The Complete Timex TS1000/Sinclair ZX81 ROM Disassembly

Includes PART A: 0000H-0F54H &
PART B: 0F55H-1DFFH

by Dr. Ian Logan
& Dr. Frank O'Hara

ROM DISASSEMBLY

The Complete Timex TS1000 & Sinclair ZX81 ROM Disassembly

by Dr. Ian Logan
& Dr. Frank O'Hara

Due to popular demand Melbourne House Publishers have combined "ROM Disassembly Part A" and "ROM Disassembly Part B" into one accessible volume. PART A starts page 1 through to page 30, PART B begins again at page 1 (2 pages after page 30) through to page 82.

Published in the United Kingdom by
Melbourne House (Publishers) Ltd.,
Glebe Cottage, Glebe House,
Station Road, Cheddington,
Leighton Buzzard, Bedfordshire. LU7 7NA

Published in Australia by
Melbourne House (Australia) Pty. Ltd.,
Suite 4, 75 Palmerston Crescent,
South Melbourne, Victoria 3205.

Published in the United States of America by
Melbourne House Software Inc.,
347 Reedwood Drive, Nashville, TN 37217

The Complete Timex TS1000 & Sinclair ZX81 ROM Disassembly PART A: 0000H-0F54H

CONTENTS

PAGE

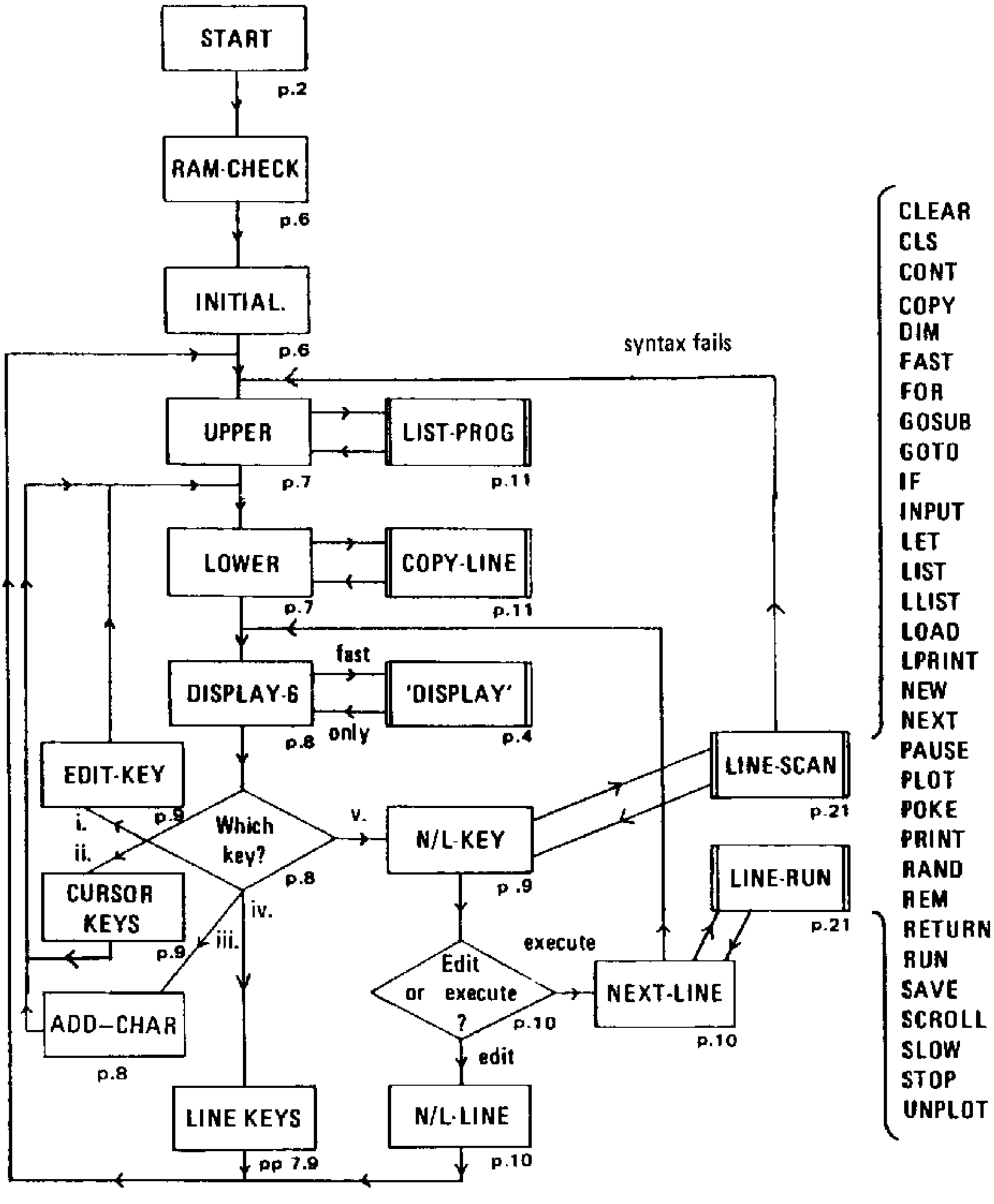
Melbourne House (Publishers) Ltd. ISBN 0 86161 113 6
National Library of Australia Card Number and
ISBN 0 86759 124 2

Note: This book should be read in conjunction with 'ZX81 BASIC PROGRAMMING' issued by Sinclair Research Ltd. with every ZX81.

The system variable 'labels' are those given by Sinclair Research Ltd., but the other 'labels' have been designated by the authors.

Printed in Hong Kong by Colorcraft Ltd.

The 'FLOW DIAGRAM' for PART A of the 8K ROM Program



NOTE: The 'display' in SLOW mode is produced by a call to 'DISPLAY' every 1/50th. of a second.

THE 'START'

The NMI generator is turned off and BC set to the 'top of possible RAM'

```
0000 START          OUT     (+FD),A
                    LD      BC,+7FFF
                    JP      03CB,RAM-CHECK
```

THE 'ERROR'RESTART

```
0008 ERROR-1        LD      HL,(CH-ADD)
                    LD      (X-PTR),HL
                    JR      0056,ERROR-2
```

THE 'PRINT A CHARACTER' RESTART

The code of the character to be printed is in the A register.

```
0010 PRINT-A        AND     A
                    JP      NZ,07F1,PRINT-CH
                    JP      07F5,PRINT-SP.
                    DEFB    +FF
```

THE 'COLLECT CHARACTER' RESTART

The A register is set with the character addressed by CH-ADD. Spaces are ignored.

```
0018 GET-CH.        LD      HL,(CH-ADD)
                    LD      A,(HL)
001C TEST-SP.       AND     A
                    RET     NZ
                    NOP
                    NOP
```

THE 'COLLECT NEXT CHARACTER' RESTART

CH-ADD is incremented before the character is fetched.

```
0020 NEXT-CH        CALL    0049,CH-ADD+1
                    JR      001C,TEST-SP.
                    DEFB    +FF
                    DEFB    +FF
                    DEFB    +FF
```

THE 'FP-CALCULATOR' RESTART

A direct jump is made to the calculator routine.

```
0028 FP-CALC.       JP      199D,CALCULATE
```

THE 'END-CALC' SUBROUTINE

The byte 34 ends a RST 0028 operation.

```
002B END-CALC.      POP     AF
                    EXX
                    EX      (SP),HL
                    EXX
                    RET
```

THE 'MAKE BC SPACES' RESTART

BC spaces are made available for different purposes.

```
0030 BC-SPACES      PUSH    BC
                    LD      HL,(E-LINE)
                    PUSH    HL
                    JP      1488,RESERVE
```

THE 'INTERRUPT' RESTART

B holds the line number and C the number of the scan line.

```
0038 INTERRUPT      DEC     C
                    JR      NZ,0045,SCAN-LINE
                    POP     HL
                    DEC     B
                    RET     Z
                    SET     3,C
0041 WAIT-INT.      LD      R,A
                    EI
                    JP      (HL)
0045 SCAN-LINE      POP     DE
                    RET     Z
                    JR      0041,WAIT-INT.
```

THE 'INCREMENT CH-ADD' SUBROUTINE

The pointer CH-ADD is incremented and the cursor ignored.

```
0049 CH-ADD+1       LD      HL,(CH-ADD)
004C CURSOR-SO      INC     HL
004D TEMP-PTR.      LD      (CH-ADD),HL
                    LD      A,(HL)
                    CP      +7F
                    RET     NZ
                    JR      004C,CURSOR-SO
```

THE 'ERROR-2' ROUTINE

L is loaded with the 'data byte'.

```
0056 ERROR-2        POP     HL
                    LD      L,(HL)
0058 ERROR-3        LD      (ERR-NR),L
                    LD      SP,(ERR-SP)
                    CALL    0207,SLOW/FAST
                    JP      14BC,SET-MEM
                    DEFB    +FF
```

THE 'NMI' ROUTINE

This routine is entered whenever a 'SLOW' NM interrupt occurs.

```
0066 NMI            EX      AF,A'F'
                    INC     A
                    JP      M,006D NMI-RET
                    JR      Z,006F NMI-CONT.
006D NMI-RET        EX      AF, A'F'
                    RET
```

THE PREPARE FOR 'SLOW' DISPLAY ROUTINE

The main registers are preserved on the stack, the NMI generator is switched off and a jump is made into the display routine. (IX holding 0281 or 028F)

```
006F NMI-CONT  EX     AF,A'F'
               PUSH   AF
               PUSH   BC
               PUSH   DE
               PUSH   HL
               LD     HL,(D-FILE)
               SET    7,H
               HALT
               OUT    (+FD),A
               JP     (IX)
```

THE KEY TABLES

The 'unshifted' character codes.

007E	3F	3D	28	3B	Z X C V
0082	26	38	29	2B	A S D F
0086	2C	36	3C	2A	G Q W E
008A	37	39	1D	1E	R T 1 2
008E	1F	20	21	1C	3 4 5 0
0092	25	24	23	22	9 8 7 6
0096	35	34	2E	3A	P O I U
009A	3E	76	31	30	Y N/L L K
009E	2F	2D	00	1B	J H Sp. .
00A2	32	33	27		M N B

The 'shifted' character codes.

00A5	OE	19	OF	18	: ; ? /
00A9	E3	E1			STOP LPRINT
00AB	E4	E5			SLOW FAST
00AD	E2	CO	D9		LLIST "" OR
00BO	EO	DB	DD		STEP <= <>
00B3	75	DA	DE		EDIT AND THEN
00B6	DF	72	77		TO ← RUBOUT
00B9	74	73	70		GRAPHICS → ←
00BC	71	OB	11	10	") (
00CO	OD	DC	79		$ = FUNCTION
00C3	14	15	16	D8	= + − **
00C7	OC	1A	12	13	£ , > <
00CB	17				*

The 'function' character codes.

00CC	CD	CE	C1	78	LN EXP AT K/L
00DO	CA	CB	CC	D1	ASN ACS ATN SGN
00D4	D2	C7	C8	C9	ABS SIN COS TAN
00D8	CF	40	78	78	INT RND K/L K/L
00DC	78	78	78	78	K/L K/L K/L K/L
00EO	78	78	78	78	K/L K/L K/L K/L
00E4	C2	D3	C4		TAB PEEK CODE
00E7	D6	D5	78		CHR$ STR$ K/L
00EA	D4	C6	C5		USR LEN VAL
00ED	DO	78	78		SQR K/L K/L
00FO	42	D7	41		PI NOT INKEY$

The 'graphic' character codes.

00F3	08	0A	09	8A	
00F7	89	81	82	07	
00FB	84	06	01	02	
00FF	87	04	05	77	RUBOUT
0103	78	85	03	83	K/L
0107	8B	91	90	8D	
010B	86	78	92	95	K/L
010F	96	88			

The 'token' tables.

0111	8F	0B	8B	26	'?' " '"' A
0115	B9	39	26	A7	'T' T A 'B'
0119	8F	28	34	29	'?' C O D
011D	AA	3B	26	B1	'E' V A 'L'
0121	31	2A	B3	38	L E 'N' S
0125	2E	B3	28	34	I 'N' C O
0129	B8	39	26	B3	'S' T A 'N'
012D	26	38	B3	26	A S 'N' A
0131	28	B8	26	39	C 'S' A T
0135	B3	31	B3	2A	'N' L 'N' E
0139	3D	B5	2E	33	X 'P' I N
013D	B9	38	36	B7	'T' S Q 'R'
0141	38	2C	B3	26	S G 'N' A
0145	27	B8	35	2A	B 'S' P E
0149	2A	B0	3A	38	E 'K' U S
014D	B7	38	39	37	'R' S T R
0151	8D	28	2D	37	'$' C H R
0155	8D	33	34	B9	'$' N O T
0159	17	97	34	B7	* '*' O 'R'
015D	26	33	A9	13	A N 'D' <
0161	94	12	94	13	'=' > '=' <
0165	92	39	2D	2A	'>' T H E
0169	B3	39	B4	38	'N' T 'O' S
016D	39	2A	B5	31	T E 'P' L
0171	35	37	2E	33	P R I N
0175	B9	31	31	2E	'T' L L I
0179	38	B9	38	39	S 'T' S T
017D	34	B5	38	31	O 'P' S L
0181	34	BC	2B	26	O 'W' F A
0185	38	B9	33	2A	S 'T' N E
0189	BC	38	28	37	'W' S C R
018D	34	31	B1	28	O L 'L' C
0191	34	33	B9	29	O N 'T' D
0195	2E	B2	37	2A	I 'M' R E
0199	B2	2B	34	B7	'M' F O 'R'
019D	2C	34	39	B4	G O T 'O'
01A1	2C	34	38	3A	G O S U
01A5	A7	2E	33	35	'B' I N P
01A9	3A	B9	31	34	U 'T' L O
01AD	26	A9	31	2E	A 'D' L I
01B1	38	B9	31	2A	S 'T' L E
01B5	B9	35	26	3A	'T' P A U
01B9	38	AA	33	2A	S 'E' N E
01BD	3D	B9	35	34	X 'T' P O
01C1	30	AA	35	37	K 'E' P R
01C5	2E	33	B9	35	I N 'T' P
01C9	31	34	B9	37	L O 'T' R
01CD	3A	B3	38	26	U 'N' S A
01D1	3B	AA	37	26	V 'E' R A
01D5	33	A9	2E	AB	N 'D' I F
01D9	28	31	B8	3A	C L 'S' U
01DD	33	35	31	34	N P L O
01E1	B9	28	31	2A	'T' C L E
01E5	26	B7	37	2A	A 'R' R E
01E9	39	3A	37	B3	T U R 'N'
01ED	28	34	35	BE	C O P 'Y'
01F1	37	33	A9	2E	R N 'D' I
01F5	33	30	2A	3E	N K E Y
01F9	8D	35	AE		'$' P 'I'

THE 'LOAD/SAVE UPDATE' SUBROUTINE

HL is incremented until it matches the current value in 'E-LINE'.

```
01FC LOAD/SAVE INC     HL
               EX      DE,HL
               LD      HL,(E-LINE)
               SCF
               SBC     HL,DE
               EX      DE,HL
               RET     NC
               POP     HL
```

THE DISPLAY ROUTINES:

i) Test for SLOW or FAST Mode.

The SLOW flag, Bit 6 of CDFLAG is tested, and a return is made if the program is in FAST mode or 'SLOW' display is not available.

```
0207 SLOW/FAST LD      HL,+CDFLAG
               LD      A,(HL)
               RLA
               XOR     (HL)
               RLA
               RET     NC
               LD      A,+7F
               EX      AF,A'F'
               LD      B,+11
               OUT     (+FE),A
0216 LOOP-11   DJNZ    0216,LOOP-11
               OUT     (+FD),A
               EX      AF,A'F'
               RLA
               JR      NC,0226,NO-SLOW
               SET     7,(HL)
               PUSH    AF
               PUSH    BC
               PUSH    DE
               PUSH    HL
               JR      0229,DISPLAY-1
0226 NO-SLOW   RES     6,(HL)
               RET
```

11) The main display routine.

The frame counter is first collected and decremented. A return is made if the frame counter reaches zero.

```
0229 DISPLAY-1 LD      HL,(FRAMES)
               DEC     HL
022D DISPLAY-P LD      A,+7F
               AND     H
               OR      L
               LD      A,H
               JR      NZ,0237,ANOTHER
               RLA
               JR      0239,OVER-NC
0237 ANOTHER   LD      B,(HL)
               SCF
0239 OVER-NC   LD      H,A
               LD      (FRAMES),HL
               RET     NC
```

The keyboard is now read, and a return is made if a new key has been pressed. Otherwise a display is produced.

```
023E DISPLAY-2 CALL    02BB,KEYBOARD
               LD      BC,(LAST-K)
               LD      (LAST-K),HL
               LD      A,B
               ADD     A,+02
               SBC     HL,BC
               LD      A,(DEBOUNCE)
               OR      H
               OR      L
               LD      E,B
               LD      B,+0B
               LD      HL,+CDFLAG
               RES     O,(HL)
               JR      NZ,0264,NO-KEY
               BIT     7,(HL)
               SET     0,(HL)
               RET     Z
               DEC     B
               NOP
               SCF
0264 NO-KEY    LD      HL,+DEBOUNCE
               CCF
               RL      B
026A LOOP-B    DJNZ    026A,LOOP-B
               LD      B,(HL)
               LD      A,E
               CP      +FE
               SBC     A,A
               LD      B,+1F
               OR      (HL)
               AND     B
               RRA
               LD      (HL),A
               OUT     (+FF),A
               LD      HL,(D-FILE)
               SET     7,H
               CALL    0292,DISPLAY-3
               LD      A,R
               LD      BC,+1901
               LD      A,+F5
               CALL    02B5,DISPLAY-5
               DEC     HL
               CALL    0292,DISPLAY-3
               JP      0229,DISPLAY-1
```

The IX register is loaded with the 'return' address, and the main registers are restored after a 'slow' display.

```
0292 DISPLAY-3 POP     IX
               LD      C,(MARGIN)
               BIT     7,(CDFLAG)
               JR      Z,02A9,DISPLAY-4
               LD      A,C
               NEG
               INC     A
               EX      AF,A'F'
               OUT     (+FE),A
               POP     HL
               POP     DE
               POP     BC
               POP     AF
               RET
```

Sets up the A and B registers for the display.

```
02A9 DISPLAY-4   LD     A,+FC
                 LD     B,+01
                 CALL   02B5,DISPLAY-5
                 DEC    HL
                 EX     (SP),HL
                 EX     (SP),HL
                 JP     (IX)
```

Sets up the refresh register and waits for an interrupt.

```
02B5 DISPLAY-5   LD     R,A
                 LD     A,+DD
                 EI
                 JP     (HL)
```

THE 'KEYBOARD SCANNING' SUBROUTINE

The keyboard is scanned eight times and the result built up in the HL register pair. MARGIN is also determined.

```
02BB KEYBOARD    LD     HL,+FFFF
                 LD     BC,+FEFE
                 IN     A,(C)
                 OR     +01
02C5 EACH-LINE   OR     +E0
                 LD     D,A
                 CPL
                 CP     +01
                 SBC    A,A
                 OR     B
                 AND    L
                 LD     L,A
                 LD     A,H
                 AND    D
                 LD     H,A
                 RLC    B
                 IN     A,(C)
                 JR     C,02C5,EACH-LINE
                 RRA
                 RL     H
                 RLA
                 RLA
                 RLA
                 SBC    A,A
                 AND    +18
                 ADD    A,+1F
                 LD     (MARGIN),A
                 RET
```

THE 'SET FAST MODE' SUBROUTINE

The NMI generator is turned off and bit 7 of CDFLAG is RESET. Bit 6 will remain SET if the overall mode is SLOW, i.e. in PAUSE.

```
02E7 SET-FAST    BIT    7,(CDFLAG)
                 RET    Z
                 HALT
                 OUT    (+FD),A
                 RES    7,(CDFLAG)
                 RET
```

REPORT F — No name supplied.

```
02F4 REPORT-F    RST    0008,ERROR-1
                 DEFB   +0E
```

THE 'SAVE' COMMAND ROUTINE

HL is set to point to the start of the program name. There is a 6 second header and then the bytes of the name and the program are passed out to the cassette recorder.

```
02F6 SAVE        CALL   03A8,NAME
                 JR     C,02F4,REPORT-F
                 EX     DE,HL
                 LD     DE,+12CB
02FF HEADER      CALL   0F46,BREAK-1
                 JR     NC,0332,BREAK-2
0304 DELAY-1     DJNZ   0304,DELAY-1
                 DEC    DE
                 LD     A,D
                 OR     E
                 JR     NZ,02FF,HEADER
030B OUT-NAME    CALL   031E,OUT-BYTE
                 BIT    7,(HL)
                 INC    HL
                 JR     Z,030B,OUT-NAME
                 LD     HL,+VERSN
0316 OUT-PROG.   CALL   031E,OUT-BYTE
                 CALL   01FC,LOAD/SAVE
                 JR     0316,OUT-PROG.
031E OUT-BYTE    LD     E,(HL)
                 SCF
0320 EACH-BIT    RL     E
                 RET    Z
                 SBC    A,A
                 AND    +05
                 ADD    +04
                 LD     C,A
0329 PULSES      OUT    (+FF),A
                 LD     B,+23
032D DELAY-2     DJNZ   032D,DELAY-2
                 CALL   0F46,BREAK-1
0332 BREAK-2     JR     NC,03A6,REPORT-D
                 LD     B,+1E
0336 DELAY-3     DJNZ   0336,DELAY-3
                 DEC    C
                 JR     NZ,0329,PULSES
033B DELAY-4     AND    A
                 DJNZ   033B,DELAY-4
                 JR     0320,EACH-BIT
```

THE 'LOAD' COMMAND ROUTINE

The bytes collected from the tape are matched against the program name and then the program is loaded into RAM.

```
0340 LOAD        CALL   03A8,NAME
                 RL     D
                 RRC    D
0347 NEXT-PROG   CALL   034C,IN-BYTE
                 JR     0347,NEXT-PROG
034C IN-BYTE     LD     C,+01
034E NEXT-BIT    LD     B,+00
0350 BREAK-3     LD     A,+7F
                 IN     A,(+FE)
                 OUT    (+FF),A
                 RRA
                 JR     NC,03A2,BREAK-4
                 RLA
                 RLA
```

```
                JR      C,0385,GET-BIT
                DJNZ    0350,BREAK-3
                POP     AF
                CP      D
0361 RESTART    JP      NC,03E5,INITIAL.
                LD      H,D
                LD      L,E
0366 IN-NAME    CALL    034C,IN-BYTE
                BIT     7,D
                LD      A,C
                JR      NZ,0371,MATCHING
                CP      (HL)
                JR      NZ,0347,NEXT-PROG
0371 MATCHING   INC     HL
                RLA
                JR      NC,0366,IN-NAME
                INC     (E-LINE-hi.)
                LD      HL,+VERSN
037B IN-PROG.   LD      D,B
                CALL    034C,IN-BYTE
                LD      (HL),C
                CALL    01FC,LOAD/SAVE
                JR      037B,IN-PROG
0385 GET-BIT    PUSH    DE
                LD      E,+94
0388 TRAILER    LD      B,+1A
038A COUNTER    DEC     E
                IN      A,(+FE)
                RLA
                BIT     7,E
                LD      A,E
                JR      C,0388,TRAILER
                DJNZ    038A,COUNTER
                POP     DE
                JR      NZ,039C,BIT-DONE
                CP      +56
                JR      NC,034E,NEXT-BIT
039C BIT-DONE   CCF
                RL      C
                JR      NC,034E,NEXT-BIT
                RET
03A2 BREAK-4    LD      A,D
                AND     A
                JR      Z,0361,RESTART
```

REPORT-D — Break pressed

```
03A6 REPORT-D   RST     0008,ERROR-1
                DEFB    +0C
```

THE 'PROGRAM NAME' SUBROUTINE

The name is checked for 'report C', 'FAST' mode is selected and the final letter of the name is inverted.

```
03A8 NAME       CALL    0F55,SCANNING
                LD      A,(FLAGS)
                ADD     A,A
                JP      M,0D9A,REPORT-C
                POP     HL
                RET     NC
                PUSH    HL
                CALL    02E7,SET-FAST
                CALL    13F8,STK-FETCH
                LD      H,D
                LD      L,E
                DEC     C
                RET     M
                ADD     HL,BC
                SET     7,(HL)
                RET
```

THE 'NEW' COMMAND ROUTINE

'FAST' mode is selected and BC is loaded with the present value of RAMTOP.

```
03C3 NEW        CALL    02E7,SET-FAST
                LD      BC,(RAMTOP)
                DEC     BC
```

THE RAM-CHECK ROUTINE

Starting with location RAMTOP-1 an attempt is made to fill each location with 02. The addresses are decremented until 3FFF is reached. Each location is then read-back until the first address that does not fetch 02 is found. This address is RAMTOP.

```
03CB RAM-CHECK  LD      H,B
                LD      L,C
                LD      A,+3F
03CF RAM-FILL   LD      (HL),+02
                DEC     HL
                CP      H
                JR      NZ,03CF,RAM-FILL
03D5 RAM-READ   AND     A
                SBC     HL,BC
                ADD     HL,BC
                INC     HL
                JR      NC,03E2,SET-TOP
                DEC     (HL)
                JR      Z,03E2,SET-TOP
                DEC     (HL)
                JR      Z,03D5,RAM-READ
03E2 SET-TOP    LD      (RAMTOP),HL
```

THE INITIALISATION ROUTINE

The different tasks of the initialisation routine are:

i. Set the top location in RAM to hold 3E.
ii. Set the stack pointer to point to the next location below.
iii. Set ERR-SP to hold the address two locations below the stack pointer.
iv. Set the I register to hold 1E.
v. Select interrupt mode 1.
vi. Set the IY register to hold ERR-NR as its base address.
vii. Select 'SLOW' mode.
viii. Set D-FILE to hold PROGRAM, i.e. No program present.
ix. Make a collapsed D-FILE.
x. Set VARS.
xi. CALL CLEAR command routine.
xii. Put the cursor in the edit line.
xiii. Produce a 'SLOW' display.

```
03E5 INITIAL    LD      HL,(RAMTOP)
                DEC     HL
     i.         LD      (HL),+3E
                DEC     HL
     ii.        LD      SP,HL
                DEC     HL
                DEC     HL
     iii.       LD      (ERR-SP),HL
                LD      A,+1E
     iv.        LD      I,A
     v.         IM1
     vi.        LD      IY,+ERR-NR
     vii.       LD      (CDFLAG),+40
                LD      HL,+PROGRAM
     viii.      LD      (D-FILE),HL
     ix.        LD      B,+19
0408 LINE       LD      (HL),+76
                INC     HL
                DJNZ    0408,LINE
     x.         LD      (VARS),HL
     xi.        CALL    149A,CLEAR
0413 N/L-ONLY   CALL    14AD,CURSOR-IN
     xiii.      CALL    0207,SLOW/FAST
```

PRODUCE THE BASIC LISTING

The 'upper' part of the display is produced by first calling the CLS command routine, then the BASIC program is listed from S-TOP.

