The Complete
SPECTRUM
ROM
DISASSEMBLY

BY

Dr Ian Logan & Dr Frank O’Hara

Transcribed by the following readers of
the comp.sys.sinclair newsgroup:-

J.R. Biesma
Biggo
Dr. J. Bland
Paul E .Collins
Chris Cowley
Dr. Rupert Goodwins
Jonathan G Harston
Marcus Lund
Joe Mackay
Russell Marks
Eduardo Yañez Parareda
Adam Stonehewer
Mark Street
Gerard Sweeney
Geoff Wearmouth
Matthew Westcott
Matthew Wilson
Witchy
Preface

The Sinclair ZX Spectrum is a worthy successor to the ZX 81 which in turn replaced the ZX 80. The Spectrum has a 16K monitor program. This program has been developed directly from the 4K program of the ZX 80 although there are now so many new features that the differences outweigh the similarities. We have both enjoyed producing this book. We have learnt a great deal about the techniques of Z80 machine code programming and now feel that between us we have unravelled the ‘secrets of the Spectrum’.

We would like to thank:

-- Our families.
-- Alfred Milgrom, our publisher who has been extremely helpful.
-- Philip Mitchell whose notes on the cassette format were most informative.
-- Clive Sinclair and his team at Sinclair Research Ltd. who have produced such a ‘challenging’ and useful machine.

January 1983

Ian Logan Lincoln, U.K.
Frank O’Hara London, U.K.
Contents

Preface
Introduction
The DISASSEMBLY
- The restart routines and tables 1
- The keyboard routines 5
- The loudspeaker routines 11
- The cassette handling routines 15
- The screen and printer handling routines 33
- The executive routines 59
- BASIC line and command interpretation 84
- Expression evaluation 127
- The arithmetic routines 164
- The floating-point calculator 190

Appendix
- BASIC programs for the main series 222 (SIN X, EXP X, LN X & ATN X)
- The ‘DRAW’ algorithm 228
- The ‘CIRCLE’ algorithm 228
- Note on small integers and -65536 229

Index to routines 231
Introduction

The 16K monitor program of the Spectrum is a complex Z80 machine code program. Its overall structure is very clear in that it is divided into three major parts:

a. Input/Output routines
b. BASIC interpreter
c. Expression handling

However these blocks are too large to be managed easily and in this book the monitor program is discussed in ten parts. Each of these parts will now be 'outlined'.

The restart routines and tables.
At the start of the monitor program are the various 'restart' routines that are called with the single byte ‘RST’ instructions. All of the restarts are used. For example ‘restart 0008’ is used for the reporting of syntax or run-time errors. The tables in this part of the monitor program hold the expanded forms of the tokens and the 'key-codes'.

The keyboard routine.
The keyboard is scanned every 1/50 th. of a second (U.K. model) and the keyboard routine returns the required character code. All of the keys of the keyboard 'repeat' if they are held down and the keyboard routine takes this into consideration.

The loudspeaker routines.
The Spectrum has a single on-board loudspeaker and a note is produced by repeatedly using the appropriate 'OUT' instruction. In the controller routine great care has been taken to ensure that the note is held at a given 'pitch' throughout its 'duration'.

The cassette handling routines.
It was a very unfortunate feature of the ZX 81 that so little of the monitor program for that machine was devoted to the cassette handling. However in the Spectrum there is an extensive block of code and now the high standard of cassette handling is one of the most successful features of the machine.

BASIC programs or blocks of data are both dealt with in the same manner of having a 'header' block (seventeen bytes) that is SAVEd first. This 'header' describes the 'data block' that is SAVEd after it. One disadvantage of this system is that it is not possible to produce programs with any 'security' whatsoever.

The screen and printer handling routines.
All of the remaining input/output routines of the Spectrum are 'vectored' through the 'channel & stream information areas'. In the standard Spectrum 'input' is only possible from the keyboard but 'output' can be directed to the printer, the upper part of the T.V. display or the lower part of the T.V. display.

The major 'input' routine in this part of the monitor program is the EDITOR that allows the user to enter characters into the lower part of the T.V. display. The PRINT-OUT routine is a rather slow routine as the same routine is used for 'all possibilities'. For example, the adding of a single byte to the 'display area' involves considering the present status of OVER and INVERSE on every occasion.

The executive routines
In this part of the monitor program are to be found the INITIALISATION procedure and the 'main execution loop' of the BASIC interpreter. In the Spectrum the BASIC line returned by the EDITOR is checked for the correctness of its syntax and then saved in the program area, if it was a line starting with a line number, or 'executed' otherwise.

This execution can in turn lead to further statements being considered. (Most clearly seen as in the case of - RUN.)
BASIC line and command interpretation.
This part of the monitor program considers a BASIC line as a set of statements and in turn each statement as starting with a particular command. For each command there is a 'command routine' and it is the execution of the machine code in the appropriate 'command routine' that effects the 'interpretation'.

Expression evaluation
The Spectrum has a most comprehensive expression evaluator allowing for a wide range of variable types, functions and operations. Once again this part of the monitor is fairly slow as all the possible alternatives have to be considered. The handling of strings is particularly well managed. All simple strings are managed 'dynamically' and old copies are 'reclaimed' once they are redundant. This means that there is no 'garbage collecting' to be done.

The arithmetic routines
The Spectrum has two forms for numbers. Integer values in the range -65535 to +65535 are in an 'integral' or 'short' form whilst all other numbers are in a five byte floating point form. The present version of the monitor is unfortunately marred by two mistakes in this part.

i. There is a mistake in 'division' whereby the 34th bit of a division is lost.

ii. The value of -65536 is sometimes put in 'short' form and at other times in 'floating-point' and this leads to troubles.

The floating-point calculator
The CALCULATOR of the Spectrum handles numbers and strings and its operations are specified by 'literals'. It can therefore be considered that there is an internal 'stack operating' language in the CALCULATOR. This part of the monitor program contains routines for all the mathematical functions. The approximations to SIN X, EXP X, LN X & ATN X are obtained by developing Chebyshev polynomials and full details are given in the appendix.

Overall the 16K monitor program offers an extremely wide range of different BASIC commands and functions. The programmers have always however been short of 'room' and hence the program is written for 'compactness' rather than 'speed'.
THE DISASSEMBLY

THE RESTART ROUTINES and THE TABLES

THE 'START'
The maskable interrupt is disabled and the DE register pair set to hold the 'top of possible RAM'.

<table>
<thead>
<tr>
<th>0000</th>
<th>START</th>
<th>DI</th>
<th>Disable the 'keyboard interrupt'.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>XOR</td>
<td>+00 for start (but +FF for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
<td>'NEW').</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LD</td>
<td>DE,+FFFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JP</td>
<td>11CB,START/NEW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Jump forward.</td>
</tr>
</tbody>
</table>

THE 'ERROR' RESTART
The error pointer is made to point to the position of the error.

<table>
<thead>
<tr>
<th>0008</th>
<th>ERROR-1</th>
<th>LD</th>
<th>HL,(CH-ADD)</th>
<th>The address reached by the interpreter is copied to the error pointer before proceeding.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LD</td>
<td>(X-PTR),HL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>JP</td>
<td>0053,ERROR-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE 'PRINT A CHARACTER' RESTART
The A register holds the code of the character that is to be printed.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>REF</td>
<td>+FF,+FF,+FF,+FF</td>
<td>Unused locations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE 'COLLECT CHARACTER' RESTART
The contents of the location currently addressed by CH-ADD are fetched. A return is made if the value represents a printable character, otherwise CH-ADD is incremented and the tests repeated.

<table>
<thead>
<tr>
<th>0018</th>
<th>GET-CHAR</th>
<th>LD</th>
<th>HL,(CH-ADD)</th>
<th>Fetch the value that is addressed by CH-ADD.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LD</td>
<td>A,(HL)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>001C</td>
<td>TEST-CHAR</td>
<td>CALL</td>
<td>007D,SKIP-OVER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RET</td>
<td>NC</td>
<td></td>
</tr>
</tbody>
</table>

THE 'COLLECT NEXT CHARACTER' RESTART
As a BASIC line is interpreted, this routine is called repeatedly to step along the line.

<table>
<thead>
<tr>
<th>0020</th>
<th>NEXT-CHAR</th>
<th>CALL</th>
<th>0074,CH-ADD+1</th>
<th>CH-ADD needs to be incremented.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>JR</td>
<td>001C,TEST-CHAR</td>
<td>Jump back to test the new value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REF</td>
<td>+FF,+FF,+FF</td>
<td>Unused locations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE 'CALCULATOR' RESTART
The floating point calculator is entered at 335B.

<table>
<thead>
<tr>
<th>0028</th>
<th>FP-CALC</th>
<th>JP</th>
<th>335B,CALCULATE</th>
<th>Jump forward immediately.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>REF</td>
<td>+FF,+FF,+FF,+FF</td>
<td>Unused locations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE 'MAKE BC SPACES' RESTART
This routine creates free locations in the work space. The number of locations is determined by the current contents of the BC register pair.

<table>
<thead>
<tr>
<th>0030</th>
<th>BC-SPACES</th>
<th>PUSH</th>
<th>BC</th>
<th>Save the 'number'.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LD</td>
<td>HL,(WORKSP)</td>
<td>Fetch the present address of the start of the work space and save that also before proceeding.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PUSH</td>
<td>HL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>JP</td>
<td>169E,RESERVE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE 'MASKABLE INTERRUPT' ROUTINE
The real time clock is incremented and the keyboard scanned whenever a maskable interrupt occurs.

<table>
<thead>
<tr>
<th>0038</th>
<th>MASK-INT</th>
<th>PUSH</th>
<th>AF</th>
<th>Save the current values held in these registers.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PUSH</td>
<td>HL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LD</td>
<td>HL,(FRAMES)</td>
<td>The lower two bytes of the</td>
</tr>
</tbody>
</table>
INC HL frame counter are incremented
LD FRAMES,HL every 20 ms. (U.K.) The highest
LD A,H byte of the frame counter is
OR L only incremented when the
JR NZ,0048,KEY-INT value of the lower two bytes
INC (FRAMES-3) is zero.
0048 KEY-INT
LD BC Save the current values held
CALL 02BF,KEYBOARD in these registers.
PUSH DE Now scan the keyboard.
POP BC Restore the values.
POP HL
POP AF
EI The maskable interrupt is en-
RET abled before returning.

THE 'ERROR-2' ROUTINE
The return address to the interpreter points to the 'DEFB' that signifies which error has occurred. This 'DEFB' is fetched and transferred to ERR-NR. The machine stack is cleared before jumping forward to clear the calculator stack.
0053 ERROR-2
LD L,HL The address on the stack points
to the error code.
0055 ERROR-3
LD (ERR-NR),L It is transferred to ERR-NR.
OR L must both be zero for the reset to occur.
JR NZ,0070,NO_RESET Note: This should have been 'JR Z'!
JP 16C5,SET-STK exiting via SET-STK.
DEFB +FF,+FF,+FF,+FF Unused locations.
DEFB +FF,+FF,+FF

THE 'NON-MASKABLE INTERRUPT' ROUTINE
This routine is not used in the standard Spectrum but the code allows for a system reset to occur following activation of the NMI line. The system variable at 5CB0, named here NMIADD, has to have the value zero for the reset to occur.
0066 RESET
PUSH AF Save the current values held
PUSH HL in these registers.
LD HL,(NMIADD) The two bytes of NMIADD
LD A,H must both be zero for the reset to occur.
OR L
JR NZ,0070,NO_RESET Note: This should have been 'JR Z'!
JP (HL) Jump to START.
0070 NO-RESET
POP HL Restore the current values to
POP AF these registers and return.
RETN

THE 'CH-ADD+1' SUBROUTINE
The address held in CH-ADD is fetched, incremented and restored. The contents of the location now addressed by CH-ADD is fetched. The entry points of TEMP-PTR1 and TEMP-PTR2 are used to set CH-ADD for a temporary period.
0074 CH-ADD+1
LD HL,(CH-ADD) Fetch the address.
0077 TEMP-PTR1 INC HL Increment the pointer.
0078 TEMP-PTR2 LD (CH-ADD),HL Set CH-ADD.
LD A,(HL) Fetch he addressed value and
RET then return.

THE 'SKIP-OVER' SUBROUTINE
The value brought to the subroutine in the A register is tested to see if it is printable. Various special codes lead to HL being incremented once, or twice, and CH-ADD amended accordingly.
007D SKIP-OVER
CP +21 Return with the carry flag reset
RET NC if ordinary character code.
CP +0D Return if the end of the line
RET Z has been reached.
CP +10 Return with codes +00 to +0F
THE TOKEN TABLE
All the tokens used by the Spectrum are expanded by reference to this table. The last code of each token is 'inverted' by having its bit 7 set.

0095 BF 52 4E C4 49 4E 4B 45 '?' R N 'D' I N K E
009D 59 44 50 C9 46 CE 50 4F Y 'S' P 'I' F 'N' P O
00A5 49 4E D4 53 43 52 45 45 I N 'T' S C R E E
00AD 4E A4 41 54 52 D4 41 D4 N 'S' A T T 'R' A 'T'
00B5 4C 52 46 51 44 C4 43 43 T A 'B' V A L 'S' C
00BD 4F 44 C5 56 41 CC 4C 45 O 'D' E' V A 'L' L 'E
00CC CE 53 49 CE 43 4F D4 53 49 'N' S I 'N' C O 'S' T
00CD 41 CE 41 53 CE 41 43 D3 A 'N' 'T' S Q 'E' S G
00DE 49 CE 41 CE 41 43 4B 45 D3 'N' A B 'S' P E E 'K'
00DF 4F 44 C5 54 48 45 CE 54 CF N 'E' T H E 'N' T 'O'
00E5 53 54 45 D0 44 45 46 20 S T E 'P' D E F
00EF 46 CE 43 41 D4 46 4F 52 F 'N' C A 'T' F O R
0105 4D 41 D4 4D 4F 56 C5 45 M A 'T' N O 'V' 'E' I
010D 52 41 53 C5 4F 50 45 4E R A S 'E' O P E N
0115 20 A3 43 4C 4F 53 45 20 'C' L O S E
011D 4E A4 43 48 52 A4 4E 4F D4 'S' C H R 'S' N O 'T'
011F 49 CE 4F D2 41 4E C4 B I 'N' O 'R' A N 'D'
0125 3C BD 3E BD 3C BD 4C 49 < '=' > '=' < 'L' L I
012D 49 CE 55 53 D2 53 54 52 I 'N' U S 'R' S T R
012F A4 43 48 52 A4 4E 4F D4 'S' C H R 'S' N O 'T'
0135 49 CE 4F D2 41 4E C4 B I 'N' O 'R' A N 'D'
013D 59 44 50 C9 46 CE 50 4F Y 'S' P 'I' F 'N' P O
013F 49 4E D4 53 43 52 45 45 I N 'T' S C R E E
0145 4E 54 49 4E 55 CE 44 43 43 T A 'B' V A L 'S' C
014D 4F 44 C5 56 41 CC 4C 45 O 'D' E' V A 'L' L 'E
0155 53 54 45 D0 44 45 46 20 S T E 'P' D E F
015D 46 CE 43 41 D4 46 4F 52 F 'N' C A 'T' F O R
0165 4D 41 D4 4D 4F 56 C5 45 M A 'T' N O 'V' 'E' I
016D 52 41 53 C5 4F 50 45 4E R A S 'E' O P E N
0175 50 52 49 4E 56 45 52 53 C5 'T' I N V E R S 'E'
017D 53 D4 53 4F D0 52 45 S 'T' S T O 'P' F R E
0185 41 C4 44 41 54 C1 52 45 A 'D' D A T 'A' R E
018D 53 4F 52 C5 4E 45 D7 S T O 'R' E 'N' E 'W'
0195 42 4F 52 44 45 D2 43 4F B O R D E R 'C' O
019D 4F 54 49 4E 55 C5 44 49 N T I N U 'E' D I
01A5 CD 52 45 CD 46 4F D2 47 'M' R E 'M' F O 'R' G
01A7 50 52 49 4E 56 45 52 53 C5 'T' I N V E R S 'E'
01A9 4E 54 49 4E 56 45 52 53 C5 'T' I N V E R S 'E'
01AD 4F 20 54 CF 47 4F 20 53 O 'T' O 'G' O S
01B5 55 C2 49 4E 50 55 54 4C U 'B' I N F U 'T' L
01BD 4F 41 C4 4C 49 53 D4 4C O 'A' D 'L' I S 'T' L
01C5 45 D4 50 41 55 53 C5 4E E 'T' P A U S 'E' N
01CD 45 58 D4 50 4F 4B C5 50 E X 'T' P O K 'E' F
01D5 52 49 4E 54 D4 50 4C 4F D4 R I N 'T' P L O 'T'
01DD 52 55 CE 53 41 56 C5 52 R U 'N' S A Y 'E' R
01E5 41 4E 44 4F 4D 49 5A C5 A N D O M I Z 'E'
01ED 49 D3 4C 43 D3 44 52 41 I 'F' C L 'S' D B A
01F5 D7 43 4C 45 D1 52 45 'W' C L E A 'R' 'E'
01FD 54 55 52 CE 43 4F 50 D9 T U R 'N' C O P 'Y'
THE KEY TABLES
There are six separate key tables. The final character code obtained depends on the particular key pressed and the 'mode' being used.

(a) The main key table - L mode and CAPS SHIFT.

<table>
<thead>
<tr>
<th>Key Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0205</td>
<td>B</td>
</tr>
<tr>
<td>0206</td>
<td>H</td>
</tr>
<tr>
<td>0207</td>
<td>Y</td>
</tr>
<tr>
<td>0208</td>
<td>T</td>
</tr>
<tr>
<td>0209</td>
<td>G</td>
</tr>
<tr>
<td>0210</td>
<td>V</td>
</tr>
<tr>
<td>0211</td>
<td>M</td>
</tr>
<tr>
<td>0212</td>
<td>K</td>
</tr>
<tr>
<td>0213</td>
<td>I</td>
</tr>
<tr>
<td>0214</td>
<td>E</td>
</tr>
<tr>
<td>0215</td>
<td>X</td>
</tr>
<tr>
<td>0216</td>
<td>T</td>
</tr>
<tr>
<td>0217</td>
<td>G</td>
</tr>
<tr>
<td>0218</td>
<td>V</td>
</tr>
<tr>
<td>0219</td>
<td>M</td>
</tr>
<tr>
<td>0220</td>
<td>K</td>
</tr>
<tr>
<td>0221</td>
<td>I</td>
</tr>
<tr>
<td>0222</td>
<td>E</td>
</tr>
<tr>
<td>0223</td>
<td>X</td>
</tr>
<tr>
<td>0224</td>
<td>T</td>
</tr>
<tr>
<td>0225</td>
<td>G</td>
</tr>
<tr>
<td>0226</td>
<td>V</td>
</tr>
<tr>
<td>0227</td>
<td>M</td>
</tr>
<tr>
<td>0228</td>
<td>K</td>
</tr>
<tr>
<td>0229</td>
<td>I</td>
</tr>
<tr>
<td>0230</td>
<td>E</td>
</tr>
<tr>
<td>0231</td>
<td>X</td>
</tr>
<tr>
<td>0232</td>
<td>T</td>
</tr>
<tr>
<td>0233</td>
<td>G</td>
</tr>
<tr>
<td>0234</td>
<td>V</td>
</tr>
<tr>
<td>0235</td>
<td>M</td>
</tr>
<tr>
<td>0236</td>
<td>K</td>
</tr>
<tr>
<td>0237</td>
<td>I</td>
</tr>
<tr>
<td>0238</td>
<td>E</td>
</tr>
<tr>
<td>0239</td>
<td>X</td>
</tr>
<tr>
<td>0240</td>
<td>T</td>
</tr>
<tr>
<td>0241</td>
<td>G</td>
</tr>
<tr>
<td>0242</td>
<td>V</td>
</tr>
<tr>
<td>0243</td>
<td>M</td>
</tr>
<tr>
<td>0244</td>
<td>K</td>
</tr>
<tr>
<td>0245</td>
<td>I</td>
</tr>
<tr>
<td>0246</td>
<td>E</td>
</tr>
</tbody>
</table>

(b) Extended mode. Letter keys and unshifted.

<table>
<thead>
<tr>
<th>Key Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0247</td>
<td>~</td>
</tr>
<tr>
<td>0248</td>
<td>BRIGHT</td>
</tr>
<tr>
<td>0249</td>
<td>PAPER</td>
</tr>
<tr>
<td>024A</td>
<td>\</td>
</tr>
<tr>
<td>024B</td>
<td>ATN</td>
</tr>
<tr>
<td>024C</td>
<td>{</td>
</tr>
<tr>
<td>024D</td>
<td>CIRCLE</td>
</tr>
<tr>
<td>024E</td>
<td>}</td>
</tr>
<tr>
<td>024F</td>
<td>CIRCLE</td>
</tr>
</tbody>
</table>

(c) Extended mode. Letter keys and either shift.

<table>
<thead>
<tr>
<th>Key Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0250</td>
<td>-</td>
</tr>
<tr>
<td>0251</td>
<td>+</td>
</tr>
<tr>
<td>0252</td>
<td>=</td>
</tr>
<tr>
<td>0253</td>
<td>;</td>
</tr>
<tr>
<td>0254</td>
<td>&quot;</td>
</tr>
<tr>
<td>0255</td>
<td>$</td>
</tr>
<tr>
<td>0256</td>
<td>END</td>
</tr>
<tr>
<td>0257</td>
<td>END</td>
</tr>
</tbody>
</table>

(d) Control codes. Digit keys and CAPS SHIFT.

<table>
<thead>
<tr>
<th>Key Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0258</td>
<td>DELETE</td>
</tr>
<tr>
<td>0259</td>
<td>EDIT</td>
</tr>
<tr>
<td>025A</td>
<td>CAPS LOCK</td>
</tr>
<tr>
<td>025B</td>
<td>TRUE VIDEO</td>
</tr>
</tbody>
</table>

(e) Symbol code. Letter keys and symbol shift.

<table>
<thead>
<tr>
<th>Key Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0260</td>
<td>STEP</td>
</tr>
<tr>
<td>0261</td>
<td>STEP</td>
</tr>
<tr>
<td>0262</td>
<td>STEP</td>
</tr>
<tr>
<td>0263</td>
<td>STEP</td>
</tr>
<tr>
<td>0264</td>
<td>STEP</td>
</tr>
<tr>
<td>0265</td>
<td>STEP</td>
</tr>
<tr>
<td>0266</td>
<td>STEP</td>
</tr>
<tr>
<td>0267</td>
<td>STEP</td>
</tr>
<tr>
<td>0268</td>
<td>STEP</td>
</tr>
<tr>
<td>0269</td>
<td>STEP</td>
</tr>
</tbody>
</table>

(f) Extended mode. Digit keys and symbol shift.

<table>
<thead>
<tr>
<th>Key Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0270</td>
<td>FORMAT</td>
</tr>
<tr>
<td>0271</td>
<td>DEF FN</td>
</tr>
<tr>
<td>0272</td>
<td>FN</td>
</tr>
<tr>
<td>0273</td>
<td>LINE</td>
</tr>
<tr>
<td>0274</td>
<td>OPEN</td>
</tr>
<tr>
<td>0275</td>
<td>CLOSE</td>
</tr>
<tr>
<td>0276</td>
<td>MOVE</td>
</tr>
<tr>
<td>0277</td>
<td>ERASE</td>
</tr>
<tr>
<td>0278</td>
<td>POINT</td>
</tr>
<tr>
<td>0279</td>
<td>CAT</td>
</tr>
</tbody>
</table>
THE KEYBOARD ROUTINES

THE 'KEYBOARD SCANNING' SUBROUTINE

This very important subroutine is called by both the main keyboard subroutine and the INKEY$ routine (in SCANNING). In all instances the E register is returned with a value in the range of +00 to +27, the value being different for each of the forty keys of the keyboard, or the value +FF, the no-key.

The D register is returned with a value that indicates which single shift key is being pressed. If both shift keys are being pressed then the D and E registers are returned with the values for the CAPS SHIFT and SYMBOL SHIFT keys respectively.

The zero flag is returned reset if more than two keys are being pressed, or neither key of a pair of keys is a shift key.

028E KEY-SCAN          LD L,+2F The initial key value for each line will be +2F, +2E,...,+28.
                      (Eight lines.)
                      LD DE,+FFFF Initialise DE to 'no-key'.
                      LD BC,+FEFE C = port address, B = counter.

Now enter a loop. Eight passes are made with each pass having a different initial key value and scanning a different line of five keys. (The first line is CAPS SHIFT, Z, X, C, V.)

0296 KEY-LINE          IN A,(C) Read from the port specified.
                      CPL A pressed key in the line will set its respective bit (from bit 0 - outer key, to bit 4 - inner key).
                      AND +1F
                      JR Z,02AB,KEY-DONE Jump forward if none of the five keys in the line are being pressed.
                      LD H,A The key-bits go to the H register whilst the initial key value is fetched.
                      LD A,L
029F KEY-3KEYS         INC D If three keys are being pressed on the keyboard then the D register will no longer hold +FF - so return if this happens.
                      RET NZ
02A1 KEY-BITS          SUB +08 Repeatedly subtract '8' from the preset key value until a key-bit is found.
                      SRL H
                      JR NC,02A1,KEY-BITS Copy any earlier key value to the D register.
                      LD D,E
                      LD E,A Pass the new key value to the E register.
                      JR NZ,029F,KEY-3KEYS If there is a second, or possibly a third, pressed key in this line then jump back.
02AB KEY-DONE          DEC L The line has been scanned so the initial key value is reduced for the next pass.
                      RLC B The counter is shifted and the jump taken if there are still lines to be scanned.
                      JR C,0296,KEY-LINE

Four tests are now made.

02AB KEY-DONE          LD A,D Accept any key value for a pair of keys if the 'D' key is CAPS SHIFT.
                      RET Z

   Four tests are now made.
THE ‘KEYBOARD’ SUBROUTINE

This subroutine is called on every occasion that a maskable interrupt occurs. In normal operation this will happen once every 20 ms. The purpose of this subroutine is to scan the keyboard and decode the key value. The code produced will, if the 'repeat' status allows it, be passed to the system variable LAST-K. When a code is put into this system variable bit 5 of FLAGS is set to show that a 'new' key has been pressed.

02BF KEYBOARD CALL 028E,KEY-SCAN
RET NZ Fetch a key value in the DE register pair but return immediately if the zero pair flag is reset.

A double system of 'KSTATE system variables' (KSTATE0 - KSTATE3 and KSTATE4 - KSTATE7) is used from now on. The two sets allow for the detection of a new key being pressed (using one set) whilst still within the 'repeat period' of the previous key to have been pressed (details in the other set). A set will only become free to handle a new key if the key is held down for about 1/10 th of a second. i.e. Five calls to KEYBOARD.

02C6 K-ST-LOOP LD HL,KSTATE0 Start with KSTATE0.
BIT 7,(HL) Jump forward if a 'set is free';
JR NZ,02D1,K-CH-SET i.e. KSTATE0/4 holds +FF.
INC HL However if the set is not free
DEC (HL) decrease its '5 call counter'
DEC HL and when it reaches zero signal
JR NZ,02D1,K-CH-SET the set as free.
LD (HL),+FF

After considering the first set change the pointer and consider the second set.

02D1 K-CH-SET LD A,L Fetch the low byte of the
LD HL,+KSTATE4 address and jump back if the
CP L second set has still to be
JR NZ,02C6,K-ST-LOOP considered.

Return now if the key value indicates 'no-key' or a shift key only.

CALL 031E,K-TEST Make the necessary tests and
RET NC return if needed. Also change
the key value to a 'main code'.

A key stroke that is being repeated (held down) is now separated from a new key stroke.

LD HL,+KSTATE0 Look first at KSTATE0.
CP (HL) Jump forward if the codes
JR Z,0310,K-REPEAT match - indicating a repeat.
EX DE,HL Save the address of KSTATE0.
LD HL,+KSTATE4 Now look at KSTATE4.
CP (HL) Jump forward if the codes
JR Z,0310,K-REPEAT match - indicating a repeat.

But a new key will not be accepted unless one of the sets of KSTATE system variables is 'free'.

BIT 7,(HL) Consider the second set.
JR NZ,02F1,K-NEW Jump forward if 'free'.
EX DE,HL Now consider the first set.
Continue if the set is 'free' but exit from the KEYBOARD subroutine if not.

The new key is to be accepted. But before the system variable LAST-K can be filled, the KSTATE system variables, of the set being used, have to be initialised to handle any repeats and the key's code has to be decoded.

The code is passed to the E register and to KSTATE0/4.

The '5 call counter' for this set is reset to '5'.

The third system variable of the set holds the REPDEL value.

The REPDEL/REPPER value, and decrement it.

Exit from the KEYBOARD subroutine if the delay period has not passed.

However once it has passed the delay period for the next repeat is to be REPPER.

The repeat has been accepted so the final code value is fetched from KSTATE3/7 and passed to K-END.

Copy the shift byte.

Clear the D register for later.

Move the key number.

Return now if the key was 'CAPS SHIFT' only or 'no-key'.

THE 'REPEATING KEY' SUBROUTINE
A key will 'repeat' on the first occasion after the delay period - REPDEL (normally 0.7 secs.) and on subsequent occasions after the delay period - REPPER (normally 0.1 secs.).
The 'main code' is found by indexing into the main key table.

```
032C K-MAIN
LD HL,+0205 The base address of the table.
ADD HL,DE Index into the table and fetch the 'main code'.
SCF Signal 'valid keystroke'
RET before returning.
```

**THE "KEYBOARD DECODING" SUBROUTINE**

This subroutine is entered with the 'main code' in the E register, the value of FLAGS in the D register, the value of MODE in the C register and the 'shift byte' in the B register.

By considering these four values and referring, as necessary, to the six key tables a 'final code' is produced. This is returned in the A register.

```
0333 K-DECODE
LD A,E Copy the 'main code'.
CP +3A Jump forward if a digit key is being considered; also SPACE, ENTER & both shifts.
JR C,0367,K-DIGIT
DEC
JR M,0341,K-LET
ADD A,+4F Add the offset.
RET Return with the 'final code'.
```

Letter keys in extended mode are considered next.

```
0341 K-E-LET
LD HL,+01EB The base address for table 'b'.
INC B Jump forward to use this table if neither shift key is being pressed.
JR Z,034A,K-LOOK-UP
LD HL,+0205 Otherwise use the base address for table 'c'.
```

Key tables 'b'-f' are all served by the following look-up routine. In all cases a 'final code' is found and returned.

```
034A K-LOOK-UP
LD D,+00 Clear the D register.
ADD HL,DE Index the required table
LD A,(HL) and fetch the 'final code'.
RET Then return.
```

Letter keys in 'K', 'L' or 'C' modes are now considered. But first the special SYMBOL SHIFT codes have to be dealt with.

```
034F K-KLC-LET
LD HL,+0229 The base address for table 'e'
BIT 0,B Jump back if using the SYMBOL 'K' mode.
BIT 3,D Jump forward if currently in SHIFT key and a letter key.
JR Z,0364,K-TOKENS 'K' mode.
JR Z,0364,K-TOKENS 'K' mode.
BIT 3,(FLAGS2) If CAPS LOCK is set then return with the 'main code'
RET NZ return in the same manner
INC B Also return in the same manner
RET NZ if CAPS SHIFT is being pressed.
ADD A,+20 However if lower case codes are required then +20 has to be added to the 'main code' to give the correct 'final code'.
RET
```

The 'final code' values for tokens are found by adding +A5 to the 'main code'.

Add the required offset and return.

Next the digit keys; and SPACE, ENTER & both shifts; are considered.

Proceed only with the digit keys. i.e. Return with SPACE (+20), ENTER (+0D) & both shifts (+0E).

Now separate the digit keys into three groups - according to the mode.

Jump with ‘K’, ‘L’ & ‘C’ modes; and also with ‘G’ mode.

Continue with ‘E’ mode.

The base address for table ‘f’.

Use this table directly for both digit key ‘9’ that is to give GRAPHICS, and digit key ‘0’ that is to give DELETE.

For keys ‘1’ to ‘8’ make the range +80 to +87.

Return with a value from this range if neither shift key is being pressed.

But if ‘shifted’ make the range +88 to +8F.

The digit keys in graphics mode are to give the block graphic characters (+80 to +8F), the GRAPHICS code (+0F) and the DELETE code (+0C).

The digit keys ‘8’ and ‘9’ are to give ‘BRIGHT’ & ‘FLASH’ codes.

Reduce the range +30 to +37 giving +10 to +17.

Return with this ‘paper colour code’ if the CAPS SHIFT is not being used.

But if it is then the range is to be +18 to +1F instead - indicating an ‘ink colour code’.

The digit keys ‘0’ to ‘7’ in extended mode are to give either a ‘paper colour code’ or an ‘ink colour code’ depending on the use of the CAPS SHIFT.

The digit keys in extended mode are to give the block graphic characters (+80 to +8F), the GRAPHICS code (+0F) and the DELETE code (+0C).

Finally consider the digit keys in ‘K’, ‘L’ & ‘C’ modes.

Return directly if neither shift key is being used. (Final codes +30 to +39.)

Use table ‘d’ if the CAPS SHIFT key is also being pressed.
The codes for the various digit keys and SYMBOL SHIFT can now be found.

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB</td>
<td>+10</td>
<td>Reduce the range to give +20 to +29.</td>
</tr>
<tr>
<td>CP</td>
<td>+22</td>
<td>Separate the '@' character from the others.</td>
</tr>
<tr>
<td>JR</td>
<td>(+3B2, K)@-CHAR</td>
<td>The '-' character has also to be separated.</td>
</tr>
<tr>
<td>CP</td>
<td>+20</td>
<td></td>
</tr>
<tr>
<td>RET</td>
<td>NZ</td>
<td>Return now with the 'final codes' +21, +23 to +29.</td>
</tr>
<tr>
<td>LD</td>
<td>A,+5F</td>
<td>Give the '=' character a code of +5F.</td>
</tr>
<tr>
<td>RET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03B2</td>
<td>K-@-CHAR</td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>A,+40</td>
<td>Give the '@' character a code of +40.</td>
</tr>
</tbody>
</table>
THE LOUDSPEAKER ROUTINES

The two subroutines in this section are the BEEPER subroutine, that actually controls the loudspeaker, and the BEEP command routine. The loudspeaker is activated by having D4 low during an OUT instruction that is using port '254'. When D4 is high in a similar situation the loudspeaker is deactivated. A 'beep' can therefore be produced by regularly changing the level of D4.

Consider now the note 'middle C' which has the frequency 261.63 hz. In order to get this note the loudspeaker will have to be alternately activated and deactivated every 1/523.26 of a second. In the SPECTRUM the system clock is set to run at 3.5 mhz. and the note of 'middle C' will require that the requisite OUT instruction be executed as close as possible to every 6,689 T states. This last value, when reduced slightly for unavoidable overheads, represents the 'length of the timing loop' in the BEEPER subroutine.

THE 'BEEPER' SUBROUTINE

This subroutine is entered with the DE register pair holding the value 'f*t', where a note of given frequency 'f' is to have a duration of 't' seconds, and the HL register pair holding a value equal to the number of T states in the 'timing loop' divided by '4'. i.e. For the note 'middle C' to be produced for one second DE holds +0105 (INT(261.3 * 1)) and HL holds +066A (derived from 6,689/4 - 30.125).

03B5 BEEPER DI Disable the interrupt for the duration of a 'beep'.
LD A,L Save L temporarily.
SRL L Each '1' in the L register is
to count '4' T states, but take iNT (L/4) and count '16' T states instead.
CPL AND +03 Go back to the original value
LD C,A in L and find how many were lost by taking INT (L/4).
LD B,+00 LD IX,+03D1 The base address of the timing loop.
ADD IX,BC Alter the length of the timing loop. Use an earlier starting point for each '1' lost by taking INT (L/4).
LD A,(BORDCR) LD IX,BC
AND +38 Fetch the present border colour and move it to bits RRCA 2, 1 & 0 of the A register.
RRCA OR +08 Ensure the MIC output is 'off'.

Now enter the sound generation loop. 'DE' complete passes are made, i.e. a pass for each cycle of the note. The HL register holds the 'length of the timing loop' with '16' T states being used for each '1' in the L register and '1,024' T states for each '1' in the H register.

03D1 BE-IX+3 NOP Add '4' T states for each earlier entry port
03D2 BE-IX+2 NOP that is used.
03D3 BE-IX+1 NOP
03D4 BE-IX+0 INC B The values in the B & C registers
INC C will come from H & L registers - see below.
03D6 BE-H&L-LP DEC C The 'timing loop'.
JR NZ,03D6,BE-H&L-LP i.e. 'BC' * '4' T states.
LD C,+3F (But note that at the half-cycle
DEC B point - C will be equal to
JP NZ,03D6,BE-H&L-LP 'L+1'.)

The loudspeaker is now alternately activated and deactivated.
XOR +10 Flip bit 4.
Perform the OUT operation; leaving the border unchanged.

Reset the B register.

Save the A register.

Jump if at the half-cycle point.

After a full cycle the DE register pair is tested.

Jump forward if the last complete pass has been made already.

Fetch the saved value.

Reset the C register.

Decrease the pass counter.

Jump back to the required starting location of the loop.

The parameters for the second half-cycle are set up.

Reset the C register.

Add '16 T states as this path is shorter.

Jump back.

Upon completion of the 'beep' the maskable interrupt has to be enabled.

Enable interrupt.

Finally return.

THE 'BEEP' COMMAND ROUTINE

The subroutine is entered with two numbers on the calculator stack. The topmost number represents the 'pitch' of the note and the number underneath it represents the 'duration'.

The floating-point calculator is used to manipulate the two values - t & P.

Stack the decimal value 'K'.

A little below \(12 \times (2^{0.5}) - 1\)

Multiply t,pK

Addition t,pK+1

Now perform several tests on I, the integer part of the 'pitch'.

Fetch the exponent of I.

Give an error if i is not in the integral (short) form.

Copy the sign byte to the C register.

Copy the low-byte to the register.

Again give report B if i does not satisfy the test:

-128< i <= +127

Copy (HL)
Fetch the low-byte and test it further.

Accept -60 <= i <= 67.

Reject -128 to -61.

The correct frequency for the 'pitch' i can now be found.

Note: The range +70 to +127 will be rejected later on.

The final frequency f is found by modifying the 'last value' according to the octave number.

The number of complete cycles in the 'beep' is given by 'f*t' so this value is now found.

The result is left on the calculator stack whilst the length of the 'timing loop' required for the 'beep' is computed;
Note: The value '437,500/f' gives the 'half-cycle' length of the note and reducing it by '30.125' allows for '120.5' T states in which to actually produce the note and adjust the counters etc.

The values can now be transferred to the required registers.

CALL 1E99,FIND-INT2 The 'timing loop' value is compressed into the BC register pair; and saved.

Note: If the timing loop value is too large then an error will occur (returning via ERROR-1); thereby excluding 'pitch' values of '+70 to +127'.

CALL 1E99,FIND-INT2 The 't' value is compressed into the BC register pair.

PUSH BC
POP HL
LD D,B
LD E,C

However before making the 'beep' test the value 't'.

LD A,D
OR E
RET
DEC DE
JP 03B5,BEEPER

Report B - integer out of range

046C REPORT-B RST 0008,ERROR-1 Call the error handling routine.
DEFB +0A

THE 'SEMI-TONE' TABLE
This table holds the frequencies of the twelve semi-tones in an octave.

frequency hz. note

<table>
<thead>
<tr>
<th>frequency</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>261.63</td>
<td>C</td>
</tr>
<tr>
<td>277.18</td>
<td>C#</td>
</tr>
<tr>
<td>293.66</td>
<td>D</td>
</tr>
<tr>
<td>311.12</td>
<td>D#</td>
</tr>
<tr>
<td>329.63</td>
<td>E</td>
</tr>
<tr>
<td>349.23</td>
<td>F</td>
</tr>
<tr>
<td>369.99</td>
<td>F#</td>
</tr>
<tr>
<td>392</td>
<td>G</td>
</tr>
<tr>
<td>415.30</td>
<td>G#</td>
</tr>
<tr>
<td>440</td>
<td>A</td>
</tr>
<tr>
<td>466.16</td>
<td>A#</td>
</tr>
<tr>
<td>493.88</td>
<td>B</td>
</tr>
</tbody>
</table>

THE 'PROGRAM NAME' SUBROUTINE (ZX81)
The following subroutine applies to the ZX81 and was not removed when the program was rewritten for the SPECTRUM.

04AA DEF8 +CD,+FB,+24,+3A
DEF8 +3B,+3C,+87,+FA
DEF8 +8A,+1C,+E1,+D0
DEF8 +E5,+CD,+F1,+2B
DEF8 +62,+6B,+0D,+F8
DEF8 +09,+CB,+FE,+C9
THE CASSETTE HANDLING ROUTINES

The 16K monitor program has an extensive set of routines for handling the cassette interface. In effect these routines form the SAVE, LOAD, VERIFY & MERGE command routines.

The entry point to the routines is at SAVE-ETC (0605). However before this point are the subroutines concerned with the actual SAVEing and LOADing (or VERIFYing) of bytes.

In all cases the bytes to be handled by these subroutines are described by the DE register pair holding the 'length' of the block, the IX register pair holding the 'base address' and the A register holding +00 for a header block, or +FF for a program/data block.

THE 'SA-BYTES' SUBROUTINE

This subroutine is called to SAVE the header information (from 09BA) and later the actual program/data block (from 099E).

04C2 SA-BYTES
LD HL,+053F
PUSH HL
LD HL,+1F80
BIT 7,A
JR Z,04D0,SA-FLAG
LD HL,+0C98

04D0 SA-FLAG
EX AF,A'F'
INC DE
DEC IX
DI
LD A,+02
LD B,A
LD A,+02
INC DE
DEC IX
JR NZ,04D8,SA-LEADER

04D8 SA-LEADER
DJNZ 04D8,SA-LEADER
OUT (+FE),A
XOR +0F
LD B,+A4
DEC L
JR NZ,04D8,SA-LEADER
DEC B
DEC H
JP P,04D8,SA-LEADER

04EA SA-SYNC-1
DJNZ 04EA,SA-SYNC-1
LD B,+2F
OUT (+FE),A

04F2 SA-SYNC-2
DJNZ 04F2,SA-SYNC-2
LD A,+0D
LD B,+37
OUT (+FE),A

The header v. program/data flag will be the first byte to be SAVEEd.
LD BC,+3B0E
+3B is a timing constant; +0E
signals 'MIC off & YELLOW'.

EX AF,A'F'
Fetch the flag and pass it to the
L register for 'sending'.

LD L,A
JP 0507,SA-START
Jump forward into the SAVEing
loop.

The byte SAVEing loop is now entered. The first byte to be SAVEd is the flag; this is followed by the actual data byte and the final byte sent is the parity byte that is built up by considering the values of all the earlier bytes.

04FE SA-LOOP
LD A,D
The 'length' counter is tested
OR E
and the jump taken when it
JR Z,050E,SA-PARITY
has reached zero.
LD L,(IX+00)
Fetch the next byte that is to
be SAVEd.

0505 SA-LOOP-P
LD A,H
Fetch the current 'parity'.
XOR L
Include the present byte.

0507 SA-START
LD H,A
Restore the 'parity'. Note that
on entry here the 'flag' value
initialises 'parity'.

LD A,+01
Signal 'MIC on & BLUE'.
SCF
Set the carry flag. This will act
as a 'marker' for the 8 bits of a
byte.

JP 0525,SA-8-BITS
Jump forward.

When it is time to send the 'parity' byte then it is transferred to the L register for SAVEing.

050E SA-PARITY
LD L,H
Get final 'parity' value.
JR 0505,SA-LOOP-P
Jump back.

The following inner loop produces the actual pulses. The loop is entered at SA-BIT-1 with the type of the bit to be SAVEd indicated by the carry flag. Two passes of the loop are made for each bit thereby making an 'off pulse' and an 'on pulse'. The pulses for a reset bit are shorter by 855 T states.

0511 SA-BIT-2
LD A,C
Come here on the second pass
BIT 7,B
and fetch 'MIC off & YELLOW'.
Set the zero flag to show
'Second pass'.

0514 SA-BIT-1
DJNZ 0514,SA-BIT-1
The main timing loop; always
JR NC,051C,SA-OUT
801 T states on a 2nd. pass.
Jump, taking the shorter path, if
SAVEing a '0'.
LD B,+42
However if SAVEing a '1' then
add 855 T states.

051A SA-SET
DJNZ 051A,SA-SET
On the 1st. pass 'MIC on &
OUT (+FE),A
BLUE' and on the 2nd. pass
'MIC off & YELLOW'.
Set the timing constant for
the second pass.
LD B,+3E
Jump back at the end of the
JR NZ,0511,SA-BIT-2
first pass; otherwise reclaim
DEC B
13 T states.
XOR A
Clear the carry flag and set
INC A
A to hold +01 (MIC on & BLUE)
before continuing into the
'8 bit loop'.

The '8 bit loop' is entered initially with the whole byte in the L register and the carry flag set. However it is re-entered after each bit has been SAVEd until the point is reached when the 'marker' passes to the carry flag leaving the L register empty.

0525 SA-8-BITS
RL L
Move bit 7 to the carry and the
'marker' leftwards.
SAVE the bit unless finished with the byte.

Decrease the ‘counter’.

Advance the ‘base address’.

Set the timing constant for the first bit of the next byte.

Return (to SA/LD-RET) if the ‘parity’ byte.

BREAK key is being pressed.

Otherwise test the ‘counter’ and jump back even if it has reached zero (so as to send the ‘parity’ byte).

Exit when the ‘counter’ gives a short delay.

The ‘SA/LD-RET’ SUBROUTINE
This subroutine is common to both SAVEing and LOADing.
The border is set to its original colour and the BREAK key tested for a last time.

Save the carry flag. (It is reset after a LOADing error.)

Fetch the original border colour from its system variable.

Move the border colour to bits 2, 1 & 0.

Set the border to its original colour.

Read the BREAK key for a last time.

Enable the maskable interrupt.

Jump unless a break is to be made.

Call the error handling routine.

Retrieve the carry flag.

Return to the calling routine.

THE ’LD-BYTES’ SUBROUTINE
This subroutine is called to LOAD the header information (from 07BE) and later LOAD, or VERIFY, an actual block of data (from 0802).

This resets the zero flag. (D cannot hold +FF.)

The A register holds +00 for a header and +FF for a block of data.

The carry flag is reset for VERIFYing and set for LOADing.

Restore D to its original value.
DI  The maskable interrupt is now disabled.
LD  A,+0F  The border is made WHITE.
OUT (+FE),A
LD  HL,+053F  Preload the machine stack
with the address - SA/LD-RET.
PUSH HL
IN  A,(+FE)  Make an initial read of port '254'
RRA  Rotate the byte obtained but
AND +20  keep only the EAR bit,
OR +02  Signal 'RED' border.
LD  C,A  Store the value in the C register. -
         (+22 for 'off' and +02 for 'on'
         - the present EAR state.)
CP A  Set the zero flag.

The first stage of reading a tape involves showing that a pulsing signal actually exist (i.e. 'On/off' or 'off/on' edges.)

056B  LD-BREAK RET NZ  Return if the BREAK key is
       being pressed.

056C  LD-START CALL 05E7,LD-EDGE-1  Return with the carry flag reset
       if there is no 'edge' within approx. 14,000 T states. But if
       an 'edge' is found the border will go CYAN.

The next stage involves waiting a while and then showing that the signal is still pulsing.

0574  LD-WAIT DJNZ 0574,LD-WAIT  The length of this waiting
       period will be almost one
       second in duration.
       DEC HL
       OR L
       JR NZ,0574,LD-WAIT
       CALL 05E3,LD-EDGE-2  Continue only if two edges are
       found within the allowed time
       period.
       JR NC,056B,LD-BREAK

Now accept only a 'leader signal'.

0580  LD-LEADER LD B,+9C  The timing constant,
       CALL 05E3,LD-EDGE-2  Continue only if two edges are
       JR NC,056B,LD-BREAK  found within the allowed time
       period.
       LD A,+C6  However the edges must have
       CP B  been found within about
       JR NC,056C,LD-START  3,000 T states of each other
       INC H  Count the pair of edges in the H
       JR NZ,0580,LD-LEADER  register until '256' pairs have
       been found.

After the leader come the 'off' and 'on' part's of the sync pulse.

058F  LD-SYNC LD B,+9C  The timing constant.
       CALL 05E7,LD-EDGE-1  Every edge is considered until
       JR NC,056B,LD-BREAK  two edges are found close
       LD A,B  together - these will be the
       CP +04  start and finishing edges of
       JR NC,058F,LD-SYNC  the 'off' sync pulse.
       CALL 05E7,LD-EDGE-1  The finishing edge of the
       RET NC  'on' pulse must exist.
       (Return carry flag reset.)

The bytes of the header or the program/data block can now be LOADed or VERIFied. But the first byte is the type flag.

LD  A,C  The border colours from now
XOR +03  on will be BLUE & YELLOW.
LD C,A  Initialise the ‘parity matching’
LD H,+00 byte to zero.
LD B,+B0 Set the timing constant for the
flag byte.
JR 05C8,LD-MARKER Jump forward into the byte
LOADING loop.

The byte LOADing loop is used to fetch the bytes one at a time. The flag byte is first. This is followed by the data bytes and the last byte is the ‘parity’ byte.

05A9   LD-LOOP EX AF,‘A’ Fetch the flags.
       JR NZ,05B3,LD-FLAG Jump forward only when
           handling the first byte.
       JR NC,05BD,LD-VERIFY Jump forward if VERIFYing a
           tape.
       LD (IX+00),L Make the actual LOAD when
           required.
       JR 05C2,LD-NEXT Jump forward to LOAD the
           next byte.

05B3   LD-FLAG RL C Keep the carry flag in a safe
       XOR L place temporarily.
       RET NZ Return now if the type flag does
           not match the first byte on the
           tape. (Carry flag reset.)
       LD A,C Restore the carry flag now.
       RRA
       LD C,A
       INC DE Increase the counter to
       JR 05CA,LD-DEC compensate for its ‘decrease’
           after the jump.

If a data block is being verified then the freshly loaded byte is tested against the original byte.

05BD   LD-VERIFY LD A,(IX+00) Fetch the original byte.
       XOR L Match it against the new byte.
       RET NZ Return if ‘no match’. (Carry
           flag reset.)

A new byte can now be collected from the tape.

05C2   LD-NEXT INC IX Increase the ‘destination’.
05C4   LD-DEC DEC DE Decrease the ‘counter’.
       EX AF,‘A’ Save the flags.
       LD B,+B2 Set the timing constant.
05C8   LD-MARKER LD L,+01 Clear the ‘object’ register apart
       from a ‘marker’ bit.

The ‘LD-8-BITS’ loop is used to build up a byte in the L register.

05CA   LD-8-BITS CALL 05E3,LD-EDGE-2 Find the length of the ‘off’
           and ‘on’ pulses of the next bit.
           Return if the time period is
           exceeded. (Carry flag reset.)
       RET NC
       LD A,+C5 Compare the length against
           approx. 2,400 T states; resetting
           the carry flag for a ‘0’ and
           setting it for ‘1’.
       CP B
       RL L Include the new bit in the L
           register.
       LD B,+B0 Set the timing constant for the
           next bit.
       JP NC,05CA,LD-8-BITS Jump back whilst there are still
           bits to be fetched.

The ‘parity matching’ byte has to be updated with each new byte.
LD A,H
XOR L
LD H,A

Fetch the ‘parity matching’ byte and include the new byte.

Save it once again.

Passes round the loop are made until the ‘counter’ reaches zero. At that point the ‘parity matching’ byte should be holding zero.

LD A,D
OR E
JR NZ,05A9,LD-LOOP
LD A,H
CP +01
RET+00

Make a further pass if the DE register pair does not hold zero.

Fetch the ‘parity matching’ byte.

Return with the carry flat set if the value is zero.
(Carry flag reset if in error.)

THE ‘LD-EDGE-2’ AND ‘LD-EDGE-1’ SUBROUTINES
These two subroutines form the most important part of the LOAD/VERIFY operation.

The subroutines are entered with a timing constant in the B register, and the previous border colour and ‘edge-type’ in the C register.
The subroutines return with the carry flag set if the required number of ‘edges’ have been found in the time allowed; and the change to the value in the B register shows just how long it took to find the ‘edge(s)’.

The carry flag will be reset if there is an error. The zero flag then signals ‘BREAK pressed’ by being reset, or ‘time-up’ by being set.
The entry point LD-EDGE-2 is used when the length of a complete pulse is required and LD-EDGE-1 is used to find the time before the next ‘edge’.

05E3 LD-EDGE-2 CALL 05E7,LD-EDGE-1
RET NC

In effect call LD-EDGE-1 twice; returning in between if there is an error.

05E7 LD-EDGE-1 LD A,+16
05E9 LD-DELAY DEC A
JR NZ,05E9,LD-DELAY
AND A

Wait 358 T states before entering the sampling loop.

The sampling loop is now entered. The value in the B register is incremented for each pass; ‘time-up’ is given when B reaches zero.

05ED LD-SAMPLE INC B
RET Z

Count each pass.

Return carry reset & zero set if ‘time-up’.

LD A,+7F
IN A,(+FE)
RRA
RET NC

Read from port +7FFE.
i.e. BREAK & EAR.
Shift the byte.

Return carry reset & zero reset if BREAK was pressed.

XOR C
AND +20
JR Z,05ED,LD-SAMPLE

Now test the byte against the ‘last edge-type’; jump back unless it has changed.

A new ‘edge’ has been found within the time period allowed for the search. So change the border colour and set the carry flag.

LD A,C
CPL
LD C,A
AND +07
OR +08
OUT (+FE),A
SCF
RET

Change the ‘last edge-type’ and border colour.

Keep only the border colour.

Signal ‘MIC off’.

Change the border colour (RED/CYAN or BLUE/YELLOW).

Signal the successful search before returning.

Note: The LD-EDGE-1 subroutine takes 465 T states, plus an additional 58 T states for each unsuccessful pass around the sampling loop.
For example, therefore, when awaiting the sync pulse (see LD-SYNC at 058F) allowance is made for ten additional passes through the sampling loop. The search is thereby for the next edge to be found within, roughly, 1,100 T states (465 + 10 * 58 + overhead). This will prove successful for the sync 'off' pulse that comes after the long 'leader pulses'.

**THE 'SAVE, LOAD, VERIFY & MERGE' COMMAND ROUTINES**

The entry point SAVE-ETC is used for all four commands. The value held in T-ADDR however distinguishes between the four commands. The first part of the following routine is concerned with the construction of the 'header information' in the work space.

```
0605 SAVE-ETC POP AF Drop the address - SCAN-LOOP.
      LD A,(T-ADDR-lo) Reduce T-ADDR-lo by +E0;
      SUB +E0 giving +00 for SAVE, +01 for LOAD, +02 for VERIFY and +03 for MERGE.
      CALL 1C8C,EXPT-EXP Pass the parameters of the 'name' to the calculator stack.
      CALL 2530,SYNTAX-Z Jump forward if checking
      JR Z,0621,SA-SPACE syntax.
      LD BC,+0011 Allow seventeen locations
      LD A,(T-ADDR-lo) for the header of a SAVE but
      AND A thirty four for the other
      JR Z,0621,SA-SPACE commands.
      LD C,+22

0621 SA-SPACE RST 0030,BC-SPACES The required amount of space is
      PUSH DE made in the work space.
      POP IX Copy the start address to the
      POP IX register pair.
      LD B,+0B A program name can have
      OR A up to ten characters but
      JR Z,0652,SA-SPACE
      CALL 1C89,SYNT-EXP first enter eleven space
      CALL 0629,SA-BLANK characters into the prepared
      INC DE area.
      DJNZ (DE),A A null name is +FF only.
      CALL 25F1,STK-FETCH The parameters of the name
      LD (IX+01),+FF are fetched and its length is
      INC BC tested.
      DEC BC In effect jump forward if the
      ADD HL,BC length of the name is not
      INC BC too long. (i.e. No more than
      JR NC,064B,SA-NAME ten characters.)
      LD A,(T-ADDR-lo) But allow for the LOADing,
      AND A VERIFYing and MERGEing of
      JR NZ,0644,SA-NULL programs with 'null' names or
      INC BC extra long names.

0642 REPORT-F RST 0008,ERROR-1 Call the error handling
      DEFB +0E routine.

Continue to handle the name of the program.

0644 SA-NULR LD A,B Jump forward if the name
      OR C has a 'null' length.
      JR Z,0652,SA-DATA
      LD BC,+000A But truncate longer names.

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

064B SA-NAME PUSH IX Copy the start address to the
      POP HL HL register pair.
      INC HL Step to the second location.
      EX DE,HL Switch the pointers over and
      LDIR copy the name.
```
The many different parameters, if any, that follow the command are now considered. Start by handling 'xxx "name" DATA'.

0652 SA-DATA RST 0018.GET-CHAR Is the present code the
CP +E4 token 'DATA'?
JR NZ,06A0,SA-SCR$ Jump if not.
LD A,(T-ADDR-lo) However it is not possible
CP +03 to have 'MERGE name DATA'.
JP Z,1C8A,REPORT-C
call
RST 0020,NEXT-CHAR Advance CH-ADD.
CALL 28B2,LOOK-VARS Look in the variables area for
SET 7,C the array.
JR NC,0672,SA-V-OLD Jump if handling an existing
LD HL,+0000 array.
LD A,(T-ADDR-lo) Consider the value in T-ADDR
DEC A and give an error if trying to
JR Z,0685,SA-V-NEW SAVE or VERIFY a new array.

Report 2 - Variable not found

0670 REPORT-2 RST 0008,ERROR-1 Call the error handling
DEFB +01 routine.
Continue with the handling of an existing array.

0672 SA-V-OLD JP NZ,1C8A,REPORT-C Note: This fails to exclude
CALL 2530,SYNTAX-Z simple strings.
JR Z,0692,SA-DATA-1 Jump forward if checking
INC HL syntax.
LD A,(HL) Point to the 'low length' of the
LD (IX+0B),A variable.
INC HL The low length byte goes into
LD A,(HL) the work space; followed by
LD (IX+0C),A the high length byte.
INC HL Step past the length bytes.

The next part is common to both 'old' and 'new' arrays. Note: Syntax path error.

0685 SA-V-NEW LD (IX+0E),C Copy the array's name.
BIT A,+01 Assume an array of numbers.
JR Z,068F,SA-V-TYPE Jump if it is so.
INC A It is an array of characters.
068F SA-V-TYPE 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.

0692 SA-DATA-1 EX DE,HL Save the pointer in DE.
RST 0020,NEXT-CHAR Is the next character
CP +29 a ']' ?
JR NZ,0672,SA-OLD Give report C if it is not.
RST 0020,NEXT-CHAR Advance CH-ADD.
CALL 1BEE,CHECK-END Move on to the next statement
if checking syntax.
EX DE,HL Return the pointer to the HL
JP 075A,SA-ALL register pair before jumping
forward. (The pointer indicates
the start of an existing array's
contents.)

Now consider 'SCREEN$'.

06A0 SA-SCR$ CP +AA Is the present code the
token SCREEN$. 
Jump if not.

LD A,(T-ADDR-lo) However it is not possible to
CP +03 have 'MERGE name SCREEN$'.
JP Z,1C8A,REPORT-C Advance CH-ADD.
RST 0020,NEXT-CHAR Call 1BEE,CHECK-END Move on to the next statement if checking syntax.
LD (IX+OB),+00 The display area and the
LD (IX+0C),+1B attribute area occupy +1800 locations and these locations start at +4000; these details
LD HL,+4000 are passed to the header area
LD (IX+0E),H in the work space.
JR 0710,SA-TYPE-3 Jump forward.

Now consider 'CODE'.

06C3 SA-CODE CP +AF Is the present code the token 'CODE'? Jump if it is not.
JR NZ,0716,SA-JC Jump if not.
LD A,(T-ADDR-lo) However it is not possible to
CP +03 have 'MERGE name CODE'.
JP Z,1C8A,REPORT-C Advance CH-ADD.
RST 0020,NEXT-CHAR Jump forward if the statement
CALL 2048,PR-ST-END has not finished.
JR NZ,06E1,SA-CODE-1 Call 1C82,EXPT-1NUM Fetch the first number.
LD A,(T-ADDR-lo) However refuse 'SAVE name
AND A 'CODE' that does not have a
JP Z,1C8A,REPORT-C 'start' and a 'length'.
CALL 1CE6,USE-ZERO Put a zero on the calculator stack - for the 'start'.
JR 06F0,SA-CODE-2 Jump forward.

Look for a 'starting address'.

06E1 SA-CODE-1 CALL 1C82,EXPT-1NUM Fetch the first number.
RST 0018,GET-CHAR Is the present character a ',' or not?
CP +2C Jump if it is - the number was a 'starting address'.
JR Z,06F5,SA-CODE-3 Call 1C82,EXPT-1NUM Enter the 'type' number.
LD A,(T-ADDR-lo) However refuse 'SAVE name
AND A 'CODE' that does not have a
JP Z,1C8A,REPORT-C 'start' and a 'length'.
CALL 1CE6,USE-ZERO Put a zero on the calculator stack - for the 'length'.
JR 06F0,SA-CODE-4 Jump forward.

Fetch the 'length' as it was specified.

06F0 SA-CODE-2 CALL 1CE6,USE-ZERO Put a zero on the calculator stack - for the 'length'.
RST 0020,NEXT-CHAR Advance CH-ADD.
CALL 1C82,EXPT-1NUM Fetch the 'length'.

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

06F9 SA-CODE-4 CALL 1BEE,CHECK-END But move on to the next statement now if checking syntax.
CALL 1E99,FIND-INT2 Compress the 'length' into
LD (IX+OB),C the BC register pair and
LD (IX+0C),B store it.
CALL 1E99,FIND-INT2 Compress the 'starting address'
LD (IX+0D),C into the BC register pair
LD (IX+0E),B and store it.
LD H,B Transfer the 'pointer' to the
LD L,C HL register pair as usual.

'SCREENS' and 'CODE' are both of type 3.

0710 SA-TYPE-3 LD (IX+00),+03 Enter the 'type' number.
JR 075A,SA-ALL  Rejoin the other pathways.

Now consider 'LINE'; and 'no further parameters'.

0716  SA-LINE  CP  +CA  Is the present code the token 'LINE'?
JR  Z,0723,SA-LINE-1  Jump if it is.
CALL  1BEE,CHECK-END  Move on to the next statement if checking syntax.
LD  (IX+0E)+80  When there are no further parameters an +80 is entered.
JR  073A,SA-TYPE-0  Jump forward.

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

0723  SA-LINE-1  LD  A,(T-ADDR-lo)  However only allow 'SAVE and 'LOAD names.
AND  A  name LINE number'.
JP  NZ,1C8A,REPORT-C  Advance CH-ADD.
RST  0020,NEXT-Char  Pass the number to the calculator stack.
CALL  1C82,EXPT-1NUM  Move on to the next statement if checking syntax.
CALL  1BEE,CHECK-END  Compress the 'line number’
LD  (IX+0D),C  into the BC register pair
LD  (IX+0E),B  and store it.

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

073A  SA-TYPE-0  LD  (IX-00)+00  Enter the 'type' number.

The parameters that describe the program, and its variables, are found and stored in the header area of the work space.

LD  HL,(E-LINE)  The pointer to the end of the variables area.
LD  DE,(PROG)  The pointer to the start of the BASIC program.
SCF  Now perform the subtraction
SBC  HL,DE  to find the length of the
LD  (IX+0B),L  'program + variables'; store the result.
LD  (IX+0C),H  'program' only.
LD  HL,(VARS)  Repeat the operation but this time storing the length of the
SBC  HL,DE  'program' only.
LD  (IX+0F),L  Transfer the 'pointer' to the
LD  (IX+10),H  HL register pair as usual.
EX  DE,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.

075A  SA-ALL  LD  A,(T-ADDR-lo)  Jump forward when handling
AND  A  a SAVE command.
JP  Z,0970,SA-CONTRL

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.

PUSH  HL  Save the 'destination' pointer.
LD BC,+0011  Form in the IX register pair
ADD IX,BC the base address of the 'second

Now enter a loop; leaving it only when a 'header' has been LOADed.

0767  LD-LOOK-H PUSH IX Make a copy of the base address.
LD DE,+0011 LOAD seventeen bytes.
XOR A Signal 'header'.
SCF
CALL 0556,LD-BYTES New look for a header.
POP IX Retrieve the base address.
JR NC,0767,LD-LOOK-H Go round the loop until

The new 'header' is now displayed on the screen but the routine will only proceed if the 'new' header matches the 'old' header.

LD A,+FE Ensure that channel 'S'
CALL 1601,CHAN-OPEN is open.
LD (SCR-CT),+03 Set the scroll counter.
LD C,+80 Signal 'names do not match'.
LD A,.(IX+00) Compare the 'new' type
CP (IX-11) against the 'old' type.
JR NZ,078A,LD-TYPE Jump if the 'types' do not
LD C,+F6 match.
But if they do; signal 'ten
characters are to match'.
078A LD-TYPE CP +04 Clearly the 'header' is
JR NC,0767,LD-LOOK-H nonsense if 'type 4 or more'.

The appropriate message - 'Program:', 'Number array:', 'Character array:' or 'Bytes:' is printed.

LD DE,+09C0 The base address of the message
PUSH BC block.
CALL 0C0A,PO the appropriate message is
POP BC printed.

The 'new name' is printed and as this is done the 'old' and the 'new' names are compared.

PUSH IX Make the DE register pair
POP DE point to the 'new type' and
LD HL,+FF0 the HL register pair to the
ADD HL,DE 'old name'.
LD B,+0A Ten characters are to be
LD A.(HL) considered.
INC A Jump forward if the match is
INC HL to be against an actual name.
JR NZ,07A6,LD-NAME But if the 'old name' is 'null'
LD A,C A,B
ADD A,B then signal 'ten characters
LD C,A already match'.

A loop is entered to print the characters of the 'new name'. The name will be accepted if the 'counter' reaches zero, at least.

07A6 LD-NAME INC DE Consider each character of the
LD A,(DE) 'new name' in turn.
CP (HL) Match it against the appropriate
INC HL character of the 'old name'.
JR NZ,07AD,LD-CH-PR Do not count it if it does not
INC C does match.
07AD LD-CH-PR RST 0010,PRINT-A-1 Print the 'new' character.
DJNZ 07A6,LD-NAME Loop for ten characters.
BIT 7,C Accept the name only if the
JR NZ,0767,LD-LOOK-H counter has reached zero.
Follow the 'new name' with RST 0010, PRINT - A - 1 a 'carriage return'.

The correct header has been found and the time has come to consider the three commands LOAD, VERIFY, & MERGE separately.

