

Bit 4 : Speaker output (1=On, 0=Off).

Bit 5-7: Not used.

INPUT:

Upper byte selects keyboard row to read.

	Bit0	Bit1	Bit2	Bit3	Bit4	Bit4	Bit3	Bit2	Bit1	Bit0	
\$F7FE	1	2	3	4	5	6	7	8	9	0	\$EFFE
\$FBFE	Q	W	E	R	T	Y	U	I	O	P	\$DFFE
\$FDFE	A	S	D	F	G	H	J	K	L	ENTER	\$BFFE
\$FEFE	SHIFT	Z	X	C	V	B	N	M	SYM	SPACE	\$7FFE

Bit 0-4 : Key states (corresponding bit is 0 if the key is pressed).

Bit 5 : Not used (always 1).

Bit 6 : EAR input.

Bit 7 : Not used (always 1).

## Cassette Header Format

A file consists of a header block followed by a data block. Each block begins with a flag that indicates whether it is a header block or a data block. Next are the header or data bytes, and finally a checksum of the flag and header/data bytes.

Flag - A value of \$00 for a header and \$FF for a data block.

Bytes - The bytes forming the header information or the file data.

Checksum - An XOR checksum of the Flag and Bytes fields.

The header information consists of 17 bytes and these describe the size and type of data that the data block contains.

The header bytes have the following meaning:

Byte \$00 : File type - \$00=Program, \$01=Numeric array, \$02=Character array, \$03=Code/Screen\$.

Bytes \$01-\$0A: File name, padding with trailing spaces.

Bytes \$0B-\$0C: Length of program/code block/screen\$/array (\$1B00 for screen\$).

Bytes \$0D-\$0E: For a program, it holds the auto-run line number (\$80 in byte \$0E if no auto-run).

For code block/screen\$ it holds the start address (\$4000 for screen\$).

For an array, it holds the variable name in byte \$0E.

Bytes \$0F-\$10: Offset to the variables (i.e. length of program) if a program.

## AY-3-8912 Programmable Sound Generator Registers

This is controlled through output I/O port \$FFFD. It is driven from a 1.77345 MHz clock.

The datasheet for the AY-3-8912 lists to the registers in octal, but below they are listed in decimal.

### Registers 0 and 1 (Channel A Tone Generator)

Forms a 12 bit pitch control for sound channel A. The basic unit of tone is the clock frequency divided by 16, i.e. 110.841 kHz. With a 12 bit counter range, 4095 different frequencies from 27.067 Hz to 110.841 kHz (in increments of 27.067 Hz) can be generated.

Bits 0-7 : Contents of register 0.

Bits 8-11 : Contents of lower nibble of register 1.

Bits 12-15: Not used.

### Registers 2 and 3 (Channel B Tone Generator)

Forms a 12 bit pitch control for sound channel B.

Bits 0-7 : Contents of register 2.

Bits 8-11 : Contents of lower nibble of register 3.

Bits 12-15: Not used.

### Registers 4 and 5 (Channel C Tone Generator)

Forms a 12 bit pitch control for sound channel C.

Bits 0-7 : Contents of register 4.

Bits 8-11 : Contents of lower nibble of register 5.

Bits 12-15: Not used.

### Register 6 (Noise Generator)

The frequency of the noise is obtained in the PSG by first counting down the input clock by 16 (i.e. 110.841 kHz), then by further counting down the result by the programmed 5 bit noise period value held in bits 0-4 of register 6. With a 5 bit counter range, 31 different frequencies from 3.576 kHz to 110.841 kHz (in increments of 3.576 kHz) can be generated.

### Register 7 (Mixer — I/O Enable)

This controls the enable status of the noise and tone mixers for the three channels, and also controls the I/O port used to drive the RS232 and Keypad sockets.

Bit 0: Channel A Tone Enable (0=enabled).

Bit 1: Channel B Tone Enable (0=enabled).

Bit 2: Channel C Tone Enable (0=enabled).

Bit 3: Channel A Noise Enable (0=enabled).

Bit 4: Channel B Noise Enable (0=enabled).

Bit 5: Channel C Noise Enable (0=enabled).