The use of the 'cursor down' key also causes the 'upper' part of the display to be rebuilt.

```
0419 UPPER      CALL    0A2A,CLS
                LD      HL,(E-PPC)
                LD      DE,(S-TOP)
                AND     A
                SBC     HL,DE
                EX      DE,HL
                JR      NC,042D,ADDR-TOP
                ADD     HL,DE
                LD      (S-TOP),HL
042D ADDR-TOP   CALL    09D8,LINE-ADDR
                JR      Z,0433,LIST-TOP
                EX      DE,HL
0433 LIST-TOP   CALL    073E,LIST-PROG
                DEC     (BERG)
                JR      NZ,0472,LOWER
                LD      HL,(E-PPC)
                CALL    09D8,LINE-ADDR
                LD      HL,(CH-ADD)
                SCF
                SBC     HL,DE
                LD      HL,+S-TOP
                JR      NC,0457,INC-LINE
                EX      DE,HL
                LD      A,(HL)
                INC     HL
                LDI
                LD      (DE),A
                JR      0419,UPPER
```

'cursor down' entry point.

```
0454 DOWN-KEY   LD      HL,+E-PPC
0457 INC-LINE   LD      E,(HL)
                INC     HL
                LD      D,(HL)
                PUSH    HL
                EX      DE,HL
                INC     HL
                CALL    09D8,LINE-ADDR
                CALL    05BB,LINE-NO.
                POP     HL
0464 KEY-INPUT  BIT     5,(FLAGX)
                JR      NZ,0472,LOWER
                LD      (HL),D
                DEC     HL
                LD      (HL),E
                JR      0419,UPPER
```

COPY THE EDIT-LINE

The 'lower' part of the display is formed by copying the edit-line from the workspace to the bottom of the screen.

First floating point numbers are removed, then the blank part of the screen is defined and finally the edit-line is copied over with the 'lower' part of the screen being expanded if necessary.

The EDIT-INP. entry point comes into use when EDIT is used in reply to a request for INPUT.

```
046F EDIT-INP.  CALL    14AD,CURSOR-IN
0472 LOWER      LD      HL,(E-LINE)
0475 EACH-CHAR  LD      A,(HL)
                CP      +7E
                JR      NZ,0482,END-LINE
                LD      BC,+0006
                CALL    0A60,RECLAIM-2
                JR      0475,EACH-CHAR
0482 END-LINE   CP      +76
                INC     HL
                JR      NZ,0475,EACH-CHAR
0487 EDIT-LINE  CALL    0537,CURSOR
048A EDIT-ROOM  CALL    0A1F,LINE-ENDS
                LD      HL,(E-LINE)
                LD      (ERR-NR),+FF
                CALL    0766,COPY-LINE
                BIT     7,(ERR-NR)
                JR      NZ,04C1,DISPLAY-6
                LD      A,(DF-SZ)
                CP      +18
                JR      NC,04C1,DISPLAY-6
                INC     A
                LD      (DF-SZ),A
                LD      B,A
                LD      C,+01
                CALL    0918,LOC.-ADDR
                LD      D,H
                LD      E,L
                LD      A,(HL)
04B1 FREE-LINE  DEC     HL
                CP      (HL)
                JR      NZ,04B1,FREE-LINE
                INC     HL
                EX      DE,HL
                LD      A,(RAMTOP-hi.)
                CP      +4D
                CALL    C,0A5D,RECLAIM-1
                JR      048A,EDIT-ROOM
```

WAITING FOR A KEY

The syntax error pointer is set to zero and a display is produced. Once a key has been pressed the display is terminated. The pressing of 'multiple keys' causes a jump back to LOWER.

```
04C1 DISPLAY-6  LD    HL,+0000
                LD    (X-PTR),HL
                LD    HL,+CDFLAG
                BIT   7,(HL)
                CALL  Z,0229,DISPLAY-1
04CF SLOW-DISP  BIT   0,(HL)
                JR    Z,04CF,SLOW-DISP
                LD    BC,(LAST-K)
                CALL  0F4B,D-BOUNCE
                CALL  07BD,DECODE
                JR    NC,0472,LOWER
```

MODE SORTING

The differing modes give differing values for the keys of the keyboard. These are obtained from the key tables.

```
04DF MODE-SORT  LD    A,(MODE)
                DEC   A
                JP    M,0508,FETCH-2
                JR    NZ,04F7,FETCH-1
                LD    (MODE),A
                DEC   E
                LD    A,E
                SUB   +27
                JR    C,04F2,FUNC-BASE
                LD    E,A
04F2 FUNC-BASE  LD    HL,+00CC
                JR    0505,TABLE-ADD
04F7 FETCH-1    LD    A,(HL)
                CP    +76
                JR    Z,052B,K/L-KEY
                CP    +40
                SET   7,A
                JR    C,051B,ENTER
                LD    HL,+00C7
0505 TABLE-ADD  ADD   HL,DE
                JR    0515,FETCH-3
0508 FETCH-2    LD    A,(HL)
                BIT   2,(FLAGS)
                JR    NZ,0516,TEST-CURS
                ADD   A,+C0
                CP    +E6
                JR    NC,0516,TEST-CURS
0515 FETCH-3    LD    A,(HL)
0516 TEST-CURS  CP    +F0
                JP    PE,052D,KEY-SORT
051B ENTER      LD    E,A
                CALL  0537,CURSOR
                LD    A,E
                CALL  0526,ADD-CHAR
0523 BACK-NEXT  JP    0472,LOWER
```

THE 'ADD-CHAR' SUBROUTINE

All of the RAM from (HL) to STKEND is moved up by one byte and the character code in the A register is entered into the extra location.

```
0526 ADD-CHAR   CALL  099B,ONE-SPACE
                LD    (DE),A
                RET
```

SORTING THE CURSOR KEYS

The addresses of the different routines for the cursor keys are obtained by adding the character code twice to the base address 0482. The address is then stacked.

```
052B K/L-KEY    LD    A,+78
052D KEY-SORT   LD    E,A
                LD    HL,+0482
                ADD   HL,DE
                ADD   HL,DE
                LD    C,(HL)
                INC   HL
                LD    B,(HL)
                PUSH  BC
```

CHOOSING K v. L MODE

The characters in the edit-line are read in turn.

Initially K-mode is selected but it will be changed to L-mode unless the line holds only the cursor or the last token is THEN. The RET Z instruction takes the program to the cursor key routines.

```
0537 CURSOR     LD    HL,(E-LINE)
                BIT   5,(FLAGX)
                JR    NZ,0556,L-MODE
0540 K-MODE     RES   2,(FLAGS)
0544 TEST-CHAR  LD    A,(HL)
                CP    +7F
                RET   Z
                INC   HL
                CALL  07B4,NUMBER
                JR    Z,0544,TEST-CHAR
                CP    +26
                JR    C,0544,TEST-CHAR
                CP    +DE
                JR    Z,0540,K-MODE
0556 L-MODE     SET   2,(FLAGS)
                JR    0544,TEST-CHAR
```

THE 'CLEAR-ONE' SUBROUTINE

The single character (HL) is overwritten by moving all of the RAM from (HL+1)-STKEND down by one byte.

```
055C CLEAR-ONE  LD    BC,+0001
                JP    0A60,RECLAIM-2
```

THE CURSOR KEY TABLE

0562	9F	05	UP-KEY	059F
0564	54	04	DOWN-KEY	0454
0566	76	05	LEFT-KEY	0576
0568	7F	05	RIGHT-KEY	057F
056A	AF	05	GRAPHICS	05AF
056C	C4	05	EDIT-KEY	05C4
056E	0C	06	N/L-KEY	060C
0570	8B	05	RUBOUT	058B
0572	AF	05	FUNCTION	05AF
0574	AF	05	FUNCTION	05AF

THE CURSOR LEFT ROUTINE

```
0576 LEFT-KEY   CALL   0593,LEFT-EDGE
                LD     A,(HL)
                LD     (HL),+7F
                INC    HL
                JR     0588,GET-CODE
```

THE CURSOR RIGHT ROUTINE

```
057F RIGHT-KEY  INC    HL
                LD     A,(HL)
                CP     +76
                JR     Z,059D,ENDED-2
                LD     (HL),+7F
                DEC    HL
0588 GET-CODE   LD     (HL),A
0589 ENDED-1    JR     0523,BACK-NEXT
```

THE RUBOUT ROUTINE

```
058B RUBOUT     CALL   0593,LEFT-EDGE
                CALL   055C,CLEAR-ONE
                JR     0589,ENDED-1
```

THE 'LEFT-EDGE' SUBROUTINE

The first character in the edit-line is tested against +7F, the cursor.

```
0593 LEFT-EDGE  DEC    HL
                LD     DE,(E-LINE)
                LD     A,(DE)
                CP     +7F
                RET    NZ
                POP    DE
059D ENDED-2    JR     0589,ENDED-1
```

THE CURSOR UP ROUTINE

```
059F UP-KEY     LD     HL,(E-PPC)
                CALL   09D8,LINE-ADDR
                EX     DE,HL
                CALL   05BB,LINE-NO.
                LD     HL,+E-PPC-hi.
                JP     0464,KEY-INPUT
```

THE FUNCTION KEY ROUTINE

```
05AF FUNCTION   LD     A,E
                AND    +07
                LD     (MODE),A
                JR     059D,ENDED-2
```

THE 'COLLECT LINE NUMBER' SUBROUTINE

The subroutine is entered at LINE-NO. with an address in HL. If a line number is to be found at that position then it is returned in DE, otherwise DE is returned with +0000.

```
05B7 ZERO-DE    EX     DE,HL
                LD     DE,+04C2
05BB LINE-NO.   LD     A,(HL)
                AND    +CO
                JR     NZ,05B7,ZERO-DE
                LD     D,(HL)
                INC    HL
                LD     E,(HL)
                RET
```

THE EDIT KEY ROUTINE

First the 'lower' part of the screen is cleared, then the flag that shows whether the INPUT command is being followed, is tested and a return made if the flag is set.

Next the line to be edited is located. Its number is printed, followed by the cursor, but before the line itself is copied into the workspace a test for sufficient room is made.

A return is made if there is not enough available RAM.

```
05C4 EDIT-KEY   CALL   0A1F,LINE-ENDS
                LD     HL,+EDIT-INP.
                PUSH   HL
                BIT    5,(FLAGX)
                RET    NZ
                LD     HL,(E-LINE)
                LD     (DF-CC),HL
                LD     HL,+1821
                LD     (S-POSN),HL
                LD     HL,(E-PPC)
                CALL   09D8,LINE-ADDR
                CALL   05BB,LINE-NO.
                LD     A,D
                OR     E
                RET    Z
                DEC    HL
                CALL   0AA5,OUT-NO.
                INC    HL
                LD     C,(HL)
                INC    HL
                LD     B,(HL)
                INC    HL
                LD     DE,(DF-CC)
                LD     A,+7F
                LD     (DE),A
                INC    DE
                PUSH   HL
                LD     HL,+001D
                ADD    HL,DE
                ADD    HL,BC
                SBC    HL,SP
                POP    HL
                RET    NC
                LDIR
                EX     DE,HL
                POP    DE
                CALL   14A6,SET-STK-B
                JR     059D,ENDED-2
```

THE NEWLINE KEY ROUTINE

The NEWLINE key can be used in three separate situations and these have to be dealt with in different ways.

The first part of the routine is common to all situations.

The 'lower' part of the screen is cleared. The PRBUFF is also cleared unless the INPUT command is being used, or the direct command COPY.

The line is then scanned to check for syntax errors. The cursor is removed and the line number found, if present.

```
060C N/L-KEY       CALL  0A1F,LINE-ENDS
                   LD    HL,+LOWER
                   BIT   5,(FLAGX)
                   JR    NZ,0629,NOW-SCAN
                   LD    HL,(E-LINE)
                   LD    A,(HL)
                   CP    +FF
                   JR    Z,0626,STK-UPPER
                   CALL  08E2,CLEAR-PRB
                   CALL  0A2A,CLS
0626 STK-UPPER     LD    HL,+UPPER
0629 NOW-SCAN      PUSH  HL
                   CALL  0CBA, LINE-SCAN
                   POP   HL
                   CALL  0537,CURSOR
                   CALL  055C,CLEAR-ONE
                   CALL  0A73,E-LINE-NO
                   JR    NZ,064E,N/L-INP.
                   LD    A,B
                   OR    C
                   JP    NZ,06E0,N/L-LINE
```

The second part sets up the required parameters for the execution of a line, either as a BASIC line or as an INPUT line.

An empty line is detected and the program jumps back to the initialisation routine.

```
                   DEC   BC
                   DEC   BC
                   LD    (PPC),BC
                   LD    (DF-SZ),+02
                   LD    DE,(D-FILE)
                   JR    0661,TEST-NULL
064E N/L-INP.      CP    +76
                   JR    Z,0664,N/L-NULL
                   LD    BC,(T-ADDR)
                   CALL  0918,LOC.-ADDR
                   LD    DE,(NXTLIN)
                   LD    (DF-SZ),+02
0661 TEST-NULL     RST   0018,GET-CH
                   CP    +76
0664 N/L-NULL      JP    Z,0413,N/L-ONLY
                   LD    (FLAGS),+80
                   EX    DE,HL
```

The third part of the routine is the 'line execution loop'. When a BASIC program is being RUN it is this 'loop' that leads to the execution of the BASIC lines in their correct order.

In the case of the INPUT command the 'line' is detected as input in the LINE-RUN subroutine.

```
066C NEXT-LINE     LD    (NXTLIN),HL
                   EX    DE,HL
                   CALL  004D,TEMP-PTR
                   CALL  0CC1,LINE-RUN
                   RES   1,(FLAGS)
                   LD    A,+CO
                   LD    (X-PTR-hl.),A
                   CALL  14A3,X-TEMP
                   RES   5,(FLAGX)
                   BIT   7,(ERR-NR)
                   JR    Z,06AE,STOP-LINE)
                   LD    HL,(NXTLIN)
                   AND   (HL)
                   JR    NZ,06AE STOP-LINE
                   LD    D,(HL)
                   INC   HL
                   LD    E,(HL)
                   LD    (PPC),DE
                   INC   HL
                   LD    E,(HL)
                   INC   HL
                   LD    D,(HL)
                   INC   HL
                   EX    DE,HL
                   ADD   HL,DE
                   CALL  0F46,BREAK-1
                   JR    C,066C,NEXT-LINE
```

The third part of the routine is used to produce the report at the end of a RUN, after other direct commands and following the use of the BREAK key.

```
                   LD    HL,+ERR-NR
                   BIT   7,(HL)
                   JR    Z,06AE,STOP-LINE
                   LD    (HL),+0C
06AE STOP-LINE     BIT   7,(PR-CC)
                   CALL  Z,0871,COPY-BUFF
                   LD    BC,+0121
                   CALL  0918,LOC.-ADDR
                   LD    A,(ERR-NR)
                   LD    BC,(PPC)
                   INC   A
                   JR    Z,06D1,REPORT
                   CP    +09
                   JR    NZ,06CA,CONTINUE
                   INC   BC
06CA CONTINUE      LD    (OLDPPC),BC
                   JR    NZ,06D1,REPORT
                   DEC   BC
06D1 REPORT        CALL  07EB,OUT-CODE
                   LD    A,+18
                   RST   0010,PRINT-A
                   CALL  0A98,OUT-NUM.
                   CALL  14AD,CURSOR-IN
                   JP    04C1,DISPLAY-6
```

The fourth part of the routine is involved in the entering of a BASIC line into its correct position in the BASIC program.

Initially a search is made to see if there is already a line with the same name number. If a line is found then it is 'reclaimed'.

The new line is then copied from the workspace to its correct place in the BASIC program.

```
06E0 N/L-LINE      LD    (E-PPC),BC
                   LD    HL,(CH-ADD)
                   EX    DE,HL
                   LD    HL,+N/L-ONLY
                   PUSH  HL
                   LD    HL,(STKBOT)
                   SBC   HL,DE
```

```
                PUSH   HL
                PUSH   BC
                CALL   02E7,SET-FAST
                CALL   0A2A,CLS
                POP    HL
                CALL   09D8,LINE-ADDR
                JR     NZ,0705,COPY-OVER
                CALL   09F2,NEXT-ONE
                CALL   0A60,RECLAIM-2
0705 COPY-OVER  POP    BC
                LD     A,C
                DEC    A
                OR     B
                RET    Z
                PUSH   BC
                INC    BC
                INC    BC
                INC    BC
                INC    BC
                DEC    HL
                CALL   099E,MAKE-ROOM
                CALL   0207,SLOW/FAST
                POP    BC
                PUSH   BC
                INC    DE
                LD     HL,(STKBOT)
                DEC    HL
                LDDR
                LD     HL,(E-PPC)
                EX     DE,HL
                POP    BC
                LD     (HL),B
                DEC    HL
                LD     (HL),C
                DEC    HL
                LD     (HL),E
                DEC    HL
                LD     (HL),D
                RET
```

THE 'LIST' COMMAND ROUTINE

The 'LIST' command will list the BASIC program from a given line, or line zero if no number is supplied.

The first part of the routine finds the 'parameter' and saves the line number in E-PPC. The second part of the routine repeatedly calls the OUT-LINE subroutine until either the screen is full or the last line has been printed.

```
072C LLIST      SET    1,(FLAGS)
0730 LIST       CALL   0EA7,FIND-INT.
                LD     A,B
                AND    +3F
                LD     H,A
                LD     L,C
                LD     (E-PPC),HL
                CALL   09D8,LINE-ADDR
073E LIST-PROG  LD     E,+00
0740 UNTIL-END  CALL   0745,OUT-LINE
                JR     0740,UNTIL-END
```

THE 'PRINT A BASIC LINE' SUBROUTINE

The first part of the routine fetches the line number of the 'current cursor line' and tests it against the line number that it is to print. The line number is then printed followed by the 'current line cursor' if required, or a space if not.

```
0745 OUT-LINE   LD     BC,(E-PPC)
                CALL   09EA,CP-LINES
                LD     D,+92
                JR     Z,0755,TEST-END
                LD     DE,+0000
                RL     E
0755 TEST-END   LD     (BERG),E
                LD     A,(HL)
                CP     +40
                POP    BC
                RET    NC
                PUSH   BC
                CALL   0AA5.OUT-NO.
                INC    HL
                LD     A,D
                RST    0010,PRINT-A
                INC    HL
                INC    HL
```

The second part of the routine prints the actual line. By comparing CH-ADD & X-PTR the routine tests to see if the syntax error marker should be printed. The routine also tests for floating point numbers and jumps over them. When a 'token' is found a call is made to the 'token printing' subroutine. When the cursor marker is found the appropriate cursor is printed.

```
0766 COPY-LINE  LD     (CH-ADD),HL
                SET    0,(FLAGS)
076D MORE-LINE  LD     BC,(X-PTR)
                LD     HL,(CH-ADD)
                AND    A
                SBC    HL,BC
                JR     NZ,077C,TEST-NUM.
                LD     A,+B8
                RST    0010,PRINT-A
077C TEST-NUM.  LD     HL,(CH-ADD)
                LD     A,(HL)
                INC    HL
                CALL   07B4,NUMBER
                LD     (CH-ADD),HL
                JR     Z,076D,MORE-LINE
                CP     +7F
                JR     Z,079D,OUT-CURS.
                CP     +76
                JR     Z,07EE,OUT-CH
                BIT    6,A
                JR     Z,079A,NOT-TOKEN
                CALL   094B,TOKENS
                JR     076D,MORE-LINE
079A NOT-TOKEN  RST    0010,PRINT-A
                JR     076D,MORE-LINE
079D OUT-CURS.  LD     A,(MODE)
                LD     B,+AB
                AND    A
```

```
              JR      NZ,07AA,FLAGS-2
              LD      A,(FLAGS)
              LD      B,+B0
07AA FLAGS-2  RRA
              RRA
              AND     +01
              ADD     A,B
              CALL    07F5,PRINT-SP.
              JR      076D,MORE-LINE
```

THE 'NUMBER' SUBROUTINE

This subroutine tests the character in the A register against the 'number marker'. If a match occurs then the value in the HL register pair is incremented five times, so as to either skip over the floating point number, or to reserve 5 bytes for such a number.

```
07B4 NUMBER   CP      +7E
              RET     NZ
              INC     HL
              INC     HL
              INC     HL
              INC     HL
              INC     HL
              RET
```

THE 'KEYBOARD DECODE' SUBROUTINE

The different 'key values', held in the BC register pair, are 'decoded' into the usual ZX81 character codes by looking-up the key table at 007E. (007D + 1) The character code is specified as (HL).

```
07BD DECODE   LD      D,+00
              SRA     B
              SBC     A,A
              OR      +26
              LD      L,+05
              SUB     L
07C7 KEY-LINE ADD     A,L
              SCF
              RR      C
              JR      C,07C7,KEY-LINE
              INC     C
              RET     NZ
              LD      C,B
              DEC     L
              LD      L,+01
              JR      NZ,07C7,KEY-LINE
              LD      HL,+007D
              LD      E,A
              ADD     HL,DE
              SCF
              RET
```

THE 'PRINTING' SUBROUTINE

The two little routines WRITE-CH & WRITE-N/L are the essential parts of the printing subroutine. However before a character can be actually printed it is necessary for S-POSN to be collected and tested, and the display expanded if required.