**THE 'VERIFY' CONTROL ROUTINE**

The verification process involves the LOADing of a block of data, a byte at a time, but the bytes are not stored - only checked. This routine is also used to LOAD blocks of data that have been described with 'SCREEN$ & CODE'.

```
07CB VR-CTRL PUSH HL Save the 'pointer'.
  LD L,(IX-06) Fetch the 'number of bytes'
  LD H,(IX-05) as described in the 'old' header.
  LD E,(IX+0B) Fetch also the number from the
  LD D,(IX+0C) 'new' header.
  LD A,H Jump forward if the 'length' is
  OR L unspecified.
  JR Z,07E9,VR-CONT-1 e.g. 'LOAD name CODE' only.
  SBC HL,DE Give report R if attempting
  JR C,0806,REPORT-R to LOAD a larger block than has
  JR Z,07E9,VR-CONT-1 been requested.
  JR A,(IX+00) Also give report R if trying
  CP +03 to VERIFY blocks that are of
  JR NZ,0806,REPORT-R unequal size. ('Old length'
  LD A,(IX+00) greater than 'new length'.)
```

The routine continues by considering the 'destination pointer'.

```
07E9 VR-CONT-1 POP HL Fetch the 'pointer', i.e. the 'start'.
  LD A,H This 'pointer' will be used
  OR L unless it is zero, in which case the 'start' found in
  JR NZ,07F4,VR-CONT-2 the 'new' header will be used
  LD L,(IX+0D) instead.
  LD H,(IX+0E)
```

The VERIFY/LOAD flag is now considered and the actual LOAD made.

```
07F4 VR-CONT-2 PUSH HL Move the 'pointer' to the IX register pair.
  LD A,(T-ADDR-lo) Jump forward unless using
  CP +02 the VERIFY command; with
  SCF the carry flag signalling
  JR NZ,0800,VR-CONT-3 'LOAD'
  AND A Signal 'VERIFY'.
  0800 VR-CONT-3 LD A,+FF Signal 'accept data block only'
  LD A,+0D Call 0556,LD-BYTES LOAD/VERIFY a data block.
  RET C Return unless an error.
```

THE 'LOAD A DATA BLOCK' SUBROUTINE

This subroutine is common to all the 'LOADing' routines. In the case of LOAD & VERIFY it acts as a full return from the cassette handling routines but in the case of MERGE the data block has yet to be 'MERGEd'.

```
0802 LD-BLOCK CALL 0556,LD-BYTES LOAD/VERIFY a data block.
  RET C Return unless an error.
```
Report R - Tape loading error

0806 REPORT-R RST 0008,ERROR-1 Call the error handling routine.

THE 'LOAD' CONTROL ROUTINE
This routine controls the LOADING of a BASIC program, and its variables, or an array.

0808 LD-CONTRL LD E,(IX+0B) Fetch the 'number of bytes' as given in the 'new header'.
LD D,(IX+0C) Save the 'destination pointer'.
PUSH HL Jump forward unless trying to LOAD a previously undeclared array.
LD A,H Jump forward if LOADing a new array.
OR L Jump forward if LOADing a BASIC program.
JR NZ,0819,LD-CONT-1 undeclared array.
INC DE Add three bytes to the length - for the name, the low length & the high length of a new variable.
INC DE
INC DE
EX DE,HL
JR 0825,LD-CONT-2 Jump forward.

Consider now if there is enough room in memory for the new data block.

0819 LD-CONT-1 LD L,(IX-06) Fetch the size of the existing 'program+variables or array'.
LD H,(IX-05) 'program+variables or array'.
EX DE,HL Jump forward if no extra room will be required; taking into account the reclaiming of the presently used memory.
SCF
SBC HL,DE
JR C,082E,LD-DATA

Make the actual test for room.

0825 LD-CONT-2 LD DE,+0005 Allow an overhead of five bytes.
ADD HL,DE
LD B,H Move the result to the BC register pair and make the test.
LD C,L CALL 1F05,TEST-ROOM

Now deal with the LOADING of arrays.

082E LD-DATA POP HL Fetch the 'pointer' anew.
LD A,(IX+00) Jump forward if LOADING a BASIC program.
AND A Jump forward if LOADING a new array.
JR Z,0873,LD-PROG
LD A,H
OR L
JR Z,084C,LD-DATA-1
DEC HL Fetch the 'length' of the existing array by collecting the length bytes from the variables area.
DEC HL
LD B,(HL) Point to its old name.
LD C,(HL) Add three bytes to the length - one for the name and two for the 'length'.
DEC HL Save the IX register pair temporarily whilst the old array is reclaimed.
INC BC
INC BC CALL 19E8,RECLAIM-2
INC BC
LD (X-PTR),IX
LD IX,(X-PTR)

Space is now made available for the new array - at the end of the present variables area.

084C LD-DATA-1 LD HL,(E-LINE) Find the pointer to the end-marker of the variables area - the '80-byte'.
DEC HL
LD C,(IX+0B) Fetch the 'length' of the new array.
LD B,(IX+0C) Save this 'length'.
PUSH BC
INC BC  Add three bytes - one for
INC BC  the name and two for the
INC BC  ‘length’.
LD A,(IX-03)  ‘IX+0E’ of the old header
           gives the name of the array.
PUSH AF  The name is saved whilst the
CALL 1655,MAKE-ROOM appropriate amount of room is
INC HL  made available. In effect ‘BC’.
POP AF  spaces before the ‘new 80-byte’.
LD (HL),A  The name is entered.
POP DE  The ‘length’ is fetched and
INC HL  its two bytes are also
LD (HL),E  entered.
INC HL  HL now points to the first
LD (HL),D  location that is to be filled
INC HL  with data from the tape.
PUSH HL  This address is moved to the
POP IX  IX register pair; the carry
SCF  flag set; ‘data block’ is
LD A,+FF  signalled; and the block
JP 0802,LD-BLOCK LOADed.

Now deal with the LOADing of a BASIC program and its variables

0873 LD-PROG EX DE,HL  Save the ‘destination pointer’.
        LD HL,(E-LINE)  Find the address of the
        DEC HL  end-marker of the current
        LD (X-PTR),IX  variables area - the ‘80-byte’.
        LD B,(IX+0C)  Save IX temporarily.
        LD C,(IX+0B)  Fetch the ‘length’ of the
        PUSH BC  new data block.
        CALL 19E5,RECLAIM-1  Keep a copy of the ‘length
        POP BC  whilst the present program and
        PUSH HL  variables areas are reclaimed.
        PUSH BC  Save the pointer to the program
        CALL 1655,MAKE-ROOM  area and the length of the new
data block.
        LD IX,(X-PTR)  Make sufficient room available
        INC HL  for the new program and its
        LD B,(IX+10)  variables.
        ADD HL,BC  If a line number was
        LD H,(IX+0E)  specified then it too has to
        LD A,H  be considered.
        AND +C0  JR NZ,08AD,LD-PROG-1  Jump if ‘no number’; otherwise
        JR NZ,08AD,LD-PROG-1  set NEWPPC & NSPPC.
        LD L,(IX+0D)  LD (NEWPPC),HL
        LD (NSPPC),+00

The data block can now be LOADed.

08AD LD-PROG-1 POP DE  Fetch the ‘length’.
POP IX  Fetch the ‘start’.
SCF  Signal ‘LOAD’.
LD A,+FF  Signal ‘data block’ only.
JP 0802,LD-BLOCK  Now LOAD it.
THE 'MERGE' CONTROL ROUTINE

There are three main parts to this routine.

I. LOAD the data block into the work space.
II. MERGE the lines of the new program into the old program.
III. MERGE the new variables into the old variables.

Start therefore with the LOADING of the data block.

08B6 ME-CONTRL       LD     C,(IX+0B)  Fetch the 'length' of the
                      LD     B,(IX+0C)  data block.
PUSH BC               INC BC    Save a copy of the 'length'.
RST                  0030,BC-SPACES Now made 'length+1' locations
                      LD      (HL),+80 Available in the work space.
EX                   DE,HL    Place an end-marker in the
                      POP      DE     extra location.
                      POP      HL     Move the 'start' pointer to the
                      POP      IX     HL register pair.
SCF                  LD      +FF    Signal 'LOAD'.
call                 0802,LD-BLOCK LOAD the data block.

The lines of the new program are MERGED with the lines of the old program.

08D2 ME-NEW-HP       LD     A,(HL)    Fetch a line number and test
                      AND      +C0     it.
                      JR       NZ,08F0,ME-VAR-HP Jump when finished with all
                              the lines.

Now enter an inner loop to deal with the lines of the old program.

08D7 ME-OLD-HP       LD     A,(DE)    Fetch the high line number
                      INC      DE     byte and compare it.
                      CP       (HL)    Jump forward if it does not
                      INC      HL     match but in any case advance
                      JR       NZ,08DF,ME-OLD-L1 both pointers.
                      LD      A,(DE)    Repeat the comparison for the
                      CP       (HL)    low line number bytes.
                      DEC      DE     Now retreat the pointers.
                      DEC      HL
                      JR       NC,08EB,ME-NEW-L2 Jump forward if the correct
                              place has been found for a line
                              of the new program.
                      CALL     19B8,NEXT-ONE Otherwise find the address of
                      PUSH     HL      the start of the next old line.
                      EX       DE,HL   Go round the loop for each of
                      CALL     08D7,ME-OLD-HP the 'old lines'.
                      POP      HL
                      JR       08D2,ME-NEW-LP Enter the 'new line' and go
                      CALL     092C,ME-ENTER round the outer loop again.
                      JR       08D2,ME-NEW-LP

In a similar manner the variables of the new program are MERGED with the variables of the old program.
A loop is entered to deal with each of the new variables in turn.

08F0 ME-VAR-HP       LD     A,(HL)    Fetch each variable name in
LD C,A turn and test it.
CP +80 Return when all the variables
RET Z have been considered.
PUSH HL Save the current new pointer.
LD HL,(VARS) Fetch VARS (for the old program).

Now enter an inner loop to search the existing variables area.

08F9 ME-OLD-VP LD A,(HL) Fetch each variable name and
CP +80 test it.
JR Z,0923,ME-VAR-L2 Jump forward once the end
marker is found. (Make an ‘addition’.)
CP c Compare the names 0 st. bytes).
JR Z,0909,ME-OLD-v2 Jump forward to consider it
further; returning here if it proves not to match fully.

0901 ME-OLD-V1 PUSH BC Save the new variable’s name
CALL 19B8,NEXT-ONE whilst the next ‘old variable’
EX DE,HL is located.
JR 08F9,ME-OLD-VP Restore the pointer to the
D E register pair and go round the loop again.

The old and new variables match with respect to their first bytes but variables with long names will need to be matched fully.

0909 ME-OLD-V2 AND +E0 Consider bits 7, 6 & 5 only.
CP +A0 Accept all the variable types
JR NZ,0921,ME-VAR-L1 except ‘long named variables’.
POP DE Make DE point to the first
PUSH DE character of the ‘new name’.
PUSH HL Save the pointer to the ‘old name’.

Enter a loop to compare the letters of the long names.

0912 ME-OLD-V3 INC HL Update both the ‘old’ and the
INC DE ‘new’ pointers.
LD A,(DE) Compare the two letters
CP (HL)
JR NZ,091E,ME-OLD-V4 Jump forward if the match fails.
RLA Go round the loop until the
JR NC,0912,ME-OLD-V3 ‘last character’ is found.
POP HL Fetch the pointer to the
JR 0921,ME-VAR-L1 start of the ‘old name’ and
INC HL jump forward - successful.
POP HL Fetch the pointer and jump
JR 0901,ME-OLD-V1 back - unsuccessful.

Come here if the match was found.

0921 ME-VAR-L1 LD A,+FF Signal ‘replace’ variable.

And here if not. (A holds +80 - variable to be ‘added’.)

0923 ME-VAR-L2 POP DE Fetch pointer to ‘new’ name.
EX DE,HL Switch over the registers.
INC A The zero flag is to be set if there
SCF is to be a ‘replacement’; reset
CALL for an ‘addition’.
092C,ME-ENTER Signal ‘handling variables’.
JR 08F0,ME-VAR-LP Now make the entry.

Go round the loop to consider the next new variable.
THE 'MERGE A LINE OR A VARIABLE' SUBROUTINE

This subroutine is entered with the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carry flag</td>
<td>- MERGE a BASIC line.</td>
</tr>
<tr>
<td>set</td>
<td>- MERGE a variable.</td>
</tr>
<tr>
<td>Zero</td>
<td>reset - It will be an 'addition'.</td>
</tr>
<tr>
<td>set</td>
<td>set - It is a 'replacement'.</td>
</tr>
<tr>
<td>HL register pair</td>
<td>reset - Points to the start of the new entry.</td>
</tr>
<tr>
<td>DE register pair</td>
<td>set - Points to where it is to MERGE.</td>
</tr>
</tbody>
</table>

092C ME-ENTER JR NZ,093E,ME-ENT-1 Jump if handling an 'addition'.
EX AF,AF' Save the flags.
LD (X-PTR),HL Save the 'new' pointer whilst
EX DE,HL the 'old' line or variable
CALL 19B8,NEXT-ONE is reclaimed.
CALL 19E8,RECLAIM-2
EX DE,HL
LD HL,(X-PTR)
EX AF,AF' Restore the flags.

093E ME-ENT-1 EX AF,AF' Save the flags.
PUSH DE Make a copy of the
CALL 19B8,NEXT-ONE 'destination' pointer.
LD (X-PTR),HL Find the length of the 'new'
LD HL,(PROG) variable/line.
EX (SP),HL Save the pointer to the 'new'
CALL 19B8,NEXT-ONE variable/line.
LD HL,(PROG) Fetch PROG - to avoid
EX AF,AF' corruption.
PUSH BC Save PROG on the stack and
EX AF,AF' fetch the 'new' pointer.
JR C,0955,ME-ENT-2 Jump forward if adding a new
DEC HL variable. A new line is added before the
CALL 1655,MAKE-ROOM 'destination' location.
INC HL Make the room for the new line.
JR 0955,ME-ENT-3
CALL 1655,MAKE-ROOM Jump forward.

0958 ME-ENT-3 INC HL Point to the 1st new location.
PUSH BC Retrieve the length.
PUSH DE Retrieve PROG and store it
LD (PROG),DE in its correct place.
LD DE,(X-PTR) Also fetch the 'new' pointer.
PUSH BC Again save the length and the
PUSH DE new' pointer.
EX DE,HL Switch the pointers and copy
LDIR the 'new' variable/line into the
room made for it.

0955 ME-ENT-2 CALL 1655,MAKE-ROOM Make the room for the new variable.

The 'new' variable/line has now to be removed from the work space.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POP</td>
<td>HL Fetch the 'new' pointer.</td>
</tr>
<tr>
<td>POP</td>
<td>BC Fetch the length.</td>
</tr>
</tbody>
</table>
| PUSH               | DE Save the 'old' pointer. (Points to the location after the 'added'
|                   | variable/line.)                           |
| CALL               | 19E8,RECLAIM-2 Remove the variable/line from
|                   | the work space.                           |
| POP                | DE Return with the 'old' pointer in the DE register pair. |
THE ‘SAVE’ CONTROL ROUTINE

The operation of SAVing a program or a block of data is very straightforward.

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0970</td>
<td>SA-CONTR</td>
<td>PUSH HL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Save the ‘pointer’.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LD A,+FD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ensure that channel 'K'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CALL 1601,CHAN-OPEN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is open.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XOR A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Signal ‘first message’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LD DE,+09A1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Print the message - Start tape,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CALL 0C0A,PO-MSG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>then press any key.’.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SET 5,(TV-FLAG)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Signal ‘screen will require to be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CAL 15D4,WAIT-KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cleared’.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Save the ‘pointer’.</td>
</tr>
</tbody>
</table>

Upon receipt of a keystroke the ‘header’ is saved.

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PUSH IX</td>
<td>Save the base address of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘header’ on the machine stack.</td>
</tr>
<tr>
<td></td>
<td>LD DE,+0011</td>
<td>Seventeen bytes are to be</td>
</tr>
<tr>
<td></td>
<td>XOR A</td>
<td>SAVEd.</td>
</tr>
<tr>
<td></td>
<td>CALL 04C2,SA-BYTES</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Send the ‘header’, with a leading</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘type’ byte and a trailing ‘parity’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>byte.</td>
</tr>
</tbody>
</table>

There follows a short delay before the program/data block is SAVEd.

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>POP IX</td>
<td>Retrieve the pointer to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘header’.</td>
</tr>
<tr>
<td></td>
<td>LD B,+32</td>
<td>The delay is for fifty</td>
</tr>
<tr>
<td></td>
<td>DJNZ 0991,SA-1-SEC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>interrupts, i.e. one second.</td>
</tr>
<tr>
<td></td>
<td>LD E,(IX+0B)</td>
<td>Fetch the length of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>data block that is to be SAVEd.</td>
</tr>
<tr>
<td></td>
<td>LD A,+FF</td>
<td>Signal ‘data block’.</td>
</tr>
<tr>
<td></td>
<td>POP IX</td>
<td>Fetch the ‘start of block</td>
</tr>
<tr>
<td></td>
<td>JP 04C2,SA-BYTES</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pointer’ and SAVE the block.</td>
</tr>
</tbody>
</table>

THE CASSETTE MESSAGES

Each message is given with the last character inverted (+80 hex.).

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>09A1</td>
<td>DEFB +80 - Initial byte is stepped over.</td>
</tr>
<tr>
<td>09A2</td>
<td>DEFM - Start tape, then press any key.</td>
</tr>
<tr>
<td>09C1</td>
<td>DEFM - ‘carriage return’ - Program:</td>
</tr>
<tr>
<td>09CB</td>
<td>DEFM - ‘carriage return’ - Number array:</td>
</tr>
<tr>
<td>09DA</td>
<td>DEFM - ‘carriage return’ - Character array:</td>
</tr>
<tr>
<td>09EC</td>
<td>DEFM - ‘carriage return’ - Bytes:</td>
</tr>
</tbody>
</table>
THE SCREEN & PRINTER HANDLING ROUTINES

THE 'PRINT-OUT' ROUTINES
All of the printing to the main part of the screen, the lower part of the screen and the printer is handled by this set of routines. The PRINT-OUT routine is entered with the A register holding the code for a control character, a printable character or a token.

09F4 PRINT-OUT CALL 0B03,PO-FETCH The current print position.
CP +20 If the code represents a printable character then jump.
JP NC,0AD9,PO-ABLE
CP +06 Print a question mark for codes in the range +00 - +05.
JR C,0A69,PO-QUEST And also for codes +18 - +1F.
CP +18
JR NC,0A69,PO-QUEST
LD HL,+0A0B Base of 'control' table.
LD E,A Move the code to the
LD D,+00 DE register pair.
ADD HL,DE Index into the table and
LD E,(HL) fetch the offset.
ADD HL,DE Add the offset and make
PUSH HL an indirect jump to the
JP 0B03,PO-FETCH appropriate subroutine.

THE 'CONTROL CHARACTER' TABLE

<table>
<thead>
<tr>
<th>address</th>
<th>offset</th>
<th>character</th>
<th>address</th>
<th>offset</th>
<th>character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0A11</td>
<td>4E</td>
<td>PRINT comma</td>
<td>0A1A</td>
<td>4F</td>
<td>not used</td>
</tr>
<tr>
<td>0A12</td>
<td>57</td>
<td>EDIT</td>
<td>0A1B</td>
<td>5F</td>
<td>INK control</td>
</tr>
<tr>
<td>0A13</td>
<td>10</td>
<td>cursor left</td>
<td>0A1C</td>
<td>5E</td>
<td>PAPER control</td>
</tr>
<tr>
<td>0A14</td>
<td>29</td>
<td>cursor right</td>
<td>0A1D</td>
<td>5D</td>
<td>FLASH control</td>
</tr>
<tr>
<td>0A15</td>
<td>54</td>
<td>cursor down</td>
<td>0A1E</td>
<td>5C</td>
<td>BRIGHT control</td>
</tr>
<tr>
<td>0A16</td>
<td>53</td>
<td>cursor up</td>
<td>0A1F</td>
<td>5B</td>
<td>INVERSE control</td>
</tr>
<tr>
<td>0A17</td>
<td>52</td>
<td>DELETE</td>
<td>0A20</td>
<td>5A</td>
<td>OVER control</td>
</tr>
<tr>
<td>0A18</td>
<td>37</td>
<td>ENTER</td>
<td>0A21</td>
<td>54</td>
<td>AT control</td>
</tr>
<tr>
<td>0A19</td>
<td>50</td>
<td>not used</td>
<td>0A22</td>
<td>53</td>
<td>TAB control</td>
</tr>
</tbody>
</table>

THE 'CURSOR LEFT' SUBROUTINE
The subroutine is entered with the B register holding the current line number and the C register with the current column number.

0A23 PO-BACK-1 INC C Move leftwards by one column.
LD A,+22 Accept the change unless
ADD C up against the lefthand side.
JP NZ,0A3A,PO-BACK-3
BIT 1,(FLAGS) If dealing with the printer
JR NZ,0A38,PO-BACK-2 jump forward.
INC B Go up one line.
LD C,+02 Set column value.
LD A,+18 Test against top line.
JR NZ,0A3A,PO-BACK-3
Note: This ought to be +19.
ADD B Accept the change unless at
LD C,+21 the top of the screen.
DEC

0A38 PO-BACK-2 INC C Unacceptable so down a line.
LD C,+21 Set to lefthand column.

0A3A PO-BACK-3 JP 0DD9,CL-SET Make an indirect return via
CL-SET & PO-STORE.

THE 'CURSOR RIGHT' SUBROUTINE
This subroutine performs an operation identical to the BASIC statement – PRINT OVER 1;CHR$ 32; .-

0A3D PO-RIGHT LD A,(P-FLAG) Fetch P-FLAG and save it on
PUSH AF the machine stack.
LD (P-FLAG),+01  Set P-FLAG to OVER 1.
LD A,+20       A 'space'.
CALL 0B65,PO-CHAR Print the character.
POP AF        Fetch the old value of
LD (P-FLAG),A P-FLAG.       Finished.

Note: The programmer has
forgotten to exit via PO-STORE.

THE 'CARRIAGE RETURN' SUBROUTINE
If the printing being handled is going to the printer then a carriage return character leads to the printer buffer being emptied. If the printing is to the screen then a test for 'scroll?' is made before decreasing the line number.

0A4F  PO-ENTER BIT 1,(FLAGS) Jump forward if handling
       JP NZ,0ECD,COPY-BUFF the printer.
       LD C,+21       Set to lefthand column.
       CALL 0C55,PO-SCR Scroll if necessary.
       DEC B          Now down a line.
       JP 0DD9,CL-SET Make an indirect return via
                 CL-SET & PO-STORE.

THE 'PRINT COMMA' SUBROUTINE
The current column value is manipulated and the A register set to hold +00 (for TAB 0) or +10 (for TAB 16).

0A5F  PO-COMMA CALL 0B03,PO-FETCH Why again?
       LD A,C       Current column number.
       DEC A        Move rightwards by two
       DEC A        columns and then test.
       AND +10      The A register will be +00 or
                     +10.
       JR 0AC3,PO-FILL Exit via PO-FILL.

THE 'PRINT A QUESTION MARK' SUBROUTINE
A question mark is printed whenever an attempt is made to print an unprintable code.

0A69  PO-QUEST LD A,+3F       The character '?'.
       JR 0AD9,PO-ABLE Now print this character instead.

THE 'CONTROL CHARACTERS WITH OPERANDS' ROUTINE
The control characters from INK to OVER require a single operand whereas the control characters AT & TAB are required to be followed by two operands.
The present routine leads to the control character code being saved in TVDATA-lo, the first operand in TVDATA-hi or the A register if there is only a single operand required, and the second operand in the A register.

0A6D  PO-TV-2 LD DE,+0A87 Save the first operand in
        JR 0A80,PO-CHANGE address of the 'output' routine
                     to PO-CONT (+0A87).
Enter here when handling the characters AT & TAB.

0A75  PO-2-OPER LD DE,+0A6D The character code will be
        JR 0A70,PO-TV-1 saved in TVDATA-lo and the
                     address of the 'output' routine
                     changed to PO-TV-2 (+0A6D).
Enter here when handling the colour items - INK to OVER.

0A7A  PO-1-OPER LD DE,+0A87 The 'output' routine is to be
        JR 0A7D PO-TV-1 changed to PO-CONT (+0A87).
The current 'output' routine address is changed temporarily.
Once the operands have been collected the routine continues.

Now deal with the AT control character.

And the TAB control character.

PRINTABLE CHARACTER CODES.

The required character (or characters) is printed by calling PO-ANY followed by PO-STORE.
THE 'POSITION STORE' SUBROUTINE

The new position's 'line & column' values and the 'pixel' address are stored in the appropriate system variables.

0ADC PO-STORE
BIT 1,(FLAGS) Jump forward if handling
JR NZ,0AFC,PO-STORE the printer.
BIT 0,(TV-FLAG) Jump forward if handling the
JR NZ,0AF0,PO-ST-E lower part of the screen.
LD (S-POSN),BC Save the values that relate
LD (DF-CC),HL to the main part of the
RET screen. Then return.
0AF0 PO-ST-E
LD (S-POSNL),BC Save the values that relate
LD (ECHO-E),BC to the lower part of the
LD (DF-CCL),HL screen.
RET Then return.
0afc PO-ST-PR
LD (P-POSN),C Save the values that relate
LD (PR-CC),HL to the printer buffer.
RET Then return.

THE 'POSITION FETCH' SUBROUTINE

The current position's parameters are fetched from the appropriate system variables.

0B03 PO-FETCH
BIT 1,(FLAGS) Jump forward if handling
JR NZ,0B1D,PO-F the printer.
LD BC,(S-POSN) Fetch the values relating
LD HL,(DF-CC) to the main part of the
BIT Z was the intention.
RET
LD BC,(S-POSNL) Otherwise fetch the values
LD HL,(DF-CCL) relating to the lower part
RET of the screen.
0B1D PO-F-PR
LD C,(P-POSN) Fetch the values relating
LD HL,(PR-CC) to the printer buffer.
RET

THE 'PRINT ANY CHARACTER(S)' SUBROUTINE

Ordinary character codes, token codes and user-defined graphic codes, and graphic codes are dealt with separately.

0B24 PO-ANY
CP +80 Jump forward with ordinary
JR C,0B65,PO-CHAR character codes.
CP +90 Jump forward with token
JR NC,0B52,PO-T&UDG codes and UDG codes.
LD B,A Move the graphic code.
CALL 0B38,PO-GR-1 Construct the graphic form.
CALL 0B03,PO-FETCH HL has been disturbed so
RET 'fetch' again.
LD DE,+5C92 Make DE point to the start of the
LD DE,+5C92 graphic form; i.e. MEMBOT.
JR 0B7F,PO-ALL Jump forward to print the
graphic character.

Graphic characters are constructed in an Ad Hoc manner in the calculator's memory area; i.e. MEM-0 & MEM-1.

0B38 PO-GR-1
LD HL,+5C92 This is MEMBOT.
CALL 0B3E,PO-GR-2 In effect call the following
CALL subroutine twice.
0B3E PO-GR-2
RR B Determine bit 0 (and later bit 2)
SBC A,A of the graphic code.
AND +0F The A register will hold +00 or
+0F depending on the value of
the bit in the code.
LD C,A
Save the result in C.
RR B
Determine bit 1 (and later bit 3) of the graphic code.
SBC A,A
The A register will hold +00 or +F0.
OR C
The two results are combined.
LD C,+04
Determine bit 1 (and later bit 3) of the graphic code.
INC HL
used four times.
DEC C
This is done for the upper half of the character form and then the lower.
LD (HL),A
character form and has to be
JR NZ,0B4C,PO-GR-3

Token codes and user-defined graphic codes are now separated.

0B62 PO-T&UDG SUB +A5 Jump forward with token codes
JR NC,0B5F,PO-T
ADD A,+15 UDG codes are now +00 - +0F.
PUSH BC Save the current position values on the machine stack.
LD BC,(UDG) Fetch the base address of the UDG area and jump forward.
JR 0B6A,PO-CHAR-2
0B5F PO-T CALL 0C10,PO-TOCKENS Now print the token and return via PO-FETCH.
JP 0B03,PO-FETCH

The required character form is identified.

0B65 PO-CHAR PUSH BC The current position is saved.
LD BC,(CHARS) The base address of the character area is fetched.
0B6A PO-CHAR-2 EX DE,HL The print address is saved.
LD HL,+5C3B This is FLAGS.
RES 0,(HL) Allow for a leading space.
CP +20 Jump forward if the character
JR NZ,0B76,PO-CHAR-3 is not a 'space'.
SET 0,(HL) But 'suppress' if it is.
0B76 PO-CHAR-3 LD H,+00 Now pass the character code to the HL register pair.
LD L,A The character code is in effect multiplied by 8.
ADD HL,HL ADD HL,HL The base address of the character form is found.
ADD HL,BC
POP BC The current position is fetched and the base address passed to the DE register pair.
EX DE,HL

THE 'PRINT ALL CHARACTERS' SUBROUTINE
This subroutine is used to print all '8*8' bit characters. On entry the DE register pair holds the base address of the character form, the HL register the destination address and the BC register pair the current 'line & column' values.

0B7F PR-ALL LD A,C Fetch the column number.
DEC A Move one column rightwards.
LD A,+21 Jump forward unless a new line is indicated.
JR NZ,0B93,PR-ALL-1
DEC B Move down one line.
LD C,A Column number is +21.
BIT 1,(FLAGS) Jump forward if handling the screen.
JR Z,0B93,PR-ALL-1
PUSH DE Save the base address whilst the printer buffer is emptied.
CALL 0ECD,COPY-BUFF the printer buffer is
POP DE Copy the new column number.
LD A,C
0B93 PR-ALL-1 CP C Test whether a new line is
PUSH DE
CALL Z,0C55,PO-SCR
POP DE

being used. If it is
see if the display requires
to be scrolled.

Now consider the present state of INVERSE & OVER'

PUSH BC
PUSH HL
LD A,(P-FLAG)
LD B,+FF
RRA
JR C,0BA4,PR-ALL-2
INC B

Save the position values
and the destination address
on the machine stack.
Fetch P-FLAG and read bit 0.
Prepare the 'OVER-mask' in
the B register; i.e. OVER 0
= +00 & OVER 1 - +FF.

0BA4 PR-ALL-2
RRA
SBC A,A
LD C,A
AND A
BIT 1,(FLAGS)
JR Z,0BB6,PR-ALL-3
SET 1,(FLAGS2)
SCF

Read bit 2 of P-FLAG and
prepare the 'INVERSE-mask'
in the C register; i.e.
INVERSE 0 = +00 & INVERSE
1 = +FF.
Set the A register to hold
the 'pixel-line' counter and
clear the carry flag.
Jump forward if handling
the screen.
Signal 'printer buffer no longer
empty.
Set the carry flag to show that
the printer is being used.

0BB6 PR-ALL-3
EX DE,HL

Exchange the destination
address with the base address
before entering the loop.

0BB7 PR-ALL-4
EX AF,A'F'
LD A,(DE)
AND B
XOR (HL)
XOR C
LD (DE),A
EX AF,A'F'
JR C,0BB8,PR-ALL-6
INC D

The carry flag is set when using
the printer. Save this flag in F'.
Fetch the existing 'pixel-line'.
Use the 'OVER-mask' and then
XOR the result with the 'pixel-
line' of the character form.
Finally consider the 'INVERSE-
mask'.
Enter the result.
Fetch the printer flag and
jump forward if required.
Update the destination address
of the character form.
Update the 'pixel-line' of
the character form.
Decrease the counter and loop
back unless it is zero.

0BC1 PR-ALL-5
INC HL
DEC A
JR NZ,0BB7,PR-ALL-4

Once the character has been printed the attribute byte is to set as required.

EX DE,HL
DEC H
BIT 1,(FLAGS)
CALL Z,0BDB,PO-ATTR
POP HL
POP BC
DEC C
INC HL
RET

Make the H register hold a
correct high-address for the
character area.
Set the attribute byte only if
handling the screen.
Restore the original
destination address and the
position values.
Decrease the column number
and increase the destination
address before returning.

When the printer is being used the destination address has to be updated in increments of +20.
THE 'SET ATTRIBUTE BYTE' SUBROUTINE
The appropriate attribute byte is identified and fetched. The new value is formed by manipulating the old value, ATTR-T, MASK-T and P-FLAG. Finally this new value is copied to the attribute area.

0BDB  PO-ATTR
LD   A,(HL)    The high byte of the attribute area is then formed.
XOR  E         The values of MASK-T and D are taken into account.
AND  D         ATTR-R are taken into account.
LD   (DE),(ATTR-T)  Jump forward unless dealing with PAPER 9.
JR   C,0BFA,PO-ATTR-1
JR   NZ,0BFA,PO-ATTR-1
AND  +C7       The old paper colour is ignored and depending on whether the ink colour is light or dark the new paper colour will be black (000) or white (111).
BIT  6,(P-FLAG) Jump forward unless dealing with INK 9.
JR   Z,0BFA,PO-ATTR-1
JR   NZ,0BFA,PO-ATTR-1
AND  +F8       The old ink colour is ignored and depending on whether the paper colour is light or dark the new ink colour will be black (000) or white (111).
BIT  4,(P-FLAG) Jump forward unless dealing with INK 9.
JR   Z,0C08,PO-ATTR-2
JR   NZ,0C08,PO-ATTR-2
AND  +07       Enter the new attribute value and return.
0C08  PO-ATTR-2
LD   (HL),A    Enter the new attribute value and return.
RET

THE 'MESSAGE PRINTING' SUBROUTINE
This subroutine is used to print messages and tokens. The A register holds the 'entry number' of the message or token in a table. The DE register pair holds the base address of the table.

0C0A  PO-MSG
PUSH  HL       The high byte of the last entry on the machine stack is made zero so as to suppress trailing spaces (see below).
PUSH  (SP),HL  Jump forward.
JR   0C14,PO-TABLE
LD   H,+00    The base address of the token table.
EX    (SP),HL  (Range +00 - +5A; RND - COPY).
JR   0C14,PO-TABLE
PUSH  AF      Save the code on the stack.
JR   0C41,PO-SEARCH  Locate the required entry.
JR   C,0C22,PO-EACH  Print the message/token.
LD   A,+20    A 'space' will be printed.
The characters of the message/token are printed in turn.

Now consider whether a 'trailing space' is required.

THE 'PO-SAVE' SUBROUTINE
This subroutine allows for characters to be printed 'recursively'. The appropriate registers are saved whilst 'PRINT-OUT' is called.

THE 'TABLE SEARCH' SUBROUTINE
The subroutine returns with the DE register pair pointing to the initial character of the required entry and the carry flag reset if a 'leading space' is to be considered.

THE 'TEST FOR SCROLL' SUBROUTINE
This subroutine is called whenever there might be the need to scroll the display. This occurs on three occasions: i. when handling a 'carriage return' character; ii. when using AT in an INPUT line; & iii. when the current line is full and the next line has to be used. On entry the B register holds the line number under test.
Return immediately if the printer is being used.

Pre-load the machine stack with the address of 'CL-SET'.

Transfer the line number.

Jump forward if considering 'INPUT ... AT ...'.

Return, via CL-SET, if the line number is greater than the value of DF-SZ; give report 5 if it is less; otherwise continue.

Jump forward unless dealing with an 'automatic listing'.

Jump forward if the listing is to be scrolled.

Otherwise open channel 'K', restore the stack pointer, flag that the automatic listing has finished and return via CL-SET.

Call the error handling routine.

Decrease the scroll counter only if it becomes zero.

The counter is reset.

The current values of ATTR-T and MASK-T are saved.

The current value of P-FLAG is saved.

Channel 'K' is opened.

The message 'scroll?' is message 'O'. This message is now printed.

Signal 'clear the lower screen after a keystroke'.

This is FLAGS.

Signal 'L mode'.

Signal 'no key yet'.

Note: DE should be pushed also.

Fetch a single key code.

Restore the registers.

Restore the value of
LD (P-FLAG),A
POP HL
LD (ATTR-T),HL

P-FLAG.
Restore the values of ATTR-T
and MASK-T.

The display is now scrolled.

0CD2 PO-SCR-3 CALL 0DFE,CL-SC-ALL
LD B(DF-SZ)
INC B
LD C,+21
PUSH BC
CALL 0E9B,CL-ADDR
LD A,H
RRCA
RRCA
RRCA
AND +03
OR +58
LD H,A

The whole display is scrolled.
The line and column numbers
for the start of the line
above the lower part of the
display are found and saved.
The corresponding attribute
byte for this character area is
then found. The HL register pair
holds the address of the
byte.

LD B(DF-SZ)
The line and column numbers
INC
LD C,+21
There are thirty two bytes.
PUSH BC
CALL 0E9B,CL-ADDR
LD A,H
RRCA
RRCA
RRCA
AND +03
OR +58
LD H,A

LD (DE),A
Make the first exchange
LD (HL),C
and then proceed to use the
same values for the thirty
two attribute bytes of the
two lines being handled.
INC DE
INC HL
DJNZ 0CF0,PO-SCR-3A
POP BC

The line in question will have 'lower part' attribute values and the new line at the bottom of the display may have 'ATTR-P' values so the attribute values are exchanged.

LD DE,+5AE0
LD A,(DE)
LD C,(HL)
LD B,+20
EX DE,HL
LD (DE),A
INC DE
INC HL
DJNZ 0CF0,PO-SCR-3A
POP BC

DE points to the first attribute
byte of the bottom line.
The value is fetched.
The 'lower part' value.
There are thirty two bytes.
Exchange the pointers.
Make the first exchange
and then proceed to use the
two attribute bytes of the
two lines being handled.
The line and column numbers
of the bottom line of the 'upper
part' are fetched before
returning.

RET

0CF8 DEFB +80
DEFB +73,+63,+72,+6F
DEFB +6C,+6C,+BF

Initial marker - stepped over.
s-c-r-o
l - l - ? (inverted).

Report 0 - BREAK - CONT repeats

0D00 REPORT-D RST 0008,ERROR-1 DEF B +0C
Call the error handling
routine.

The lower part of the display is handled as follows:

0D02 PO-SCR-4 CP +02
JR C,0C86,REPORT-5
ADD A,(DF-SZ)
SUB +19
RET NC
NEG
PUSH BC
LD B,A
LD HL,(ATTR-T)
PUSH HL
LD HL,(P-FLAG)
PUSH HL
CALL 0D40,TEMPS

The 'out of screen' error is
given if the lower part is
going to be 'too large' and a
return made if scrolling is
unnecessary.
The A register will now hold
'the number of scrolls to be
made'.
The line and column numbers
are now saved.
The 'scroll number'. ATTR-T
MASK-T & P-FLAG are all
saved.
The 'permanent' colour items
are to be used.
LD A,B
The 'scroll number' is fetched.

The lower part of the screen is now scrolled 'A' number of times.

```
0D1C  PO-SCR-4A  PUSH  AF      Save the 'number'.
       HL,+5C6B     This is DF-SZ.
       B,(HL)      The value in DF-SZ is
       A,B         incremented; the B register
INC   A          set to hold the former value and
LD    (HL),A     the A register the new value.
CP    HL,+5C89   The jump is taken if only the
JR    C,0D2D,PO-SCR-4B  lower part of the display is
call  INC    to be scrolled. (B = old DF-SZ).
LD    B,+18     Otherwise S-POSN-hi is
0D2D  PO-SCR-4B  CALL  0E00,CL-SCROLL  incremented and the whole
       POP  HL      display scrolled. (B = +18)
       POP  AF     Scroll 'B' lines.
       POP  HL     Fetch and decrement the
       JR  A,NZ,0D1C,PO-SCR-4A  scroll number'.
       POP HL      Jump back until finished.
       LD (P-FLAG),L P-FLAG.
       POP HL      Restore the values of
       LD (ATTR-T),HL  and MASK-T.
       LD BC,(S-POSN)  In case S-POSN has been
       RES 0,(TV-FLAG)  changed CL-SET is called to
call  SET 0,(TV-FLAG)  give a matching value to DF-CC.
       POP BC      Reset the flag to indicate that
       RET         the lower screen is being
```

THE 'TEMPORARY COLOUR ITEMS' SUBROUTINE
This is a most important subroutine. It is used whenever the 'permanent' details are required to be copied to the 'temporary' system variables. First ATTR-T & MASK-T are considered

```
0D4D  TEMPS  XOR  A      A is set to hold +00.
       HL,(ATTR-P)  The current values of ATTR-P
       JR Z,0D5B,TEMPS-1  and MASK-P are fetched.
       LD H,A       Jump forward if handing the
       LD L,(BORDCR)  main part of the screen.
6      Otherwise use +00 and the
       LD   (ATTR-T),HL  value in BORDCR instead.
0D65  TEMPS-2  XOR  (HL)  Proceed to copy the even bits
       AND +55     of A to P-FLAG.
       LD (HL),A     Now set ATTR-T & MASK-T.
```

Next P-FLAG is considered.

```
LD HL,+5C91  This is P-FLAG.
JR NZ,0D65,TEMPS-2  Jump forward if dealing with
       LD A,(HL)  the lower part of the screen
       RRCA     (A = +00).
       LD P-FLAG  Otherwise fetch the value of
       A,(HL)     P-FLAG and move the odd bits
to the even bits.
0D65  TEMPS-2  XOR  (HL)  Proceed to copy the even bits
       AND +55  of A to P-FLAG.
       LD (HL),A
       RET
```

THE 'CLS COMMAND' ROUTINE
In the first instance the whole of the display is 'cleared' - the 'pixels' are all reset and the attribute bytes are set to equal the value in ATTR-P - then the lower part of the display is reformed.
The whole of the display is 'cleared'.

This is TV-FLAG.

Signal 'do not clear the lower screen after keystroke'.

Signal 'lower part'.

Use the permanent values, i.e. ATTR-T is copied from BORDCR.

The lower part of the screen is now 'cleared' with these values.

With the exception of the attribute bytes for lines '22' & '23' the attribute bytes for the lines in the lower part of the display will need to be made equal to ATTR-P.

It now remains for the following 'house keeping' tasks to be performed.

The size of the lower part of the display can now be fixed.

It will be two lines in size.

The 'Clearing the Whole Display Area' Subroutine

This subroutine is called from: i. the CLS command routine. ii. the main execution routine, and iii. the automatic listing routine.

The system variable C00RDS is reset to zero.

Signal 'the screen is clear'.

Perform the 'house keeping' tasks.

Open channel 'S'.

Use the 'permanent' values.

Now 'clear' the 24 lines of the display.

Ensure that the current output address is +09F4.
LD (HL),E (PRINT-OUT).
INC HL
LD (HL),D
LD (SCR-CT)+01
LD BC,+1821

Reset the scroll counter.
As the upper part of the display
is being handled the 'upper print
line' will be Line '0'.
Continue into CL-SET.

THE 'CL-SET' SUBROUTINE
This subroutine is entered with the BC register pair holding the line and column numbers of a character areas, or the C register holding the column number within the printer buffer. The appropriate address of the first character bit is then found. The subroutine returns via PO-STORE so as to store all the values in the required system variables.

0DD9 CL-SET LD HL,+5B00
BIT 1,(FLAGS) JR NZ,0DF4,CL-SET-2
LD A,B JR Z,0DEE,CL-SET-1
BIT 0,(TV-FLAG) ADD A,(DF-SZ)
SUB +18

0DEE CL-SET-1 PUSH BC
LD B,A CALL 0E9B,CL-ADDR
POP BC

0DF4 CL-SET-2 PUSH BC
LD B,+17 CALL 0E9B,CL-ADDR
LD A,B POP BC

ADD HL,DE
JP 0ADC,PO-STORE

THE 'SCROLLING' SUBROUTINE
The number of lines of the display that are to be scrolled has to be held on entry to the main subroutine in the B register.

0DFE CL-SCR-ALL LD B,+17

The entry point after 'scroll?'

The main entry point - from above and when scrolling for INPUT..AT.

0E00 CL-SCROLL CALL 0E9B,CL-ADDR
LD C,+08

Find the starting address of
the line.
There are eight pixel lines to
a complete line.

Now enter the main scrolling loop. The B register holds the number of the top line to be scrolled, the HL register pair the starting address in the display area of this line and the C register the pixel line counter.

0E05 CL-SCR-1 PUSH BC
LD A,B
AND +07
LD A,B
JR NZ,0E19,CL-SCR-3

Save both counters.
Jump forward unless
dealing at the present
moment with a 'third' of
the display.

The pixel lines of the top lines of the 'thirds' of the display have to be moved across the 2K boundaries. (Each 'third' = 2K.)
The result of this manipulation is to leave HL unchanged and DE pointing to the required destination. There are +20 characters. Decrease the counter as one line is being dealt with.

Now move the thirty two bytes.

The pixel lines within the 'thirds' can now be scrolled. The A register holds, on the first pass, +01 - +07, +09 - +0F or +11 - +17.

Again DE is made to point to the required destination. This time only thirty two locations away. Save the line number in B. Now find how many characters there are remaining in the 'third'.

Pass the 'character total' to the C register. Fetch the line number. BC holds the 'character total' and a pixel line from each of the characters is 'scrolled'.

Now prepare to increment the address to jump across a 'third' boundary. Increase HL by +0700. Jump back if there are any 'thirds' left to consider.

Now find if the loop has been used eight times - once for each pixel line.

Fetch the original address. Address the next pixel line. Fetch the counters. Decrease the pixel line counter and jump back unless eight lines have been moved.

Next the attribute bytes are scrolled. Note that the B register still holds the number of lines to be scrolled and the C register holds zero.

The required address in the attribute area and the number of characters in 'B' lines are found.

The displacement for all the attribute bytes is thirty two locations away. The attribute bytes are 'scrolled'.

It remains now to clear the bottom line of the display.

The B register is loaded with +01 and CL-LINE is entered.

THE 'CLEAR LINES' SUBROUTINE
This subroutine will clear the bottom 'B' lines of the display.

The line number is saved for the duration of the subroutine. The starting address for the line is formed in HL.
Again there are eight pixel lines to be considered.

Now enter a loop to clear all the pixel lines.

<table>
<thead>
<tr>
<th>Opcode</th>
<th>CLINE-1</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0E4A</td>
<td>PUSH</td>
<td>BC</td>
<td>Save the line number and the pixel line counter.</td>
</tr>
<tr>
<td></td>
<td>PUSH</td>
<td>HL</td>
<td>Save the address.</td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>A,B</td>
<td>Save the line number in A.</td>
</tr>
<tr>
<td>0E4D</td>
<td>AND</td>
<td>HL</td>
<td>Find how many characters are involved in 'B mod 8' lines.</td>
</tr>
<tr>
<td></td>
<td>RRCA</td>
<td>+07</td>
<td>Pass the result to the C register. (C will hold +00 i.e. 256 dec. for a 'third'.)</td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>C,A</td>
<td>Fetch the line number.</td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>A,B,+00</td>
<td>Make the BC register pair hold 'one less' than the number of characters.</td>
</tr>
<tr>
<td></td>
<td>DEC</td>
<td>C</td>
<td>Make DE point to the first character.</td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>D,H</td>
<td>Clear the pixel-byte of the first character.</td>
</tr>
<tr>
<td></td>
<td>INC</td>
<td>DE</td>
<td>Make DE point to the second character and then clear the pixel-bytes of all the other characters.</td>
</tr>
<tr>
<td></td>
<td>LDIR</td>
<td></td>
<td>For each 'third' of the display HL has to be increased by +0701.</td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>DE,+0701</td>
<td>Now decrease the line number.</td>
</tr>
<tr>
<td></td>
<td>ADD</td>
<td>HL,DE</td>
<td>Discard any extra lines and pass the 'third' count to B.</td>
</tr>
<tr>
<td></td>
<td>DEC</td>
<td>A</td>
<td>Jump back if there are still 'thirds' to be dealt with.</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>+F8</td>
<td>Update the address for each pixel line.</td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>HL</td>
<td>Fetch the counters.</td>
</tr>
<tr>
<td></td>
<td>INC</td>
<td>H</td>
<td>Decrease the pixel line counter and jump back unless finished.</td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>BC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEC</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JR</td>
<td>NZ,0E4D,CLINE-2</td>
<td></td>
</tr>
</tbody>
</table>

Now find if the loop has been used eight times.

<table>
<thead>
<tr>
<th>Opcode</th>
<th>CLINE-3</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0E80</td>
<td>CALL</td>
<td>0E88,CL-ATTR</td>
<td>The address of the first attribute byte and the number of bytes are found.</td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>H,D</td>
<td>HL will point to the first attribute byte and DE the second.</td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>L,E</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INC</td>
<td>DE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>A,(ATTR-P)</td>
<td>Fetch the value in ATTR-P.</td>
</tr>
<tr>
<td></td>
<td>BIT</td>
<td>0,(TV-FLAG)</td>
<td>Jump forward if handling the main part of the screen.</td>
</tr>
<tr>
<td></td>
<td>JR</td>
<td>Z,0E80,CLINE-3</td>
<td>Otherwise use BORDCR instead.</td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>A,(BORDCR)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEC</td>
<td>BC</td>
<td>One byte has been done.</td>
</tr>
<tr>
<td></td>
<td>LDIR</td>
<td></td>
<td>Now copy the value to all the attribute bytes.</td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>BC</td>
<td>Restore the line number.</td>
</tr>
<tr>
<td></td>
<td>LD</td>
<td>C,+21</td>
<td>Set the column number to the lefthand column and return.</td>
</tr>
</tbody>
</table>

RET
THE 'CL-ATTR' SUBROUTINE
This subroutine has two separate functions.

i. For a given display area address the appropriate attribute address is returned in the DE register pair. Note that the value on entry points to the 'ninth' line of a character.

ii. For a given line number, in the B register, the number of character areas in the display from the start of that line onwards is returned in the BC register pair.

```
0E88  CL-ATTR
   LD   A,H
   RRCA
   RRCA
   DEC  A
   OR   +50
   LD   H,A
   EX   DE,HL
   LD   H,C
   LD   L,B
   ADD  HL,HL
   ADD  HL,HL
   ADD  HL,HL
   ADD  HL,HL
   LD   B,H
   LD   C,L
   RET
```

THE 'CL-ADDR' SUBROUTINE
For a given line number, in the B register, the appropriate display file address is formed in the HL register pair.

```
0E9B  CL-ADDR
   LD   A,+18
   SUB  B
   LD   D,A
   RRCA
   RRCA
   AND  +E0
   LD   L,A
   LD   A,D
   AND  +18
   OR   +40
   LD   H,A
   RET
```

THE 'COPY' COMMAND ROUTINE
The one hundred and seventy six pixel lines of the display are dealt with one by one.

```
0EAC  COPY
   DI
   LD   B,+B0
   LD   HL,+4000
   INC  H
   DI
```

The following loop is now entered.

```
0EB2  COPY-1
   PUSH  HL
   PUSH  BC
   CALL  0EF4,COPY-LINE
   POP   BC
   POP   HL
   INC   H
   RET
```
LD A,H
AND +07
JR NZ,0EC9,COPY-2

Jump forward and hence round the loop again directly for the eight pixel lines of a character line.

For each new line of characters the base address has to be updated.

LD A,L
ADD A,+20
LD L,A
CCF
SBC A,A
AND +F8
ADD A,H
LD H,A

Copy the low byte.
Update it by +20 bytes.
The carry flag will be reset when 'within thirds' of the display.
The A register will hold +F8 when within a 'third' but +00 when a new third is reached.
The high byte of the address is now updated.

Copy -2
DJNZ 0EB2,COPY-1
Jump back until '176' lines have been printed.

JR 0EDA,COPY-END
Jump forward to the end routine.

THE 'COPY-BUFF' SUBROUTINE
This subroutine is called whenever the printer buffer is to have its contents passed to the printer.

0ECD COPY-BUFF DI
LD HL,+5800
LD B,+08
PUSH BC
CALL 0EF4,COPY-LINE
POP BC
DJNZ 0ED3,COPY-3
Jump back until '8' lines have been printed.

0EDA COPY-END LD A,+04
OUT (+FB),A
EI
Stop the printer motor.
Enable the maskable interrupt and continue into CLEAR-PRB.

THE 'CLEAR PRINTER BUFFER' SUBROUTINE
The printer buffer is cleared by calling this subroutine.

0EDF CLEAR-PRB LD HL,+5B00
LD (PR-CC-lo),L
XOR A
LD B,A
DJNZ 0EE7,PRB-BYTES
INC HL
LD 00E7,PRB-BYTES
RES 1,(FLAGS2)
LD C,+21
JP 0DD9,CL-SET

The base address of the printer buffer.
Reset the printer 'column'.
Clear the A register.
Also clear the B register in effect B holds dec.256).
The '256' bytes of the printer buffer are all cleared in turn.
Signal 'the buffer is empty'.
Set the printer position and return via CL-SET & P0-STORE.

THE 'COPY-LINE' SUBROUTINE
The subroutine is entered with the HL register pair holding the base address of the thirty two bytes that form the pixel-line and the B register holding the pixel-line number.

0EF4 COPY-LINE LD A,B
CP +03
SBC A,A
AND +02

Copy the pixel-line number.
The A register will hold +00 until the last two lines are being handled.
OUT (+FB),A
LD D,A

Slow the motor for the last two pixel lines only.
The D register will hold either +00 or +02.

There are three tests to be made before doing any 'printing'.

0EFD  COPY-L-1  CALL  1F54,BREAK-KEY
JR  C,0F0C,COPY-L-2  Jump forward unless the
LD  A,+04  BREAK key is being pressed.
OUT (+FB),A  But if it is then;
EI  stop the motor,
CALL 0EDF,CLEAR-PRB  enable the maskable interrupt,
RST 0008,ERROR-1  clear the printer buffer and exit
DEFB +0C  via the error handling routine
'BREAK-CONT repeats'.

0F0C  COPY-L-2
IN A,(+FB)  'Print' each bit.
ADD A,A  Fetch the status of the
RET M  printer.
JR NC,0EFD,COPY-L-1  Make an immediate return if the
LD C,+20  printer is not present.

There are thirty two bytes.

Now enter a loop to handle these bytes.

0F14  COPY-L-3
LD E,(HL)  Fetch a byte.
INC HL  Update the pointer.
LD B,+08  Eight bits per byte.

0F18  COPY-L-4
RL D  Move D left.
RL E  Move each bit into the carry.
RR D  Move D back again, picking up the carry from E.

0F1E  COPY-L-5
IN A,(+FB)  Again fetch the status of the
RRA  printer and wait for the
JR NC,0F1E,COPY-L-5  signal from the encoder.
LD A,D  Now go ahead and pass the
OUT (+FB),A  'bit' to the printer.

D,INZ 0F18,COPY-L-4  Note: bit 2 - low starts the
DEC C  motor, bit 1 - high slows the
JR NZ,0F14,COPY-L-3  motor and bit 7 is high for the
RET actual 'printing'.

A loop is now entered to handle each keystroke.

0F38  ED-LOOP  CALL  15D4,WAIT-KEY
PUSH AF  Return once a key has been

THE 'EDITOR' ROUTINES
The editor is called on two occasions:
i. From the main execution routine so that the user can enter a BASIC line into the system.

ii. From the INPUT command routine.

First the 'error stack pointer' is saved and an alternative address provided.

0F2C  EDITOR  LD HL,(ERR-SP)  The current value is saved on
PUSH HL  the machine stack.
0F30  ED-AGAIN  LD HL,+107F  This is ED-ERROR.
PUSH HL  Any event that leads to the
LD (ERR-SP),SP  error handling routine being used will come back to

ED-ERROR.

A loop is now entered to handle each keystroke.

0F38  ED-LOOP  CALL  15D4,WAIT-KEY
PUSH AF  Save the code temporarily.
LD D,+00 Fetch the duration of the keyboard click.
LD E,(PIP) And the pitch.
CALL 03B5,BEEPER Now make the 'pip'.
POP AF Restore the code.
LD HL,+0F38 Pre-load the machine stack with the address of ED-LOOP.

Now analyse the code obtained.

CP +18 Accept all character codes, graphic codes and tokens.
JR NC,0F81,ADD-CHAR Also accept ','.
CP +07 Jump forward if the code
JR C,0F81,ADD-CHAR represents an editing key.

The control keys - INK to TAB - are now considered.

LD BC,+0002 INK & PAPER will require two locations.
LD D,A Copy the code to 0.
CP +16 Jump forward with INK &
JR C,0F6C,ED-CONTR PAPER'

AT & TAB would be handled as follows:

INC BC Three locations required.
BIT 7,(FLAGX) Jump forward unless dealing
CALL 15D4,WAIT-KEY Get the second code.
LD E,A and put it in E.

The other bytes for the control characters are now fetched.

0F6C ED-CONTR CALL 15D4,WAIT-KEY Get another code.
PUSH DE Save the previous codes.
LD HL,(K-CUR) Fetch K-CUR.
RES 0,(MODE) Signal 'K mode'.
CALL 1655,MAKE-ROOM Make two or three spaces.
ADD HL,DE The entry is addressed and
LD (HL),B Enter first code.
INC HL Then enter the second code
LD (HL),C which will be Overwritten if
JR 0F8B,ADD-CH-1 if there are only two codes - i.e. with INK & PAPER.

THE 'ADDCHAR' SUBROUTINE
This subroutine actually adds a code to the current EDIT or INPUT line.

0F81 ADD-CHAR RES 0,(MODE) Signal 'K mode'.
LD HL,(K-CUR) Fetch the cursor position.
CALL 1652,ONE-SPACE Make a single space.

0F8B ADD-CH-1 LD (DE),A Enter the code into the space
INC DE and signal that the cursor is to
LD (K-CUR),DE occur at the location after. Then
RET return indirectly to ED-LOOP.

The editing keys are dealt with as follows:

0F92 ED-KEYS LD E,A The code is transferred to the DE register pair.
LD C,+00 The base address of the editing key table.
LD HL,+0F99 The entry is addressed and then fetched into E.
ADD HL,DE The address of the handling
PUSH HL
LD HL,(K-CUR)
RET

routine is saved on the machine stack.
The HL register pair is set and an indirect jump made to the required routine.

THE 'EDITING KEYS' TABLE

<table>
<thead>
<tr>
<th>address</th>
<th>offset</th>
<th>character</th>
<th>address</th>
<th>offset</th>
<th>character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0FA0</td>
<td>09</td>
<td>EDIT</td>
<td>0FA5</td>
<td>70</td>
<td>DELETE</td>
</tr>
<tr>
<td>0FA1</td>
<td>66</td>
<td>cursor left</td>
<td>0FA6</td>
<td>7E</td>
<td>ENTER</td>
</tr>
<tr>
<td>0FA2</td>
<td>6A</td>
<td>cursor right</td>
<td>0FA7</td>
<td>CF</td>
<td>SYMBOL SHIFT</td>
</tr>
<tr>
<td>0FA3</td>
<td>50</td>
<td>cursor down</td>
<td>0FA8</td>
<td>D4</td>
<td>GRAPHICS</td>
</tr>
<tr>
<td>0FA4</td>
<td>85</td>
<td>cursor up</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE 'EDIT KEY' SUBROUTINE

When in 'editing mode' pressing the EDIT key will bring down the 'current BASIC line'. However in 'INPUT mode' the action of the EDIT key is to clear the current reply and allow a fresh one.

0FA9 ED-EDIT
LD HL,(E-PPC)
BIT 5,(FLAGX)
JP NZ,1097,CLEAR-SP
CALL 1695,LINE-NO
LD A,D
OR E
JP Z,1097,CLEAR-SP
PUSH HL
INC HL
LD C,(HL)
INC HL
LD B,(HL)
ADD HL,BC
LD B,H
LD C,L
CALL 1F05,TEST-ROOM
CALL 1097,CLEAR-SP
LD HL,(CURCHL)
EX (SP),HL
PUSH HL
LD A,FF
CALL 1601,CHAN-OPEN
POP HL
DEC HL
DEC (E-PPC-lo)
CALL 1855,OUT-LINE
INC (E-PPC-lo)
LD HL,(E-LINE)
INC HL
INC HL
INC HL
INC HL

Note: The decrementing of the line number does not always stop the cursor from being printed.
LD HL,(E-LINE)
INC HL
INC HL
INC HL
INC HL

for K-CUR.
LD (K-CUR),HL
CALL 1615,CHAN-FLAG
RET

THE 'CURSOR DOWN EDITING' SUBROUTINE

The former channel address and set the appropriate flags before returning to ED-LOOP.

LD HL, +5C49
CALL 190F, LN-FETCH
JR 106E, ED-LIST

0FF3 ED-DOWN
BIT 5,(FLAGX)
JR NZ, 1001, ED-STOP
LD HL, +5C49
CALL 190F, LN-FETCH
JR 106E, ED-LIST

1001 ED-STOP
LD (ERR - NR), +10
JR 1024, ED-ENTER

THE 'CURSOR LEFT EDITING' SUBROUTINE

The cursor is moved.

LD A, (HL)
CP +0D
RET Z
INC HL

1007 ED-LEFT
CALL 1031, ED-EDGE
JR 1011, ED-CUR

THE 'CURSOR RIGHT EDITING' SUBROUTINE

The current character is tested and if it is 'carriage return' then return. Otherwise make the cursor come after the character.

1011 ED-CUR
LD (K-CUR), HL
RET

THE 'DELETE EDITING' SUBROUTINE

Move the cursor leftwards.

LD BC, +0001
JP 19E8, RECLAIM-

1015 ED-DELETE
CALL 1031, ED-EDGE
LD BC, +0001
JP 19E8, RECLAIM-

THE 'ED-IGNORE' SUBROUTINE

The next two codes from the key-input routine are ignored.

101E ED-IGNORE
CALL 15D4, WAIT-KEY
CALL 15D4, WAIT-KEY

THE 'ENTER EDITING' SUBROUTINE

The address of ED-LOOP and ED-ERROR are discarded.

1024 ED-ENTER
POP HL
POP HL
LD (ERR-SP), HL
BIT 7, (ERR-SP)
RET NZ
LD SP, HL

1026 ED-END
POP HL
POP HL
BIT 7, (ERR-SP)
RET NZ
LD SP, HL

THE 'ED-EDGE' SUBROUTINE

The address of the cursor is in the HL register pair and will be decremented unless the cursor is already at the start of the line. Care is taken not to put the cursor between control characters and their parameters.

1031 ED-EDGE
SCF
CALL 1195, SET-DE
SBC HL, DE
ADD HL, DE
INC HL

DE will hold either E-LINE (for editing) or WORKSP (for INPUTing).
The carry flag will become set if the cursor is already to be at the start of the line.
Correct for the subtraction.
Now enter a loop to check that control characters are not split from their parameters.

103E  ED-EDGE-1
  LD  H,D
  LD  L,E
  INC  HL
  LD  A,(DE)
  AND  +F0
  CP  +10
  JR  NZ,1051,ED-EDGE-2
  INC  HL
  LD  A,(DE)
  SUB  +17
  ADC  A,+00
  JR  NZ,1051,ED-EDGE-2
  INC  HL
  LD  A,(DE)
  AND  A
  SBC  HL,BC
  ADD  HL,BC
  EX  DE,HL
  JR  C,103E,ED-EDGE-1

1051  ED-EDGE-2
  AND  A
  SBC  HL,BC
  ADD  HL,BC
  EX  DE,HL
  JR  NZ,1051,ED-EDGE-2
  INC  HL

THE 'CURSOR UP EDITING' SUBROUTINE
1059  ED-UP
  BIT  5,(FLAGX)
  RET  NZ
  LD  HL,(E-PPC)
  CALL  196E,LINE-ADDR
  EX  DE,HL
  CALL  1695,LINE-NO
  LD  HL,+5C4A
  CALL  1910,LN-STORE
  CALL  1795,AUTO-STORE
  LD  A,+00
  JP  1601,CHAN-OPEN

106E  ED-LIST
  CALL  1795,AUTO-LIST
  LD  A,+00
  JP  1601,CHAN-OPEN

THE 'ED-SYMBOL' SUBROUTINE
If SYMBOL & GRAPHICS codes were used they would be handled as follows:

1076  ED-SYMBOL
  BIT  7,(FLAGX)
  JR  Z,1024,ED-ENTER

107C  ED-GRAPH
  JP  0F81,ADD-CHAR

THE 'ED-ERROR' SUBROUTINE
Come here when there has been some kind of error.

107F  ED-ERROR
  BIT  4,(FLAGS2)
  JR  Z,1026,ED-END
LD (ERR-NR),+FF Cancel the error number and
LD D,+00 give a 'rasp' before going
LD E,(RASP) around the editor again.
LD HL,+1A90
CALL 0385.BEEPER
JP 0F30,ED-AGAIN

THE 'CLEAR-SP' SUBROUTINE
The editing area or the work space is cleared as directed.

1097 CLEAR-SP PUSH HL Save the pointer to the space.
CALL 1190,SET-HL DE will point to the first
call 19E5,RECLAIM-1 character and HL the last.
DEC HL The correct amount is now
CALL 19E5,RECLAIM-1 reclaimed.
LD (K-CUR),HL The system variables K-CUR
LD (MODE),+00 and MODE ('K mode') are
POP HL initialised before fetching
RET the pointer and returning.

THE 'KEYBOARD INPUT' SUBROUTINE
This important subroutine returns the code of the last key to have been pressed but note that CAPS LOCK, the changing of the mode and the control parameters are handled within the subroutine.

10A8 KEY-INPUT BIT 3,(TV-FLAG) Copy the edit-linear the
CALL NZ,111D,ED-COPY INPUT-line to the screen if
AND A Return with both carry
BIT 5,(FLAGS) and zero flags reset if no
RET Z new key has been pressed
LD A,(LAST-K) Otherwise fetch the code and
RES 5,(FLAGS) signal that it has been taken
PUSH AF Save the code temporarily.
BIT 5,(TV-FLAG) Clear the lower part of the
CALL NZ,0D6E,CLS-LOWER display if necessary;
POP AF e.g. after 'scroll?';
CP +20 Accept all characters and
JR NC,111B,KEY-DONE token codes.
CP +10 Jump forward with most of
JR NC,10FA,KEY-CONTR the control character codes.
CP +06 Jump forward with the 'mode'.
JR NC,10DB,KEY=M&CL codes and the CAPS LOCK code.

Now deal with the FLASH, BRIGHT & INVERSE codes.

LD B,A Save the code.
AND +01 Keep only bit 0.
LD C,A C holds +00 (= OFF) or
RRA C holds +01 (= ON).
LD A,B Fetch the code.
ADD A,+12 Rotate it once (losing bit 0).
JR 1105,KEY-DATA Increase it by +12 giving for
FLASH - +12, BRIGHT - +13
and INVERSE - +14.

The CAPS LOCK code and the mode codes are dealt with 'locally'.

10DB KEY-M&CL JR NZ,10E6,KEY-MODE Jump forward with 'mode' codes.
LD HL,+5CB6A This is FLAGS2
LD A,+08 Flip bit 3 of FLAGS2. This is
XOR the CAPS LOCK flag.
LD (HL),A
JR 10F4,KEY-FLAG Jump forward.

10E6 KEY-MODE CP +0E Check the lower limit.
The control key codes (apart from FLASH, BRIGHT & INVERSE) are manipulated.

10FA KEY-CONTR
LD B,A
AND +07
LD C,A
LD A,+10
BIT 3,B
JR NZ,1105
INC A

The parameter is saved in K-DATA and the channel address changed from KEY-INPUT to KEY-NEXT.

1105 KEY-DATA
LD (K-DATA),C
LD DE,+110D
JR 1113

Note: On the first pass entering at KEY-INPUT the A register is returned holding a control code’ and then on the next pass, entering at KEY-NEXT, it is the parameter that is returned.

110D KEY-NEXT
LD A,(K-DATA)
LD DE,+10A8

Now set the input address in the first channel area.

1113 KEY-CHAN
LD HL,(CHANS)
INC HL
INC HL
LD (HL),E
INC HL
LD (HL),D

Finally exit with the required code in the A register.

111B KEY-DONE
SCF
RET

THE 'LOWER SCREEN COPYING' SUBROUTINE
This subroutine is called whenever the line in the editing area or the INPUT area is to be printed in the lower part of the screen.