Bit 6: I/O Port Enable (0=input, 1=output).

Bit 7: Not used.

### Register 8 (Channel A Volume)

This controls the volume of channel A.

Bits 0-4: Channel A volume level.

Bit 5 : 1=Use envelope defined by register 13 and ignore the volume setting.

Bits 6-7: Not used.

## Register 9 (Channel B Volume)

This controls the volume of channel B.

Bits 0-4: Channel B volume level.

Bit 5 : 1=Use envelope defined by register 13 and ignore the volume setting.

Bits 6-7: Not used.

## Register 10 (Channel C Volume)

This controls the volume of channel C.

Bits 0-4: Channel C volume level.

Bit 5 : 1=Use envelope defined by register 13 and ignore the volume setting.

Bits 6-7: Not used.

## Register 11 and 12 (Envelope Period)

These registers allow the frequency of the envelope to be selected.

The frequency of the envelope is obtained in the PSG by first counting down the input clock by 256 (6.927 kHz), then further counting down the result by the programmed 16 bit envelope period value. With a 16 bit counter range, 65535 different frequencies from 1.691 Hz to 110.841 kHz (in increments of 1.691 Hz) can be generated.

Bits 0-7 : Contents of register 11.

Bits 8-15: Contents of register 12.

## Register 13 (Envelope Shape)

This register allows the shape of the envelope to be selected.

The envelope generator further counts down the envelope frequency by 16, producing a 16-state per cycle envelope pattern. The particular shape and cycle pattern of any desired envelope is accomplished by controlling the count pattern of the 4 bit counter and by defining a single cycle or repeat cycle pattern.

Bit 0 : Hold.

Bit 1 : Alternate.

Bit 2 : Attack.

Bit 3 : Continue.

Bits 4-7: Not used.

These control bits can produce the following envelope waveforms:

Bit: 3 2 1 0

0 0 X X



Single decay then off.  
Used by W0 PLAY command.

0 1 X X



Single attack then off.  
Used by W1 PLAY command.

1 0 0 0



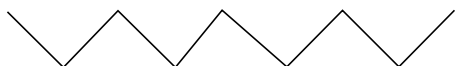
Repeated decay.  
Used by W4 PLAY command.

1 0 0 1



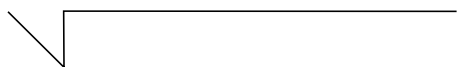
Single decay then off.  
Not used by PLAY command (use W0 instead).

1 0 1 0



Repeated decay-attack.  
Used by W7 PLAY command.

1 0 1 1



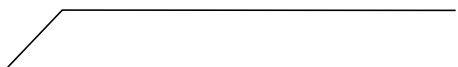
Single decay then hold.  
Used by W2 PLAY command.

1 1 0 0



Repeated attack.  
Used by W5 PLAY command.

1 1 0 1



Single attack then hold.  
Used by W3 PLAY command.

1 1 1 0

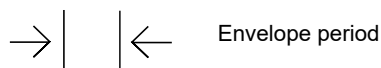


Repeated attack-delay.  
Used by W6 PLAY command.

1 1 1 1



Single attack then off.  
Not used by PLAY command (use W1 instead).



## Register 14 (I/O Port)

This controls the RS232 and Keypad sockets.

Once the register has been selected, it can be read via port \$FFFD and written via port \$BFFD.

Bit 0: KEYPAD CTS (out) - 0=Spectrum ready to receive, 1=Busy

Bit 1: KEYPAD RXD (out) - 0=Transmit high bit, 1=Transmit low bit

Bit 2: RS232 CTS (out) - 0=Spectrum ready to receive, 1=Busy

Bit 3: RS232 RXD (out) - 0=Transmit high bit, 1=Transmit low bit

Bit 4: KEYPAD DTR (in) - 0=Keypad ready for data, 1=Busy

Bit 5: KEYPAD TXD (in) - 0=Receive high bit, 1=Receive low bit

Bit 6: RS232 DTR (in) - 0=Device ready for data, 1=Busy

Bit 7: RS232 TXD (in) - 0=Receive high bit, 1=Receive low bit

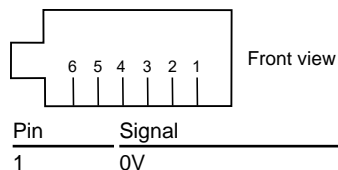
The RS232 port also doubles up as a MIDI port, with communications to MIDI devices occurring at 31250 baud.