The various entry points to the subroutine are involved with the conversion of Hex. codes to ZX81 character codes.

i) Printing digits:

```
07DC LEAD-SP.   LD      A,E
                AND     A
                RET     M
                JR      07F1,PRINT-CH.
07E1 OUT-DIGIT  XOR     A
07E2 DIGIT-INC  ADD     HL,BC
                INC     A
                JR      C,07E2,DIGIT-INC
                SBC     HL,BC
                DEC     A
                JR      Z,07DC,LEAD-SP.
07EB OUT-CODE   LD      E,+1C
                ADD     A,E
07EE OUT-CH     AND     A
                JR      Z,07F5,PRINT-SP.
```

ii) Printing characters:

```
07F1 PRINT-CH.  RES     0,(FLAGS)
07F5 PRINT-SP.  EXX
                PUSH    HL
                BIT     1,(FLAGS)
                JR      NZ,0802,LPRINT-A
                CALL    0808,ENTER-CH
                JR      0805,PRINT-EXX
0802 LPRINT-A   CALL    0851,LPRINT-CH
0805 PRINT-EXX  POP     HL
                EXX
                RET
```

iii) Testing S-POSN:

```
0808 ENTER-CH   LD      D,A
                LD      BC,(S-POSN)
                LD      A,C
                CP      +21
                JR      Z,082C,TEST-LOW
0812 TEST-N/L   LD      A,+76
                CP      D
                JR      Z,0847,WRITE-N/L
                LD      HL,(DF-CC)
                CP      (HL)
                LD      A,D
                JR      NZ,083E,WRITE-CH
                DEC     C
                JR      NZ,083A,EXPAND-1
                INC     HL
                LD      (DF-CC),HL
                LD      C,+21
                DEC     B
                LD      (S-POSN),BC
082C TEST-LOW   LD      A,B
                CP      (DF-SZ)
                JR      Z,0835,REPORT-5
                AND     A
                JR      NZ,0812,TEST-N/L
```

iv) REPORT-5 — insufficient room:

```
0835 REPORT-5   LD      L,+04
                JP      0058,ERROR-3
```

v) Expand the display:

```
083A EXPAND-1   CALL   099B,ONE-SPACE
                EX     DE,HL
```

vi) Writing an actual code:

```
083E WRITE-CH   LD     (HL),A
                INC    HL
                LD     (DF-CC),HL
                DEC    (S-POSN-lo.)
                RET
```

vii) Writing a N/L:

This is performed by decrementing the 'line counter' and using LOC.ADDR to give the correct values for DF-CC & S-POSN.

```
0847 WRITE-N/L  LD     C,+21
                DEC    B
                SET    0,(FLAGS)
                JP     0918,LOC.-ADDR
```

THE 'LPRINT-CH' SUBROUTINE

Characters are added one by one to the printer buffer. Once the buffer is full, or a N/L character is entered the buffer is emptied.

```
0851 LPRINT-CH  CP     +76
                JR     Z,0871,COPY-BUFF
                LD     C,A
                LD     A,(PR-CC)
                AND    +7F
                CP     +5C
                LD     L,A
                LD     H,+40
                CALL   Z,0871,COPY-BUFF
                LD     (HL),C
                INC    L
                LD     (PR-CC),L
                RET
```

THE 'COPY' COMMAND ROUTINE

The COPY command routine starts with the D register being loaded with Hex.16, being the number of lines in a full display. The Copy*D routine is then used to output these lines to the printer.

```
0869 COPY       LD     D,+16
                LD     HL,+D-FILE
                INC    HL
                JR     0876,COPY*D
```

In COPY-BUFF the D register is only required to be given the value Hex.01.

```
0871 COPY-BUFF  LD     D,+01
                LD     HL,+PRBUFF
```

In COPY*D a loop is set up with D being the counter.

```
0876 COPY*D     CALL   02E7,SET-FAST
                PUSH   BC
087A COPY-LOOP  PUSH   HL
                XOR    A
                LD     E,A
087D COPY-TIME  OUT    (+FB),A
                POP    HL
0880 COPY-BRK   CALL   0F46,BREAK-1
                JR     C,088A,COPY-CONT
                RRA
                OUT    (+FB),A
0888 REPORT-D2  RST    0008,ERROR-1
                DEFB   +0C
088A COPY-CONT  IN     A,(+FB)
                ADD    A,A
                JP     M,08DE,COPY-END
                JR     NC,0880,COPY-BRK
                PUSH   HL
                PUSH   DE
                LD     A,D
                CP     +02
                SBC    A,A
                AND    E
                RLCA
                AND    E
                LD     D,A
089C COPY-NEXT  LD     C,(HL)
                LD     A,C
                INC    HL
                CP     +76
                JR     Z,08C7,COPY-N/L
                PUSH   HL
                SLA    A
                ADD    A,A
                ADD    A,A
                LD     H,+0F
                RL     H
                ADD    A,E
                LD     L,A
                RL     C
                SBC    A,A
                XOR    (HL)
                LD     C,A
                LD     B,+08
08B5 COPY-BITS  LD     A,D
                RLC    C
                RRA
                LD     H,A
08BA COPY-WAIT  IN     A,(+FB)
                RRA
                JR     NC,08BA,COPY-WAIT
                LD     A,H
                OUT    (+FB),A
                DJNZ   08B5,COPY-BITS
                POP    HL
                JR     089C,COPY-NEXT
08C7 COPY-N/L   IN     A,(+FB)
                RRA
                JR     NC,08C7,COPY-N/L
                LD     A,D
                RRCA
                OUT    (+FB),A
                POP    DE
                INC    E
                BIT    3,E
                JR     Z,087D,COPY-TIME
                POP    BC
                DEC    D
                JR     NZ,087A,COPY-LOOP
                LD     A,+04
```

```
                  OUT    (+FB),A
08DE COPY-END     CALL   0207,SLOW/FAST
                  POP    BC
```

THE 'CLEAR PRINTER BUFFER' SUBROUTINE

The printer buffer is cleared by overwriting it with Hex.00 characters and setting the final location to Hex.76.

```
08E2 CLEAR-PRB    LD     HL,+405C
                  LD     (HL),+76
                  LD     B,+20
08E9 PRB-BYTES    DEC    HL
                  LD     (HL),+00
                  DJNZ   08E9,PRB-BYTES
                  LD     A,L
                  SET    7,A
                  LD     (PR-CC),A
                  RET
```

THE 'PRINT AT' SUBROUTINE

This routine checks the validity of the parameters given with the PRINT AT command. If the parameters are invalid an error is signalled otherwise the correct S-POSN & DF-CC is obtained by using the LOC.-ADDR routine.

```
08F5 PRINT-AT     LD     A,+17
                  SUB    B
                  JR     C,0905,WRONG-VAL
08FA TEST-VAL.    CP     (DF-SZ)
                  JP     C,0835,REPORT-5
                  INC    A
                  LD     B,A
                  LD     A,+1F
                  SUB    C
0905 WRONG-VAL    JP     C,0EAD,REPORT-B
                  ADD    A,+02
                  LD     C,A
090B SET-FIELD    BIT    1,(FLAGS)
                  JR     Z,0918,LOC.-ADDR
```

The LPRINT AT command sets the value of PR-CC.

```
                  LD     A,+5D
                  SUB    C
                  LD     (PR-CC),A
                  RET
```

THE 'LOC.-ADDR' SUBROUTINE

This important subroutine sets the value of DF-CC for given values of a display location. If the display is collapsed and thereby does not truly hold the position then the required line is expanded.

```
0918 LOC.-ADDR    LD     (S-POSN),BC
                  LD     HL,(VARS)
                  LD     D,C
                  LD     A,+22
                  SUB    C
                  LD     C,A
                  LD     A,+76
                  INC    B
0927 LOOK-BACK    DEC    HL
                  CP     (HL)
                  JR     NZ,0927,LOOK-BACK
                  DJNZ   0927,LOOK-BACK
                  INC    HL
                  CPIR
                  DEC    HL
                  LD     (DF-CC),HL
                  SCF
                  RET    PO
                  DEC    D
                  RET    Z
                  PUSH   BC
                  CALL   099E,MAKE-ROOM
                  POP    BC
                  LD     B,C
                  LD     H,D
                  LD     L,E
0940 EXPAND-2     LD     (HL),+00
                  DEC    HL
                  DJNZ   0940,EXPAND-2
                  EX     DE,HL
                  INC    HL
                  LD     (DF-CC),HL
                  RET
```

THE 'EXPAND TOKENS' SUBROUTINE

The character codes that are considered to be tokens are expanded using this subroutine. The address of each 'expanded token' in the 'token table' is found using TOKEN-ADD. The leading space is printed if specified by bit 0 of FLAGS, the letters of the token-word are then printed and a trailing space is added if needed.

```
094B TOKENS       PUSH   AF
                  CALL   0975,TOKEN-ADD
                  JR     NC,0959,ALL-CHARS
                  BIT    0,(FLAGS)
                  JR     NZ,0959,ALL-CHARS
                  XOR    A
                  RST    0010,PRINT-A
0959 ALL-CHARS    LD     A,(BC)
                  AND    +3F
                  RST    0010,PRINT-A
                  LD     A,(BC)
                  INC    BC
                  ADD    A,A
                  JR     NC,0959,ALL-CHARS
                  POP    BC
                  BIT    7,B
                  RET    Z
                  CP     +1A
                  JR     Z,096D,TRAIL-SP.
                  CP     +38
                  RET    C
096D TRAIL-SP.    XOR    A
                  SET    0,(FLAGS)
                  JP     07F5,PRINT-SP.
```

In TOKEN-ADD the base address of the TOKEN TABLE is Hex.0111. The words in this table are found in turn and when the required word has been located a return is made with BC pointing to the start of the word.

```
0975 TOKEN-ADD PUSH   HL
               LD     HL,+0111
               BIT    7,A
               JR     Z,097F,TEST-HIGH
               AND    +3F
097F TEST-HIGH CP     +43
               JR     NC,0993,FOUND
               LD     B,A
               INC    B
0985 WORDS     BIT    7,(HL)
               INC    HL
               JR     Z,0985,WORDS
               DJNZ   0985,WORDS
               BIT    6,A
               JR     NZ,0992,COMP-FLAG
               CP     +18
0992 COMP-FLAG CCF
0993 FOUND     LD     B,H
               LD     C,L
               POP    HL
               RET    NC
               LD     A,(BC)
               ADD    A,+E4
               RET
```

THE 'ONE-SPACE' SUBROUTINE

Whenever a single space is required in the program area or the display file then this subroutine is called.

```
099B ONE-SPACE LD     BC,+0001
```

THE 'MAKE-ROOM' SUBROUTINE

This routine creates BC spaces from the location (HL).

```
099E MAKE-ROOM PUSH   HL
               CALL   0EC5,TEST-ROOM
               POP    HL
               CALL   09AD,POINTERS
               LD     HL,(STKEND)
               EX     DE,HL
               LDDR
               RET
```

THE 'CHANGE ALL POINTERS' SUBROUTINE

Whenever some of the pointers require to be changed this subroutine is called with the amount of change in BC, and HL determining which pointers are to be changed. All pointers that point lower than HL will not be altered.

```
09AD POINTERS  PUSH   AF
               PUSH   HL
               LD     HL,+D-FILE
               LD     A,+09
09B4 NEXT-PTR  LD     E,(HL)
               INC    HL
               LD     D,(HL)
               EX     (SP),HL
               AND    A
               SBC    HL,DE
               ADD    HL,DE
               EX     (SP),HL
               JR     NC,09C8,PTR-DONE
               PUSH   DE
               EX     DE,HL
               ADD    HL,BC
               EX     DE,HL
               LD     (HL),D
               DEC    HL
               LD     (HL),E
               INC    HL
               POP    DE
09C8 PTR-DONE  INC    HL
               DEC    A
               JR     NZ,09B4,NEXT-PTR
               EX     DE,HL
               POP    DE
               POP    AF
               AND    A
               SBC    HL,DE
               LD     B,H
               LD     C,L
               INC    BC
               ADD    HL,DE
               EX     DE,HL
               RET
```

THE 'LINE-ADDR' SUBROUTINE

For a given BASIC line number this subroutine will return the starting address of the actual line (and the Z flag set) or the starting address of the following line if it does not exist (C reset).

```
09D8 LINE-ADDR PUSH   HL
               LD     HL,+PROGRAM
               LD     D,H
               LD     E,L
09DE NEXT-TEST POP    BC
               CALL   09EA,CP-LINES
               RET    NC
               PUSH   BC
               CALL   09F2,NEXT-ONE
               EX     DE,HL
               JR     09DE,NEXT-TEST
```

THE 'COMPARE LINE NUMBERS' SUBROUTINE

The line number in (HL) is compared to the number in BC.

```
09EA CP-LINES  LD     A,(HL)
               CP     B
               RET    NZ
               INC    HL
               LD     A,(HL)
               DEC    HL
               CP     C
               RET
```

THE 'NEXT LINE or VARIABLE' SUBROUTINE

This subroutine very cleverly finds the start of the next BASIC line or the start of the next variable in the variable area. Line numbers are identified by the high byte being less than Hex.40, and the different types of variables are identified by their differing bits 6 & 7.

```
09F2 NEXT-ONE  PUSH   HL
               LD     A,(HL)
```

```
                  CP     +40
                  JR     C,0A0F,LINES
                  BIT    5,A
                  JR     Z,0A10,BIT-5-NIL
                  ADD    A,A
                  JP     M,0A01,NEXT+FIVE
                  CCF
0A01 NEXT+FIVE    LD     BC,+0005
                  JR     NC,0A08,NEXT-LETT
                  LD     C,+11
0A08 NEXT-LETT    RLA
                  INC    HL
                  LD     A,(HL)
                  JR     NC,0A08,NEXT-LETT
                  JR     0A15,NEXT-ADD
0A0F LINES        INC    HL
0A10 BIT-5-NIL    INC    HL
                  LD     C,(HL)
                  INC    HL
                  LD     B,(HL)
                  INC    HL
0A15 NEXT-ADD     ADD    HL,BC
                  POP    DE
```

THE 'DIFFERENCE' SUBROUTINE

This subroutine finds the difference in value between the contents of the HL and DE register pairs. The result is returned in the BC register pair.

```
0A17 DIFFER       AND    A
                  SBC    HL,DE
                  LD     B,H
                  LD     C,L
                  ADD    HL,DE
                  EX     DE,HL
                  RET
```

THE 'LINE ENDS' SUBROUTINE

The lines of the 'lower' screen are cleared by this subroutine.

```
0A1F LINE-ENDS    LD     B,(DF-SZ)
                  PUSH   BC
                  CALL   0A2C,B-LINES
                  POP    BC
                  DEC    B
                  JR     0A2C,B-LINES
```

THE 'CLS' COMMAND ROUTINE

i) The B register is loaded with Hex.18, the number of lines in the display file.

```
0A2A CLS          LD     B,+18
```

ii) The address of the start of that part of the display file that is to be cleared is found and a test is made to see if more, or less, than 3¼ K. of RAM is fitted.

```
0A2C B-LINES      RES    1,(FLAGS)
                  LD     C,+21
                  PUSH   BC
                  CALL   0918,LOC.-ADDR
                  POP    BC
                  LD     A,(RAMTOP-hi.)
                  CP     +4D
                  JR     C,0A52,COLLAPSE
```

iii) As an expanded display file is required, a suitable number of spaces is printed so as to clear the specified number of lines.

```
                  SET    7,(S-POSN-hi.)
0A42 CLEAR-LOC    XOR    A
                  CALL   07F5,PRINT-SP.
                  LD     HL,(S-POSN)
                  LD     A,L
                  OR     H
                  AND    +7E
                  JR     NZ,0A42,CLEAR-LO
                  JP     0918,LOC.-ADDR
```

iv) As a collapsed display file is required a LDIR instruction is used to copy a N/L character the number of times specified in the C register (formerly B). The system variable VARS is then found and excess memory reclaimed.

```
0A52 COLLAPSED    LD     D,H
                  LD     E,L
                  DEC    HL
                  LD     C,B
                  LD     B,+00
                  LDIR
                  LD     HL,(VARS)
```

THE 'RECLAIMING' SUBROUTINES

The pointers are first changed and then the specified area of RAM is reclaimed by using a LDIR instruction to overwrite the unwanted part of the RAM contents.

```
0A5D RECLAIM-1    CALL   0A17,DIFFER
0A60 RECLAIM-2    PUSH   BC
                  LD     A,B
                  CPL
                  LD     B,A
                  LD     A,C
                  CPL
                  LD     C,A
                  INC    BC
                  CALL   09AD,POINTERS
                  EX     DE,HL
                  POP    HL
                  ADD    HL,DE
                  PUSH   DE
                  LDIR
                  POP    HL
                  RET
```

THE 'E-LINE NUMBER' SUBROUTINE

This routine is used to find out whether the current E-Line starts with a valid line number. i.e. 1-9999. The pointer CH-ADD is used temporarily to point along the E-LINE. A return is made if the INPUT command is being executed. The INT-TO-FP routine is called to collect the possible number and the FP-TO-BC routine called to form an integer value. The value is then tested against dec.0-10,000. The subroutine returns via the SET-MEM subroutine that resets STKEND.

```
0A73 E-LINE-NO  LD     HL,(E-LINE)
                CALL   004D,TEMP-PTR
                RST    0018,GET-CH.
                BIT    5,(FLAGX)
                RET    NZ
                LD     HL,+MEMBOT
                LD     (STKEND),HL
                CALL   1548,INT-TO-FP
                CALL   158A,FP-TO-BC
                JR     C,0A91,NO-NUMBER
                LD     HL,+D8F0
                ADD    HL,BC
0A91 NO-NUMBER  JP     C,0D9A,REPORT-C
                CP     A
                JP     14BC,SET-MEM
```

THE 'REPORT & LINE NUMBER' PRINTING SUBROUTINES

The OUT-NUM. entry point is used to print the error report line numbers and the OUT-NO. entry point is used for printing line numbers at the start of BASIC lines.

```
0A98 OUT-NUM.   PUSH   DE
                PUSH   HL
                XOR    A
                BIT    7,B
                JR     NZ,0ABF,UNITS
                LD     H,B
                LD     L,C
                LD     E,+FF
                JR     0AAD,THOUSAND
0AA5 OUT-NO.    PUSH   DE
                LD     D,(HL)
                INC    HL
                LD     E,(HL)
                PUSH   HL
                EX     DE,HL
                LD     E,+00
0AAD THOUSAND   LD     BC,+FC18
                CALL   07E1,OUT-DIGIT
                LD     BC,FF9C
                CALL   07E1,OUT-DIGIT
                LD     C,+F6
                CALL   07E1,OUT-DIGIT
                LD     A,L
0ABF UNITS      CALL   07EB,OUT-CODE
                POP    HL
                POP    DE
                RET
```

THE 'UNSTACK-Z' SUBROUTINE

Bit 7 of FLAGS is set during the execution of a BASIC line but reset during syntax checking. This subroutine calls SYNTAX-Z and then either simply 'returns' using a JP (HL) instruction during the execution of a BASIC line, or uses a RET Z instruction to 'return' to the address above on the stack during syntax checking.

```
0AC5 UNSTACK-Z  CALL   0DA6,SYNTAX-Z
                POP    HL
                RET    Z
                JP     (HL)
```

THE 'LPRINT' COMMAND ROUTINE

Bit 1 of FLAGS is set whenever a LPRINT command is executed.

```
0ACB LPRINT     SET    1,(FLAGS)
```

THE 'PRINT' COMMAND ROUTINE

This routine is fairly complex but fortunately it can be broken into simple parts.

i) Test for PRINT alone.

```
0ACF PRINT      LD     A,(HL)
                CP     +76
                JP     Z,0B84,PRINT-END
```

ii) A loop is now set up to deal with each constituent part of a PRINT line.

First, the next character is tested to see if it is a 'comma' or a 'semi-colon'.

```
0AD5 PRINT-1    SUB    +1A
                ADC    A,+00
                JR     Z,0B44,SPACING
```

iii) If the next character is an 'AT' it is dealt with as follows:

Test for 'AT'

```
                CP     +A7
                JR     NZ,0AFA,NOT-AT
```

The next character is collected.

```
                RST    0020,NEXT-CH.
```

The next expression is identified.

```
                CALL   0D92,CLASS-6
```

A test is made for the correct separator — a comma.

```
                CP     +1A
                JP     NZ,0D9A,REPORT-C
```

The next character is collected.

```
                RST    0020,NEXT-CH
```

The next expression is identified.

```
                CALL   0D92,CLASS-6
```

A test is made to see if a line is being executed or syntax being checked. An indirect jump is made to PRINT-ON if syntax is being checked.

```
                CALL   0B4E,SYNTAX-ON
```

The particulars of the two expressions are both on the calculator stack but they need to be switched over. This is done using a RST 0028 instruction and the literal 01.

```
                RST     0028,FP-CALC.
                DEFB    +01
                        (exchange, 1A72)
                DEFB    +34
                        (end-calc.,002B)
```

The two expressions on the stack are then 'loaded' into the BC register pair by calling STK-TO-BC.

```
                CALL    0BF5,STK-TO-BC
```

With the PRINT AT parameters now in BC the usual routine can be called to set DF-CC & S-POSN and a jump is then made to PRINT-ON.

```
                CALL    08F5,PRINT-AT
                JR      0B37,PRINT-ON
```

iv) If the next character is a 'TAB' it is dealt with as follows:

Test for 'TAB'

```
0AFA NOT-AT     CP      +A8
                JR      NZ,0B31,NOT-TAB
```

The single 'following expression' is collected. The syntax flag is checked and the value of the expression 'loaded' into the A register.

```
                RST     0020,NEXT-CH.
                CALL    0D92,CLASS-6
                CALL    0B4E,SYNTAX-ON
                CALL    0C02,STK-TO-A
```

The 'parameter' is then tested and the new values of DF-CC & S-POSN are found by calling TEST-VAL.

```
                JP      NZ,0EAD,REPORT-B
                AND     +1F
                LD      C,A
                BIT     1,(FLAGS)
                JR      Z,0B1E,TAB-TEST
                SUB     (PR-CC)
                SET     7,A
                ADD     A,+3C
                CALL    NC,0871,COPY-BUFF
0B1E TAB-TEST   ADD     A,(S-POSN-lo.)
                CP      +21
                LD      A,(S-POSN-hi.)
                SBC     A,+01
                CALL    08FA,TEST-VAL
                SET     0,(FLAGS)
                JR      0B37,PRINT-ON
```

v) The expression that comes next is collected and printed by using the PRINT-STK subroutine.

```
0B31 NOT-TAB    CALL    0F55,SCANNING
                CALL    0B55,PRINT-STK
```

vi) The routine now proceeds to check for another expression.

```
0B37 PRINT-ON   RST     0018,GET-CH.
                SUB     +1A
                ADC     A,+00
                JR      Z,0B44,SPACING
                CALL    0D1D,CHECK-END
                JP      0B84,PRINT-END
```

vii) The two characters 'comma & semi-colon' are now separated.

```
0B44 SPACING    CALL    NC,0B8B,FIELD
                RST     0020,NEXT-CH.
                CP      +76
                RET     Z
                JP      0AD5,PRINT-1
```

viii) The SYNTAX-ON subroutine causes a jump to PRINT-ON if syntax is being checked.

```
0B4E SYNTAX-ON  CALL    0DA6,SYNTAX-Z
                RET     NZ
                POP     HL
                JR      OB37,PRINT-ON
```

ix) The PRINT-STK routine collects the details of a string from the calculator stack. A number is dealt with by jumping to PRINT-FP, whereas a string is dealt with in the 'print string' section. First the syntax flag is read.

```
0B55 PRINT-STK  CALL    0AC5,UNSTACK-Z
                BIT     6,(FLAGS)
                CALL    Z,13F8,STK-FETCH
                JR      Z,0B6B,PR-STR-4
                JP      15DB,PRINT-FP
```

x) The string printing routine.