111D ED-COPY
CALL 0D4D,TEMPS
RES 3,(TV-FLAG)
RES 5,(TV-FLAG)
LD HL,(S-POSNL)
PUSH HL
LD HL,(ERR-SP)
PUSH HL
LD HL,+1167
PUSH HL
LD (ERR-SP),SP

Use the permanent colours.
Signal that the ‘mode is to be considered unchanged’ and the ‘lower screen does not need clearing’.
Save the current value of
Keep the current value of
This is ED-FULL.
Push this address on to the machine stack to make ED-FULL the entry point following an error.
LD HL,(ECHO-E)  Push the value of ECHO-E
PUSH HL      on to the stack.
SCF          Make HL point to the start
CALL 1195,SET-HL  of the space and DE the end.
EX DE,HL     Now print the line.
CALL 187D,OUT-LINE2
EX DE,HL     Exchange the pointers and
CALL 1BE1,OUT-CURS
LD HL,(S-POSNL)  print the cursor.
EX (SP),HL    Next fetch the Current value
LD DE,(S-POSNL)  of S-POSNL and exchange it
EX DE,HL     with ECHO-E.
CALL 0D4D,TEMPS
LD A,(S-POSNL-hi)  Again fetch the permanent
SUB D        colours.
JR C,117C,ED-C-DONE
JR NZ,115E,ED-SPACES
LD A,E      Jump forward if no 'blanking'
SUB (S-POSNL-lo)  of lines required.
JR NC,117C,ED-C-DONE
1150 ED-BLANK  Fetch the current line number
LD A,(S-POSNL-hi)  and subtract the old line number.
SUB D        Jump forward if no on the
JR C,117C,ED-C-DONE  same line.
LD A,E      Fetch the old column number
SUB (S-POSNL-lo)  and subtract the new column number.
JR NC,117C,ED-C-DONE
115E ED-SPACES  Jump if no spaces required.
LD A,+20     A 'space'.
PUSH DE      Save the old values,
CALL 09F4,PRINT-OUT
LD DE,(S-POSNL)  Print it.
POP DE      Fetch the old values.
JR 1150,ED-BLANK  Back again.
New deal with any errors.
1167 ED-FULL  Give out a 'rasp'.
LD D,+00  Fetch the current value of
LD E,(RASP)  ERR-NR) +FF  S-POSNL and jump forward.
LD HL,+1A90
CALL 03B5,BEEPER
LD (ERR-NR) +FF
LD DE,(S-POSNL)
JR 117E,ED-C-END
117C ED-C-DONE  The new position value.
POP DE  The 'error address'.
POP HL
But come here after an error.
117E ED-C-END  The old value of ERR-SP is
POP HL      restored.
LD (ERR-SP),HL  Fetch the old value of
POP BC      S-POSNL.
PUSH DE     Save the new position values.
CALL 0DD9,CL-SET
POP HL      Set the system variables.
LD (ECHO-E),HL  The old value of S-POSNL
LD (X-PTR-hi),+00  goes into ECHO-E.
RET         X-PTR is cleared in a
            suitable manner and the return
THE 'SET-HL' AND 'SET-DE' SUBROUTINES
Made.
These subroutines return with HL pointing to the first location and DE the 'last' location of either the editing area or the work space.
1190 SET-HL  Point to the last location
LD HL,(WORKSP)
DEC HL of the editing area.
AND A Clear the carry flag.

1195 SET-DE LD DE,(E-LINE) Point to the start of the
BIT 5,(FLAGX) editing area and return if
RET Z in ‘editing mode’.
LD DE,(WORKSP) Otherwise change DE.
RET C Return if now intended.
LD HL,(STKBOT) Fetch STKBOT and then
RET return.

THE ‘REMOVE-FP’ SUBROUTINE
This subroutine removes the hidden floating-point forms in a BASIC line.

11A7 REMOVE-FP LD A,(HL) Each character in turn is
CP +0E examined.
LD BC,+0006 It will occupy six locations.
CALL Z,19E8,RECLAIM-2 Reclaim the F-P number.
LD A,(HL) Fetch the code again.
INC HL Update the pointer.
CP +0D ‘Carriage return’?
JR NZ,11A7,REMOVE-FP Back if not. But make a
RET simple return if it is.
THE EXECUTIVE ROUTINES

THE 'INITIALISATION' ROUTINE
The main entry point to this routine is at START/NEW (11CB). When entered from START (0000), as when power is first applied to the system, the A register holds zero and the DE register the value +FFFF. However the main entry point can also be reached following the execution of the NEW command routine.

THE 'NEW COMMAND' ROUTINE

11B7  NEW
DI  Disable the maskable interrupt.
LD A,+FF  The NEW flag.
LD DE,(RAMTOP)  The existing value of RAMTOP is preserved.
EXX  Load the alternate registers
LD BC,(P-RAMT)  with the following system variables. All of which will
LD DE,(RASP/PIP)  also be preserved.
LD HL,(UDG)
EXX

The main entry point.

11CB  START/NEW
LD B,A  Save the flag for later.
LD A,+07  Make the border white in
OUT (+FE),A  colour.
LD A,+3F  Set the I register to hold
LD I,A  the value of +3F.
DEFB +00,+00,+00  Wait 24 T states.
DEFB +00,+00,+00

Now the memory is checked.

11DA  RAM-CHECK
LD H,D  Transfer the value in DE
LD L,E  (START = +FFFF, NEW = RAMTOP).
11DC  RAM-FILL
LD (HL),+02  Enter the value of +02 into
DEC HL  every location above +3FFF.
GP H
JR NZ,11DC,RAM-FILL
11E2  RAM-READ
AND A  Prepare for true subtraction.
SBC HL,DE  The carry flag will become
ADD HL,DE  reset when the top is reached.
INC HL  Jump when at top.
JR NC,11EF,RAM-DONE
DEC (HL)  But if zero then RAM is faulty.
DEC (HL)  Use current HL as top.
JR Z,11EF,RAM-DONE
JR Z,11E2,RAM-READ  Step to the next test unless it fails.

11EF  RAM-DONE
DEC HL  HL points to the last actual location in working order.

Next restore the 'preserved' system variables. (Meaningless when coming from START.)
EXX  Switch registers.
LD (P-RAMT),BC  Restore P-RAMT,RASP/PIP
LD (RASP/PIP),DE &UDG
LD (UDG),HL
EXX
INC B  Test the START/NEW flag.
JR Z,1219,RAM-SET  Jump forward if coming from
the NEW command routine.
Overwrite the system variables when coming from START and initialise the user-defined graphics area.

LD (P-RAMT),HL Top of physical RAM.
LD DE,+3EAF Last byte of ‘U’ in character set.
LD BC,+00A8 There are this number of bytes in twenty one letters.
EX DE,HL Switch the pointers.
LDDR DE,HL Now copy the character forms of the letter ‘A’ to ‘U’.
INC HL Point to the first byte.
LD (UDG),HL Now set UDG.
DEC HL Down one location.
LD BC,+0040 Set the system variables
LD (RASP/PIP),BC RASP & PIP.

The remainder of the routine is common to both the START and the NEW operations.

1219 RAM-SET LD (RAMTOP),HL Set RAMTOP.
LD HL,+3C00 Initialise the system variable
LD (CHARS),HL CHAR.

Next the machine stack is set up.

LD HL,(RAMTOP) The top location is made to
LD (HL),+3E hold +3E.
DEC HL The next location is left holding zero.
LD SP,HL These two locations represent the ‘last entry’.
DEC HL Step down two locations to
DEC HL find the correct value for
LD (ERR-SP),HL ERR-SP.

The initialisation routine continues with:

IM 1 Interrupt mode 1 is used.
LD IY,+5C3A IY holds +ERR-NR always.
EI The maskable interrupt can now be enabled. The real-time clock will be updated and the keyboard scanned every 1/50th of a second.

LD HL,+5CB6 The base address of the
LD (CHANS),HL channel information area.
LD DE,15AF The initial channel data
LD BC,+0015 is moved from the table
EX DE,HL (15AF) to the channel
LDIR information area.
EX DE,HL The system variable DATADD
DEC HL is made to point to the last
LD (DATADD),HL location of the channel data.
INC HL And PROG & VARS to the
LD (PROG),HL the location after that.
LD (VARS),HL
LD (HL),+80 The end-marker of the variables area.
INC HL Move on one location to find
LD (E-LINE),HL the value for E-LINE.
LF (HL),+0D Make the edit-line be a single
INC HL ‘carriage return’ character.
LD (HL),+80 Now enter an end-marker.
INC HL Move on one location to find
LD (WORKSP),HL the value for WORKSP, STKBOT & STKEND.
LD (STKBOT),HL
LD (STKEND),HL Initialise the colour system
LD A,+38
LD (ATTR-P),A variables to : FLASH 0,
LD (ATTR-T),A BRIGHT 0, PAPER 7, & INK 0.
LD HL,+0523 Initialise the system
LD (REPDEL),HL variables REPDEL & REPPER.
DEC (KSTATE-0) Make KSTATE-0 hold +FF
DEC (KSTATE-4) Make KSTATE-4 hold +FF
LD HL,+15C6 Next move the initial stream
LD DE,+5C10 data from its table to the
LD BC,+000E streams area.
LDIR SET 1,(FLAGS) Signal 'printer in use'
CALL 0EDF,CLEAR-PRB and clear the printer buffer.
CALL 0D6B,CLS part of the display and clear
XOR A the whole display.
LD DE,+1538 '© 1982 Sinclair Research Ltd'
CALL 0C0A,PO-MSG on the bottom line.
SET 5,(TV-FLAG) Signal 'the lower part will
required to be cleared.
JR 12A9,MAIN-1 Jump forward into the main
execution loop.
THE 'MAIN EXECUTION' LOOP
The main loop extends from location 12A2 to location 15AE and it controls the 'editing mode', the execution of direct commands and the production of reports.

12A2 MAIN-EXEC LD (DF-SZ),+02 The lower part of the screen
CALL 1795,AUTO-LIST is to be two lines in size.
CALL 16B0,SET-MIN Produce an automatic listing.
12A9 MAIN-1 CALL All the areas from E-LINE onwards are given their
LD DE,+00 minimum configurations.
12AC MAIN-2 CALL 1601,CHAN-OPEN Channel 'K' is opened before
CALL 0F2C,EDITOR calling the EDITOR.
CALL 1B17,LINE-SCAN The EDITOR is called to allow
BIT 7,(ERR-NR) the user to build up a BASIC line.
JR NZ,12CF,MAIN-3 Jump forward if the syntax is
correct.
BIT 4,(FLAGS2) Jump forward if other than
JR Z,1303,MAIN-4 channel 'K' is being used.
LD HL,(E-LINE)
CALL 11A7,REMOVE-FP Point to the start of the line
LD (ERR-NR),+FF with the error.
JR 12AC,MAIN-2 Remove the floating-point
LD (ERR-NR),+FF forms from this line.
JR 12AC,MAIN-2 Reset ERR-NR and jump back
LD A,B to MAIN-2 leaving the listing
OR C unchanged.
JR NZ,155D,MAIN-ADD The 'edit-line' has passed syntax and the three types of line that are possible have to be distinguished from each other.
12CF MAIN-3 LD HL,(E-LINE) Point to the start of the line.
LD (CH-ADD),HL Set CH-ADD to the start also.
CALL 19FB,E-LINE-NO Fetch any line number into BC.
LD A,B Is the line number a valid
OR C one?
JR NZ,155D,MAIN-ADD Jump if it is so, and add the new
LD HL,(E-LINE) line to the existing program.
Fetch the first character of the line and see if the line is 'carriage return only'.

If it is then jump back.

The 'edit-line' must start with a direct BASIC command so this line becomes the first line to be interpreted.

BIT 0,(FLAGS2) Clear the whole display unless the flag says it is unnecessary.
CALL 006E,CLS-LOWER Clear the lower part anyway.
LD A,+19 Set the appropriate value for the scroll counter.
SUB (S-POSN-hi) for the scroll counter.
LD (SCR-CT),A
SET 7,(FLAGS) Signal 'line execution'.
LD (ERR-NR),+FF Ensure ERR-NR is correct.
LD (NSPPC),+01 Deal with the first statement in the line.
CALL 1B8A,PROG-RUN Now the line is interpreted.

Note: The address 1303 goes on to the machine stack and is addressed by ERR-SP.

After the line has been interpreted and all the actions consequential to it have been completed a return is made to MAIN-4, so that a report can be made.

1303 MAIN-4 HALT The maskable interrupt must be enabled.
RES 5,(FLAGS) Signal 'ready for a new key'.
BIT 1,(FLAGS2) Empty the printer buffer if it has been used.
CALL NZ,0ECD,COPY-BUFF Fetch the error number and increment it.
INC A

1313 MAIN-G PUSH AF Save the new value.
LD HL,+0000 The system variables
LD (FLAGX),H FLAGX, X-PTR &
LD (X-PTR-hi),H DEFADD are all set to zero.
LD (DEFADD),HL Ensure that stream +00 points to channel 'K'
LD HL,+0001 Clear all the work areas and the calculator stack.
CALL 16B0,SET-MIN
RES 5,(FLAGX) Signal 'editing mode'.
CALL 006E,CLS-LOWER Signal 'the lower screen will require clearing'.
SET 5,(TV-FLAG) Call the lower screen.

POP AF Fetch the report value.
LD B,A Make a copy in B.
CP +0A Jump forward with report numbers '0 to 9'.
JR C,133C,MAIN-5 Add the ASCII letter offset value.
ADD A,+07

133C MAIN-5 CALL 15EF,OUT-CODE Print the report code and follow it with a 'space'.
LD A,+20
RST 0010,PRINT-A-1
LD A,B Fetch the report value and use it to identify the required report message.
LD DE,+1391 Print the message and follow it by a 'comma' and a 'space'.
CALL 0C0A,PO-MSG
XOR A
LD DE,+1536 Now fetch the current line
CALL 0C0A,PO-MSG
LD BC,(PPC) number and print it as well.
LD A,+3A Follow it by a ':
RST 0010,PRINT-A-1
LD C,(SUBPPC)  Fetch the current statement
LD B,+00  number into the BC register
CALL 1A1B,OUT-NUM1 pair and print it.
CALL 1097,CLEAR-SP Clear the editing area.
LD A,(ERR-NR)  Fetch the error number again.
INC A  Increment it as usual.
JR Z,1386,MAIN-9 If the program was completed
               successfully there cannot be
               any 'CONTinuing' so jump.
CP +09  If the program halted with
       'STOP statement' or 'BREAK
JR Z,1373,MAIN-6' into program' CONTinuing will
CP +15  be from the next statement;
JR NZ,1376,MAIN-7 otherwise SUBPPC is unchanged.
1373  MAIN-6  INC (SUBPPC)
1376  MAIN-7  LD BC,+0003  The system variables OLDPPC
               & OSPCC have now to be made
               to hold the CONTinuing line
               and statement numbers.
LD HL,+5C44  The values used will be those in
BIT 7,(NSPPC)
JR Z,1384,MAIN-8 PPC & SUBPPC unless NSPPC
ADD HL,BC occurred before a 'jump'.
1384  MAIN-8  LDDR (i.e. after a GO TO statement
1386  MAIN-9  LD (NSPPC),+FF etc.)
RES 3,(FLAGS)  NSPPC is reset to indicate
JP 12AC,MAIN-2 'no jump'.
               'K mode' is selected.
               And finally the jump back is
               made but no program listing
               will appear until requested.

THE REPORT MESSAGES
Each message is given with the last character inverted (+80 hex.).

1391  DEFB +80  - initial byte is stepped over.
1392  Report 0  - 'OK'
1394  Report 1  - 'NEXT without FOR'
13A4  Report 2  - 'Variable not found'
13B6  Report 3  - 'Subscript wrong'
13C6  Report 4  - 'Out of memory'
13D2  Report 5  - 'Out of screen'
13E6  Report 6  - 'Number too big'
13EB  Report 7  - 'RETURN without GOSUB'
1401  Report 8  - 'End of file'
140C  Report 9  - 'STOP statement'
141A  Report A  - 'Invalid argument'
142A  Report B  - 'Integer out of range'
143E  Report C  - 'Nonsense in BASIC'
144F  Report D  - 'BREAK - CONT repeats'
1463  Report E  - 'Out of DATA'
146E  Report F  - 'Invalid file name'
147F  Report G  - 'No room for line'
148F  Report H  - 'STOP in INPUT'
149C  Report I  - 'FOR without NEXT'
14AC  Report J  - 'Invalid I/O device'
14BE  Report K  - 'Invalid colour'
14CC  Report L  - 'BREAK into program'
14DE  Report M  - 'RAMTOP no good'
14EC  Report N  - 'Statement lost'
14FA  Report O  - 'Invalid stream'
1508  Report P  - 'FN without DEF'
1516  Report Q  - 'Parameter error'
1525  Report R  - 'Tape loading error'
There are also the following two messages.

1537 ' , ' - a 'comma' and a 'space'
1539 '© 1982 Sinclair Research Ltd'

Report G - No room for line

1555 REPORT-G LD A,+10 'G' has the code '10+07+30'
LD BC,+0000 Clear BC.
JP 1313,MAIN-G Jump back to give the report.

THE 'MAIN-ADD' SUBROUTINE
This subroutine allows for a new BASIC line to be added to the existing BASIC program in the program area. If a line has both an old and a new version then the old one is 'reclaimed'. A new line that consists of only a line number does not go into the program area.

155D MAIN-ADD LD (E-PPC),BC Make the new line number the 'current line'.
LD HL,(CH-ADD) Fetch CH-ADD and save the address in DE.
EX DE,HL
LD HL,+1555 Push the address of REPORT-G on to the machine stack.
PUSH HL ERR-SP will now point to REPORT-G.
LD HL,(WORKSP) Fetch WORKSP.
SCF Find the length of the line
SBC, HL,DE from after the line number to the 'carriage return' character inclusively.
PUSH HL Save the length.
LD H,B Move the line number to the HL register pair.
LD L,C HL
CALL 196E,LINE-ADDR Is there an existing line with this number?
JR NZ,157D,MAIN-ADD1 Jump if there was not.
CALL 19B8,NEXT-ONE Find the length of the 'old' line and reclaim it.
CALL 19E8,RECLAIM-2
157D MAIN-ADD1 POP BC Fetch the length of the
LD A,C 'new' line and jump forward
DEC A if it is only a 'line number
OR B and a carriage return'.
JR 15AB,MAIN-ADD2
PUSH BC Save the length.
INC BC Four extra locations will be needed.
INC BC i.e. two for the number &
INC BC two for the length.
DEC HL Make HL point to the location before the 'destination'.
LD DE,(PROG) Save the current value of
PUSH DE PROG to avoid corruption when adding a first line.
CALL 1655,MAKE-ROOM Space for the new line is created.
POP HL The old value of PROG is
LD (PROG),HL fetched and restored.
POP BC A copy of the line length
PUSH BC (without parameters) is taken.
INC DE Make DE point to the end
LD HL,(WORKSP) location of the new area
DEC HL and HL to the 'carriage
DEC HL return' character of the new
LDDR line in the editing area.
LD HL,(E-PPC) Now copy over the line.
Fetch the line's number.
EX DE,HL  Destination into HL & number into DE.
POP BC  Fetch the new line's length.
LD (HL),B  The high length byte.
DEC HL  The low length byte.
LD (HL),C  The low line number byte.
DEC HL  The high line number byte.
LD (HL),D  The high line number byte.

15AB MAIN-ADD2 POP AF  Drop the address of REPORT-G.
JP 12A2,MAIN-EXEC  Jump back and this time do produce and automatic listing.

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.

15AF DEFB F4 09  - PRINT-OUT
DEFB A8 10  - KEY-INPUT
15B4 DEFB F4 09  - PRINT-OUT
DEFB C4 15  - REPORT-J
DEFB 53  - 'S'
15B9 DEFB 81 0F  - ADD-CHAR
DEFB C4 15  - REPORT-J
DEFB 52  - 'R'
15BE DEFB F4 09  - PRINT-OUT
DEFB C4 15  - REPORT-J
DEFB 50  - 'P'
15C3 DEFB 80  - End marker.

Report J - Invalid I/O device
15C4 REPORT-J RST 0008,ERROR-1 Call the error handling routine
DEFB +12

THE 'INITIAL STREAM DATA'
Initially there are seven streams - +FD to +03.

15C6 DEFB 01 00  - stream +FD leads to channel 'K'
15C8 DEFB 06 00  - stream +FE " " 'S'
15CA DEFB 0B 00  - stream +FF " " 'R'
15CC DEFB 01 00  - stream +00 " " 'K'
15CE DEFB 01 00  - stream +01 " " 'K'
15D0 DEFB 06 00  - stream +02 " " 'S'
15D2 DEFB 10 00  - stream +03 " " 'P'

THE 'WAIT-KEY' SUBROUTINE
This subroutine is the controlling subroutine for calling the current input subroutine.

15D4 WAIT-KEY BIT 5,(TV-FLAG) Jump forward if the flag
JR NZ,15DE,WAIT-KEY1 indicates the lower screen
does not require clearing.
SET 3,(TV-FLAG) Otherwise signal 'consider
the mode as having changed'.
15DE WAIT-KEY1 CALL 15E6,INPUT-AD Call the input subroutine
RET C indirectly via INPUT-AD.
JR Z,15DE,WAIT-KEY1 Return with acceptable codes.
Both the carry flag and the zero flag are reset if 'no key is being pressed'; otherwise signal an error.
THE 'INPUT-AD' SUBROUTINE
The registers are saved and HL made to point to the input address.

THE 'MAIN PRINTING' SUBROUTINE
The subroutine is called with either an absolute value or a proper character code in the A register.

THE 'CHAN-OPEN' SUBROUTINE
This subroutine is called with a valid stream number - normally +FD to +03. Then depending on the stream data a particular channel will be made the current channel.
ADD HL,DE  Form the required address in this area.

THE 'CHAN-FLAG' SUBROUTINE
The appropriate flags for the different channels are set by this subroutine.

1615  CHAN-FLAG LD (CURCHL),HL The HL register pair holds the base address for a particular channel.
      RES  4,(FLAGS2) Signal 'using other than channel 'K'.
      INC HL Step past the output
      INC HL and the input addresses and
      INC HL make HL point to the
      LD C,(HL) channel code.
      LD HL,+162D The base address of the 'channel code look-up table'.
      CALL 16DC,INDEXER Index into this table and locate the required offset; but return if there is not a matching channel code.
      RET NC
      LD D,+00 Pass the offset to the
      LD E,(HL) DE register pair.
      ADD HL,DE Jump forward to the appropriate
      JL (HL) flag setting routine.

162C CALL-JUMP JP

THE 'CHANNEL CODE LOOK-UP' TABLE

162D DEFB 48 06 - channel 'K', offset +06, address 1634
162F DEFB 53 12 - channel 'S', offset +12, address 1642
1631 DEFB 50 1B - channel 'P', offset +1B, address 164D
1633 DEFB 00 - end marker.

THE 'CHANNEL 'K' FLAG' SUBROUTINE

1634  CHAN-K SET 0,(TV-FLAG) Signal 'using lower screen'.
      RES  5,(FLAGS) Signal 'ready for a key'.
      SET  4,(FLAGS2) Signal 'using channel 'K'.
      JR 1646,CHAN-S-1 Jump forward.

THE 'CHANNEL 'S' FLAG' SUBROUTINE

1642  CHAN-S RES 0,(TV-FLAG) Signal 'using main screen'.
1646  CHAN-S-1 RES 1,(FLAGS) Signal 'printer not being used'.
      JP 004D,TEMPS Exit via TEMPS so as to set the colour system variables.

THE 'CHANNEL 'P' FLAG' SUBROUTINE

164D  CHAN-P SET 1,(FLAGS) Signal 'printer in use'.
      RET

THE 'MAKE-ROOM' SUBROUTINE
This is a very important subroutine. It is called on many occasions to 'open up' an area. In all cases the HL register pair points to the location after the place where 'room' is required and the BC register pair holds the length of the 'room' needed. When a single space only is required then the subroutine is entered at ONE-SPACE.

1652  ONE-SPACE LD BC,+0001 Just the single extra location is required.
1655  MAKE-ROOM PUSH HL Save the pointer.
      CALL 1F05,TEST-ROOM Make sure that there is sufficient memory available for the task being undertaken.
      POP HL Restore the pointer.
CALL 1664,POINTERS  Alter all the pointers before
making the 'room'.
LD  HL,(STKEND)  Make HL hold the new STKEND.
EX  DE,HL  Switch 'old' and 'new'.
LDDR  Now make the 'room'
RET  and return.

Note: This subroutine returns with the HL register pair pointing to the location before the new 'room' and the DE register pair pointing to
the last of the new locations. The new 'room' therefore has the description: '(HL)+1' to '(DE)+1'.
However as the 'new locations' still retain their 'old values' it is also possible to consider the new 'room' as having been made after the
original location '(HL)' and it thereby has the description: '(HL)+2' to (DE)+1'.
In fact the programmer appears to have a preference for the 'second description' and this can be confusing.

THE 'POINTERS' SUBROUTINE
Whenever an area has to be 'made' or 'reclaimed' the system variables that address locations beyond the 'position' of the change have
to be amended as required. On entry the BC register pair holds the number of bytes involved and the HL register pair addresses the
location before the 'position'.

1664 POINTERS  PUSH  AF  These registers are saved.
PUSH  HL  Copy the address of the
'position'.
LD  HL,+5C4B  This is VARS, the first of the
LD  A,+0E  fourteen system pointers.

A loop is now entered to consider each pointer in turn. Only those pointers that point beyond the 'position' are changed.

166B PTR-NEXT  LD  E,(HL)  Fetch the two bytes of the
INC  HL  current pointer.
LD  D,(HL)  Exchange the system variable
EX  (SP),HL  with the address of the 'position'.
AND  A  The carry flag will become
SBC  HL,DE  set if the system variable's
ADD  HL,DE  address is to be updated.
EX  (SP),HL  Restore the 'position'.
JR  NC,167F,PTR-DONE  Jump forward if the pointer is
to be left; otherwise change it.
PUSH  DE  Save the old value.
EX  DE,HL  Now add the value in BC
to the old value.
ADD  HL,BC  
EX  DE,HL  
LD  (HL),D  Enter the new value into the
DEC  HL  system variable - high byte
LD  (HL),E  before low byte.
INC  HL  Point again to the high byte.
POP  DE  Fetch the old value.
167F PTR-DONE  INC  HL  Point to the next system
DEC  A  variable and jump back until all
JR  NZ,166B,PTR-NEXT  fourteen have been considered.

Now find the size of the block to be moved.

EX  DE,HL  Put the old value of STKEND in
POP  DE  HL and restore the other
POP  AF  registers.
AND  A  Now find the difference
SBC  HL,DE  between the old value of
LD  B,H  STKEND and the 'position'.
LD  C,L  Transfer the result to BC
INC  BC  and add '1' for the inclusive
byte.
THE 'COLLECT A LINE NUMBER' SUBROUTINE

On entry the HL register pair points to the location under consideration. If the location holds a value that constitutes a suitable high byte for a line number then the line number is returned in DE. However if this is not so then the location addressed by DE is tried instead; and should this also be unsuccessful line number zero is returned.

168F LINE-ZERO DEFB +00 Line number zero.
1691 LINE-NO-A EX DE,HL Consider the other pointer.
      LD DE,+168F Use line number zero.

The usual entry point is at LINE-NO.

1695 LINE-NO LD A, (HL) Fetch the high byte and
      JR NZ,1691,LINE-NO-A test it.
      LD D, (HL) Fetch the high byte.
      INC HL LD E, (HL) Fetch the low byte and
      RET return.

THE 'RESERVE' SUBROUTINE

This subroutine is normally called by using RST 0030,BC-SPACES.
On entry here the last value on the machine stack is WORKSP and the value above it is the number of spaces that is to be 'reserved'. This subroutine always makes 'room' between the existing work space and the calculator stack.

169E RESERVE LD HL, (STKBOT) Fetch the current value of
      DEC HL STKBOT and decrement it to
      CALL 1655,MAKE-ROOM get the last location of the
      INC HL work space.
      INC HL Point to the first new space
      POP BC Fetch the old value of
      LD (WORKSP), BC WORKSP and restore it.
      POP BC Restore BC - number of spaces.
      EX DE, HL Switch the pointers.
      INC HL Make HL point to the first of
      RET the displaced bytes.

Note: It can also be considered that the subroutine returns with the DE register pair pointing to a 'first extra byte' and the HL register pair pointing to a 'last extra byte', these extra bytes having been added after the original '(HL)+1' location.

THE 'SET-MIN' SUBROUTINE

This subroutine resets the editing area and the areas after it to their minimum sizes. In effect it 'clears' the areas.

16B0 SET-MIN LD HL, (E-LINE) Fetch E-LINE.
      LD (HL), +0D Make the editing area hold
      LD (K-CUR), HL only the 'carriage return'
      INC HL character and the end marker.
      LD (HL), +80 Move on to clear the work
      INC HL space.
      LD (WORKSP), HL

Entering here will 'clear' the work space and the calculator stack.

16BF SET-WORK LD HL, (WORKSP) Fetch the WORKSP.
      LD (STKBOT), HL This clears the work space.
Entering here will 'clear' only the calculator stack.

16C5 SET-STK
LD HL,(STKBOT)
LD (STKEND),HL
Fetch STKBOT.
This clears the stack.

In all cases make MEM address the calculator's memory area.

PUSH HL
Save STKEND.
LD HL,+5C92
The base of the memory area.
LD (MEM),HL
Set MEM to this address.
POP HL
Restore STKEND to the HL
RET register pair before returning.

THE 'RECLAIM THE EDIT-LINE' SUBROUTINE

16D4 REC-EDIT
LD DE,(E-LINE)
JP 19E5,RECLAIM-1
Fetch E-LINE.
Reclaim the memory.

THE 'INDEXER' SUBROUTINE
This subroutine is used on several occasions to look through tables. The entry point is at INDEXER.

16DB INDEXER-1
INC HL
Move on to consider the next pair of entries.
16DC INDEXER
LD A,(HL)
Fetch the first of a pair of entries but return if it is
AND Z zero - the end marker.
CP C Compare it to the supplied code.
INC HL Point to the second entry.
JR NZ,16DB,INDEXER-1 Jump back if the correct entry
SCF The carry flag is set upon a successful search.

THE 'CLOSE #' COMMAND ROUTINE
This command allows the user to CLOSE streams. However for streams +00 to +03 the 'initial' stream data is restored and these streams cannot therefore be CLOSEd.

16E5 CLOSE
CALL 171E,STR-DATA The existing data for the stream is fetched.
CALL 1701,CLOSE-2 Check the code in that stream's channel.
LD BC,+0000 Prepare to make the stream's data zero.
LD DE,+A3E2 Prepare to identify the use of streams +00 to +03.
EX DE,HL The carry flag will be set with streams +04 to +0F.
ADD HL,DE
JR C,16FC,CLOSE-1 Jump forward with these streams; otherwise find the correct entry in the 'initial stream data' table.
LD BC,+15D4
ADD HL,BC
LD C,(HL)
INC HL
LD B,(HL)
Now enter the data; either zero & zero, or the initial values.

16FC CLOSE-1
EX DE,HL
LD (HL),C
INC HL
LD (HL),B
RET

THE 'CLOSE-2' SUBROUTINE
The code of the channel associated with the stream being closed has to be 'K', 'S', or 'P'.

1701 CLOSE-2
PUSH HL Save the address of the stream's data.
LD HL,(CHANS) Fetch the base address of the channel information area and find the channel data for the stream being CLOSED.
ADD HL,BC Step past the subroutine addresses and pick up the code for that channel.
INC HL
INC HL
INC HL
LD C,(HL) Save the pointer.
EX DE,HL
LD HL,+1716 The base address of the 'CLOSE stream look-up' table.
CALL 16DC,INDEXER Index into this table and locate the required offset.
LD C,(HL) Pass the offset to the BC register pair.
LD B,+00
ADD HL,BC Jump forward to the appropriate routine.
JP (HL)

THE 'CLOSE STREAM LOOK-UP' TABLE

| 1716 | DEFB 4B 05 | - channel 'K', offset +05, address 171C |
| 1718 | DEFB 53 03 | - channel 'S', offset +03, address 171C |
| 171A | DEFB 50 01 | - channel 'P', offset +01, address 171C |

Note: There is no end marker at the end of this table.

THE 'CLOSE STREAM' SUBROUTINE.

| 171C | CLOSE-STR POP HL Fetch the channel information pointer and return. |
| RET |

THE 'STREAM DATA' SUBROUTINE

This subroutine returns in the BC register pair the stream data for a given stream.

| 171E | STR-DATA CALL 1E94,STK-TO-A The given stream number is taken off the calculator stack. |
| CP | +10 Give an error if the stream number is greater than +0F. |
| JR C,1727,STR-DATA Jump forward to the appropriate routine. |

Report O - Invalid stream

| 1725 | REPORT-O RST 0008,ERROR-1 Call the error handling routine. |
| DEF B,+17 |

Continue with valid stream numbers.

| 1727 | STR-DATA ADD A,+03 Range now +03 to +12; |
| ADD | A,+03 and now +06 to +24. |
| RLCA | |
| LD | HL,+5C10 The base address of the stream data area. |
| LD | C,A Move the stream code to the BC register pair. |
| ADD | HL,BC Index into the data area and fetch the two data bytes |
| LD | C,(HL) into the BC register pair. |
| INC | HL |
| LD | B,(HL) Make the pointer address the first of the data bytes before returning. |
| DEC | HL |
| RET |

THE 'OPEN #' COMMAND ROUTINE

This command allows the user to OPEN streams. A channel code must be supplied and it must be 'K', 'k', 'S', 's', 'P', or 'p'. Note that no attempt is made to give streams +00 to +03 their initial data.

| 1736 | OPEN RST 0028,FP-CALC Use the CALCULATOR. |
| DEF B,+01,exchange Exchange the stream number |
DEFB +38,end-calc and the channel code.
CALL 171E,STR-DATA Fetch the data for the stream.
LD A,B Jump forward if both bytes of
OR C the data are zero, i.e. the
JR Z,1756,OPEN-1 stream was in a closed state.
EX DE,HL Save DE.
LD HL,(CHANS) Fetch CHANS - the base
ADD HL,BC address of the channel
INC HL information and find the
INC HL code of the channel
INC HL associated with the stream
LD A,(HL) being OPENed.
EX DE,HL Return DE.
CP +4B The code fetched from the
JR Z,1756,OPEN-1 channel infor
CP +53 must be 'K', 'S' or 'P';
JR Z,1756,OPEN-1 give an error if it is not.
CP +50 JR NZ,1767,OPEN-3
LD (HL),E Enter the data into the
INC HL two bytes in the stream
LD (HL),D information area.
RET Finally return.

THE 'OPEN-2' SUBROUTINE
The appropriate stream data bytes for the channel that is associated with the stream being OPENed are found.

175D OPEN-2 PUSH HL Save HL
CALL 2BF1,STK-FETCH Fetch the parameters of the
LD A,B channel code.
OR C Give an error if the
JR NZ,1767,OPEN-3 expression supplied is a null expression; i.e. OPEN #5,"".

Report F - Invalid file name

1765 REPORT-F RST 0008,ERROR-1 Call the error handling
DEFB +0E routine.

Continue if no error occurred.

1767 OPEN-3 PUSH BC The length of the expression
LD A,(DE) is saved.
AND +DF Fetch the first character.
LD C,A Convert lower case codes to upper case ones.
LD HL,+177A Move code to the C register.
CALL 16DC,INDEXER The base address of the
JR NC,1765,REPORT-F 'OPEN stream look-up' table.
LD C,(HL) Pass the offset to the BC
LD B,+00 register pair.
ADD HL,BC Make HL point to the start of
POP BC the appropriate subroutine.
JP (HL) Fetch the length of the
expression before jumping to
the subroutine.

THE 'OPEN STREAM LOOK-UP' TABLE
177A DEFB 4B 06 - channel 'K', offset +06, address 1781
THE 'OPEN-K' SUBROUTINE

1781 OPEN-K LD E,+01 The data bytes will be +01 & +00.
JR 178B,OPEN-END

THE 'OPEN-S' SUBROUTINE

1785 OPEN-S LD E,+06 The data bytes will be +06 & +00.
JR 178B,OPEN-END

THE 'OPEN-P' SUBROUTINE

1789 OPEN-P LD E,+10 The data bytes will be +10 & +00.
JR 178B,OPEN-END

178B OPEN-END DEC BC Decrease the length of the
LD A,B expression and give an error
OR C if it was not a single
JR NZ,1765,REPORT-F character; otherwise clear the
LD D,A D register, fetch HL and
POP HL return.

RET

THE 'CAT, ERASE, FORMAT & MOVE' COMMAND ROUTINES

In the standard SPECTRUM system the use of these commands leads to the production of report O - Invalid stream.

1793 CAT-ETC. JR 1725,REPORT-O Give this report.

THE 'LIST & LLIST' COMMAND ROUTINES

The routines in this part of the 16K program are used to produce listings of the current BASIC program. Each line has to have its line number evaluated, its tokens expanded and the appropriate cursors positioned.

The entry point AUTO-LIST is used by both the MAIN EXECUTION routine and the EDITOR to produce a single page of the listing.

1795 AUTO-LIST LD (LIST-SP),SP The stack pointer is saved allowing the machine stack to be reset when the listing is finished. (see PO-SCR,0C55)
LD (TV-FLAG),+10 Signal 'automatic listing in the main screen'.
CALL 0DAF,CL-ALL Clear this part of the screen.
SET 0,(TV-FLAG) Switch to the editing area.
LD B,(DF-SZ) Now clear the lower part
CALL 0E44,CL-LINE of the screen as well.
RES 0,(FLAGS2) Signal 'screen is clear'.
LD DE,(E-PPC) Now fetch the 'current' line number and the 'automatic' line number.
LD DE,(S-TOP) Then switch back.
AND A If the 'current' number is
SBC HL,DE less than the 'automatic'
ADD HL,DE number then jump forward to
JR C,17E1,AUTO-L-2 update the 'automatic' number.

The 'automatic' number has now to be altered to give a listing with the 'current' line appearing near the bottom of the screen.

PUSH DE Save the 'automatic' number,
CALL 196E,LINE-ADDR Find the address of the
LD DE,+02C0 start of the 'current' line
EX DE,HL and produce an address roughly
A loop is now entered. The 'automatic' line number is increased on each pass until it is likely that the 'current' line will show on a listing.

Now the 'automatic' listing can be made.

THE 'LLIST' ENTRY POINT
The printer channel will need to be opened.

THE 'LIST' ENTRY POINT
The 'main screen' channel will need to be opened.

RES 4,(TV-FLAG) The return will be to here unless scrolling was needed to show the current line.

LD (S-TOP),DE Now S-TOP can be updated and the test repeated with the new line.

LD (S-TOP),HL When E-PPC is less than S-TOP.

LD HL,(S-TOP) Fetch the top line's number and hence its address.

CALL 196E.LINE-ADDR If the line cannot be found use DE instead.

DE.HL Perform the computation and collect its line number.

INC HL JR 17CE,AUTO-L-1

ADD HL,BC JR 17CE,AUTO-L-1

JR 17CE,AUTO-L-1

THE 'LLIST' ENTRY POINT
The printer channel will need to be opened.

THE 'LIST' ENTRY POINT
The 'main screen' channel will need to be opened.

LD A,+03 Use stream +03.

JR 17FB,LIST-1 Jump forward.
Come here if the stream was unaltered.

181F LIST-4 CALL 1CDE,FETCH-NUM Fetch any line or use zero if none supplied.

1822 LIST-5 CALL 1BEE,CHECK-END If checking the syntax of the edit-line move on to the next statement.

CALL 1E99,FIND-INT Line number to BC.
LD A,B High byte to A.
AND +3F Limit the high byte to the correct range and pass the whole line number to HL.
LD L,C
LD (E-PPC),HL Set E-PPC and find the address of the start of this line or the first line after it if the actual line does not exist.

1833 LIST-ALL LD E,+01 Flag 'before the current line'.

Now the controlling loop for printing a series of lines is entered.

1835 LIST-ALL-1 CALL 1855,OUT-LINE Print the whole of a BASIC line.
RST 0010,PRINT-A-1 This will be a 'carriage return'.
BIT 4,(TV-FLAG) Jump back unless dealing
JR Z,1835,LIST-ALL-1 with an automatic listing.
LD A,(DF-SZ) Also jump back if there is still part of the main screen that can be used.
SUB (S-POSN-hi) JR NZ,1835,LIST-ALL-1 A return can be made at this point if the screen is full and the current line has been printed (E = +00)
PUSH HL However if the current line is missing from the listing
PUSH DE POP HL
LD HL,+5C6C (using scrolling).
CALL 190F,LN-FETCH and a further line printed
POP DE (E = +01)
POP HL JR 1835,LIST-ALL-1

THE 'PRINT A WHOLE BASIC LINE' SUBROUTINE
The HL register pair points to the start of the line - the location holding the high byte of the line number.
Before the line number is printed it is tested to determine whether it comes before the 'current' line, is the 'current' line or comes after.

1855 OUT-LINE LD BC,(E-PPC) Fetch the 'current' line number and compare it.
CALL 1980,CP-LINES Pre-load the D register with the current line number.
LD D,+3E Jump forward if printing the 'current' line.
JR Z,1865,OUT-LINE1 Load the D register with zero (it is not the cursor) and set E to hold +01 if the line is before the 'current' line and +00 if after. (The carry flag comes from CP-LINES.)
LD DE,+0000
RL E

1865 OUT-LINE1 LD (BREG),E Save the line marker.
LD A,(HL) Fetch the high byte of the line number and make a full return if the listing has been finished.
CP +40
POP BC
RET NC
PUSH BC CALL 1A28,OUT-NUM-2 The line number can now be printed - with leading spaces.
INC HL  Move the pointer on to address
INC HL  the first command code in
INC HL  the line.
RES 0,(FLAGS)  Signal 'leading space allowed'
LD A,D  Fetch the cursor code and
AND A  jump forward unless the
JR Z,18B1,OUT-LINE3  cursor is to be printed.
RST 0010,PRINT-A-1  So print the cursor now.

187D OUT-LINE2 SET 0,(FLAGS)  Signal 'no leading space now'.
1881 OUT-LINE3 PUSH DE  Save the registers.
EX DE,HL  Move the pointer to DE.
RES 2,(FLAGS2)  Signal 'not in quotes'.
LD HL,+5C3B  This is FLAGS.
RES 2,(HL)  Signal 'print in K-mode'.
BIT 5,(FLAGX)  Jump forward unless in
JR 1894,OUT-LINE4  INPUT mode.
SET 2,(HL)  Signal 'print in L-mode'.

Now enter a loop to print all the codes in the rest of the BASIC line - jumping over floating-point forms as necessary.

1894 OUT-LINE4 LD HL,(X-PTR)  Fetch the syntax error
AND A  pointer and jump forward
SBC HL,DE  unless it is time to print
JR NZ,18A1,OUT-LINE5  the error marker.
LD A,+3F  Print the error marker now.
CALL 18C1,OUT-FLASH  It is a flashing '?'.
18A1 OUT-LINE5 CALL 18E1,OUT-CURS  Consider whether to print the
EX DE,HL  cursor.
LD A,(HL)  Move the pointer to HL now.
CALL 18B6,ZER  If the character is a 'number
INC HL  marker' then the hidden floating-
CP +0D  point form is not to be printed.
JR Z,18B4,OUT-LINE6  It is if it is.
EX DE,HL  Switch the pointer to DE.
CALL 1937,OUT-CHAR  Print the character.
JR 1894,OUT-LINE4  Go around the loop for at least

The line has now been printed.

18B4 OUT-LINE6 POP DE  Restore the DE register pair
RET and return.

THE 'NUMBER' SUBROUTINE
If the A register holds the 'number marker' then the HL register pair is advanced past the floating-point form.

18B6 NUMBER CP +0E  Is the character a 'number
RET NZ  marker'. Return if not.
INC HL  Advance the pointer six
INC HL  times so as to step past the
INC HL  'number marker' and the five
INC HL  locations holding the
INC HL  floating-point form.
INC HL  
LD A,(HL)  Fetch the current code before
RET returning.
THE 'PRINT A FLASHING CHARACTER' SUBROUTINE
The 'error cursor' and the 'mode cursors' are printed using this subroutine.

18C1 OUT-FLASH
EXX
LD HL,(ATTR-T)
LD HL,7,H
RES 7,L
LD (ATTR-T),HL
LD HL,+5C91
LD D,(HL)
PUSH HL
RES 7,H
LD (ATTR-T),HL
LD HL,+00
CALL 09F4,PRINT-OUT
POP HL
LD D,(HL)
PUSH HL
RES 7,L
LD (ATTR-T),HL
LD HL,+00
CALL 09F4,PRINT-OUT
POP HL
EXX
RET

THE 'PRINT THE CURSOR' SUBROUTINE
A return is made if it is not the correct place to print the cursor but if it is then either 'C', 'E', 'G', 'K' or 'L' will be printed.

18E1 OUT-CURS
LD HL,(K-CUR)
AND A
SBC HL,DE
RET NZ
LD A,(MODE)
RLC A
JR Z,18F3,OUT-C-1
ADD A,+43
JR 1909,OUT-C-2
LD HL,+5C3B
RES 3,(HL)
LD A,+4B
BIT 2,(HL)
JR Z,1909,OUT-C-2
SET 3,(HL)
INC A
BIT 3,(FLAGS2)
JR Z,1909,OUT-C-2
LD A,+43
PUSH DE
CALL 18C1,OUT-FLASH
POP DE
RET

The 'LN-FETCH' SUBROUTINE
This subroutine is entered with the HL register pair addressing a system variable - S-TOP or E-PPC. The subroutine returns with the system variable holding the line number of the following line.

190F LN-FETCH
LD E,(HL)
INC HL
The line number held by the system variable is collected.
LD D,(HL) The pointer is saved.
PUSH HL The line number is moved to the
EX DE,HL HL register pair and incremented.
INC HL The address of the start of this
CALL 196E,LINE-ADDR line is found, or the next line
if the actual line number is not
CALL 1695,LINE-NO being used.
PUSH HL The number of that line is
POP HL fetched.
The pointer to the system
variable is restored.

The entry point LN-STORE is used by the EDITOR.

191C LN-STORE BIT 5,(FLAGX) Return if in 'INPUT mode';
RET NZ otherwise proceed to
LD (HL),D enter the line number into
DEC HL the two locations of the
LD (HL),E system variable.
RET Return when it has been done.

THE 'PRINTING CHARACTERS IN A BASIC LINE' SUBROUTINE
All of the character/token codes in a BASIC line are printed by repeatedly calling this subroutine.
The entry point OUT-SP-NO is used when printing line numbers which may require leading spaces.

1925 OUT-SP-2 LD A,E The A register will hold +20 for
AND A a space or +FF for no-space.
RET M there is not to be a space.
JR 1937,OUT-CHAR Jump forward to print a space
192A OUT-SP-NO XOR A Clear the A register.

The HL register pair holds the line number and the BC register the value for 'repeated subtraction'. (BC holds `-1000, -100 or -10'.)

192B OUT-SP-1 ADD HL,BC The 'trial subtraction'.
INC A Count each 'trial'.
JR C,192B,OUT-SP-1 Jump back until exhausted.
SBC HL,BC Restore last 'subtraction'
DEC A and discount it.
JR Z,1925,OUT-SP-2 If no 'subtractions' were possible
JR 15EF,OUT-CHAR jump back to see if a space is to
be printed.
JP 15EF,OUT-CHAR Otherwise print the digit.

The entry point OUT-CHAR is used for all characters, tokens and control characters.

1937 OUT-CHAR CALL 2D1B,NUMERIC Return carry reset if handling a
call digit code.
JR NC,196C,OUT-CH-3 Jump forward to print the digit.
CP +21 Also print the control
JR C,196C,OUT-CH-3 characters and 'space'.
RES 2,(FLAGS) Signal 'print in K-mode':
GP +CB Jump forward if dealing
JR Z,196C,OUT-CH-3 with the token 'THEN'.
CP +3A Jump forward unless dealing
JR Z,195A,OUT-CH-1 with ':'
BIT 5,(FLAGS) Jump forward to print the
JR Z,196C,OUT-CH-2 ':' if in 'INPUT mode'.
BIT 2,(FLAGS2) Jump forward if the ':'
JR Z,196C,OUT-CH-3 is 'not in quotes', i.e.
JR 1968,OUT-CH-2 an inter-statement marker.
JR 1968,OUT-CH-2 The ':' is inside quotes and can now be printed.
Accept for printing all characters except "".
Save the character code whilst changing the 'quote mode'.
Fetch FLAGS2 and flip bit 2.
Enter the amended value and restore the character code.
Signal 'the next character is to be printed in L-mode'.
The present character is printed before returning.

Note: It is the consequence of the tests on the present character that determines whether the next character is to be "printed in 'K' or 'L' mode".
Also note how the program does not cater for ':' in REM statements.

THE 'LINE-ADDR' SUBROUTINE
For a given line number, in the HL register pair, this subroutine returns the starting address of that line or the 'first line after', in the HL register pair, and the start of the previous line in the DE register pair.
If the line number is being used the zero flag will be set. However if the 'first line after' is substituted then the zero flag is returned reset.

Save the given line number.
Fetch the system variable PROG and transfer the address to the DE register pair.
The given line number.
Compare the given line number against the addressed line number. Return if carry reset; otherwise address the next line's number.
Switch the pointers and jump back to consider the next line of the program.
The given line number.
Fetch the high byte of the addressed line number and compare it. Return if they do not match.
Next compare the low bytes.
Return with the carry flag set if the addressed line number has yet to reach the given line number.

THE 'COMPARE LINE NUMBERS' SUBROUTINE
The given line number in the BC register pair is matched against the addressed line number.

Fetch the high byte of the addressed line number and compare it. Return if they do not match.
Next compare the low bytes.
Return with the carry flag set if the addressed line number has yet to reach the given line number.

THE 'FIND EACH STATEMENT' SUBROUTINE
This subroutine has two distinct functions.

I. It can be used to find the 'D'th. statement in a BASIC line - returning with the HL register pair addressing the location before the start of the statement and the zero flag set.

II. Also the subroutine can be used to find a statement, if any, that starts with a given token code (in the E register).
Enter a loop to handle each statement in the BASIC line.

```
1990 EACH-S-1  DEC  D    Decrease 'D' and return if
    RET  Z    the required statement has
                   been found.
    RST  0020,NEXT-CHAR    Fetch the next character code
    CP  E    and jump if it does not match
    JR  NZ,199A,EACH-S-3    the given token code.
    AND  A    But should it match then
    RET

Now enter another loop to consider the individual characters in the line to find where the statement ends.

```

```
1998 EACH-S-2  INC  HL    Update the pointer and fetch
    LD  A,(HL)    the new code.
199A EACH-S-3  CALL  18B6,NUMBER    Step over any number.
    LD  (CH-ADD),HL    Update CH-ADD.
    CP  +22    Jump forward if the character
                is not a "."
    JR  NZ,19A5,EACH-S-3    Otherwise set the 'quotes flag'.
    DEC  C    Jump forward if the character is a "":
19A5 EACH-S-4  CP  +3A    Jump forward unless the code
    JR  Z,19AD,EACH-S-5    is the token "THEN".
    CP  +CB    Jump forward unless the code
    JR  NZ,19B1,EACH-S-6    is the token "THEN".
19AD EACH-S-5  BIT  0,C    Read the 'quotes flag' and
    JR  Z,1990,EACH-S-1    jump back at the end of each
                          statement (including after
                          "THEN").
19B1 EACH-S-6  CP  +0D    Jump back unless at the end
    DEC  D    Decrease the statement
    SCF    counter and set the carry
    RET    flag before returning.
```

THE 'NEXT-ONE' SUBROUTINE
This subroutine can be used to find the 'next line' in the program area or the 'next variable' in the variables area. The subroutine caters for the six different types of variable that are used in the SPECTRUM system.

```
19B8 NEXT-ONE  PUSH  HL    Save the address of the
    LD  A,(HL)    current line or variable.
    CP  +40    Fetch the first byte.
    JR  C,19D5,NEXT-O-3    Jump forward if searching
                          for a 'next line'.
    BIT  5,A    Jump forward if searching for
                the next string or array variable.
    JR  Z,19D6,NEXT-O-4    Jump forward with simple
                numeric and FOR-NEXT
    ADD  A,A    variables.
    JP  M,19C7,NEXT-O-1    Long name numeric variables
    CCF    only.
19C7 NEXT-O-1  LD  BC,+0005    A numeric variable will
    JR  NC,19CE,NEXT-O-2    occupy five locations but a
    LD  C,+12    FOR-NEXT control variable
19CE NEXT-O-2  RLA    will need eighteen locations.
```

The carry flag becomes reset for long named variables only; until the final character of the
Increment the pointer and fetch the new code.

Jump back unless the previous code was the last code of the variable's name.

Now jump forward (BC = +0005 or +0012).

Step past the low byte of the line number.

Now point to the low byte of the length.

Fetch the length into the BC register pair.

Allow for the inclusive byte.

In all cases the address of the 'next' line or variable is found.

Add the first byte of the 'next' line or variable.

Fetch the address of the previous one and continue into the 'difference' subroutine.

Prepare for a true subtraction.

Find the length from one 'start' to the next and pass it to the BC register pair.

Reform the address and exchange them before returning.

The 'length' between two 'starts' is formed in the BC register pair. The pointers are reformed but returned exchanged.

Use the 'difference' subroutine to develop the appropriate values.

Save the number of bytes to be reclaimed.

All the system variable pointers above the area have to be reduced by 'BC' so this number is 2's complemented before the pointers are altered.

Return the 'first location' address to the DE register pair and reform the address of the first location to the left.

Save the 'first location' whilst the actual reclamation occurs.

Now return.
THE 'E-LINE-NO' SUBROUTINE
This subroutine is used to read the line number of the line in the editing area. If there is no line number, i.e. a direct BASIC line, then the line number is considered to be zero. In all cases the line number is returned in the BC register pair.

19FB E-LINE-NO LD HL,(E-LINE) Pick up the pointer to the edit-line.
DEC HL Set the CH-ADD to point to the location before any number.
RST 0020,NEXT-CHAR Pass the first code to the A register.
LD HL,+5C92 However before considering the code make the calculator's memory area a temporary calculator stack area.
LD (STKEND),HL CALL 2D3B,INT-TO-FP Now read the digits of the line number. Return zero if no number exists.
CALL 2DA2,FP-TO-BC Compress the line number into the BC register pair.
JR C,1A15,E-L-1 Jump forward if the number exceeds '65,536'.
LD HL,+D8F0 Otherwise test it against '10,000'.
ADD HL,BC JP 1C8A,REPORT-C Give report C if over '9,999'.
JR 16C5,SET-STK Return via SET-STK that restores the calculator stack to its rightful place.

THE 'REPORT AND LINE NUMBER PRINTING' SUBROUTINE
The entry point OUT-NUM-1 will lead to the number in the BC register pair being printed. Any value over '9,999' will not however be printed correctly.
The entry point OUT-NUM-2 will lead to the number indirectly addressed by the HL register pair being printed. This time any necessary leading spaces will appear. Again the limit of correctly printed numbers is '9,999'.

1A1B OUT-NUM-1 PUSH DE Save the other registers throughout the subroutine.
PUSH HL
XOR A Clear the A register.
BIT 7,B Jump forward to print a zero rather than '2' when reporting on the edit-line.
JR NZ,1A42,OUT-NUM-4
LD H,B Move the number to the HL register pair.
LD L,C
LD E,+FF Flag 'no leading spaces'.
JR 1A30,OUT-NUM-3 Jump forward to print the number.
1A28 OUT-NUM-2 PUSH DE Save the DE register pair.
LD D,(HL) Fetch the number into the DE register pair and save the pointer (updated).
INC HL
LD E,(HL)
PUSH HL
EX DE,HL Move the number to the HL register pair and flag 'leading space are to be printed'.
LD E,+20

Now the integer form of the number in the HL register pair is printed.

1A30 OUT-NUM-3 LD BC,+FC18 This is '-1,000'.
CALL 192A,OUT-SP-NO Print a first digit.
LD BC,+FF9C This is '-100'.
CALL 192A,OUT-SP-NO Print the second digit.
LD C,+F6 This is '-10'.
CALL 192A,OUT-SP-NO Print the third digit.
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A42</td>
<td>LD A,L</td>
<td>Move any remaining part of the number to the A register.</td>
</tr>
<tr>
<td></td>
<td>OUT-NUM-4</td>
<td>Print the digit.</td>
</tr>
<tr>
<td></td>
<td>CALL 15EF,OUT-CODE</td>
<td>Restore the registers before returning.</td>
</tr>
<tr>
<td></td>
<td>POP HL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>POP DE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RET</td>
<td></td>
</tr>
</tbody>
</table>
### BASIC LINE AND COMMAND INTERPRETATION

#### THE SYNTAX TABLES

**i. The offset table**
There is an offset value for each of the fifty BASIC commands.

<table>
<thead>
<tr>
<th>command</th>
<th>address</th>
<th>command</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFN</td>
<td>1AF9</td>
<td>DEFB</td>
<td>+94</td>
</tr>
<tr>
<td>CAT</td>
<td>1A62</td>
<td>CONTINUE</td>
<td>1AB8</td>
</tr>
<tr>
<td>FORMAT</td>
<td>1A63</td>
<td>+3F</td>
<td>DIM</td>
</tr>
<tr>
<td>MOVE</td>
<td>1A64</td>
<td>+41</td>
<td>REM</td>
</tr>
<tr>
<td>ERASE</td>
<td>1A65</td>
<td>+2B</td>
<td>FOR</td>
</tr>
<tr>
<td>OPEN #</td>
<td>1A66</td>
<td>+17</td>
<td>GO TO</td>
</tr>
<tr>
<td>CLOSE #</td>
<td>1A67</td>
<td>+1F</td>
<td>GO SUB</td>
</tr>
<tr>
<td>MERGE</td>
<td>1A68</td>
<td>+37</td>
<td>INPUT</td>
</tr>
<tr>
<td>VERIFY</td>
<td>1A69</td>
<td>+77</td>
<td>LOAD</td>
</tr>
<tr>
<td>BEEP</td>
<td>1A6A</td>
<td>+44</td>
<td>LIST</td>
</tr>
<tr>
<td>CIRCLE</td>
<td>1A6B</td>
<td>+0F</td>
<td>LET</td>
</tr>
<tr>
<td>OPEN</td>
<td>1A6C</td>
<td>+59</td>
<td>PAUSE</td>
</tr>
<tr>
<td>FORMAT</td>
<td>1A6D</td>
<td>+2B</td>
<td>NEXT</td>
</tr>
<tr>
<td>OPEN</td>
<td>1A6E</td>
<td>+43</td>
<td>POKE</td>
</tr>
<tr>
<td>BRIGHT</td>
<td>1A6F</td>
<td>+2D</td>
<td>PRINT</td>
</tr>
<tr>
<td>INVERSE</td>
<td>1A70</td>
<td>+51</td>
<td>PLOT</td>
</tr>
<tr>
<td>OVER</td>
<td>1A71</td>
<td>+3A</td>
<td>RUN</td>
</tr>
<tr>
<td>OUT</td>
<td>1A72</td>
<td>+6D</td>
<td>SAVE</td>
</tr>
<tr>
<td>LPRINT</td>
<td>1A73</td>
<td>+42</td>
<td>RANDOMIZE</td>
</tr>
<tr>
<td>BEEP</td>
<td>1A74</td>
<td>+0D</td>
<td>IF</td>
</tr>
<tr>
<td>CIRCLE</td>
<td>1A75</td>
<td>+49</td>
<td>CLS</td>
</tr>
<tr>
<td>INVERSE</td>
<td>1A76</td>
<td>+5C</td>
<td>DRAW</td>
</tr>
<tr>
<td>OVER</td>
<td>1A77</td>
<td>+44</td>
<td>CLEAR</td>
</tr>
<tr>
<td>OUT</td>
<td>1A78</td>
<td>+15</td>
<td>RETURN</td>
</tr>
<tr>
<td>NEW</td>
<td>1A79</td>
<td>+5D</td>
<td>COPY</td>
</tr>
<tr>
<td>NEW</td>
<td>1A80</td>
<td>+6D</td>
<td>AD6</td>
</tr>
</tbody>
</table>

**ii. The parameter table**
For each of the fifty BASIC commands there are up to eight entries in the parameter table. These entries comprise command class details, required separators and, where appropriate, command routine addresses.

<table>
<thead>
<tr>
<th>command</th>
<th>address</th>
<th>command</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>-LET</td>
<td>01</td>
<td>CLASS-01</td>
<td></td>
</tr>
<tr>
<td>-3D</td>
<td>'w'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-02</td>
<td>CLASS-02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-06</td>
<td>CLASS-06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-00</td>
<td>CLASS-00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+67,+1E</td>
<td>GO-TO,1E67</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-06</td>
<td>CLASS-06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+CB</td>
<td></td>
<td>THEN</td>
<td></td>
</tr>
<tr>
<td>+05</td>
<td>CLASS-05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+F0,+1C</td>
<td>IF,1CF0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-06</td>
<td>CLASS-06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-00</td>
<td>CLASS-00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+0E,+1E</td>
<td>GO-SUB,1EED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-00</td>
<td>CLASS-00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+EE,+1C</td>
<td>STOP,1CEE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-00</td>
<td>CLASS-00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+23,+1F</td>
<td>RETURN,1F23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-04</td>
<td>CLASS-04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+3D</td>
<td>'w'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+06</td>
<td>CLASS-06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+CC</td>
<td>'TO'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+06</td>
<td>CLASS-06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+05</td>
<td>CLASS-05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+03,+1D</td>
<td>FOR,1D03</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Note: 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 routines.

THE 'MAIN PARSER' OF THE BASIC INTERPRETER

The parsing routine of the BASIC interpreter is entered at LINE-SAN when syntax is being checked, and at LINE-RUN when a BASIC program of one or more statements is to be executed.

Each statement is considered in turn and the system variable CH-ADD is used to point to each code of the statement as it occurs in the program area or the editing area.
THE STATEMENT LOOP.
Each statement is considered in turn until the end of the line is reached.

1B28 STMT-LOOP RST 0020,NEXT-CHAR Advance CH-ADD along the line.
1B29 STMT-L-1 CALL 16BF,SET-WORK The work space is cleared.
                INC (SUBPPC) Increase SUBPPC on each
                passage around the loop.
                JP M,1C8A,REPORT-C But only "127" statements are
                allowed in a single line.
                RST 0018,GET-CHAR Fetch a character.
                LD B,+00 Clear the register for later.
                CP +0D Is the character a 'carriage
                return'; jump if it is.
                JR Z,1BB3,LIN-END Go around the loop again if
                it is a ':

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

LD HL,+1B76 Pre-load the machine stack
PUSH HL with the return address
- STMT-RET.
LD C,A Save the command temporarily
RST 0020,NEXT-CHAR in the C register whilst
LD A,C CH-ADD is advanced again.
SUB +CE Reduce the command's code by
        +CE; giving the range +00 to
        +31 for the fifty commands.
                JP C,1C8A,REPORT-C Give the appropriate error if
                not a command code.
                LD C,A Move the command code to the
                BC register pair (B holds +00).
                LD HL,+1A48 The base address of the syntax
                offset table.
                ADD HL,BC The required offset is passed to
                the C register and used to
                compute the base address for
                the command's entries in the
                parameter table.
                JR 1B55,GET-PARAM 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.

1B52 SCAN-LOOP LD HL,(T-ADDR) The temporary pointer to the
                entries in the parameter table.
1B55 GET-PARAM LD A,(HL) Fetch each entry in turn.
                INC HL Update the pointer to the
                entries for the next pass.
                LD (T-ADDR),HL Pre-load the machine stack
                with the return address -
                SCAN-LOOP.
                LD C,A Copy the entry to the C register
                for later.
                CP +20 Jump forward if the entry is
                a 'separator'.
                JR NC,1B6F,SEPARATOR The base address of the
                'command class' table.
                LD HL,+1C01 Clear the B register and
                index into the table.
                LD B,+00 Fetch the offset and compute
                the starting address of the
                required command class routine
                PUSH HL Push the address on to the
                machine stack.
RST 0018,GET-CHAR
DEC B
RET

Before making an indirect
jump to the command class
to the A register and set the B
register to +FF.

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'.

1B6F SEPARATOR RST 0018,GET-CHAR
CP C
JP NZ,1C8A,REPORT-C
RST 0020,NEXT-CHAR

The current character is
fetch and compared to the
entry in the parameter table.
Give the error report if there
is not a match.
Step past a correct character
and return.

THE 'STMT-RET' SUBROUTINE
After the correct interpretation of a statement a return is made to this entry point.

1B76 STMT-RET CALL 1F54,BREAK-KEY
JR C,1B7D,STMT-R

The BREAK key is tested after
every statement.
Jump forward unless it has
been pressed.

Report L - 'BREAK into program'

1B7B REPORT-L RST 0008,ERROR-1
DEFB +14

Call the error handling
routine.

Continue here as the BREAK key was not pressed.

1B7D STMT-R-1 BIT 7,(NSPPC)
JR NZ,1BF4,STMT-NEW
LD HL,(NEWPPC)
BIT 7,H
JR Z,1B9E,LINDEXE-W

Jump forward if there is not
a 'jump' to be made.
Fetch the 'new line' number
and 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).

1B8A LINE-RUN LD HL,FFFE
LD (PPC),HL
LD HL,(WORKSP)
DEC HL
LD DE,(E-LINE)
DEC DE
LD A,(NSPPC)
JR 1BD1,NEXT-LINE

A line in the editing area
is considered as line '-2'.
Make HL point to the end
marker of the editing area
and DE to the location before
the start of that area.
Fetch the number of the next
statement to be handled
before jumping 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.

1B9E LINE-NEW CALL 196,E-LINDEXE-ADDR
LD A,(NSPPC)
JR Z,1BBF,LINDEXE-USE
AND A

The starting address of the line,
or the 'first line after' is found.
Collect the statement number.
Jump forward if the required
line was found; otherwise
JR NZ,1BEC,REPORT-N    check the validity of the state-
ment number - must be zero.
LD B,A                       Also check that the 'first
LD A,(HL)                  line after' is not after the
AND +C0                     actual 'end of program'.
LD A,B                        Jump forward with valid
JR Z,1BBF,LINE-USE        addresses; otherwise signal the
                          error 'OK'.

Report 0 - 'OK'

1BB0 REPORT-0 RST 0008,ERROR-1    Use the error handling
DEFB +FF                       routine.

Note: Obviously not an error in the normal sense — but rather a jump past the program.

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

1BB2 REM POP BC    Drop the address - STMT-RET.

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.

1BB3 LINE-END CALL 2530,SYNTAX-Z    Return if syntax is being
RET Z                       checked; otherwise fetch
LD HL,(NXTLIN)               the address in NXTLIN.
LD A,+C0                     Return also if the address is
RET NZ                       after the end of the program
XOR A                        - the 'run' is finished.

THE 'LINE-USE' ROUTINE
This short routine has three functions; i. Change statement zero to statement '1'; ii. Find the number of the new line and enter it into
PPC; & iii. Form the address of the start of the line after.

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

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.

1BD1 NEXT-LINE LD (NXTLIN),HL     Set NXTLIN for use once the
                             current line has been completed.
                             As usual CH-ADD points to the
LD (CH-ADD),HL  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 EACH-STMT is used.
LD (NSPPC),+FF  Signal 'no jump'.
DEC D  The statement number minus one goes into SUBPPC.
LD (SUBPPC),D
JP Z,1B28,STMT-LOOP  A first statement can now be considered.
INC D  However for later statements each-STMT is used.
CALL 198B,EACH-STMT  the 'starting address' has to be found.
JR Z,1BF4,STMT-NEXT  Jump forward unless the state-

Report N: 'Statement lost'

1BEC REPORT-N RST 0008,ERROR-1  Call the error handling routine.
DEFB +16

THE 'CHECK-END' SUBROUTINE
This is an important routine and is called from many places in the monitor program 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.

1BEE CHECK-END CALL 2530,SYNTAX-Z  Do not proceed unless checking syntax.
RET NZ
POP BC  Drop the addresses of SCAN-LOOP & STMT-RET
POP BC  before continuing into STMT-NEXT.

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.