Commands and data can be sent to MIDI devices. Command bytes have the most significant bit set, whereas data bytes have it reset.

## Socket Pin Outs

### RS232/MIDI Socket

The RS232/MIDI socket is controlled by register 14 of the AY-3-8912 sound generator.



2	TXD - In (Bit 7)
3	RXD - Out (Bit 3)
4	DTR - In (Bit 6)
5	CTS - Out (Bit 2)
6	12V

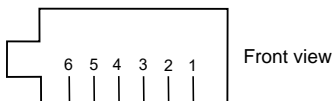
## Keypad Socket

The keypad socket is controlled by register 14 of the AY-3-8912 sound generator.

Only bits 0 and 5 are used for communications with the keypad (pins 2 and 5).

Writing a 1 to bit 0 (pin 2) will eventually force the keypad to reset.

Summary information about the keypad and its communications protocol can be found in the Spectrum 128 Service Manual and a detailed description can be found at [www.fruitcake.plus.com](http://www.fruitcake.plus.com).



Pin	Signal
1	0V
2	OUT - Out (Bit 0)
3	n/u - In (Bit 4)
4	n/u - Out (Bit 1)
5	IN - In (Bit 5)
6	12V

n/u = Not used for keypad communications.

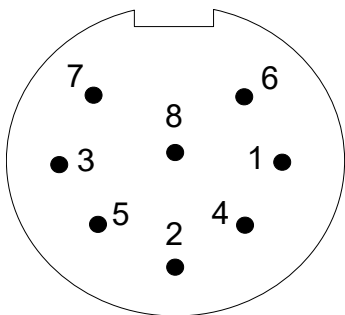
The keypad socket was later used by Amstrad to support a lightgun. There are no routines within the ROMs to handle communication with the lightgun so each game has to implement its own control software. Only bits 4 and 5 are used for communication with the lightgun (pins 3 and 5).

The connections to the lightgun are as follows:

Pin	Signal
1	0V
2	n/u - Out (Bit 0)
3	SENSOR - In (Bit 4)
4	n/u - Out (Bit 1)
5	TRIGGER - In (Bit 5)
6	12V

n/u = Not used for lightgun communication.

## Monitor Socket



Pin	Signal	Level
1	Composite PAL	1.2V pk-pk (75 Ohms)
2	0 Volts	0V
3	Bright Output	TTL
4	Composite Sync	TTL
5	Vertical Sync	TTL
6	Green	TTL
7	Red	TTL
8	Blue	TTL

A detailed description of the monitor socket and circuitry, and how to construct a suitable RGB SCART cable can be found at [www.fruitcake.plus.com](http://www.fruitcake.plus.com).

## Edge Connector

Pin	Side A	Side B
1	A15	A14
2	A13	A12
3	D7	+5V
4	n/u	+9V
5	Slot	Slot
6	D0	0V
7	D1	0V
8	D2	/CLK
9	D6	A0
10	D5	A1
11	D3	A2
12	D4	A3
13	/INT	/IORQULA
14	/NMI	0V
15	/HALT	n/u (On 48K Spectrum = VIDEO)
16	/MREQ	n/u (On 48K Spectrum = /Y)
17	/IORQ	n/u (On 48K Spectrum = V)
18	/RD	n/u (On 48K Spectrum = U)
19	/WR	/BUSREQ
20	-5V	/RESET
21	/WAIT	A7
22	+12V	A6
23	-12V	A5
24	/M1	A4
25	/RFSH	/ROMCS
26	A8	/BUSACK
27	A10	A9
28	n/u	A11

Side A=Component Side, Side B=Underside.

n/u = Not used.