The length of the string is held in the BC register pair and the starting address of the string is held in the DE register pair.

```
0B64 PR-STR-1   LD      A,+0B
0B66 PR-STR-2   RST     0010,PRINT-A
0B67 PR-STR-3   LD      DE,(X-PTR)
0B6B PR-STR-4   LD      A,B
                OR      C
                DEC     BC
                RET     Z
                LD      A,(DE)
                INC     DE
                LD      (X-PTR),DE
                BIT     6,A
                JR      Z,0B66,PR-STR-2
                CP      +CO
                JR      Z,0B64,PR-STR-1
                PUSH    BC
                CALL    094B,TOKENS
                POP     BC
                JR      0B67,PR-STR-3
```

xi) The PRINT-END routine.

The syntax flag is read and a N/L character is printed during line execution.

```
0B84 PRINT-END CALL   0AC5,UNSTACK-Z
               LD     A,+76
               RST    0010,PRINT-A
               RET
```

xii) The FIELD subroutine.

The appropriate value of S-POSN (and PR-CC if required) is found.

```
0B8B FIELD     CALL   0AC5,UNSTACK-Z
               SET    O,(FLAGS)
               XOR    A
               RST    0010,PRINT-A
               LD     BC,(S-POSN)
               LD     A,C
               BIT    1,(FLAGS)
               JR     Z,0BA4,CENTRE
               LD     A,+5D
               SUB    (PR-CC)
0BA4 CENTRE    LD     C,+11
               CP     C
               JR     NC,0BAB,RIGHT
               LD     C,+01
0BAB RIGHT     CALL   090B,SET-FIELD
               RET
```

THE 'PLOT & UNPLOT' COMMAND ROUTINES

Initially the x & y co-ordinates are fetched and tested. Then they are converted to row & column numbers. The value formed in the A register distinguishes which pixel is being identified.

```
0BAF PLOT/UNP. CALL   0BF5,STK-TO-BC
               LD     (COORDS),BC
               LD     A,+2B
               SUB    B
               JP     C,0EAD,REPORT-B
               LD     B,A
               LD     A,+01
               SRA    B
               JR     NC,0BC5,COLUMNS
               LD     A,+04
0BC5 COLUMNS   SRA    C
               JR     NC,0BCA,FIND-ADDR
               RLCA
0BCA FIND-ADDR PUSH   AF
               CALL   08F5,PRINT-AT
               LD     A,(HL)
               RLCA
               CP     +10
               JR     NC,0BDA,TABLE-PTR
               RRCA
               JR     NC,0BD9,SQ-SAVED
               XOR    +8F
0BD9 SQ-SAVED  LD     B,A
0BDA TABLE-PTR LD     DE,+0C9E
```

The two operations of PLOTting and UNPLOTting are distinguished by referring to T-ADDR and comparing the value against the constant 0C9E that is the value of the address of the UNPLOT command in the syntax table.

```
               LD     A,(T-ADDR)
               SUB    E
               JP     M,0BE9,PLOT
               POP    AF
               CPL
               AND    B
               JR     0BEB,UNPLOT
0BE9 PLOT      POP    AF
               OR     B
0BEB UNPLOT    CP     +08
               JR     C,0BF1,PLOT-END
               XOR    +8F
0BF1 PLOT-END  EXX
               RST    0010,PRINT-A
               EXX
               RET
```

THE 'STK-TO-BC' SUBROUTINE

This subroutine 'loads' two floating point numbers into the BC register pair. The subroutine is therefore used to pick up parameters in the range 00-FF.

```
0BF5 STK-TO-BC CALL   0C02,STK-TO-A
               LD     B,A
               PUSH   BC
               CALL   0C02,STK-TO-A
               LD     E,C
               POP    BC
               LD     D,C
               LD     C,A
               RET
```

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.

```
0C02 STK-TO-A  CALL   15CD,FP-TO-A
               JP     C,0EAD,REPORT-B
               LD     C,+01
               RET    Z
               LD     C,+FF
               RET
```

THE 'SCROLL' COMMAND ROUTINE

The first part of the routine sets the correct values of DF-CC and S-POSN to allow for the next printing to occur at the start of the bottom line + 1.

Next the end address of the first line in the display file is identified and the whole of the display file moved to overwrite this line.

```
0C0E SCROLL    LD     B,(DF-SZ)
               LD     C,+21
               CALL   0918,LOC.-ADDR
               CALL   099B,ONE-SPACE
               LD     A,(HL)
               LD     (DE),A
               INC    (S-POSN-hi.)
               LD     HL,(D-FILE)
               INC    HL
               LD     D,H
               LD     E,L
               CPIR
               JP     0A5D,RECLAIM-1
```

THE SYNTAX TABLES

i) The offset table.

There is an offset value for each of the BASIC commands and by adding this offset to the value of the address where it is found, the correct address for the command in the parameter table is obtained.

0C29	8B	LPRINT	0CB4
0C2A	8D	LLIST	0CB7
0C2B	2D	STOP	0C58
0C2C	7F	SLOW	0CAB
0C2D	81	FAST	0CAE
0C2E	49	NEW	0C77
0C2F	75	SCROLL	0CA4
0C30	5F	CONT	0C8F
0C31	40	DIM	0C71
0C32	42	REM	0C74
0C33	2B	FOR	0C5E
0C34	17	GOTO	0C4B
0C35	1F	GOSUB	0C54
0C36	37	INPUT	0C6D
0C37	52	LOAD	0C89
0C38	45	LIST	0C7D
0C39	0F	LET	0C48
0C3A	6D	PAUSE	0CA7
0C3B	2B	NEXT	0C66
0C3C	44	POKE	0C80
0C3D	2D	PRINT	0C6A
0C3E	5A	PLOT	0C98
0C3F	3B	RUN	0C7A
0C40	4C	SAVE	0C8C
0C41	45	RAND	0C86
0C42	0D	IF	0C4F
0C43	52	CLS	0C95
0C44	5A	UNPLOT	0C9E
0C45	4D	CLEAR	0C92
0C46	15	RETURN	0C5B
0C47	6A	COPY	0CB1

ii) The parameter table.

For each of the BASIC commands there are between 3 & 8 entries in the parameter table. The command classes for each of the commands are given, together with the required separators and these are followed by the address of the appropriate routine.

0C48 P-LET	01	CLASS-1
	14	'='
	02	CLASS-2
0C4B P-GOTO	06	CLASS-6
	00	CLASS-0
	81	
	0E	GOTO,0E81
0C4F P-IF	06	CLASS-6
	DE	'THEN'
	05	CLASS-5
	AB	
	0D	IF,0DAB
0C54 P-GOSUB	06	CLASS-6
	00	CLASS-0
	B5	
	0E	GOSUB,0EB5
0C58 P-STOP	00	CLASS-0
	DC	
	0C	STOP,0CDC
0C5B P-RETURN	00	CLASS-0
	D8	
	0E	RETURN,0ED8
0C5E P-FOR	04	CLASS-4
	14	'='
	06	CLASS-6
	DF	'TO'
	06	CLASS-6
	05	CLASS-5
	B9	
	0D	FOR,0DB9
0C66 P-NEXT	04	CLASS-4
	00	CLASS-0
	2E	
	0E	NEXT,0E2E
0C6A P-PRINT	05	CLASS-5
	CF	
	0A	PRINT,0ACF
0C6D P-INPUT	01	CLASS-1
	00	CLASS-0
	E9	
	0E	INPUT,0EE9
0C71 P-DIM	05	CLASS-5
	09	
	14	DIM,1409
0C74 P-REM	05	CLASS-5
	6A	
	0D	REM,0D6A
0C77 P-NEW	00	CLASS-0
	C3	
	03	NEW,03C3
0C7A P-RUN	03	CLASS-3
	AF	
	0E	RUN,0EAF
0C7D P-LIST	03	CLASS-3
	30	
	07	LIST,0730
0C80 P-POKE	06	CLASS-6
	1A	','
	06	CLASS-6
	92	
	0E	POKE,0E92
0C86 P-RAND	03	CLASS-3
	6C	
	0E	RAND,0E6C
0C89 P-LOAD	05	CLASS-5
	40	
	03	LOAD,0340

```
0C8C P-SAVE     05   CLASS-5
                F6
                02   SAVE,02F6

0C8F P-CONT     00   CLASS-0
                7C
                0E   CONT,0E7C

0C92 P-CLEAR    00   CLASS-0
                9A
                14   CLEAR,149A

0C95 P-CLS      00   CLASS-0
                2A
                0A   CLS,0A2A

0C98 P-PLOT     06   CLASS-6
                1A   ','
                06   CLASS-6
                00   CLASS-0
                AF
                0B   PLOT/UNP.,0BAF

0C9E P-UNPLOT   06   CLASS-6
                1A   ','
                06   CLASS-6
                00   CLASS-0
                AF
                0B   PLOT/UNP.,0BAF

0CA4 P-SCROLL   00   CLASS-0
                0E
                0C   SCROLL,0C0E

0CA7 P-PAUSE    06   CLASS-6
                00   CLASS-0
                32
                0F   PAUSE,0F32

0CAB P-SLOW     00   CLASS-0
                2B
                0F   SLOW,0F2B

0CAE P-FAST     00   CLASS-0
                23
                0F   FAST,0F23

0CB1 P-COPY     00   CLASS-0
                69
                08   COPY,0869

0CB4 P-LPRINT   05   CLASS-5
                CB
                0A   LPRINT,0ACB

0CB7 P-LLIST    03   CLASS-3
                2C
                07   LLIST,072C
```

THE 'LINE SCANNING' ROUTINE

The BASIC interpreter scans each line for BASIC commands and as each one is found the appropriate command routine is followed.

The different parts of the routine are:

i) The LINE-SCAN entry point leads to the line number being checked for validity.

```
0CBA LINE-SCAN LD      (FLAGS),+01
               CALL    0A73,E-LINE-NO
```

ii) The LINE-RUN entry point is used when replying to an INPUT prompt and this fact has to be identified.

```
0CC1 LINE-RUN  CALL    14BC,SET-MEM
               LD      HL,+ERR-NR
               LD      (HL),+FF
               LD      HL,+FLAGX
               BIT     5,(HL)
               JR      Z,0CDE,LINE-NULL
```

iii) The INPUT reply is tested to see if STOP was entered.

```
               CP      +E3
               LD      A,(HL)
               JP      NZ,0D6F,INPUT-REP
               CALL    0DA6,SYNTAX-Z
               RET     Z
```

iv) If appropriate, report D is given.

```
               RST     0008,ERROR-1
               DEFB    +0C
```

THE 'STOP' COMMAND ROUTINE

The only action is to give report 9.

```
0CDC STOP      RST     0008,ERROR-1
               DEFB    +08
```

v) A return is made if the line is 'null'.

```
0CDE LINE-NULL RST     0018,GET-CH.
               LD      B,+00
               CP      +76
               RET     Z
```

vi) The first character is tested so as to check that it is a command.

```
               LD      C,A
               RST     0020,NEXT-CH.
               LD      A,C
               SUB     +E1
               JR      C,0D26,REPORT-C2
```

vii) The offset for the command is found from the offset table.

```
               LD      C,A
               LD      HL,+0C29
               ADD     HL,BC
               LD      C,(HL)
               ADD     HL,BC
               JR      0CF7,GET-PARAM
```

viii) The parameters are fetched in turn by a loop that returns to 0CF4.

The separators are identified by the test against +0B.

```
0CF4 SCAN-LOOP LD      HL,(T-ADDR)
0CF7 GET-PARAM LD      A,(HL)
```

```
                    INC     HL
                    LD      (T-ADDR),HL
                    LD      BC,+0CF4
                    PUSH    BC
                    LD      C,A
                    CP      +0B
                    JR      NC,0D10,SEPARATOR
```

ix) The address of the command class routine is obtained by reference to the command class table at 0D16. A jump is made to the appropriate routine.

```
                    LD      HL,+0D16
                    LD      B,+00
                    ADD     HL,BC
                    LD      C,(HL)
                    ADD     HL,BC
                    PUSH    HL
                    RST     0018,GET-CH.
                    RET
```

x) The correctness of the separator is simply tested by the following routine.

```
0D10 SEPARATOR      RST     0018,GET-CH.
                    CP      C
                    JR      NZ,0D26,REPORT-C2
                    RST     0020,NEXT-CH.
                    RET
```

THE COMMAND CLASS TABLE

The addresses for the seven different command classes are found from this table.

0D16	17	CLASS-0,0D2D
0D17	25	CLASS-1,0D3C
0D18	53	CLASS-2,0D6B
0D19	0F	CLASS-3,0D28
0D1A	6B	CLASS-4.0D85
0D1B	13	CLASS-5,0D2E
0D1C	76	CLASS-6,0D92

THE 'CHECK-END' SUBROUTINE

Line scanning is finished when the N/L character is reached.

```
0D1D CHECK-END      CALL    0DA6,SYNTAX-Z
                    RET     NZ
                    POP     BC
0D22 CHECK-2        LD      A,(HL)
                    CP      +76
                    RET     Z
0D26 REPORT-C2      JR      0D9A,REPORT-C
```

THE 'COMMAND CLASS 3' ROUTINE

The commands RUN, LIST, RAND and LLIST can be followed by a N/L or a number.

```
0D28 CLASS-3        CP      +76
                    CALL    0D9C,NO-TO-STK
```

THE 'COMMAND CLASS 0' ROUTINE

An entry here will cause the zero flag to be set prior to a call to CHECK-END.

```
0D2D CLASS-0        CP      A
```

THE 'COMMAND CLASS 5' ROUTINE

The commands IF, FOR, PRINT, DIM, REM, LOAD, SAVE and LPRINT all have class 5 as their last command class. A jump is made to the command routine directly.

```
0D2E CLASS-5        POP     BC
                    CALL    Z,0D1D,CHECK-END
                    EX      DE,HL
                    LD      HL,(T-ADDR)
                    LD      C,(HL)
                    INC     HL
                    LD      B,(HL)
                    EX      DE,HL
0D3A CLASS-END      PUSH    BC
                    RET
```

THE 'COMMAND CLASS 1' ROUTINE

The commands LET and INPUT both require that a variable be specified. The command class 1 routine collects the details of the variable and stores them in the required places.

```
0D3C CLASS-1        CALL    111C,LOOK-VARS
0D3F CLASS-4-2      LD      (FLAGX),+00
                    JR      NC,0D4D,SET-STK
                    SET     1,(FLAGX)
                    JR      NZ,0D63,SET—STRLN
0D4B REPORT-2       RST     0008,ERROR-1
                    DEFB    +01
0D4D SET-STK        CALL    Z,11A7,STK-VAR
                    BIT     6,(FLAGS)
                    JR      NZ,0D63,SET-STRLN
                    XOR     A
                    CALL    0DA6,SYNTAX-Z
                    CALL    NZ,13F8,STK-FETCH
                    LD      HL,+FLAGX
                    OR      (HL)
                    LD      (HL),A
                    EX      DE,HL
0D63 SET-STRLN      LD      (STRLEN),BC
                    LD      (DEST),HL
0D6A REM            RET
```

THE 'COMMAND CLASS 2' ROUTINE

The value assigned to a variable in a LET command or in reply to an INPUT prompt is evaluated by calling SCANNING. If the value is appropriate then an indirect jump is made to the LET routine at 1321.

```
0D6B CLASS-2        POP     BC
                    LD      A,(FLAGS)
0D6F INPUT-REP      PUSH    AF
                    CALL    0F55,SCANNING
                    POP     AF
                    LD      BC,+1321
                    LD      D,(FLAGS)
```

```
                XOR     D
                AND     +40
                JR      NZ,0D9A,REPORT-C
                BIT     7,D
                JR      NZ,0D3A,CLASS-END
                JR      0D22,CHECK-2
```

THE 'COMMAND CLASS 4' ROUTINE

The specified variable for the FOR and the NEXT commands are dealt with by this routine. Only single character variables are allowed and these are identified by their having both bits 5 & 6 set.

```
0D85 CLASS-4    CALL    111C,LOOK-VARS
                PUSH    AF
                LD      A,C
                OR      +9F
                INC     A
                JR      NZ,0D9A,REPORT-C
                POP     AF
                JR      0D3F,CLASS-4-2
```

THE 'COMMAND CLASS 6' ROUTINE

CLASS-6 denotes that the following expression must yield an integer value.

The SCANNING routine evaluates the expression and a numeric value will give bit 6 of FLAGS set.

```
0D92 CLASS-6    CALL    0F55,SCANNING
                BIT     6,(FLAGS)
                RET     NZ
```

REPORT-C — no numeric value.

```
0D9A REPORT-C   RST     0008,ERROR-1
                DEFB    +0B
```

THE 'NO-TO-STK' SUBROUTINE

During execution of a line this routine leads to a number being placed on the calculator stack. If the zero flag is reset on entry the number put on the stack will be the result of evaluating the 'next' expression, but if the zero flag is set then zero will be placed on the stack by using a RST 0028 instruction.

```
0D9C NO-TO-STK  JR      NZ,0D92,CLASS-6
                CALL    0DA6,SYNTAX-Z
                RET     Z
                RST     0028,FP-CALC.
                DEFB    +A0
                        (stk-zero,1A51)
                DEFB    +34
                        (end-calc.,002B)
                RET
```

THE 'SYNTAX-Z' SUBROUTINE

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.

```
0DA6 SYNTAX-Z   BIT     7,(FLAGS)
                RET
```

THE 'IF' COMMAND ROUTINE

At this point the value of the expression between the 'IF' and the 'THEN' is known, and is on the top of the calculator stack.

During execution the result is deleted from the stack but the pointer DE is still available. The logical value of (DE) is tested and a return made if zero, otherwise the routine jumps to LINE-NULL to execute the rest of the line.

```
0DAB IF         CALL    0DA6,SYNTAX-Z
                JR      Z,0DB6,IF-END
                RST     0028,FP-CALC.
                DEFB    +02
                        (delete,19E3)
                DEFB    +34
                        (end-calc.,002B)
                LD      A,(DE)
                AND     A
                RET     Z
0DB6 IF-END     JP      0CDE,LINE-NULL
```

THE 'FOR' COMMAND ROUTINE

This routine is made up of the following parts:

i) If a STEP variable is given then this is found and put on the stack, otherwise the value one is used.

```
0DB9 FOR        CP      +E0
                JR      NZ,0DC6,USE-ONE
                RST     0020,NEXT-CH.
                CALL    0D92,CLASS-6
                CALL    0D1D,CHECK-END
                JR      0DCC,REORDER
0DC6 USE-ONE    CALL    0D1D,CHECK-END
                RST     0028,FP-CALC.
                DEFB    +A1
                        (stk-one,1A51)
                DEFB    +34
                        (end-calc.,002B)
```

ii)
The top three values on the stack, the 'value', the 'limit' & the 'step' are re-ordered to give 'limit-step-value'.

```
0DCC REORDER    RST     0028,FP-CALC.
                DEFB    +C0
                        (st-mem-0,1A63)
                DEFB    +02
                        (delete,19E3)
                DEFB    +01
                        (exchange,1A72)
                DEFB    +E0
                        (get-mem-0,1A45)
                DEFB    +01
                        (exchange,1A72)
                DEFB    +34
                        (end-calc.,002B)
```

iii) The LET routine is used to locate an address in the VARS area for the FOR variable. If the variable already exists then it is overwritten, if not then the variable is added to the end of the VARS. The 'limit' & the 'step' are then transferred.

```
                        CALL    1321,LET
                        LD      (MEM),HL
                        DEC     HL
                        LD      A,(HL)
                        SET     7,(HL)
                        LD      BC,+0006
                        ADD     HL,BC
                        RLCA
                        JR      C,0DEA,LMT+STEP
                        SLA     C
                        CALL    099E,MAKE-ROOM
                        INC     HL
0DEA LMT+STEP           PUSH    HL
                        RST     0028,FP-CALC.
                        DEFB    +02
                                (delete,19E3)
                        DEFB    +02
                                (delete,19E3)
                        DEFB    +34
                                (end-calc.,002B)
                        POP     HL
                        EX      DE,HL
                        LD      C,+0A
                        LDIR
```

iv) The current line number is fetched, incremented and added to the variable.

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

v) The NEXT-LOOP subroutine is called to check that a 'looping' is possible. If it is not possible then NXTLIN is set to the appropriate line number for jumping over the whole of the FOR-NEXT loop.

```
                        CALL    0E5A,NEXT-LOOP
                        RET     NC
                        BIT     7,(PPC-hi.)
                        RET     NZ
                        LD      B,(STRLEN)
                        RES     6,B
                        LD      HL,(NXTLIN)
0E0E NXTLIN-NO          LD      A,(HL)
                        AND     +C0
                        JR      NZ,0E2A,FOR-END
                        PUSH    BC
                        CALL    09F2,NEXT-ONE
                        POP     BC
                        INC     HL
                        INC     HL
                        INC     HL
                        CALL    004C,CURSOR-SO
                        RST     0018,GET-CH.
                        CP      +F3
                        EX      DE,HL
                        JR      NZ,0E0E,NXTLIN-NO
                        EX      DE,HL
                        RST     0020,NEXT-CH
                        EX      DE,HL
                        CP      B
                        JR      NZ,0E0E,NXTLIN-NO
0E2A FOR-END            LD      (NXTLIN),HL
                        RET
```

THE 'NEXT' COMMAND ROUTINE

In this routine the address of the variable is collected from DEST. Next MEM is loaded with this address so that a RST 0028 instruction can be used to manipulate the different parts of the variable when the 'step' is added to the 'value'.

```
0E2E NEXT               BIT     1,(FLAGX)
                        JP      NZ,0D4B,REPORT-2
                        LD      HL,(DEST)
                        BIT     7,(HL)
                        JR      Z,0E58,REPORT-1
                        INC     HL
                        LD      (MEM),HL
                        RST     0028,FP-CALC.
                        DEFB    +E0
                                (get-mem-0,1A45)
                        DEFB    +E2
                                (get-mem-2,1A45)
                        DEFB    +0F
                                (addition,1755)
                        DEFB    +C0
                                (st-mem-0,1A63)
                        DEFB    +02
                                (delete,19E3)
                        DEFB    +34
                                (end-calc.,002B)
                        CALL    0E5A,NEXT-LOOP
                        RET     C
```

An indirect jump is now made to the line number given in the last two bytes of the variable.

```
                        LD      HL,(MEM)
                        LD      DE,+000F
                        ADD     HL,DE
                        LD      E,(HL)
                        INC     HL
                        LD      D,(HL)
                        EX      DE,HL
                        JR      0E86,GOTO-2
```

REPORT-1 — 'NEXT' without 'FOR' error

```
0E58 REPORT-1           RST     0008,ERROR-1
                        DEFB    +00
```

THE 'NEXT-LOOP' SUBROUTINE

This subroutine is called by both the 'FOR' and the 'NEXT' command routines.

When called by the 'FOR' routine it determines whether or not a jump past the whole of the FOR-NEXT loop is to be made.

When called by the 'NEXT' command routine it determines whether another loop is, or is not, possible.