1BF4 STMT-NEXT RST 0018,GET-CHAR  Fetch the present character.
CP +0D  Consider the 'next line' if it is a 'carriage return'.
JR Z,1BB3,LIN-END  Otherwise there has been a syntax error.
CP +3A  Consider the 'next statement'
JP Z,1B28,STMT-LOOP  if it is a ':'.
JP 1C8A,REPORT-C

THE 'COMMAND CLASS' TABLE

<table>
<thead>
<tr>
<th>address</th>
<th>offset</th>
<th>class number</th>
<th>address</th>
<th>offset</th>
<th>class number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1C01</td>
<td>0F</td>
<td>CLASS-00-1C10</td>
<td>1C07</td>
<td>7B</td>
<td>CLASS-06,1C82</td>
</tr>
<tr>
<td>1C02</td>
<td>1D</td>
<td>CLASS-01,1C1F</td>
<td>1C08</td>
<td>8E</td>
<td>CLASS-07,1C96</td>
</tr>
<tr>
<td>1C03</td>
<td>4B</td>
<td>CLASS-02,1C4E</td>
<td>1C09</td>
<td>71</td>
<td>CLASS-08,1C7A</td>
</tr>
<tr>
<td>1C04</td>
<td>09</td>
<td>CLASS-03,1C0D</td>
<td>1C0A</td>
<td>B4</td>
<td>CLASS-09,1CBE</td>
</tr>
<tr>
<td>1C05</td>
<td>67</td>
<td>CLASS-04,1C6C</td>
<td>1C0B</td>
<td>81</td>
<td>CLASS-0A,1C8C</td>
</tr>
<tr>
<td>1C06</td>
<td>0B</td>
<td>CLASS-05,1C11</td>
<td>1C0C</td>
<td>CF</td>
<td>CLASS-0B,1CDB</td>
</tr>
</tbody>
</table>

THE 'COMMAND CLASSES - 00, 03 & 05'
The commands of class-03 may, or may not, be followed by a number. e.g. RUN & RUN 200.

1C0D CLASS-03 CALL 1CDE,FETCH-NUM  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.

1C10 CLASS-00 CP A Set the zero flag for later.

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

1C11 CLASS-05 POP BC In all cases drop the address
CALL Z,1BEE,CHECK-END if handling commands of classes
EX DE,HL 00 & 03 AND syntax is being
considered the next statement.

Save the line pointer in the DE
register pair.

THE 'JUMP-C-R' ROUTINE
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.

1C16 JUMP-C-R LD HL,(T ADDR) Fetch the pointer to the
LD C,(HL) 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 and make an indirect jump
RET to the command routine.

THE 'COMMAND CLASSES - 01, 02 & 04'
These three command classes are used by the variable handling commands - LET, FOR & NEXT and indirectly by READ & INPUT. Command class 01 is concerned with the identification of the variable in a LET, READ or INPUT statement.

1C1F CLASS-01 CALL 28B2,LOOK-VARS Look in the variables area to
look whether or not the variable has been used already.

THE 'VARIABLE IN ASSIGNMENT' SUBROUTINE
This subroutine develops the appropriate values for the system variables DEST & STRLEN.

1C22 VAR-A-1 LD (FLAGX),+00 Initialise FLAGX to +00.
JR NC,1C30,VAR-A-2 Jump forward if the variable
SET 1,(FLAGX) has been used before.
JR NZ,1C46,VAR-A-3 Signal 'a new variable'.
Give an error if trying to use
an 'undimensioned array'.

Report 2 - Variable not found

1C2E REPORT-2 RST 0008,ERROR-1 Call the error handling
DEFB +01 routine.

Continue with the handling of existing variables.

1C30 VAR-A-2 CALL Z,2996,STK-VARS The parameters of simple string
variables and all array variables
are passed to the calculator
stack. (STK-VARS will 'slice' a
string if required.)

BIT 6,(FLAGS) Jump forward if handling a
JR NZ,1C46,VAR-A-3 numeric variable.
XOR A Clear the A register.
CALL 2530,SYNTAX-Z The parameters of the string of
CALL NZ,2BF1,STK-FETCH string array variable are fetched
unless syntax is being checked.
LD HL,+5C71 This is FLAGX.
The pathways now come together to set STRLEN & DEST as required. For all numeric variables and 'new' string & string array variables STRLEN-lo holds the 'letter' of the variable's name. But for 'old' string & string array variables whether 'sliced' or complete it holds the 'length' in 'assignment'.

1C46  VAR-A-3  LD  (STRLEN),BC  Set STRLEN as required.

DEST holds the address for the 'destination of an 'old' variable but in effect the 'source' for a 'new' variable.

Command class 02 is concerned with the actual calculation of the value to be assigned in a LET statement.

1C4E  CLASS-02  POP  BC  The address - SCAN-LOOP is dropped.
   CALL  1C56,VAL-FET-1  The assignment is made.
   CALL  1BEE,CHECK-END  Move on to the next statement
   RET  return.

THE 'FETCH A VALUE' SUBROUTINE
This subroutine is used by LET, READ & INPUT statements to first evaluate and then assign values to the previously designated variable.
   The entry point VAL-FET-1 is used by LET & READ and considers FLAGS whereas the entry point VAL-FET-2 is used by INPUT and considers FLAGX.

1C56  VAL-FET-1  LD  A,(FLAGS)  Use FLAGS.
1C59  VAL-FET-2  PUSH  AF  Save FLAGS or FLAGX.
   CALL  24FB,SCANNING  Evaluate the next expression.
   POP  AF  Fetch the old FLAGS or FLAGX.
   LD  D,(FLAGS)  Fetch the new FLAGS.
   XOR  D  The nature - numeric or string
   AND  +40  of the variable and the expression must match.
   JR  NZ,1C8A,REPORT-C  Give report C if they do not.
   BIT  7,D  Jump forward to make the
   JP  NZ,2AFF,LET  actual assignment unless
   RET  checking syntax when simply

THE 'COMMAND CLASS 04' ROUTINE
The command class 04 entry point is used by FOR & NEXT statements.

1C6C  CLASS-04  CALL  2BB2,LOOK-VARS  Look in the variables area for
   PUSH  AF  the variable being used.
   LD  A,C  Save the AF register pair whilst
   OR  +9F  to ensure that the variable
   INC  A  is a FOR-NEXT control
   JR  NZ,1C8A,REPORT-C  variable.
   POP  AF  Restore the flags register and
   JR  1C22,VAR-A-1  jump back to make the variable
   that has been found the 'variable in assignment'.

THE 'EXPECT NUMERIC/STRING EXPRESSIONS' SUBROUTINE
There is a series of short subroutines that are used to fetch the result of evaluating the next expression. The result from a single expression is returned as a 'last value' on the calculator stack.

The entry point NEXT-2NUM is used when CH-ADD needs updating to point to the start of the first expression.

1C79  NEXT-2NUM  RST  0020,NEXT-CHAR  Advance CH-ADD.

The entry point EXPT-2NUM (EQU. CLASS-08) allows for two numeric expressions, separated by a comma, to be evaluated.

1C7A  EXPT-2NUM (CLASS-08)  CALL  1C82,EXPT-1NUM  Evaluate each expression in turn - so evaluate the first.
CP    +2C  Give an error report if the separator is not a comma.
JR    NZ,1C8A
RST   0020,NEXT-CHAR  Advance CH-ADD.

The entry point EXPT-1NUM (EQU. CLASS-06) allows for a single numeric expression to be evaluated.

1C82  EXPT-1NUM (CLASS-06)  CALL  24FB,SCANNING  Evaluate the next expression.
BIT   6,(FLAGS)  Return as long as the result was numeric; otherwise it is an error.
RET   NZ

Report C - Nonsense in BASIC

1C8A  REPORT-C  RST  0008,ERROR-1  Call the error handling routine.
DEBF  +0B

The entry point EXPT-EXP (EQU. CLASS-0A) allows for a single string expression to be evaluated.

1C8C  EXPT-EXP (CLASS-0A)  CALL  24FB,SCANNING  Evaluate the next expression.
BIT   6,(FLAGS)  This time return if the result indicates a string; otherwise give an error report.
RET   Z
JR    1C8A,REPORT-C

THE 'SET PERMANENT COLOURS' SUBROUTINE (EQU. CLASS-07)
This subroutine allows for the current temporary colours to be made permanent. As command class 07 it is in effect the command routine for the six colour item commands.

1C96  PERMS (CLASS-07)  BIT  7,(FLAGS)  The syntax/run flag is read.
RES   0,(TV-FLAG)  Signal 'main screen'.
CALL  NZ,0D4D,TEMPS  Only during a 'run' call TEMPS to ensure the temporary colours are the main screen colours.
POP   AF  Drop the return address - SCAN-LOOP.
LD    A,(T-ADDR)  Fetch the low byte of T-ADDR and subtract +13 to give the range +D9 to +DE which are the token codes for INK to OVER.
SUB   +13
CALL  21FC,CO-TEMP-4  Jump forward to change the temporary colours as directed by the BASIC statement.
CALL  1BEE,CHECK-END  Move on to the next statement if checking syntax.
LD    HL,(ATTR-T)  Now the temporary colour values are made permanent (both ATTR-P & MASK-P). This is P-FLAG; and that too has to be considered.
LD    (ATTR-P),HL
LD    HL,+5C91
LD    A,(HL)
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
XOR (HL)
AND +AA
XOR (HL)
LD (HL),A
RET
```

**THE 'COMMAND CLASS 09' ROUTINE**

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.

```
1CBE CLASS-09 CALL 2530,SYNTAX-Z Jump forward if checking syntax.
JR Z,1CD6,CL-09-1 SIGNAL 'main screen'.
RES 0,(TV-FLAG) Set the temporary colours for the main screen.
CALL 0D4D,TEMPS Signal 'main screen'.
LD HL,+5C90 This is MASK-T.
LD A,(HL) Fetch its present value but keep only its INK part 'unmasked'.
OR +F8
LD (HL),A Restore the value which now indicates 'FLASH 8; BRIGHT 8; PAPER 8;'.
RES 6,(P-FLAG) Also ensure NOT 'PAPER 9'.
RST 0018,GET-CHAR Fetch the present character before continuing to deal with embedded colour items.
1CD6 CL-09-1 CALL 21E2,CO-TEMP Deal with the locally dominant colour items.
JR 1C7A,EXPT-2NUM Now get the first two operands for PLOT, DRAW or CIRCLE.
```

**THE 'COMMAND CLASS 0B' ROUTINE**

This routine is used by SAVE, LOAD, VERIFY & MERGE statements.

```
1CDB CLASS-0B JP 0605,SAVE-ETC Jump to the cassette handling routine.
```

**THE 'FETCH A NUMBER' SUBROUTINE**

This subroutine leads to a following numeric expression being evaluated but zero being used instead if there is no expression.

```
1CDE FETCH-NUM CP +0D Jump forward if at the end of a line.
JR Z,1CE6,USE-ZERO But jump to EXPT-1NUM unless at the end of a statement.
CP +3A
JR NZ,1C82,EXPT-1NUM
```

The calculator is now used to add the value zero to the calculator stack.

```
1CE6 USE-ZERO CALL 2530,SYNTAX-Z Do not perform the operation if syntax is being checked.
RET Z Use the calculator.
RST 0028,FP-CALC The 'last value' is now zero.
DEFB +A0,stk-zero DEF B +98,end-calc Return with zero added to the stack.
RET
```

**THE COMMAND ROUTINES**

The section of the 16K monitor program from 1CEE to 23FA contains most of the command routines of the BASIC interpreter.
THE 'STOP' COMMAND ROUTINE
The command routine for STOP contains only a call to the error handling routine.

1CEE STOP RST 0008,ERROR-1 DEFB +08 
(REPORT-9) Call the error handling routine.

THE 'IF' COMMAND 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.

1CF0 IF POP BC CALL 2530,SYNTAX-Z CALL Z,1D00,IF-1 JR Z,1D00,IF-1 Jump forward if checking syntax.
Now use the calculator to 'delete' the last value on the calculator stack but leave the DE register pair addressing the first byte of the value.

RST 0028,FP-CALC Use the calculator.
DEFB +02,delete The present 'last value' is deleted.
EX DE,HL Make HL point to the first byte and call TEST-ZERO.
CALL 34E9.TEST-ZERO If the value was 'FALSE' jump to the next line.
JP C,1BB3.LINE-END JR 1B29,STMT-L-1 But if 'TRUE' jump to the next statement (after the THEN).

THE 'FOR' COMMAND ROUTINE
This command routine is entered with the VALUE and the LIMIT of the FOR statement already on the top of the calculator stack.

1D03 FOR CP JR RST CALL JR CALL CALL JR CP +CD NZ,1D10,F-USE-1 0020,NEXT-CHAR 1C82,EXPT-1NUM 1BEE,CHECK-END 1D16,F-REORDER +CD +02,delete v, l, s +38,end-calc v, l, s (mem-0 = s) DEFB DEFB DEFB DEFB If checking syntax; otherwise jump forward.

There has not been a STEP supplied so the value '1' is to be used.

1D10 F-USE-1 CALL RST DEFB DEFB 0028,FP-CALC +01,exchange +01,exchange,STK-1 +A1,stk-one +38,end-calc Move on to the next statement on the calculator stack. Move on to the next statement if checking syntax; otherwise use the calculator to place a '1'

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.

1D16 F-REORDER RST DEFB DEFB DEFB DEFB 0028,FP-CALC +C0,ST-MEM-0 +02,delete +01,exchange +38,end-calc v, l, s v, l l, v l, v, s l, s, v A FOR control variable is now established and treated as a temporary calculator memory area.

CALL 2AFF,LET The variable is found, or created if needed (v is used).
LD (MEM),HL Make it a 'memory area'.

The variable that has been found may be a simple numeric variable using only six locations in which case it will need extending.
DEC HL Fetch the variable's single character name.
LD A,(HL) It will have six locations at least.
SET 7,(HL) Make HL point after them.
ADD HL,BC
RLCA Rotate the name and jump if it was already a FOR variable.
JR C,1D34,F-L&S
LD C,+0D Otherwise create thirteen more locations.
CALL 1655,MAKE-ROOM
INC HL Again make HL point to the LIMIT position.

The initial values for the LIMIT and the STEP are now added.

1D34 F-L&S PUSH HL The pointer is saved.
RST 0028,FP-CALC I, s
DEBF +02,delete -
DEBF +02,delete DE still points to 'T'.
POP HL The pointer is restored and both pointers exchanged.
LD C,+0A The ten bytes of the LIMIT and the STEP are moved.
LD (HL),E
LD (HL),D
LDIR

The looping line number and statement number are now entered.

LD HL,(PPC) The current line number.
EX DE,HL Exchange the registers before adding the line number to the FOR control variable.
INC HL
LD (HL),D
LD (HL),D The looping statement is always the next statement -
INC D whether it exists or not.
INC HL
LD (HL),D

The NEXT-LOOP subroutine is called to test the possibility of a 'pass' and a return is made if one is possible; otherwise the statement after for FOR - NEXT loop has to be identified.

CALL 1DDA,NEXT-LOOP Is a 'pass' possible?
RET NC Return now if it is.
LD B,(STRLEN-lo) Fetch the variable's name.
LD HL,(PPC) Copy the present line number to NEWPPC.
LD (NEWPPC),HL Fetch the current statement number and two's complement it.
NEG LD D,A Transfer the result to the D register.
LD HL,(CH-ADD) Fetch the current value of CH-ADD.
LD E,+F3 The search will be for 'NEXT'.

Now a search is made in the program area, from the present point onwards, for the first occurrence of NEXT followed by the correct variable.

1D64 F-LOOP PUSH BC Save the variable's name.
LD BC,(NXTLIN) Fetch the current value of NXTLIN.
CALL 1D86,LOOK-PROG The program area is now searched and BC will change with each new line examined.
LD (NXTLIN),BC Upon return save the pointer.
POP BC Restore the variable's name.
JR C,1D84,REPORT-I If there are no further NEXTs then give an error.
RST 0020,NEXT-CHAR
OR +20
CP B
JR Z,1D7C,F-FOUND
RST 0020,NEXT-CHAR
JR 1D64,F-LOOP

Advance past the NEXT that was found.
Allow for upper and lower case letters before the new variable name is tested.
Jump forward if it matches.
Advance CH-ADD again and jump back if not the correct variable.

NEWPPC holds the line number of the line in which the correct NEXT was found. Now the statement number has to be found and stored in NSPPC.

1D7C F-FOUND  RST 0020,NEXT-CHAR  Advance CH-ADD.
LD A,+01
SUB D
LD (NSPPC),A
RET

The statement counter in the D register counted statements back from zero so it has to be subtracted from '1'.
The result is stored.
Now return - to STMT-RET.

REPORT I - FOR without NEXT
1D84 REPORT-I  RST 0008,ERROR-1  Call the error handling routine.
DEFB +11

THE 'LOOK-PROG' SUBROUTINE
This subroutine is used to find occurrences of either DATA, DEF FN or NEXT. On entry the appropriate token code is in the E register and the HL register pair points to the start of the search area.

1D86 LOOK-PROG  LD A,(HL)  Fetch the present character.
CP +3A
JR Z,1D84,LOOK-PROG  Jump forward if it is a ':'.

Now a loop is entered to examine each further line in the program.

1D8B LOOK-P-1  INC HL  Fetch the high byte of the line number and return with carry set if there are no further lines in the program.
LD A,(HL)
AND +CO
SCF
RET
NZ
LD B,(HL)
INC HL
LD C,(HL)
LD (NEWPPC),BC
INC HL
LD C,(HL)
PUSH HL
ADD HL,BC
LD B,H
LD C,L
POP HL
LD D,00

The pointer is saved whilst the address of the end of the line is formed in the BC register pair.
The pointer is restored.
Set the statement counter to zero.

1DA3 LOOK-P-2  PUSH BC  The end-of-line pointer is saved whilst the statements of the line are examined.
CALL 198B,EACH-STMT
POP BC
RET NC
JR 1D8B,LOOK-P-1

Make a return if there was an 'occurrence'; otherwise consider the next line.
THE 'NEXT' COMMAND ROUTINE

The 'variable in assignment' has already been determined (see CLASS-04.1C6C); and it remains to change the VALUE as required.

1DAB NEXT BIT 1,(FLAGX) Jump to give the error report
JP N2,1C2E,REPORT-2 if the variable was not found.
LD HL,(DEST) The address of the variable
BIT 7,(HL) is fetched and the name
JR Z,1DD8,REPORT-1 tested further.

Next the variable's VALUE and STEP are manipulated by the calculator.

INC HL Step past the name.
LD (MEM),HL Make the variable a temporary 'memory area'.
RST 0028,FP-CALC -
DEFB +E0,get-mem-0 v
DEFB +E2,get-mem-2 v, s
DEFB +0F,addition v+s
DEFB +C0,stm-mem-0 v+s
DEFB +02,delete -
DEFB +38,end-calc -

The result of adding the VALUE and the STEP is now tested against the LIMIT by calling NEXT-LOOP.

CALL 1DDA,NEXT-LOOP Test the new VALUE against the LIMIT
RET C Return now if the FOR-NEXT loop has been completed.

Otherwise collect the 'looping' line number and statement.

LD HL,(MEM) Find the address of the
LD DE,+000F low byte of the looping
ADD HL,DE line number.
LD E,(HL) Now fetch this line number.
INC HL
LD D,(HL)
INC HL
LD H,(HL) Followed by the statement number.
EX DE,HL Exchange the numbers before
JP 1E73,GO-TO-2 jumping forward to treat them as the destination line of a GO TO command.

Report 1 - NEXT without FOR

1DD8 REPORT-1 RST 0008,ERROR-1 Call the error handling DEF +00 routine.
DEFB

THE 'NEXT-LOOP SUBROUTINE

This subroutine is used to determine whether the LIMIT has been exceeded by the present VALUE. Note has to be taken of the sign of the STEP.

The subroutine returns the carry flag set if the LIMIT is exceeded.

1DDA NEXT-LOOP RST 0028,FP-CALC -
DEFB +E1,get-mem-1 l
DEFB +E0,get-mem-0 l, v
DEFB +E2,get-mem-2 l, v, s
DEFB +36,less-0 l, v,(1/0)
DEFB +00,jump-true l, v,(1/0)
DEFB +02,to NEXT-1 l, v,(1/0)
DEFB +01,exchange v, l
1DE2 NEXT-1 DEF +03,subtract v-l or l-v
DEFB +37,greater-0 (1/0)
DEFB +00,jump-true (1/0)
DEFB +04, to NEXT-2
DEFB +38, end-calc
AND A
RET
Clear the carry flag and return - loop is possible.

However if the loop is impossible the carry flag has to be set.

THE ‘READ’ COMMAND ROUTINE

The READ command allows for the reading of a DATA list and has an effect similar to a series of LET statements. Each assignment within a single READ statement is dealt with in turn. The system variable X-PTR is used as a storage location for the pointer to the READ statement whilst CH-ADD is used to step along the DATA list.

1DEC READ-3 RST 0020, NEXT-CHAR
Come here on each pass, after the first, to move along the READ statement.

1DED READ CALL 1C1F, CLASS-01
Consider whether the variable has been used before; find the existing entry if it has.

CALL 2530, SYNTAX-Z
Jump forward if checking syntax.
RST 0018, GET-CHAR
Save the current pointer.
LD (X-PTR), HL
CH-ADD in X-PTR.
LD HL, (DATAADD)
Fetch the current DATA list pointer and jump forward.
LD A, (HL)
CP +2C
unless a new DATA statement has to be found.
JR Z, 1D0A, READ-1
The search is for ‘DATA’.
LD E, +E4
CALL 1D86, LOOK-PROG
Jump forward if the search is successful.
JR NC, 1E0A, READ-1

Report E - Out of DATA

1E08 REPORT-E RST 0008, ERROR-1
Call the error handling routine.
DEFB +0D
Continue - picking up a value from the DATA list.

1E0A READ-1 CALL 0077, TEMP-PTR1
Advance the pointer along the DATA list and set CH-ADD.
CALL 1C56, VAL-FET-1
Fetch the value and assign it to the variable.
RST 0018, GET-CHAR
Fetch the current value of DATA.
LD (DATAADD), HL
CH-ADD and store it in DATAADD.
LD HL, (X-PTR)
Fetch the pointer to the READ statement and clear X-PTR.
LD (X-PTR-hi)+00

1E1E READ-2 RST 0018, GET-CHAR
GET the present character and see if it is a ‘,’.
CP +2C
If it is then jump back as there are further items;
JR Z, 1DEC, READ-3
otherwise return either via CHECK-END (if checking syntax) or the RET instruction (to STMT-RET).
THE 'DATA' COMMAND 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.

A loop is now entered to deal with each expression in the DATA statement.

The DATA statement has to be passed-by in 'run-time'.

THE 'PASS-BY' SUBROUTINE

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'.

THE 'RESTORE' COMMAND ROUTINE

The operand for a RESTORE command is taken as a line number, zero being used if no operand is given.

The REST-RUN entry point is used by the RUN command routine.

THE 'RANDOMIZE' COMMAND ROUTINE

Once again the operand is compressed into the BC register pair and transferred to the required system variable. However if the operand is zero the value in FRAMES1 and FRAMES2 is used instead.
THE 'CONTINUE' COMMAND ROUTINE
The required line number and statement number within that line are made the object of a jump.

```
1E5F CONTINUE LD HL,(OLDPPC)  # The line number.
     LD D,(OSPPC)       # The statement number.
     JR 1E73,GO-TO-2   # Jump forward.
```

THE 'GO TO' COMMAND ROUTINE
The operand of a GO TO ought to be a line number in the range '1' to '9999' but the actual test is against an upper value of '61439'.

```
1E67 GO-TO CALL 1E99,FIND-INT2  # Fetch the operand and transfer
     LD H,B             # it to the HL register pair.
     LD L,C             # Set the statement number to zero.
     LD D,+00           # Give the error message
     JR +F0             # - Integer out of range -
     NC,1E9F,REPORT-B  # with lines over '614139'
```

The entry point GO-TO-2 is used to determine the line number of the next line to be handled in several instances.

```
1E73 GO-TO-2 LD (NEWPPC),HL # Enter the line number and
     LD (NSPPC),D      # then the statement number.
     RET Return; - to STMT-RET.
```

THE 'OUT' COMMAND ROUTINE
The two parameters for the OUT instruction are fetched from the calculator stack and used as directed.

```
1E7A OUT CALL 1E85,TWO-PARAM  # The operands are fetched.
     OUT (C),A          # The actual OUT instruction.
     RET Return; - to STMT-RET.
```

THE 'POKE' COMMAND ROUTINE
In a similar manner the POKE operation is performed.

```
1E80 POKE CALL 1E85,TWO-PARAM  # The operands are fetched.
     LD (BC),A          # The actual POKE operation.
     RET Return; - to STMT-RET.
```

THE 'TWO-PARAM' SUBROUTINE
The topmost parameter on the calculator stack must be compressible into a single register. It is two's complemented if it is negative.

```
1E85 TWO-PARAM CALL 2DD5,FP-TO-A  # The parameter is fetched.
     JR C,1E9F,REPORT-B  # Give an error if it is too high
     JR Z,1E8E,TWO-P-1   # a number.
     NEG                 # Jump forward with positive
     JR -Z,1E8E,TWO-P-1   # numbers but two's complement
     JR +Z,1E8E,TWO-P-1   # negative numbers.
```

```
1E8E TWO-P-1 PUSH AF      # Save the first parameter
     CALL 1E99,FIND-INT2  # whilst the second is fetched.
     POP AF               # The first parameter is
     RET                  # restored before returning.
```

THE 'FIND INTEGERS' SUBROUTINE
The 'last value' on the calculator stack is fetched and compressed into a single register or a register pair by entering at FIND-INT1 AND FIND-INT2 respectively.

```
1E94 FIND-INT1 CALL 2DD5,FP-TO-A  # Fetch the 'last value'.
     JR 1E9C,FIND-I-1    # Jump forward.
```