## Sound Socket

## ROM 0 Differences Between Models

The Spectrum +2 contains all of the functionality of the Spectrum 128 but excludes all routines relating to the Tape Tester option. Aside from this, the only other changes are to the copyright message and the message displayed when the Tape Loader option is invoked. All of the bugs that exist in the Spectrum 128 ROM 0 are still present in the Spectrum +2. English, Spanish and French versions of the Spectrum +2 were produced and these differed only in the language of the menu and error messages. However, a consequence of these translations was that the location of various routines were shifted.

The following shows a comparison of ROM 0 for the range of Spectrum +2 models and the Spectrum 128, and details how the address ranges correspond between them.

Spectrum 128	Spectrum +2	French +2	Spanish +2
\$0000-\$0565	\$0000-\$0565	\$0000-\$0565	\$0000-\$0565
\$0566-\$057C	\$0566-\$059B	\$0566-\$059B	\$0566-\$059B
\$057D-\$2743	\$059C-\$276C	\$059C-\$276C	\$059C-\$276C
\$2744	\$2763	\$2763	\$2763
\$2745-\$2750	\$2764-\$276F	\$2764-\$276F	\$2764-\$276F
\$2751-\$2753			
\$2754	\$2770	\$2770	\$2770
\$2755-\$275D	\$2771-\$2779	\$2771-\$2779	\$2771-\$2779
\$275E-\$2768	\$277A-\$2784	\$277A-\$2781	\$277A-\$2784
\$2769-\$2771	\$2785-\$278D	\$2782-\$278B	\$2785-\$278D
\$2772-\$278B	\$278E-\$2797	\$278C-\$2797	\$278E-\$2798
\$277C-\$2783	\$2798-\$279F	\$2798-\$27A0	\$2799-\$27A0
\$2784-\$278E			
\$278F-\$27A0	\$27A0-\$27B1	\$27A1-\$27B2	\$27A1-\$27B2
\$27A1-\$27A9	\$27B2-\$27BA	\$27B3-\$27BB	\$27B3-\$27BB

# SPECTRUM +2 ROM o DISASSEMBLY

\$27AA-\$27B2	\$27BB-\$27C3	\$27BC-\$27C5	\$27BC-\$27C4
\$27B3-\$27BA	\$27C4-\$27CB	\$27C6-\$27D0	\$27C5-\$27CD
\$27BB-\$27C0	\$27CC-\$27D1	\$27D1-\$27D5	\$27CE-\$27D5
\$27C1-\$27C5	\$27D2-\$27D6	\$27D6-\$27DF	\$27D6-\$27DD
\$27C6-\$27C9	\$27D7-\$27DA	\$27E0-\$27E5	\$27DE-\$27E3
\$27CA-\$27D2	\$27DB-\$27E3	\$27E6-\$27EE	\$27E4-\$27EC
\$27D3-\$27DB	\$27E4-\$27EC	\$27EF-\$27F7	\$27ED-\$27F5
\$27DC-\$27E5	\$27ED-\$27F6	\$27F8-\$2803	\$27F6-\$2800
\$27E6-\$27E9	\$27F7-\$27FA	\$2804-\$2809	\$2801-\$2806
\$27EA-\$27EB	\$27FB-\$27FC	\$280A-\$280B	\$2807-\$2808
\$27EC	\$27FD	\$280C	\$2809
\$27ED-\$27F3	\$27FE-\$2804	\$280D-\$2813	\$280A-\$2810
\$27F4-\$2810	\$2805-\$283C	\$2814-\$2851	\$2811-\$284E
\$2811-\$2815	\$283D-\$2841	\$2852-\$2856	\$284F-\$2853
\$2816-\$281B			
\$281C-\$3854	\$2842-\$387A	\$2857-\$388F	\$2854-\$388C
\$3855-\$3859			
\$385A-\$3BE8	\$387B-\$3C09	\$3890-\$3C1E	\$388D-\$3C1B
\$3BE9-\$3C62			
\$3C63-\$3FFE	\$3C0A-\$3FA5	\$3C1F-\$3FBA	\$3C1C-\$3FB7
	\$3FA6-\$3FFE	\$3FBB-\$3FFE	\$3FB8-\$38FE
\$3FFF	\$3FFF	\$3FFF	\$3FFF