The routine tests the 'step' and then compares the 'limit' and the 'value'. The carry flag is set, or reset, as required.

```
0E5A NEXT-LOOP RST    0028,FP-CALC.
               DEFB   +E1
                      (get-mem-1,1A45)
               DEFB   +E0
                      (get-mem-0,1A45)
               DEFB   +E2
                      (get-mem-2,1A45)
               DEFB   +32
                      (less-0,1ACE)
               DEFB   +00
                      (jump-true,1C2F)
               DEFB   +02, to 0E62
               DEFB   +01
                      (exchange,1A72)
0E62 LMT-V-VAL DEFB   +03
                      (subtract,174C)
               DEFB   +33
                      (greater-0,1ADB)
               DEFB   +00
                      (jump-true,1C2F)
               DEFB   +04, to 0E69)
               DEFB   +34
                      (end-calc.,002B)
               AND    A
               RET
0E69 IMPOSS.   DEFB   +34
                      (end-calc.,002B)
               SCF
               RET
```

THE 'RAND' COMMAND ROUTINE

The FIND-INT. subroutine is called to show whether a number was given with the RAND command. If not then FRAMES is used.

```
0E6C RAND      CALL   0EA7,FIND-INT.
               LD     A,B
               OR     C
               JR     NZ,0E77,SET-SEED
               LD     BC,(FRAMES)
0E77 SET-SEED  LD     (SEED),BC
               RET
```

THE 'CONT' COMMAND ROUTINE

The value of OLDPPC is fetched and used.

```
0E7C CONT      LD     HL,(OLDPPC)
               JR     0E86,GOTO-2
```

THE 'GOTO' COMMAND ROUTINE

The line number is collected, tested and then passed to LINE-ADDR. The address returned is loaded into NXTLIN.

```
0E81 GOTO      CALL   0EA7, FIND-INT.
               LD     H,B
               LD     L,C
0E86 GOTO-2    LD     A,H
               CP     +F0
               JR     NC,0EAD,REPORT-B
               CALL   09D8,LINE-ADDR.
               LD     (NXTLIN),HL
               RET
```

THE 'POKE' COMMAND ROUTINE

The value to be entered is collected from the stack using FP-TO-A and the address of the location to be filled is collected using FIND-INT.

```
0E92 POKE      CALL   15CD,FP-TO-A
               JR     C,0EAD,REPORT-B
               JR     Z,0E9B,POKE-SAVE
               NEG
0E9B POKE-SAVE PUSH   AF
               CALL   0EA7,FIND-INT.
               POP    AF
               BIT    7,(ERR-NR)
               RET    Z
               LD     (BC),A
               RET
```

THE 'FIND-INT.' SUBROUTINE

The integer value of the floating point number on the top of the stack is found. Report B is given if the value exceeds 65535 decimal.

```
0EA7 FIND-INT. CALL   158A,FP-TO-BC
               JR     C,0EAD,REPORT-B
               RET    Z
```

REPORT-B — integer out of range

```
0EAD REPORT-B  RST    0008,ERROR-1
               DEFB   +0A
```

THE 'RUN' COMMAND ROUTINE

The line number is determined and a jump made to the CLEAR command routine.

```
0EAF RUN       CALL   0E81,GOTO
               JP     149A,CLEAR
```

THE 'GOSUB' COMMAND ROUTINE

The current line number is fetched, incremented and stacked. The line number of the subroutine is determined and the registers set up for the TEST-ROOM subroutine.

```
0EB5 GOSUB     LD     HL,(PPC)
               INC    HL
               EX     (SP),HL
               PUSH   HL
               LD     (ERR-SP),SP
               CALL   0E81,GOTO
               LD     BC,+0006
```

THE 'TEST-ROOM' SUBROUTINE

This subroutine tests the value of STKEND against the stack pointer allowing 36 bytes for other variables. Report 4 is given if there is insufficient room.

```
0EC5 TEST-ROOM LD    HL,(STKEND)
               ADD   HL,BC
               JR    C,0ED3,REPORT-4
               EX    DE,HL
               LD    HL,+0024
               ADD   HL,DE
               SBC   HL,SP
               RET   C
```

REPORT-4 — out of RAM

```
0ED3 REPORT-4  LD    L,+03
               JP    0058,ERROR-3
```

THE 'RETURN' COMMAND ROUTINE

The 'return' line number is taken off the 'gosub stack' and tested to show that it is a real line number. Report 7 is given if there is a mistake.

```
0ED8 RETURN    POP   HL
               EX    (SP),HL
               LD    A,H
               CP    +3E
               JR    Z,0EE5,REPORT-7
               LD    (ERR-SP),SP
               JR    0E86,GOTO-2
```

REPORT-7 — return without gosub

The stack is restored and report 7 given.

```
0EE5 REPORT-7  EX    (SP),HL
               PUSH  HL
               RST   0008,ERROR-1
               DEFB  +06
```

THE 'INPUT' COMMAND ROUTINE

A test for report 8 is made and the workspace cleared. Then the appropriate prompt characters are printed and the cursor marker added. Finally a jump to LOWER is made so that the edit-line can be printed.

```
0EE9 INPUT     BIT   7,(PPC-hi.)
               JR    NZ,0F21,REPORT-8
               CALL  14A3,X-TEMP
               LD    HL,+FLAGX
               SET   5,(HL)
               RES   6,(HL)
               LD    A,(FLAGS)
               AND   +40
               LD    BC,+0002
               JR    NZ,0F05,PROMPT
               LD    C,+04
0F05 PROMPT    OR    (HL)
               LD    (HL),A
               RST   0030,BC-SPACES
               LD    (HL),+76
               LD    A,C
               RRCA
               RRCA
               JR    C,0F14,ENTER-CUR
               LD    A,+0B
               LD    (DE),A
               DEC   HL
               LD    (HL),A
0F14 ENTER-CUR DEC   HL
               LD    (HL),+7F
               LD    HL,(S-POSN)
               LD    (T-ADDR),HL
               POP   HL
               JP    0472,LOWER
```

REPORT-8 — input as direct command

```
0F21 REPORT-8  RST   0008,ERROR-1
               DEFB  +07
```

THE 'FAST' COMMAND ROUTINE

The SET-FAST routine is called to reset bit 7 of CDFLAG, and then bit 6 is reset.

```
0F23 FAST      CALL  02E7,SET-FAST
               RES   6,(CDFLAG)
               RET
```

THE 'SLOW' COMMAND ROUTINE

The 'true' slow/fast flag — bit 6 of CDFLAG is set and a jump made to SLOW/FAST that copies this flag to bit 7 for compute and display.

```
0F2B SLOW      SET   6,(CDFLAG)
               JP    0207,SLOW/FAST
```

THE 'PAUSE' COMMAND ROUTINE

The parameter of the PAUSE command is determined. Fast mode is selected for the period of the PAUSE and the DISPLAY-P routine called.

On returning the correct mode, SLOW or FAST, is selected and the value of FRAMES-hi. set to hex.FF. A jump to D-BOUNCE is then made.

Note: In the 'unimproved' ROM the value given to FRAMES-hi. was determined by a SET 7, (FRAMES-hi.) instruction and this failed to ensure that the 15th. bit of FRAMES would remain set as the first action of DISPLAY routine is to decrement FRAMES.

```
0F32 PAUSE     CALL  0EA7,FIND-INT.
               CALL  02E7,SET-FAST
               LD    H,B
               LD    L,C
               CALL  022D,DISPLAY-P
               CALL  0207,SLOW/FAST
               LD    (FRAMES-hi.),+FF
               JR    0F4B,D-BOUNCE
```

THE 'BREAK-1' SUBROUTINE

The 'break' key is tested.

```
0F46 BREAK-1      LD      A,+7F
                  IN      A,(+FE)
                  RRA
```

THE 'DEBOUNCE' SUBROUTINE

The system variable is set to its required valu of Hex.FF.

```
0F4B D-BOUNCE     RES     0,(CDFLAG)
                  LD      A,+FF
                  LD      (DEBOUNCE),A
                  RET
```

****

The forward references:

0F55	SCANNING
111C	LOOK-VARS
11A7	STK-VAR
1321	LET
13F8	STK-FETCH
1488	RESERVE
149A	CLEAR
14A3	X-TEMP
14A6	SET-STK-B
14AD	CURSOR-IN
14BC	SET-MEM
1548	INT-TO-FP
158A	FP-TO-BC
15CD	FP-TO-A
15DB	PRINT-FP
199D	CALCULATE

RST 0028 literals:

00	jump-true	1C2F
01	exchange	1A72
02	delete	19E3
03	subtract	174C
0F	addition	1755
32	less-0	1ADB
33	greater-0	1ACE
34	end-calc.	002B
A0	stk-zero	1A51
A1	stk-one	1A51
C0	st-mem-0	1A63
C1	st-mem-1	1A63
C2	st-mem-2	1A63
E0	get-mem-0	1A45
E1	get-mem-1	1A45
E2	get-mem-2	1A45

****

Notes on the SYSTEM VARIABLES

dec.	Hex.	Name.	Notes.
16384	4000	ERR-NR	The 'report code'. The value is incremented before being printed.
16385	4001	FLAGS	Bit 0 — suppression of leading space. Bit 1 — control flag for the printer. Bit 2 — selects K or L mode; or, F or G. Bit 6 — F.P. number or string parameters. Bit 7 — Reset during syntax checking.
16386	4002 4003	ERR-SP	Points to the GOSUB stack.
16388	4004 4005	RAMTOP	The top of available RAM, or as specified.
16390	4006	MODE	Holds the code for K or F.
16391	4007 4008	PPC	The line number of the current statement.
16393	4009	VERSN	Marks the start of RAM that is SAVEd.
16394	400A 400B	E-PPC	The BASIC line with the cursor.
16396	400C 400D	D-FILE	Pointer for the display file.
16398	400E 400F	DF-CC	Address for the PRINT AT position.
16400	4010 4011	VARS	Pointer for the variable area.
16402	4012 4013	DEST	The address of the current variable within the program area.
16404	4014 4015	E-LINE	The pointer for the workspace.
16406	4016 4017	CH-ADD	The pointer for scanning a line, either in the program area or the workspace.
16408	4018 4019	X-PTR	The syntax error address.
16410	401A 401B	STKBOT	The pointer for the bottom of the calculator stack.
16412	401C 401D	STKEND	The pointer for the top of the calculator stack.
16414	401E	BERG	A location used for many different counting purposes.
16415	401F 4020	MEM	The pointer to the base of a table of floating point numbers, either in the calculator stack or the variable area.

16417	4021	—	Not used.
16418	4022	DF-SZ	The number of lines in the lower screen.
16419	4023 4024	S-TOP	The current line number for the automatic listing.
16421	4025 4026	LAST-K	The 'key-value' of the last key that was pressed.
16423	4027	DEBOUNCE	The debounce status.
16424	4028	MARGIN	Adjusts for differing T.V. standards.
16425	4029 402A	NXTLIN	The line number of the next BASIC line to be interpreted.
16427	402B 402C	OLDPPC	The last line number is saved in case needed.
16429	402D	FLAGX	Bit 0 — **Reset** indicates an arrayed variable. Bit 1 — Reset indicates a given variable exists. Bit 5 — Set during INPUT mode. Bit 6 — Set when the INPUT is to be numeric.
16430	402E 402F	STRLEN	Length of string variable , or a BASIC line.
16432	4030 4031	T-ADDR	Pointer for the 'parameter' table. Also used to distinguish between PLOT and UNPLOT.
16434	4032 4033	SEED	The random function seed value.
16436	4034 4035	FRAMES	The counter for the frames.
16438	4036 4037	COORDS	The X & Y values of PLOT.
16440	4038	PR-CC	The counter for the printer buffer.
16441	4039 403A	S-POSN	The column and line numbers for PRINT AT.
16443	403B	CDFLAG	Bit 0 — set whenever a key is pressed. Bit 6 — the 'true' fast/slow flag. Bit 7 — the 'copy' of the fast/slow flag. It will be reset when FAST is needed.
16444	403C — 405C	PRBUFF	The printer buffer.
16477	405D — 407A	MEMBOT	A memory area that can hold 6 floating point numbers. (mem-0, ... mem-5.)
16507	407B 407C	—	Not used.
16509	407D	PROGRAM	The BASIC program starts here.

****

INDEX

COMMAND ROUTINES

RESTARTS

ROUTINES

SUBROUTINES

TABLES

The Complete Timex TS1000 & Sinclair ZX81 ROM Disassembly PART B: 0F55H-1DFFH

CONTENTS

Note: Readers of this book who are using machines fitted with the 'unimproved' ROM will have to bear the following points in mind.

* Three bytes were added at 0EEF, hence the code formerly at 0EEF-102E is now at 0EF2-1031.
* The code at 102F-1034 was rewritten using an extra location and is now at 1032-1038.
* The code formerly at 1035-1732 is now at 1039-1736.
* The three bytes formerly at 1733-1735 have been deleted, hence the code formerly at 1736-1DFD is now at 1737-1DFE.

AUTHORS' COMMENTS:

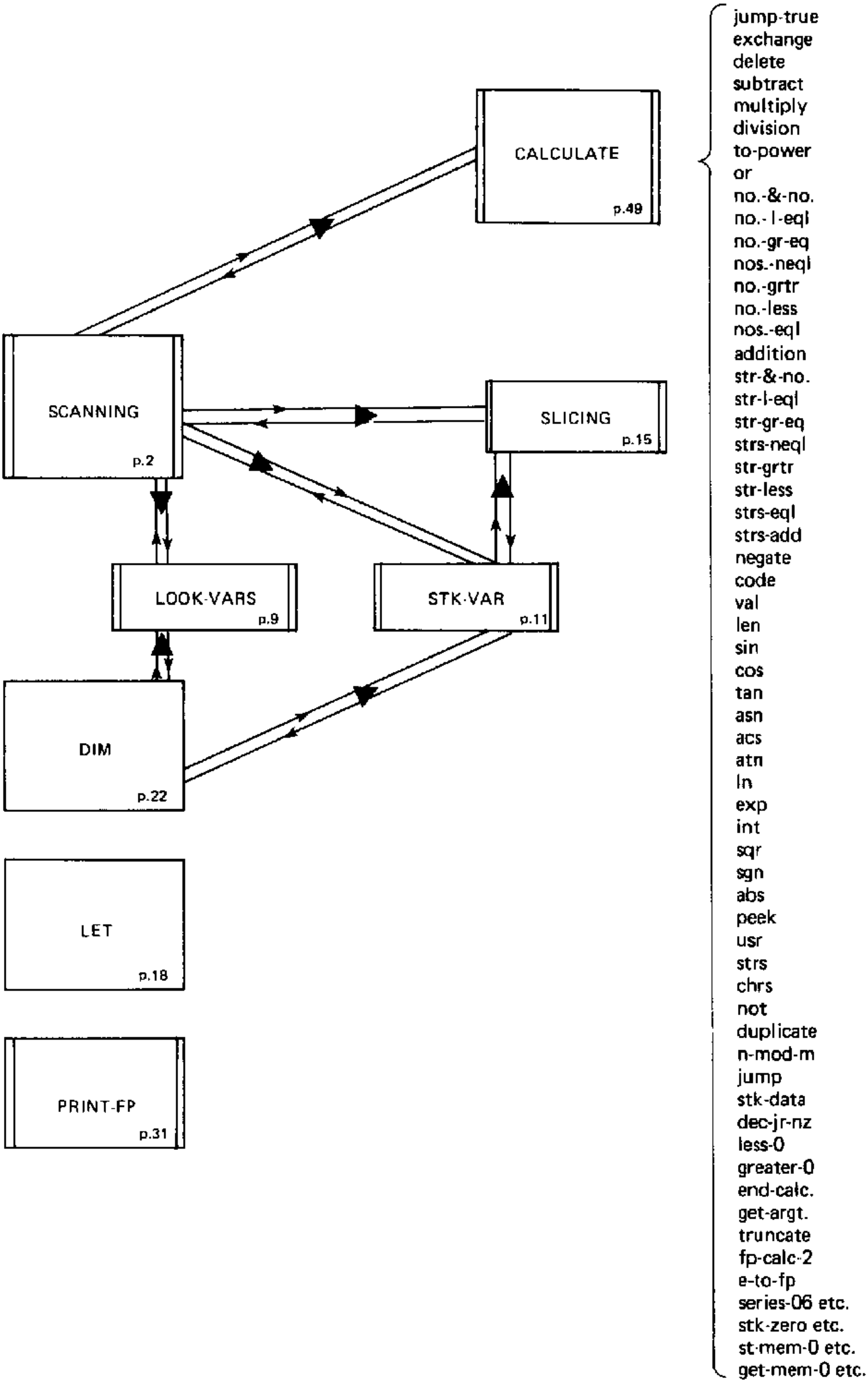
The production of this book has only been possible because of the immense help given by Dr. Frank O'Hara, to whom floating point arithmetic is almost second nature. I therefore wish to record my grateful thanks to Frank.

Ian Logan, Lincoln January 1982

I am very pleased to have been able to help Ian Logan sort out the arithmetic of the ZX81. I remain amazed at the ease with which he works out what the machine is doing, without any help from the people who designed the hardware or those who wrote the programs.

Frank O'Hara, London January 1982

THE 'FLOW DIAGRAM' FOR PART B of the 8K ROM Program



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)+38)..., a hierachy 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.

```
OF55  SCANNING  RST   0018,GET-CH.        The first character is fetched.
                LD    B,+00               The starting priority marker.
                PUSH  BC                  It is stacked.
```

The character is tested against the code for 'RND' and a jump made if it does not match.

```
OF59  S–RND     CP    +40                 Is it 'RND'?
                JR    NZ,0F8C,S-PI        Jump if it is not so.
```

Unless syntax is being checked the required random number is calculated and forms a 'last value' on the calculator stack.

```
                CALL  0DA6,SYNTAX-Z       Test for syntax checking.
                JR    Z,0F8A,S-RND-END    Jump if required.
                LD    BC,(SEED)           Fetch the current value of SEED.
                CALL  1520,STACK-BC       Put it on the calculator stack.
                RST   0028,FP-CALC.       Now use the calculator.
                DEFB  +A1,stk-one,1A51    The 'last value' is now
                DEFB  +0F,addition,1755   SEED+1.
                DEFB  +30,stk-data,19FC   Put the decimal number 75
                DEFB  +37,exponent 87     on the calculator stack.
                DEFB  +16,(+00,+00,+00)
                DEFB  +04,multiply,17C6   'Last value' = (SEED+1)*75.
                DEFB  +30,stk-data,19FC   See STACK LITERALS to see how
                DEFB  +80,four bytes      bytes are expanded so as to put the
                DEFB  +41,exponent 91     decimal number 65537 on the
                DEFB  +00,+00,+80,(+00)   calculator stack.
```

```
                    DEFB +2E,n-mod-m,1C37      Divide (SEED+1)*75 by 65537 to give
                                               a 'remainder' and an 'answer'.
                    DEFB +02,delete,19E3       Discard the 'answer'.
                    DEFB +A1,stk-one,1A51      The 'last value' is now
                    DEFB +03,subtract,174C     'remainder'-1.
                    DEFB +2D,duplicate,19F6    Make a copy of the 'last value'.
                    DEFB +34,end-calc.,002B    The calculation is finished.
                    CALL 158A,FP-TO-BC         Use the 'last value' to give the new
                    LD   (SEED),BC             value for SEED.
                    LD   A,(HL)                Fetch the exponent of 'last value'.
                    AND  A                     Jump forward if the exponent
                    JR   Z,0F8A,S-RND-END      is zero.
                    SUB  +10                   Reduce the exponent, i.e. divide 'last
                    LD   (HL),A                value' by 65536 to give the required
                                               'last value'.
0F8A S-RND-END      JR   0F99,S-PI-END         Jump past the 'PI' routine.
```

The character is tested against the code for 'PI' and a jump made if it does not match.

```
0F8C S-PI           CP   +42                   Is it 'PI'?
                    JR   NZ,0F9D,S-INKEY$      Jump if it is not so.
```

Unless syntax is being checked the value of 'PI' is calculated and forms the 'last value' on the calculator stack.

```
                    CALL 0DA6,SYNTAX-Z         Test for syntax checking.
                    JR   Z,0F99,S-PI-END       Jump if required.
                    RST  0028,FP-CALC.         Now use the calculator.
                    DEFB +A3,stk-pi/2,1A51     The value of PI/2 is put on the
                    DEFB +34,end-calc.,002B    calculator stack as the 'last value'.
                    INC  (HL)                  The exponent is incremented thereby
                                               doubling the 'last value' giving 'PI'.
0F99 S-PI-END       RST  0020,NEXT-CH          Move on to the next character.
                    JP   1083,S-NUMERIC        Jump forward.
```

The character is tested against the code for 'INKEY$' and a jump made if it does not match.

```
0F9D S-INKEY$       CP   +41                   Is it 'INKEY$'?
                    JR   NZ,0FB2,S-ALPHNUM     Jump if it is not so.
```

The keyboard is now scanned and the parameters for the INKEY$ string calculated. A null string will result in the BC register pair holding the value zero, whereas when a key has been pressed it holds the value one. The DE register pair points to the appropriate character in the key tables and that entry forms the actual string.

```
                    CALL 02BB,KEYBOARD         Scan the keyboard & reset carry flag.
                    LD   B,H                   Copy the 'key value' into the
                    LD   C,L                   BC register pair.
                    LD   D,C                   Set the zero flag when dealing with
                    INC  D                     a null string.
                    CALL NZ,07BD,DECODE        Decode the 'key value'. The carry flag is
                                               set when only one key is pressed.
                    LD   A,D                   D register always holds zero.
                    ADC  A,D                   A now holds the value of the carry.
                    LD   B,D                   Clears the B register.
                    LD   C,A                   C holds zero or one.
                    EX   DE,HL                 The start pointer goes into DE.
                    JR   0FED,S-STRING         Jump forward.
```

The character is tested to see if it is alphanumeric.

```
0FB2 S-ALPHNUM      CALL 14D2,ALPHANUM         Test the character.
                    JR   C,1025,S-LET-NUM      Jump if a letter or a digit.
```

The character is tested against the code for '.', hence identifying a decimal number without a leading zero.

```
                    CP    +1B                 Is it a '.' ?
                    JP    Z,1047,S-DECIMAL    Jump forward if it is so.
```

The character is tested against the code for '-', hence identifying the 'unary minus' operation.

Before the actual test the B register is set to hold the priority 9 and the C register the operation code D8 that are required for this operation.

```
                    LD    BC,+09D8            Priority 9, operation code D8.
                    CP    +16                 Is it a '-' ?
                    JR    Z,1020,S-PUSH-PO    Jump forward if it is 'unary minus'.
```

The character is tested against the code for '(', hence identifying the presence of a parenthesised expression.

```
                    CP    +10                 Is it a '(' ?
                    JR    NZ,0FD6,S-QUOTE     Jump if it is not so.
```

A parenthesised expression is dealt with in a recursive manner. An error is reported if there is no closing bracket.

```
                    CALL  0049,CH-ADD+1       Points to the next character.
                    CALL  0F55,SCANNING       Call the present subroutine.
                    CP    +11                 Is the present character a ')' ?
                    JR    NZ,0FFF,S-RPRT-C1   Report C if no closing bracket.
                    CALL  0049,CH-ADD+1       Point to the next character.
                    JR    0FF8,S-CONT-1       Jump forward.
```

The character is tested against the code for ' " ', hence identifying a string of characters.

```
0FD6  S-QUOTE       CP    +0B                 Is it a ' " ' ?
                    JR    NZ,1002,S-FUNCT     Jump if it is not so.
```

The parameters for this string of characters are now calculated.

```
                    CALL  0049,CH-ADD+1       Point to the next character.
                    PUSH  HL                  Save the 'start' address.
                    JR    0FE3,S-Q-END?       Jump past the re-entry point.
0FE0  S-Q-NEXT      CALL  0049,CH-ADD+1       Point to the next character.
0FE3  S-Q-END?      CP    +0B                 Is it another ' " ' ?
                    JR    NZ,0FFB,S-N/L-ERR   Before re-entering the loop check
                                              that the line has not been finished.
                    POP   DE                  Get the 'start' into DE.
                    AND   A                   Clear the carry flag.
                    SBC   HL,DE               Now find the 'length'.
                    LD    B,H                 Move the 'length' to the
                    LD    C,L                 BC register pair.
```

A string result has now been identified, either an INKEY$ or a string of characters, therefore bit 6 of FLAGS must be reset. Unless syntax is being checked the parameters of the string are put on the calculator stack to form a 'last value'.

```
0FED  S-STRING      LD    HL,+FLAGS           Make HL point to FLAGS.
                    RES   6,(HL)              Reset this bit – string result.
                    BIT   7,(HL)              Test for line execution.
                    CALL  NZ,12C3,STK-STORE   Stack the parameters if executing a line.
                    RST   0020,NEXT-CH        Move to the next character.
0FF8  S-CONT-1      JP    1088,S-CONT-3       Jump forward.
```

A NEWLINE character will lead to an error being reported.

```
0FFB  S-N/L-ERR  CP    +76                  Is it a 'N/L' ?
                 JR    NZ,0FE0,S-Q-NEXT     Re-enter the loop if it is not the end of
                                            a line.
0FFF  S-RPRT-C1  JP    0D9A,REPORT-C        Jump back to give report C.
```

The present character must now represent a function.

```
1002  S-FUNCT    SUB   +C4                  The range of the functions is changed
                                            from C4-D7 to 00-13 Hex.
                 JR    C,0FFF,S-RPRT-C1     Report an error if out of range.
```

The function 'NOT' is identified and dealt with separately from the others.

```
                 LD    BC,+04EC             Priority 4, operation code EC.
                 CP    +13                  Is it the function 'NOT' ?
                 JR    Z,1020,S-PUSH-PO     Jump if it is so.
                 JR    NC,0FFF,S-RPRT-C1    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 reset in their operation codes.