```
1E99 FIND-INT2 CALL 2DA2,FP-TO-BC # Fetch the 'last value'.
     JR C,1E9F,REPORT-B  # In both cases overflow is
```

101
THE 'RUN' COMMAND 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.

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E01</td>
<td>CALL 1E67,GO-TO</td>
<td>Set NEWPPC as required.</td>
</tr>
<tr>
<td>LD</td>
<td>BC,+0000</td>
<td>Now perform a 'RESTORE 0'.</td>
</tr>
<tr>
<td>CALL</td>
<td>1E45,REST-RUN</td>
<td>Exit via the CLEAR command routine.</td>
</tr>
<tr>
<td>JR</td>
<td>1EAF,CLEAR-1</td>
<td></td>
</tr>
</tbody>
</table>

THE 'CLEAR' COMMAND 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.

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1EDC</td>
<td>CALL 1E99,FIND-INT2</td>
<td>Fetch the operand - using zero by default.</td>
</tr>
<tr>
<td>LD</td>
<td>A,B</td>
<td>Jump forward if the operand is other than zero. When called from RUN there is no jump.</td>
</tr>
<tr>
<td>OR</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>JR</td>
<td>NZ,1EB7,CLEAR-1</td>
<td>If zero use the existing value in RAMTOP.</td>
</tr>
<tr>
<td>LD</td>
<td>BC,(RAMTOP)</td>
<td></td>
</tr>
<tr>
<td>PUSH</td>
<td>BC</td>
<td>Save the value.</td>
</tr>
<tr>
<td>LD</td>
<td>DE,(VARS)</td>
<td>Next reclaim all the bytes of the present variables area.</td>
</tr>
<tr>
<td>LD</td>
<td>HL,(E-LINE)</td>
<td></td>
</tr>
<tr>
<td>DEC</td>
<td>HL</td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>19E5,RECLAIM-1</td>
<td>Clear the display area.</td>
</tr>
<tr>
<td>CALL</td>
<td>0D68,CLS</td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>HL,(STKEN)</td>
</tr>
<tr>
<td>ADD</td>
<td>HL,DE</td>
</tr>
<tr>
<td>SBC</td>
<td>HL,DE</td>
</tr>
<tr>
<td>JR</td>
<td>NC,1EDA,REPORT-M</td>
</tr>
<tr>
<td>LD</td>
<td>HL,(P-RAMT)</td>
</tr>
<tr>
<td>AND</td>
<td>A</td>
</tr>
<tr>
<td>SBC</td>
<td>HL,DE</td>
</tr>
<tr>
<td>JR</td>
<td>NC,1EDC,CLEAR-2</td>
</tr>
</tbody>
</table>

Report M - RAMTOP no good

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1EDA</td>
<td>REPORT-M</td>
<td>Call the error handling routine.</td>
</tr>
<tr>
<td>RST</td>
<td>0008,ERROR-1</td>
<td></td>
</tr>
<tr>
<td>DEFB</td>
<td>+15</td>
<td></td>
</tr>
</tbody>
</table>

Continue with the CLEAR operation.

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1EDC</td>
<td>CLEAR-2</td>
<td>Now the value can actually be passed to RAMTOP.</td>
</tr>
<tr>
<td>EX</td>
<td>DE,HL</td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>(RAMTOP),HL</td>
<td>Fetch the address - STMT-RET.</td>
</tr>
<tr>
<td>POP</td>
<td>DE</td>
<td>Fetch the 'error address'.</td>
</tr>
<tr>
<td>POP</td>
<td>BC</td>
<td>Enter a GO SUB stack end marker.</td>
</tr>
<tr>
<td>LD</td>
<td>(HL),+3E</td>
<td></td>
</tr>
<tr>
<td>DEC</td>
<td>HL</td>
<td>Leave one location.</td>
</tr>
<tr>
<td>LD</td>
<td>SP,HL</td>
<td>Make the stack pointer point</td>
</tr>
</tbody>
</table>
PUSH BC  to an empty GO SUB stack.
LD (ERR-SP),SP   Next pass the 'error address'
EX DE,HL   to the stack and save its
JP (HL)   address in ERR-SP.

Note: When the routine is called from RUN the values of NEWPPC & NSPPC will have been affected and no statements coming after
RUN can ever be found before the jump is taken.

THE 'GO SUB' COMMAND ROUTINE
The present value of PPC and the incremented value of SUBPPC are stored on the GO SUB stack.
1EED GO-SUB
POP DE   Save the address - STMT-RET.
LD H,(SUBPPC) Fetch the statement number
INC H   and increment it.
EX (SP),HL Exchange the 'error address'
INC SP   with the statement number.
LD BC,(PPC) Next save the present line
PUSH BC number.
PUSH HL Return the 'error address'
PUSH HL Return the error address'
LD (ERR-SP),SP to the machine stack and
PUSH DE Reset ERR-SP to point to it.
PUSH DE Return the address -
CALL 1E67,GO-TO-1 Now set NEWPPC & NSPPC to
LD BC,+0014 the required values.

THE 'TEST-ROOM' SUBROUTINE
A series of tests is performed to ensure that there is sufficient free memory available for the task being undertaken.
1F05 TEST-ROOM
LD HL,(STKEND) Increase the value taken from
ADD HL,BC STKEND by the value carried
JR C,1F15,REPORT-4 into the routine by the BC
EX DE,HL Try it again allowing for a
LD HL,+0050 further eighty bytes.
ADD HL,DE
JR C,1F15,REPORT-4
SBC HL,SP Finally test the value against the
RET C address of the machine stack.
Report 4 - Out of memory
1F15 REPORT-4 LD L,+03 This is a 'run-time' error and the
JP 0055,ERROR-3 error marker is not to be used.

THE 'FREE MEMORY' SUBROUTINE
There is no BASIC command 'FRE' in the SPECTRUM but there is a subroutine for performing such a task.
An estimate of the amount of free space can be found at any time by using:
'PRINT 65536-USR 7962'
1F1A FREE-MEM LD BC,+0000 Do not allow any overhead.
CALL 1F05,TEST-ROOM Make the test and pass the
THE 'RETURN' COMMAND 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.

1F23  RETURN  POP  BC  Fetch the address - STMT-RET.
      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
      JR  Z,1F36,REPORT-7  marker; jump if it is.
      DEC  SP  The full entry uses three
      EX  (SP),HL  locations only.
      EX  DE,HL  Exchange the statement number
      LD  (ERR-SP),SP  with the 'error address'.
      PUSH  BC  Reset the error pointer.
      JP  1E73,GO-TO-2  Replace the address
            -       STMT-RET.
Report 7 - RETURN without GOSUB
1F36  REPORT-7  PUSH  DE  Replace the end marker and
      PUSH  HL  the 'error address'.
      RST  0008,ERROR-1  Call the error handling
      DEFB  +06  routine.

THE 'PAUSE' COMMAND ROUTINE
The period of the PAUSE is determined by counting the number of maskable interrupts as they occur every 1/50th of a second.
A PAUSE is finished either after the appropriate number of interrupts or by the system Variable FLAGS indicating that a key has been pressed.

1F33  PAUSE  CALL  1E99,FIND-INT2  Fetch the operand.
1F3D  PAUSE-1  HALT  1E99,FIND-INT2  Wait for a maskable interrupt.
      DEC  BC  Decrease the counter.
      LD  A,B  If the counter is thereby
      OR  C  reduced to zero the PAUSE
      JR  Z,1F4F,PAUSE-END  has come to an end.
      LD  A,B  If the operand was zero BC
      AND  C  will now hold +FFFF and this
      INC  A  value will be returned to
      JR  NZ,1F49,PAUSE-2  zero. Jump will all other
      INC  BC  operand values.
      JP  1E73,GO-TO-2  Jump back unless a key has
            -       been pressed.
The period of the PAUSE has now finished.
1F4F  PAUSE-END  RES  5,(FLAGS)  Signal 'no key pressed'.
      RET  Now return; - to STMT-RET.

THE 'BREAK-KEY' SUBROUTINE
This subroutine is called in several instances to read the BREAK key. The carry flag is returned reset only if the SHIFT and the BREAK keys are both being pressed.

1F54  BREAK-KEY  LD  A,+7F  Form the port address
      IN  A,(+FE)  +7FFE and read in a byte.
      RRA  Examine only bit 0 by shifting
            it into the carry position.
RET C Return if the BREAK key is not being pressed.
LD A,+FE Form the port address
IN A,(+FE) +FEFE and read in a byte.
RRA Again examine bit 0.
RET Return with carry reset if both keys are being pressed.

THE 'DEF FN' COMMAND 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.

1F60 DEF-FN CALL 2530,SYNTAX-Z Jump forward if checking syntax.
JR Z,1F6A,DEF-FN-1 Otherwise bass-by the 'DEF FN' statement.
LD A,+CE
JR 1E39,PASS-BY

First consider the variable of the function.
1F6A DEF-FN-1 SET 6,(FLAGS) Signal 'a numeric variable'.
CALL 2C8D,ALPHA Check that the present code is a letter.
JR NC,1F89,DEF-FN-4 Jump forward if not.
RST 0020,NEXT-CHAR Fetch the next character.
CP +24 Jump forward unless it is a $.
JR NZ,1F7D,DEF-FN-2 Change bit 6 as it is a string variable.
RES 6,(FLAGS)
RST 0020,NEXT-CHAR Fetch the next character.
1F7D DEF-FN-2 CP +28 A (') must follow the variable's name.
JR NZ,1FB6,DEF-FN-7
RST 0020,NEXT-CHAR Fetch the next character.
CP +29 Jump forward if it is a ')' as there are no parameters of the function.
JR Z,1FA6,DEF-FN-6

A loop is now entered to deal with each parameter in turn.
1F86 DEF-FN-3 CALL 2C8D,ALPHA The present code must be a letter.
1F89 DEF-FN-4 JP NC,1C8A,REPORT-C
EX DE,HL Save the pointer in DE.
RST 0020,NEXT-CHAR Fetch the next character.
CP +24 Jump forward unless it is a $.
JR NZ,1F94,DEF-FN-5 Otherwise save the new pointer in DE instead.
EX DE,HL
1F94 DEF-FN-5 RST 0020,NEXT-CHAR Fetch the next character.
EX DE,HL Move the pointer to the last character of the name to the HL register pair.
LD BC,+0006
CALL 1655,MAKE-ROOM Now make six locations after that last character and enter a 'number marker' into the first of the new locations.
INC HL
INC HL
LD (HL),+0E
CP +2C If the present character is a .', then jump back as there should be a further parameter; otherwise jump out of the loop.
JR NZ,1FB6,DEF-FN-6
RST 0020,NEXT-CHAR
JR 1F86,DEF-FN-3

Next the definition of the function is considered.
1FA6 DEF-FN-6 CP +29 Check that the ')' does exist.
JR NZ,1FB6,DEF-FN-7
The next character is fetched.

It must be an '='.

Fetch the next character.

Save the nature - numeric or string - of the variable.

Now consider the definition as an expression.

Fetch the nature of the variable and check that it is of the same type as found for the definition.

Give an error report if it is required.

Exit via the CHECK-END subroutine. (Thereby moving on to consider the next statement in the line.)

This subroutine is called in several instances in order to 'return early' from a subroutine when checking syntax. The reason for this is to avoid actually printing characters or passing values to/from the calculator stack.

Is syntax being checked?

Fetch the return address but ignore it in 'syntax-time'.

In 'run-time' make a simple return to the calling routine.

The appropriate channel is opened as necessary and the items to be printed are considered in turn.

Prepare to open channel 'P'.

Jump forward.

Prepare to open channel 'S'.

Unless syntax is being checked open a channel.

Set the temporary colour system variables.

Call the print controlling subroutine.

Move on to consider the next statement; via CHECK-END IF checking syntax.

The print controlling subroutine is called by the PRINT, LPRINT and INPUT command routines.

Jump forward if already at the end of the item list.

Deal with any consecutive position controllers.

Deal with a single print item.

Check for further position controllers and print items until there are none left.

Return now if the present character is a '>'; otherwise consider performing a 'carriage return'.

106
THE 'PRINT A CARRIAGE RETURN' SUBROUTINE

1FF5 PRINT-CR CALL 1FC3.UNSTACK-Z Return if changing syntax.
LD A,+0D Print a carriage return
RST 0010,PRINT-A-1 character and then return.
RET

THE 'PRINT ITEMS' SUBROUTINE

This subroutine is called from the PRINT, LPRINT and INPUT command routines.
The various types of print item are identified and printed.

1FFC PR-ITEM-1 RST 0018,GET-CHAR The first character is fetched.
CP +AC Jump forward unless it is
JR NZ,200E,PR-ITEM-2 an 'AT'.

Now deal with an 'AT'.

CALL 1C79,NEXT-2NUM The two parameters are transferred to the calculator stack.
CALL 1FC3,UNSTACK-Z Return now if checking syntax.
CALL 2307,STK-TO-BC The parameters are compressed into the BC register pair.
LD A,+16 The A register is loaded with
JR 201E,PR-AT-TAB the AT control character before
the jump is taken.

Next look for a 'TAB'.

200E PR-ITEM-2 CP +AD Jump forward unless it is
JR NZ,2024,PR-ITEM-3 a 'TAB'.

Now deal with a 'TAB'.

RST 0020,NEXT-CHAR Get the next character.
CALL 1C82,EXPT-1NUM Transfer one parameter to the calculator stack.
CALL 1FC3,UNSTACK-Z Return now if checking syntax.
CALL 1E99,FIND-INT2 The value is compressed into the BC register pair.
LD A,+17 The A register is loaded with
the TAB control character.

Next consider embedded colour items.

201E PR-AT-TAB RST 0010,PRINT-A-1 Print the control character.
LD A,C Follow it with the first
RST 0010,PRINT-A-1 value.
LD A,B Finally print the second
RST 0010,PRINT-A-1 value; then return.
RET

The 'AT' and the 'TAB' print items are printed by making three calls to

2024 PR-ITEM-3 CALL 21F2,CO-TEMP-3 Return with carry reset if a colour items was found.
RET NC Continue if none were found.
CALL 2070,STR-ALTER Next consider if the stream is to be changed.
RET NC Continue unless it was altered.

The print item must now be an expression, either numeric or string.

CALL 24FB,SCANNING Evaluate the expression but
CALL 1FC3,UNSTACK-Z return now if checking syntax.
BIT 6,(FLAGS) Test the nature of the expression.
CALL Z,2BF1,STK-FETCH If it is string then fetch the neces-
JP NZ,2DE3,PRINT-FP sary parameters; but if it is
numeric then exit via PRINT-FP.
A loop is now set up to deal with each character in turn of the string.

203C PR-STRING
LD A,B
OR C
DEC BC
RET Z
LD A,+(DE)
INC DE
RST 0010,PRINT-A-1
JR 203C,PR-STRING

THE 'END OF PRINTING' SUBROUTINE
The zero flag will be set if no further printing is to be done.

2045 PR-END-Z
CP +29
RET Z
JR 2067,PR-POSN-3

2048 PR-ST-END
CP +0D
RET Z
JR 2067,PR-POSN-3
CP +3A
RET

THE 'PRINT POSITION' SUBROUTINE
The various position controlling characters are considered by this subroutine.

204E PR-POSN-1
RST 0018,GET-CHAR
CP +3B
JR Z,2067,PR-POSN-3
CP +2C
JR NZ,2061,PR-POSN-2
CALL 2530,SYNTAX-Z
JR Z,2067,PR-POSN-3
LD A,+06
RST 0010,PRINT-A-1
JR 2067,PR-POSN-3
CP +27
RET NZ
CALL 1FF5,PR-CR

2061 PR-POSN-2
JR 2045,PR-END-Z
CALL 1C82,EXPT-1NUM
AND A
CALL 1FC3,UNSTACK-Z
CALL 1E94,FIND-INT1
CP +10
RET

THE 'ALTER STREAM' SUBROUTINE
This subroutine is called whenever there is the need to consider whether the user wishes to use a different stream.

2070 STR-ALTER
CP +23
SCF
RET NZ
RST 0020,NEXT-CHAR
CALL 1C82,EXPT-1NUM
AND A
CALL 1FC3,UNSTACK-Z
CALL 1E94,FIND-INT1
CP +10
RET

Return now if there are no characters remaining in the string; otherwise decease the counter. Fetch the code and increment the pointer. The code is printed and a jump taken to consider any further characters.

Return now if the character is a ");
Return now if the character is a 'carriage return'. Make a final test against ";
Return now if the character is a ')'.
Return now if the character is a 'carriage return'.
Return now if the character is a ')'.
Return now if the character is a 'carriage return'.

The zero flag will be reset if the end of the print statement has not been reached.

Give report O if the value is
THE 'INPUT' COMMAND 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.

2089 INPUT
CALL  2530,SYNTAX-Z
JR   Z,2096,INPUT-1
LD   A,+01
CALL 1601,CHAN-OPEN
CALL 0D6E,CLS-LOWER

2096 INPUT-1
LD   (TV-FLAG),+01
CALL 20C1,IN-ITEM-1
CALL 1BEE,CHECK-END
LD   A,(DF-SZ)
JR   C,20AD,INPUT-2
LD   B,A
LD   (S-POSN),BC
LD   A,+19
SUB  B
LD   (SCR-CT),A
RES  0,(TV-FLAG)
CALL 0D09,CL-SET
JP   0D6E,CLS-LOWER

The INPUT items and embedded PRINT items are dealt with in turn by the following loop.

20C1 IN-ITEM-1
CALL 204E,PR-POSN-1
JR   Z,20C1,IN-ITEM-1
CP   +28
JR   NZ,20D8,IN-ITEM-2
RST  0020,NEXT-CHAR
CALL 1FDF,PRINT-2
RST  0018,GET-CHAR
CP   +29
JP   NZ,1C8A,REPORT-C
RST  0020,NEXT-CHAR
JP   21B2,IN-NEXT-2

Now consider whether INPUT LINE is being used.

20D8 IN-ITEM-2
CP   +CA
JR   NZ,20ED,IN-ITEM-3
RST  0020,NEXT-CHAR
CALL 1C1F,CLASS-01
SET  7,(FLAGX)
BIT  6,(FLAGS)
JP   NZ,1C8A,REPORT-C
Jump forward to issue the prompt message.

Proceed to handle simple INPUT variables.

Jump to consider going round the loop again if the present character is not a letter.

Determine the destination address for the variable.

Signal 'not INPUT LINE'.

The prompt message is now built up in the work space.

Jump forward if only checking syntax.

The work space is set to null.

Signal 'string result'.

Signal 'INPUT mode'.

Allow the prompt message only a single location.

Jump forward if using 'LINE'.

A string entry will need three locations.

A 'carriage return' goes into the last location.

Test bit 6 of the C register and jump forward if only one location was required.

The position of the cursor can now be saved.

In the case of INPUT LINE the EDITOR can be called without further preparation but for other types of INPUT the error stack has to be changed so as to trap errors.

Jump forward with INPUT 'LINE'.

Save the current values of CH-ADD & ERR-SP on the machine stack.

This will be the 'return point' in case of errors.

Only change the error stack pointer if using channel 'K'.

Set HL to the start of the INPUT line and remove any floating-point forms. (There will not be any except perhaps after an error.)

Signal 'no error yet'.
CALL 0F2C,EDITOR  Now get the INPUT and with the syntax/run flag indicating syntax, check the INPUT for errors; jump if in order; return to IN-VAR-1 if not.
CALL 21B9,IN-ASSIGN
JR 2161,IN-VAR-4

All the system variables have to be reset before the actual assignment of a value can be made.

215E IN-VAR-3
CALL 0F2C,EDITOR

Get a 'LINE'.

2161 IN-VAR-4
LD (K-CUR-hi)+00 The cursor address is reset.
CALL 21D6,IN-CHAN-K
JR NZ,2174,IN-VAR-5
CALL 111D,ED-COPY
LD BC,(ECHO-E)
CALL 0DD9,CL-SET

The input-line is copied to the display and the position in ECHO-E made the current position in the lower screen.

2174 IN-VAR-5
LD HL,+5C71 This is FLAGX.
RES 5,(HL)
BIT 7,(HL) Jump forward if handling an INPUT LINE.
RES 7,(HL)
JR NZ,219B,IN-VAR-6
POP HL Drop the address IN-VAR-1.
POP HL Reset the ERR-SP to its original address.
POP HL Save the original CH-ADD address in X-PTR.
LD (X-PTR),HL
SET 7,(FLAGS) Now with the syntax/run flag indicating 'run' make the assignment.
CALL 21B9,IN-ASSIGN
LD HL,(X-PTR) Restore the original address to CH-ADD and clear X-PTR.
LD (X-PTR-hi),+00
LD (CH-ADD),HL
JR 21B2,IN-NEXT-2 Jump forward to see if there are further INPUT items.

219B IN-VAR-6
LD HL,(STKBOT) The length of the 'LINE' in the work space is found.
LD DE,(WORKSP)
SCF
SBC, HL,DE DE points to the start and BC holds the length.
LD B,H
LD C,L
CALL 2AB2,STK-ST-$ These parameters are stacked and the actual assignment made.
CALL 2AFF,LET
JR 21B2,IN-NEXT-2 Also jump forward to consider further items.

Further items in the INPUT statement are considered.

21AF IN-NEXT-1 CALL 1FFC,PR-ITEM-1 Handle any print items.
21B2 IN-NEXT-2 CALL 204E,PR-POSN-1 Handle any position controllers.
JP Z,20C1,IN-ITEM-1 Go around the loop again if there are further items; otherwise return.
RET

THE 'IN-ASSIGN' SUBROUTINE
This subroutine is called twice for each INPUT value. Once with the syntax/run flag reset (syntax) and once with it set (run).

2189 IN-ASSIGN
LD HL,(WORKSP) Set CH-ADD to point to the first location of the work.
LD (CH-ADD),HL space and fetch the character.
RST 0016,GET-CHAR
CP +E2 Is it a 'STOP'?
JR Z,21D0,IN-STOP
LD A,(FLAGX) Otherwise make the assignment
CALL 1C59,VAL-FET-2 of the 'value' to the variable.
Report C - Nonsense in BASIC

\[
\text{RST 0018,GET-CHAR}\]
\[
\text{Get the present character and check it is a 'carriage return'. Return if it is.}\n\]

\[
\text{CP +0D}\]
\[
\text{Report CP+0D and check it is a 'carriage return'. Return if it is.}\n\]

\[
\text{RET}\]
\[
\text{Come here if the INPUT line starts with 'STOP'.}\n\]

\[
\text{DEFB +0B}\]
\[
\text{Call the error handling routine.}\n\]

21D0 IN-STOP CALL 2530,SYNTAX-Z
\[
\text{But do not give the error report on the syntax-pass.}\n\]

Report H - STOP in INPUT

\[
\text{RST 0008,ERROR-1}\]
\[
\text{Call the error handling routine.}\n\]

THE 'IN-CHAN-K' SUBROUTINE
This subroutine returns with the zero flag reset only if channel 'K' is being used.

\[
\text{IN-CHAN-K}\]
\[
\text{LD HL,(CURCHL)}\]
\[
\text{The base address of the channel information for the current channel is fetched.}\n\]

\[
\text{INC HL}\]
\[
\text{and the channel code compared to the character 'K'.}\n\]

\[
\text{INC HL}\]
\[
\text{LD A,(HL)}\]
\[
\text{Jump back if it is either a '.', or a ';'; otherwise there has been an error.}\n\]

\[
\text{JR Z,21E1,CO-TEMP-1}\]
\[
\text{Exit via 'report C'.}\n\]

\[
\text{RST 0018,GET-CHAR}\]
\[
\text{Fetch the present character.}\n\]

\[
\text{CP +2C}\]
\[
\text{Jump back if it is either a '.', or a ';'; otherwise there has been an error.}\n\]

\[
\text{JR Z,21E1,CO-TEMP-1}\]
\[
\text{Set if the code is not in the range +D9 to +DE (INK to OVER).}\n\]

\[
\text{JP 1C8A,REPORT-C}\]
\[
\text{The colour item code is preserved whilst CH-ADD is advanced to address the parameter that follows it.}\n\]

\[
\text{RST 0020,NEXT-CHAR}\]
\[
\text{Consider the next character in the BASIC statement.}\n\]

\[
\text{CALL 21F2,CO-TEMP-3}\]
\[
\text{Jump forward to see if the present code represents an embedded 'temporary' colour item. Return carry set if not a colour item.}\n\]

\[
\text{RET C}\]
\[
\text{Exit via 'report C'.}\n\]

THE 'COLOUR ITEM' ROUTINES
This set of routines can be readily divided into two parts:

i. The embedded colour item handler.

ii. The 'colour system variable' handler.

i. Embedded colour items are handled by calling the PRINT-OUT subroutine as required.

A loop is entered to handle each item in turn. The entry point is at CO-TEMP-2.

\[
\text{CO-TEMP-1}\]
\[
\text{RST 0020,NEXT-CHAR}\]
\[
\text{Consider the next character in the BASIC statement.}\n\]

\[
\text{CALL 21F2,CO-TEMP-3}\]
\[
\text{Jump forward to see if the present code represents an embedded 'temporary' colour item. Return carry set if not a colour item.}\n\]

\[
\text{RET C}\]
\[
\text{Exit via 'report C'.}\n\]

\[
\text{RST 0018,GET-CHAR}\]
\[
\text{Fetch the present character.}\n\]

\[
\text{CP +2C}\]
\[
\text{Jump back if it is either a '.', or a ';'; otherwise there has been an error.}\n\]

\[
\text{JR Z,21E1,CO-TEMP-1}\]
\[
\text{Set if the code is not in the range +D9 to +DE (INK to OVER).}\n\]

\[
\text{JP 1C8A,REPORT-C}\]
\[
\text{Return with the carry flag set if the code is not in the range +D9 to +DE (INK to OVER).}\n\]

\[
\text{CCF}\]
\[
\text{The colour item code and the parameter are now 'printed' by calling PRINT-OUT on two occasions.}\n\]

\[
\text{SUB +C9}\]
\[
\text{The token range (+D9 to +DE)}\n\]
is reduced to the control character range (+10 to +15).

PUSH AF
CALL 1C82,EXPT-1NUM
POP AF
AND A
CALL 1FC3,UNSTACK-Z
PUSH AF
CALL 1E94,FIND-INT1
LD D,A
POP AF
RST 0010,PRINT-A-1

The control character code is preserved whilst the parameter is moved to the calculator stack.

A return is made at this point if syntax is being checked.

The control character code is preserved whilst the parameter is moved to the calculator stack.

The control character is sent out.

The parameter is fetched and sent out before returning.

The colour control code will now be +01 for INVERSE and +02 for OVER and the system variable P-FLAG is altered accordingly.

The colour code will now be +01 for INVERSE and +02 for OVER and the system variable P-FLAG is altered accordingly.

Note that all changes are to the 'temporary' system variables.

The colour system variables - ATTR-T, MASK-T & P-FLAG - are altered as required. This subroutine is called by PRINT-OUT. On entry the control character code is in the A register and the parameter is in the D register.

Note that all changes are to the 'temporary' system variables.

2211 CO-TEMP-5 SUB +11 Reduce the range and jump
ADC A,+00
JR Z,2234,CO-TEMP-7
SUB +02 Reduce the range once again
ADC A,+00
JR Z,2273,CO-TEMP-C

The colour control code will now be +01 for INVERSE and +02 for OVER and the system variable P-FLAG is altered accordingly.

CP +01 Prepare to jump with OVER.
LD A,D
LD B,+01 Prepare the mask for OVER.
JR NZ,2228,CO-TEMP-6
RLCA Bit 2 of the A register is to be reset for INVERSE 0 and set for INVERSE 1; the mask is to have bit 2 set.
LD B,+04

2228 CO-TEMP-6 LD C,A
Save the A register whilst the range is tested.
LD A,D
CP +02 INVERSE and OVER is only '0-1'.
JR NC,2244,REPORT-K
LD A,C Fetch the A register.
LD HL,+5C91 It is P-FLAG that is to be changed.
JR 226C,CO-CHANGE
Exit via CO-CHANGE and alter P-FLAG using 'B' as a mask.

2234 CO-TEMP-7 LD A,D
Fetch the parameter.
LD B,+07 Prepare the mask for INK.
JR C,223E,CO-TEMP-8 Jump forward with INK.
RLCA Multiply the parameter for PAPER by eight.
RLCA

223E CO-TEMP-8 LD B,+38 Prepare the mask for PAPER.
LD C,A Save the parameter in the C register whilst the range of the parameter is tested.
LD   A,D
CP   +0A
JR   C,2246,CO-TEMP-9

Fetch the original value.

Only allow PAPER/INK a range of '0' to '9'.

Report K - Invalid colour
2244  REPORT-K   RST  0008,ERROR-1
       DEFB  +13

Call the error handling routine.

Continue to handle PAPER & INK;
2246  CO-TEMP-9   LD   HL,+5C8F
       CP   +08
       JR   C,2258,CO-TEMP-B
       LD   A,(HL)
       JR   Z,2257,CO-TEMP-A
       OR   B
       CPL
       AND  +24
       JR   Z,2257,CO-TEMP-A
LD   A,B

Prepare to alter ATTR-T, MASK-T & P-FLAG.

Fetch the current value of ATTR-T and use it unchanged, by jumping forward, with PAPER/INK '8'.

But for PAPER/INK '9' the PAPER and INK colours have to be black and white.

Jump for black INK/PAPER; but continue for white INK/PAPER.

2257  CO-TEMP-A   LD   C,A
       LD   A,+07
       CP   D
       SBC  A,A
       CALL 226C,CO-CHANGE

The bits of MASK-T are set only when using PAPER/INK '8' or '9'.

Now change MASK-T as needed.

Next P-FLAG is considered.
RLCA
RLCA
AND  +50
LD   B,A
LD   A,+08
CP   D
SBC  A,A
CALL 226C,CO-CHANGE

The bits of P-FLAG are set only when using PAPER/INK '9'.

Continue into CO-CHANGE to manipulate P-FLAG.

Next MASK-T is considered.
LD   A,C
CALL 226C,CO-CHANGE

Move the value.

Now change ATTR-T as needed.

THE 'CO-CHANGE' SUBROUTINE
This subroutine is used to 'impress' upon a system variable the 'nature' of the bits in the A register, The B register holds a mask that shows which bits are to be 'copied over' from A to (HL).
226C  CO-CHANGE   XOR  (HL)
       AND  B
       XOR  (HL)
       LD  (HL),A
       INC  HL
LD   A,B
       RET

The bits, specified by the mask in the B register, are changed in the value and the result goes to form the system variable.

Move on to address the next system variable.

Return with the mask in the A register.

FLASH & BRIGHT are handled by the following routine.
2273 CO-TEMP-C SBC A,A  The zero flag will be set for BRIGHT.
     LD A,D  The parameter is fetched and rotated.
     RRCA
     LD B,+80  Prepare the mask for FLASH.
     JR NZ,227D,CO-TEMP-D  Jump forward with FLASH.
     RRCA  Rotate an extra time and
     LD B,+40  prepare the mask for BRIGHT.
227D CO-TEMP-D LD C,A  Save the value in the C register.
     LD A,D  Fetch the parameter and test its range; only '0', '1' & '8' are allowable.
     CP +08
     JR Z,2287,CO-TEMP-E
     CP +02
     JR NC,2244,REPORT-K
     The system variable ATTR-T can now be altered.
2287 CO-TEMP-E LD A,C  Fetch the value.
     LD HL,+5C8F  This is ATTR-T.
     CALL 226C,CO-CHANGE  Now change the system variable.
     The value in MASK-T is now considered.
     LD A,C  The value is fetched anew.
     RRCA  The set bit of FLASH/BRIGHT
     RRCA  '8' (bit 3) is moved to
     RRCA  bit 7 (for FLASH) or bit 6
     JR 226C,CO-CHANGE  (for BRIGHT).
     Exit via CO-CHANGE.

THE 'BORDER' COMMAND ROUTINE
The parameter of the BORDER command is used with an OUT command to actually alter the colour of the border. The parameter is then saved in the system variable BORDCR.
2294 BORDER CALL 1E94,FIND-INT1  The parameter is fetched
     CP +08  and its range is tested.
     JR NC,2244,REPORT-K  The OUT instruction is then used to set the border colour.
     OUT (+FE),A  The parameter is then multiplied by eight.
     RLCA
     RLCA
     RLCA
     JR NZ,22A6,BORDER-1  If the border colour is a 'light' colour then the INK colour in the editing area is to be black - make the jump.
     XOR +07  Change the INK colour.
22A6 BORDER-1 LD (BORDCR),A  Set the system variable as required and return.
     RET

THE 'PIXEL ADDRESS' SUBROUTINE
This subroutine is called by the POINT subroutine and by the PLOT command routine. It is entered with the co-ordinates of a pixel in the BC register pair and returns with HL holding the address of the display file byte which contains that pixel and A pointing to the position of the pixel within the byte.
22AA PIXEL-ADD LD A,+AF  Test that the y co-ordinate (in
     SUB B  B) is not greater than 175.
     JP C,24F9,REPORT-B  B now contains 175 minus y.
     LD B,A  A holds b7b5b4b3b2b1b0,
     AND A  the bite of B. And now
     RRA 0b7b6b5b4b3b2b1.
     SCF
     RRA  Now 10b7b6b5b4b3b2.
AND A
RRA B
AND +F8
XOR B
Finally 010b7b6b2b1b0, so that
XOR B
H becomes 64 + 8*INT (B/64) +
LD H,A
B (mod 8), the high byte of the
LD A,C
pixel address. C contains X.
RLCA
A starts as c7c6c5c4c3c2c1c0.
RLCA
And is now c2c1c0c7c6c5c4c3.
XOR B
AND +C7
Now c2c1b5b4b3c5c4c3.
RLCA
RLCA
Finally b5b4b3c7c6c5c4c3, so
LD L,A
that L becomes 32*INT (B(mod
LD A,C
64)/8) + INT(x/8), the low byte.
AND +07
A holds x(mod 8): so the pixel
RET
is bit (A - 7) within the byte.

THE 'POINT' SUBROUTINE
This subroutine is called by the POINT function in SCANNING. It is entered with the co-ordinates of a pixel on the calculator stack, and returns a last value of 1 if that pixel is ink colour, and 0 if it is paper colour.

THE 'PLOT' COMMAND ROUTINE
This routine consists of a main subroutine plus one line to call it and one line to exit from it. The main routine is used twice by CIRCLE and the subroutine is called by DRAW. The routine is entered with the co-ordinates of a pixel on the calculator stack. It finds the address of that pixel and plots it, taking account of the status of INVERSE and OVER held in the P-FLAG.
THE 'STK-TO-BC' SUBROUTINE
This subroutine loads two floating point numbers into the BC register pair. It is thus used to pick up parameters in the range +00-+FF. It also obtains in DE the 'diagonal move' values (+/-1,+/-1) which are used in the line drawing subroutine of DRAW.

2307 STK-TO-BC CALL 2314,STK-TO-A First number to A.
LD B,A Hence to B.
PUSH BC Save it briefly.
CALL 2314,STK-TO-A Second number to A.
LD E,C Its sign indicator to E.
POP BC Restore first number.
LD D,C Its signs indicator to D.
LD C,A Second number to C.
RET BC, DE are now as required.

THE 'STK-TO-A' SUBROUTINE
This subroutine loads the A register with the floating point number held at the top of the calculator stack. The number must be in the range 00-FF.

2314 STK-TO-A CALL 2DD5,FP-TO-A Modulus of rounded last value to
JP C,24F9,REPORT-B A if possible; else, report error.
LD C,+01 One to C for positive last value.
RET Z Return if value was positive.
LD C,+FF Else change C to +FF (i.e. minus one). Finished.

THE 'CIRCLE' COMMAND 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. It is illustrated in the BASIC program in the appendix. The notation of that program is followed here.

CIRCLE has four parts:

I. Tests the radius. If its modulus is less than 1, just plot X,Y;
II. Calls CD-PRMS-1 at 2470-24B6, which is used to set the initial parameters for both CIRCLE and DRAW;
III. Sets up the remaining parameters for CIRCLE, including the initial displacement for the first 'arc' (a straight line in fact);
IV. Jumps into DRAW to use the arc-drawing loop at 2420-24FA.

Parts i. to iii. will now be explained in turn.
i. 2320-23AA. The radius, say Z, is obtained from the calculator stack. Its modulus Z is formed and used from now on. If Z is less than 1, it is deleted from the stack and the point X,Y is plotted by a jump to PLOT.

2320 CIRCLE RST 0017,GET-CHAR Get the present character.
CP +2C Test for comma.
JP NZ,1C8A,REPORT-C If not so, report the error.
RST 0020,NEXT-CHAR Get next character (the radius).
CALL 1C82,EXPT-1NUM Radius to calculator stack.
CALL 1BEE,CHECK-END Move to consider next statement if checking syntax.
RST 0028,FP-CALC Use calculator: the stack holds:
DEFB +2A.abs X, Y, Z
DEFB +3D,re-stack Z is re-stacked; its exponent
DEFB +38,end-calc is therefore available.
LD A,(HL) Get exponent of radius.
ii. 233B-2346 and the call to CD-PRMS1. 2*PI is stored in mem-5 and CD-PRMS1 is called. This subroutine stores in the B register the number of arcs required for the circle, viz. A=4*INT (PI*SQR Z^4)+4, hence 4, 8, 12 ..., up to a maximum of 32. It also stores in mem-0 to mem-4 the quantities 2*PI/A, SIN(PI/A), 0, COS (2*PI/A) and SIN (2*PI/A).

233B  C-R-GRE-1  RST  0028,FP-CALC  X, Y, Z, PI/2.
  DEFB  +A3,stk-pl/2
  DEFB  +38, end-calc
  LD  (HL),+83  now increase exponent to 83
  DEFB  +C5, st-mem-5  (2*PI is copied to mem-5).
  DEFB  +02, delete  X, Y, Z
  DEFB  +38, end-calc
  CALL  247D,CD-PRMS1  Set the initial parameters.

iii. 2347-2381: the remaining parameters and the jump to DRAW. A test is made to see whether the initial 'arc' length is less than 1. If it is, a jump is made simply to plot X, Y. Otherwise, the parameters are set: X+Z and X-Z*SIN (PI/A) are stacked twice as start and end point, and copied to COORDS as well; zero and 2*Z*SIN (PI/A) are stored in mem-1 and mem-2 as initial increments, giving as first 'arc' the vertical straight line joining X+Z, y-Z*SIN (PI/A) and X+Z, Y+Z*SIN (PI/A). The arc-drawing loop of DRAW will ensure that all subsequent points remain on the same circle as these two points, with incremental angle 2*PI/A. But it is clear that these 2 points in fact subtend this angle at the point X+Z*(1-COS (PI/A)), Y not at X, Y. Hence the end points of each arc of the circle are displaced right by an amount 2*(1-COS (PI/A)), which is less than half a pixel, and rounds to one pixel at most.

2347  C-ARC-GE1  PUSH  BC  Save the arc-count in B.
  RST  0028,FP-CALC  X,Y,Z.
  DEFB  +31, duplicate  X,Y,Z,Z
  DEFB  +E1, get-mem-1  X,Y,Z,Z,SIN (PI/A)
  DEFB  +04, multiply  X,Y,Z,Z*SIN (PI/A)
  DEFB  +38,end-calc  Z*SIN (PI/A) is half the initial
  LD  A,(HL)  'arc' length; it is tested to see
  CP  +80  whether it is less than 0.5.
  JR  NC,235A,C-ARC-GE1  If not, the jump is made.
  RST  0028,FP-CALC  Otherwise, Z is deleted from the
  DEFB  +02, delete  stack, with the half-arc too; the
  DEFB  +02, delete  machine stack is cleared; and a
  DEFB  +38,end-calc  jump is made to plot X, Y.
  POP  BC
  JP  22DC,PLOT

235A  RST  0028,FP-CALC  X,Y,Z,Z*SIN (PI/A)
  DEFB  +C2, st-mem-2  (Z*SIN (PI/A) to mem-2 for
    now).
  DEFB  +01, exchange  X,Y,Z,Z*SIN (PI/A),Z
  DEFB  +C0, st-mem-0  X,Y,Z,Z*SIN (PI/A),Z
  DEFB  +02, delete  X,Y,Z,Z*SIN (PI/A)
  DEFB  +03, subtract  X, Y - Z*SIN (PI/A)
  DEFB  +01, exchange  Y - Z*SIN (PI/A), X
  DEFB  +E0, get-mem-0  Y - Z*SIN (PI/A), X, Z
  DEFB  +0F, addition  Y - Z*SIN (PI/A), X+Z
  DEFB  +CO, st-mem-0  (X+Z is copied to mem-0)
  DEFB  +01, exchange  X+Z, Y - Z*SIN (PI/A)
  DEFB  +31, duplicate  X+Z, Y-Z*SIN (PI/A), Y-Z*SIN
  DEFB  +E0, get-mem-0  (PI/A)
    sa, sb, sb, sa
THE DRAW COMMAND 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.

The routine has four parts:

I. Just draws a line if only 2 parameters are given or if the diameter of the implied circle is less than 1;
II. Calls CD-PRMS1 at 247D-24B6 to set the first parameters;
III. Sets up the remaining parameters, including the initial displacements for the first arc;
IV. Enters the arc-drawing loop and draws the arc as a series of smaller arcs approximated by straight lines, calling the line-drawing subroutine at 24B7-24FA as necessary.

Two subroutines, CD-PRMS1 and DRAW-LINE, follow the main routine. The above 4 parts of the main routine will now be treated in turn.

i. If there are only 2 parameters, a jump is made to LINE-DRAW at 2477. A line is also drawn if the quantity Z=(ABS X + ABS Y)/ABS SIN(G/2) is less than 1. Z lies between 1 and 1.5 times the diameter of the implied circle. In this section mem-0 is set to SIN (G/2), mem-1 to Y, and mem-5 to G.
ii. Just calls CD-PRMS1. This subroutine saves in the B register the number of shorter arcs required for the complete arc, viz. A=4*INT (G’*SQR Z/8)+4, where G’ = mod G, or 252 if this expression exceeds 252 (as can happen with a large chord and a small angle). So A is 4, 8, 12, ... , up to 252. The subroutine also stores in mem-0 to mem-4 the quantities G/A, SIN (G/2*A), 0, COS (G/A), SIN (G/A).

iii. Sets up the rest of the parameters as follows. The stack will hold these 4 items, reading up to the top: X0+X and Y0+Y as end of last arc; then X0 and Y0 as beginning of first arc. Mem-0 will hold X0 and mem-5 Y0. Mem-1 and mem-2 will hold the initial displacements for the first arc, U and V; and mem-3 and mem-4 will hold COS (G/A) and SIN (G/A) for use in the arc-drawing loop.

The formulae for U and V can be explained as follows. Instead of stepping along the final chord, of length L, say, with displacements X and Y, we want to step along an initial chord (which may be longer) of length L*W, where W=SIN (G/2*A)/SIN (G/2), with displacements X*W and Y*W, but turned through an angle (G/2 - G/2*A), hence with true displacements:

\[ U = Y^*W^*\sin(G/2 - G/2*A) + X^*W^*\cos(G/2 - G/2*A) \]
\[ Y = Y^*W^*\cos(G/2 - G/2*A) - X^*W^*\sin(G/2 - G/2*A) \]

These formulae can be checked from a diagram, using the normal expansion of \( \cos(P - Q) \) and \( \sin(P - Q) \), where \( Q = G/2 - G/2*A \).
DEFB +02,delete X,Y
DEFB +01,exchange Y,X
DEFB +31,duplicate Y,Y
DEFB +E1,get -mem Y,X,W
DEFB +04,multiply Y,X,W
DEFB +C2,stk -mem X*W is copied to mem-2).
DEFB +02,delete Y,X
DEFB +01,exchange X,Y
DEFB +31,duplicate X,Y,Y
DEFB +E1,get -mem Y,X,X,W
DEFB +04,multiply Y,X,X*W
DEFB +E2,stk -mem (X*W is copied to mem-2).
DEFB +02,delete Y,X
DEFB +01,exchange X,Y
DEFB +31,duplicate X,Y,Y
DEFB +E1,get -mem Y,X,Y,W
DEFB +04,multiply Y,X,Y*W
DEFB +E2,get -mem X,Y,Y*W,X*W
DEFB +E5,get -mem X,Y,Y*W,X*W,G
DEFB +E0,get -mem X,Y,Y*W,X*W,G,G/A
DEFB +03,subtract X,Y,Y*W,X*W,G
DEFB +A2,stk -half X,Y,Y*W,X*W, ½
DEFB +04,multiply X,Y,Y*W,X*W, G/2
DEFB +31,duplicate X,Y,Y*W,X*W, F, F
DEFB +1F,sin X,Y,Y*W,X*W, F, SIN F
DEFB +C5,stk -mem (SIN F is copied to mem-5).
DEFB +02,delete X,Y,Y*W,X*W,F
DEFB +20,cos X,Y,Y*W,X*W, COS F
DEFB +C0,stk -mem (COS F is copied to mem-0).
DEFB +02,delete X,Y,Y*W
DEFB +C1,stk -mem (Y*W is copied to mem-1).
DEFB +E5,get -mem X,Y,Y*W,SIN F
DEFB +04,multiply X,Y,Y*W*SIN F
DEFB +E0,get -mem X,Y,Y*W*SIN F,X*W
DEFB +E2,get -mem X,Y,Y*W*SIN F,X*W,COS F
DEFB +04,multiply X,Y,Y*W*SIN F,X*W*COS F
DEFB +0F,addition X,Y,Y*W*SIN F+X*W*COS F=U
DEFB +E1,get -mem X,Y,U,Y*W
DEFB +01,exchange X,Y,Y*W,U
DEFB +C1,stk -mem (U is copied to mem-1)
DEFB +02,delete X,Y,Y*W
DEFB +E0,get -mem X,Y,Y*W,COS F
DEFB +04,multiply X,Y,Y*W*COS F
DEFB +E2,get -mem X,Y,Y*W*COS F,X*W
DEFB +E5,get -mem X,Y,Y*W*COS F,X*W,SIN F
DEFB +04,multiply X,Y,Y*W*COS F,X*W*SIN F
DEFB +03,subtract X,Y,Y*W*COS F - X*W*SIN
F = V
DEFB +C2,stk -mem (V is copied to mem-2).
DEFB +2A,abs X, Y, V' (V' = ABS V)
DEFB +E1,get -mem X, Y, U, V
DEFB +2A,abs X, Y, V', U' (U' = ABS U)
DEFB +0F,addition X, Y, U' + V
DEFB +02,delete X, Y
DEFB +38,stk -mem (DE now points to U' + V').
LD A,(DE)
CP +81
POP BC
JP C,2477,LINE-DRAW from X0, Y0 to X0+X, Y0+Y.
PUSH BC
RST 0028,FP-CALC
DEFB +01,exchange Y, X
DEFB +38,stk -mem (DE now points to U' + V').
LD A,(COORDS-lo)
CALL 2D28,STACK-A
RST 0028,FP-CALC Y, X, X0
LD A,(DE)
CP +81
POP BC
JP C,2477,LINE-DRAW from X0, Y0 to X0+X, Y0+Y.
PUSH BC
RST 0028,FP-CALC
DEFB +01,exchange Y, X
DEFB +38,stk -mem (DE now points to U' + V').
LD A,(COORDS-lo)
CALL 2D28,STACK-A
RST 0028,FP-CALC Y, X, X0
iv. The arc-drawing loop. This is entered at 2439 with the co-ordinates of the starting point on top of the stack, and the initial displacements for the first arc in mem-1 and mem-2. It uses simple trigonometry to ensure that all subsequent arcs will be drawn to points that lie on the same circle as the first two, subtending the same angle at the centre. It can be shown that if 2 points X1, Y1 and X2, Y2 lie on a circle and subtend an angle N at the centre, which is also the origin of co-ordinates, then X2 = X1*COS N - Y1*SIN N, and Y2 = X1*SIN N + Y1*COS N. But because the origin is here at the increments, say Un = Xn+1 - Xn and Vn = Yn+1 - Yn, thus achieving the desired result. The stack is shown below on the (n+1)th pass through the loop, as Xn and Yn are incremented by Un and Vn, after these are obtained from Un-1 and Vn-1. The 4 values on the top of the stack at 2425 are, in DRAW, reading upwards, X0+X, Y0+Y, Xn and Yn but to save space these are not shown until 2439. For the initial values in CIRCLE, see the end of CIRCLE, above. In CIRCLE too, the angle G must be taken to be 2*PI.

2420 DRW-STEPS DEC B B counts the passes through the loop.
JR Z,245F,ARC-END Jump when B has reached zero.
JR 2439,ARC-START Jump into the loop to start.

2425 ARC-LOOP RST 0028,FP-CALC (See text above for the stack).
DEFB +E1,get-mem-1 Un-1
DEFB +31,duplicate Un-1,Un-1
DEFB +E3,get-mem-3 Un-1,Un-1,COS(G/A)
DEFB +04,multiply Un-1,Un-1*COS(G/A)
DEFB +E2,get-mem-2 Un-1,Un-1*COS(G/A),Vn-1
DEFB +E4,get-mem-4 Un-1,Un-1*COS(G/A),Vn-1,
SIN(G/A)
DEFB +04,multiply Un-1,Un-1*COS(G/A),Vn-1*
SIN(G/A)
DEFB +03,subtract Un-1,Un-1*COS(G/A),Vn-1*
SIN(G/A) = Un
DEFB +C1,st-mem-1 (Un is copied to mem-1).
DEFB +02,delete Un-1
DEFB +E4,get-mem-4 Un-1,SIN(G/A)
DEFB +04,multiply Un-1*SIN(G/A)
DEFB +E2,get-mem-2 Un-1*SIN(G/A),Vn-1
DEFB +E3,get-mem-3 Un-1*SIN(G/A),Vn-1,COS(G/A)
DEFB +04,multiply Un-1*SIN(G/A),Vn-1*COS(G/A)
DEFB +0F,addition Un-1*SIN(G/A)+Vn-1*COS
(G/A) = Vn
DEFB +C2,st-mem-2 (Vn is copied to mem-2).
DEFB +02,delete (As noted in the text, the stack
DEFB +38,end-calc in fact holds X0+X,Y0+Y, Xn
and Yn).

2439 ARC-START PUSH BC Save the arc-counter.
RST 0028,FP-CALC X0+X, Y0+Y, Xn, Yn
DEFB +C0,mem-0 (Yn is copied to mem-0).
DEFB +02,delete X0+X, Y0+Y, Xn
DEFB +E1,get-mem-1 X0+X, Y0+Y, Xn, Un
DEFB +0F,addition X0+X, Y0+Y, Xn+Un = Xn+1
DEFB  +31, duplicate X0 + X, Y0 + Y, Xn+1, Xn+1
DEFB  +38, end-calc Next Xn', the approximate value of Xn reached by
       the line-drawing subroutine

LD   A, (COORDS-lo) is copied to A
CALL 2D28, STACK-A and hence to the stack.
RST  0028, FP-CALC X0 + X, Y0 + Y, Xn+1, Xn'
DEFB +03, subtract X0 + X, Y0 + Y, Xn+1, Xn',
                - Xn' = Un'
DEFB +E0, get-mem-0 X0 + X, Y0 + Y, Xn+1, Un', Yn
DEFB +E2, addition X0 + X, Y0 + Y, Xn+1, Un', Yn +
               Vn = Yn+1
DEFB +C0, st-mem-0 (Yn+1 is copied to mem-0).
DEFB +01, exchange X0 + X, Y0 + Y, Xn+1, Un'
DEFB +E0, get-mem-0 X0 + X, Y0 + Y, Xn+1, Yn+1,
               Un', Yn+1
LD   A, (COORDS-hi), Yn', approximate like Xn', is
CALL 2D28, STACK-A copied to A and hence to the
       stack.
RST  0028, FP-CALC X0 + X, Y0 + Y, Xn+1, Yn+1,
DEFB +03, subtract X0 + X, Y0 + Y, Xn+1, Yn+1,
               Un', Yn+1
DEFB +38, end-calc
CALL 24B7, DRAW-LINE The next 'arc' is drawn.
POP BC The arc-counter is restored.
DJNZ 2425, ARC-LOOP Jump if more arcs to draw.
       245F   ARC-END
RST  0028, FP-CALC The co-ordinates of the end
DEFB +02, delete of the last arc that was drawn
DEFB +02, delete are now deleted from the stack.
DEFB +01, exchange Y0 + Y, X0 + X
DEFB +38, end-calc
LD   A, (COORDS-lo) The X-co-ordinate of the end of
CALL 2D28, STACK-A the last arc that was drawn, say
RST  0028, FP-CALC Xz', is copied to the stack.
DEFB +03, subtract Y0 + Y, X0 + X - Xz'
DEFB +01, exchange X0 + X - Xz', Y0 + Y
DEFB +38, end-calc
LD   A, (COORDS-hi) The Y-co-ordinate is obtained.
CALL 2D28, STACK-A
RST  0028, FP-CALC X0 + X - Xz', Y0 + Y, Yz'
DEFB +03, subtract X0 + X - Xz', Y0 + Y - Yz'
DEFB +38, end-calc
       2477  LINE-DRAW CALL 24B7, DRAW-LINE The final arc is drawn to reach
               X0 + X, Y0 + Y (or close the
               circle).
JP  0040, TEMPS Exit, setting temporary colours.

THE 'INITIAL PARAMETERS' SUBROUTINE
This subroutine is called by both CIRCLE and DRAW to set their initial parameters. It is called by CIRCLE with X, Y and the radius Z on
the top of the stack, reading upwards. It is called by DRAW with its own X, Y, SIN (G/2) and Z, as defined in DRAW i. above, on the top
of the stack. In what follows the stack is only shown from Z upwards.

The subroutine returns in B the arc-count A as explained in both CIRCLE and DRAW above, and in mem-0 to mem-5 the quantities
G/A, SIN (G/2*A), 0, COS (G/A), SIN (G/A) and G. For a circle, G must be taken to be equal to 2*Pi.

       247D   CD-PRMS1 RST  0028, FP-CALC Z
DEFB +31, duplicate Z, Z
DEFB +28, sqr Z, SQR Z
DEFB +34, stk-data Z, SQR Z, 2

123
THE LINE-DRAWING SUBROUTINE

This subroutine is called by DRAW to draw an approximation to a straight line from the point X0, Y0 held in COORDS to the point X0+X, Y0+Y, where the increments X and Y are on the top of the calculator stack. The subroutine was originally intended for the ZX80 and ZX81 8K ROM, and it is described in a BASIC program on page 121 of the ZX81 manual. It is also illustrated here in the Circle program in the appendix.

The method is to intersperse as many horizontal or vertical steps as are needed among a basic set of diagonal steps, using an algorithm that spaces the horizontal or vertical steps as evenly as possible.

24B7  DRAW-LINE CALL  2307,STK-TO-BC  ABS Y to B; ABS X to C;
      LD A,C  JUMP IF ABS X is greater than
      CP B  or equal to ABS Y, so that the

124
JR NC,24C4,DL-X-GE-Y smaller goes to L, and the
LD L,C larger (later) goes to H.
PUSH DE Save diag. step (±1,±1) in DE.
XOR A Insert a vertical step (±1, 0)
LD E,A into DE (D holds SGN Y).
JR 24C8,DL-LARGER Now jump to set H.
24C4 DL-X-GE-Y OR C Return if ABS X and ABS Y
RET Z are both zero.
LD L,B The smaller (ABS Y here) goes
to L.
LD B,C ABS X to B here, for H.
PUSH DE Save the diagonal step here too.
LD D,+00 Hor. step (0, ±1) to DE here.
24CB DL-LARGER LD H,B Larger of ABS X, ABS Y to H
now.

The algorithm starts here. The larger of ABS X and ABS Y, say H, is put into A and reduced to INT (H/2). The H - L horizontal or vertical steps and L diagonal steps are taken (where L is the smaller of ABS X and ABS Y) in this way: L is added to A; if A now equals or exceeds H, it is reduced by H and a diagonal step is taken; otherwise a horizontal or vertical step is taken. This is repeated H times (B also holds H). Note that meanwhile the exchange registers H' and L' are used to hold COORDS.

LD A,B B to A as well as to H.
RRA A starts at INT (H/2).
24CE D-L-LOOP ADD A,L L is added to A.
JR C,24D4,D-L-DIAG If 256 or more, jump - diag.
step.
CP JR H C,24DB,D-L-HR-VT If A is less than H, jump for
SUB H horizontal or vertical step.
LD C,A Reduce A by H.
EXX POP BC Now use the exchange registers.
PUSH BC Save it too.
JR 24DF,D-L-STEP Jump to take the step.
24D4 D-L-DIAG LD C,A Save A (unreduced) in C.
SUB H Reduce A by H.
LD C,A Restore it to C.
EXX POP BC Now use the exchange registers.
PUSH BC Save it too.
24DF D-L-STEP JR 24DF,D-L-STEP Jump to take the step.
LD C,A Save A (unreduced) in C.
PUSH DE Step to stack briefly.
EXX POP BC Get exchange registers.
24DB D-L-HR-VT JR 24DF,D-L-STEP Jump to take the step.
LD C,A Save A (unreduced) in C.
PUSH DE Step to stack briefly.
EXX POP BC Get exchange registers.
24EC D-L-LOOP LD HL,(COORDS) Now take the step: first, 
ADD A,H COORDS to HL′ as the start
point.
LD A,B Y-step from B′ to A.
ADD A,H Add in H′.
INC A Now the X-step; it will be tested
LD A,C for range (Y will be tested in
INC A PLOT).
ADD A,L Add L′ to C′ in A, jump on
24F7,D-L-RANGE carry for further test.
JR Z,24F9,REPORT-B Zero after no carry denotes
X-position -1, out of range.
JR 24F9,REPORT-B

24EC D-L-LOOP DEC A Restore true value to A.
LD C,A Value to C for plotting.
CALL 22E5,PLOT-SUB Plot the step.
EXX LD A,C Restore main registers.
LD A,C C back to A to continue
algorithm.
DJNZ 24CE,D-L-LOOP Loop back for 8 steps (i.e. H
steps).
POP DE Clear machine stack.
RET Finished.

125
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>24F7</td>
<td>D-L-RANGE</td>
<td>JR</td>
<td>Z,24EC,D-L-PLOT</td>
</tr>
</tbody>
</table>

Zero after carry denotes X. position 255, in range.

Report B - Integer out of range

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>24F9</td>
<td>REPORT-B</td>
<td>RST</td>
<td>0008,ERROR-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DEFB</td>
<td>+0A</td>
</tr>
</tbody>
</table>

Call the error handling routine.
EXPRESSION EVALUATION

THE 'SCANNING' SUBROUTINE

This subroutine is used to produce an evaluation result of the 'next expression'.

The result is returned as the 'last value' on the calculator stack. For a numerical result, the last value will be the actual floating point number. However, for a string result the last value will consist of a set of parameters. The first of the five bytes is unspecified, the second and third bytes hold the address of the start of the string and the fourth and fifth bytes hold the length of the string.

Bit 6 of FLAGS is set for a numeric result and reset for a string result.

When a next expression consists of only a single operand, e.g. ... A ..., ... RND ..., ... A$ (4, 3 TO 7) ..., then the last value is simply the value that is obtained from evaluating the operand.

However when the next expression contains a function and an operand, e.g. ..., CHR$ A ..., ..., NOT A ..., SIN 1 ..., the operation code of the function is stored on the machine stack until the last value of the operand has been calculated. This last value is then subjected to the appropriate operation to give a new last value.

In the case of there being an arithmetic or logical operation to be performed, e.g. ... A+B ..., A*B ..., A=B ..., then both the last value of the first argument and the operation code have to be kept until the last value of the second argument has been found. Indeed the calculation of the last value of the second argument may also involve the storing of last values and operation codes whilst the calculation is being performed.

It can therefore be shown that as a complex expression is evaluated, e.g. ..., CHR$ (T+A - 26*INT ((T+A)/26)+65) ..., a hierarchy of operations yet to be performed is built up until the point is reached from which it must be dismantled to produce the final last value.

Each operation code has associated with it an appropriate priority code and operations of higher priority are always performed before those of lower priority.

The subroutine begins with the A register being set to hold the first character of the expression and a starting priority marker - zero - being put on the machine stack.

24FB  SCANNING RST 0018,GET-CHAR The first character is fetched.
LD B,+00 The starting priority marker.
PUSH BC It is stacked.

24FF  S-LOOP-1 LD C,A The main re-entry point.
LD HL,+2596 Index into scanning function
CALL 16DC,INDEXER table with the code in C.
LD A,C Restore the code to A.
JP NC,2684,S-QUOTE Jump if code not found in table.
LD B,+00 Use the entry found in the table
ADD HL,BC to build up the required address
JP (HL) in HL, and jump to it.

Four subroutines follow; they are called by routines from the scanning function table. The first one, the 'scanning quotes subroutine', is used by S-QUOTE to check that every string quote is matched by another one.

250F  S-QUOTE-S CALL 0074,CH-ADD+1 Point to the next character.
INC BC Increase the length count by one.
CP +0D Is it a carriage return?
JP Z,1C8A,REPORT-C Report the error if so.
CP +22 Is it another ""?
JR NZ,250F,S-QUOTE-S Loop back if it is not.
CALL 0074,CH-ADD+1 Point to next character; set zero
CP +22 flag if it is another "".
RET Finished.

The next subroutine, the 'scanning: two co-ordinates' subroutine, is called by S-SCREEN$, S-ATTR and S-POINT to make sure the required two co-ordinates are given in their proper form.

2522  S-2-COORD RST 0020, NEXT-CHAR Fetch the next character.
CP +28 Is it a ""?
THE 'SYNTAX-Z' SUBROUTINE

At this point the 'SYNTAX-Z' subroutine is interpolated. It is called 32 times, with a saving of just one byte each call. A simple test of bit 7 of FLAGS will give the zero flag reset during execution and set during syntax checking.

i.e. SYNTAX gives Z set.

2530 SYNTAX-Z BIT 7,(FLAGS) Test bit 7 of FLAGS. Finished.

The next subroutine is the 'scanning SCREEN$ subroutine', which is used by S-SCREENS$ to find the character that appears at line x, column y of the screen. It only searches the character set 'pointed to' to CHARS.

Note: This is normally the characters +20 (space) to +7F (©) although the user can alter CHARS to match for other characters, including user-defined graphics.

2535 S-SCRN$-S CALL 2307,STK-TO-BC x to C, y to B; 0<=x<23
LD HL,(CHARS) decimal; 0<=y<=31 decimal.
LD DE,+0100 CHARS plus 256 decimal gives
ADD HL,DE HL pointing to the character set.
LD A,C x is copied to A.
RRCA The number 32 (decimal) * (x
RRCA mod 8) + y is formed in A and
copied to E.
RRCA This is the low byte of the
AND +E0 required screen address.
XOR +18 Now the number 64 (decimal) +
XOR +40 8*INT (x/8) is inserted into D.
LD D,A DE now holds the screen address.
LD B,+60 B counts the 96 characters.
254F S-SCRN-LP PUSH BC Save the count.
PUSH DE And the screen pointer.
PUSH HL And the character set pointer.
PUSH A,(DE) Get first row of screen character.
XOR (HL) Match with row from character
INC A Jump if direct match found.
JR C,257A,S-SC-MTCH Now test for match with inverse
counts through the other 7
JR A,S-SC-MTCH rows.
INC D Move DE to next row (add 256
INC HL dec.).
INC HL Move HL to next row (i.e. next byte).
LD A,(DE) Get the screen row.
XOR (HL) Match with row from the ROM.
XOR A Include the inverse status.
JR NZ,2573,S-SC-NXT Jump if row fails to match.
DJNZ 255D,S-SC-ROWS Jump back till all rows done.
PUSH BC Discard character set pointer.
POP BC And screen pointer.
POPC
LD  A,+80
SUB  B
LD  BC,+0001
RST 0030,BC-SPACES
LD  (DE),A
JR  257D,S-SCR-STO

2573  S-SCR-NXT
POPHL
LD  DE,+0008
ADDHL,DE
POP  DE
POP  BC
DJNZ  254F,S-SCRN-LP
LD  C,B

257D  S-SCR-STO
JP 2AB2,STK-STO-$

Note: This exit, via STK-STO-$, is a mistake as it leads to ‘double storing’ of the string result (see S-STRING, 25DB). The instruction line should be ‘RET’.

The last of these four subroutines is the ‘scanning attributes subroutine’. It is called by S-ATTR to return the value of ATTR (x,y) which codes the attributes of line x, column y on the television screen.

2580  S-ATTR-S
CALL  2307,STK-TO-BC
LD  A,C
RRCA
RRCA
RRCA
LD  C,A
AND  +0E
XOR  B
LD  L,A
LD  A,C
AND  +03
XOR  +58
LD  H,A
LD  A,(HL)
JP  2D28,STACK-A

THE SCANNING FUNCTION TABLE
This table contains 8 functions and 4 operators. It thus incorporates 5 new Spectrum functions and provides a neat way of accessing some functions and operators which already existed on the ZX81.

<table>
<thead>
<tr>
<th>location</th>
<th>code</th>
<th>offset</th>
<th>name</th>
<th>address of handling routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>2596</td>
<td>22</td>
<td>1C</td>
<td>S-QUOTE</td>
<td>25B3</td>
</tr>
<tr>
<td>2598</td>
<td>28</td>
<td>4F</td>
<td>S-BRACKET</td>
<td>25E8</td>
</tr>
<tr>
<td>259A</td>
<td>2E</td>
<td>F2</td>
<td>S-DECIMAL</td>
<td>268D</td>
</tr>
<tr>
<td>259C</td>
<td>2B</td>
<td>12</td>
<td>S-U-PLUS</td>
<td>25AF</td>
</tr>
<tr>
<td>259E</td>
<td>A8</td>
<td>56</td>
<td>S-FN</td>
<td>25F5</td>
</tr>
<tr>
<td>25A0</td>
<td>A5</td>
<td>57</td>
<td>S-AND</td>
<td>25F8</td>
</tr>
<tr>
<td>25A2</td>
<td>A7</td>
<td>84</td>
<td>S-PI</td>
<td>2627</td>
</tr>
<tr>
<td>25A4</td>
<td>A6</td>
<td>8F</td>
<td>S-KEYS</td>
<td>2634</td>
</tr>
<tr>
<td>25A6</td>
<td>C4</td>
<td>E6</td>
<td>S-BIN (EQU. S-DECIMAL)</td>
<td>268D</td>
</tr>
<tr>
<td>25A8</td>
<td>AA</td>
<td>BF</td>
<td>S-SCREENS</td>
<td>2668</td>
</tr>
<tr>
<td>25AA</td>
<td>AB</td>
<td>C7</td>
<td>S-ATTR</td>
<td>2672</td>
</tr>
</tbody>
</table>

129
THE SCANNING FUNCTION ROUTINES

25AF S-U-PLUS RST 0020,NEXTCHAR For unary plus, simply move on to the next character and jump back to the main re-entry of SCANNING.

The 'scanning QUOTE routine': This routine deals with string quotes, whether simple like "name" or more complex like "a "white" lie" or the seemingly redundant VAL$ ""a"".

25BE S-Q-AGAIN CALL 250F,S-QUOTE-S Call it again for a third quote. And again for the fifth, seventh etc.

CALL 2530,SYNTAX-Z If testing syntax, jump to reset bit 6 of FLAGS and to continue scanning.

RST 0030,BC-SPACES Make space in the work space for the string and the terminating quote.

25CB S-Q-COPY LD A,(HL) Get a character from the string.
INC HL Point to the start of the string.
LD (DE),A Copy last one to work space.
INC DE Point to the next space.
CP +22 Is last character a ""?
JR NZ,25CB,S-Q-COPY If not, jump to copy next one. But if it was, do not copy next one; if next one is a ",", jump to copy the one after it; otherwise, finished with copying.
LD A,(HL)
INC HL
CP +22
JR Z,25CB,S-Q-COPY
PUSH HL Get the pointer to the start.
PUSH DE Save the pointer to the first space.
DEC BC Get true length to BC.

Note that the first quote was not counted into the length; the final quote was, and is discarded now. Inside the string, the first, third, fifth, etc., quotes were counted in but the second, fourth, etc., were not.

25D9 S-Q-PRMS DEC BC Get start of copied string.
RST DE Restore start of copied string. This is FLAGS; this entry point is used whenever bit 6 is to be reset and a string stacked if executing a line. This is done now.

25E8 S-BRACKET RST 0020,NEXT-CHAR The 'scanning BRACKET routine' simply gets the character and calls SCANNING recursively.

CALL 24FB,SCANNING
CP +29
JP NZ,1C8A,REPORT-C Report the error if no matching bracket; then continue scanning.
RST 0020,NEXT-CHAR
This routine, for user-defined functions, just jumps to the 'scanning FN subroutine'.

25F8 S-RND CALL 2530,SYNTAX-Z Unless syntax is being checked, JR Z,2626,S-RND-END jump to calculate a random number.
LD BC,(SEED) Fetch the current value of SEED.
CALL 2D28,STACK-BC Put it on the calculator stack.
RST 0028,FP-CALC Now use the calculator,
DEFB +A1,stk-one The 'last value' is now
DEFB +0F,addition SEED+1.
DEFB +34,stk-data Put the decimal number 75
DEFB +37,exponent+87 on the calculator stack.
DEFB +16,(+00,+00,+00) 'last value' (SEED+1)*75.
DEFB +04,multiply
DEFB +34,stk-data See STACK LITERALS to see
DEFB +80,(four bytes) how bytes are expanded so as to
DEFB +41,exponent +91 put the decimal number 65537
DEFB +00,+00,+80,(+00) on the calculator stack.
DEFB +32,n-mod-m Divide (SEED+1)*75 by 65537
to give a 'remainder' and an
'answer'.
DEFB +02,delete Discard the 'answer'.
DEFB +A1,stk-one The 'last value' is now
DEFB +03,subtract 'remainder' - 1.
DEFB +31,duplicate Make a copy of the 'last value'.
DEFB +38,end-calc The calculation is finished.
CALL 2DA2,FP-TO-BC Use the 'last value' to give the
LD (SEED),BC new value for SEED.
LD A,(HL) Fetch the exponent of 'last
LD (HL),A value'.
AND A Jump forward if the exponent is
JR Z,2625,S-RND-END zero.
SUB +10 Reduce the exponent, i.e. divide
LD (HL),A 'last value' by 65536 to give the
required 'last value'.
INC (HL) The exponent is incremented
thereby doubling the 'last value'
giving PI.

The 'scanning-PI routine': unless syntax is being checked the value of 'PI' is calculated and forms the 'last value' on the calculator stack.

2627 S-PI CALL 2530,SYNTAX-Z Test for syntax checking.
JR Z,2630,S-PI-END Jump if required.
RST 0028,FP-CALC Now use the calculator.
DEFB +A3,stk-pi/2 The value of PI/2 is put on the
calculator stack as the 'last value'.
INC (HL) The exponent is incremented
thereby doubling the 'last value'
giving PI.

2634 S-INKEY$ LD BC,+105A Priority +10 hex, operation
RST 0020,NEXT-CHAR code +5A for the 'read-in'
CP +23 subroutine.
JP Z,270D,S-PUSH-PO If next char. is '!', jump.
LD HL,+5C3B This is FLAGS.
RES 6,(HL) Reset bit 6 for a string result.
BIT 7,(HL) Test for syntax checking.

JP 27BD,S-FN-SBRN The 'scanning FN routine'.

This routine, for user-defined functions, just jumps to the 'scanning FN subroutine'.

25F8 S-RND CALL 2530,SYNTAX-Z Unless syntax is being checked, JR Z,2626,S-RND-END jump to calculate a random number.
LD BC,(SEED) Fetch the current value of SEED.
CALL 2D28,STACK-BC Put it on the calculator stack.
RST 0028,FP-CALC Now use the calculator,
DEFB +A1,stk-one The 'last value' is now
DEFB +0F,addition SEED+1.
DEFB +34,stk-data Put the decimal number 75
DEFB +37,exponent+87 on the calculator stack.
DEFB +16,(+00,+00,+00) 'last value' (SEED+1)*75.
DEFB +04,multiply
DEFB +34,stk-data See STACK LITERALS to see
DEFB +80,(four bytes) how bytes are expanded so as to
DEFB +41,exponent +91 put the decimal number 65537
DEFB +00,+00,+80,(+00) on the calculator stack.
DEFB +32,n-mod-m Divide (SEED+1)*75 by 65537
to give a 'remainder' and an
'answer'.
DEFB +02,delete Discard the 'answer'.
DEFB +A1,stk-one The 'last value' is now
DEFB +03,subtract 'remainder' - 1.
DEFB +31,duplicate Make a copy of the 'last value'.
DEFB +38,end-calc The calculation is finished.
CALL 2DA2,FP-TO-BC Use the 'last value' to give the
LD (SEED),BC new value for SEED.
LD A,(HL) Fetch the exponent of 'last
LD (HL),A value'.
AND A Jump forward if the exponent is
JR Z,2625,S-RND-END zero.
SUB +10 Reduce the exponent, i.e. divide
LD (HL),A 'last value' by 65536 to give the
required 'last value'.
INC (HL) The exponent is incremented
thereby doubling the 'last value'
giving PI.

The 'scanning-PI routine': unless syntax is being checked the value of 'PI' is calculated and forms the 'last value' on the calculator stack.
JR Z,2665,S-INK$-EN Jump if required.
CALL 028E,KEY-SCAN Fetch a key-value in DE.
LD C,+00 Prepare empty string; stack it if too many keys pressed.
JR NZ,2660,S-IK$-STK Test the key value; stack empty string if unsatisfactory.
CALL 031E,K-TEST +FF to D for L made (bit 3 set).
LD E,A Key-value to E for decoding.
PUSH AF Save the ASCII value briefly.
LD BC,+00 One space is needed in the work space.
RST 0030,BC Make it now.
POP AF Restore the ASCII value.
LD (DE),A Prepare to stack it as a string.
LD C,+01 Its length is one.
2660 S-IK$-STK LD B,+00 Complete the length parameter.
CALL 2AB2,STK-STK$ Stack the required string.
2668 S-SCREEN$ CALL 2522,S-2-COORD Check that 2 co-ordinates are given.
CALL NZ,2535,S-SCRNS$-S Call the subroutine unless checking syntax; then get next character and jump back.
RST 0020,NEXT-CHAR CALL 25DB,S-STRING
JP 26C3,S-NUMERIC JR 26C3,S-DECIMAL
The 'scanning DECIMAL routine' which follows deals with a decimal point or a number that starts with a digit. It also takes care of the expression 'BIN', which is dealt with in the 'decimal to floating-point' subroutine.

266B S-ATTR CALL 2522,S-2-COORD Check that 2 co-ordinates are given.
CALL NZ,2580,S-ATTR-S Call the subroutine unless checking syntax; then get next character and jump forward.
RST 0020,NEXT-CHAR CALL 26C3,S-NUMERIC
JR 26C3,S-ALPHNUM
267A S-POUNTP CALL 2522,S-2-COORD Check that 2 co-ordinates are given.
CALL NZ,22CB,POINT-SUB Call the subroutine unless checking syntax; then get the next character and jump forward.
RST 0020,NEXT-CHAR CALL 26C3,S-NUMERIC
JR 26C3,S-DECIMAL
2684 S-ALPHNUM CALL 2C88,ALPHANUM Is the character alphanumeric?
JR NC,26DF,S-NEGATE Jump if not a letter or a digit.
CP +41 Now jump if it a letter; otherwise continue on into S-DECIMAL.

The action taken is now very different for syntax checking and line execution. If syntax is being checked then the floating-point form has to be calculated and copied into the actual BASIC line. However when a line is being executed the floating-point form will always be available so it is copied to the calculator stack to form a 'last value'.

During syntax checking:
CALL 2C9B,DEC-TO-FP The floating-point form is found.
RST 0018,GET-CHAR Set HL to point one past the last digit.
LD BC,+0006 Six locations are required.
CALL 1655,MAKE-ROOM Make the room in the BASIC line.
INC HL Point to the first free space.
LD (HL),+0E Enter the number marker code.
INC HL Point to the second location.
EX DE,HL This pointer is wanted in DE.
LD HL,(STKEND) Fetch the 'old' STKEND.
LD C,+05 There are 5 bytes to move.
AND A Clear the carry flag.
SBC HL,BC The 'new' STKEND='old' STKEND -5.
LD (STKEND),HL Move the floating-point number from the calculator stack to the line.
LDIR DE,HL Put the line pointer in HL.
DEC HL Point to the last byte added.
CALL 0077,TEMP-PTR1 This sets CH-ADD.
JR 26C3,S-NUMERIC Jump forward.

During line execution:
26B5 S-STK-DEC RST 0018,GET-CHAR Get the current character.
26B6 S-SD-SKIP INC HL Now move on to the next character in turn until
LD A,(HL) the number marker code is found.
CP +OE INC HL Point to the first byte of the number.
JR NZ,26B6,S-SD-SKIP CALL 33B4,STACK-NUM Move the floating-point number.
LD (CH-ADD),HL Set CH-ADD.

A numeric result has now been identified, coming from RND, PI, ATTR, POINT or a decimal number, therefore bit 6 of FLAGS must be set.
26C3 S-NUMERIC SET 6,(FLAGS) Set the numeric marker flag.
JR 26DD,S-CONT-1 Jump forward.

THE SCANNING VARIABLE ROUTINE
When a variable name has been identified a call is made to LOOK-VARS which looks through those variables that already exist in the variables area (or in the program area at DEF FN statements for a user-defined function FN). If an appropriate numeric value is found then it is copied to the calculator stack using STACK-NUM. However a string or string array entry has to have the appropriate parameters passed to the calculator stack by the STK-VAR subroutine (or in the case of a user-defined function, by the STK-F-ARG subroutine as called from LOOK-VARS).

26C9 S-LETTER CALL 28B2,LOOK-VARS Look in the existing variables for the matching entry.
JP C,1C2E,REPORT-2 An error is reported if there is no existing entry.
CALL Z,2996,STK-VARS Stack the parameters of the string entry/return numeric element base address.
LD A,(FLAGS) Fetch FLAGS.
CP +C0 JR C,26BD,S-CONT-1 Test bits 6 and 7 together.
INC HL IRC 1 or both bits are reset.
CALL 33B4,STACK-NUM CALL 33B4,STACK-NUM Move the number.
LD (CH-ADD),HL 26DD S-CONT-1 Jump forward.
JR 2712,S-CONT-2

The character is tested against the code for '-', thus identifying the 'unary minus' operation.

Before the actual test the B register is set to hold the priority +09 and the C register the operation code +D8 that are required for this operation.

CP +2D JR Z,270D,S-PUSH-PO Is it a '-'?
JR Z,270D,S-PUSH-PO Jump forward if it is 'unary minus'.

Next the character is tested against the code for 'VAL$', with priority 16 decimal and operation code 18 hex.
LD BC,+1018  Priority 16 dec, operation code
         +18 hex.
CP +AE     Is it 'VAL$'?  
JR Z,270D,S-PUSH-PO Jump forward if it is 'VAL$'.

The present character must now represent one of the functions CODE to NOT, with codes +AF to +C3.

SUB +AF    The range of the functions is changed from +AF to +C3 to range +00 to +14 hex.
JP C,1C8A,REPORT-C Report an error if out of range.

The function 'NOT' is identified and dealt with separately from the others.

LD BC,+04F0 Priority +04, operation code +F0.
CP +14    Is it the function 'NOT'?  
JR Z,270D,S-PUSH-PO Jump if it is so.
JP NC,1C8A,REPORT-C Check the range again.

The remaining functions have priority 16 decimal. The operation codes for these functions are now calculated. Functions that operate on strings need bit 6 reset and functions that give string results need bit 7 set in their operation codes.

LD B,+10  Priority 16 decimal.
ADD A,+DC The function range is now +DC +EF.
LD C,A    Transfer the operation code.
CP +DF    Separate CODE, VAL and LEN
JR NC,2707,S-NO-TO-S which operate on strings to give numerical results.
RES 6,C

2707 S-NO-TO-S CP +EE Separate STR$ and CHR$ which operate on numbers to give string results.
JR C,2700,S-PUSH-PO Mark the operation codes.
RES 7,C The other operation codes have bits 6 and 7 both set.

The priority code and the operation code for the function being considered are now pushed on to the machine stack. A hierarchy of operations is thereby built up.

270D S-PUSH-PO PUSH BC Stack the priority and operation codes before moving on to consider the next part of the expression.
RST 0020,NEXT-CHAR
JP 24FF,S-LOOP-1

The scanning of the line now continues. The present argument may be followed by a ')', a binary operator or, if the end of the expression has been reached, then e.g. a carriage return character or a colon, a separator or a 'THEN'.

2712 S-CONT-2 RST 0018,GET-CHAR Fetch the present character.
2713 S-CONT-3 CP +28 Jump forward if it is not a ')', which indicates a parenthesised expression.
JR NZ,2723,S-OPTR

If the 'last value' is numeric then the parenthesised expression is a true sub-expression and must be evaluated by itself. However if the 'last value' is a string then the parenthesised expression represents an element of an array or a slice of a string. A call to SLICING modifies the parameters of the string as required.

BIT 6,(FLAGS) Jump forward if dealing with a numeric parenthesised expression.
JR NZ,2734,S-LOOP
CALL 2A52,SLICING Modify the parameters of the 'last value'.
RST 0020,NEXT-CHAR Move on to consider the next character.
JR 2713,S-CONT-3
If the present character is indeed a binary operator it will be given an operation code in the range +C3 - +CF hex, and the appropriate priority code.

2723 S-OPERTR LD B,+00 Original code to BC to index
    LD C,A into table of operators.
    LD HL,+2795 The pointer to the table.
    CALL 16DC,INDEXER Index into the table.
    JR NC,2734,SLOOP Jump forward if no operation found.
    LD C,(HL) Get required code from the table.
    LD HL,+26ED The pointer to the priority table: i.e. 26ED +C3 gives 27B0 as the first address.
    ADD HL,BC Index into the table.
    LD B,(HL) Fetch the appropriate priority.

The main loop of this subroutine is now entered. At this stage there are:
I. A 'last value' on the calculator stack.
II. The starting priority market on the machine stack below a hierarchy, of unknown size, of function and binary operation codes. This hierarchy may be null.
III. The BC register pair holding the 'present' operation and priority, which if the end of an expression has been reached will be priority zero.

Initially the 'last' operation and priority are taken off the machine stack and compared against the 'present' operation and priority.

If the 'present' priority is higher than the 'last' priority then an exit is made from the loop as the 'present' priority is considered to bind tighter than the 'last' priority.

However, if the present priority is less binding, then the operation specified as the 'last' operation is performed. The 'present' operation and priority go back on the machine stack to be carried round the loop again. In this manner the hierarchy of functions and binary operations that have been queued are dealt with in the correct order.

2734 S-LOOP POP DE Get the 'last' operation and priority.
    LD A,D The priority goes to the A register.
    CP B Compare 'last' against 'present'.
    JR C,2773,S-TIGHTER Exit to wait for the argument.
    AND A Are both priorities zero?
    JP Z,0018,GET-CHAR Exit via GET-CHAR thereby making 'last value' the required result.

Before the 'last' operation is performed, the 'USR' function is separated into 'USR number' and 'USR string' according as bit 6 of FLAGS was set or reset when the argument of the function was stacked as the 'last value'.

PUSH BC Stack the 'present' values.
    LO HL,+5C3B This is FLAGS.
    LD A,E The 'last' operation is compared with the code for USR, which will give 'USR number' unless modified; jump if not 'USR'.
    CP +ED BIT 6,(HL) Test bit 6 of FLAGS.
    JR NZ,274C,S-STK-LST Jump if it is set ('USR number').
    LD E,+99 Modify the 'last' operation code: 'offset' 19, +80 for string input and numerical result ('USR string').

274C S-STK-LST PUSH DE Stack the 'last' values briefly.
    CALL 2530,SYNTAX-Z Do not perform the actual operation if syntax is being checked.
LD A,E          The 'last' operation code.
AND +3F         Strip off bits 6 and 7 to convert
LD B,A          the operation code to a
calculator offset.
RST 0028,FP,CALC Now use the calculator.
DEFB +3B,fp-calc-2 Perform the actual operation
DEFB +38,end-calc It has been done.
JR 2764,S-RUNTEST Jump forward.

An important part of syntax checking involves the testing of the operation to ensure that the nature of the 'last value' is of the correct
type for the operation under consideration.

275B S-SYNTST LD A,E Get the 'last' operation code.
XOR (FLAGS) This tests the nature of the 'last
AND +40 value' against the requirement
of the operation. They are to be
the same for correct syntax.
2761 S-RPORT-C JP NZ,1C8A,REPORT-C Jump if syntax fails.

Before jumping back to go round the loop again the nature of the 'last value' must be recorded in FLAGS.

2764 S-RUNTEST POP DE Get the 'last' operation code.
LD HL,+5C3B This is FLAGS.
SET 6,(HL) Assume result to be numeric.
BIT 7,E Jump forward if the nature of
JR NZ,2770,S-LOOPEND 'last value' is numeric.
RES 6,(HL) It is string.
2770 S-LOOPEND POP BC Get the 'present' values into BC;
JR 2734,S-LOOP Jump back.

Whenever the 'present' operation binds tighter, the 'last' and the 'present' values go back on the machine stack. However if the 'present'
operation requires a string as its operand then the operation code is modified to indicate this requirement.

2773 S-TIGHTER PUSH DE The 'last' values go on the stack.
LD A,C Get the 'present' operation
code.
BIT 6,(FLAGS) Do not modify the operation
JR NZ,2790,S-NEXT code if dealing with a numeric
AND +3F operand.
ADD A,+08 Increase the code by +08 hex.
LD C,A Return the code to the C
registry.
CP +10 Is the operation 'AND'?
JR NZ,2788,S-NOT-AND Jump if it is not so.
SET 6,C 'AND' requires a numeric
JR 2790,S-NEXT operand.
2788 S-NOT-AND JR C,2761,S-RPORT-C The operations -,*,/,,^ and OR
JR NZ,2790,S-NEXT are not possible between strings.
CP +17 Is the operation a '+'?
JR Z,2790,S-NEXT Jump if it is so.
SET 7,C The other operations yield a
numeric result.
2790 S-NEXT PUSH BC The 'present' values go on the
JR 24FF,S-LOOP-1 machine stack.
RST 0020,NEXT-CHAR Consider the next character.
JP 24FF,S-LOOP-1 Go around the loop again.
THE TABLE OF OPERATORS

<table>
<thead>
<tr>
<th>location</th>
<th>code</th>
<th>operator</th>
<th>location</th>
<th>code</th>
<th>operator</th>
<th>location</th>
<th>code</th>
<th>operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>2795</td>
<td>2B</td>
<td>CF</td>
<td>27A3</td>
<td>3C</td>
<td>CD</td>
<td>27AF</td>
<td>00</td>
<td>End marker</td>
</tr>
<tr>
<td>2797</td>
<td>2D</td>
<td>C3</td>
<td>27A5</td>
<td>C7</td>
<td>C9</td>
<td>27A7</td>
<td>C8</td>
<td>CA</td>
</tr>
<tr>
<td>2799</td>
<td>2A</td>
<td>C4</td>
<td>27A9</td>
<td>C9</td>
<td>CB</td>
<td>27AB</td>
<td>C6</td>
<td>C7</td>
</tr>
<tr>
<td>279B</td>
<td>2F</td>
<td>C5</td>
<td>27AD</td>
<td>C6</td>
<td>C8</td>
<td>27AE</td>
<td>C5</td>
<td>C7</td>
</tr>
<tr>
<td>279D</td>
<td>5E</td>
<td>C6</td>
<td>27AF</td>
<td>00</td>
<td>End marker</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE TABLE OF PRIORITIES (precedence table)

<table>
<thead>
<tr>
<th>location</th>
<th>priority</th>
<th>operator</th>
<th>location</th>
<th>priority</th>
<th>operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>27B0</td>
<td>06</td>
<td>-</td>
<td>27B7</td>
<td>05</td>
<td>&gt;=</td>
</tr>
<tr>
<td>27B1</td>
<td>08</td>
<td>*</td>
<td>27B8</td>
<td>05</td>
<td>&lt;&gt;</td>
</tr>
<tr>
<td>27B2</td>
<td>08</td>
<td>/</td>
<td>27B9</td>
<td>05</td>
<td>&gt;</td>
</tr>
<tr>
<td>27B3</td>
<td>0A</td>
<td>^</td>
<td>27BA</td>
<td>05</td>
<td>&lt;</td>
</tr>
<tr>
<td>27B4</td>
<td>02</td>
<td>OR</td>
<td>27BB</td>
<td>05</td>
<td>=</td>
</tr>
<tr>
<td>27B5</td>
<td>03</td>
<td>AND</td>
<td>27BC</td>
<td>06</td>
<td>+</td>
</tr>
<tr>
<td>27B6</td>
<td>05</td>
<td>&lt;=</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE 'SCANNING FUNCTION' SUBROUTINE

This subroutine is called by the 'scanning FN routine' to evaluate a user defined function which occurs in a BASIC line. The subroutine can be considered in four stages:

I. The syntax of the FN statement is checked during syntax checking.
II. During line execution, a search is made of the program area for a DEF FN statement, and the names of the functions are compared, until a match is found - or an error is reported.
III. The arguments of the FN are evaluated by calls to SCANNING.
IV. The function itself is evaluated by calling SCANNING, which in turn calls LOOK-VARS and so the "STACK FUNCTION ARGUMENT" subroutine.

27BD S-FN-SBRN CALL 2530,SYNTAX-Z Unless syntax is being checked, JR NZ,27F7,SF-RUN a jump is made to SF-RUN. RST 0020,NEXT-CHAR Get the first character of the name. CALL 2C8D,ALPHA If it is not alphabetic, then JP NC,1C8A,REPORT-C report the error. RST 0020,NEXT-CHAR Get the next character. CP +24 Is it a '$'? PUSH AF Save the zero flag on the stack. JR NZ,27D0,SF-BRKT-1 Jump if it was not a '$'. RST 0020,NEXT-CHAR But get the next character if it was. 27D0 SF-BRKT-1 CP +28 If the character is not a '(', then JR NZ,27E6,SF-RPRT-C report the error. RST 0020,NEXT-CHAR Get the next character. CP +29 Is it a ')'? JR Z,27E9,SF-FLAG-6 Jump if it is; there are no arguments. 27D9 SF-ARGMTS CALL 24FB,SCANNING Within the loop, call SCANNING to check the syntax of each argument and to insert floating-point numbers. RST 0018,GET-CHAR Get the character which follows the argument; if it is not a ').' JR NZ,27E4,SF-BRKT-2 then jump - no more arguments. RST 0020,NEXT-CHAR Get the first character in the next argument.
JR 27D9, SF-ARGMTS Loop back to consider this argument.

27E4 SF-BRKT-2 CP +29 Is the current character a ')'? 
27E6 SF-RPRT-C JP NZ,1C8A,REPORT-C Report the error if it is not. 
27E9 SF-FLAG-6 RST 0020,NEXT-CHAR Point to the next character in the BASIC line. 
LD HL, +5C3B This is FLAGS; assume a string-valued function and reset bit 6 of FLAGS. 
RES 6,(HL) 
POP AF 
JR Z,27F4, SF- SYN-EN Restore the zero flag, jump if the FN is indeed string valued. 
SET 6,(HL) Otherwise, set bit 6 of FLAGS 
27F4 SF-SYN-EN JP 2712, S- CONT-2 Jump back to continue scanning the line. 

ii. During line execution, a search must first be made for a DEF FN statement.

27F7 SF-RUN RST 0020, NEXT-CHAR Get the first character of the name. 
AND +DF Reset bit 5 for upper case. 
LD B, A Copy the name to B. 
RST 0020, NEXT-CHAR Get the next character. 
SUB +24 Subtract 24 hex, the code for '$'. 
LD C, A Copy the result to C (zero for a string, non-zero for a numerical function). 
JR NZ,2802, SF- ARGMT1 Jump if non-zero: numerical function. 
RST 0020, NEXT-CHAR Get the next character, the '. 
2802 SF-ARGMT1 RST 0020, NEXT-CHAR Get 1st character of 1st argument. 
PUSH HL Save the pointer to it on the stack. 
LD HL,(PROG) Point to the start of the program. 
2808 SF-FND-DF DEC HL Go back one location. 
LD DE, +00CE The search will be for 'DEF FN'. 
PUSH BC Save the name and 'string status'. 
CALL 1D86, LOOK-PROG Search the program now. 
POP BC Restore the name and status. 
JR NC,2814, SF-CP-DEF Jump if a DEF FN statement found. 

REPORT P - FN without DEF. 

2812 REPORT-P RST 0008, ERROR-1 Call the error handling routine. 
DEFB +18 

When a DEF FN statement is found, the name and status of the two functions are compared: if they do not match, the search is resumed. 

2814 SF-CP-DEF PUSH HL Save the pointer to the DEF FN character in case the search has to be resumed. 
CALL 28AB, FN-SKPOVR Get the name of the DEF FN function. 
AND +DF Reset bit 5 for upper case. 
CP B Does it match the FN name? 
JR NZ,2825, SF-NOT-FD Jump if it does not match. 
CALL 28AB, SF-SKPOVR Get the next character in the DEF FN. 
SUB +24 Subtract 24 hex, the code for '$'. 
CP C Compare the status with that of FN.
Jump if complete match now found.

Jump back for a further search.

The correct DEF FN statement has now been found. The arguments of the FN statement will be evaluated by repeated calls of SCANNING, and their 5 byte values (or parameters, for strings) will be inserted into the DEF FN statement in the spaces made there at syntax checking. HL will be used to point along the DEF FN statement (calling RST 0020, NEXT-CHAR, as needed).

If HL is now pointing to a '$', move on to the '('.

Discard the pointer to 'DEF FN'.

Get the pointer to the first argument of FN, and copy it to CH-ADD.

Move past the '(' now.

Save this pointer on the stack.

Save the 'string status' of the argument.

Point to the 1st of the 5 bytes in DEF FN.

Save this pointer on the stack.

Now evaluate the argument.

Get the no./string flag into A.

Test bit 6 of it against the result of SCANNING.

Give report Q if they did not match.

Get the pointer to the first of the 5 spaces in DEF FN into DE.

Point HL at STKEND.

BC will count 5 bytes to be moved.

First, decrease STKEND by 5,
LD (STKEND),HL so deleting the 'last value' from the stack.
LDIR Copy the 5 bytes into the spaces in DEF FN.
EX DE,HL Point HL at the next code.
DEC HL Ensure that HL points to the character after the 5 bytes.
CALL 28AB,FN-SKPOVR Is it a ')'?
CP +29 Jump if it is: no more arguments in the DEF FN statement.
JR Z,2885,SF-R-BR-2
PUSH HL It is a ';': save the pointer to it.
RST 0018,GET-CHAR Get the character after the last argument that was evaluated from FN.
CP +2C If it is not a ';': jump: mismatched arguments of FN and DEF FN.
JR NZ,288B,REPORT-Q
RST 0020,NEXT-CHAR Point CH-ADD to the next argument of FN.
POP HL Point HL to the '),' in DEF FN again.
CALL 28AB,FN-SKPOVR Move HL on to the next argument in DEF FN.
JR 2843,SF-ARG-LP Jump back to consider this argument.
2885 SF-R-BR-2 PUSH HL Save the pointer to the ')') in DEF FN.
RST 0018,GET-CHAR Get the character after the last argument in FN.
CP +29 Is it a ')')?
JR Z,288D,SF-VALUE If so, jump to evaluate the function; but if not, give report Q.
REPORT Q - Parameter error.
288B REPORT-Q RST 0008,ERROR-1 Call the error handling routine.
DEFB +19
iv. Finally, the function itself is evaluated by calling SCANNING, after first setting DEFADD to hold the address of the arguments as they occur in the DEF FN statement. This ensures that LOOK-VARS, when called by SCANNING, will first search these arguments for the required values, before making a search of the variables area.
288D SF-VALUE POP DE Restore pointer to ')') in DEF FN.
EX DE,HL Get this pointer into HL.
LD (CH-ADD),HL Insert it into CH-ADD.
LD HL,(DEFADD) Get the old value of DEFADD.
EX (SP),HL Stack it, and get the start address of the arguments area of DEF FN into DEFADD.
LD (DEFADD),HL
PUSH DE Save address of ')') in FN.
RST 0020,NEXT-CHAR Move CH-ADD on past ')') and 'w' to the start of the expression for the function in DEF FN.
RST 0020,NEXT-CHAR CALL 24FB,SCANNING Now evaluate the function.
POP HL Restore the address of ')') in FN.
LD (CH-ADD),HL Store it in CH-ADD.
POP HL Restore original value of DEFADD.
LD (DEFADD),HL Put it back into DEFADD.
THE 'FUNCTION SKIPOVER' SUBROUTINE

This subroutine is used by FN and by STK-F-ARG to move HL along the DEF FN statement while leaving CM-ADD undisturbed, as it points along the FN statement.

28AB  FN-SKPOVR  INC  HL  Point to the next code in the statement.
LD  A,(HL)  Copy the code to A.
CP  +21  Jump back to skip over it if it is a control code or a space.
JR  C,28AB,FN-SKPOVR  a control code or a space.
RET  Finished.

THE 'LOOK-VARS' SUBROUTINE

This subroutine is called whenever a search of the variables area or of the arguments of a DEF FN statement is required. The subroutine is entered with the system variable CH-ADD pointing to the first letter of the name of the variable whose location is being sought. The name will be in the program area or the work space. The subroutine initially builds up a discriminator byte, in the C register, that is based on the first letter of the variable's name. Bits 5 & 6 of this byte indicate the type of the variable that is being handled.

The B register is used as a bit register to hold flags.

28B2  LOOK-VARS  SET  6,(FLAGS)  Presume a numeric variable.
RST  0018,GET-CHAR  Get the first character into A.
CALL  2C8D,ALPHA  Is it alphabetic?
JP  NC,1C8A,REPORT  Give an error report if it is not so.
PUSH  HL  Save the pointer to the first letter.
AND  +1F  Transfer bits 0 to 4 of the letter to the C register; bits 5 & 7 are always reset.
LD  C,A  presumption.
RST  0020,NEXT-CHAR  Get the 2nd character into A.
PUSH  HL  Save this pointer also.
CP  +28  Is the 2nd character a '?'?
JR  Z,28EF,V-RUN/SYN  Separate arrays of numbers.
SET  6,C  Now set bit 6.
CP  +24  Is the 2nd character a '$'?
JR  Z,28DE,V-STR-VAR  Separate all the strings.
SET  5,C  Now set bit 5.
CALL  2C88,ALPHANUM  If the variable's name has only one character then jump forward.
JR  NC,28E3,V-TEST-FN Lookup.

Now find the end character of a name that has more than one character.

28D4  V-CHAR  CALL  2C88,ALPHANUM  Is the character alphanumeric?
JR  NC,28EF,V-RUN/SYN  Jump out of the loop when the end of the name is found.
RES  6,C  Mark the discriminator byte.
RST  0020,NEXT-CHAR  Get the next character.
JR  28D4,V-CHAR  Go back to test it.

Simple strings and arrays of strings require that bit 6 of FLAGS is reset.

28DE  V-STR-VAR  RST  0020,NEXT-CHAR  Step CH-ADD past the '$'.
RES  6,(FLAGS)  Reset the bit 6 to indicate a string.

If DEFADD-hi is non-zero, indicating that a 'function' (a 'FN') is being evaluated, and if in 'run-time', a search will be made of the arguments in the DEF FN statement.

28E3  V-TEST-FN  LD  A,(DEFADD-hi)  Is DEFADD-hi zero?
AND A
JR Z,28EF,V-RUN/SYN If so, jump forward.
CALL 2530,SYNTAX-Z In 'run-time'?
JP NZ,2951,STK-F-ARG If so, jump forward to search the DEF FN statement.

Otherwise (or if the variable was not found in the DEF FN statement) a search of variables area will be made, unless syntax is being checked.

28EF V-RUN/SYN LD B,C Copy the discriminator bytes to the B register.
CALL 2530,SYNTAX-Z Jump forward if in
JR NZ,28FD,V-RUN 'run-time'.
LD A,C Move the discriminator to A.
AND +E0 Drop the character code part.
SET 7,A Indicate syntax by setting bit 7.
LD C,A Restore the discriminator.
JR 2934,V-SYNTAX Jump forward to continue.

A BASIC line is being executed so make a search of the variables area.

28FD V-RUN LD HL,(VARS) Pick up the VARS pointer.

Now enter a loop to consider the names of the existing variables.

2900 V-EACH LD A,(HL) The 1st. letter of each existing variable.
AND +7F Match on bits 0 to 6.
JR Z,2932,V-80-BYTE Jump when the '80-byte' is reached.
CP C The actual comparison.
JR NZ,292A,V-NEXT Jump forward if the 1st characters do not match.
RLA Rotate A leftwards and then ADD A,A double it to test bits 5 & 6.
JR P,293F,V-FOUND-2 Strings and array variables.
JR C,293F,V-FOUND-2 Simple numeric and FOR-NEXT variables.

Long names are required to be matched fully.

POP DE Take a copy of the pointer PUSH DE to the 2nd. character.
PUSH HL Save the 1st letter pointer.

2912 V-MATCHES INC HL Consider the next character.
2913 V-SPACES LD A,(DE) Fetch each character in turn.
INC DE Point to the next character.
CP A The character a 'space'?
JR Z,2912,V-SPACES Ignore the spaces.
OR +20 Set bit 5 so as to match lower and upper case letters.
CP (HL) Make the comparison.
JR Z,2912,V-MATCHES Back for another character if it does match.
OR +80 Will it match with bit 7 set?
CP (HL) Try it.
JR NZ,2929,V-GET-PTR Jump forward if the 'last characters' do not match.
LD A,(DE) Check that the end of the CALL 2C88,ALPHANUM name has been reached before
JR NC,293E,V-FOUND-1 jumping forward.

In all cases where the names fail to match the HL register pair has to be made to point to the next variable in the variables area.

2929 V-GET-PTR POP HL Fetch the pointer.
292A V-NEXT PUSH BC Save B & C briefly.
CALL 19B8,NEXT-ONE DE is made to point to the next variable.
Switch the two pointers.
Get B & C back.
Go around the loop again.

Switch the two pointers.
Get B & C back.
Go around the loop again.

Switch the two pointers.
Get B & C back.
Go around the loop again.

Switch the two pointers.
Get B & C back.
Go around the loop again.

Switch the two pointers.
Get B & C back.
Go around the loop again.

Switch the two pointers.
Get B & C back.
Go around the loop again.

Switch the two pointers.
Get B & C back.
Go around the loop again.

Switch the two pointers.
Get B & C back.
Go around the loop again.

Switch the two pointers.
Get B & C back.
Go around the loop again.

Switch the two pointers.
Get B & C back.
Go around the loop again.
THE 'STACK FUNCTION ARGUMENT' SUBROUTINE

This subroutine is called by LOOK-VARS when DEFADD-hi in non-zero, to make a search of the arguments area of a DEF FN statement, before searching in the variables area. If the variable is found in the DEF FN statement, then the parameters of a string variable are stacked and a signal is given that there is no need to call STK/VAR. But it is left to SCANNING to stack the value of a numerical variable at 26DA in the usual way.

2951 STK-F-ARG
LD HL,(DEFADD) Point to the 1st character in the arguments area and put it into A.
LD A,(HL) Is it a ')'?
CP +29
JP Z,28EF,V-RUN/SYN Jump to search the variables area.

295A SFA-LOOP
LD A,(HL) Get the next argument in the loop.
OR +60 Set bits 5 & 6, assuming a simple numeric variable; copy it to B.
LD B,A
INC HL Point to the next code.
LD A,(HL) Put it into the A register.
CP +0E Is it the 'number marker' code 0E hex?
JR Z,296B,SFA-LOOP Jump if so: numeric variable.
DEC HL Ensure that HL points to the character, not to a space or control code.
CALL 28AB,FN-SKPOVR Pass on to the next character.
INC HL HL now points to the 'number marker'.
INC HL
RES 5,B Reset bit 5 of B: string variable.
LD A,B Get the variable name into A.
CP C Is it the one we are looking for?
JR Z,2981,SFA-LOOP Jump if it matches.
INC HL Now pass over the 5 bytes of the floating-point number or string parameters to get to the next argument.
INC HL
INC HL
INC HL
CALL 28AB,FN-SKPOVR Pass on to the next character.
CP +29 Is it a ')'?
JP Z,28EF,V-RUN/SYN If so, jump to search the variables area.
CALL 28AB,FN-SKPOVR Point to the next argument.
JR 295A,SFA-LOOP Jump back to consider it.

A match has been found. The parameters of a string variable are stacked, avoiding the need to call the STK-VAR subroutine.

2981 SFA-MATCH
BIT 5,C Test for a numeric variable.
JR NZ,2991,SFA-END Jump if the variable is numeric; SCANNING will stack it.
INC HL Point to the first of the 5 bytes to be stacked.
LD DE,(STKEND) Point DE to STKEND.
CALL 33C0,MOVE-FP Stack the 5 bytes.
EX DE,HL Point HL to the new position of STKEND, and reset the system variable.
LD (STKEND),HL

2991 SFA-END
POP DE Discard the LOOK-VARS pointers (2nd & 1st character pointers).
POP DE
XOR A Return from the search with both the carry and zero flags reset - signalling that a call STK-VAR is not required.
INC A
RET Finished.
THE 'STK-VAR' SUBROUTINE

This subroutine is usually used either to find the parameters that define an existing string entry in the variables area or to return in the HL register pair the base address of a particular element or an array of numbers. When called from DIM the subroutine only checks the syntax of the BASIC statement.

Note that the parameters that define a string may be altered by calling SLICING if this should be specified.

Initially the A and the B registers are cleared and bit 7 of the C register is tested to determine whether syntax is being checked.

```
2996  STK-VAR   XOR    A Clear the array flag.
       LD      B,A Clear the B register for later.
       BIT     7,C Jump forward if syntax is
                   being checked.
       JR      NZ,29E7,SV-COUNT

Next, simple strings are separated from array variables.

       BIT     7,(HL) Jump forward if dealing with
       JR      NZ,29AE,SV-ARRAYS an array variable.

The parameters for a simple string are readily found.

       INC     A Signal 'a simple string'.
       INC     HL Move along the entry.
       LD      C,(HL) Pick up the low length counter.
       INC     HL Advance the pointer.
       LD      B,(HL) Pick up the high length
                   pointer.
       INC     HL Advance the pointer.
       EX      DE,HL Transfer the pointer to the
                   actual string.
       CALL    2AB2,STK-STORE Pass these parameters to the
                   calculator stack.
       RST     0018,GET-CHAR Fetch the present character
                   and jump forward to see if a
                   'slice' is required.

The base address of an element in an array is now found. Initially the 'number of dimensions' is collected.

       INC     HL Step past the length bytes.
       INC     HL
       INC     HL
       LD      B,(HL) Collect the 'number of
                   dimensions'.
       BIT     6,C Jump forward if handling an
                   array of numbers.
       JR      Z,29C0,SV-PTR

If an array of strings has its 'number of dimensions' equal to '1' then such an array can be handled as a simple string.

       DEC     B Decrease the 'number of
       JR      Z,29A1,SV-SIMPLE$ dimensions' and jump if the
                   number is now zero.

Next a check is made to ensure that in the BASIC line the variable is followed by a subscript.

       EX      DE,HL Save the pointer in DE.
       RST     0018,GET-CHAR Get the present character.
       CP      +28 Is it a ')'?
       JR      NZ,2A20,REPORT-3 Report the error if it is not so.
       EX      DE,HL Restore the pointer.

For both numeric arrays and arrays of strings the variable pointer is transferred to the DE register pair before the subscript is evaluated.

       29C0  SV-PTR  EX DE,HL Pass the pointer to DE.
       JR      29E7,SV-COUNT Jump forward.
```
The following loop is used to find the parameters of a specified element within an array. The loop is entered at the mid-point - SV-COUNT -, where the element count is set to zero.

The loop is accessed 'B' times, this being, for a numeric array, equal to the number of dimensions that are being used, but for an array of strings 'B' is one less than the number of dimensions in use as the last subscript is used to specify a 'slice' of the string.

For an array of strings the present subscript may represent a 'slice', or the subscript for a 'slice' may yet be present in the BASIC line.

Enter the loop here.

For an array of strings the present subscript may represent a 'slice', or the subscript for a 'slice' may yet be present in the BASIC line.

Enter the loop here.
count the elements occurring before the specified element. Multiply the counter by the dimension-size.

Add the result of 'INT-EXP1' to the present counter.

Fetch the variable pointer.

Fetch the dimension-number and the discriminator byte.

Keep going round the loop until 'B' equals zero.

The SYNTAX/RUN flag is checked before arrays of strings are separated from arrays of numbers.

When dealing with an array of numbers the present character must be a ')'.

When dealing with an array of strings the length of an element is given by the last 'dimension-size'. The appropriate parameters are calculated and then passed to the calculator stack.

The address of the location before the actual floating-point form can now be calculated.

When dealing with an array of strings the length of an element is given by the last 'dimension-size'. The appropriate parameters are calculated and then passed to the calculator stack.

Note: The first parameter is zero indicating a string from an 'array of strings'
and hence the existing entry is not to be reclaimed.

There are three possible forms of the last subscript. The first is illustrated by - A$(2,4 TO 8) -, the second by - A$(2)(4 TO 8) - and the third by - A$(2) - which is the default form and indicates that the whole string is required.

**RST** 0018, **GET** - **CHAR**  
Get the present character.

**CP** +29  
Is it a ')'?

**JR** Z,2A48, **SV** - **DIM**  
Jump if it is so.

**CP** +2C  
Is it a '('.

**JR** Z,2A20, **REPORT** - 3  
Report the error if not so.

**2A45** **SV** - **SLICE**  
**CALL** 2A52, **SLICING**  
Use SLICING to modify the set of parameters.

**2A48** **SV** - **DIM**  
**RST** 0020, **NEXT** - **CHAR**  
Fetch the next character.

**2A49** **SV** - **SLICE**?  
**CP** +28  
Is it a '('?

**JR** Z,2A45, **SV** - **SLICE**  
Jump back if there is a 'slice' to be considered.

When finished considering the last subscript a return can be made.

**RES** 6,(FLAGS)  
Signal - string result.

**RET**  
Return with the parameters of the required string forming a 'last value' on the calculator stack.

**THE 'SLICING' SUBROUTINE**

The present string can be sliced using this subroutine. The subroutine is entered with the parameters of the string being present on the top of the calculator stack and in the registers A, B, C, D & E. Initially the SYNTAX/RUN flag is tested and the parameters of the string are fetched only if a line is being executed.

**2A52** **SLICING**  
**CALL** 2530, **SYNTAX** - **Z**  
Check the flag.

**CALL** N2,2BF1, **STK** - **FETCH**  
Take the parameters off the stack in 'run-time'.

The possibility of the 'slice' being '(' has to be considered.

**RST** 0020, **NEXT** - **CHAR**  
Get the next character.

**CP** +29  
Is it a ')'?

**JR** Z,2AAD, **SL** - **STORE**  
Jump forward if it is so.

Before proceeding the registers are manipulated as follows:

**PUSH** DE  
The 'start' goes on the machine stack.

**XOR** A  
The A register is cleared and saved.

**PUSH** AF  
The 'length' is saved briefly.

**LD** DE,+0001  
Presume that the 'slice' is to begin with the first character.

**RST** 0018, **GET** - **CHAR**  
Get the first character.

**POP** HL  
Pass the 'length' to HL.

The first parameter of the 'slice' is now evaluated.

**CP** +CC  
Is the present character a 'TO'?

**JR** Z,2A81, **SL** - **SECOND**  
The first parameter, by default, will be '1' if the jump is taken.

**POP** AF  
At this stage A is zero.

**CALL** 2ACD, **INT** - **EXP2**  
BC is made to hold the first parameter. A will hold +FF if there has been an 'out of range' error.

**PUSH** AF  
Save the value anyway.

**LD** D,B  
Transfer the first parameter to DE.

**LD** E,C  
Save the 'length' briefly.
RST 0018,GET-CHAR  Get the present character.
POP HL  Restore the 'length'.
CP +CC  Is the present character a 'TO'?
JR Z,2A81,SL-SECOND Jump forward to consider the second parameter if it is so;
CP +29 otherwise show that there is a closing bracket.

2A7A SL-RPT-C JP NZ,1C8A,REPORT-C

At this point a 'slice' of a single character has been identified. e.g. - A$(4).

LD H,D The last character of the 'slice'
LD L,E is also the first character.
JR 2A94,SL-DEFINE Jump forward.

The second parameter of a 'slice' is now evaluated.

2A81 SL-SECOND PUSH HL Save the 'length' briefly.
RST 0020,NEXT-CHAR Get the next character.
POP HL Restore the 'length'.
CP +29 Is the present character a ')?'
JR Z,2A94,SL-DEFINE Jump if there is not a second parameter.
POP AF If the first parameter was in range A will hold zero;
CALL 2ACD,INT-EXP2 otherwise +FF.
Make BC hold the second parameter.
PUSH AF Save the 'error register'.
RST 0018,GET-CHAR Get the present character.
LD H,B Pass the result obtained from
LD L,C INT-EXP2 to the HL register pair.
CP +29 Check that there is a closing bracket now.
JR NZ,2A7A,SL-RPT-C

The 'new' parameters are now defined.

2A94 SL-DEFINE POP AF Fetch the 'error register'.
EX (SP),HL The second parameter goes on the stack and the 'start' goes to HL.
ADD HL,DE The first parameter is added to the 'start'.
DEC HL Go back a location to get it correct.
EX (SP),HL The 'new start' goes on the stack and the second parameter goes to HL.
AND A Subtract the first parameters from the second to find the length of the 'slice'.
SBC HL,DE A negative 'slice' is a 'null string' rather than an error condition. (See manual.)
LD BC,+0000 Initialise the 'new length'.
JR C,2AA8,SL-OVER
INC HL Allow for the inclusive byte.
AND A Only now test the 'error register'.
JP M,2A20,REPORT-3 Jump if either parameter was out of range for the string.
LD B,H Transfer the 'new length'
LD C,L to BC.
POP DE Get the 'new start'.
RES 6,(FLAGS) Ensure that a string is still indicated.

149
THE 'STK-STORE' SUBROUTINE

This subroutine passes the values held in the A, B, C, D & E registers to the calculator stack. The stack thereby grows in size by 5 bytes with every call to this subroutine.

The subroutine is normally used to transfer the parameters of strings but it is also used by STACK-BC and LOG (2^A) to transfer 'small integers' to the stack.

Note that when storing the parameters of a string the first value stored (coming from the A register) will be a zero if the string comes from an array of strings or is a 'slice' of a string. The value will be '1' for a complete simple string. This 'flag' is used in the 'LET' command routine when the '1' signals that the old copy of the string is to be 'reclaimed'.

THE 'INT-EXP' SUBROUTINE

This subroutine returns the result of evaluating the 'next expression' as an integer value held in the BC register pair. The subroutine also tests this result against a limit-value supplied in the HL register pair. The carry flag becomes set if there is an 'out of range' error. The A register is used as an 'error register' and holds +00 of there is no 'previous error' and +FF if there has been one.
The state of the carry flag and the value held in the D register are now manipulated so as to give the appropriate value for the 'error register'.

```
2AE8  I-CARRY  LD  A,D  Fetch the 'old error value'
       SBC  A,+00  Form the 'new error value';
       +00 if no error at anytime/ +FF or less if an 'out of
       range' error on this pass or on previous ones.
```

Restore the registers before returning.

```
2AEB  I-RESTORE  POP  HL  Restore HL & DE.
       POP  DE
       RET  Return; 'error register' is the A register.
```

**THE 'DE,(DE+1)' SUBROUTINE**

This subroutine performs the construction - LD DE,(DE+1) - and returns HL pointing to 'DE+2'.

```
2AEE  DE,(DE+1)  EX  DE,HL  Use HL for the construction.
       INC  HL  Point to 'DE+1'.
       LD  E,(HL)  In effect - LD E,(DE+1).
       INC  HL  Point to 'DE+2'.
       LD  D,(HL)  In effect - LD D,(DE+2).
       RET  Finished.
```

**THE 'GET-HL"DE' SUBROUTINE**

Unless syntax is being checked this subroutine calls 'HL=HL"DE' which performs the implied construction. Overflow of the 16 bits available in the HL register pair gives the report 'out of memory'. This is not exactly the true situation but it implies that the memory is not large enough for the task envisaged by the programmer.

```
2AF4  GET-HL"DE  CALL  2530,SYNTAX-Z  Return directly if syntax is
       RET  Z  being checked.
       CALL  30A9,HL=HL"DE  Perform the multiplication.
       JP  C,1F15,REPORT-4  Report 'Out of memory'.
       RET  Finished.
```

**THE 'LET' COMMAND ROUTINE**

This is the actual assignment routine for the LET, READ and INPUT commands.

When the destination variable is a 'newly declared variable' then DEST will point to the first letter of the variable's name as it occurs in the BASIC line. Bit 1 of FLAGX will be set.

However if the destination variable 'exists already' then bit 1 of FLAGX will be reset and DEST will point for a numeric variable to the location before the five bytes of the
'old number'; and for a string variable to the first location of the 'old string'. The use of DEST in this manner applies to simple variables and to elements of arrays.
Bit 0 of FLAGX is set if the destination variable is a 'complete' simple string variable. (Signalling - delete the old copy.) Initially the current value of DEST is collected and bit 1 of FLAGS tested.

```plaintext
2AFF  LET  LD   HL,(DEST) Fetch the present address in DEST.
       BIT  1,(FLAGX) Jump if handling a variable
       JR   Z,2B66,L-EXISTS that 'exists already'.
```

A 'newly declared variable' is being used. So first the length of its name is found.

```plaintext
LD   BC,+0005 Presume dealing with a numeric variable - 5 bytes.
```

Enter a loop to deal with the characters of a long name. Any spaces or colour codes in the name are ignored.

```plaintext
2B0B  L-EACH-CH INC   BC Add '1' to the counter for each character of a name.
2B0C  L-NO-SP INC   HL Move along the variable's name.
       CP   A,(HL) Fetch the 'present code'.
       JR   +20 Jump back if it is a 'space'; thereby ignoring spaces.
       JR   Z,2BC0,L-NEWS Jump forward if the code is +21 to +FF.
       CP   +10 Accept, as a final code, those in the range +00 to +0F.
       CP   +16 Also accept the range +16 to +1F.
       JR   NC,2B29,L-SPACES Jump forward as handling a 'newly declared' simple string.
       INC  HL Step past the control code after any of INK to OVER.
       JR   2B0C,L-NO-SP Jump back as these control codes are treated as spaces.
```

Separate 'numeric' and 'string' names.

```plaintext
2B1F  L-TEST-CH CALL  2C88,ALPHANUM Is the code alphanumeric?
       JR   C,2B0B,L-EACH-CH If it is so then accept it as a character of a long name.
       CP   +24 Is the present code a 'S'?
       JP   Z,2BC0,L-NEWS Jump forward as handling a 'newly declared' simple string.
```

The 'newly declared numeric variable' presently being handled will require 'BC' spaces in the variables area for its name and its value. The room is made available and the name of the variable is copied over with the characters being 'marked' as required.

```plaintext
2B29  L-SPACES LD   A,C Copy the 'length' to A.
       LD   HL,(E-LINE) Make HL point to the '80-byte' at the end of the variables area.
       DEC  HL Now open up the variables area.
       CALL 1655,MAKE-ROOM Note: In effect 'BC' spaces are made before the displaced '80-byte'.
       INC  HL Point to the first 'new' byte.
       INC  HL Make DE point to the second 'new' byte.
       EX   DE,HL 'new' byte.
       PUSH DE Save this pointer.
       LD   HL,(DEST) Fetch the pointer to the start of the name.
       DEC  DE Make DE point to the first 'new' byte.
       SUB  +06 Make B hold the 'number of extra letters' that are found in a 'long name'.
       LD   B,A
```
Jump forward if dealing with a variable with a 'short name'.

The 'extra' codes of a long name are passed to the variables area.

```
2B3E  L-CHAR  INC  HL   Point to each 'extra' code.
LD    A,(HL)  Fetch the code.
CP    +21     Accept codes from +21 to +FF;
JR    C,2B3E,L-CHAR  Ignore codes +00 to +20.
OR    +20     Set bit 5, as for lower case letters.
INC    DE     Transfer the codes in turn to the 2nd 'new' byte onwards.
LD    (DE),A  
DJNZ   2B3E,L-CHAR  Go round the loop for all the 'extra' codes.
```

The last code of a 'long' name has to be ORed with +80.

```
OR    +80     Mark the code as required
LD    (DE),A  and overwrite the last code.
```

The first letter of the name of the variable being handled is now considered.

```
LD    A,+C0  Prepare the mark the letter of a 'long' name.
2B4F  L-SINGLE LD    HL,(DEST)  Fetch the pointer to the letter.
XOR   (HL)  A holds +00 for a 'short' name and +C0 for a 'long' name.
OR    +20     Set bit 5, as for lower case letters.
POP    HL     Drop the pointer now.
```

The subroutine L-FIRST is now called to enter the 'letter' into its appropriate location.

```
CALL   2BEA,L-FIRST  Enter the letter and return with HL pointing to 'new 80-byte'.
```

The 'last value' can now be transferred to the variables area. Note that at this point HL always points to the location after the five locations allotted to the number. A 'RST 0028' instruction is used to call the CALCULATOR and the 'last value' is deleted. However this value is not overwritten.

```
2B59  L-NUMERIC  PUSH   HL   Save the 'destination' pointer.
RST   0028,FP-CALC  Use the calculator.
DEFB  +02,delete  This moves STKEND back five bytes.
DEFB  +38,end-calc  
POPB  HL  Restore the pointer.
LD    BC,+0005  Give the number a 'length' of five bytes.
AND   A  Make HL point to the first of the five locations and
SBC   HL,BC  jump forward to make the actual transfer.
JR    2BA6,L-ENTER  
```

Come here if considering a variable that 'exists already'. First bit 6 of FLAGS is tested so as to separate numeric variables from string or array of string variables.

```
2B66  L-EXISTS  BIT   6,(FLAGS)  Jump forward if handling any
JR    Z,2B72,_DELETE  kind of string variable.
```

For numeric variables the 'new' number overwrites the 'old' number. So first HL has to be made to point to the location after the five bytes of the existing entry. At present HL points to the location before the five bytes.

```
LD    DE,+0006  The five bytes of a number +1'.
ADD   HL,DE  HL now points 'after'.
JR    2B59,L-NUMERIC  Jump back to make the actual transfer.
```
The parameters of the string variable are fetched and complete strings separated from 'sliced' strings and array strings.

2B72 L-DELETE$ LD HL,(DEST) Fetch the 'start'. Note: This line is redundant.
LD BC,(STRLEN) Jump if dealing with a complete simple string; the old string will need to be 'deleted' in this case only.
BIT 0,(FLAGX)
JR NZ,2BAF,L-ADD$

When dealing with a 'slice' of an existing simple string, a 'slice' of a string from an array of strings or a complete string from an array of strings there are two distinct stages involved. The first is to build up the 'new' string in the work space, lengthening or shortening it as required. The second stage is then to copy the 'new' string to its allotted room in the variables area. However do nothing if the string has no 'length'.

LD A,B Return if the string is a null string.
OR C
RET Z

Then make the required number of spaces available in the work space.

PUSH HL Save the 'start' (DEST).
RST 0030,BC-SPACES Make the necessary amount of room in the work space.
PUSH DE Save the pointer to the first location.
PUSH BC Save the 'length' for use later on.
LD D,H Make DE point to the last location.
LD E,L INC HL Make HL point 'one past' the new locations.
LD (HL),+20 LDDR Enter a 'space' character. Finish with HL pointing to the first new location.

The parameters of the string being handled are now fetched from the calculator stack.

PUSH HL Save the pointer briefly.
CALL 2BF1,STK-FETCH Fetch the 'new' parameters.
POP HL Restore the pointer.

Note: At this point the required amount of room has been made available in the work space for the 'variable in assignment'. e.g. For statement - LET A$(4 to 8)="abcdefg" - five locations have been made.
The parameters fetched above as a 'last value' represent the string that is to be copied into the new locations with Procrustean lengthening or shortening as required.
The length of the 'new' string is compared to the length of the room made available for it.

EX (SP),HL 'Length' of new area to HL.
AND A 'Pointer' to new area to stack.
SBC HL,BC Compare the two 'lengths' and jump forward if the 'new' string will fit into the room.
ADD HL,BC
JR NC,2B9B,L-LENGTH i.e. No shortening required.
LD B,H However modify the 'new'
LD C,L length if it is too long.
2B9B L-LENGTH EX (SP),HL 'Length' of new area to stack.

As long as the new string is not a 'null string' it is copied into the work space. Procrustean lengthening is achieved automatically if the 'new' string is shorter than the room available for it.
EX DE,HL 'Start' of new string to HL.
  'Pointer' to new area to DE.
LD A,B Jump forward if the
OR C 'new' string is a 'null'
JR Z,2BA3,L-IN-W/S string.
LDIR Otherwise move the 'new'
'new' string to the work space.

The values that have been saved on the machine stack are restored.

2BA3 L-IN-W/S POP BC 'Length' of new area.
POP DE 'Pointer' to new area.
POP HL The start - the pointer
to the 'variable in assignment'
which was originally in DEST.
L-ENTER is now used to pass
the 'new' string to the variables
area.

THE 'L-ENTER' SUBROUTINE
This short subroutine is used to pass either a numeric value, from the calculator stack, or a string, from the work space, to its appropriate position in the variables area. The subroutine is therefore used for all except 'newly declared' simple strings and 'complete & existing' simple strings.

2BA6 L-ENTER EX DE,HL Change the pointers over.
LD A,B Check once again that the
OR C length is not zero.
RET PUSH DE Save the destination pointer.
LDIR Move the numeric value or the string
POP HL Return with the HL register pair pointing to the first byte of the numeric value or the string.

THE LET SUBROUTINE CONTINUES HERE
When handling a 'complete & existing' simple string the new string is entered as if it were a 'newly declared' simple string before the existing version is 'reclaimed'.

2BAF L-ADD$ DEC HL Make HL point to the letter
DEC HL of the variable's name.
DEC HL i.e. DEST - 3.
LD A,(HL) Pick up the letter.
PUSH HL Save the pointer to the 'existing
version'.
PUSH BC Save the 'length' of the
'existing string'.
cALL 2BC6,L-STRING Use L-STRING to add the new string to the variables area.
POP BC Restore the 'length'.
POP HL Restore the pointer.
INC BC Allow one byte for the letter
INC BC and two bytes for the length.
INC BC
JP 19E8,RECLAIM-2 Exit by jumping to RECLAIM-2 which will reclaim the whole
'Newly declared' simple strings are handled as follows:

2BC0 L-NEWS$ LD A,+,DF Prepare for the marking of
the variable's letter.
LD HL,(DEST)     Fetch the pointer to the letter.
AND (HL)         Mark the letter as required.

L-STRING
L-STRING is now used to add the new string to the variables area.

THE 'L-STRING' SUBROUTINE
The parameters of the 'new' string are fetched, sufficient room is made available for it and the string is then transferred.

2BC6    L-STRING
PUSH    AF     Save the variable's letter
CALL    2BF1,STK-FETCH     Fetch the 'start' and the
                          'length' of the 'new' string.
EX DE,HL     Move the 'start' to HL.
ADD HL,BC     Make HL point 'one-past' the string.
PUSH BC     Save the 'length'.
DEC HL     Make HL point to the end of the string.
LD (DEST),HL     Save the pointer briefly.
INC BC     Allow one byte for the letter
        and two bytes for the length.
INC BC
LD HL,(E-LINE)     Make HL point to the
        '80-byte' at the end of the variables area.
CALL    1655,MAKE-ROOM     Now open up the variables area.
                          Note: In effect 'BC' spaces are made before the displaced '80-byte'.
LD HL,(DEST)     Restore the pointer to the end of the 'new' string.
POP BC     Make a copy of the length
PUSH BC     of the 'new' string.
INC BC     Add one to the length in case the 'new' string is a 'null'
        string.
LDDR     Now copy the 'new' string + one byte.
EX DE,HL     Make HL point to the byte
        that is to hold the high-length.
INC HL     Fetch the 'length'.
LD (HL),B     Enter the high-length.
DEC HL     Back one.
LD (HL),C     Enter the low-length.
POP AF     Fetch the variable's letter.

THE 'L-FIRST' SUBROUTINE
This subroutine is entered with the letter of the variable, suitably marked, in the A register. The letter overwrites the 'old 80-byte' in the variables area. The subroutine returns with the HL register pair pointing to the 'new 80-byte'.

2BEA    L-FIRST
DEC HL     Make HL point to the 'old
        80-byte'.
LD (HL),A     It is overwritten with the
        letter of the variable.
LD HL,(E-LINE)     Make HL point to the 'new
        80-byte'.
DEC HL     Finished with all the
        'newly declared variables'.
THE 'STK-FETCH' SUBROUTINE
This important subroutine collects the 'last value' from the calculator stack. The five bytes can be either a floating-point number, in 'short' or 'long' form, or set of parameters that define a string.

2BF1 STK-FETCH  LD    HL,(STKEND)      Get STKEND.
       DEC    HL       Back one.
       LD    B,(HL)    The fifth value.
       DEC    HL       Back one.
       LD    C,(HL)    The fourth value.
       DEC    HL       Back one.
       LD    D,(HL)    The third value.
       DEC    HL       Back one.
       LD    E,(HL)    The second value.
       DEC    HL       Back one.
       LD    A,(HL)    The first value.
       LD    (STKEND),HL Reset STKEND to its new position.
       LD    HL        
       RET            Finished.

THE 'DIM' COMMAND 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.

2C02 DIM       CALL    28B2,LOOK-VARS     Search the variables area.
2C05 D-REPORT-C  JP    NZ,1C8A,REPORT-C       Give report C as there has been an error.
            CALL    2530,SYNTAX-Z       Jump forward if in 'run time'.
            JR    NZ,2C15,D-RUN        Test the syntax for string arrays as if they were numeric.
            RES   6,C                   Check the syntax of the parenthesised expression.
            CALL    2996,STK-VAR       Move on to consider the next statement as the syntax was satisfactory.
            CALL    1BEE,CHECK-END     An 'existing array' is reclaimed.

2C15 D-RUN      JR    C,2C1F,D-LETTER      Jump forward if there is no 'existing array'.
            PUSH   BC                   Save the discriminator byte.
            CALL    19B8,NEXT-ONE      Find the start of the next variable
            CALL    19E8,RECLAIM-2     Reclaim the 'existing array'.
            POP    BC                   Restore the discriminator byte.

The initial parameters of the new array are found.

2C1F D-LETTER   SET    7,C                   Set bit 7 in the discriminator byte.
            LD    B,+00                  Make the dimension counter zero.
            PUSH   BC                   Save the counter and the discriminator byte.
            LD    HL,+0001               The HL register pair is to hold the size of the elements in the array, '1' for a string array/ '5' for a numeric array.
            BIT    6,C                   
            JR    NZ,2C2D,D-SIZE       
            LD    L,+05                 
2C2D D-SIZE     EX    DE,HL                  Element size DE.
The following loop is accessed for each dimension that is specified in the parenthesised expression of the DIM statement. The total number of bytes required for the elements of the array is built up in the DE register pair.

2C2E D-NO-LOOP RST 0020,NEXT-CHAR Advance CH-ADD on each pass.
LD H,+FF Set a 'limit value'.
CALL 2ACC,INT-EXP1 Evaluate a parameter.
JP C,2A20,REPORT-3 Give an error if 'out of range'.
PUSH HL Fetch the dimension-counter and the discriminator byte.
INC H Increase the dimension counter on each pass also.
PUSH BC Save the parameter on each pass through the loop.
LD A,B The parameter is moved to LD L,C the HL register pair.
CALL 2AF4,GET-HL*DE The byte total is built up in HL and the transferred to DE.
RST 0018,GET-CHAR Get the present character.
CP +2C and go around the loop again.
JP Z,2C2E,D-NO-LOOP if there is another dimension.

Note: At this point the DE register pair indicates the number of bytes required for the elements of the new array and the size of each dimension is stacked, on the machine stack.

Now check that there is indeed a closing bracket to the parenthesised expression.

All allowance is now made for the dimension-sizes.

The parameters are now entered.
LD (HL),A  The letter, suitably marked, is entered first.
POP BC    The overall length is fetched
DEC BC    and decreased by '3'.
DEC BC
INC HL    Advance HL.
LD (HL),C Enter the low length.
INC HL    Advance HL.
LD (HL),B Enter the high length.
POP BC    Fetch the dimension counter.
LD A,B    Move it to the A register.
INC HL    Advance HL.
LD (HL),A Enter the dimension count.

The elements of the new array are now 'cleared'.

LD H,D    HL is made to point to the
LD L,E    last location of the array
DEC DE    and DE to the location before
LD (HL),+00 that one.
BIT 6,C   Enter a zero into the last
JR Z,2C7C,DIM-CLEAR location but overwrite it with 'space' if dealing
LD (HL),+20 with an array of strings.

2C7C      DIM-CLEAR
LDDR      Fetch the element byte total.
PPO BC    Clear the array + one extra location.

The 'dimension-sizes' are now entered.

2C7F      DIM-SIZES
POP BC    Get a dimension-size.
LD (HL),B Enter the high byte.
DEC HL    Back one.
LD (HL),C Enter the low byte.
DEC A     Decrease the dimension counter.
JR NZ,2C7F,DIM-SIZES Repeat the operation until all the dimensions have been
RET      considered; then return.

THE 'ALPHANUM' SUBROUTINE
This subroutine returns with the carry flag set if the present value of the A register denotes a valid digit or letter.

2C88      ALPHANUM
CALL 2D1B,NUMERIC Test for a digit; carry will be
CCF reset for a digit.
RET      Complement the carry flag.

THE 'ALPHA' SUBROUTINE
This subroutine returns with the carry flag set if the present value of the A register denotes a valid letter of the alphabet.

2C8D      ALPHA
CP +41 Test against 41 hex, the code for 'A'.
CCF      Complement the carry flag.
RET      Return if not a valid character code.
CP +5B Test against 5B hex, 1 more than code for 'Z'.
RET      Return if an upper case letter.
CP +61 Test against 61 hex, the code for 'a'.

THE 'ALPHANUM' SUBROUTINE
This subroutine returns with the carry flag set if the present value of the A register denotes a valid digit or letter.
THE 'DECIMAL TO FLOATING POINT' SUBROUTINE

As part of syntax checking decimal numbers that occur in a BASIC line are converted to their floating-point forms. This subroutine reads the decimal number digit by digit and gives its result as a 'last value' on the calculator stack. But first it deals with the alternative notation BIN, which introduces a sequence of 0's and 1's giving the binary representation of the required number.

For other numbers, first any integer part is converted; if the next character is a decimal, then the decimal fraction is considered.

THE 'DECIMAL TO FLOATING POINT' SUBROUTINE

As part of syntax checking decimal numbers that occur in a BASIC line are converted to their floating-point forms. This subroutine reads the decimal number digit by digit and gives its result as a 'last value' on the calculator stack. But first it deals with the alternative notation BIN, which introduces a sequence of 0's and 1's giving the binary representation of the required number.

For other numbers, first any integer part is converted; if the next character is a decimal, then the decimal fraction is considered.

THE 'DECIMAL TO FLOATING POINT' SUBROUTINE

As part of syntax checking decimal numbers that occur in a BASIC line are converted to their floating-point forms. This subroutine reads the decimal number digit by digit and gives its result as a 'last value' on the calculator stack. But first it deals with the alternative notation BIN, which introduces a sequence of 0's and 1's giving the binary representation of the required number.

For other numbers, first any integer part is converted; if the next character is a decimal, then the decimal fraction is considered.
DEFB +02,delete
DEFB +38,end-calc

2CDA NXT-DGT-1
RST 0018,GET-CHAR Get the present character.
CALL 2D22,STK-DIGIT If it is a digit then stack it.
JR C,2CEB,E FORMAT If not jump forward.
RST 0028,FP-CALC Now use the calculator.
DEFB +E0,get-mem-0 For each passage of the loop,
DEFB +A4,stk-ten the number saved in the memory
DEFB +05,division area is fetched, divided by 10
DEFB +C0,st-mem-0 and restored: i.e. going from .1
to .01 to .001 etc.
DEFN +04,multiply The present digit is multiplied
DEFB +38,end-calc by the 'saved number' and
DEFB +38,end-calc added to the 'last value'.
RST 0020,NEXT-CHAR Get the next character.
JR 2CDA,NXT-DGT-1 Jump back (one more byte than
needed) to consider it.

Next consider any 'E notation', i.e. the form \( xE^m \) or \( xe^m \) where \( m \) is a positive or negative integer.

2CEB E-FORMAT CP +45 Is the present character an 'E'?
JR Z,2CF2,SIGN-FLAG Jump forward if it is.
CP +65 Is it an 'e'?
RET NZ Finished unless it is so.

2CF2 SIGN-FLAG LD B,+FF Use B as a sign flag, FF for '+'.
RST 0020,NEXT-CHAR Get the next character.
CP +2B Is it a '+'?
JR Z,2CFE,DONE Jump forward.
CP +2D Is it a '-'?
JR NZ,2CFF,ST-E-PART Jump if neither '+' not '-'.
INC B Change the sign of the flag.

2CFE SIGN-DONE RST 0020,NEXT-CHAR Point to the first digit.
2CFF ST-E-PART CALL 2D18,NUMERIC Is it indeed a digit?
JR C,2CCF,DEC-RPT-C Report the error if not.
PUSH BC Save the flag in B briefly.
CALL 2D3B,INT-TO-FP Stack ABS \( m \), where \( m \) is the exponent.
CALL 2DD5,FP-TO-A Transfer ABS \( m \) to A.
POP BC Restore the sign flag to B.
AND A ABS \( m \) is greater than 255 or indeed greater than 127 (other values greater than about 39 will be detected later).
INC B Test the sign flag in B; '+' (i.e. +FF) will now set the zero flag.
JR Z,2D18,E-FP-JUMP Jump if sign of \( m \) is '+'.
NEG Negate \( m \) if sign is '-'.

2D18 E-FP-JUMP JP 2D4F,E-TOO-FP Jump to assign to the 'last value' the result of \( x \cdot 10^m \).

THE 'NUMERIC' SUBROUTINE
This subroutine returns with the carry flag reset if the present value of the A register denotes a valid digit.

2D1B NUMERIC CP +30 Test against 30 hex, the code for '0'.
RET C Return if not a valid character code.
CP +3A Test against the upper limit.
CCF Complement the carry flag.
RET Finished.
THE 'STK DIGIT' SUBROUTINE
This subroutine simply returns if the current value held in the A register does not represent a digit but if it does then the floating-point form for the digit becomes the 'last value' on the calculator stack.

```
2D22  STK-DIGIT  CALL  2D1B.NUMERIC  Is the character a digit?
         RET  C  Return if not in range.
         SUB  +30  Replace the code by the actual digit.
```

THE 'STACK-A' SUBROUTINE
This subroutine gives the floating-point form for the absolute binary value currently held in the A register.

```
2D28  STACK-A  LD  C,A  Transfer the value to the C register.
         LD  B,+00  Clear the B register.
```

THE 'STACK-BC' SUBROUTINE
This subroutine gives the floating-point form for the absolute binary value currently held in the BC register pair.

The form used in this and hence in the two previous subroutines as well is the one reserved in the Spectrum for small integers \( n \), where \(-65535 \leq n \leq 65535\). The first and fifth bytes are zero; the third and fourth bytes are the less significant and more significant bytes of the 16 bit integer \( n \) in two's complement form (if \( n \) is negative, these two bytes hold \( 65536+n \)); and the second byte is a sign byte, 00 for '+' and FF for '-'.

```
2D2B  STACK-BC  LD  IY,+5C3A  Re-initialise IY to ERR-NR.
         XOR  A  Clear the A register.
         LD  E,A  And the E register, to indicate '+'.
         LD  D,C  Copy the less significant byte to D.
         LD  C,B  And the more significant byte to C.
         LD  B,A  Clear the B register.
         CALL  2AB6,STK-STORE  Now stack the number.
         RST  0028,FP-CALC  Make HL point to
         DEFB  +38,end-calc  STKEND-5.
         AND  A  Clear the carry flag.
         RET  Finished.
```

THE 'INTEGER TO FLOATING-POINT' SUBROUTINE
This subroutine returns a 'last value' on the calculator stack that is the result of converting an integer in a BASIC line, i.e. the integer part of the decimal number or the line number, to its floating-point form.

Repeated calls to CH-ADD+1 fetch each digit of the integer in turn. An exit is made when a code that does not represent a digit has been fetched.

```
2D3B  INT-TO-FP  PUSH  AF  Save the first digit - in A.
         RST  0028,FP-CALC  Use the calculator.
         DEFB  +A0,stk-zero  The 'last value' is now zero.
         DEFB  +38,end-calc  STKEND-5.
         POP  AF  Restore the first digit.
```

Now a loop is set up. As long as the code represents a digit then the floating-point form is found and stacked under the 'last value'. The 'last value' is then multiplied by decimal 10 and added to the 'digit' to form a new 'last value' which is carried back to the start of the loop.
If the code represents a digit then stack the floating-point form.

Use the calculator.

'Digit' goes under 'last value'.

Define decimal 10.

'Last value' = 'last value' *10.

'Last value' = 'last value' + 'digit'.

The next code goes into A.

Loop back with this code.
THE ARITHMETIC ROUTINES

THE 'E-FORMAT TO FLOATING-POINT' SUBROUTINE
(Offset 3C - see CALCULATE below: 'e-to-fp')

This subroutine gives a 'last value' on the top of the calculator stack that is the result of converting a number given in the form xEm, where m is a positive or negative integer. The subroutine is entered with x at the top of the calculator stack and m in the A register.

The method used is to find the absolute value of m, say p, and to multiply or divide x by $10^{p}$ according to whether m is positive or negative.

To achieve this, p is shifted right until it is zero, and x is multiplied or divided by $10^{2^{n}}$ for each set bit b(n) of p. Since p is never more than decimal 39, bits 6 and 7 of p will not normally be set.

2D4F  E-TO-FP  
RLCA  Test the sign of m by rotating bit 7 of A into the carry without changing A.
RRCA
JR  NC,2D55,E-SAVE  Jump if m is positive.
CPL  Negate m in A without disturbing the carry flag.
INC A
2D55  E-SAVE  
PUSH AF  Save m in A briefly.
LD HL,+5C92  This is MEMBOT: a sign flag is now stored in the first byte of mem-0, i.e. 0 for '+' and 1 for mem-0.
CALL  350B,FP-0/1  The stack holds x.
RST 0028,FP-CALC  x,10 (decimal)
DEFB +A4,stk-ten  x,10
DEFB +38,end-calc  x
POP AF  Restore m in A.
2D60  E-LOOP  
SRL A  In the loop, shift out the next bit of m, modifying the carry and zero flags appropriately; jump if carry reset.
JR NC,2D71,E-TST-END  Save the rest of m and the flags.
PUSH AF  The stack holds x' and $10^{2^{n}}$, where x' is an interim stage in the multiplication of x by $10^{m}$, and n=0,1,2,3,4 or 5.
RST 0028,FP-CALC  x'', $10^{2^{n}}$
DEFB +C1,stk-mem-1  (10$(2^{n})$ is copied to mem-1).
DEFB +E0,get-mem-0  $x',10^{2^{(2^{n})}}$ (1/0)
DEFB +00,jump-true  $x',10^{2^{(2^{n})}}$
DEFB +04,to E-DIVSN  $x',10^{2^{(2^{n})}}$
DEFB +04,multiply  $x''10^{2^{(2^{n})}}=x''$
DEFB +33,jump  $x''$
DEFB +02,to E-FETCH  $x''$
2D6D  E-DIVSN  
DEFB +05,division  $x''10^{2^{(2^{n})}}=x''$ (x'' is N''*10$(2^{n})$ or x'/10$(2^{n})$ according as m is '+' or '-').
2D6E  E-FETCH  
DEFB +E1,get-mem-1  $x''10^{2^{(2^{n})}}$
DEFB +38,end-calc  $x''10^{2^{(2^{n})}}$
POP AF  Restore the rest of m in A, and the flags.
2D71  E-TST-END  
JR Z,2D7B,E-END  Jump if m has been reduced to zero.
PUSH AF  Save the rest of m in A.
RST 0028,FP-CALC  $x''10^{2^{(2^{n})}}$
DEFB +31,duplicate  $x'',10^{2^{(2^{n})}},10^{2^{(2^{n})}}$
DEFB +04,multiply  $x'',10^{2^{(2^{n})}}$
DEFB +38,end-calc  $x'',10^{2^{(2^{n})}}$
POP AF  Restore the rest of m in A.
JR 2D60,E-LOOP  Jump back for all bits of m.
2D7B  E-END  
RST 0028,FP-CALC  Use the calculator to delete the
THE 'INT-FETCH' SUBROUTINE

This subroutine collects in DE a small integer $n$ ( $-65535 \leq n \leq 65535$ ) from the location addressed by HL: i.e. $n$ is normally the first (or second) number at the top of the calculator stack; but HL can alls access (by exchange with DE) a number which has been deleted from the stack. The subroutine does not itself delete the number from the stack or from memory; it returns HL pointing to the fourth byte of the number in its original position.

```机器语言
2D7F INT-FETCH INC HL Point to the sign byte of the number.
LD C,(HL) Copy the sign byte to C.

The following mechanism will twos complement the number if it is negative (C is FF) but leave it unaltered if it is positive (C is 00)

INC HL Point to the less significant byte.
LD A,(HL) Collect the byte in A.
XOR C Ones complement it if negative
SUB C This adds 1 for negative numbers; it sets the carry unless the byte was 0.
LD E,A Less significant byte to E now.
INC HL Point to the more significant byte.
LD A,(HL) Collect it in A.
ADC A,C Finish two complementing in the case of a negative number; note that the carry is always left reset.
LD D,A More significant byte to D now.
RET Finished.
```
LD (HL),A  Store the byte.
INC HL       Point to the fifth location.
LD (HL),+00  The fifth byte is set to zero.
POP HL      Return with HL pointing to the first byte on n on the stack

THE 'FLOATING-POINT TO BC' SUBROUTINE
This subroutine is called from four different places for various purposes and is used to compress the floating-point 'last value' into the BC register pair. If the result is too large, i.e. greater than 65536 decimal, then the subroutine returns with the carry flag set. If the 'last value' is negative then the zero flag is reset. The low byte of the result is also copied to the A register.

2DA2 FP-TO-BC
RST 0028,FP-CALC Use the calculator to make HL
DEFB +38,end-calc point to STKEND-5
LD A,(HL) Collect the exponent byte of
AND A the 'last value'; jump if it is
JR Z,2DAD,FP-DELETE zero, indicating a 'small integer'.
RST 0028,FP-CALC Now use the calculator to round
DEFB +A2.stk-half the 'last value' to the nearest
DEFB +0F.addition integer, which also changes it to
DEFB +27.int 'small integer' form on the
DEFB +38,end-calc calculator stack if that is possible, i.e. if -65535.5 <= x < 65535.3

2DAD FP-DELETE
RST 0028,FP-CALC Use the calculator to delete the
DEFB +92.delete integer from the stack; DE still
DEFB +38,end-calc points to it in memory (at
PUSH HL STKEND).
PUSH DE Save both stack pointers.
LD B,(HL) HL now points to the number.
CALL 2D7F,INT-FETCH Copy bytes 2, 3 and 4 to C, E and D.
XOR A Clear the A register.
SUB B This sets the carry unless B is zero.
BIT 7,C This sets the zero flag if the number is positive (NZ denotes negative).
LD B,D Copy the high byte to B.
LD C,E And the low byte to C.
LD A,E Copy the low byte to A too.
POP DE Restore the stack pointers.
POP HL
RET

THE 'LOG (2^A)' SUBROUTINE
This subroutine is called by the 'PRINT-FP' subroutine to calculate the approximate number of digits before the decimal in x, the number to be printed, or, if there are no digits before the decimal, then the approximate number of leading zeros after the decimal. It is entered with the A register containing e', the true exponent of \( x \), or \( e'-2 \), and calculates \( z=\log_{10}(2^A) \). It then sets A equal to ABS INT (Z + 0.5), as required, using FP-TO-A for this purpose.

2DC1 LOG(2^A)
RLA D,A The integer A is stacked, either
SBC A,A as 00 00 A 00 00 (for positive
LD E,A or as 00 FF A FF 00 (for
LD C,A negative A).
LD B,A A, E, D, C, B and then STK-
XOR A STORE is called to put the
LD B,A number on the calculator stack.
CALL 2AB6,STK-STORE
RST 0028,FP-CALC
DEFB +34,stk-data
DEFB +EF,exponent+7F
DEFB +1A,+20,+9A,+85
DEFB +04,multiply
DEFB +27,int
DEFB +38,end-calc

The calculator is used
Log 2 to the base 10 is now stacked.
The stack now holds a, log 2.
A*log 2 i.e. log (2^A)
INT log (2^A)

The subroutine continues on into FP-TO-A to complete the calculation.

THE 'FLOATING-POINT TO A' SUBROUTINE
This short but vital subroutine is called at least 8 times for various purposes. It uses the last but one subroutine, FP-TO-BC, to get the 'last value' into the A register where this is possible. It therefore tests whether the modulus of the number rounds to more than 255 and if it does the subroutine returns with the carry flag set. Otherwise it returns with the modulus of the number, rounded to the nearest integer, in the A register, and the zero flag set to imply that the number was positive, or reset to imply that it was negative.

2DD5 FP-TO-A CALL 2DA2,FP-TO-BC Compress the 'last value' into BC.
RET C Return if out of range already.
PUSH AF Save the result and the flags.
DEC B Again it will be out of range if the B register does not hold zero.
INC B
JR Z,2DE1,FP-A-END Jump if in range.
POP AF Fetch the result and the flags.
SCF Signal the result is out of range.
RET Finished - unsuccessful.

2DE1 FP-A-END POP AF Fetch the result and the flags.
RET Finished - successful.

THE 'PRINT A FLOATING-POINT NUMBER' SUBROUTINE
This subroutine is called by the PRINT command routine at 2039 and by STR$ at 3630, which converts to a string the number as it would be printed. The print format never occupies more than 14 spaces. The 8 most significant digits of x, correctly rounded, are stored in an ad hoc print buffer in mem-3 and mem-4. Small numbers, numerically less than 1, and large numbers, numerically greater than 2^27, are dealt with separately. The former are multiplied by 10^n, where n is the approximate number of leading zeros after the decimal, while the latter are divided by 10^(n-7), where n is the approximate number of digits before the decimal. This brings all numbers into the middle range, and the numbers of digits required before the decimal is built up in the second byte of mem-5. Finally the printing is done, using E-format if there are more than 8 digits before the decimal or, for small numbers, more than 4 leading zeros after the decimal.

The following program shows the range of print formats:
10 FOR a=-11 TO 12: PRINT SGN a*9^a,: NEXT a

i. First the sign of x is taken care of:
   If X is negative, the subroutine jumps to PF-NEGATIVE, takes ABS x and prints the minus sign.
   If x is zero, x is deleted from the calculator stack, a '0' is printed and a return is made from the subroutine.
   If x is positive, the subroutine just continues.

2DE3 PRINT-FP RST 0028,FP-CALC Use the calculator
DEFB +31,duplicate x,x
DEFB +36,less-0 x, (1/0) Logical value of x.
DEFB +00,jump-true x
DEFB +0B,to PF-NEGTEV x
DEFB +31,duplicate x,x
DEFB +37,greater-0 x, (1/0) Logical value of X.
DEFB +00,jump-true x
DEFB +0D, to PF-POSTVE x Hereafter x'=ABS x.
DEFB +02, delete -
DEFB +38, end-calc -
LD A,+30 Enter the character code for '0'.
RST 0010, PRINT-A-1 Print the '0'.
RET
Finished as the 'last value' is zero.

2DF2 PF-NEGTEVE
DEFB +2A, abs x
DEFB +38, end-calc x
LD A,+3D Enter the character code for '.
RST 0010, PRINT-A-1 Print the '.
RST 0028, FP-CALC Use the calculator again.

2DF8 PF-POSTVE
DEFB +A0, stk-zero The 15 bytes of mem-3, mem-4
DEFB +C3, st-mem-3 and mem-5 are now initialised to
DEFB +C4, st-mem-4 zero to be used for a print
DEFB +E2, get-mem-2 The stack is cleared, except for
DEFB +03, delete x'
EXX HL', which is used to hold
PUSH HL calculator offsets, (e.g. for
EXX
'
ii. This is the start of a loop which deals with large numbers. However every number x is first split into its integer part i and the fractional part f. If i is a small integer, i.e. if -65535 <= i <= 65535, it is stored in D'E' for insertion into the print buffer.

2E01 PF-LOOP
RST 0028, FP-CALC Use the calculator again.
DEFB +31, duplicate x x'
DEFB +27, int x', INT (x')=i
DEFB +C2, st-mem-2 (i is stored in mem-2).
DEFB +03, subtract x'-i=f
DEFB +E2, get-mem-2 f,i
DEFB +01, exchange i,f
DEFB +C2, st-mem-2 (f is stored in mem-2).
DEFB +03, delete i
DEFB +38, end-calc
LD A,(HL) Is i a small integer (first byte
AND A zero) i.e. is ABS i <= 65535?
JR NZ, 2E56, PF-LARGE Jump if it is not
CALL 2D7F, INT-FETCH i is copied to DE (i, like x', >=0).
LD B,+10 B is set to count 16 bits.
LD A,D D is copied to A for testing:
AND A Is it zero?
JR NZ, 2E1E, PF-SAVE Jump if it is not zero.
OR E Now test E.
JR Z, 2E24, PF-SMALL Jump if DE zero: x is a pure
fraction.
LD D,E Move E to D and set B for 8
LD B,+08 bits: D was zero and E was not.

2E1E PF-SAVE
PUSH DE Transfer DE to D'E', via the
EXX machine stack, to be moved
POP DE into the print buffer at
EXX
PF-BITS.
JR 2E78, PF-BITS Jump forward.

iii. Pure fractions are multiplied by 10^n, where n is the approximate number of leading zeros after the decimal; and -n is added to the second byte of mem-5, which holds the number of digits needed before the decimals; a negative number here indicates leading zeros after the decimal.

2E24 PF-SMALL
RST 0028, FP-CALC i (i=zero here),
DEFB +E2, get-mem-2 i,f
Note that the stack is now unbalanced. An extra byte 'DEFB +02, delete' is needed at 2E25, immediately after the RST 0028. Now an expression like "2 + STRS 0.5" is evaluated incorrectly as 0.5; the zero left on the stack displaces the "2" and is treated as a null string.

Similarly all the string comparisons can yield incorrect values if the second string takes the form STRS x where x is numerically less than 1; e.g. the expression "50<STRS 0.1" yields the logical value "true"; once again "" is used instead of "50".

LD A,(HL) The exponent byte e of f is copied to A.
SUB +7E A becomes e - 126 dec i.e. e'+2, where e' is the true exponent of f.
CALL 2DC1,LOG (2^A) The construction A = ABS INT (LOG (2^A)) is performed (LOG is to base 10); i.e. A = n, say: n is copied from A to D.
LD D,(mem-5-2nd) The current count is collected from the second byte of mem-5 and n is subtracted from it.
LD D, (mem-5-2nd),A n is copied from D to A.
CALL 2D4F,E-TO-FP y = f * 10^n is formed and stacked.
RST 0028,FP-CALC i, y
DEBF +31, duplicate i, y, y
DEBF +27, int i, y, (INT (y) = 12)
DEBF +C1, st-mem-1 (I2 is copied to mem-1).
DEBF +03, subtract i, y - I2
DEBF +E1, get-mem-1 i, y - I2, I2
DEBF +38, end-calc i, I2, I2 (I2 = y - I2)
CALL 2DD5,FP-TO-A I2 is transferred from the stack to A.
PUSH HL The pointer to I2 is saved.
LD (mem-3-1st),A I2 is stored in the first byte of mem-3: a digit for printing.
DEC A I2 will not count as a digit for printing if it is zero; A is manipulated so that zero will produce zero but a non-zero digit will produce 1.
LD HL,+5CAB The zero or one is inserted into
LD (HL),A the first byte of mem-5 (the no. of digits for printing) and added
INC A to the second byte of mem-5
LD (HL),A (the number of digits before the decimal).
POP HL The pointer to I2 is restored.
JP 2ECF,PF-FRACTN Jump to store I2 in buffer (HL now points to I2, DE to I2).

iv. Numbers greater than $2^{27}$ are similarly multiplied by $2^{(-n+7)}$, reducing the number of digits before the decimal to 8, and the loop is re-entered at PF-LOOP.

2E56 PF-LARGE SUB +80 e - 80 hex = e’, the true exponent of i.
CP +1C Is e’ less than 28 decimal?
JR C,2E6F,PF-MEDIUM Jump if it is less.
CALL 2DC1,LOG (2^A) n is formed in A.
SUB +07 And reduced to n - 7.
LD B,A Then copied to B.
LD HL,+5CAB n - 7 is added in to the second
ADD A,(HL) byte of mem-5, the number of
LD (HL),A digits required before the
The integer part of x is now stored in the print buffer in mem-3 and mem-4.

Note that the case where i us a small integer (less than 65536) re-enters here.

Decimal adjusting each byte of mem-4 gave 2 decimal digits per byte, there being at most 9 digits. The digits will now be re-packed, one to a byte, in mem-3 and mem-4, using the instruction RLD.

Decimal in x. Then i is multiplied by $10^{-(n+7)}$.

This will bring it into medium range for printing.

Round the loop again to deal with the now medium-sized number.

v. The integer part of x is now stored in the print buffer in mem-3 and mem-4.

Note that the case where i us a small integer (less than 65536) re-enters here.

Decimal adjusting each byte of mem-4 gave 2 decimal digits per byte, there being at most 9 digits. The digits will now be re-packed, one to a byte, in mem-3 and mem-4, using the instruction RLD.

Decimal adjusting each byte of mem-4 gave 2 decimal digits per byte, there being at most 9 digits. The digits will now be re-packed, one to a byte, in mem-3 and mem-4, using the instruction RLD.
INC DE
INC (mem-5-1st) Point to next destination.
INC (mem-5-2nd) One more digit for printing, and
LD C,+00 one more before the decimal.

2EB3 PF-TEST-2 BIT 0,B The source pointer needs to be
JR Z,2EB8,PF,ALL-9 incremented on every second
INC HL passage through the loop, when

2EB8 PF-ALL-9 DJNZ 2EA1,PF-DIGITS B is odd.
LD A,(mem-5-1st) Jump back for all 9 digits.
SUB +09 Get counter: were there 9 digits
JR C,2ECB,PF,ALL excluding leading zeros?
DEC (mem-5-1st) If not, jump to get more digits.
LD A,+04 Prepare to round: reduce count
CP (mem-4-4th) to 8.

2ECB PF-MORE DJNZ 2F0C,PF-ROUND Jump forward to round up.
RST 002B,FP-CALC Use the calculator again.
DEFB +02.delete (i is now deleted).
DEB +E2,get-mem-2 f
DEFB +38,end-calc

vi. The fractional part of x is now stored in the print buffer.

2ECF PF-FRACTN EX DE,HL DE now points to f.
CALL 2FBA,FETCH-TWO The mantissa of f is now in
EXX D,E',D,E.
LD A,+80 Get the exchange registers.
SUB L The exponent of f is reduced to
LD L,+00 zero, by shifting the bits of f 80
SET 7,D hex - e places right, where L' contained e.
EXX Restore numerical bit to bit 7 of
CALL 2FDD,SHIFT-FL D'.

2EDF PF-FRN-LP EXX Restore main registers and jump
LP A,(mem-5-1st) forward to round up.
CP +08 Get the digit count.
JR C,2EEC,PR-FR-DGT If not, jump forward.
EXX If 8 digits, just use f to round i
RL D up, rotating D' left to set the
carry.
EXX Restore main registers and jump
CALL 2F0C,PF-ROUND forward to round up.

2EEC PF-FR-DGT JR 2F0C,PF-ROUND LD BC,+0200 Initial zero to C, count of 2 to B.
2EEF PF-FR-EXX CALL 2F8B,CA=10*A+C D'E'D'E is multiplied by 10 in 2
LD E,A stages, first DE then D'E', each
LD A,D byte by byte in 2 steps, and the
CALL 2F68,CA=10*A+C integer part of the result is
LD D,A obtained in C to be passed into
PUSH BC the print buffer.
EXX The count and the result
POPs BC alternate between BC and B'C'.
DJNZ 2EEF,PF-FR-EXX Look back once through the
LD HL,+5CA1 exchange registers.
LD A,C The start - 1st byte of mem-3.
LD C,(mem-5-1st) Result to A for storing.
ADD HL,BC Count of digits so far in number
LD (HL),A to C.
ADD HL,BC Address the first empty byte.
LD (HL),A Store the next digit.
vii. The digits stored in the print buffer are rounded to a maximum of 8 digits for printing.

2F0C PF-ROUND PUSH AF Save the carry flag for the rounding.
LD HL,+5CA1 Base address of number: mem-3, byte 1.
LD C,(mem-5-1st) Offset (number of digits in number) to BC.
ADD HL,BC Address the last byte of the number.
LD B,C Copy C to B as the counter.
POP AF Restore the carry flag.
2F18 PF-RND-LP DEC HL This is the last byte of the number.
LD A,(HL) Get the byte into A.
ADC A,+00 Add in the carry i.e. round up.
LD (HL),A Store the rounded byte in the buffer.
AND A If the byte is 0 or 10, B will be decremented and the final zero (or the 10) will not be counted for printing.
JR Z,2F25,PF-ROUND Jump if carry reset.
2F2D PF-RND-LP LD (mem-5-1st),B B now sets the count of the digits to be printed (final zeros will not be printed).
RST 0028,FP-CALC f is to be deleted.
DEFB +02,delete -
DEFB +38,end-calc -
EXX The calculator offset saved on the stack is restored to H'L'.

viii. The number can now be printed. First C will be set to hold the number of digits to be printed, not counting final zeros, while B will hold the number of digits required before the decimal.

2F46 PF-NOT-E CALL Z,15EF,OUT-CODE Are there no digits before the decimal? If so, print an initial zero.

The next entry point is also used to print the digits needed for E-format printing.
THE 'CA=10*A+C' SUBROUTINE'
This subroutine is called by the PRINT-FP subroutine to multiply each byte of D'E'DE by 10 and return the integer part of the result in the C register. On entry, the A register contains the byte to be multiplied by 10 and the C register contains the carry over from the previous byte. On return, the A register contains the resulting byte and the C register the carry forward to the next byte.
THE 'PREPARE TO ADD' SUBROUTINE.
This subroutine is the first of four subroutines that are used by the main arithmetic operation routines - SUBTRACTION, ADDITION, MULTIPLICATION and DIVISION.
This particular subroutine prepares a floating-point number for addition, mainly by replacing the sign bit with a true numerical bit 1, and negating the number (two's complement) if it is negative. The exponent is returned in the A register and the first byte is set to Hex.00 for a positive number and Hex.FF for a negative number.

THE 'FETCH TWO NUMBERS' SUBROUTINE
This subroutine is called by ADDITION, MULTIPLICATION and DIVISION to get two numbers from the calculator stack and put them into the register, including the exchange registers.
On entry to the subroutine the HL register pair points to the first byte of the first number and the DE register pair points to the first byte of the second number.
When the subroutine is called from MULTIPLICATION or DIVISION the sign of the result is saved in the second byte of the first number.
2FBA FETCH-TWO PUSH HL
    HL is preserved.
PUSH AF
    AF is preserved.

Call the five bytes of the first number - M1, M2, M3, M4 & M5.
and the second number - N1, N2, N3, N4 & N5.

LD C,(HL)    M1 to C.
INC HL       Next.
LD B,(HL)    M2 to B.
LD (HL),A    Copy the sign of the result to
              (HL).
INC HL       Next.
LD A,C       M1 to A.
LD C,(HL)    M3 to C.
PUSH BC      Save M2 & M3 on the machine
             stack.
INC HL       Next.
LD C,(HL)    M4 to C.
INC HL       Next.
LD B,(HL)    M5 to B.
EX DE,HL     HL now points to N1.
LD D,A       M1 to D.
LD E,(HL)    N1 to E.
PUSH DE      Save M1 & N1 on the machine
             stack.
INC HL       Next.
LD D,(HL)    N2 to D.
INC HL       Next.
LD E,(HL)    N3 to E.
PUSH DE      Save N2 & N3 on the machine
             stack.
EXX           Get the exchange registers.
POP DE       N2 to D' & N3 to E'.
POP HL       M1 to H' & N1 to L'.
POP BC       M2 to B' & M3 to C'.
EXX           Get the original set of registers.
INC HL       Next.
LD D,(HL)    N4 to D.
INC HL       Next.
LD E,(HL)    N5 to E.
POP AF       Restore the original AF.
POP HL       Restore the original HL.
RET           Finished.

Summary:
  M1 - M5 are in H', B', C', C, B.
  N1 - N5 are in: L', D', E', D, E.
  HL points to the first byte of the first number.

THE 'SHIFT ADDEND' SUBROUTINE

This subroutine shifts a floating-point number up to 32 decimal, Hex.20, places right to line it up properly for addition. The number with the smaller exponent has been put in the addend position before this subroutine is called. Any overflow to the right, into the carry, is added back into the number. If the exponent difference is greater than 32 decimal, or the carry ripples right back to the beginning of the number then the number is set to zero so that the addition will not alter the other number (the augend).

2FDD SHIFT-FP
    AND A
    If the exponent difference is
    RET Z
    zero, the subroutine returns at
    CP +21
    once. If the difference is greater
    JR NC,2FF9,ADDEND-0
    than Hex.20, jump forward.
PUSH BC
    Save BC briefly.
LD B,A
    Transfer the exponent difference
to B to count the shifts right.

2FE5 ONE-SHIFT
    EXX
    Arithmetic shift right for L',
    SRA L
    preserving the sign marker bits.
THE 'ADD-BACK' SUBROUTINE
This subroutine adds back into the number any carry which has overflowed to the right. In the extreme case, the carry ripples right back to the left of the number. When this subroutine is called during addition, this ripple means that a mantissa of 0.5 was shifted a full 32 places right, and the addend will now be set to zero; when called from MULTIPLICATION, it means that the exponent must be incremented, and this may result in overflow.

3004 ADD-BACK INC E Add carry to rightmost byte.
RET NZ Return if no overflow to left.
INC D Continue to the next byte.
RET NZ Return if no overflow to left.
EXX INC E Increment it too.
JR NZ,300D,ALL-ADDED Jump if no overflow.
INC D Increment the last byte.
300D ALL-ADDED EXX Restore the original registers.
RET Finished.

THE 'SUBTRACTION' OPERATION
(Offset 03 - see CALCULATE below: 'subtract')
This subroutine simply changes the sign of the subtrahend and carried on into ADDITION. Note that HL points to the minuend and DE points to the subtrahend. (See ADDITION for more details.)

300F SUBTRACT EX DE,HL Exchange the pointers.
CALL 346E,NEGATE Change the sign of the subtrahend.
EX DE,HL Exchange the pointers back and continue into ADDITION.

THE 'ADDITION' OPERATION
(Offset 0F - see CALCULATE below: 'addition')
The first of three major arithmetical subroutines, this subroutine carries out the floating-point addition of two numbers, each with a 4-byte mantissa and a 1-byte exponent. In these three subroutines, the two numbers at the top of the calculator stack are added/multiplied/divided to give one number at the top of the calculator stack, a 'last value'.
HL points to the second number from the top, the augend/multiplier/dividend. DE points to the number at the top of the calculator stack, the addend/multiplicand/divisor. Afterwards HL points to the resultant 'last value' whose address can also be considered to be STKEND - 5.

But the addition subroutine first tests whether the 2 numbers to be added are 'small integers'. If they are, it adds them quite simply in HL and BC, and puts the result directly on the stack. No twos complementing is needed before or after the addition, since such numbers are held on the stack in twos complement form, ready for addition.