```
                 LD    B,+10                Priority 16 decimal.
                 ADD   A,+D9                The function range is now D9-EB.
                 LD    C,A                  Transfer the operation code.
                 CP    +DC                  Separate CODE, VAL & LEN which
                 JR    NC,101A,S-NO-TO-$    operate on strings to give
                 RES   6,C                  numerical results.
101A  S-NO-TO-$  CP    +EA                  Separate STR$ & CHR$ which operate
                 JR    C,1020,S-PUSH-PO     on numbers to give string results.
                 RES   7,C                  Mark the operation codes.
                                            The other operation codes have
                                            bits 6 & 7 both set.
```

The priority code and the operation code for the function being considered are now pushed to the machine stack. A hierachy of operations is thereby built up.

```
1020  S-PUSH-PO  PUSH  BC                   Stack the priority and operation codes
                 RST   0020,NEXT-CH         before moving on to consider the
                 JP    0F59,S-RND           next part of the expression.
```

The present character has been identified as being alphanumeric. If it is a letter then a variable name has been found; however if it is a digit then a decimal number has been found.

```
1025  S-LET-NUM  CP    +26                  Jump if dealing with a digit.
                 JR    C,1047,S-DECIMAL
```

When a variable name has been identified a call is made to LOOK-VARS, which looks through those variables that already exist in the variable area. If an appropriate numeric value is found then it is copied to the calculator stack using MOVE-FP. However a string or string array entry has to have the appropriate parameters passed to the calculator stack by the STK-VAR subroutine.

```
                 CALL  111C,LOOK-VARS       Look in the existing variables for the
                                            matching entry.
                 JP    C,0D4B,REPORT-2      An error is reported if there is no
                                            existing entry.
                 CALL  Z,11A7,STK-VAR       Stack the parameters of the string
                                            entry/return numeric element base
                                            address.
                 LD    A,(FLAGS)            Fetch FLAGS.
```

```
                      CP    +C0                  Test bits 6 & 7 together.
                      JR    C,1087,S-CONT-2      One or both bits are reset.
                      INC   HL                   A numeric value is to be stacked.
                      LD    DE,(STKEND)          Fetch the 'old' STKEND.
                      CALL  19F6,MOVE-FP         Move the actual number.
                      EX    DE,HL                Move the pointer to HL.
                      LD    (STKEND),HL          Enter the 'new' STKEND.
                      JR    1087,S-CONT-2        Jump forward.
```

When a decimal number has been identified the action taken is 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'.

```
1047  S-DECIMAL  CALL  0DA6,SYNTAX-Z          Jump forward if a line is being
                 JR    NZ,106F,S-STK-DEC      executed.
```

During syntax checking:

```
                 CALL  14D9,DEC-TO-FP         The floating-point form is found.
                 RST   0018,GET-CH.           Set HL to point one – past the last digit.
                 LD    BC,+0006               Six locations are required.
                 CALL  099E,MAKE-ROOM         Make the room in the BASIC line.
                 INC   HL                     Point to the first 'new' location.
                 LD    (HL),+7E               Enter the number marker character.
                 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
                 LDIR                         the calculator stack to the line.
                 EX    DE,HL                  Put the line pointer in HL.
                 DEC   HL                     Point to the last byte added.
                 CALL  004C,CURSOR-S0         This sets CH-ADD.
                 JR    1083,S-NUMERIC         Jump forward.
```

During line execution:

```
106F  S-STK-DEC  RST   0020,NEXT-CH           Move on to the next character in
                 CP    +7E                    turn until the number marker
                 JR    NZ,106F,S-STK-DEC      character is found.
                 INC   HL                     Point to the first byte of the number.
                 LD    DE,(STKEND)            Fetch the 'old' STKEND.
                 CALL  19F6,MOVE-FP           Move the floating point number.
                 LD    (STKEND),DE            Save the 'new' STKEND.
                 LD    (CH-ADD),HL            This sets CH-ADD.
```

A numeric result has now been identified, coming from RND, PI or a decimal number, therefore bit 6 of FLAGS must be set.

```
1083  S-NUMERIC  SET   6,(FLAGS)              Set the numeric marker flag.
```

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, a NEWLINE character or a command.

```
1087  S-CONT-2   RST   0018,GET-CH.           Fetch the present character.
1088  S-CONT-3   CP    +10                    Jump forward if it is not a '(', which
                 JR    NZ,1098,S-OPERTR       indicates a parenthesised expression.
```

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
                   JR    NZ,10BC,S-LOOP    parenthesised expression.
                   CALL  1263,SLICING      Modify the parameters of the 'last value'.
                   RST   0020,NEXT-CH      Move on to consider the next character.
                   JR    1088,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.

```
1098  S-OPERTR     LD    BC,+00C3          Set default priority zero and the
                                           operation code offset to C3.
                   CP    +12               Compare the character against the lowest
                   JR    C,10BC,S-LOOP     operator. Jump if out of range.
                   SUB   +16               The ranges of the operators are changed
                                           from 12-18 & D8-DD to FC-FF, 00-02
                                           & C2-C7 Hex.
                   JR    NC,10A7,S-HIGH-OP Jump forward with 00-02 & C2-C7.
                   ADD   A,+0D             The original range 12-15 is now
                                           09-0C Hex.
                   JR    10B5,S-END-OP     Jump forward.
10A7  S-HIGH-OP    CP    +03               Leave the original range 16-18
                   JR    C,10B5,S-END-OP   as 00-02 Hex.
                   SUB   +C2               The original range C2-C7 is now
                                           00-05 Hex.
                   JR    C,10BC,S-LOOP     Again jump if out of range.
                   CP    +06               Test the upper limit.
                   JR    NC,10BC,S-LOOP    Again jump if out of range.
                   ADD   A,+03             The original range C2-C7 is now
                                           03-08 Hex.
10B5  S-END-OP     ADD   A,C               The offset C3 is added to give the range
                   LD    C,A               of operation codes C3-CF Hex.
                   LD    HL,+104C          The pointer to the priority table.
                                           i.e. 104C+C3=110F 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 marker on the machine stack below a hierachy, of unknown size, of function and binary operation codes. This hierachy 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 is taken off the machine stack and is 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 priorities are 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 hierachy of functions and binary operations that have been queued are dealt with in the correct order.

```
10BC  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,10ED,S-TIGHTER      Exit to wait for the argument.
                    AND   A                     Are both priorities zero?
                    JP    Z,0018, GET-CH.       Exit via GET-CH. thereby making
                                                'last value' the required result.
                    PUSH  BC                    Stack the 'present' values.
                    PUSH  DE                    Stack the 'last' values briefly.
                    CALL  0DA6,SYNTAX-Z         Do not perform the actual operation
                    JR    Z,10D5,S-SYNTEST      if syntax is being checked.
                    LD    A,E                   The 'last' operation code.
                    AND   +3F                   Strip off bits 6 & 7 to convert the
                                                operation code to a calculator-offset.
                    LD    B,A                   It is required in the B register.
                    RST   0028,FP-CALC.         Now use the calculator.
                    DEFB  +37,fp-calc-2,19E4    Perform the actual operation.
                    DEFB  +34,end-calc.,002B    It has been done.
                    JR    10DE,S-RUNTEST        Jump forward.
```

An important part of syntax checking involves the testing of the operations to ensure that the nature of the 'last value' is of the correct type for the operation under consideration.

```
10D5  S-SYNTEST  LD    A,E                  Get the 'last' operation code.
                 XOR   (FLAGS)              This tests the nature of the 'last value'
                 AND   +40                  against the requirement of the operation.
                                            They are to be the same for correct syntax.
10DB  S-RPRT-C2  JP    NZ,0D9A,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.

```
10DE  S-RUNTEST  POP   DE                   Get the 'last' operation code.
                 LD    HL,+FLAGS            Point to FLAGS.
                 SET   6,(HL)               Assume result to be numeric.
                 BIT   7,E                  Jump forward if the nature of 'last value'
                 JR    NZ,10EA,S-ENDLOOP    is numeric.
                 RES   6,(HL)               It is string.
10EA  S-ENDLOOP  POP   BC                   Get the 'present' values into BC.
                 JR    10BC,S-LOOP          Jump back.
```

Whenever the operations bind 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.

```
10ED  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 code if
                 JR    NZ,110A,S-NEXT       dealing with a numeric operand.
                 AND   +3F                  Clear bits 6 & 7.
                 ADD   A,+08                Increase the code by 08 Hex.
                 LD    C,A                  Return the code to the C register.
                 CP    +10                  Is the operation 'AND' ?
                 JR    NZ,1102,S-NOT-AND    Jump if it is not so.
                 SET   6,C                  'AND' requires a numeric operand.
                 JR    110A,S-NEXT          Jump forward.
1102  S-NOT-AND  JR    C,10DB,S-RPRT-C2     The operations -,*,/,** & OR are not
                                            possible.
                 CP    +17                  Is the operation '+' ?
                 JR    Z,110A,S-NEXT        Jump if it is so.
                 SET   7,C                  The other operations yield a numeric
                                            result.
110A  S-NEXT     PUSH  BC                   The 'present' values go on the stack.
```

```
                RST  0020,NEXT-CH          Move on to consider the next character
                                           in the expression.
                JP   0F59,S-RND            Start by testing against 'RND'.
```

THE PRIORITY TABLE

address	priority	operation	address	priority	operation
110F	06	-	1116	05	>=
1110	08	*	1117	05	<>
1111	08	/	1118	05	>
1112	0A	**	1119	05	<
1113	02	OR	111A	05	=
1114	03	AND	111B	06	+
1115	05	<=			

THE 'LOOK-VARS' SUBROUTINE

This subroutine is called whenever a search of the variable area is required. The subroutine is entered with CH-ADD pointing to the first letter of the variable name as it occurs in the BASIC line, either 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 name. Bits 5 & 6 of this byte indicate which type of variable is being handled.

The B register is used as a bit register to hold flags.

```
111C  LOOK-VARS SET  6,(FLAGS)          Presume a numeric variable.
                RST  0018,GET-CH.      Get the first character into A.
                CALL 14CE,ALPHA        Is it alphabetic?
                JP   NC,0D9A,REPORT-C  Give an error report if it is not so.
                PUSH HL                Save the pointer to the first letter.
                LD   C,A               Transfer the letter to C.
                RST  0020,NEXT-CH      Get the 2nd character into A.
                PUSH HL                Save the pointer to the 2nd character.
                RES  5,C               Start with bit 5 reset.
                CP   +10               Is the 2nd character a '(' ?
                JR   Z,1148,V-RUN/SYN  Separate arrays of numbers.
                SET  6,C               Now set bit 6.
                CP   +0D               Is the 2nd character a '$' ?
                JR   Z,1143,V-STR-VAR  Separate all the strings.
                SET  5,C               Now set bit 5.
```

Now find the end character of a variable name which has more than one character.

```
1139  V-CHAR    CALL 14D2,ALPHANUM     Is the character alphanumeric?
                JR   NC,1148,V-RUN/SYN Jump when the end is reached.
                RES  6,C               Mark the discriminator byte.
                RST  0020,NEXT-CH      Get the next character.
                JR   1139,V-CHAR       Go back to test it.
```

Simple strings and arrays of strings require that bit 6 of FLAGS is reset.

```
1143  V-STR-VAR RST  0020,NEXT-CH      Move CH-ADD on past the '$'.
                RES  6,(FLAGS)         Reset the bit 6 to indicate a string.
```

Now test the syntax flag.

```
1148  V-RUN/SYN LD   B,C               Copy the discriminator to B.
                CALL 0DA6,SYNTAX-Z     Test for syntax checking.
```

```
                    JR    NZ,1156,V-RUN          Jump forward if executing a line.
                    LD    A,C                    Move it to A for manipulation.
                    AND   +E0                    Drop the character code part.
                    SET   7,A                    Indicate syntax by setting bit 7.
                    LD    C,A                    Restore the discriminator to C.
                    JR    118A,V-SYNTAX          Jump forward.
```

A BASIC line is being executed so make a search of the variable area.

```
1156  V-RUN        LD    HL,(VARS)              Pick up the VARS pointer.
1159  V-EACH       LD    A,(HL)                 The 1st letter of each variable.
                   AND   +7F                    Match on bits 0-6.
                   JR    Z,1188,V-80-BYTE       Jump when the '80-byte' is reached.
                   CP    C                      The actual comparison.
                   JR    NZ,1180,V-NEXT         Jump if the 1st letter does not
                                                match the discriminator byte.
                   RLA                          Rotate A leftwards and then double
                   ADD   A,A                    it to test bits 5 & 6.
                   JP    P,1195,V-FOUND-2       Strings and array variables.
                   JR    C,1195,V-FOUND-2       Simple numeric and FOR-NEXT
                                                variables.
                   POP   DE                     Get the pointer to the 2nd character.
                   PUSH  DE                     Put it back.
                   PUSH  HL                     Save the variable pointer.
116B  V-MATCHES    INC   HL                     Go on to consider the next character.
116C  V-SPACES     LD    A,(DE)                 Fetch each character in turn.
                   INC   DE                     Point to the next.
                   AND   A                      Is the character a 'space' ?
                   JR    Z,116C,V-SPACES        Ignore the spaces.
                   CP    (HL)                   Make the comparison.
                   JR    Z,116B,V-MATCHES       Back for another if it does match.
                   OR    +80                    Will it match with bit 7 set?
                   CP    (HL)                   Try it.
                   JR    NZ,117F,V-GET-PTR      Jump if it does not match after all.
                   LD    A,(DE)                 Get the next character.
                   CALL  14D2,ALPHANUM          Is it alphanumeric?
                   JR    NC,1194,V-FOUND-1      Jump if the correct entry has been
                                                located in the variable area.
117F  V-GET-PTR    POP   HL                     Fetch the variable pointer.
1180  V-NEXT       PUSH  BC                     Save B & C briefly.
                   CALL  09F2,NEXT-ONE          DE will then point to the next
                                                variable in the variable area.
                   EX    DE,HL                  Transfer the pointer to HL.
                   POP   BC                     Get B & C back.
                   JR    1159,V-EACH            Round the loop again.
```

The variable name was not present in the variable area.

```
1188  V-80-BYTE    SET   7,B                    Indicates – no variable found.
```

The syntax path re-enters here.

```
1194  V-FOUND-1  POP   DE                  Drop the saved variable pointer.
1195  V-FOUND-2  POP   DE                  Drop the 2nd character pointer.
                 POP   DE                  Drop the first letter pointer.
                 PUSH  HL                  Save the 'last' letter pointer.
                 RST   0018,GET-CH.        Fetch the current character.
```

If the matching variable name has more than a single letter then the other characters must be passed-over.

```
1199  V-PASS     CALL  14D2,ALPHANUM       Is it alphanumeric?
                 JR    NC,11A1,V-END       Jump when the end of the name is
                                           reached, otherwise test again.
                 RST   0020,NEXT-CH        Fetch the next character.
                 JR    1199,V-PASS         Go back to test it.
```

The exit-parameters require to be set.

```
11A1  V-END      POP   HL                  HL holds the 'first' or the 'last' letter
                                           pointer.
                 RL    B                   Rotate the whole register.
                 BIT   6,B                 The zero flag is specified.
                 RET                       Finished.
```

The exit-parameters for the subroutine are:

The system variable CH-ADD points to the first character after the variable name as it occurs in the BASIC line.

If no matching variable name was found in the variable area then:

- i The carry flag is set.
- ii The zero flag is set when the search was for an array variable.
- iii The HL register pair points to the first letter of the variable name.

If the search yielded a matching entry in the variable area then:

- i The carry flag is reset.
- ii The zero flag is set for both simple string variables and all array variables.
- iii The HL register pair points to the letter of a single lettered variable name, or the last character of a long variable name, as it occurs in the variable area.

Bit 6 of the C register is reset when dealing with an array of numbers and set when dealing with an array of strings.

Bit 7 of the C register is reset during line execution and set during syntax checking.

THE 'STK-VAR' SUBROUTINE

This subroutine is usually used either to find the parameters that define an existing string entry in the variable area, or to return in the HL register pair the base address of a particular element of an array of numbers. When called from DIM the subroutine only checks the syntax of the BASIC line.

Note that the parameters that define a string may be altered by calling SLICING if this should be specified.

Initially the A and B registeres are cleared and bit 7 of the C register is tested to determine whether syntax is being checked.

```
11A7  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
                 JR    NZ,11F8,SV-COUNT    checked.
```

Next, simple strings are separated from array variables.

```
                        BIT   7,(HL)               Jump forward if dealing with an
                        JR    NZ,11BF,SV-ARRAYS    array variable.
```

The parameters for a simple string are readily found.

```
                        INC   A                    Specify a simple string.
11B2   SV-SMPLE$        INC   HL                   Move along the variable entry.
                        LD    C,(HL)               Pick up the low length counter.
                        INC   HL                   On one.
                        LD    B,(HL)               Pick up the high length counter.
                        INC   HL                   On one.
                        EX    DE,HL                Transfer the start pointer to DE.
                        CALL  12C3,STK-STORE       Pass these parameters to the stack.
                        RST   0018,GET-CH.         Fetch the present character.
                        JP    125A,SV-SLICE?       Jump to see if a 'slice' is required.
```

The base address of an element of an array is now found. Initially the 'number of dimensions' is collected.

```
11BF   SV-ARRAYS        INC   HL                   Go past the total length counter.
                        INC   HL
                        INC   HL
                        LD    B,(HL)               Collect the 'number of dimensions'.
                        BIT   6,C                  Jump forward if dealing with an
                        JR    Z,11D1,SV-PTR        array of numbers.
```

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 dimensions'.
                        JR    Z,11B2,SV-SMPLE$     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 variable pointer in DE.
                        RST   0018,GET-CH.         Get the present character.
                        CP    +10                  Is it a '(' ?
                        JR    NZ,1231,REPORT-3     Report the error if it is not so.
                        EX    DE,HL                Restore the variable pointer.
```

For both numeric arrays and arrays of strings the variable pointer is transferred to the DE register pair before the subscript is evaluated.

```
11D1   SV-PTR           EX    DE,HL                Variable pointer into DE.
                        JR    11F8,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 counter 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.

```
11D4   SV-COMMA         PUSH  HL                   Save the 'counter'.
                        RST   0018,GET-CH.         Get the present character.
                        POP   HL                   Restore the 'counter'.
                        CP    +1A                  Is the present character a ',' ?
                        JR    Z,11FB,SV-LOOP       Jump to consider another subscript.
                        BIT   7,C                  If a line is being executed then
                        JR    Z,1231,REPORT-3      there is an error.
```

```
                    BIT   6,C                   Jump if dealing with an
                    JR    NZ,11E9,SV-CLOSE      array of strings.
                    CP    +11                   Is the present character a ')' ?
                    JR    NZ,1223,SV-RPT-C      Report an error if it is not so.
                    RST   0020,NEXT-CH          Move CH-ADD to point to the next
                                                character in the BASIC line.
                    RET                         Return as the syntax is correct.
```

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.

```
11E9  SV-CLOSE      CP    +11                   Is the present character a ')' ?
                    JR    Z,1259,SV-DIM         Jump forward and check whether there
                                                is another subscript.
                    CP    +DF                   Is the present character a 'TO' ?
                    JR    NZ,1223,SV-RPT-C      It must not be otherwise.
11F1  SV-CH-ADD     RST   0018,GET-CH.          Get the present character.
                    DEC   HL                    Point to the preceding character.
                    LD    (CH-ADD),HL           Make CH-ADD point to this location.
                    JR    1256,SV-SLICE         Evaluate this 'slice'.
```

Enter the loop here.

```
11F8  SV-COUNT      LD    HL,+0000              Set the 'counter' to zero.
11FB  SV-LOOP       PUSH  HL                    Save the 'counter' briefly.
                    RST   0020,NEXT-CH          Makes CH-ADD point to the next
                                                character.
                    POP   HL                    Restore the 'counter'.
                    LD    A,C                   Fetch the discriminator byte.
                    CP    +C0                   Jump unless checking the syntax for
                    JR    NZ,120C,SV-MULT       an array of strings.
                    RST   0018,GET-CH.          Get the present character.
                    CP    +11                   Is it a ')' ?
                    JR    Z,1259,SV-DIM         Jump forward as finished counting
                                                elements.
                    CP    +DF                   Is it a 'TO' ?
                    JR    Z,11F1,SV-CH-ADD      Jump back if dealing with a 'slice'.
120C  SV-MULT       PUSH  BC                    Save the dimension-number counter
                                                and the discriminator byte.
                    PUSH  HL                    Save the element 'counter'.
                    CALL  12FF,DE,(DE+1)        Get a 'dimension-size' into DE.
                    EX    (SP),HL               The 'counter' moves to HL and the
                                                variable pointer is stacked.
                    EX    DE,HL                 The 'counter' moves to DE and the
                                                'dimension-size' to HL.
                    CALL  12DD,INT.-EXP1        Evaluate the next subscript.
                    JR    C,1231,REPORT-3       Give the error if out of range.
                    DEC   BC                    The result of the evaluation is
                                                decremented as the 'counter' is to
                                                count the elements occurring before
                                                the specified element.
                    CALL  1305,HL=HL*DE         Multiply 'counter' by 'dimension-size'.
                    ADD   HL,BC                 Add the result of 'INT.-EXP1'-1 to the
                                                'present counter.
                    POP   DE                    Fetch the variable pointer.
                    POP   BC                    Fetch the dimension-number counter
                                                and the discriminator byte.
                    DJNZ  11D4,SV-COMMA         Keep going round the loop until
                                                'B' equals zero.
```

The syntax flag is checked before arrays of strings are separated from numeric arrays.

```
                    BIT   7,C                   Syntax or line execution?
```

```
1223  SV-RPT-C   JR    NZ,128B,SL-RPT-C      Report the error if checking syntax.
                 PUSH  HL                    Save the 'counter'.
                 BIT   6,C                   Jump forward if dealing with
                 JR    NZ,123D,SV-ELEM$      an array of strings.
```

When dealing with a numeric array the present character must be a ')'.

```
                 LD    B,D                   Transfer the variable pointer to
                 LD    C,E                   the BC register pair.
                 RST   0018,GET-CH.          Fetch the present character.
                 CP    +11                   Is it a ')' ?
                 JR    Z,1233,SV-NUMBER      Report an error if it is not so.
```

Give report 3.

```
1231  REPORT-3   RST   0008,ERROR-1          Subscript out of range.
                 DEFB  +02
```

The address of the location before the actual floating-point number can now be calculated.

```
1233  SV-NUMBER  RST   0020,NEXT-CH          Move CH-ADD on one location.
                 POP   HL                    Fetch the 'counter'.
                 LD    DE,+0005              There are 5 bytes to each element.
                 CALL  1305,HL=HL*DE         Compute the total number of bytes.
                 ADD   HL,BC                 Add this number to the variable
                                             pointer, thereby HL will point to the
                                             location before the required element.
                 RET                         Finished with numeric arrays.
```

When dealing with an array of strings the length of an element is given by the last dimension-size. The appropriate parameters are calculated before being put on the calculator stack.

```
123D  SV-ELEM$   CALL  12FF,DE,(DE+1)        Fetch the last 'dimension-size'.
                 EX    (SP),HL               The variable pointer goes on the
                                             stack and the 'counter' to HL.
                 CALL  1305,HL=HL*DE         Multiply 'counter' by 'dimension-size'.
                 POP   BC                    Fetch the 'variable pointer'.
                 ADD   HL,BC                 This gives HL pointing to the location
                                             before the actual element.
                 INC   HL                    So point to the start of the string.
                 LD    B,D                   Transfer the last 'dimension-size'
                 LD    C,E                   to BC to form the length.
                 EX    DE,HL                 Transfer the start pointer to DE.
                 CALL  12C2,STK-ST-0         Pass the parameters to the
                                             calculator stack.
```

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 value indicating that the whole string is required.

```
                 RST   0018,GET-CH.          Get the present character.
                 CP    +11                   Is it a ')' ?
                 JR    Z,1259,SV-DIM         Jump if it is so.
                 CP    +1A                   Is it a ',' ?
                 JR    NZ,1231,REPORT-3      Report an error if it is not so.
1256  SV-SLICE   CALL  1263,SLICING          Use SLICING to modify the parameters.
1259  SV-DIM     RST   0020,NEXT-CH          Get the next character.
125A  SV-SLICE?  CP    +10                   Is it a '(' ?
                 JR    Z,1256,SV-SLICE       Jump back to evaluate the 'slice'.
                 RES   6,(FLAGS)             Indicate a string result.
                 RET                         Finished with arrays of strings.
```

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.

Initially the syntax flag is checked and the parameters of the string are fetched only if a line is being executed.

```
1263  SLICING    CALL 0DA6,SYNTAX-Z        Check the syntax flag.
                 CALL NZ,13F8,STK-FETCH    Collect the parameters if a line is
                                           being executed.
```

The possibility of the 'slice' being '()' has to be considered.

```
                 RST  0020,NEXT-CH         Get the next character.
                 CP   +11                  Is it a ')' ?
                 JR   Z,12BE,SL-STORE      Jump forward if it is so.
```

Before proceeding the registers are set up as required.

```
                 PUSH DE                   The 'start' goes on the stack.
                 XOR  A                    The A register is cleared and also
                 PUSH AF                   saved on the stack.
                 PUSH BC                   Save the 'length' briefly.
                 LD   DE,+0001             Assume that the 'slice' is to begin with
                                           the first character.
                 RST  0018,GET-CH.         Get the first character into A.
                 POP  HL                   Put the 'length' into HL.
```

The first parameter of the 'slice' is now evaluated.

```
                 CP   +DF                  Is the present character a 'TO' ?
                 JR   Z,1292,SL-SECOND     The first parameter by default will be
                                           the current value of DE, i.e. '1'.
                 POP  AF                   At this stage A will hold zero.
                 CALL 12DE,INT.-EXP2       BC will hold the first parameter and
                 PUSH AF                   A will hold Hex.FF if there has been an
                                           'out of range' error.
                 LD   D,B                  Transfer the first parameter to the
                 LD   E,C                  DE register pair.
                 PUSH HL                   Save the 'length' briefly.
                 RST  0018,GET-CH.         Get the present character.
                 POP  HL                   Restore the 'length'.
                 CP   +DF                  Is the present character a 'TO'?
                 JR   Z,1292,SL-SECOND     Jump forward to consider the
                                           second parameter.
                 CP   +11                  Is the present character a ')' ?
128B  SL-RPT-C   JP   NZ,0D9A,REPORT-C     There must be a closing bracket.
```

There is no second value to the 'slice' under consideration.

```
                 LD   H,D                  The last character of the 'slice' is
                 LD   L,E                  also the first character.
                 JR   12A5,SL-DEFINE       Jump forward.
```

The second parameter of the 'slice' is now evaluated.

```
1292  SL-SECOND  PUSH HL                   Save the 'length' briefly.
                 RST  0020,NEXT-CH         Get the next character.
                 POP  HL                   Restore the 'length'.
                 CP   +11                  Is the present character a ')' ?
```

```
                    JR    Z,12A5,SL-DEFINE      Jump if there is no second parameter.
                    POP   AF                    If the first parameter was in range
                                                A will hold zero, otherwise Hex. FF.
                    CALL  12DE,INT.-EXP2        BC will hold the second parameter.
                    PUSH  AF                    Save the error register again.
                    RST   0018,GET-CH.          Get the present character.
                    LD    H,B                   The value held in BC is the last
                    LD    L,C                   character of the 'slice'.
                    CP    +11                   Is the present character a ')' ?
                    JR    NZ,128B,SL-RPT-C      Report the error if it is not so.
```

The 'new' parameters are now defined.

```
12A5  SL-DEFINE     POP   AF                    Fetch the error register.
                    EX    (SP),HL               Second parameter goes on the stack
                                                and the start goes to HL.
                    ADD   HL,DE                 Add the first parameter to the start
                                                of the string.
                    DEC   HL                    Go back a location to get it correct.
                    EX    (SP),HL               The 'new start' goes on the stack and
                                                the second parameter to HL.
                    AND   A                     Prepare for subtraction.
                    SBC   HL,DE                 Finds the 'new length'.
                    LD    BC,+0000              By default the 'new length' is zero.
                    JR    C,12B9,SL-OVER        A 'negative slice' is a null string.
                    INC   HL                    Add the inclusive byte.
                    AND   A                     Now test the error register.
                    JP    M,1231,REPORT-3       Jump if there was an 'out of range'
                                                error whilst in INT.-EXP2.
                    LD    B,H                   Transfer the 'new length' to the
                    LD    C,L                   BC register pair.
12B9  SL-OVER       POP   DE                    Get the 'new start' from the stack.
                    RES   6,(FLAGS)             Ensure a string is indicated.
```

When a line is being executed this subroutine enters the STK-STORE subroutine directly so as to stack the parameters of the string.

```
12BE  SL-STORE      CALL  0DA6,SYNTAX-Z         Check the syntax flag and return if
                    RET   Z                     syntax is being checked.
```

THE 'STK-STORE'SUBROUTINE

This subroutine passes the values held in the A, B, C, D and E registers to the calculator stack. The stack thereby grows in size by 5 bytes.

Although this subroutine could be used to transfer floating-point numbers it is, however, only used to transfer the parameters of strings.

Note that the A register is used as a flag to show whether the string is a simple string or part of an array of strings. However this flag would appear to be redundant in the final program.

```
12C2  STK-ST-0      XOR   A                     Clear the array flag.
12C3  STK-STORE     PUSH  BC                    Save the BC register pair briefly.
                    CALL  19EB, TEST-5-SP       Is there room for the 5 bytes?
                    POP   BC                    Restore BC.
                    LD    HL,(STKEND)           Fetch the current value of STKEND.
                    LD    (HL),A                Pass the array flag.
                    INC   HL                    On one.
                    LD    (HL),E                Pass the low address pointer.
                    INC   HL                    On one.
                    LD    (HL),D                Pass the high address pointer.
```

```
                INC   HL                On one.
                LD    (HL),C            Pass the low length counter.
                INC   HL                On one.
                LD    (HL),B            Pass the high length counter.
                INC   HL                On one.
                LD    (STKEND),HL       Save the value in HL as STKEND.
                RES   6,(FLAGS)         Show that the 'last value' is a string.
                RET                     Finished.
```

THE 'INT.-EXP' SUBROUTINE

This subroutine returns to the calling routine the evaluation result of 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 Hex.00 if there has not been a previous error and Hex.FF if there was an error when the subroutine was last called.

```
12DD  INT.-EXP1  XOR   A                 Clear the error register.
12DE  INT.-EXP2  PUSH  DE                Save both DE and HL for the duration
                 PUSH  HL                of the subroutine.
                 PUSH  AF                Save the error register briefly.
                 CALL  0D92,CLASS-6      The 'next expression' is evaluated to give
                                         a 'last value'.
                 POP   AF                Restore the error register.
                 CALL  0DA6,SYNTAX-Z     Jump forward if syntax is being
                 JR    Z,12FC,I-RESTORE  checked.
                 PUSH  AF                Save the error register briefly.
                 CALL  0EA7,FIND-INT.    The 'last value' is compressed into the
                                         16 bits of the BC register pair.
                 POP   DE                Get the error register into D.
                 LD    A,B               Test the evaluation result.
                 OR    C
                 SCF                     Presume the error condition.
                 JR    Z,12F9,I-CARRY    Jump if evaluation result is zero.
                 POP   HL                Copy the 'limit value'. This will be
                 PUSH  HL                either 'dimension-size', 'DIM-limit'
                                         or 'string length'.
                 AND   A                 Prepare for the subtraction.
                 SBC   HL,BC             Make the test.
12F9  I-CARRY    LD    A,D               Fetch the error register.
                 SBC   A,+00             If there is no error and no previous
                                         error then A holds zero and carry
                                         is reset.
                                         Otherwise A holds Hex.FF or FE, and
                                         carry is set.
12FC  I-RESTORE  POP   HL                Restore the HL and DE
                 POP   DE                register pairs.
                 RET                     Finished.
```

THE 'DE,(DE+1)' SUBROUTINE

This subroutine performs the construction – LD DE,(DE+1) – and returns HL pointing to DE+2.

```
12FF  DE,(DE+1)  EX    DE,HL             Use HL for the construction.
                 INC   HL                Points to 'DE+1'.
                 LD    E,(HL)            In effect – LD E,(DE+1).
                 INC   HL                Points to 'DE+2'.
                 LD    D,(HL)            In effect – LD D,(DE+2).
                 RET                     Finished.
```

THE 'HL=HL* DE' SUBROUTINE

Unless syntax is being checked this subroutine performs the multiplication as stated.

Overflow of the 16 bits available gives 'REPORT 4'. This is not exactly the true situation but it implies that the machine is not large enough for the task envisaged by the programmer.

```
1305  HL=HL*DE   CALL  0DA6,SYNTAX-Z        Return if syntax is being
                 RET   Z                    checked.
                 PUSH  BC                   BC is saved.
                 LD    B,+10                It is to be a 16 bit multiplication.
                 LD    A,H                  A holds the high byte.
                 LD    C,L                  C holds the low byte.
                 LD    HL,+0000             Initialise the result to zero.
1311  HL-LOOP    ADD   HL,HL                Double the result.
                 JR    C,131A,HL-OVER       Jump if overflow.
                 RL    C                    Rotate bit 7 of C into the carry.
                 RLA                        Rotate the carry into bit 0 and
                                            bit 7 into the carry flag.
                 JR    NC,131D,HL-AGAIN     Jump if the carry flag is reset.
                 ADD   HL,DE                Otherwise add DE in once.
131A  HL-OVER    JP    C,0ED3,REPORT-4      Report the overflow error.
131D  HL-AGAIN   DJNZ  1311,HL-LOOP         Until 16 passes have been made.
                 POP   BC                   Restore BC.
                 RET                        Finished.
```

THE 'LET' COMMAND ROUTINE

This is the actual assignment routine for both the LET and the INPUT commands.

When the destination variable is a newly declared variable then DEST will point to the first letter of the variable name as it occurs in the current BASIC line. Bit 1 of FLAGX will be set.

However if the destination variable has been used previously 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 existing number; and for a string variable to the *first* location used by the existing string. The use of DEST in this manner applies to simple variables and to the elements of arrays.

Bit 0 of FLAGX is reset if the variable name indicates an array variable.

Initially the current value of DEST is collected and bit 1 of FLAGS tested.

```
1321  LET        LD    HL,(DEST)            Fetch the present value of DEST.
                 BIT   1,(FLAGX)            Jump if dealing with an existing
                 JR    Z,136E,L-EXISTS      variable.
```

A new variable is being used so the length of the name is found.

```
                 LD    BC,+0005             Assume a numeric variable.
132D  L-EACH-CH  INC   BC                   For each character of a name:
132E  L-NO-SP    INC   HL                   Move along the name.
                 LD    A,(HL)               Put the character in the A register.
                 AND   A                    Is the character a 'space'?
                 JR    Z,132E,L-NO-SP       Ignore any spaces in a name.
                 CALL  14D2,ALPHANUM        Is the character alphanumeric?
                 JR    C,132D,L-EACH-CH     Jump back for another if it is so.
                 CP    +0D                  Is the present character a '$'?
                 JP    Z,13C8,L-NEW$        Jump as dealing with a new string
                                            variable – a simple string.
```

The appropriate amount of room for the variable name and its value is made available in the work space. The characters of a long name, with the exception of the first letter, are transferred. The last letter is ORed with Hex.80.

```
                RST   0030,BC-SPACES     Make the appropriate amount of free
                                         space available in the work space.
                PUSH  DE                 DE points to the 2nd new space.
                LD    HL,(DEST)          Pointer to the start of the name.
                DEC   DE                 DE points to the 1st new space.
                LD    A,C                Get the size of the variable.
                SUB   +06                The minimum size is 6.
                LD    B,A                B equals the number of extra letters.
                LD    A,+40              Prepare to mark the first letter.
                JR    Z,1359,L-SINGLE    Jump forward if name is short.
1348 L-CHAR     INC   HL                 For each letter of a long name:
                LD    A,(HL)             Put the character in the A register.
                AND   A                  Again ignore any spaces.
                JR    Z,1348,L-CHAR      Jump back if it is a 'space'.
                INC   DE                 For each location in the work space.
                LD    (DE),A             Transfer the character of the name.
                DJNZ  1348,L-CHAR        Until the whole name is done.
                OR    +80                Prepare to mark the last letter.
                LD    (DE),A             Now mark it.
                LD    A,+80              Prepare to mark the first letter.
1359 L-SINGLE   LD    HL,(DEST)          Pointer to the start of the name.
                XOR   (HL)               Mark the first letter as is required.
                POP   HL                 Fetch the pointer to the 2nd free
                                         location.
```

The work space is now cleared up to the current entry and this entry is included in the variable area.

```
                CALL  13E7,L-CLEAR       Clear the work space from E-LINE to
                                         (HL) & include the new entry in the
                                         variable area by changing the pointers.
```

An RST 0028 instruction is used to 'delete' the 'last value' on the calculator stack. However this value is not overwritten.

```
1361 L-NUMERIC  PUSH  HL                 Save the pointer to the location
                                         after the 'value' of the variable.
                RST   0028,FP-CALC.      Now use the calculator.
                DEFB  +02,delete,19E3    This moves STKEND back five
                DEFB  +34,end-calc.,002B locations.
                POP   HL                 Restore the pointer.
```

The HL register pair is made to point to the first location of the 'value' of the variable.

```
                LD    BC,+0005           There are 5 locations.
                AND   A                  Prepare for subtraction.
                SBC   HL,BC              HL now points to the first location.
                JR    13AE,L-ENTER       Jump forward to enter the value.
```

Enter here if dealing with a variable name that has already been used. Bit 6 of FLAGS is tested to separate numeric variables from string, or array of string variables.

```
136E L-EXISTS   BIT   6,(FLAGS)          Jump forward if dealing with a
                JR    Z,137A,L-DELETE$   string variable.
```

The new numeric value overwrites the old value, but first the HL register pair must be set to point 'one location past' the old value.

```
                LD    DE,+0006          Six bytes for a numeric variable.
                ADD   HL,DE             HL now points one past.
                JR    1361,L-NUMERIC    Jump back to do the actual overwriting.
```

The parameters of the string variable are fetched and simple string variables separated from array of string variables.

```
137A L-DELETE$  LD    HL,(DEST)         Fetch the start pointer.
                LD    BC,(STRLEN)       Fetch the length counter.
                BIT   0,(FLAGX)         Jump if dealing with a
                JR    NZ,1387,L-ADD$    simple string.
```

The new string must not be a null string.

```
                LD    A,B               High length counter.
                OR    C                 Low length counter.
                RET   Z                 Return if the string is null.
```

The next stage involves making available an appropriate amount of room for the new string in the work space.

```
                PUSH  HL                Save the start pointer.
                RST   0030,BC-SPACES    Make room in the work space.
                PUSH  DE                Save the pointer to the 2nd space.
                PUSH  BC                Save the length.
                LD    D,H               HL holds the address of STKBOT - 1.
                LD    E,L
                INC   HL                HL now points to STKBOT.
                LD    (HL),+00          A space is entered.
                LDDR                    All of the new locations are now set to
                                        zero (except the 1st space).
```

The pointer to this 'new' area in the workspace is saved whilst the parameters of the 'new' string are fetched from the calculator stack.

```
                PUSH  HL                Save the 'new' area pointer.
                CALL  13F8,STK-FETCH    Fetch the parameters.
                POP   HL                Restore the pointer.
```

The length of the string is now compared to the amount of room that has been made available for it.

```
                EX    (SP),HL           'Length' of new area to HL.
                                        'Pointer' to new area to stack.
                AND   A                 Prepare for subtraction.
                SBC   HL,BC             Find the difference in the lengths.
                ADD   HL,BC             Add it back.
                JR    NC,13A3,L-LENGTH  Jump if the 'new' string will fit.
                LD    B,H               The procrustean shortening of a
                LD    C,L               string that is too long.
13A3 L-LENGTH   EX    (SP),HL           'Length' of new area to stack.
                                        'Pointer' to new area to HL.
```

As long as the new string is not a null string it is copied into the workspace. Procrustean lengthening is achieved by only moving the number of characters specified in the BC register pair.

```
                EX    DE,HL             'Start' of string to HL.
                                        'Pointer' to new area to DE.
                LD    A,B               Test the 'length' of
                OR    C                 the new string.
                JR    Z,13AB,L-IN-W-S   Jump forward if a null string.
                LDIR                    The string is copied to the area reserved
                                        for it in the work space.
```

```
13AB  L-IN-W-S   POP   BC                 'Length' of new area.
                 POP   DE                 Pointer to the 2nd space.
                 POP   HL                 The pointer to the start of the element
                                          in the array.
```

The string is now copied from the work space to its specified place in the variable area.
Note: Also used to transfer numeric values.

```
13AE  L-ENTER    EX    DE,HL              Change pointers over.
                 LD    A,B                There is no need to move a string
                 OR    C                  or number that has 'no length'
                 RET   Z                  attributed to it.
                 PUSH  DE                 Save the address of the element.
                 LDIR                     Move the string or number.
                 POP   HL                 The address of the element is in HL.
                 RET                      Finished with array variables.
```

When a new string is to replace an old string the new string is entered as if it were a totally new variable before the old copy of that variable is reclaimed.

```
13B7  L-ADD$     DEC   HL                 HL is made to point to the
                 DEC   HL                 variable name of the old copy of the
                 DEC   HL                 string in the variable area.
                 LD    A,(HL)             The name goes into the A register.
                 PUSH  HL                 The pointer to the name is saved.
                 PUSH  BC                 The length of the old copy is saved.
```

The new string is copied into the work space and included in the variable area by calling L-STRING before the old copy is reclaimed.

```
                 CALL  13CE,L-STRING      Add the new string to the variables.
                 POP   BC                 The length of the old copy.
                 POP   HL                 The starting address of the old copy.
                 INC   BC                 The total length of a string
                 INC   BC                 variable is given by adding three
                 INC   BC                 to the number of characters.
                 JP    0A60,RECLAIM-2     Exit by jumping to RECLAIM-2 which
                                          reclaims BC bytes starting at (HL).
```

A totally new string variable is added to the variable area as follows:

The variable's name is collected from the BASIC line and marked as representing a simple string.

```
13C8  L-NEW$     LD    A,+60              Prepare for the marking of the name.
                 LD    HL,(DEST)          Fetch the address of the name.
                 XOR   (HL)               Mark the name.
```

The parameters of the string are fetched and the appropriate amount of room is made for the string in the work space.

```
13CE  L-STRING   PUSH  AF                 Save the name.
                 CALL  13F8,STK-FETCH     Fetch the parameters of the string.
                 EX    DE,HL              Switch over the pointers.
                 ADD   HL,BC              Find the end of the new string + 1.
                 PUSH  HL                 Save the pointer to the 'end + 1'.
                 INC   BC                 Add three to the number of
                 INC   BC                 characters to make the full length
                 INC   BC                 that is required.
                 RST   0030,BC-SPACES     Make the room in the work space.
                 EX    DE,HL              End of the work space in DE.
                 POP   HL                 Restore 'end + 1' of string.
```

The new string can now be copied into the room prepared for it in the work space. The 'length' is calculated and added to the variable.

```
                DEC   BC              Move the new string and
                DEC   BC              one extra byte.
                PUSH  BC              Save the count of the bytes.
                LDDR                  Copy the string to the work space.
                EX    DE,HL           The location before the string to HL.
                POP   BC              Restore the count.
                DEC   BC              The length of the new string.
                LD    (HL),B          Enter high-length.
                DEC   HL              Back one.
                LD    (HL),C          Enter low-length.
                POP   AF              Restore the variable name.
```

All of the work space before the location pointed to by the HL register pair is reclaimed and the variable name is entered into the '80 byte'.

```
13E7  L-CLEAR   PUSH  AF              Save the variable name briefly.
                CALL  14C7,RECLAIM-3  Reclaim the work space up to (HL).
                POP   AF              Restore the variable name.
                DEC   HL              Now point to the '80 byte'.
                LD    (HL),A          Overwrite with the variable name.
```

The system variable E-LINE is set to equal STKBOT and hence clears the work space and an '80' is entered into the extra location at the end of the new string.

```
                LD    HL,(STKBOT)     Get the pointer STKBOT.
                LD    (E-LINE),HL     Make E-LINE equal STKBOT.
                DEC   HL              The extra byte after the new string.
                LD    (HL),+80        Make the new '80 byte'.
                RET                   Finished adding a new string.
```

THE 'STK-FETCH' SUBROUTINE

This subroutine collects either a five byte floating-point number, or a set of parameters that define a string, from the calculator stack. These five bytes represent the current 'last value'.

```
13F8  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     The new value for STKEND.
                RET                   Finished.
```

THE 'DIM' COMMAND ROUTINE

The routine starts with a search of the variable area to ascertain if a variable with the same variable name already exists. If such a variable is found then it is deleted by reclaiming the bytes involved.

The size of the new array is calculated and the appropriate amount of room is made available in the variable area. The parameters of the variable are entered and all of the elements are set to zero.

```
1409  DIM        CALL  111C,LOOK-VARS      Look for an existing variable.
140C  D-RPORT-C  JP    NZ,0D9A,REPORT-C    Give report C as there is an error.
                 CALL  0DA6,SYNTAX-Z       Jump forward to D-RUN unless
                 JR    NZ,141C,D-RUN       syntax is being checked.
                 RES   6,C                 Presume a numeric array.
                 CALL  11A7,STK-VAR        Check the syntax further.
                 CALL  0D1D,CHECK-END      Exit via CHECK-END.
141C  D-RUN      JR    C,1426,D-LETTER     Jump if no existing variable.
                 PUSH  BC                  Save the variable name.
                 CALL  09F2,NEXT-ONE       Find the start of the next variable.
                 CALL  0A60,RECLAIM-2      Reclaim the bytes of the existing variable.
                 POP   BC                  Restore the variable name.
```

The initial parameters of the variable are set.

```
1426  D-LETTER   SET   7,C                 An array variable name has bit 7 set.
                 LD    B,+00               Make the dimension counter zero.
                 PUSH  BC                  Save the counter and the name.
                 LD    HL,+0001            Element length for an array of strings.
                 BIT   6,C                 Jump if dealing with an
                 JR    NZ,1434,D-SIZE      array of strings.
                 LD    L,+05               Element length for a numerical array.
1434  D-SIZE     EX    DE,HL               Element length is to be in DE.
```

The following loop is passaged for each dimension that is specified in the BASIC line. The total number of bytes required for the elements of the array is built up in the DE register pair.

```
1435  D-NO-LOOP  RST   0020,NEXT-CH        Move CH-ADD on one byte.
                 LD    H,+40               Set a 'limit-value'.
                 CALL  12DD,INT.-EXP1      Evaluate the parameter.
                 JP    C,1231,REPORT-3     Give an error if out of range.
                 POP   HL                  Restore the counter and the name.
                 PUSH  BC                  Stack the result of INT.-EXP1.
                 INC   H                   Increase the dimension count.
                 PUSH  HL                  Save the counter and the name.
                 LD    H,B                 Result of INT.-EXP1 is
                 LD    L,C                 required in HL.
                 CALL  1305,HL=HL*DE       Check that enough RAM is available
                 EX    DE,HL               and transfer the byte total to DE.
                 RST   0018,GET-CH.        Get the present character.
                 CP    +1A                 Is it a ',' ?
                 JR    Z,1435,D-NO-LOOP    Jump back if there is another dimension
                                           to be included.
```

The final values of the parameters are calculated.