```assembly
3014 addition LD A,(DE) Test whether the first bytes of both numbers are zero.
OR (HL)  If not, jump for full addition.
JR NZ,303E,FULL-ADDN
PUSH DE Save the pointer to the second number.
INC HL Point to the second byte of the first number and save that pointer too.
PUSH HL Point to the less significant byte.
INC HL Point to the more significant byte.
LD E,(HL) Fetch it in E.
INC HL Point to the less significant byte.
LD D,(HL) Fetch it in D.
INC HL Move on to the second byte of the second number.
INC HL
LD A,(HL) Fetch it in A (this is the sign byte).
INC HL Point to the less significant byte.
LD C,(HL) Fetch it in C.
INC HL Point to the more significant byte.
LD B,(HL) Fetch is in B.
POP HL Fetch the pointer to the sign byte of the first number; put it in DE, and the number in HL.
ADD HL,BC Perform the addition: result in HL.
EX DE,HL Result to DE, sign byte to HL.
ADC A,(HL) Add the sign bytes and the carry into A; this will detect any overflow.
ADC A,+00 A non-zero A now indicates overflow.
JR NZ,303C,ADDN-OFLW Jump to reset the pointers and to do full addition.
SBC A,A Define the correct sign byte for the result.
3032 LD (HL),A Store it on the stack.
INC HL Point to the next location.
LD (HL),E Store the low byte of the result.
INC HL Point to the next location.
LD (HL),D Store the high byte of the result.
DEC HL Move the pointer back to address the first byte of the result.
DEC HL
DEC HL
POP DE Restore STKEND to DE.
RET Finished.
```

Note that the number -65536 decimal can arise here in the form 00 FF 00 00 00 as the result of the addition of two smaller negative integers, e.g. -65000 and -536. It is simply stacked in this form. This is a mistake. The Spectrum system cannot handle this number.
Most functions treat it as zero, and it is printed as -1E-38, obtained by treating it as 'minus zero' in an illegitimate format. One possible remedy would be to test for this number at about byte 3032 and, if it is present, to make the second byte 80 hex and the first byte 91 hex, so producing the full five byte floating-point form of the number, i.e. 91 80 00 00 00, which causes no problems. See also the remarks in 'truncate' below, before byte 3225, and the Appendix.

303C ADDN-OFLW DEC HL Restore the pointer to the first number.
P0P DE Restore the pointer to the second number.
303E FULL-ADDN CALL 3293,RE-ST-TWO Re-stack both numbers in full five byte floating-point form.

The full ADDITION subroutine first calls PREP-ADD for each number, then gets the two numbers from the calculator stack and puts the one with the smaller exponent into the addend position. It then calls SHIFT-FP to shift the addend up to 32 decimal places right to line it up for addition. The actual addition is done in a few bytes, a single shift is made for carry (overflow to the left) if needed, the result is twos complemented if negative, and any arithmetic overflow is reported; otherwise the subroutine jumps to TEST-NORM to normalise the result and return it to the stack with the correct sign bit inserted into the second byte.

EXX Exchange the registers.
PUSH HL Save the next literal address.
EXX Exchange the registers.
PUSH DE Save pointer to the addend.
PUSH HL Save pointer to the augend.
CALL 2F9B,PREP-ADD Prepare the augend.
LD B,A Save its exponent in B.
EX DE,HL Exchange its pointers.
CALL 2F9B,PREP-ADD Prepare the addend.
LD C,A Save its exponent in C.
CP B If the first exponent is smaller, JR NC,3055,SHIFT-LEN keep the first number in the
LD A,B addend position; otherwise
LD B,C change the exponents and the
EX DE,HL pointers back again.
3055 SHIFT-LEN PUSH AF Save the larger exponent in A.
SUB B The difference between the
CALL 2FBA,FETCH-TWO exponents is the length of the
CALL 2FD3,SHIFT-FP shift right.
PUSH HL Save the next literal address.
POP AF Restore the larger exponent.
POP HL Restore pointer to the
LD (HL),A result.
ADD HL,DE Add the two right bytes.
EXX N2 to H' & N3 to L',
EX DE,HL (see FETCH-TWO).
ADC HL,BC Add left bytes with carry.
EX DE,HL Result back in D'E'.
LD A,H Add H', L' and the carry; the
ADC A,L resulting mechanisms will ensure
LD LA that a single shift right is called
RRA if the sum of 2 positive numbers
XCR L has overflowed left, or the sum
EXX of 2 negative numbers has not
overflowed left.
The result is now in DED'E.

Get the pointer to the exponent.

The test for shift (H', L' were Hex. 00 for positive numbers and Hex FF for negative numbers).

A counts a single shift right.

The shift is called.

Add 1 to the exponent; this may lead to arithmetic overflow.

Test for negative result: get sign bit of L' into A (this now correctly indicates the sign of the result).

If it is zero, then do not twos complement the result.

Get the first byte.

Negate it.

Complement the carry for continued negation, and store byte.

Get the next byte.

Ones complement it.

Proceed to get next byte into the A register.

Ones complement it.

Add in the carry for negation.

Store the byte.

Store the last byte.

Ones complement it.

Add in the carry for negation.

Done if no carry.

Else, get .5 into mantissa and add 1 to the exponent; this will be needed when two negative numbers add to give an exact power of 2, and it may lead to arithmetic overflow.

Store the error if required.

Store the last byte.

Clear the carry flag.

Exit via TEST-NORM.

The 'HL=HL*DE' subroutine

This subroutine is called by 'GET-HL*DE' and by 'MULTIPLICATION' to perform the 16-bit multiplication as stated. Any overflow of the 16 bits available is dealt with on return from the subroutine.

BC is saved.

It is to be a 16 bit multiplication.

A holds the high byte.

C holds the low byte.

Initialise the result to zero.

Double the result.
THE 'PREPARE TO MULTIPLY OR DIVIDE' SUBROUTINE
This subroutine prepares a floating-point number for multiplication or division, returning with carry set if the number is zero, getting the sign of the result into the A register, and replacing the sign bit in the number by the true numeric bit, 1.

30C0  PREP-M/D CALL 34E9,TEST-ZERO If the number is zero, return
      RET C with the carry flag set.
      INC HL Point to the sign byte.
      XOR (HL) Get sign for result into A (like signs give plus, unlike give minus); also reset the carry flag.
SET 7,(HL) Set the true numeric bit.
DEC HL Point to the exponent again.
RET Return with carry flag reset.

THE 'MULTIPLICATION' OPERATION
(Offset 04 - see CALCULATE below: 'multiply')
This subroutine first tests whether the two numbers to be multiplied are 'small integers'. If they are, it uses INT-FETCH to get them from the stack, HL=HL*DE to multiply them and INT-STORE to return the result to the stack. Any overflow of this 'short multiplication' (i.e. if the result is not itself a 'small integer') causes a jump to multiplication in full five byte floating-point form (see below).

30CA multiply LD A,(DE) Test whether the first bytes of
      OR (HL) both numbers are zero.
      JR NZ,30F0,MULT-LONG If not, jump for 'long' multipli-
      PUSH DE Save the pointers: to the second number.
      PUSH HL And to the first number.
      PUSH DE And to the second number yet again.
      CALL 2D7F,INT-FETCH Fetch sign in C, number in DE.
      EX DE,HL Number to HL now.
      EX (SP),HL Number to stack, second pointer to HL.
      LD B,C Save first sign in B.
      CALL 2D7F,INT-FETCH Fetch second sign in C, number in DE.
      LD A,B Form sign of result in A: like
      XOR C signs give plus (00), unlike give
      LD C,A minus (FF).
      POP HL Store sign of result in C.
      CALL 30A9,HL=HL*DE Restore the first number to HL.
      PERFORM the actual multipli-
      EX DE,HL cation.
      POP HL Store the result in DE.
      JR C,30EF,MULT-OFLW Restore the pointer to the first
      LD A,D number.
      Jump on overflow to 'full' multiplication.
      These 5 bytes ensure that
00 FF 00 00 00 is replaced by zero; that they should not be
needed if this number were excluded from the system (see
after 303B above).

JR    NZ,30EA,MULT-RSLT
LD    C,A

30EA  MULT-RSLT  CALL  2D8E,INT-STORE  Now store the result on the
POP   DE            stack.
RET   Finished.

30EF  MULT-OFLW  POP   DE  Restore the pointer to the
d second number.

30F0  MULT-LONG  CALL  3293,RE-ST-TWO  Re-stack both numbers in full
five byte floating-point form.

The full MULTIPLICATION subroutine prepares the first number for multiplication by calling PREP-M/D, returning if it is zero; otherwise the second number is prepared by again calling PREP-M/D, and if it is zero the subroutine goes to set the result to zero. Next it fetches the two numbers from the calculator stack and multiplies their mantissas in the usual way, rotating the first number (treated as the multiplier) right and adding in the second number (the multiplicand) to the result whenever the multiplier bit is set. The exponents are then added together and checks are made for overflow and for underflow (giving the result zero). Finally, the result is normalised and returned to the calculator stack with the correct sign bit in the second byte.

XOR    A  A is set to Hex.00 so that the
CALL   30C0,PREP-M/D  sign of the first number will go
RET    C  into A.
EXX
PUSH   HL  Exchange the registers.
EXX
PUSH   DE  Save the next literal address.
            Exchange the registers.
EX
CALL   30C0,PREP-M/D  Prepare the 2nd number.
EX
DE,HL  Exchange the pointers again.
JR    C,315D,ZERO-RSLT  Jump forward if 2nd number is
PUSH   HL  zero.
CALL   2FBA,FETCH-TWO  Get the two numbers from
LD     A,B  the stack.
AND    A  M5 to A (see FETCH-TWO).
SBC    HL,HL  Prepare for a subtraction.
EXX
PUSH   HL  Save the pointers.
SBC    HL,HL  Initialise HL to zero for the
            result.
EXX
PUSH   HL  Exchange the registers.
SBC    HL,HL  Also initialise H'L' for the
            result.
EXX
LD     B,+21  B counts 33 decimal, Hex.21, shifts.
JR    3125,STRT-MLT  Jump forward into the loop.

Now enter the multiplier loop.

3114  MLT-LOOP  JR    NC,311B,NO-ADD  Jump forward to NO-ADD if no
carry, i.e. the multiplier bit was
ADD    HL,DE  reset.
EXX    HL,DE  Else, add the multiplicand in
ADC    HL,DE  D'E'DE (see FETCH-TWO) into
            the result being built up on
Whether multiplicand was added or not, shift result right in \texttt{EXX}.

Shift right the multiplier in \texttt{B'C'CA} (see FETCH-TWO \& above).

A final bit dropping into the carry will trigger another add of the multiplicand to the result.

Loop 33 times to get all the bits.

Move the result from:

\begin{verbatim}
DE,HL
\end{verbatim}

H'L'HL to D'E'DE.

Now add the exponents together.

\begin{verbatim}
BC
HL
A,B
A,C
NZ,313B
A
\end{verbatim}

Prepare to increase the exponent by Hex.80.

These few bytes very cleverly make the correct exponent byte.

Rotating left then right gets the exponent byte (true exponent plus Hex.80) into \texttt{A}.

If the sign flag is reset, no report of arithmetic overflow needed.

Report the overflow if carry reset.

Clear the carry now.

The exponent byte is now complete; but if \texttt{A} is zero a further check for overflow is needed.

If there is no carry set and the result is already in normal form (bit 7 of D' set) then there is overflow to report; but if bit 7 of D' is reset, the result in just in range, i.e. just under $2^{127}$.

Store the exponent byte, at last.

Pass the fifth result byte to \texttt{A} for the normalisation sequence, i.e. the overflow from \texttt{L} into \texttt{B'}.
The remainder of the subroutine deals with normalisation and is common to all the arithmetic routines.

3155 TEST-NORM JR NC,316C,NORMALISE If no carry then normalise now.
LD A,(HL) Else, deal with underflow (zero result) or near underflow
AND A (result 2**-128):
3159 NEAR-ZERO JR Z,315E,Skip-ZERO return exponent to A, test if A
LD A.+80 is zero (case 2**-128) and if so
315D ZERO-RSLT XOR A produce 2**-128 if number
315E SKIP-ZERO CALL 2FFB,ZEROS normal; otherwise produce zero.
LD (HL),A The exponent must then be set
to zero (for zero) or 1 (for
3159 NEAR-ZERO LD A,+80
ZERO) return exponent to A, test if A
LD (HL),A is zero (case 2**-128) and if so
315D SKIP-ZERO JR 3195,OFLOW produce 2**-128 if number
LD (HL),A is normal; otherwise produce zero.
315E CALL 2FFB,ZEROS
The exponent is decremented
on each shift.
316C NORMALISE LD B,+20 Normalise the result by up to 32
definal, Hex.20, shifts left of
316E SHIFT-ONE EXX 7,D D'ED'E'DE (with A adjoined) until
BIT EXX bit 7 of D' is set. A holds zero
JR NZ,3186,NORM-NOW after addition so no precision is
RLCA gained or lost; A holds the fifth
EXX byte from B' after multipli-
RL Dcation or division; but as only
EXX about 32 bits can be correct, no
RL E precision is lost. Note that A is
EXX rotated circularly, with branch
RL D at carry .... eventually a random
EXX process.
DEC (HL) The exponent is decremented
3159 NEAR-ZERO JR Z,3159,NEAR-ZERO on each shift.
DJNZ 316E,SHIFT-ONE If the exponent becomes zero, then number
JR 315D,ZERO-RSLT from 2**-129 are
Loop back, up to 32 times.
3186 NORM-NOW RLA After normalisation add back
JR NC,3195,OFLW-CLR any final carry that went into A.
CALL 3004,ADD-BACK Jump forward if the carry does
JR NZ,3195,OFLW-CLR not ripple right back.
EXX LD D,+80 If it should ripple right back
EXX then set mantissa to 0.5 and
INC (HL) increment the exponent.
3195 OFLOW-CLR PUSH HL This action may lead to arith-
INC HL metic overflow (final case).
EXX The result is moved from its
PUSH DE present registers, D'E' DE, to
EXX BCDE; and then to ACDE.

The actual normalisation operation.

316E SHIFT-ONE LD (HL),A Restore the exponent byte.
JR C,3195,OFLOW-CLR Jump if case 2**-128.
INC HL Otherwise, put zero into second
DEC HL byte of result on the calculator
stack.
JR 3195,OFLOW-CLR Jump forward to transfer the
result.

The final part of the subroutine involves passing the result to the bytes reserved for it on the calculator stack and resetting the pointers.

3195 OFLOW-CLR PUSH HL Save the result pointer.
INC HL Point to the sign byte in the
result.
EXX The result is moved from its
PUSH DE present registers, DE' ED', to
BCDE; and then to ACDE.
The sign bit is retrieved from its temporary store and transferred to its correct position of bit 7 of the first byte of the mantissa.

The first byte is stored.

The second byte is stored.

The third byte is stored.

The fourth byte is stored.

Restore the pointer to the result.

Restore the pointer to second number.

Exchange the register.

Exchange the next literal address.

Exchange the registers.

Finished.

Report 6 - Arithmetic overflow

31AD REPORT-6 RST 0008,ERROR-1 Call the error handling routine.

DEFB +05

THE 'DIVISION' OPERATION

(Offset 05 - see CALCULATE below: 'division')

This subroutine first prepared the divisor by calling PREP-M/D, reporting arithmetic overflow if it is zero; then it prepares the dividend again calling PREP-M/D, returning if it is zero. Next fetches the two numbers from the calculator stack and divides their mantissa by means of the usual restoring division, trial subtracting the divisor from the dividend and restoring if there is carry, otherwise adding 1 to the quotient. The maximum precision is obtained for a 4-byte division, and after subtracting the exponents the subroutine exits by joining the later part of MULTIPLICATION.

31AF division CALL 3293,RE-ST-TWO Use full floating-point forms.
EX DE,HL Exchange the pointers.
XOR A A is set to Hex.00, so that the sign of the first number will go into A.
CALL 30C0,PREP-M/D Prepare the divisor and give the report for arithmetic overflow if it is zero.
JR C,31AD,REPORT-6
EX DE,HL Exchange the pointers.
CALL 30C0,PREP-M/D Prepare the dividend and return
RET C If it is zero (result already zero).
EXX Exchange the registers.
PUSH HL Save the next literal address.
EXX Exchange the registers.
PUSH DE Save pointer to divisor.
PUSH HL Save pointer to dividend.
CALL 2FBA,FETCH-TWO Get the two numbers from the stack.
EXX Exchange the registers.
PUSH HL Save M1 & N1 on the machine stack.
LD H,B Copy the four bytes of the dividend from registers B'C'CB (i.e. M2, M3, M4 & M5; see FETCH-TWO) to the registers HL'HL.
LD L,B
XOR A  Clear A and reset the carry flag.
LD B, +DF  B will count upwards from -33 to -1, twos complement,
           Hex. DF to FF, looping on minus and will jump again on zero for extra precision.
JR 31E2, DIV-START  Jump forward into the division loop for the first trial subtraction.

Now enter the division loop.

31D2 DIV-LOOP  RLA                                                                                   \Shift the result left into B'C'CA,
RL    C                                                                                           \shifting out the bits already
EXX                                                                                             \there, picking up 1 from the
RL    C                                                                                           \carry whenever it is set, and
EXX  B  \rotating left each byte with
EXX  carry to achieve the 32 bit shift.

31DB DIV-34TH  ADD HL, HL  \Move what remains of the dividend left in HL'HL before
EXX  ADC HL, HL  \the next trial subtraction; if a bit drops into the carry, force
EXX  EXX  \no restore and a bit for the quotient, thus retrieving the lost
JR C, 31F2, SUBN-ONLY \bit and allowing a full 32-bit divisor.

31E2 DIV-START  SBC HL, DE  \Trial subtract divisor in D'E'DE
EXX  ADD HL, DE  \from rest of dividend in HL'HL; there is no initial carry (see
EXX  EXX  \previous step).
JR NC, 31F9, NO-RSTORE  \Jump forward if there is no carry.
ADD HL, DE  \Otherwise restore, i.e. add back
EXX  ADC HL, DE  \the divisor. Then clear the carry
EXX  AND A  \so that there will be no bit for
EXX  JR 31FA, COUNT-ONE  \the quotient (the divisor ‘did not go’).
AND A  \Just subtract with no restore
SBC HL, DE  \and go on to set the carry flag
EXX  SBC HL, DE  \because the lost bit of the divisor is to be retrieved and used
EXX  JR 31F2, SUBN-ONLY  \for the quotient.
SCF  \One for the quotient in B'C'CA.
INC B  \Step the loop count up by one.
JP M, 31D2, DIV-LOOP  \Loop 32 times for all bits.
PUSH AF  \Save any 33rd bit for extra
JR Z, 31E2, DIV-START  \Trial subtract yet again for any

Note: This jump is made to the wrong place. No 34th bit will ever be obtained without first shifting the dividend. Hence important results like 1/10 and 1/1000 are not rounded up as they should be. Rounding up never occurs when it depends on the 34th bit. The jump should have been to 31DB DIV-34TH above; i.e. byte 3200 hex in the ROM should read DA hex (128 decimal) instead of E1 hex (225 decimal).

LD E, A  Now move the four bytes that
LD D, C  form the mantissa bytes of the
EXX  result from B'C'CA to D'E'DE.
LD E, C
LD D, B
POP AF  Then put the 34th and 33rd bits
RR B into 'B' to be picked up on normalisation.
POP AF
RR B
EXX
POP BC Restore the exponent bytes, M1 & N1.
POP HL Restore the pointer to the result.
LD A,B Get the difference between the two exponent bytes into A and set the carry flag if required.
JP 313D,DIVN-EXPT Exit via DIVN-EXPT.

THE 'INTEGER TRUNCATION TOWARDS ZERO' SUBROUTINE
(Offset 3A - see CALCULATE below: 'truncate')
This subroutine (say I(x)) returns the result of integer truncation of x, the 'last value', towards zero. Thus I(2.4) is 2 and I(-2.4) is -2. The subroutine returns at once if x is in the form of a 'short integer'. It returns zero if the exponent byte of x if less than 81 hex (ABS x is less than 1). If I(x) is a 'short integer' the subroutine returns it in that form. It returns x if the exponent byte is A0 hex or greater (x has no significant non-integral part). Otherwise the correct number of bytes of x are set to zero and, if needed, one more byte is split with a mask.

3214 truncate LD A,(HL) Get the exponent byte of X into A.
AND A If A is zero, return since x is already a small integer.
RET Z
CP +81 Compare e, the exponent, to 81 hex.
JR NC,3221,T-GR-ZERO Jump if e is greater than 80 hex.
LD (HL),+00 Else, set the exponent to zero;
LD A,+20 enter 32 decimal, 20 hex, into A
JR 3272,NIL-BYTES and jump forward to NIL-BYTES to make all the bits of x be zero.

3221 T-GR-ZERO CP +91 Compare e to 91 hex, 145 decimal.
3223 JR NZ,323F,T-SMALL Jump if e not 91 hex.

The next 26 bytes seem designed to test whether x is in fact -65536 decimal, i.e. 91 80 00 00 00, and if it is, to set it to 00 FF 00 00 00. This is a mistake. As already stated at byte 303B above, the Spectrum system cannot handle this number. The result here is simply to make INT (-65536) return the value -1. This is a pity, since the number would have been perfectly all right if left alone. The remedy would seem to be simply to omit the 28 bytes from 3223 above to 323E inclusive from the program.

3225 INC HL HL is pointed at the fourth byte of x, where the 17 bits of the integer part of x end after the first bit.
INC HL
LD A,+80 The first bit is obtained in A.
AND (HL) using 80 hex as a mask.
DEC HL That bit and the previous 8 bits are tested together for zero.
OR (HL) HL is pointed at the second byte of x.
DEC HL If already non-zero, the test can end.
JR NZ,3233,T-FIRST
LD A,+80 Otherwise, the test for -65536 is now completed: 91 80 00 00 00 will leave the zero flag set now.
3233 T-FIRST DEC HL HL is pointed at the first byte of x.
JR NZ,326C,T-EXPNENT If zero reset, the jump is made.
LD (HL),A The first byte is set to zero.
INC HL HL points to the second byte.
LD (HL),+FF  The second byte is set to FF.
DEC HL  HL again points to the first byte.
LD A,+18  The last 24 bits are to be zero.
JR 3272:NIL-BYTES  The jump to NIL-BYTES completes the number 00 FF 00 00 00.

If the exponent byte of x is between 81 and 90 hex (129 and 144 decimal) inclusive, I(x) is a ‘small integer’, and will be compressed into one or two bytes. But first a test is made to see whether x is, after all, large.

323F  T-SMALL  JR NC,326D,X-LARGE  Jump with exponent byte 92 or more (it would be better to jump with 91 too).
    PUSH DE  Save STKEND in DE.
    CPL  Range 129 <= A <= 144 becomes 126 >= A >= 111.
    ADD A,+91  Range is now 15 dec >= A >= 0.
    INC HL  Point HL at second byte.
    LD D,(HL)  Second byte to D.
    INC HL  Point HL at third byte.
    LD E,(HL)  Third byte to E.
    DEC HL  Point HL at first byte again.
    LD C,+00  Assume a positive number.
    BIT 7,D  Now test for negative (bit 7 set).
    JR Z,3252,T  Jump if positive after all.
    JR NC,326D,X  Jump with exponent byte 92 or more (it would be better to jump with 91 too).
3252  T-NUMERIC  SET 7,D  Insert true numeric bit, 1, in D.
    LD B,+08  Now test whether A >= 8 (one byte only) or two bytes needed.
    SUB B  Leave A unchanged.
    ADD A,B  Leave A unchanged.
    JR C,325E,T-TEST  Jump if two bytes needed.
    LD E,D  Put the one byte into E.
    LD D,+00  And set D to zero.
    SUB B  Now 1 <= A <= 7 to count the shifts needed.
    JR Z,3252,T-NUMERIC  Jump if positive after all.
    JR NC,326D,X  Jump with exponent byte 92 or more (it would be better to jump with 91 too).
325E  T-TEST  JR Z,3267,T-STORE  Jump if no shift needed.
    LD B,A  B will count the shifts.
3261  T-SHIFT  SRL D  Shift D and E right B times to produce the correct number.
    RR E  Loop until B is zero.
3267  T-STORE  CALL 2D8E,INT-STORE  Store the result on the stack.
    POP DE  Restore STKEND to DE.
    RET  Finished.

Large values of x remains to be considered.

326C  T-EXPONENT  LD A,(HL)  Get the exponent byte of x into A.
326D  X-LARGE  SUB +A0  Subtract 160 decimal, A0 hex, from e.
    RET  Return on plus - x has no significant non-integral part. (If the true exponent were reduced to zero, the ‘binary point’ would come at or after the end of the four bytes of the mantissa).
    NEG  Else, negate the remainder; this gives the number of bits to become zero (the number of bits after the ‘binary point’).
Now the bits of the mantissa can be cleared.

3272 NIL-BYTES PUSH DE
3273 EX DE,HL
3274 DEC HL
3275 LD B,A
3276 SRL B
3277 SRL B
3278 JR Z,3283,BITS-ZERO

Save the current value of DE (STKEND).
Make HL point one past the fifth byte.
HL now points to the fifth byte of x.
Get the number of bits to be set to zero in B and divide it by B to give the number of whole bytes implied.
Jump forward if the result is zero.

327E BYTE-ZERO LD (HL),+00
327F DEC HL
3280 DJNZ 327E,BYTE-ZERO

Else, set the bytes to zero; B counts them.

3283 BITS-ZERO AND +07
3284 JR Z,3290,IX-END

Get A (mod 8); this is the number of bits still to be set to zero.
Jump to the end if nothing more to do.

328A LESS-MASK SLA A
328B DJNZ 328A,LESS-MASK

With each loop a zero enters the mask from the right and thereby a mask of the correct length is produced.

328E HL,+00
328F DEC HL
3290 LD (HL),A

The unwanted bits of (HL) are lost as the masking is performed.

3291 IX-END EX DE,HL
3292 POP DE
3293 RET

Return the pointer to HL.
Return STKEND to DE.
Finished.

THE 'RE-STACK TWO' SUBROUTINE
This subroutine is called to re-stack two 'small integers' in full five byte floating-point form for the binary operations of addition, multiplication and division. It does so by calling the following subroutine twice.

3295 RE-ST-TWO CALL 3269,RESTK-SUB
3296 RESTK-SUB EX DE,HL

Call the subroutine, and then continue into it for the second call.
Exchange the pointers at each call.

THE 'RE-STACK TWO' SUBROUTINE
(Offset 3D - see CALCULATE below: 're-stack')
This subroutine is called to re-stack one number (which could be a 'small integer') in full five byte floating-point form. It is used for a single number by ARCTAN and also, through the calculator offset, by EXP, LN and 'get-argt'.

3299 RE-STACK LD A,(HL)
329A AND A
329B RET NZ
329C PUSH DE CALL 2D7F,INT-FETCH
329D XOR A INC HL
329E LD (HL),A DEC HL
329F LD (HL),A

If the first byte is not zero, return - the number cannot be a 'small integer', Save the 'other' pointer in DE.
Fetch the sign in C and the number in DE.
Clear the A register.
Point to the fifth location.
Set the fifth byte to zero.
Point to the fourth location.
Set the fourth byte to zero: bytes 2 and 3 will hold the mantissa.
LD B,+91
LD A,D
AND A
JR NZ,32B1,RS-NRMLSE
OR E
LD B,D
JR Z,32BD,RS-STORE
LD D,E
LD E,B
LD B,+89
JR NZ,32B1,RS-STORE
EX DE,HL
RSTK LOOP
ADD HL,HL
JR NC,32B2,RSTK-LOOP
RRC C
RR H
RR L
EX DE,HL
DEC B
ADD HL,HL
JR NC,32B2,RSTK-LOOP
RRC C
RR H
RR L
EX DE,HL
DEC HL
LD (HL),E
DEC HL
LD (HL),D
DEC HL
LD (HL),B
POP DE
RET

LD A,D
Test whether D is zero so that at most 8 bits would be needed.
LD B,D
Save the zero in B (it will give zero exponent if E too is zero).
JR Z,32BD,RS-STORE
Jump if E is indeed zero.
LD D,E
Move E to D (D was zero, E not).
LD E,B
Set E to zero now.
LD B,+89
Set B to 137 dec for the exponent - no more than 8 bits now.

32B1 RS-NRMLSE EX DE,HL Pointer to DE, number to HL.
32B2 RSTK LOOP DEC B Decrement the exponent on each shift.
ADD HL,HL Shift the number right one position.
JR NC,32B2,RSTK-LOOP Until the carry is set.
RRC C Sign bit to carry flag now.
RR H Insert it in place as the number is shifted back one place - normal now.
RR L
EX DE,HL Pointer to byte 4 back to HL.
DEC HL Point to the third location.
LD (HL),E Store the third byte.
DEC HL Point to the second location.
LD (HL),D Store the second byte.
DEC HL Point to the first location.
LD (HL),B Store the exponent byte.
POP DE Restore the ‘other’ pointer to DE.
RET Finished.
THE FLOATING-POINT CALCULATOR

THE TABLE OF CONSTANTS
This first table holds the five useful and frequently needed numbers zero, one, a half, a half of pi and ten. The numbers are held in a condensed form which is expanded by the STACK LITERALS subroutine, see below, to give the required floating-point form.

<table>
<thead>
<tr>
<th>data:</th>
<th>constant</th>
<th>when expanded gives: exp. mantissa: (Hex.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>stk-zero</td>
<td>DEFB +00</td>
<td>zero</td>
</tr>
<tr>
<td></td>
<td>DEFB +B0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEFB +00</td>
<td></td>
</tr>
<tr>
<td>stk-one</td>
<td>DEFB +40</td>
<td>one</td>
</tr>
<tr>
<td></td>
<td>DEFB +B0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEFB +00</td>
<td></td>
</tr>
<tr>
<td>stk-half</td>
<td>DEFB +30</td>
<td>a half</td>
</tr>
<tr>
<td></td>
<td>DEFB +00</td>
<td></td>
</tr>
<tr>
<td>stk-pi/2</td>
<td>DEFB +F1</td>
<td>a half of pi</td>
</tr>
<tr>
<td></td>
<td>DEFB +49</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEFB +0F</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEFB +DA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEFB +A2</td>
<td></td>
</tr>
<tr>
<td>stk-ten</td>
<td>DEFB +40</td>
<td>ten</td>
</tr>
<tr>
<td></td>
<td>DEFB +B0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEFB +00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEFB +0A</td>
<td></td>
</tr>
</tbody>
</table>

THE TABLE OF ADDRESSES:
This second table is a look-up table of the addresses of the sixty-six operational subroutines of the calculator. The offsets used to index into the table are derived either from the operation codes used in SCANNING, see 2734, etc., or from the literals that follow a RST 0028 instruction.

<table>
<thead>
<tr>
<th>offset</th>
<th>label</th>
<th>address</th>
<th>offset</th>
<th>label</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>32D7</td>
<td>00</td>
<td>jump-true 8F</td>
<td>32D9</td>
<td>01</td>
<td>exchange 3C 34</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Note: The last four subroutines are multi-purpose subroutines and are entered with a parameter that is a copy of the right hand five bits of the original literal. The full set follows:

Offset 3E: series-06, series-08, & series-0C; literals 86,88 & 8C.
Offset 3F: stk-zero, stk-one, stk-half, stk-pi/2 & stk-len; literals A0 to A4.
Offset 41: get-mem-0, get-mem-1, get-mem-2, get-mem-3, get-mem-4 & get-mem-5; literals E0 to E5.
THE 'CALCULATE' SUBROUTINE

This subroutine is used to perform floating-point calculations. These can be considered to be of three types:

I. Binary operations, e.g. addition, where two numbers in floating-point form are added together to give one 'last value'.

II. Unary operations, e.g. sin, where the 'last value' is changed to give the appropriate function result as a new 'last value'.

III. Manipulatory operations, e.g. st-mem-0, where the 'last value' is copied to the first five bytes of the calculator's memory area.

The operations to be performed are specified as a series of data-bytes, the literals, that follow an RST 0028 instruction that calls this subroutine. The last literal in the list is always '38' which leads to an end to the whole operation.

In the case of a single operation needing to be performed, the operation offset can be passed to the CALCULATOR in the B register, and operation '3B', the SINGLE CALCULATION operation, performed.

It is also possible to call this subroutine recursively, i.e. from within itself, and in such a case it is possible to use the system variable BREG as a counter that controls how many operations are performed before returning.

The first part of this subroutine is complicated but essentially it performs the two tasks of setting the registers to hold their required values, and to produce an offset, and possibly a parameter, from the literal that is currently being considered.

Note: A floating-point number may in reality be a set of string parameters.

Presume a unary operation and therefore set HL to point to the start of the 'last value' on the calculator stack and DE one past this floating-point number (STKEND).

Either, transfer a single operation offset to BREG temporarily, or, when using the subroutine recursively pass the parameter to BREG to be used as a counter.

The return address of the subroutine is store in HL'. This saves the pointer to the first literal. Entering the CALCULATOR at GEN-ENT-2 is used whenever BREG is in use as a counter and is not to be disturbed.

A loop is now entered to handle each literal in the list that follows the calling instruction; so first, always set to STKEND. Go to the alternate register set, and fetch the literal for this loop.
INC HL

Make HL point to the next literal.

336C SCAN-ENT PUSH HL

This pointer is saved briefly on the machine stack. SCAN-ENT is used by the SINGLE CALCULATION subroutine to find the subroutine that is required.

AND A

Test the A register.

JP P, 3380, FIRST-3D

Separate the simple literals from the multi-purpose literals. Jump with literals 00 - 3D.

LD D, A

Save the literal in D.

AND +60

Continue only with bits 5 & 6.

RRCA

Four right shifts make them now bits 1 & 2.

RRCA

ADD A, +7C

The offsets required are 3E - 41.

LD L, A

and L will now hold double the required offset.

LD A, D

Now produce the parameter by taking bits 0, 1, 2, 3 & 4 of the literal; keep the parameter in A.

JR 338E, ENT-TABLE

Jump forward to find the address of the required subroutine.

3380 FIRST-3D CP +18

Jump forward if performing a unary operation.

JR NC, 338C, DOUBLE-A

All of the subroutines that perform binary operations require that HL points to the first operand and DE points to the second operand (the ‘last value’) as they appear on the calculator stack.

EXX

LD BC, +FFFB

(See COMPARISON at 353B)

LD D, H

LD E, L

ADD HL, BC

LD A, +7C

The offsets required are 3E - 41.

LD L, A

and L will now hold double the required offset.

LD A, D

Now produce the parameter by taking bits 0, 1, 2, 3 & 4 of the literal; keep the parameter in A.

JR 338E, ENT-TABLE

Jump forward to find the address of the required subroutine.

3380 FIRST-3D CP +18

Jump forward if performing a unary operation.

JR NC, 338C, DOUBLE-A

All of the subroutines that perform binary operations require that HL points to the first operand and DE points to the second operand (the ‘last value’) as they appear on the calculator stack.

EXX

LD BC, +FFFB

LD D, H

LD E, L

ADD HL, BC

LD A, +7C

The offsets required are 3E - 41.

LD L, A

and L will now hold double the required offset.

LD A, D

Now produce the parameter by taking bits 0, 1, 2, 3 & 4 of the literal; keep the parameter in A.

JR 338E, ENT-TABLE

Jump forward to find the address of the required subroutine.

338C DOUBLE-A RLCA

As each entry in the table of addresses takes up two bytes the offset produced is doubled.

LD L, A

338E ENT-TABLE LD DE, +32D7

The base address of the table.

LD H, +00

The address of the required table entry is formed in HL; and the required subroutine address is loaded into the DE register pair.

ADD HL, DE

LD E, (HL)

INC HL

LD D, (HL)

LD HL, +3365

EX (SP), HL

PUSH DE

EXX

Return to the main set of registers.

LD BC, (STKEND - hi)

The current value of BREG is transferred to the B register thereby returning the single operation offset.

(See COMPARISON at 353B)

33A1 delete RET

An indirect jump to the required subroutine.
THE 'DELETE' SUBROUTINE
(Offset 02: 'delete')

This subroutine contains only the single RET instruction at 33A1, above. The literal '02' results in this subroutine being considered as a binary operation that is to be entered with a first number addressed by the HL register pair and a second number addressed by the DE register pair, and the result produced again addressed by the HL register pair.

The single RET instruction thereby leads to the first number being considered as the resulting 'last value' and the second number considered as being deleted. Of course the number has not been deleted from the memory but remains inactive and will probably soon be overwritten.

THE 'SINGLE OPERATION' SUBROUTINE
(Offset 3B: 'fp-calc-2')

This subroutine is only called from SCANNING at 2757 hex and is used to perform a single arithmetic operation. The offset that specifies which operation is to be performed is supplied to the calculator in the B register and subsequently transferred to the system variable BREG.

The effect of calling this subroutine is essentially to make a jump to the appropriate subroutine for the single operation.

```
33A2  fp-calc-2    POP       AF       Discard the RE-ENTRY address.
      LD        A,(BREG)    Transfer the offset to A.
      EXX       Enter the alternate register set.
      JR        336C,SCAN-ENT Jump back to find the required address; stack
                                 the RE-ENTRY address and jump to the
                                 subroutine for the operation.
```

THE 'TEST 5-SPACES' SUBROUTINE

This subroutine tests whether there is sufficient room in memory for another 5-byte floating-point number to be added to the calculator stack.

```
33A9  TEST-5-SP    PUSH      DE       Save DE briefly.
      PUSH      HL       Save HL briefly.
      LD        BC,+0005  Specify the test is for 5 bytes.
      CALL      1F05,TEST-ROOM Make the test.
      POP       HL       Restore HL.
      POP       DE       Restore DE.
      RET       Finished.
```

THE 'STACK NUMBER' SUBROUTINE

This subroutine is called by BEEP and SCANNING twice to copy STKEND to DE, move a floating-point number to the calculator stack, and reset STKEND from DE. It calls 'MOVE-FP' to do the actual move.

```
33B4  STACK-NUM    LD        DE,(STKEND) Copy STKEND to DE as
data destination address.
      CALL      33C0,MOVE-FP Move the number.
      LD        (STKEND),DE  Reset STKEND from DE.
      RET       Finished.
```
THE 'MOVE A FLOATING-POINT NUMBER' SUBROUTINE
(Offset 31: 'duplicate')

This subroutine moves a floating-point number to the top of the calculator stack (3 cases) or from the top of the stack to the calculator's memory area (1 case). It is also called through the calculator when it simply duplicates the number at the top of the calculator stack, the 'last value', thereby extending the stack by five bytes.

```
33C0  MOVE-FP  CALL  33A9,TEST-5-SP  A test is made for room.
       LDIR               Move the five bytes involved.
       RET               Finished.
```

THE 'STACK LITERALS' SUBROUTINE
(Offset 34: 'stk-data')

This subroutine places on the calculator stack, as a 'last value', the floating-point number supplied to it as 2, 3, 4 or 5 literals. When called by using offset '34' the literals follow the '34' in the list of literals; when called by the SERIES GENERATOR, see below, the literals are supplied by the sub-routine that called for a series to be generated; and when called by SKIP CONSTANTS & STACK A CONSTANT the literals are obtained from the calculator's table of constants (32C5-32D6).

In each case, the first literal supplied is divided by Hex.40, and the integer quotient plus 1 determines whether 1, 2, 3 or 4 further literals will be taken from the source to form the mantissa of the number. Any unfilled bytes of the five bytes that go to form a 5-byte floating-point number are set to zero. The first literal is also used to determine the exponent, after reducing mod Hex.40, unless the remainder is zero, in which case the second literal is used, as it stands, without reducing mod Hex.40. In either case, Hex.50 is added to the literal, giving the augmented exponent byte, e (the true exponent e' plus Hex.80). The rest of the 5 bytes are stacked, including any zeros needed, and the subroutine returns.

```
33C6  STK-DATA  LD  H,D                      This subroutine performs the
       LD  L,E                      manipulatory operation of
       This subroutine performs the
       adding a 'last value' to the cal-
      culator stack; hence HL is set to
       point one-past the present
       'last value' and hence point to
       the result.
33C8  STK-CONST CALL  33A9,TEST-5-SP  Now test that there is indeed
       room.
       EXX                     Go to the alternate register set
       PUSH                    and stack the pointer to the
       EXX                     next literal.
       EX                      Switch over the result pointer
       (SP),HL                  and the next literal pointer.
       PUSH                    Save BC briefly.
       BC                      The first literal is put into A
       LD  A,(HL)               and divided by Hex.40 to give
       AND                     the integer values 0, 1, 2 or 3.
       +C0                     The integer value is transferred
       RLCA                    to C and incremented, thereby
       LD  C,A                  giving the range 1, 2, 3 or 4
       INC                     for the number of literals that will
       C                       be needed.
       A,(HL)                  The literal is fetched anew,
       LD                     reduced mod Hex.40 and dis-
       AND +3F                  carded as inappropriate if the
       JR  NZ,33DE,F-E          remainder is zero; in which case
       INC  HL                  Finished.
```
LD A,(HL) the next literal is fetched and used unreduced.

33DE FORM-EXP ADD A,+50 The exponent, e, is formed by the addition of Hex.50 and passed to the calculator stack as the first of the five bytes of the result.
LD (DE),A

LD A,+05 The number of literals specified in C are taken from the source and entered into the bytes of the result.
SUB C
INC HL
INC DE
LD B,+00
LDIR
POP BC Restore BC.
EX (SP),HL Return the result pointer to HL its usual position in H' & L'.
EXX
LD B,A The number of zero bytes required at this stage is given by
33F1 STK-ZEROS XOR A 5-C-1; and this number of zeros is added to the result to make up the required five bytes.
LD (DE),A
INC DE
JR 33F1, STK-ZEROS

THE 'SKIP CONSTANTS' SUBROUTINE
This subroutine is entered with the HL register pair holding the base address of the calculator's table of constants and the A register holding a parameter that shows which of the five constants is being requested. The subroutine performs the null operations of loading the five bytes of each unwanted constant into the locations 0000, 0001, 0002, 0003 and 0004 at the beginning of the ROM until the requested constant is reached. The subroutine returns with the HL register pair holding the base address of the requested constant within the table of constants.

33F7 SKIP-CONS AND A The subroutine returns if the parameter is zero, or when the requested constant has been reached.
33F8 SKIP-NEXT RET Z

PUSH AF Save the parameter.
PUSH DE Save the result pointer.
LD DE,+0000 The dummy address.
CALL 33C8,STK-CONST Perform imaginary stacking of an expanded constant.
POP DE Restore the result pointer.
POP AF Restore the parameter.
DEC A Count the loops.
JR 33F8,SKIP-NEXT Jump back to consider the value of the counter.

THE 'MEMORY LOCATION' SUBROUTINE
This subroutine finds the base address for each five byte portion of the calculator's memory area to or from which a floating-point number is to be moved from or to the calculator stack. It does this operation by adding five times the parameter supplied to the base address for the area which is held in the HL register pair. Note that when a FOR-NEXT variable is being handled then the pointers are changed so that the variable is treated as if it were the calculator's memory area (see address 1D20).

9406 LOC-MEM LD C,A Copy the parameter to C.
RLCA Double the parameter.
RLCA Double the result.
ADD A,C Add the value of the parameter to give five times the original
THE 'GET FROM MEMORY AREA' SUBROUTINE
(Offsets E0 to E5: 'get-mem-0' to 'get-mem-5')

This subroutine is called using the literals E0 to E5 and the parameter derived from these literals is held in the A register. The subroutine calls MEMORY LOCATION to put the required source address into the HL register pair and MOVE A FLOATING-POINT NUMBER to copy the five bytes involved from the calculator's memory area to the top of the calculator stack to form a new 'last value'.

```
340F get-mem-0 etc.
PUSH DE Save the result pointer.
LD HL,(MEM) Fetch the pointer to the current memory area (see above).
CALL 3406,LOC-MEM The base address is found.
CALL 33C0,MOVE-FP The five bytes are moved.
POP HL Set the result pointer.
RET Finished.
```

THE 'STACK A CONSTANT' SUBROUTINE
(Offsets A0 to A4: 'stk-zero','stk-one','stk-half','stk-pi/2' & 'stk-ten')

This subroutine uses SKIP CONSTANTS to find the base address of the requested constants from the calculator's table of constants and then calls STACK LITERALS, entering at STK-CONST, to make the expanded form of the constant the 'last value' on the calculator stack.

```
341B stk-zero etc.
LD H,D Set HL to hold the result pointer.
LD L,E Go to the alternate register set and save the next literal pointer.
EX DE,HL Source to DE briefly.
PUSH HL The base address of the calculator's table of constants.
LD HL,+32C5 Back to the main set of registers.
EXX EX DE,HL 'Last value' +5, i.e. STKEND, to DE.
EXX CALL 33F7,SKIP-CONS Find the requested base address.
EXX CALL 33C8,STK-CONST Expand the constant.
PO\nHL Restore the next literal pointer.
EXX POP HL Result pointer to HL.
RET Finished.
```

THE 'STORE IN MEMORY AREA' SUBROUTINE
(Offsets C0 to C5: 'st-mem-0' to 'st-mem-5')

This subroutine is called using the literals C0 to C5 and the parameter derived from these literals is held in the A register. This subroutine is very similar to the GET FROM MEMORY subroutine but the source and destination pointers are exchanged.

```
342D st-mem-0 etc.
PUSH HL Save the result pointer.
EX DE,HL Source to DE briefly.
LD HL,(MEM) Fetch the pointer to the current memory area.
CALL 3406,LOC-MEM The base address is found.
EX CALL 3406,LOC-MEM The base address is found.
EX CALL 33C0,MOVE-FP The five bytes are moved.
EX CALL 33C0,MOVE-FP 'Last value' +5, i.e. STKEND, to DE.
POP HL Result pointer to HL.
RET Finished.
```

197
Note that the pointers HL and DE remain as they were, pointing to STKEND-5 and STKEND respectively, so that the 'last value' remains on the calculator stack. If required it can be removed by using 'delete'.

THE 'EXCHANGE' SUBROUTINE
(Offset 01: 'exchange')

This binary operation 'exchanges' the first number with the second number, i.e. the topmost two numbers on the calculator stack are exchanged.

343C EXCHANGE LD B,+05 There are five bytes involved.
343E SWAP-BYTE LD A,(DE) Each byte of the second number.
LD C,(HL) Each byte of the first number.
EX DE,HL Switch source and destination.
LD (DE),A Now to the first number.
LD (HL),C Now to the second number.
INC HL Move to consider the next pair
INC DE of bytes.
DJNZ 343E,SWAP Exchange the five bytes.
EX DE,HL Get the pointers correct as the number 5 is an odd number.
RET Finished.

THE 'SERIES GENERATOR' SUBROUTINE
(Offsets 86,88 & 8C: 'series-06','series-08' & 'series-0C')

This important subroutine generates the series of Chebyshev polynomials which are used to approximate to SIN, ATN, LN and EXP and hence to derive the other arithmetic functions which depend on these (COS, TAN, ASN, ACS, ** and SQR).

The polynomials are generated, for n=1,2,..., by the recurrence relation:

\[ T_{n+1}(z) = 2zT_n(z) - T_{n-1}(z), \]

where \( T_n(z) \) is the nth Chebyshev polynomial in \( z \).

The series in fact generates:

\[ T_0, 2T_1, 2T_2, \ldots, 2T_{n-1}, \]

where \( n \) is 6 for SIN, 8 for EXP and 12 decimal, for LN and ATN.

The coefficients of the powers of \( z \) in these polynomials may be found in the Handbook of Mathematical Functions by M. Abramowitz and I.A. Stegun (Dover 1965), page 795.

BASIC programs showing the generation of each of the four functions are given here in the Appendix.

In simple terms this subroutine is called with the 'last value' on the calculator stack, say Z, being a number that bears a simple relationship to the argument, say X, when the task is to evaluate, for instance, SIN X. The calling subroutine also supplies the list of constants that are to be required (six constants for SIN). The SERIES GENERATOR then manipulates its data and returns to the calling routine a 'last value' that bears a simple relationship to the requested function, for instance, SIN X.

This subroutine can be considered to have four major parts:

i. The setting of the loop counter:

The calling subroutine passes its parameters in the A register for use as a counter. The calculator is entered at GEN-ENT-1 so that the counter can be set.

3449 series-06 etc. LD B.A Move the parameter to B.
CALL 335E,GEN-ENT-1 In effect a RST 0028 instruction but sets the counter.

ii. The handling of the 'last value', Z:

The loop of the generator requires \( 2^2Z \) to be placed in mem-0, zero to be placed in mem-2 and the 'last value' to be zero.

 calculator stack
DEFB +31,duplicate Z.Z
DEFB +0F,addition 2^2Z
DEFB +C0,store-mem-0 2^2Z mem-0 holds 2^2Z
DEFB +02,delete -
iii. The main loop:

The main loop is generated by looping, using BREG as a counter; the constants in the calling subroutine are stacked in turn by calling STK-DATA; the calculator is re-entered at GEN-ENT-2 so as not to disturb the value of BREG; and the series is built up in the form:

\[ B(R) = 2^R B(R-1) - B(R-2) + A(R), \]

for \( R = 1, 2, \ldots, N \), where \( A(1), A(2), \ldots, A(N) \) are the constants supplied by the calling subroutine (\( \sin, \\tan, \ln \) and \( \exp \)) and \( B(0) = 0 = B(-1) \).

The (R+1)th loop starts with \( B(R) \) on the stack and with \( 2^R Z, B(R-2) \) and \( B(R-1) \) in mem-0, mem-1 and mem-2 respectively.

```
3453 G-LOOP
DEFB +31, duplicate B(R), B(R)
DEFB +06, get-mem-0 B(R), B(R), 2*R
DEFB +04, multiply B(R), 2*B(R)*Z
DEFB +E2, get-mem-2 B(R), 2*B(R)*Z, B(R-1)
DEFB +C1, st-mem-1 mem-1 holds B(R-1)
```

The next constant is placed on the calculator stack.

```
CALL 33C6, STK-DATA B(R), 2*B(R)*Z - B(R-1), A(R+1)
```

The Calculator is re-entered without disturbing BREG.

```
CALL 3362, GEN-ENT-2
DEFB +0F, addition B(R), 2*B(R)*Z - B(R-1) + A(R+1)
DEFB +C2, st-mem-2 mem-2 holds B(R)
DEFB +02, delete 2*B(R)*Z - B(R-1) + A(R+1) = B(R+1)
DEFB +35, dec-jr-nz B(R+1)
DEFB +EE, to 3453, G-LOOP
```

iv. The subtraction of \( B(N-2) \):

The loop above leaves \( B(N) \) on the stack and the required result is given by \( B(N) - B(N-2) \).

```
3453 G-LOOP
DEFB +31, duplicate B(R), B(R)
DEFB +00, get-mem-0 B(R), B(R), 2*R
DEFB +04, multiply B(R), 2*B(R)*Z
DEFB +E2, get-mem-2 B(R), 2*B(R)*Z, B(R-1)
DEFB +C1, st-mem-1 mem-1 holds B(R-1)
```

```
DEFB +03, subtract B(R), 2*B(R)*Z - B(R-1) + A(R+1)
DEFB +01, exchange 2*B(R)*Z - B(R-1) + A(R+1), B(R)
DEFB +C2, st-mem-2 mem-2 holds B(R)
DEFB +02, delete 2*B(R)*Z - B(R-1) + A(R+1) = B(R+1)
DEFB +35, dec-jr-nz B(R+1)
DEFB +EE, to 3453, G-LOOP
```

THE 'ABSOLUTE MAGNITUDE' FUNCTION

(Offset 2A: 'abs')
This subroutine performs its unary operation by ensuring that the sign bit of a floating-point number is reset.

'Small integers' have to be treated separately. Most of the work is shared with the 'unary minus' operation.

```
346A abs LD B, +FF B is set to FF hex.
JR 3474, NEG-TEST The jump is made into 'unary minus'.
```

THE 'UNARY MINUS' OPERATION

(Offset 1B: 'negate')
This subroutine performs its unary operation by changing the sign of the 'last value' on the calculator stack.

Zero is simply returned unchanged. Full five byte floating-point numbers have their sign bit manipulated so that it ends up reset (for 'abs') or changed (for 'negate'). 'Small integers' have their sign byte set to zero (for 'abs') or changed (for 'negate').

```
346E NEGATE CALL 34E9, TEST-ZERO If the number is zero, the
RET C subroutine returns leaving
00 00 00 00 00 unchanged.
```
LD B,+00

B is set to +00 hex for 'negate'.

'ABS' enters here.

3474 NEG-TEST
LD A,(HL) If the first byte is zero, the
AND A jump is made to deal with a
JR Z,3483,INT-CASE 'small integer'.
INC HL Point to the second byte.
LD A,B Get +FF for 'abs', +00 for
AND +80 'negate'.
OR (HL) Now +80 for 'abs', +00 for
RLA 'negate'.
CCF This sets bit 7 for 'abs', but
RRA changes nothing for 'negate'.
LD (HL),A The new second byte is stored.
DEC HL HL points to the first byte
RET again.

The 'integer case' does a similar operation with the sign byte.

3483 INT-CASE
PUSH DE Save STKEND in DE.
PUSH HL Save pointer to the number in
CALL 2D7F,INT-FETCH HL.
POP HL Restore the pointer to the
LD A,B number in HL.
OR C Get +FF for 'abs', +00 for
CPL 'negate'. Now +FF for 'abs', no change for
LD C,A 'negate'.
CALL 2D8E,INT-STORE Store result on the stack.
POP DE Return STKEND to DE.
RET

THE 'SIGNUM' FUNCTION
(Offset 29: 'sgn')

This subroutine handles the function SGN X and therefore returns a 'last value' of 1 if X is positive, zero if X is zero and -1 if X is negative.

3492 sgn
CALL 34E9,TEST-ZERO
RET C If X is zero, just return with
PUSH DE zero as the 'last value'.
LD DE,+0001 Save the pointer to STKEND.
INC HL Store 1 in DE.
RL (HL) Point to the second byte of X.
DEC HL Rotate bit 7 into the carry flag.
SBC A,A Point to the destination again.
LD C,A Set C to zero for positive X and
to FF hex for negative X.
CALL 2D8E,INT-STORE Stack 1 or -1 as required.
POP DE Restore the pointer to
RET STKEND.

THE 'IN' FUNCTION
(Offset 2C: 'in')

This subroutine handles the function IN X. It inputs at processor level from port X, loading BC with X and performing the instruction IN A,(C).
THE 'PEEK' FUNCTION
(Offset 2B: 'peek')

This subroutine handles the function PEEK X. The 'last value' is unstacked by calling FIND-INT2 and replaced by the value of the contents of the required location.

34AC  peek  CALL  1E99,FIND-INT2  Evaluate the 'last value', rounded to the nearest integer; test that it is in range and return it in BC.

LD  A,(BC)  Fetch the required byte.

34B0  IN-PK-STK  JP  2D28,STACK-A  Exit by jumping to STACK-A.

THE 'USR' FUNCTION
(Offset 2D: 'usr-no')

This subroutine ('USR number' as distinct from 'USR string') handles the function USR X, where X is a number. The value of X is obtained in BC, a return address is stacked and the machine code is executed from location X.

34B3  usr-no  CALL  1E99,FIND-INT2  Evaluate the 'last value', rounded to the nearest integer; test that it is in range and return it in BC.

LD  HL,+2D2B  Make the return address be that of the subroutine STACK-BC.

PUSH  HL  Make an indirect jump to the required location.

Note: It is interesting that the IY register pair is re-initialised when the return to STACK-BC has been made, but the important HL' that holds the next literal pointer is not restored should it have been disturbed. For a successful return to BASIC, HL' must on exit from the machine code contain the address in SCANNING of the 'end-calc' instruction, 2758 hex (10072 decimal).

THE 'USR-STRING' FUNCTION
(Offset 19: 'usr-$')

This subroutine handles the function USR X$, where X$ is a string. The subroutine returns in BC the address of the bit pattern for the user-defined graphic corresponding to X$. It reports error A if X$ is not a single letter between a and u or a user-defined graphic.

34BC  usr-$  CALL  2BF1,STK-FETCH  Fetch the parameters of the string X$.

DEC  BC  Decrease the length by 1 to test it.

LD  A,B  If the length was not 1, then

OR  C  jump to give error report A.

JR  NZ,34E7,REPORT-A  Give report A if out of range.

LD  A,(DE)  Fetch the single code of the string.

CALL  2C8D,ALPHA  Does it denote a letter?

JR  C,34D3,USR-RANGE  If so, jump to gets its address.

SUB  +90  Reduce range for actual user-defined graphics to 0 - 20 decimal.

JR  C,34E7,REPORT-A  Give report A if out of range.

CP  +15  Test the range again.

JR  NC,34E7,REPORT-A  Give report A if out of range.

INC  A  Make range of user-defined graphics 1 to 21 decimal, as for a to u.
Now make the range 0 to 20 decimal in each case.

Multiply by 8 to get an offset for the address.

Test the range of the offset.

Give report A if out of range.

Fetch the address of the first user-defined graphic in BC.

Add C to the offset.

Store the result back in C.

Increment B to complete the address.

Jump to stack the address.

Call the error handling routine.

The 'Test-Zero' Subroutine
This subroutine is called at least nine times to test whether a floating-point number is zero. This test requires that the first four bytes of the number should each be zero. The subroutine returns with the carry flag set if the number was in fact zero.

Save HL on the stack.

Save BC on the stack.

Save the value of A in B.

Get the first byte.

Point to the second byte.

OR first byte with second.

Point to the third byte.

OR the result with the third byte.

Point to the fourth byte.

OR the result with the fourth byte.

Restore the original value of A. And of BC.

Restore the pointer to the number to HL.

Return with carry reset if any of the four bytes was non-zero.

Set the carry flag to indicate that the number was zero, and return.

The 'Greater Than Zero' Operation
(Offset 37: 'greater-0')
This subroutine returns a 'last value' of one if the present 'last value' is greater than zero and zero otherwise. It is also used by other subroutines to 'jump on plus'.

Is the 'last-value' zero?
If so, return.

Jump forward to LESS THAN ZERO but signal the opposite action is needed.

The 'Not' Function
(Offset 30: 'not')
This subroutine returns a 'last value' of one if the present 'last value' is zero and zero otherwise. It is also used by other subroutines to 'jump on zero'.
THE 'LESS THAN ZERO' OPERATION
(Offset 36: 'less-0')
This subroutine returns a 'last value' of one if the present 'last value' is less than zero and zero otherwise. It is also used by other subroutines to 'jump on minus'.

3506            less-0    XOR    A
3507            SIGN-TO-C  INC    HL
                      XOR    (HL)
                      DEC    HL
                      RLCA

THE 'ZERO OR ONE' SUBROUTINE
This subroutine sets the 'last value' to zero if the carry flag is reset and to one if it is set. When called from 'E-TO-FP' however it creates the zero or one not on the stack but in mem-0.

350B            FP-0/1  PUSH    HL
                      LD    A,+00
                      LD    (HL),A
                      INC    HL
                      LD    (HL),A
                      INC    HL
                      RLA
                      LD    (HL),A
                      RRA
                      INC    HL
                      LD    (HL),A
                      POP    HL

THE 'OR' OPERATION
(Offset 07: 'or')
This subroutine performs the binary operation 'X OR Y' and returns X if Y is zero and the value 1 otherwise.

351B            or       EX    DE,HL
                      CALL    34E9,TEST-ZERO
                      EX    DE,HL
                      RET    C
                      SCF
                      JR    350B,FP-0/1

THE 'NUMBER AND NUMBER' OPERATION
(Offset 08: 'no-&-no')
This subroutine performs the binary operation 'X AND Y' and returns X if Y is non-zero and the value zero otherwise.
THE 'STRING AND NUMBER' OPERATION
(Offset 10: 'str-&-no')

This subroutine performs the binary operation 'X$ AND Y' and returns X$ if Y is non-zero and a null string otherwise.

THE 'COMPARISON' OPERATIONS
(Offsets 09 to 0E & 11 to 16: 'no-l-eql', 'no-gr-eq', 'nos-neqI', 'no-grtr', 'no-less', 'nos-eqI', 'str-l-eql', 'str-gr-eq', 'strs-neqI', 'str-grtr', 'str-less' & 'strs-eqI')

This subroutine is used to perform the twelve possible comparison operations. The single operation offset is present in the B register at the start of the subroutine.
3559 STRINGS JR 358C,END-TESTS
RRCA the final tests.
PUSH AF The string comparisons now
CALL 2BF1,STK-FETCH have the range 02-03 with carry
PUSH DE set for 'equal' and 'not equal'.
PUSH BC Save the offset.
call 2BF1,STK-FETCH
POP HL The lengths and starting
3564 BYTE-COMP addresses of the strings are
LD A,H fetched from the calculator
OR L stack.
EX (SP),HL
LD A,B Jump unless the second string
JR NZ,3575,SEC-PLUS is null.
OR C Here the second string is either
POP BC null or less than the first.
356B SECND-LOW JR Z,3572,BOTH-NULL
POP AF
CCF The carry is complemented to
JR 3588,STR-TEST give the correct test results.
POP AF Here the carry is used as it
JR 3588,STR-TEST stands.
3572 BOTH-NULL OR C The first string is now null,
JR Z,3585,FRST-LESS the second not.
LD A,(DE) Neither string is null, so their
SUB (HL) next bytes are compared.
JR C,3585,FRST-LESS The first byte is less.
INC BC The second byte is less.
DEC DE The lengths are decremented and a
INC HL jump is made to BYTE-COMP
EX (SP),HL to compare the next bytes of
DEC HL the reduced strings.
3575 SEC-PLUS JR 3564,BYTE-COMP
POP BC The carry is cleared here for the
POP AF correct test results.
AND A For the string tests, a zero is
3585 FRST-LESS put on to the calculator stack.
PUSH AF
DEFB +A0,stk-zero
DEFB +38, end-calc
3588 STR-TEST POP AF These three tests, called as
RST 0028,FP-CALC needed, give the correct results
DEFB +A0,stk-zero for all twelve comparisons. The
DEFB +38, end-calc initial carry is set for 'not equal'
358C END-TESTS and 'equal', and the final carry
POP AF is set for 'greater than', 'less
PUSH AF than' and 'equal'.
CALL C,3501,NOT
POP AF Finished.
RRCA
CALL NC,3501,NOT
RET
THE 'STRING CONCATENATION' OPERATION
(Offset 17: 'strs+add')

This subroutine performs the binary operation 'A$+B$'. The parameters for these strings are fetched and the total length found.
Sufficient room to hold both the strings is made available in the work space and the strings are copied over. The result of this subroutine is therefore to produce a temporary variable A$+B$ that resides in the work space.
THE 'STK-PNTRS' SUBROUTINE
This subroutine resets the HL register pair to point to the first byte of the 'last value', i.e. STKEND-5, and the DE register pair to point one past the 'last value', i.e. STKEND.

THE 'CHR$' FUNCTION
(Offset 2F: 'chrs')
This subroutine handles the function CHR$ X and creates a single character string in the work space.
THE 'VAL' AND 'VAL$' FUNCTION
(Offsets 1D: 'val' and 18: 'val$')

This subroutine handles the functions VAL X$ and VAL$ X$. When handling VAL X$, it return a 'last value' that is the result of evaluating the string (without its bounding quotes) as a numerical expression. When handling VAL$ X$, it evaluates X$ (without its bounding quotes) as a string expression, and returns the parameters of that string expression as a 'last value' on the calculator stack.

35DE val
( also val$)
LD HL,(CH-ADD) The current value of CH-ADD is
PUSH HL preserved on the machine stack.
LD A,B The 'offset' for 'val' or 'val$'
ADD A,+E3 must be in the B register; it is
SBC A,A now copied to A.
PUSH AF Produce +00 and carry set for
CALL 2BF1,STK-FETCH 'val', +FB and carry reset for
PUSH DE 'val$'.
INC BC
RST 0030,BC-SPACES
POP HL The parameters of the string are
LD (CH-ADD),DE fetched; the starting address is
PUSH DE saved; one byte is added to the
LDIR length and room made available
for the string (+1) in the work
space.
EX DE,HL The starting address of
DEC HL the string goes to HL as a source
LD (HL)+0D address.
RES 7,(FLAGS) The pointer to the first new
CALL 24FB,SCANNING space goes to CH-ADD and to
RST 0018,GET-CHAR the machine stack.
LD (CH-ADD),DE The string is copied to the work
PUSH DE space, together with an extra
LDIR byte.
EX DE,HL Switch the pointers.
DEC HL The extra byte is replaced by a
LD (HL)+0D 'carriage return' character.
RES 7,(FLAGS) The string is scanned for correct
CALL 24FB,SCANNING syntax.
RST 0018,GET-CHAR The character after the string is
CP +0D fetched.
JR NZ,360C,V-RPORT-C A check is made that the end of
POP HL the expression has been reached.
If not, the error is reported.
POP AF The 'flag' for 'val/val$' is
XOR (FLAGS) fetched and bit 6 is compared
AND +40 with bit 6 of the result of the syntax scan.
360C V-RPORT-C JP NZ,1C8A,REPORT-C Report the error if they do not
LD (CH-ADD),HL match.
SET 7,(FLAGS) The flag is set for line
execution.
THE 'STR$' FUNCTION

(Offset 2E: 'str$')

This subroutine handles the function STR$ X and returns a 'last value' which is a set of parameters that define a string containing what would appear on the screen if X were displayed by a PRINT command.

```
361F  str$
  RST  0030,BC-SPACES
  LD   (K-CUR),HL
  PUSH HL
  LD   HL,(CURCHL)
  PUSH HL
  LD   A,+FF
  CALL 1601,CHAN-OPEN
  CALL 2DE3,PRINT-FP
  POP HL
  CALL 1615,CHAN-FLAG
  POP DE
  LD   HL,(K-CUR)
  AND A
  SBC HL,DE
  LD   B,H
  LD   C,L
  CALL 2AB2,STK-STO-$
  EX DE,HL
  RET
```

**Note:** See PRINT-FP for an explanation of the 'PRINT "A"+STR$ 0.1' error.

THE 'READ-IN' SUBROUTINE

(Offset 1A: 'read-in')

This subroutine is called via the calculator offset through the first line of the S:INKEY$ routine in SCANNING. It appears to provide for the reading in of data through different streams from those available on the standard Spectrum. Like INKEY$ the subroutine returns a string.

```
3645  read-in
  CALL 1E94,FIND-INT1
  CP +10
  JP NC,1E9F,REPORT-B
  LD HL,(CURCHL)
  PUSH HL
  CALL 1601,CHAN-OPEN
```
CALL 15E6,INPUT-AD  The signal is now accepted, like
LD  BC,+0000  a 'key-value'.
JR NC,365F,R-I-STORE  The default length of the
INC C  resulting string is zero.
RST 0030,BC-SPACES  Jump if there was no signal.
LD (DE),A  Make a space in the work space.
CALL 2AB2,STK-STO-$  Put the string into it.
LD A,B  Pass the parameters of the string
OR C  to the calculator stack.
JR Z,3671,STK-CODE  Restore CURCHL and the
LD A,(DE)  appropriate flags.
JP 35BF,STK-PNTRS  Exit, setting the pointers.

THE 'CODE' FUNCTION
(Offset 1C: 'code')

This subroutine handles the function CODE A$ and returns the Spectrum code of the first character in A$, or zero if A$ should be null.

CALL 2BF1,STK-FETCH  The parameters of the string are
LD A,B  fetched.
OR C  The length is tested and the A
JR Z,3671,STK-CODE  register holding zero is carried
LD A,(DE)  forward is A$ is a null string.

THE 'LEN' FUNCTION
(Offset 1E: 'len')

This subroutine handles the function LEN A$ and returns a 'last value' that is equal to the length of the string.

CALL 2BF1,STK-FETCH  The parameters of the string are
JP 2D2B,STACK-BC  fetched.

THE 'DECREASE THE COUNTER' SUBROUTINE
(Offset 35: 'dec-jr-nz')

This subroutine is only called by the SERIES GENERATOR subroutine and in effect is a 'DJNZ' operation but the counter is the system variable, BREG, rather than the B register.

EXX  Go to the alternative register set
PUSH HL  and save the next literal pointer
LD HL,+5C67  on the machine stack.
DEC (HL)  Make HL point to BREG.
POP HL  Decrease BREG.
JR NZ,3687,JUMP-2  Restore the next literal pointer.
INC HL  The jump is made on non-zero.
EXX  The next literal is passed over.
RET  Return to the main register set.

THE 'JUMP' SUBROUTINE
(Offset 33: 'jump')

This subroutine executes an unconditional jump when called by the literal '33'. It is also used by the subroutines DECREASE THE COUNTER and JUMP ON TRUE.
THE 'JUMP ON TRUE' SUBROUTINE
(Offset 00: 'jump-true')
This subroutine executes a conditional jump if the 'last value' on the calculator stack, or more precisely the number addressed currently by the DE register pair, is true.

368F jump-true INC DE Point to the third byte, which is zero or one.
INC DE Collect this byte in the A register.
LD A,(DE) Point to the first byte once again.
DEC DE Test the third byte: is it zero?
JR NZ,3686,JUMP Make the jump if the byte is non-zero, i.e. if the number is not-false.
EXX Go to the alternate register set.
INC HL Pass over the jump length.
EXX Back to the main set of registers.
RET Finished.

THE 'END-CALC' SUBROUTINE
(Offset 38: 'end-calc')
This subroutine ends a RST 0028 operation.

369B end-calc POP AF The return address to the calculator ('RE-ENTRY') is discarded.
EXX Instead, the address in HL' is put on the machine stack and
EXX (SP).HL an indirect jump is made to it.
EXX HL' will now hold any earlier address in the calculator chain of addresses.
RET Finished.

THE 'MODULUS' SUBROUTINE
(Offset 32: 'n-mod-m')
This subroutine calculates M (mod M), where M is a positive integer held at the top of the calculator stack, the 'last value', and N is the integer held on the stack beneath M.
The subroutine returns the integer quotient INT (N/M) at the top of the calculator stack, the 'last value', and the remainder N-INT (N/M) in the second place on the stack.
This subroutine is called during the calculation of a random number to reduce N mod 65537 decimal.
THE 'INT' FUNCTION

(Offset 27: 'int')

This subroutine handles the function INT X and returns a 'last value' that is the 'integer part' of the value supplied. Thus INT 2.4 gives 2 but as the subroutine always rounds the result down INT -2.4 gives -3.

The subroutine uses the INTEGER TRUNCATION TOWARDS ZERO subroutine at 3214 to produce I (X) such that I (2.4) gives 2 and I (-2.4) gives -2. Thus, INT X is given by I (X) for values of X that are greater than or equal to zero, and I (X)-1 for negative values of X that are not already integers, when the result is, of course, I (X).

![The 'INT' function subroutine code](image)

For values of X that have been shown to be greater than or equal to zero there is no jump and I (X) is readily found.

![Jump for positive X](image)

when X is a negative integer I (X) is returned, otherwise I (X)-1 is returned.

![Jump for negative X](image)

In either case the subroutine finishes with;

![Finish subroutine](image)

THE 'EXPONENTIAL' FUNCTION

(Offset 26: 'exp')

This subroutine handles the function EXP X and is the first of four routines that use SERIES GENERATOR to produce Chebyshev polynomials.

The approximation to EXP X is found as follows:

i. X is divided by LN 2 to give Y, so that 2 to the power Y is now the required result.

ii. The value N is found, such that N=INT Y.

iii. The value W is found, such that W=Y-N, where 0 <= W < 1, as required for the series to converge.

![The 'EXPONENTIAL' function subroutine code](image)
iv. The argument Z if formed, such that Z=2^w-1.

v. The SERIES GENERATOR is used to return 2^W.

vi. Finally N is added to the exponent, giving 2^(N+W), which is 2^Y and therefore the required answer for EXP X.

The method is illustrated using a BASIC program in the Appendix.

<table>
<thead>
<tr>
<th>36C4</th>
<th>EXP</th>
<th>RST</th>
<th>0028,FP-CALC</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defb</td>
<td>+3D, re-stack</td>
<td>X (in full floating-point form)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defb</td>
<td>+34, stk-data</td>
<td>X, 1/LN 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defb</td>
<td>+F1, exponent+81</td>
<td>Y, Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defb</td>
<td>+38, +AA, +3B, +29</td>
<td>Y, INT Y = N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defb</td>
<td>+04, multiply</td>
<td>X/LN 2 = Y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Perform step ii.

| Defb | +31, duplicate | Y, Y | |
| Defb | +27, int, 1C46 | Y, INT Y = N | |
| Defb | +C3, st-mem-3 | Y, N mem-3 holds N | |

Perform step iii.

| Defb | +03, subtract | Y-N = W | |

Perform step iv.

| Defb | +31, duplicate | W, W | |
| Defb | +0F, addition | 2^W | |
| Defb | +A1, stk-one | 2^W, 1 | |
| Defb | +03, subtract | 2^W-1 = Z | |

Perform step v, passing to the SERIES GENERATOR the parameter '8' and the eight constants required.

| Defb | +88, series-08 | Z | |
| Defb | +13, exponent+63 | | |
| Defb | +36, (+00, +00, +00) | | |
| Defb | +58, exponent+68 | | |
| Defb | +65, (+00, +00) | | |
| Defb | +9D, exponent+6D | | |
| Defb | +78, +65, +40, (+00) | | |
| Defb | +A2, exponent+72 | | |
| Defb | +60, +32, +C9, (+00) | | |
| Defb | +E7, exponent+77 | | |
| Defb | +21, +F7, +AF, +24 | | |
| Defb | +EB, exponent+7B | | |
| Defb | +2F, +B0, +B0, +14 | | |
| Defb | +EE, exponent+7E | | |
| Defb | +7E, +BB, +94, +58 | | |
| Defb | +F1, exponent+81 | | |
| Defb | +3A, +7E, +F8, +CF | | |

At the end of the last loop the 'last value' is 2**W.

Perform step vi.

| Defb | +E3, get-mem-3 | 2**W, N | |
| Defb | +38, end-calc | | |
| CALL | 2DD5, FP-TO-A | | |
| JR | NZ, 3705, N-NEGTV | Jump forward if N was negative. Error if ABS N greater than 255 dec. | |
| JR | C, 3703, REPORT-6 | | |
| ADD | A, (HL) | Now add ABS N to the exponent. | |
| JR | NC, 370C, RESULT-OK | Jump unless e greater than 255 dec. | |
THE 'NATURAL LOGARITHM' FUNCTION
(Offset 25: 'ln')

This subroutine handles the function LN X and is the second of the four routines that use SERIES GENERATOR to produce Chebyshev polynomials.

The approximation to LN X is found as follows:

I. X is tested and report A is given if X is not positive.
II. X is then split into its true exponent, e', and its mantissa X' = X/(2**e'), where X' is greater than, or equal to, 0.5 but still less than 1.
III. The required value Y1 or Y2 is formed. If X' is greater than 0.8 then Y1=e'*LN 2 and if otherwise Y2 = (e'-1)*LN 2.
IV. If X' is greater than 0.8 then the quantity X' - 1 is stacked; otherwise 2*X' - 1 is stacked.
V. Now the argument Z is formed, being if X' is greater than 0.8, Z = 2.5*X' - 3; otherwise Z = 5*X' - 3. In each case, -1 <= Z <= 1, as required for the series to converge.
VI. The SERIES GENERATOR is used to produce the required function.
VII. Finally a simply multiplication and addition leads to LN X being returned as the 'last value'.

Perform step i.

Report A - Invalid argument

Perform step ii.
DEFB +00,(+00,+00,+00)
DEFB +03,subtract X', e'

Perform step iii.

DEFB +01,exchange e', X'
DEFB +31,duplicate e', X', X'
DEFB +34,stk-data e', X', X', 0.8 (decimal)
DEFB +F0,exponent+80
DEFB +4C,+CC,+CC,+CD
DEFB +03,subtract e', X', X'-0.8
DEFB +37,greater-0 e', X', (1/0)
DEFB +00,jump-true e', X'
DEFB +08,to 373D, GRE.8 e', X'
DEFB +01,exchange X', e'
DEFB +A1,stk-one X', e', 1
DEFB +03,subtract X', e'-1
DEFB +01,exchange e'-1, X'
DEFB +38,end-calc e'-1, X'

RST 0028,FP-CALC
e'-1,2*X' - X' large.

373D GRE.8
DEFB +01,exchange X', e' - X' large.
DEFB +34,stk-data X', e', LN 2
DEFB +F0,exponent+80 2*X', e'-1, LN 2
DEFB +31,+72,+17,+F8
DEFB +04,multiply X', e'*LN 2 = Y1
2*X', (e'-1)*LN 2 = Y2

Perform step iv.

DEFB +01,exchange Y1, X', - X' large.
DEFB +A2,stk-half Y1, X', .5 (decimal)
DEFB +03,subtract Y1, X', .5
DEFB +A2,stk-half Y1, X', .5
DEFB +03,subtract Y1, X', .5

Perform step v.

DEFB +31,duplicate Y, X'-1, X'-1
DEFB +34,stk-data Y1, X'-1, X'-1, 2.5 (decimal)
DEFB +32,exponent+82
DEFB +20,(+00,+00,+00)
DEFB +04,multiply Y1, X'-1,2.5*X'-3 = Z
2*X'-1, 5*X'-3 = Z

Perform step vi, passing to the SERIES GENERATOR the parameter '12' decimal, and the twelve constant required.

DEFB +8C,series-0C Y1, X'-1, Z or Y2, 2*X'-1, Z
1. DEFB +11,exponent+61
2. DEFB +AC,(+00,+00,+00)
3. DEFB +14,exponent+64
4. DEFB +09,(+00,+00,+00)
5. DEFB +56,exponent+66
6. DEFB +DA,+A5,(+00,+00)
At the end of the last loop the 'last value' is:

either \( \ln \frac{x}{(x-1)} \) for the larger values of \( x \)
or \( \ln \frac{2x}{(2x-1)} \) for the smaller values of \( x \).

Perform step vii.

\[ \text{DEFB } +04, \text{multiply } Y_1 = 4 \times \ln (2^{(\varphi')}, \ln x) \]
\[ \text{DEFB } +0F, \text{addition } Y_2 = 4 \times \ln (2^{(\varphi'-1)}), \ln (2^x) \]
\[ \text{DEFB } +38, \text{end-cal } \text{LN } x \]

Finished: 'last value' is \( \ln x \).

THE 'REDUCE ARGUMENT' SUBROUTINE

(Offset 39: 'get-argt')

This subroutine transforms the argument \( x \) of \( \sin x \) or \( \cos x \) into a value \( v \).

The subroutine first finds a value \( y \) such that:
\[ y = \frac{x}{2\pi} - \lfloor \frac{x}{2\pi} \rfloor + 0.5, \]
where \( y > 0.5 \) or \( y < -0.5 \).

The subroutine returns with:
\[ v = 4y \] if \( -1 < 4y < 1 \),
\[ v = -4y \] if \( 1 < 4y < 2 \),
\[ v = -4y - 2 \] if \( -2 < 4y < -1 \).

In each case, \( -1 < v < 1 \) and \( \sin (\pi v/2) = \sin x \)

<table>
<thead>
<tr>
<th>3783</th>
<th>get-argt</th>
</tr>
</thead>
<tbody>
<tr>
<td>RST</td>
<td>0028,FP,CALC</td>
</tr>
<tr>
<td>DEFB</td>
<td>+3D,re-stack</td>
</tr>
<tr>
<td>DEFB</td>
<td>+4E,stk-data</td>
</tr>
<tr>
<td>DEFB</td>
<td>+EE,exponent+7E</td>
</tr>
<tr>
<td>DEFB</td>
<td>+22,F9,+83,+6E</td>
</tr>
<tr>
<td>DEFB</td>
<td>+04,multiply</td>
</tr>
<tr>
<td>DEFB</td>
<td>+31,duplicate</td>
</tr>
<tr>
<td>DEFB</td>
<td>+A2,stk-half</td>
</tr>
<tr>
<td>DEFB</td>
<td>+0F,addition</td>
</tr>
<tr>
<td>DEFB</td>
<td>+27,int,1C46</td>
</tr>
<tr>
<td>DEFB</td>
<td>+03,subtract,174C</td>
</tr>
</tbody>
</table>
Note: Adding 0.5 and taking INT rounds the result to the nearest integer.

DEFB +31,duplicate Y, Y
DEFB +0F,addition 2*Y
DEFB +31,duplicate 2*Y, 2*Y
DEFB +0F,addition 4*Y
DEFB +31,duplicate 4*Y, 4*Y
DEFB +2A.abs 4*Y, ABS (4*Y)
DEFB +A1,stk-one 4*Y, ABS (4*Y), 1
DEFB +03,subtract 4*Y, ABS (4*Y)-1 = Z
DEFB +31,duplicate 4*Y, Z, Z
DEFB +37,greater-0 4*Y, Z, (1/0)
DEFB +C0,stk-mem-0 Mem-0 holds the result of the test.
DEFB +00,jump-true 4*Y, Z
DEFB +04, to 37A1,ZPLUS 4*Y, Z
DEFB +02,delete 4*Y
DEFB +38,end-calc 4*Y = V - case i.
RET Finished.

If the jump was made then continue.

37A1  ZPLUS  DEFB +A1,stk-one 4*Y, Z, 1
DEFB +03,subtract 4*Y, Z-1
DEFB +01,exchange Z-1,4*Y
DEFB +36,less-0 Z-1,(1/0)
DEFB +02,move 4*Y, Z-1
DEFB +1B, negate 1-Z
DEFB +38,end-calc 1-Z = V - case ii.
DEFB +02,move Z-1, ABS V = V - case iii.
RET Finished.

THE 'COSINE' FUNCTION
(Offset 20: 'cos')

This subroutine handles the function COS X and returns a "last value" that is an approximation to COS X.
The subroutine uses the expression:
COS X = SIN (PI*W/2), where -1 <=W <=1.
In deriving W for X the subroutine uses the test result obtained in the previous subroutine and stored for this purpose in mem-0. It then jumps to the SINE subroutine, entering at C-ENT, to produce a "last value" of COS X.

37AA  cos  RST 0028,FP-CALC. X
DEFB +39,get-argt V
DEFB +2A.abs ABS V
DEFB +A1,stk-one ABS V, 1
DEFB +03,subtract ABS V-1
DEFB +E0,move ABS V-1, (1/0)
DEFB +00,move ABS V-1
DEFB +06, to 37B7,C-ENT ABS V-1 = W

If the jump was not made then continue.

DEFB +1B, negate 1-ABS V
DEFB +33, jump 1-ABS V
DEFB +03, to 37B7,C-ENT 1-ABS V = W

THE 'SINE' FUNCTION
(Offset 1F: 'sin')

This subroutine handles the function SIN X and is the third of the four routines that use SERIES GENERATOR to produce Chebyshev polynomials.
The approximation to SIN X is found as follows:

i. The argument X is reduced and in this case W = V directly.
Note that \(-1 \leq W \leq 1\), as required for the series to converge.

ii. The argument \(Z\) is formed, such that \(Z = 2W^2 - 1\).

iii. The SERIES GENERATOR is used to return \((\sin (\pi W/2))/W\)
iv. Finally a simple multiplication gives \(\sin X\).

37B5 \sin \hspace{0.5cm} \text{RST} \hspace{0.5cm} 0028 \hspace{0.5cm} \text{FP-CALC} \hspace{0.5cm} X

Perform step i.

```assembly
DEFB +39.get-argt \hspace{1cm} W
```

Perform step ii. The subroutine from now on is common to both the SINE and COSINE functions.

37B7 C-ENT \hspace{0.5cm} DEFB +31.duplicate \hspace{0.5cm} W, W
DEFB +31.duplicate \hspace{0.5cm} W, W, W
DEFB +04.multiply \hspace{0.5cm} W, W*W
DEFB +31.duplicate \hspace{0.5cm} W, W*W, W*W
DEFB +0F.addition \hspace{0.5cm} W, 2W*W
DEFB +A1.stk-one \hspace{0.5cm} W, 2W*W, 1
DEFB +03.subtract \hspace{1cm} W, 2W*W - 1 = Z

Perform step iii, passing to the SERIES GENERATOR the parameter '6' and the six constants required.

```assembly
DEFB +86.series-06 \hspace{0.5cm} W, Z
1. DEFB +14.exponent+64
DEFB +E6.(+00,+00,+00)
2. DEFB +5C.exponent+6C
DEFB +1F.+0B.(+00,+00)
3. DEFB +A3.exponent+73
DEFB +6F.+38.+EE.(+00)
4. DEFB +E9.exponent+79
DEFB +15.+63.+BB,+23
5. DEFB +EE.exponent+7E
DEFB +92.+0D.+CD.+ED
6. DEFB +F1.exponent+81
DEFB +23.+5D.+1B.+EA
```

At the end of the last loop the 'last value' is \((\sin (\pi W/2))/W\).

Perform step v.

```assembly
DEFB +04.multiply \hspace{1cm} \sin (\pi W/2) = \sin X \text{ (or = COS X)}
DEFB +38.end-calc
RET
```

Finished: 'last value' = \(\sin X\).

OR ('last value' = \(\cos X\))

THE 'TAN' FUNCTION

(Offset 21: 'tan')

This subroutine handles the function \(TAN\ X\). The subroutine simply returns \(\sin X/\cos X\), with arithmetic overflow if \(\cos X = 0\).

37DA \tan \hspace{0.5cm} \text{RST} \hspace{0.5cm} 0028,\text{FP-CALC} \hspace{0.5cm} X
DEFB +31.duplicate \hspace{0.5cm} X, X
DEFB +1F.sin \hspace{0.5cm} X, \sin X
DEFB +01.exchange \hspace{0.5cm} \sin X, X
DEFB +20.cos \hspace{0.5cm} \sin X,\cos X
DEFB +05.division \hspace{1cm} \sin X/\cos X = TAN X
Report arithmetic overflow if needed.
DEFB +38.end-calc \hspace{1cm} TAN X
RET
Finished: 'last value' = TAN X.
THE 'ARCTAN' FUNCTION
(Offset 24: 'atn')

This subroutine handles the function ATN X and is the last of the four routines that use SERIES GENERATOR to produce Chebyshev polynomials. It returns a real number between -PI/2 and PI/2, which is equal to the value in radians of the angle whose tan is X.

The approximation to ATN X is found as follows:

i. The values W and Y are found for three cases of X, such that:
   if -1 < X < 1  then W = 0 & Y = X  - case i.
   if -1 < = X < 1 then W = Pi/2 & Y = 1/X  - case ii.
   if X < = -1 then W = -Pi/2 & Y = -1/X  - case iii.

In each case, -1 < = Y < =1, so that the series converge.

ii. The argument Z is formed, such that:
   if -1 < X < 1 then Z = 2*Y*Y - 1 = 2*X*X - 1  - case i.
   if 1 < X then Z = 2*Y*Y - 1 = 1/(X*X) - 1  - case ii.
   if X < = -1 then Z = 2*Y*Y - 1 = 1/(X*X) - 1  - case iii.

iii. The SERIES GENERATOR is used to produce the required function.

iv. Finally a simple multiplication and addition give ATN X.

Perform stage i.

37E2  atn    CALL   3297,RE-STACK    Use the full floating-point form of X.
LD    A,(HL)    Fetch the exponent of X.
CP    +81
JR    C,37F8.SMALL    Jump forward for case i: Y = X.
RST   0028,FP-CALC    X
DEFB  +A1,stk-one    X, 1
DEFB  +1B,negate    -1, X
DEFB  +05,division    -1/X
DEFB  +31,duplicate    -1/X, -1/X
DEFB  +36,less-0    -1/X, (1/0)
DEFB  +A3,stk-pi/2    -1/X, (1/0), PI/2
DEFB  +01,exchange    -1/X, PI/2
DEFB  +00,jump-true    -1/X, PI/2
DEFB  +06, to 37FA,CASES
DEFB  +1B, negate    Y = -1/X, W = PI/2
DEFB  +33, jump    -1/X, -PI/2
DEFB  +03, to 37FA,CASES
RST   0028,FP-CALC    Y
DEFB  +A0,stk-zero    Y, 0
                Continue for case i: W = 0

Perform step ii.

37F8  SMALL   RST   0028,FP-CALC    Y
DEFB  +00, to 37FA,CASES
DEFB  +01, exchange    W, Y
DEFB  +31, duplicate    W, Y, Y
DEFB  +31, duplicate    W, Y, Y, Y
DEFB  +04, multiply    W, Y, Y*Y
DEFB  +31, duplicate    W, Y, Y*Y, Y*Y
DEFB  +0F, add    W, Y, 2*Y*Y
DEFB  +A1, stk-one    W, Y, 2*Y*Y, 1
DEFB  +03, subtract    W, Y, 2*Y*Y+1 = Z

Perform step iii, passing to the SERIES GENERATOR the parameter '12' decimal, and the twelve constants required.

1. DEF  +8C, series-0C    W, Y, Z
DEFB  +10, exponent+60
DEFB  +B2,(+00,+00,+00)
At the end of the last loop the 'last value' is:

ATN X/X - case i.
ATN (-1/X)/(-1/X) - case ii.
ATN (-1/X)/(-1/X) - case iii.

Perform step iv.

THE 'ARCSIN' FUNCTION
(Offset 22: 'asn')

This subroutine handles the function ASN X and return a real real number from -PI/2 to PI/2 inclusive which is equal to the value in radians of the angle whose sine is X. Thereby if Y = ASN X then X = SIN Y.

This subroutine uses the trigonometric identity:

TAN (Y/2) = SIN Y/(1+COS Y)

to obtain TAN (Y/2) and hence (using ATN) Y/2 and finally Y.
**THE 'ARCCOS' FUNCTION**

(Offset 23: 'acs')

This subroutine handles the function ACS X and returns a real number from zero to PI inclusive which is equal to the value in radians of the angle whose cosine is X.

This subroutine uses the relation:

\[
\text{ACS } X = \frac{\pi}{2} - \text{ASN } X
\]

3843 acs
RST 0028,FP-CALC X
DEFB +22,asm ASN X
DEFB +A3,stk-pi/2 ASN X.PI/2
DEFN +03,subtract ASN X.PI/2
DEFB +1B,negate PI/2-ASN X = ACS X
DEFB +38,end-calc
RET

Finished: 'last value' = ACS X.

**THE 'SQUARE ROOT' FUNCTION**

(Offset 28: 'sqr')

This subroutine handles the function SQR X and returns the positive square root of the real number X if X is positive, and zero if X is zero. A negative value of X gives rise to report A - invalid argument (via ln in the EXPONENTIATION subroutine).

This subroutine treats the square root operation as being \(X^{0.5}\) and therefore stacks the value .5 and proceeds directly into the EXPONENTIATION subroutine.

384A sqr
RST 0028,FP-CALC X
DEFB +31,duplicate X,X
DEFB +30,not X,(1/0)
DEFB +00,jump-true X
DEFB +1E,to 386C,LAST X

The jump is made if X = 0, otherwise continue with:

DEFB +A2,stk-half X,.5
DEFB +39,end-calc

and then find the result of \(X^{0.5}\).

**THE 'EXPONENTIATION' OPERATION**

(Offset 06: 'to-power')

This subroutine performs the binary operation of raising the first number, X, to the power of the second number, Y.

The subroutine treats the result \(X^Y\) as being equivalent to EXP \((YLN X)\). It returns this value unless X is zero, in which case it returns 1 if Y is also zero \((0^0=1)\), returns zero if Y is positive and reports arithmetic overflow if Y is negative.

3851 to-power
RST 0028,FP-CALC X,Y
DEFB +01,exchange Y,X
DEFB +31,duplicate Y,X,X
DEFB +30,not Y,X,(1/0)
DEFB +00,jump-true Y,X
DEFB +07,to 385D,XIS0 Y,X

The jump is made if X = 0, otherwise EXP \((YLN X)\) is formed.

DEFB +25,ln Y,LN X

Giving report A if X is negative.

DEFB +04,multiply Y*LN X
DEFB +38,end-calc
JP 36C4,EXP Exit via EXP to form EXP \((Y*LN X)\).

The value of X is zero so consider the three possible cases involved.

385D XIS0
DEFB +02,delete Y
DEFB +31,duplicate Y,Y
DEFB +30,not Y,(1/0)
DEFB +00,jump-true Y
The jump is made if $X = 0$ and $Y = 0$, otherwise proceed.

DEFB +09, to 386A, ONE

```
Y
```

The jump is made if $X = 0$ and $Y$ is positive, otherwise proceed.

DEFB +A0, stk-zero

```
Y, 0
```

DEFB +01, exchange

```
0, Y
```

DEFB +37, greater-0

```
0, (1/0)
```

DEFB +00, jump-true

```
0
```

DEFB +06, to 386C, LAST

```
0
```

The result is to be 1 for the operation.

386A ONE

```
DEFB +02, delete
-DEFB +A1, stk-one
```

Now return with the 'last value' on the stack being $0^* Y$.

386C LAST

```
DEFB +38, end-calc
(1/0)
RET
```

Finished: 'last value' is 0 or 1.

386E - 3CFF These locations are 'spare'. They all hold +FF.

3D00 - 3FFF These locations hold the 'character set'. There are 8 byte representations for all the characters with codes $+20$ (space) to $+7F$ (©).

- e.g. the letter 'A' has the representation 00 3C 42 42 7E 42 42 00 and thereby the form:

```
00000000
00111100
01000010
01000010
01111110
01000010
01000010
00000000
```
APPENDIX

BASIC PROGRAMS FOR THE MAIN SERIES

The following BASIC programs have been included as they give a good illustration of how Chebyshev polynomials are used to produce the approximations to the functions SIN, EXP, LN and ATN.

The series generator:

This subroutine is called by all the 'function' programs.

```
500 REM SERIES GENERATOR, ENTER
510 REM USING THE COUNTER BREG
520 REM AND ARRAY - A HOLDING THE
530 REM CONSTANTS.
540 REM FIRST VALUE IN Z.
550 LET M0=2*Z
560 LET M2=0
570 LET T=0
580 FOR I=BREG TO 1 STEP -1
590 LET M1=M2
600 LET U=T*M0
610 LET M2=T
620 LET T=U
630 NEXT I
640 LET T=T-M1
650 RETURN
660 REM LAST VALUE IN T.
```

In the above subroutine the variables are:

- `Z` - the entry value.
- `T` - the exit value.
- `M0` - mem-0
- `M1` - mem-1
- `M2` - mem-2
- `I` - the counter for BREG.
- `U` - a temporary variable for T.
- `A(1)` to `A(BREG)` - the constants.
- `BREG` - the number of constants to be used.

To see how the Chebyshev polynomials are generated, record on paper the values of U, M1 and T through the lines 550 to 630, passing, say, 6 times through the loop, and keeping the algebraic expressions for `A(1)` to `A(6)` without substituting numerical values.

Then record T-M1. The multipliers of the constants `A(1)` to `A(6)` will then be the required Chebyshev polynomials. More precisely, the multiplier of `A(1)` will be `2*T5(Z)`, for `A(2)` it will be `2*T4(Z)` and so on to `2*T1(Z)` for `A(5)` and finally `T0(Z)` for `A(6)`.

Note that `T0(Z)=1`, `T1(Z)=Z` and, for `n>=2`, `Tn(Z)=2*Z*Tn-1(Z)-Tn-2(Z)`. 

---

222
SIN X

10 REM DEMONSTRATION FOR SIN X
20 REM USING THE 'SERIES GENERATOR'.
30 DIM A(6)
40 LET A(1)=-0.000000003
50 LET A(2)=0.0000000592
60 LET A(3)=-.000068294
70 LET A(4)=0.004559008
80 LET A(5)=-.142630785
90 LET A(6)=1.276278962
100 PRINT
110 PRINT "ENTER START VALUE IN DEGREES"
120 INPUT C
130 CLS
140 LET C=C-10
150 PRINT "BASIC PROGRAM","ROM PROGRAM"
160 PRINT "-------------","-----------"
170 PRINT
180 FOR J=1 TO 4
190 LET C=C+10
200 LET Y=C/360-INT (C/360+.5)
210 LET W=4*Y
220 IF W > 1 THEN LET W=2-W
230 IF W < -1 THEN LET W=-W-2
240 LET Z=2*W^2-1
250 LET BREG=6
260 REM USE 'SERIES GENERATOR'
270 GO SUB 550
280 PRINT TAB 6; "SIN ",C," DEGREES"
290 PRINT
300 PRINT T*W,SIN (PI*C/180)
310 PRINT
320 NEXT J
330 GO TO 100

NOTES:

I. When C is entered this program calculates and prints SIN C degrees, SIN (C+10) degrees, SIN (C+20) degrees and SIN (C+30) degrees. It also prints the values obtained by using the ROM program. For a specimen of results, try entering these values in degrees: 0; 5; 100; -80; -260; 3600; -7200.

II. The constants A(1) to A(6) in lines 40 to 90 are given (apart from a factor of 1/2) in Abramowitz and Stegun Handbook of Mathematical Functions (Dover 1965) page 76. They can be checked by integrating (SIN (PI*X/2))/X over the interval U=0 to PI, after first multiplying by COS (N*U) for each constant (i.e. N=1,2,...,6) and substituting COS U=2*X*X-1. Each result should then be divided by PI. (This integration can be performed by approximate methods e.g. using Simpson's Rule if there is a reasonable computer or programmable calculator to hand.)
**EXP X**

10 REM DEMONSTRATION FOR EXP X
20 REM USING THE 'SERIES GENERATOR'
30 LET T=0          (This makes T the first variable.)
40 DIM A(8)
50 LET A(1)=0.000000001
60 LET A(2)=0.000000053
70 LET A(3)=0.000001851
80 LET A(4)=0.000053453
90 LET A(5)=0.001235714
100 LET A(6)=0.021446556
110 LET A(7)=0.248762434
120 LET A(8)=1.456999875
130 PRINT
140 PRINT "ENTER START VALUE"
150 INPUT C
160 CLS
170 LET C=C-10
180 PRINT "BASIC PROGRAM", "ROM PROGRAM"
190 PRINT "-------------", "-----------"
200 PRINT
210 FOR J=1 TO 4
220 LET C=C+10
230 LET D=C/1.442695041     (D=C/(1/LN 2);EXP C=2**D).
240 LET N=INT D
250 LET Z=D-N             (2**(N+Z) is now required).
260 LET Z=2*Z-1
270 LET BREG=8
280 REM USE "SERIES GENERATOR"
290 GO SUB 550
300 LET V=PEEK 23627+256*PEEK 23628+1     (V=(VARS)+1)
310 LET N=N+PEEK V
320 IF N > 255 THEN STOP     (STOP with arithmetic overflow).
330 IF N < 0 THEN GO TO 360
340 POKE V,N
350 GO TO 370
360 LET T=0
370 PRINT TAB 11;"EXP ";C
380 PRINT
390 PRINT T,EXP C
400 PRINT
410 NEXT J
420 GO TO 130

**NOTES:**

I. When C is entered this program calculates and prints EXP C, EXP (C+10), EXP (C+20) and EXP (C+30). It also prints the values obtained by using the ROM program. For a specimen of results, try entering these values: 0; 15; 65 (with overflow at the end); -100; -40.

II. The exponent is tested for overflow and for a zero result in lines 320 and 330. These tests are simpler in BASIC than in machine code, since the variable N, unlike the A register, is not confined to one byte.

III. The constants A(1) to A(8) in lines 50 to 120 can be obtained by integrating 2**X over the interval U=0 to PI, after first multiplying the COS (N*U) for each constant (i.e. for N=1,2,...,8) and substituting COS U = 2**X-1. Each result should then be divided by PI.
REM DEMONSTRATION FOR LN X
REM USING THE 'SERIES GENERATOR'
LET D=0
DIM A(12)
LET A(1)=-.0000000003
LET A(2)=0.0000000020
LET A(3)=-.0000000127
LET A(4)=-0.0000000823
LET A(5)=-.0000005389
LET A(6)=0.0000035828
LET A(7)=-.0000243013
LET A(8)=0.001693953
LET A(9)=-.0012282837
LET A(10)=.0094766116
LET A(11)=-.0818414567
LET A(12)=.9302292213
PRINT
PRINT "ENTER START VALUE"
INPUT C
CLS
PRINT "BASIC PROGRAM", "ROM PROGRAM"
PRINT "-------------","-----------"
LET C=SQR C
FOR J=1 TO 4
LET C=C*C
IF C=0 THEN STOP
LET D=C
LET V=PEEK 23627+256*PEEK 23628+1
LET N=PEEK V-128
(N holds e').
POKE V,128
IF D<=0.8 THEN GO TO 360
LET S=D
LET Z=2.5*D-3
GO TO 390
360 LET N=N-1
LET S=2*D-1
LET Z=5*D-3
LET R=N*6.931471806
(R holds N*LN 2).
LET BREG=12
REM USE 'SERIES GENERATOR'
GO SUB 550
PRINT TAB 8;"LN ";C
PRINT S*T+R,LN C
NEXT J
GO TO 170

NOTES:
I. When C is entered this program calculates and prints LN C, LN (C^2), LN (C^4) and LN (C^8). It also prints the values obtained by using the ROM program. For a specimen of results, try entering these values: 1.1; 0.9; 300; 0.004; 1E5 (for overflow) and 1E-5 (STOP as 'invalid argument').
II. The constants A(1) to A(12) in lines 50 to 160 can be obtained by integrating 5*LN (4* (X+1)/5)/(4*X-1) over the interval U=0 to PI, after first multiplying by COS (N*U) for each constant (i.e. for N=1,2,...,12) and substituting COS U=2*X-1. Each result should then be divided by PI.
ATN X:

10 REM DEMONSTRATION FOR ATN X
20 REM USING THE 'SERIES GENERATOR'
30 DIM A(12)
40 LET A(1)= 0.0000000002
50 LET A(2)= 0.0000000010
60 LET A(3)= 0.0000000066
70 LET A(4)= 0.0000000432
80 LET A(5)= 0.000002850
90 LET A(6)= 0.000019105
100 LET A(7)= 0.000131076
110 LET A(8)= 0.000928715
120 LET A(9)= 0.006905975
130 LET A(10)= 0.055679210
140 LET A(11)= 0.0529464623
150 LET A(12)= 0.8813735870
160 PRINT
170 PRINT "ENTER START VALUE"
180 INPUT C
190 CLS
200 PRINT "BASIC PROGRAM", "ROM PROGRAM"
210 PRINT "-----------", "-----------"
220 PRINT
230 FOR J=1 TO 4
240 LET B=J*C
250 LET D=B
260 IF ABS B>=1 THEN LET D=-1/B
270 LET Z=2*D*D-1
280 LET BREG=12
290 REM USE "SERIES GENERATOR"
300 GO SUB 550
310 LET T=D*T
320 IF B >=1 THEN LET T=T+PI/2
330 IF B <=-1 THEN LET T=T-PI/2
340 PRINT TAB 8; "ATN", B
350 PRINT
360 PRINT T,ATN B (or PRINT T*180/PI,ATN B*180/PI to obtain the answers in degrees)
370 PRINT
380 NEXT J
390 GO TO 160

NOTES:

I. When C is entered this program calculates and prints ATN C, ATN (C*2), ATN (C*3) and ATN (C*4). For a specimen of results, try entering these values: 0.2; -1; 10 and -100. The results may be found more interesting if converted to yield degrees by multiplying the answers in line 360 by 180/PI.

II. The constants A(1) to A(12) in lines 40 to 150 are given (apart from a factor of 1/2) in Abramowitz and Stegun Handbook of Mathematical Functions (Dover 1965) page 82. They can be checked by integrating ATN X/X over the interval U=0 to PI, after first multiplying by COS (N*U) for each parameter (i.e. for n=1,2,...,12) and substituting COS U=2*X*X-1. Each result should then be divided by PI.
An alternative subroutine for SIN X:

It is straightforward to produce the full expansion of the Chebyshev polynomials and this can be written in BASIC as follows:

550 LET T = (32*Z*Z*Z*Z-40*Z*Z*Z+10*Z)*A(1) 
+ (16*Z*Z*Z*Z-19*Z*Z+2)*A(2) 
+ (8*Z*Z*Z-6*Z)*A(3) 
+ (4*Z*Z)+A(4) 
+ 2*Z*A(5) 
+ A(6)

560 RETURN

This subroutine is called instead of the SERIES GENERATOR and can be seen to be of a similar accuracy.

An alternative subroutine for EXP X:

The full expansion for EXP X is:

The full expansion for EXP X is:

+ (64*Z*Z*Z*Z*Z*Z-96*Z*Z*Z*Z+36*Z*Z*Z-2)*A(2) 
+ (32*Z*Z*Z*Z*Z-40*Z*Z*Z+10*Z)*A(3) 
+ (16*Z*Z*Z*Z-16*Z*Z+2)*A(4) 
+ (8*Z*Z*Z-6*Z)*A(5) 
+ (4*Z*Z)*A(6) 
+ 2*Z*A(7) 
+ A(8)

560 RETURN

The expansion for LN X and A TN X, given algebraically, will be:

(2048z^11-5632z^9+5632z^7-2464z^5+440z^3-22^2) * A(1) 
+ (1024z^10-2560z^8+2240z^6-890z^4+100z^2-2) * A(2) 
+ (512z^9-1152z^7+864z^5-240z^3+18z) * A(3) 
+ (256z^8-512z^6+320z^4-64z^2+2) * A(4) 
+ (128z^7-224z^5+112z^3-14z) * A(5) 
+ (64z^6-96z^4+36z^2-2) * A(6) 
+ (32z^5-40z^3+10z) * A(7) 
+ (16z^4-16z^2+2) * A(8) 
+ (8z^3-6z) * A(9) 
+ (4z^2-2) * A(10) 
+ (2z) * A(11) 
+ A(12)
THE 'DRAW' ALGORITHM
The following BASIC program illustrates the essential parts of the DRAW operation when being used to produce a straight line. The program in its present form only allows for lines where \( X > Y \).

```basic
10 REM DRAW 255,175 PROGRAM
20 REM SET ORIGIN
30 LET PLOTx=0: LET PLOTy=0
40 REM SET LIMITS
50 LET X=255: LET Y=175
60 REM SET INCREMENT,i
70 LET i=X/2
80 REM ENTER LOOP
90 FOR B=X TO 1 STEP -1
100 LET A=Y+i
110 IF X> A THEN GO TO 160
120 REM UP A PIXEL ON THIS PASS
130 LET A=A-X
140 LET PLOTy=PLOTy+1
150 REM RESET INCREMENT,i
160 LET i=A
170 REM ALWAYS ALONG ONE PIXEL
180 LET PLOTx=PLOTx+1
190 REM NOW MAKE A PLOT
200 PLOT PLOTx,PLOTy
210 NEXT B
```

A complete algorithm is to found in the following program, as a subroutine that will 'DRAW A LINE' from the last position to \( X,Y \).

THE 'CIRCLE' ALGORITHM
The following BASIC program illustrates how the CIRCLE command produces its circles.
Initially the number of arcs required is calculated. Then a set of parameters is prepared in the 'memory area' and the 'calculator stack'.
The arcs are then drawn by repeated calls to the line drawing subroutine that on each call draws a single line from the 'last position' to the position \( X,Y \).

Note: In the ROM program there is a final 'closing' line but this feature has not been included here.

```basic
10 REM A CIRCLE PROGRAM
20 LET X=127: LET Y=87: LET Z=87
30 REM How many arcs?
40 LET Arcs=4*INT (INT (ABS (PI*SQR Z)+0.5)/4)+4
50 REM Set up memory area; M0-M5
60 LET M0=X+Z
70 LET M1=0
80 LET M2=2*Z*SIN (PI/Arcs)
90 LET M3=1-2*(SIN (PI/Arcs)) ^ 2
100 LET M4=SIN (2*PI/Arcs)
110 LET M5=2*PI
120 REM Set up stack; Sa-Sd
130 LET Sa=X+Z
140 LET Sb=Y-Z*SIN (PI/Arcs)
150 LET Sc=Sa
160 LET Sd=Sb
170 REM Initialise COORDS
180 POKE 23677,Sa: POKE 23678,Sb
190 LET M0=Sd
200 REM 'DRAW THE ARCS'
210 LET M0=M0+M2
220 LET Sc=Sc+M1
230 LET X=Sc-PEEK 23677
240 LET Y=M0-PEEK 23678
```
NOTE ON SMALL INTEGERS AND -65536.

1. Small integers n are those for which -65535 is less than or equal to n which is less than or equal to 65535. The form in which they are held is described in 'STACK-BC'. Note that the manual is inaccurate when it says that the third and fourth bytes hold n plus 131072 if n is negative. Since the range of n is then -1 to -65535, the two bytes can only hold n plus 131072 if it is taken mod 65536; i.e. they hold n plus 65536. The manual is fudging the issue. The fact is that this is not a true twos complement form (as the form n plus 131072, in other circumstances, could be). Here the same number can stand for two different numbers according to the sign byte: e.g. 00 0 stands for 1 if the sign byte is 00 and for -65535 if the sign byte is FF; similarly FF FF stands for 65535 if the sign byte is 00 and for -1 if the sign byte is FF.

2. Accepting that negative numbers are given a special 'twos complement' form, the main feature about this method of holding numbers is that they are ready for 'short addition' without any further twos complementing. They are simply fetched and stored direct by the addition subroutine. But for multiplication they need to be fetched by INT-FETCH and stored afterwards by INT-STOR. These subroutines twos complement the number when fetching or storing it. The calls to INT-STOR are from 'multiply' (after 'short multiplication'), from 'truncate' (after forming a 'small integer' between -65535 and 65535 inclusive), from 'negate'/'abs' for the 'integer case' and from 'sgn' to store 1 or -1. The calls to INT-FETCH are from PRINT-FP to fetch the integer part of the number when it is 'small', from 'multiply' twice to fetch two 'small integers', from 'RE-STACK' to fetch a 'small integer' for re-stacking, from 'negate'/'abs' to fetch a 'small integer' for manipulation and from FP-TO-BC to fetch the integer for transfer to BC.
The Number -65536.

3. The number -65536 can fit into the 'small integer' format as 00 FF 00 00 00. It is then the 'limiting number', the one which when twos complemented overflows (cf. 80 hex in a simple one byte or 7 bit system, i.e. -128 decimal, which when twos complemented still gives 80 hex i.e. -128 decimal since the positive number 128 decimal does not fit into the system).

4. Some awareness of this may have inspired the abortive attempt to create 00 FF 00 00 00 in 'truncate'. It is abortive since it does not even survive the INT routine of which 'truncate' is a part. It just leads to the mistake INT (-65536) equals -1.

5. But the main error is that this number has been allowed to arise from 'short addition' of two smaller negative integers and then simply put on the stack as 00 FF 00 00 00. The system cannot cope with this number. The solution proposed in 'addition' is to form the full five byte floating-point form at once; i.e. test for the number first, at about byte 3032, as follows:

```assembly
3032  PUSH  AF  ; Save the sign byte in A.
3033  INC   A   ; Make any FF in A into 00.
3034  OR    E   ; Test all 3 bytes now for zero.
3035  OR    D   ;
3036  JR    NZ,3040,ADD-STORE  ; Jump if not -65536.
3038  POP   AF  ; Clear the stack.
3039  LD    (HL),+80  ; Enter 80 hex into second byte.
303B  DEC   HL  ; Point to the first byte.
303C  LD    (HL),+91  ; Enter 91 hex into the first byte.
303E  JR    3049,ADD-RSTOR  ; Jump to set the pointer and exit.
3040  ADD-STORE POP  AF  ; Restore the sign byte in A.
3041  LD    (HL),A  ; Store it on the stack.
3042  INC   HL  ; Point to the next location.
3043  LD    (HL),E  ; Store the low byte of the result.
3044  INC   HL  ; Point to the next location.
3045  LD    (HL),D  ; Store the high byte of the result.
3046  DEC   HL  ; Move the pointer back to
3047  DEC   HL  ; address the first byte of the
3048  DEC   HL  ; result.
3049  ADD-RSTOR POP  DE  ; Restore STKEND to DE.
304A  RET  ; Finished.
```

6. The above amendment (i.e. 15 extra bytes) with the omission of bytes 3223 to 323E inclusive from 'truncate' should solve the problems. It would be nice to be able to test this. The calls of INT-STORE should not lead to 00 FF 00 00 00 being stacked. In 'multiply' the number will lead to overflow if it occurs, since 65536 will set the carry flag; so 'long' multiplication will be used. As noted at 30E5, the 5 bytes starting there could probably be omitted if the above amendments were made. 'Negate' avoids stacking 00 FF 00 00 00 by treating zero separately and returning it unaltered. Truncate deals separately with -65536, as noted above. SGN stores only 1 and -1.
# INDEX TO ROUTINES

<table>
<thead>
<tr>
<th>address</th>
<th>routine</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>START</td>
<td>1</td>
</tr>
<tr>
<td>0008</td>
<td>Error</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>Print a character</td>
<td>1</td>
</tr>
<tr>
<td>0018</td>
<td>Collect character</td>
<td>1</td>
</tr>
<tr>
<td>0020</td>
<td>Collect next character</td>
<td>1</td>
</tr>
<tr>
<td>0028</td>
<td>Calculator</td>
<td>1</td>
</tr>
<tr>
<td>0030</td>
<td>Make BC spaces</td>
<td>1</td>
</tr>
<tr>
<td>0038</td>
<td>Maskable interrupt</td>
<td>1</td>
</tr>
<tr>
<td>0053</td>
<td>ERROR-2</td>
<td>2</td>
</tr>
<tr>
<td>0066</td>
<td>Non-maskable Interrupt</td>
<td>2</td>
</tr>
<tr>
<td>0074</td>
<td>CH-ADD+1</td>
<td>2</td>
</tr>
<tr>
<td>007D</td>
<td>SKIP-OVER</td>
<td>2</td>
</tr>
<tr>
<td>0095</td>
<td>Token tables</td>
<td>3</td>
</tr>
<tr>
<td>0205</td>
<td>Key tables</td>
<td>4</td>
</tr>
</tbody>
</table>

## THE RESTART ROUTINES and TABLES

### THE KEYBOARD ROUTINES

<table>
<thead>
<tr>
<th>address</th>
<th>routine</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>028E</td>
<td>Keyboard scanning</td>
<td>5</td>
</tr>
<tr>
<td>02BF</td>
<td>KEYBOARD</td>
<td>6</td>
</tr>
<tr>
<td>0310</td>
<td>Repeating key</td>
<td>7</td>
</tr>
<tr>
<td>031E</td>
<td>K-TEST</td>
<td>7</td>
</tr>
<tr>
<td>0333</td>
<td>Keyboard decoding</td>
<td>8</td>
</tr>
</tbody>
</table>

### THE LOUDSPEAKER ROUTINES

<table>
<thead>
<tr>
<th>address</th>
<th>routine</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>03B5</td>
<td>BEEPER</td>
<td>11</td>
</tr>
<tr>
<td>03F8</td>
<td>BEEP</td>
<td>12</td>
</tr>
<tr>
<td>046E</td>
<td>Semi-tone table</td>
<td>14</td>
</tr>
</tbody>
</table>

### THE CASSETTE HANDLING ROUTINES

<table>
<thead>
<tr>
<th>address</th>
<th>routine</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>04C2</td>
<td>SA-BYTES</td>
<td>15</td>
</tr>
<tr>
<td>053F</td>
<td>SA/LD-RET</td>
<td>17</td>
</tr>
<tr>
<td>0556</td>
<td>LD-BYTES</td>
<td>17</td>
</tr>
<tr>
<td>05E3</td>
<td>LD-EDGE-2</td>
<td>20</td>
</tr>
<tr>
<td>0605</td>
<td>SAVE-ETC</td>
<td>21</td>
</tr>
<tr>
<td>07CB</td>
<td>VERIFY control</td>
<td>26</td>
</tr>
<tr>
<td>0802</td>
<td>Load a data block</td>
<td>26</td>
</tr>
<tr>
<td>0808</td>
<td>LOAD control</td>
<td>27</td>
</tr>
<tr>
<td>0886</td>
<td>MERGE control</td>
<td>29</td>
</tr>
<tr>
<td>0970</td>
<td>SAVE control</td>
<td>32</td>
</tr>
<tr>
<td>09A1</td>
<td>Cassette messages</td>
<td>32</td>
</tr>
</tbody>
</table>

### THE SCREEN & PRINTER HANDLING ROUTINES

<table>
<thead>
<tr>
<th>address</th>
<th>routine</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>09F4</td>
<td>PRINT-OUT</td>
<td>33</td>
</tr>
<tr>
<td>0A11</td>
<td>Control character table</td>
<td>33</td>
</tr>
<tr>
<td>0A23</td>
<td>Cursor left</td>
<td>33</td>
</tr>
<tr>
<td>0A3D</td>
<td>Cursor right</td>
<td>33</td>
</tr>
<tr>
<td>0A4F</td>
<td>Carriage return</td>
<td>34</td>
</tr>
<tr>
<td>0A5F</td>
<td>Print comma</td>
<td>34</td>
</tr>
<tr>
<td>0A69</td>
<td>Print a question mark</td>
<td>34</td>
</tr>
<tr>
<td>0A6D</td>
<td>Control characters with operands</td>
<td>34</td>
</tr>
<tr>
<td>0AD9</td>
<td>PO-ABLE</td>
<td>35</td>
</tr>
<tr>
<td>0ADC</td>
<td>Position store</td>
<td>36</td>
</tr>
<tr>
<td>0B03</td>
<td>Position fetch</td>
<td>36</td>
</tr>
<tr>
<td>0B24</td>
<td>Print any character</td>
<td>36</td>
</tr>
<tr>
<td>0B7F</td>
<td>Print all characters</td>
<td>37</td>
</tr>
<tr>
<td>0BDB</td>
<td>Set attribute byte</td>
<td>39</td>
</tr>
<tr>
<td>0C0A</td>
<td>Message printing</td>
<td>39</td>
</tr>
<tr>
<td>0C3B</td>
<td>PO-SAVE</td>
<td>40</td>
</tr>
<tr>
<td>0C41</td>
<td>Table search</td>
<td>40</td>
</tr>
<tr>
<td>address</td>
<td>routine</td>
<td>page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>0C55</td>
<td>Test for scroll</td>
<td>40</td>
</tr>
<tr>
<td>0CF8</td>
<td>'scroll?' message</td>
<td>42</td>
</tr>
<tr>
<td>0D4D</td>
<td>Temporary colour items</td>
<td>43</td>
</tr>
<tr>
<td>0D6B</td>
<td>CLS command</td>
<td>43</td>
</tr>
<tr>
<td>0DAF</td>
<td>Clearing the whole display area</td>
<td>44</td>
</tr>
<tr>
<td>0DD9</td>
<td>CL-SET</td>
<td>45</td>
</tr>
<tr>
<td>0DFE</td>
<td>Scrolling</td>
<td>45</td>
</tr>
<tr>
<td>0E44</td>
<td>Clear lines</td>
<td>46</td>
</tr>
<tr>
<td>0E8B</td>
<td>CL-ATTR</td>
<td>48</td>
</tr>
<tr>
<td>0E9B</td>
<td>CL-ADDR</td>
<td>48</td>
</tr>
<tr>
<td>0EAC</td>
<td>COPY command</td>
<td>48</td>
</tr>
<tr>
<td>0EC0</td>
<td>COPY-BUFF</td>
<td>49</td>
</tr>
<tr>
<td>0EF4</td>
<td>COPY-LINE</td>
<td>49</td>
</tr>
<tr>
<td>0F2C</td>
<td>EDITOR</td>
<td>50</td>
</tr>
<tr>
<td>0F81</td>
<td>ADD-CHAR</td>
<td>51</td>
</tr>
<tr>
<td>0FA0</td>
<td>Editing keys table</td>
<td>52</td>
</tr>
<tr>
<td>0FA9</td>
<td>EDIT key</td>
<td>52</td>
</tr>
<tr>
<td>0FF3</td>
<td>Cursor down editing</td>
<td>53</td>
</tr>
<tr>
<td>1007</td>
<td>Cursor left editing</td>
<td>53</td>
</tr>
<tr>
<td>100C</td>
<td>Cursor right editing</td>
<td>53</td>
</tr>
<tr>
<td>1015</td>
<td>DELETE editing</td>
<td>53</td>
</tr>
<tr>
<td>101E</td>
<td>ED-IGNORE</td>
<td>53</td>
</tr>
<tr>
<td>1024</td>
<td>ENTER editing</td>
<td>53</td>
</tr>
<tr>
<td>1031</td>
<td>ED-EDGE</td>
<td>53</td>
</tr>
<tr>
<td>1059</td>
<td>Cursor up editing</td>
<td>54</td>
</tr>
<tr>
<td>1076</td>
<td>ED-SYMBOL</td>
<td>54</td>
</tr>
<tr>
<td>107F</td>
<td>ED-ERROR</td>
<td>54</td>
</tr>
<tr>
<td>1097</td>
<td>CLEAR-SP</td>
<td>55</td>
</tr>
<tr>
<td>10A8</td>
<td>Keyboard input</td>
<td>55</td>
</tr>
<tr>
<td>111D</td>
<td>Lower screen copying</td>
<td>56</td>
</tr>
<tr>
<td>1190</td>
<td>SET-HL</td>
<td>57</td>
</tr>
<tr>
<td>11A7</td>
<td>REMOVE-FP</td>
<td>58</td>
</tr>
</tbody>
</table>

**THE EXECUTIVE ROUTINES**

<p>| 11B7    | NEW command                            | 59   |
| 11CB    | Main entry (Initialisation)            | 59   |
| 11DA    | RAM-CHECK                              | 59   |
| 12A2    | Main execution loop                    | 61   |
| 1391    | Report messages                        | 63   |
| 155D    | MAIN-ADD                               | 64   |
| 15AF    | Initial channel information            | 65   |
| 15C6    | Initial stream data                    | 65   |
| 15D4    | WAIT-KEY                               | 65   |
| 15E6    | INPUT-AD                               | 66   |
| 15EF    | Main printing                          | 66   |
| 1601    | CHAN-OPEN                              | 66   |
| 1615    | CHAN-FLAG                              | 67   |
| 162D    | Channel code look -up table            | 67   |
| 1634    | Channel K flag                         | 67   |
| 1642    | Channel S flag                         | 67   |
| 164D    | Channel P flag                         | 67   |
| 1652    | ONE-SPACE                              | 67   |
| 1655    | MAKE-ROOM                              | 67   |
| 1664    | POINTERS                               | 68   |
| 168F    | Collect a line number                  | 69   |
| 169E    | RESERVE                                | 69   |
| 16B0    | SET-MIN                                | 69   |
| 16D4    | Reclaim the edit-line                  | 70   |
| 16DB    | INDEXER                                | 70   |
| 16E5    | CLOSE # commend                        | 70   |</p>
<table>
<thead>
<tr>
<th>address</th>
<th>routine</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1716</td>
<td>CLOSE stream look-up table</td>
<td>71</td>
</tr>
<tr>
<td>171E</td>
<td>Stream data</td>
<td>71</td>
</tr>
<tr>
<td>1736</td>
<td>OPEN # command</td>
<td>71</td>
</tr>
<tr>
<td>177A</td>
<td>OPEN stream look-up table</td>
<td>72</td>
</tr>
<tr>
<td>1793</td>
<td>CAT, ERASE, FORMAT &amp; MOVE commands</td>
<td>73</td>
</tr>
<tr>
<td>1795</td>
<td>LIST &amp; LLIST commands</td>
<td>73</td>
</tr>
<tr>
<td>1795</td>
<td>AUTO-LIST</td>
<td>73</td>
</tr>
<tr>
<td>17F5</td>
<td>LLIST</td>
<td>74</td>
</tr>
<tr>
<td>17F9</td>
<td>LIST</td>
<td>74</td>
</tr>
<tr>
<td>1855</td>
<td>Print a whole BASIC line</td>
<td>75</td>
</tr>
<tr>
<td>1886</td>
<td>NUMBER</td>
<td>76</td>
</tr>
<tr>
<td>18C1</td>
<td>Print a flashing character</td>
<td>77</td>
</tr>
<tr>
<td>18E1</td>
<td>Print the cursor</td>
<td>77</td>
</tr>
<tr>
<td>190F</td>
<td>LN-FETCH</td>
<td>77</td>
</tr>
<tr>
<td>1925</td>
<td>Printing characters in a BASIC line</td>
<td>78</td>
</tr>
<tr>
<td>196E</td>
<td>LINE-ADDR</td>
<td>79</td>
</tr>
<tr>
<td>1980</td>
<td>Compare line numbers</td>
<td>79</td>
</tr>
<tr>
<td>1988</td>
<td>Find each statement</td>
<td>79</td>
</tr>
<tr>
<td>1988</td>
<td>NEXT-ONE</td>
<td>80</td>
</tr>
<tr>
<td>19DD</td>
<td>Difference</td>
<td>81</td>
</tr>
<tr>
<td>19E5</td>
<td>Reclaiming</td>
<td>81</td>
</tr>
<tr>
<td>19FB</td>
<td>E-LINE-NO</td>
<td>82</td>
</tr>
<tr>
<td>1A1B</td>
<td>Report and line number printing</td>
<td>82</td>
</tr>
</tbody>
</table>

**BASIC LINE AND COMMAND INTERPRETATION**

1A48    | Syntax tables | 84   |
1B17    | Main parser (BASIC interpreter) | 86   |
1B28    | Statement loop | 87   |
1B52    | SCAN-LOOP | 87   |
1B6F    | SEPARATOR | 88   |
1B76    | STMT-RET | 88   |
1B8A    | LINE-RUN | 88   |
1B9E    | LINE-NEW | 88   |
1BB2    | REM command | 89   |
1BB3    | LINE-END | 89   |
1BBF    | LINE-USE | 89   |
1BD1    | NEXT-LINE | 89   |
1BEE    | CHECK-END | 90   |
1BF4    | STMT-NEXT | 90   |
1C01    | Command class table | 90   |
1C0D    | Command classes - 00, 03 & 05 | 90   |
1C16    | JUMP-C-R | 91   |
1C1F    | Command classes - 01, 02 & 04 | 91   |
1C22    | Variable In assignment | 91   |
1C56    | Fetch a value | 92   |
1C79    | Expect numeric/string expressions | 93   |
1C96    | Set permanent colours (class 07) | 93   |
1CBE    | Command class - 09 | 94   |
1CDB    | Command class - 0B | 94   |
1CDE    | Fetch a number | 94   |
1CEE    | STOP command | 95   |
1CF9    | IF command | 95   |
1D03    | FOR command | 95   |
1D86    | LOOK-PROG | 96   |
1DAB    | NEXT command | 97   |
1DDA    | NEXT-LOOP | 97   |
1DEC    | READ command | 99   |
1E27    | DATA command | 100  |
1E39    | PASS-BY | 100  |
1E42    | RESTORE command | 100  |
<table>
<thead>
<tr>
<th>address</th>
<th>routine</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E4F</td>
<td>RANDOMIZE command</td>
<td>100</td>
</tr>
<tr>
<td>1E5F</td>
<td>CONTINUE command</td>
<td>101</td>
</tr>
<tr>
<td>1E67</td>
<td>GO TO command</td>
<td>101</td>
</tr>
<tr>
<td>1E7A</td>
<td>OUT command</td>
<td>101</td>
</tr>
<tr>
<td>1E80</td>
<td>POKE command</td>
<td>101</td>
</tr>
<tr>
<td>1E85</td>
<td>TWO-PARAM</td>
<td>101</td>
</tr>
<tr>
<td>1E94</td>
<td>Find integers</td>
<td>101</td>
</tr>
<tr>
<td>1EA1</td>
<td>RUN command</td>
<td>102</td>
</tr>
<tr>
<td>1EAC</td>
<td>CLEAR command</td>
<td>102</td>
</tr>
<tr>
<td>1EED</td>
<td>GO SUB command</td>
<td>103</td>
</tr>
<tr>
<td>1F05</td>
<td>TEST-ROOM</td>
<td>103</td>
</tr>
<tr>
<td>1F1A</td>
<td>Free memory</td>
<td>103</td>
</tr>
<tr>
<td>1F23</td>
<td>RETURN command</td>
<td>104</td>
</tr>
<tr>
<td>1F3A</td>
<td>PAUSE command</td>
<td>104</td>
</tr>
<tr>
<td>1F4</td>
<td>BREAK-KEY</td>
<td>104</td>
</tr>
<tr>
<td>1F60</td>
<td>DEF FN command</td>
<td>105</td>
</tr>
<tr>
<td>1FC3</td>
<td>UNSTACK-Z</td>
<td>106</td>
</tr>
<tr>
<td>1FC9</td>
<td>LPRINT command</td>
<td>106</td>
</tr>
<tr>
<td>1FCF</td>
<td>PRINT command</td>
<td>106</td>
</tr>
<tr>
<td>1FF5</td>
<td>Print a carriage return</td>
<td>107</td>
</tr>
<tr>
<td>1FFC</td>
<td>Print items</td>
<td>107</td>
</tr>
<tr>
<td>2045</td>
<td>End of printing</td>
<td>108</td>
</tr>
<tr>
<td>204E</td>
<td>Print position</td>
<td>108</td>
</tr>
<tr>
<td>2070</td>
<td>Alter stream</td>
<td>108</td>
</tr>
<tr>
<td>2089</td>
<td>INPUT command</td>
<td>109</td>
</tr>
<tr>
<td>21B9</td>
<td>IN-ASSIGN</td>
<td>110</td>
</tr>
<tr>
<td>21D6</td>
<td>IN-CHAN-K</td>
<td>112</td>
</tr>
<tr>
<td>21E1</td>
<td>Colour item routines</td>
<td>112</td>
</tr>
<tr>
<td>226C</td>
<td>CO-CHANGE</td>
<td>114</td>
</tr>
<tr>
<td>2294</td>
<td>BORDER command</td>
<td>115</td>
</tr>
<tr>
<td>22AA</td>
<td>Pixel address</td>
<td>115</td>
</tr>
<tr>
<td>22BC</td>
<td>Point</td>
<td>116</td>
</tr>
<tr>
<td>22DC</td>
<td>PLOT command</td>
<td>116</td>
</tr>
<tr>
<td>2307</td>
<td>STK-TO-BC</td>
<td>117</td>
</tr>
<tr>
<td>2314</td>
<td>STK-TO-A</td>
<td>117</td>
</tr>
<tr>
<td>2320</td>
<td>CIRCLE command</td>
<td>117</td>
</tr>
<tr>
<td>2382</td>
<td>DRAW command</td>
<td>119</td>
</tr>
<tr>
<td>247D</td>
<td>Initial parameters</td>
<td>123</td>
</tr>
<tr>
<td>24B7</td>
<td>Line drawing</td>
<td>124</td>
</tr>
<tr>
<td>24FB</td>
<td>SCANNING</td>
<td>127</td>
</tr>
<tr>
<td>2530</td>
<td>SYNTAX-Z</td>
<td>128</td>
</tr>
<tr>
<td>2535</td>
<td>Scanning SCREEN$</td>
<td>128</td>
</tr>
<tr>
<td>2580</td>
<td>Scanning ATTR</td>
<td>129</td>
</tr>
<tr>
<td>2596</td>
<td>Scanning function table</td>
<td>129</td>
</tr>
<tr>
<td>25AF</td>
<td>Scanning function routines</td>
<td>130</td>
</tr>
<tr>
<td>26C9</td>
<td>Scanning variable routine</td>
<td>133</td>
</tr>
<tr>
<td>2734</td>
<td>Scanning main loop</td>
<td>135</td>
</tr>
<tr>
<td>2795</td>
<td>Table of operators</td>
<td>137</td>
</tr>
<tr>
<td>27B0</td>
<td>Table of priorities</td>
<td>137</td>
</tr>
<tr>
<td>27BD</td>
<td>Scanning function (FN)</td>
<td>137</td>
</tr>
<tr>
<td>28AB</td>
<td>FN-SKPOVR</td>
<td>140</td>
</tr>
<tr>
<td>28B2</td>
<td>LOOK-VARS</td>
<td>141</td>
</tr>
<tr>
<td>2951</td>
<td>Stack function argument</td>
<td>144</td>
</tr>
<tr>
<td>2996</td>
<td>STK-VAR</td>
<td>145</td>
</tr>
<tr>
<td>2A52</td>
<td>SLICING</td>
<td>148</td>
</tr>
<tr>
<td>2AB6</td>
<td>STK-STORE</td>
<td>150</td>
</tr>
<tr>
<td>2ACC</td>
<td>INT-EXP</td>
<td>150</td>
</tr>
<tr>
<td>2AEE</td>
<td>DE,(DE+1)</td>
<td>151</td>
</tr>
</tbody>
</table>

**EXPRESSION EVALUATION**

<table>
<thead>
<tr>
<th>address</th>
<th>routine</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>24FB</td>
<td>SCANNING</td>
<td>127</td>
</tr>
<tr>
<td>2530</td>
<td>SYNTAX-Z</td>
<td>128</td>
</tr>
<tr>
<td>2535</td>
<td>Scanning SCREEN$</td>
<td>128</td>
</tr>
<tr>
<td>2580</td>
<td>Scanning ATTR</td>
<td>129</td>
</tr>
<tr>
<td>2596</td>
<td>Scanning function table</td>
<td>129</td>
</tr>
<tr>
<td>25AF</td>
<td>Scanning function routines</td>
<td>130</td>
</tr>
<tr>
<td>26C9</td>
<td>Scanning variable routine</td>
<td>133</td>
</tr>
<tr>
<td>2734</td>
<td>Scanning main loop</td>
<td>135</td>
</tr>
<tr>
<td>2795</td>
<td>Table of operators</td>
<td>137</td>
</tr>
<tr>
<td>27B0</td>
<td>Table of priorities</td>
<td>137</td>
</tr>
<tr>
<td>27BD</td>
<td>Scanning function (FN)</td>
<td>137</td>
</tr>
<tr>
<td>28AB</td>
<td>FN-SKPOVR</td>
<td>140</td>
</tr>
<tr>
<td>28B2</td>
<td>LOOK-VARS</td>
<td>141</td>
</tr>
<tr>
<td>2951</td>
<td>Stack function argument</td>
<td>144</td>
</tr>
<tr>
<td>2996</td>
<td>STK-VAR</td>
<td>145</td>
</tr>
<tr>
<td>2A52</td>
<td>SLICING</td>
<td>148</td>
</tr>
<tr>
<td>2AB6</td>
<td>STK-STORE</td>
<td>150</td>
</tr>
<tr>
<td>2ACC</td>
<td>INT-EXP</td>
<td>150</td>
</tr>
<tr>
<td>2AEE</td>
<td>DE,(DE+1)</td>
<td>151</td>
</tr>
<tr>
<td>Address</td>
<td>Routine</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>2AF4</td>
<td>GET-HL*DE</td>
<td>151</td>
</tr>
<tr>
<td>2AFF</td>
<td>LET command</td>
<td>151</td>
</tr>
<tr>
<td>2BF1</td>
<td>STK-FETCH</td>
<td>157</td>
</tr>
<tr>
<td>2C02</td>
<td>DIM command</td>
<td>157</td>
</tr>
<tr>
<td>2C88</td>
<td>ALPHANUM</td>
<td>159</td>
</tr>
<tr>
<td>2C8D</td>
<td>ALPHA</td>
<td>159</td>
</tr>
<tr>
<td>2C9B</td>
<td>Decimal to floating-point</td>
<td>160</td>
</tr>
<tr>
<td>2D1B</td>
<td>NUMERIC</td>
<td>161</td>
</tr>
<tr>
<td>2D22</td>
<td>STK-DIGIT</td>
<td>162</td>
</tr>
<tr>
<td>2D28</td>
<td>STACK-A</td>
<td>162</td>
</tr>
<tr>
<td>2D2B</td>
<td>STACK-BC</td>
<td>162</td>
</tr>
<tr>
<td>2D3B</td>
<td>Integer to floating-point</td>
<td>162</td>
</tr>
</tbody>
</table>

**THE ARITHMETIC ROUTINES**

<table>
<thead>
<tr>
<th>Address</th>
<th>Routine</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D4F</td>
<td>E-format to floating-point</td>
<td>164</td>
</tr>
<tr>
<td>2D7F</td>
<td>INT-FETCH</td>
<td>165</td>
</tr>
<tr>
<td>2D8E</td>
<td>INT-STORE</td>
<td>165</td>
</tr>
<tr>
<td>2DA2</td>
<td>Floating-point to BC</td>
<td>166</td>
</tr>
<tr>
<td>2DC1</td>
<td>LOG (2^A)</td>
<td>166</td>
</tr>
<tr>
<td>2DD5</td>
<td>Floating-point to A</td>
<td>167</td>
</tr>
<tr>
<td>2DE3</td>
<td>Print a floating-point number</td>
<td>167</td>
</tr>
<tr>
<td>2F8B</td>
<td>CA=10^A+C</td>
<td>173</td>
</tr>
<tr>
<td>2F9B</td>
<td>Prepare to add</td>
<td>174</td>
</tr>
<tr>
<td>2FBA</td>
<td>Fetch two numbers</td>
<td>174</td>
</tr>
<tr>
<td>2FD2</td>
<td>Shift addend</td>
<td>175</td>
</tr>
<tr>
<td>3004</td>
<td>ADD-BACK</td>
<td>176</td>
</tr>
<tr>
<td>300F</td>
<td>Subtraction (03)</td>
<td>176</td>
</tr>
<tr>
<td>3014</td>
<td>Addition (0F)</td>
<td>176</td>
</tr>
<tr>
<td>30A9</td>
<td>HL=HL*DE</td>
<td>179</td>
</tr>
<tr>
<td>33C0</td>
<td>Prepare to multiply or divide</td>
<td>180</td>
</tr>
<tr>
<td>33CA</td>
<td>Multiplication (04)</td>
<td>180</td>
</tr>
<tr>
<td>33AF</td>
<td>Division (05)</td>
<td>184</td>
</tr>
<tr>
<td>3214</td>
<td>Integer truncation towards zero (3A)</td>
<td>186</td>
</tr>
<tr>
<td>3293</td>
<td>Re-stack two</td>
<td>188</td>
</tr>
<tr>
<td>3297</td>
<td>RE-STACK (3D)</td>
<td>188</td>
</tr>
</tbody>
</table>

**THE FLOATING-POINT CALCULATOR**

<table>
<thead>
<tr>
<th>Address</th>
<th>Routine</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>32C5</td>
<td>Table of constants</td>
<td>190</td>
</tr>
<tr>
<td>32D7</td>
<td>Table of addresses</td>
<td>190</td>
</tr>
<tr>
<td>335B</td>
<td>CALCULATE</td>
<td>192</td>
</tr>
<tr>
<td>33A1</td>
<td>Delete (02)</td>
<td>194</td>
</tr>
<tr>
<td>33A2</td>
<td>Single operation (3B)</td>
<td>194</td>
</tr>
<tr>
<td>33A9</td>
<td>Test 5-spaces</td>
<td>194</td>
</tr>
<tr>
<td>33B4</td>
<td>Stack number</td>
<td>194</td>
</tr>
<tr>
<td>33C0</td>
<td>Move a floating-point number (31)</td>
<td>195</td>
</tr>
<tr>
<td>33C6</td>
<td>Stack literals (34)</td>
<td>195</td>
</tr>
<tr>
<td>33F7</td>
<td>Skip constants</td>
<td>196</td>
</tr>
<tr>
<td>3406</td>
<td>Memory location</td>
<td>196</td>
</tr>
<tr>
<td>340F</td>
<td>Get from memory area (E0 etc.)</td>
<td>197</td>
</tr>
<tr>
<td>341B</td>
<td>Stack a constant (A0 etc.)</td>
<td>197</td>
</tr>
<tr>
<td>342D</td>
<td>Store in memory area (C0 etc.)</td>
<td>197</td>
</tr>
<tr>
<td>343C</td>
<td>EXCHANGE (01)</td>
<td>198</td>
</tr>
<tr>
<td>3449</td>
<td>Series generator (86 etc.)</td>
<td>198</td>
</tr>
<tr>
<td>346A</td>
<td>Absolute magnitude (2A)</td>
<td>199</td>
</tr>
<tr>
<td>346E</td>
<td>Unary minus (1B)</td>
<td>199</td>
</tr>
<tr>
<td>3492</td>
<td>Signum (29)</td>
<td>200</td>
</tr>
<tr>
<td>34A5</td>
<td>IN (2C)</td>
<td>200</td>
</tr>
<tr>
<td>34AC</td>
<td>PEEK (2B)</td>
<td>201</td>
</tr>
<tr>
<td>34B3</td>
<td>USR number (2D)</td>
<td>201</td>
</tr>
<tr>
<td>34BC</td>
<td>USR string (19)</td>
<td>201</td>
</tr>
<tr>
<td>address</td>
<td>routine</td>
<td>page</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>34E9</td>
<td>TEST-ZERO</td>
<td>202</td>
</tr>
<tr>
<td>34F9</td>
<td>Greater than zero (37)</td>
<td>202</td>
</tr>
<tr>
<td>3501</td>
<td>NOT (30)</td>
<td>202</td>
</tr>
<tr>
<td>3506</td>
<td>Less than zero (36)</td>
<td>203</td>
</tr>
<tr>
<td>350B</td>
<td>Zero or one</td>
<td>203</td>
</tr>
<tr>
<td>351B</td>
<td>OR (07)</td>
<td>203</td>
</tr>
<tr>
<td>3524</td>
<td>Number AND number (08)</td>
<td>203</td>
</tr>
<tr>
<td>352D</td>
<td>String AND number (10)</td>
<td>204</td>
</tr>
<tr>
<td>353B</td>
<td>Comparison (09-0E, 11-16)</td>
<td>204</td>
</tr>
<tr>
<td>359C</td>
<td>String concatenation (17)</td>
<td>205</td>
</tr>
<tr>
<td>35BF</td>
<td>STK-PNTRS</td>
<td>206</td>
</tr>
<tr>
<td>35C9</td>
<td>CHR$ (2F)</td>
<td>206</td>
</tr>
<tr>
<td>35DE</td>
<td>VAL and VAL$ (1D,18)</td>
<td>207</td>
</tr>
<tr>
<td>361F</td>
<td>STRS (2E)</td>
<td>208</td>
</tr>
<tr>
<td>3645</td>
<td>Read-in (1A)</td>
<td>208</td>
</tr>
<tr>
<td>3669</td>
<td>CODE (1C)</td>
<td>209</td>
</tr>
<tr>
<td>3674</td>
<td>LEN (1E)</td>
<td>209</td>
</tr>
<tr>
<td>367A</td>
<td>Decrease the counter (35)</td>
<td>209</td>
</tr>
<tr>
<td>3686</td>
<td>Jump (33)</td>
<td>209</td>
</tr>
<tr>
<td>368F</td>
<td>Jump on true (00)</td>
<td>210</td>
</tr>
<tr>
<td>369B</td>
<td>END-CALC (38)</td>
<td>210</td>
</tr>
<tr>
<td>36A0</td>
<td>Modulus (32)</td>
<td>210</td>
</tr>
<tr>
<td>36AF</td>
<td>INT (27)</td>
<td>211</td>
</tr>
<tr>
<td>36C4</td>
<td>Exponential (26)</td>
<td>211</td>
</tr>
<tr>
<td>3713</td>
<td>Natural logarithm (25)</td>
<td>213</td>
</tr>
<tr>
<td>3783</td>
<td>Reduce argument (39)</td>
<td>215</td>
</tr>
<tr>
<td>37AA</td>
<td>Cosine (20)</td>
<td>216</td>
</tr>
<tr>
<td>37B5</td>
<td>SINE (1 F)</td>
<td>216</td>
</tr>
<tr>
<td>37DA</td>
<td>Tan (21)</td>
<td>217</td>
</tr>
<tr>
<td>37E2</td>
<td>ARCTAN (24)</td>
<td>218</td>
</tr>
<tr>
<td>3833</td>
<td>Arcsin (22)</td>
<td>219</td>
</tr>
<tr>
<td>3843</td>
<td>Arccos (23)</td>
<td>220</td>
</tr>
<tr>
<td>384A</td>
<td>Square root (28)</td>
<td>220</td>
</tr>
<tr>
<td>3851</td>
<td>Exponentiation (06)</td>
<td>220</td>
</tr>
</tbody>
</table>

**APPENDIX**

BASIC programs for the main series.
- Series generator 222
- SIN X 223
- EXP X 224
- LN X 226
- ATN X 228

The 'DRAW' algorithm 228
The 'CIRCLE' algorithm 229

Note on Small Integers and -65536 229
Spectrum ROM Disassembly

Customer Registration Card

Please fill out this page (or a photocopy of it) and return it so that we may keep you informed of new books, software and special offers. Post to the appropriate address on the back.

Date.................. 19......

Name..........................................................................

Street & No..............................................................

City............................Postcode/Zipcode...............  

Model of computer owned.................................

Where did you learn of this book:

O FRIEND     O RETAIL SHOP

O MAGAZINE (give name) ...........................

O OTHER (specify) ...........................

Age?  O 10-15  O 16-19  O 20-24  O 25 and over

How would you rate this book?

QUALITY:  O Excellent  O Good  O Poor

VALUE:  O Overpriced  O Good  O Underpriced

What other books and software would you like to see produced for your computer?

.....................................................................................

.....................................................................................

.....................................................................................

EDITION 7 6 5 4 3 2 1
Melbourne House addresses

Put this Registration Card (or photocopy) in an envelope and post it to the appropriate address:

United Kingdom
Melbourne House (Publishers) Ltd
Castle Yard House
Castle Yard
Richmond, TW10 6TF

Australia and New Zealand
Melbourne House (Australia) Pty Ltd
2nd Floor, 70 Park Street
South Melbourne, Victoria 3205