```
                 CP    +11                 Is it a ')' ?
                 JR    NZ,140C,D-RPORT-C   Jump if there has been an error.
                 RST   0020,NEXT-CH        Move CH-ADD on one byte.
                 POP   BC                  Restore the counter and the name.
                 LD    A,C                 Move the variable name to A.
                 LD    L,B                 Move the dimension counter to L.
                 LD    H,+00               Clear the H register.
                 INC   HL                  Increase the dimension count by
                 INC   HL                  two and then double the result to
                 ADD   HL,HL               obtain the number of bytes required
                                           for the parameters.
                 ADD   HL,DE               Add to this the number of bytes
                                           required for the elements.
                 JP    C,0ED3,REPORT-4     'Out of RAM' if result too great.
                 PUSH  DE                  Save the element-byte total.
```

```
                PUSH  BC                    Save the counter and the name.
                PUSH  HL                    Save the 'total'.
                LD    B,H                   Move the 'total' to the
                LD    C,L                   BC register pair.
```

The appropriate amount of room is now made in the variable area.

```
                LD    HL,(E-LINE)           Fetch E-LINE.
                DEC   HL                    Point to the '80 byte'.
                CALL  099E,MAKE-ROOM        Make BC spaces before the '80 byte'.
                INC   HL                    Make HL point to the first space.
```

The parameters are now entered.

```
                LD    (HL),A                Enter the variable name.
                POP   BC                    Fetch the 'total' and
                DEC   BC                    decrease it by three to give
                DEC   BC                    the required number.
                DEC   BC
                INC   HL                    Now point to the second location.
                LD    (HL),C                Enter the low-total.
                INC   HL                    Point to the third location.
                LD    (HL),B                Enter the high-total.
                POP   AF                    Fetch the 'dimension counter'.
                INC   HL                    Point to the fourth location.
                LD    (HL),A                Enter the counter.
```

The elements of the array are all set to zero.

```
                LD    H,D                   HL is made to point to the last
                LD    L,E                   byte.
                DEC   DE                    DE now points to the last but one.
                LD    (HL),+00              Enter a 'zero'.
                POP   BC                    Fetch the element-byte total.
                LDDR                        Enter a 'zero' into all the other bytes
                                            and finish with HL pointing to the
                                            byte before the first element.
```

The 'dimension-sizes' are now entered.

```
147F  DIM-SIZES POP   BC                    Get the last dimension-size.
                LD    (HL),B                Enter the high byte.
                DEC   HL                    Go back one location.
                LD    (HL),C                Enter the low byte.
                DEC   HL                    Go back another location.
                DEC   A                     Decrease the dimension counter.
                JR    NZ,147F,DIM-SIZES     Repeat the operation until the
                                            counter reaches zero.
                RET                         Finished.
```

THE 'RESERVE' SUBROUTINE

This subroutine is a continuation of RST 0030,BC-SPACES, and is used to increase the size of the work space by the number of bytes specified.

```
1488  RESERVE   LD    HL,(STKBOT)           Fetch the current value of STKBOT.
                DEC   HL                    Make HL point to the last location of
                                            the current work space.
                CALL  099E,MAKE-ROOM        Create BC spaces in the work space
                                            before the last location.
```

```
          INC   HL                HL points to the 1st new space.
          INC   HL                HL points to the 2nd new space.
          POP   BC                Fetch the old value of E-LINE
          LD    (E-LINE),BC       and restore it unaltered.
          POP   BC                Restore BC, the number of new spaces.
          EX    DE,HL             Now DE points to the 2nd new space.
          INC   HL                Make HL point to the last location
                                  of the work space once again.
          RET                     Finished.
```

THE 'CLEAR' COMMAND ROUTINE

This routine 'clears' the variable area.

```
149A  CLEAR      LD    HL,(VARS)      Fetch the current value of VARS.
                 LD    (HL),+80       Make this byte the '80 byte'.
                 INC   HL             Point to the next location.
                 LD    (E-LINE),HL    Make E-LINE point to this location.
```

THE 'X-TEMP' SUBROUTINE

This subroutine 'clears' the work space.

```
14A3  X-TEMP     LD    HL,(E-LINE)    Fetch the current value of E-LINE.
```

THE 'SET-STK-B' SUBROUTINE

This subroutine 'places' an 'empty' calculator stack at the position pointed to by the HL register pair.

```
14A6  SET-STK-B  LD    (STKBOT),HL    Set the bottom of the stack.
14A9  SET-STK-E  LD    (STKEND),HL    Set the top of the stack.
                 RET                  Finished.
```

THE 'CURSOR-IN' SUBROUTINE

This subroutine sets the workspace to hold a line consisting of only the cursor marker and the NEWLINE characters. The lower screen is set to be two lines in size and the calculator stack is cleared.

```
14AD  CURSOR-IN  LD    HL,(E-LINE)    Fetch the current value of E-LINE.
                 LD    (HL),+7F       Enter the cursor marker.
                 INC   HL             Move to the next location.
                 LD    (HL),+76       Enter the NEWLINE character.
                 INC   HL             Make HL point to the next location.
                 LD    (DF-SZ),+02    Lower screen is to be two lines.
                 JR    14A6,SET-STK-B Jump back to clear the calculator stack.
```

THE 'SET-MEM' SUBROUTINE

This subroutine makes MEM point to MEMBOT and returns STKEND pointing to the top of the calculator stack.

```
14BC  SET-MEM    LD    HL,+MEMBOT     Make HL point to MEMBOT.
                 LD    (MEM),HL       Make MEM point to MEMBOT.
                 LD    HL,(STKBOT)    Make HL point to the bottom of the
                                      calculator stack.
                 JR    14A9,SET-STK-E Jump back to make STKEND once
                                      again refer to the calculator stack.
```

THE 'RECLAIM-3' SUBROUTINE

This subroutine 'clears' the work space from its start to the location before that pointed to by the HL register pair.

```
14C7  RECLAIM-3 LD   DE,(E-LINE)        Fetch the current value of E-LINE.
                JP   0A5D,RECLAIM-1     Jump back to perform the clearance.
```

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.

```
14CE  ALPHA     CP   +26                Test against Hex. 26. The code for 'A'.
                JR   14D4,ALPHA-2       Jump forward.
```

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.

```
14D2  ALPHANUM  CP   +1C                Test against Hex. 1C. The code for '0'.
14D4  ALPHA-2   CCF                     Complement the carry flag.
                RET  NC                 Return if not a valid character code.
                CP   +40                Test against the upper limit.
                RET                     Finished.
```

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.

Firstly any integer part is converted.

```
14D9  DEC-TO-FP CALL 1548,INT-TO-FP     Forms a 'last value' of the integer.
```

If the next character is a '.', then consider the decimal fraction.

```
                CP   +1B                Is the character a '.' ?
                JR   NZ,14F5,E-FORMAT   Jump forward to see if it is an 'E'.
                RST  0028,FP-CALC.      Now use the calculator.
                DEFB +A1,stk-one,1A51   Find the floating-point form of the
                DEFB +C0,st-mem-0,1A63  decimal number '1', and save it in
                DEFB +02,delete,19E3    the memory area.
                DEFB +34,end-calc.,002B
14E5  NXT-DGT-1 RST  0020,NEXT-CH       Get the next character.
                CALL 1514,STK-DIGIT     If it is a digit then stack it.
                JR   C,14F5,E-FORMAT    If not jump forward.
                RST  0028,FP-CALC.      Now use the calculator.
                DEFB +E0,get-mem-0,1A45 For each passage of the loop, the
                DEFB +A4,stk-ten,1A51   number saved in the memory area is
                DEFB +05,division,1882  fetched, divided by 10 and restored.
                DEFB +C0,st-mem-0,1A63  i.e. going from .1 to .01 to .001 etc.
                DEFB +04,multiply,17C6  The present digit is multiplied by
                DEFB +0F,addition,1755  the 'saved number' and added to the
                DEFB +34,end-calc.,002B 'last value'.
                JR   14E5,NXT-DGT-1     Jump back to consider the next
                                        character.
```

Next consider any 'E-notation', i.e. the form xEm where m is a positive or negative integer.

```
14F5  E-FORMAT  CP    +2A                  Is the present character an 'E' ?
                RET   NZ                   Finished unless it is so.
                LD    (MEMBOT),+FF         Use the first byte of 'mem-O' as a
                                           sign-flag.
                RST   0020,NEXT-CH         Get the next character.
                CP    +15                  Is it a '+' ?
                JR    Z,1508,SIGN-DONE     Jump forward.
                CP    +16                  Is it a '-' ?
                JR    NZ,1509,ST-E-PART    Jump if neither '+' nor '-'.
                INC   (MEMBOT)             Change the sign of the flag.
1508  SIGN-DONE RST   0020,NEXT-CH         Point to the first digit.
1509  ST-E-PART CALL  1548,INT-TO-FP       Use this subroutine to stack the whole
                                           of the exponent, i.e. ABS m.
                RST   0028,FP-CALC.        Now use the calculator.
                DEFB  +E0,get-mem-0,1A45   Fetch the sign-flag.
                DEFB  +00,jump-true,1C2F   Jump if the sign-flag denotes '+'.
                DEFB  +02, to 1511,E-FP
                DEFB  +18,negate,1AA0      Negate the value of the exponent.
1511  E-FP      DEFB  +38,e-to-fp,155A     The 'last value' is given the result
                                           of x*10**m.
                DEFB  +34,end-calc.,002B
                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.

```
1514  STK-DIGIT CP    +1C                  Is the value Hex.1C ?
                RET   C                    Return if not in range.
                CP    +26                  Is the value Hex.26 ?
                CCF                        Complement the carry flag.
                RET   C                    Return if not in range.
                SUB   +1C                  Replace 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.

```
151D  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.

```
1520  STACK-BC  LD    IY,+ERR-NR           Re-initialise the IY register pair.
                PUSH  BC                   Save BC briefly.
                RST   0028,FP-CALC.        Use the calculator.
                DEFB  +A0,stk-zero,1A51    Put zero on the stack so as to
                DEFB  +34,end-calc.,002B   reserve 5 bytes. (Last value = 0)
                POP   BC                   Restore BC.
                LD    (HL),+91             Set exponent to 17 decimal for a
                LD    A,B                  16-bit number, and then test
                AND   A                    whether B is in fact zero.
```

```
                  JR    NZ,1536,NORML-FP     Jump forward when B is non-zero.
                  LD    (HL),A               Else, zero to exponent byte.
                  OR    C                    Return if C is also zero as the
                  RET   Z                    'last value' is to be zero.
                  LD    B,C                  Transfer C to B.
                  LD    C,(HL)               Clear the C register.
                  LD    (HL),+89             Set exponent to 9 decimal for an
                                             8-bit number.
1536  NORML-FP    DEC   (HL)                 Normalize the floating-point form by
                  SLA   C                    shifting C & B left until a set
                  RL    B                    bit is found. The exponent is
                  JR    NC,1536,NORML-FP     decremented on each loop.
                  SRL   B                    Now shift B & C right, resetting the
                  RR    C                    set bit for a positive number.
                  INC   HL                   Point to the 2nd byte.
                  LD    (HL),B               Copy over the B register.
                  INC   HL                   Point to the 3rd byte.
                  LD    (HL),C               Copy over the C register.
                  DEC   HL                   Return with the HL register
                  DEC   HL                   pair pointing to the exponent.
                  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 a decimal number or the line number, to its floating-point form.

Repeated calls to NEXT-CH fetch each digit of the integer in turn. An exit is made when a character that is not a digit has been fetched.

```
1548  INT-TO-FP   PUSH  AF                     Save the first digit – in A.
                  RST   0028,FP-CALC.          Use the calculator.
                  DEFB  +A0,stk-zero,1A51      The 'last value' is now zero.
                  DEFB  +34,end-calc.,002B
                  POP   AF                     Restore the first digit.
```

Now a loop is set up. As long as the character is a digit then its 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.

```
154D  NXT-DGT-2   CALL  1514,STK-DIGIT         If the character is a digit then
                  RET   C                      stack its floating-point form.
                  RST   0028,FP-CALC.          Use the calculator.
                  DEFB  +01,exchange,1A72      'Digit' goes under 'last value'.
                  DEFB  +A4,stk-ten,1A51       Define decimal 10.
                  DEFB  +04,multiply,17C6      'Last value' = 'last value' * 10.
                  DEFB  +0F,addition,1755      'Last value' = 'last value' + 'digit'.
                  DEFB  +34,end-calc.,002B
                  RST   0020,NEXT-CH           Next character goes into A.
                  JR    154D,NXT-DGT-2         Loop back with this character.
```

THE 'E-FORMAT TO FLOATING-POINT' SUBROUTINE
(Offset 38 – 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 m at the top of the calculator stack and x underneath m.

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 reduced by 7 for as long as possible and then by 1 until it is exhausted. Since p is usually less than decimal 38, no more than 8 loops are commonly taken.

i Once again the first byte of mem-0 is used as a sign flag. It shows whether multiplication or division by 10**p is required.

```
                                                 Calculator Stack
155A  e-to-fp     RST   0028,FP-CALC.            x, m
                  DEFB  +2D,duplicate,19F6       x, m, m
                  DEFB  +32,less-0,1ADB          x, m, (1/0)     Logical value of m.
                  DEFB  +C0,st-mem-0,1A63        x, m, (1/0)     Store sign flag in
                  DEFB  +02,delete,19E3          x, m            first byte of mem-0.
                  DEFB  +27,abs,1AAA             x, p            p = ABS m.
```

ii Now the main loop is entered. It starts by testing p to see whether it is exhausted.

```
1560  E-YET       DEFB  +A1,stk-one,1A51         x, p, 1
                  DEFB  +03,subtract,174C        x, p-1
                  DEFB  +2D,duplicate,19F6       x, p-1, p-1
                  DEFB  +32,less-0,1ADB          x, p-1, (1/0)
                  DEFB  +00,jump-true,1C2F       x, p-1
                  DEFB  +22, to 1587,END-E       x, p-1
```

iii Next p is reduced by 7 if possible, by 1 otherwise; and 10**7 or 10**1 is put on the calculator stack preparatory to multiplying or dividing.

```
                  DEFB  +2D,duplicate,19F6       x, p-1, p-1
                  DEFB  +30,stk-data,19FC        x, p-1, p-1, 6
                  DEFB  +33,exponent 83
                  DEFB  +40,(+00,+00,+00)
                  DEFB  +03,subtract,174C        x, p-1, p-7
                  DEFB  +2D,duplicate,19F6       x, p-1, p-7, p-7
                  DEFB  +32,less-0,1ADB          x, p-1, p-7, (1/0)
                  DEFB  +00,jump-true,1C2F       x, p-1, p-7
                  DEFB  +0C, to 157A,E-ONE       x, p-1, p-7
                  DEFB  +01,exchange,1A72        x, p-7, p-1
                  DEFB  +02,delete,19E3          x, p-7
                  DEFB  +01,exchange,1A72        p-7, x
                  DEFB  +30,stk-data,19FC        p-7, x, 10**7
                  DEFB  +80,four bytes
                  DEFB  +48,exponent 98
                  DEFB  +18,+96,+80,(+00)
                  DEFB  +2F,jump,1C23            p-7, x, 10**7
                  DEFB  +04, to 157D,E-M/D       p-7, x, 10**7
157A  E-ONE       DEFB  +02,delete,19E3          x, p-1
                  DEFB  +01,exchange,1A72        p-1, x
                  DEFB  +A4,stk-ten,1A51         p-1, x, 10
```

iv The sign-flag is collected and tested thereby showing whether to multiply or divide by 10**i, where i=1 or 7.After the arithmetic operation a jump is made back to E-YET.

```
157D  E-M/D       DEFB  +E0,get-mem-0,1A45       p-i, x, 10**i, (1/0)
                  DEFB  +00,jump-true,1C2F       p-i, x, 10**i
                  DEFB  +04, to 1583,E-DIV       p-i, x, 10**i
                  DEFB  +04,multiply,17C6        p-i, x*10**i
                  DEFB  +2F,jump,1C23            p-i, x*10**i
                  DEFB  +02, to 1584,E-EXC       p-i, x*10**i
1583  E-DIV       DEFB  +05,division,1882        p-i, x*10**-i
1584  E-EXC       DEFB  +01,exchange,1A72        x*10** +/-i, p-i
                  DEFB  +2F,jump,1C23            x*10** +/-i, p-i
                  DEFB  +DA, to 1560,E-YET       x*10** +/-i, p-i
```

v An exit is made from the subroutine with the required 'last value'.

```
1587  END-E      DEFB +02,delete,19E3        x*10**m
                 DEFB +34,end-calc.,002B
                 RET
```

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 65535 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.

```
158A  FP-TO-BC   CALL  13F8,STK-FETCH        Get the 'last value'.
                 AND   A                     Is the exponent zero?
                 JR    NZ,1595,NOT-ZERO      Jump if it is not so.
                 LD    B,A                   Set B to hold zero.
                 LD    C,A                   Set C to hold zero.
                 PUSH  AF                    Save the carry and the zero flag.
                 JR    15C6,FBC-END          Jump forward.
```

Once the special case of zero has been excluded, the upper limit is considered by comparing the value of the exponent against Hex.91.

```
1595  NOT-ZERO   LD    B,E                   1st byte of mantissa to B.
                 LD    E,C                   3rd byte of mantissa to E.
                 LD    C,D                   2nd byte of mantissa to C.
                 SUB   +91                   Reduce the exponent by 145 decimal.
                 CCF                         Complement the carry flag.
                 BIT   7,B                   The zero flag complements the sign bit,
                                             i.e. NZ for -ve numbers.
                 PUSH  AF                    Save the zero and the carry flags.
                 SET   7,B                   Restore the true numeric bit.
                 JR    C,15C6,FBC-END        Jump to the end if the exponent is
                                             too great.
```

Note that the exponent byte e holds 128 decimal plus the true exponent, e'.

So far the cases of the exponent byte being zero, or greater than 144 decimal, have been dealt with. The exponent byte is currently in the A register and now has the range -144 to -1 decimal which corresponds to the true exponent e' range of -127 to 16 decimal.

Numbers whose true exponent is in the range 1 to 8 decimal, will compress into a single register, whereas an exponent in the range 9 to 16 requires two registers. Numbers whose true exponents are negative will vanish.

```
                 INC   A                     Range is now -143 to 0 decimal.
                 NEG                         Range is now 143 to 0 decimal.
                 CP    +08                   Define the true exponents 9 to 16
                 JR    C,15AF,SHIFT-TST      and jump forward with them.
                 LD    E,C                   Move 2nd byte of mantissa to E.
                 LD    C,B                   Move 1st byte of mantissa to C.
                 LD    B,+00                 Clear the B register.
                 SUB   +08                   Range is now 135 to 0 decimal here.
```

Note that if the A register now holds zero it means that no shift of BC is needed (e' is 8 or 16 dec.). Otherwise the A register gives the length of the shift right needed. If the shift is to be greater than 8 places then the number will vanish (for true exponents -127 to -1).

```
15AF  SHIFT-TST  AND   A                    If zero then no shift is needed.
                 LD    D,A                  Transfer shift counter to D.
                 LD    A,E                  Prepare 9th/17th bit for
                 RLCA                       rounding up.
                 JR    Z,15BC,IN-PLACE      Jump if A was zero; no shift.
15B5  SHIFT-BC   SRL   B                    Shift B & C right D times to
                 RR    C                    produce the correct number.
                 DEC   D                    Decrement the shift counter.
                 JR    NZ,15B5,SHIFT-BC     Loop until D becomes zero.
15BC  IN-PLACE   JR    NC,15C6,FBC-END      End if no rounding-up needed;
                 INC   BC                   else round up.
                 LD    A,B                  Test if number now equals 65536 dec.
                 OR    C                    i.e. BC now zero – out of range.
                 JR    NZ,15C6,FBC-END      Jump if in range.
                 POP   AF                   Fetch zero and carry flags.
                 SCF                        Set carry flag as out of range.
                 PUSH  AF                   Save the zero and carry flags.
15C6  FBC-END    PUSH  BC                   Save the result briefly.
                 RST   0028,FP-CALC.        Use the calculator.
                 DEFB  +34,end-calc.,002B   This makes HL point to STKEND - 5.
                 POP   BC                   Restore the result.
                 POP   AF                   Restore the zero and the carry flags.
                 LD    A,C                  Copy over the low byte of the result.
                 RET                        Finished.
```

THE 'FLOATING-POINT TO A' SUBROUTINE

This short but vital subroutine is called at least 5 times for various purposes. It uses the previous 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.

```
15CD  FP-TO-A    CALL  158A,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
                 INC   B                    if the B register does not hold zero.
                 JR    Z,15D9,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.
15D9  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 0B61 and by STR$ at 1BD5, which converts to a string the number as it would be printed. The subroutine prints X, the 'last value' on the calculator stack. The print format never occupies more than 14 spaces. The subroutine first calculates:

n = INT ((e' – .5)*$\log_{10}2$) , where e' is the true exponent.

The number of digits before the decimal point of X is always n, n+1 or n+2.

Next the subroutine calculates:

m = INT (10** (8-n)* ABS X+.5) , the decimal representation of which is stored in an *ad hoc* print buffer in mem-2 to mem-4.

The 8 most significant digits of X, correctly rounded, are printed out from m; 1 or 2 leading zeros in m as needed ensure that the correct number of digits are printed before the decimal point (without the leading zeros of course); trailing zeros are suppressed; and E-format is printed if needed.

So many cases are possible that it is best to try examples, referring to the ZX81 manual as needed.

i First the sign of X is taken care of:

If X is negative, the subroutine jumps to P-NEG, 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 routine just continues.

```
15DB PRINT-FP   RST  0028,FP-CALC.          Use the calculator.
                DEFB +2D,duplicate,19F6     X, X
                DEFB +32,less-0,1ADB        X, (1/0)       Logical value of X.
                DEFB +00,jump-true,1C2F     X
                DEFB +0B, to 15EA,P-NEG     X
                DEFB +2D,duplicate,19F6     X, X
                DEFB +33,greater-0,1ACE     X, (1/0)       Logical value of X.
                DEFB +00,jump-true,1C2F     X
                DEFB +0D, to 15F0,P-POS     X              Hereafter X' = ABS X.
                DEFB +02,delete,19E3        -
                DEFB +34,end-calc., 002B    -
                LD   A,+1C                  Enter the character code for '0'.
                RST  0010,PRINT-A           Print the '0'.
                RET                         Finished as the 'last value' is equal to zero.
15EA P-NEG      DEFB +27,abs,1AAA           X'             X' = ABS X.
                DEFB +34,end-calc.,002B     X'
                LD   A,+16                  Enter the character code for '-'.
                RST  0010,PRINT-A           Print the '-'.
                RST  0028,FP-CALC.          Use the calculator again.
15F0 P-POS      DEFB +34,end-calc.,002B     Exit with HL pointing to the exponent
                                            byte of X'.
```

ii The number n is calculated and stored in mem-1, to be recalled for use after the 'print buffer' has been set up. Note that e' is obtained by subtracting Hex.80 from the full exponent e presently addressed by the HL register pair. In fact 128.5 decimal is subtracted all at once. It and log of 2 to base 10 are both stacked as immediate data by calling 'stk-data' at 19FC.

```
                LD   A,(HL)                 Fetch the exponent e of X'.
                CALL 151D,STACK-A           X', e
                RST  0028,FP-CALC.          Use the calculator.
                DEFB +30,stk-data,19FC      X', e, 128.5 (dec)
                DEFB +78,exponent 88
                DEFB +00,+80,(+00,+00)
                DEFB +03,subtract,174C      X', e'-.5
                DEFB +30,stk-data,19FC      X', e'-.5, log 2 (base 10)
                DEFB +EF,exponent 7F
                DEFB +1A,+20,+9A,+85
                DEFB +04,multiply,17C6      X', (e'-.5)*log 2
                DEFB +24,int,1C46           X', n
                DEFB +C1,st-mem-1,1A63      X', n          (n is copied to mem-1)
```

iii Next m is calculated, providing enough digits to give a print buffer from which the 8 most significant digits of X, correctly rounded, can be printed out.

```
                DEFB +30,stk-data,19FC      X', n, 8
                DEFB +34,exponent 84
                DEFB +00,(+00,+00,+00)
                DEFB +03,subtract,174C      X', n-8
                DEFB +18,negate,1AA0        X', 8-n
```

```
                    DEFB  +38,e-to-fp,155A        10**(8-n)*X'
                    DEFB  +A2,stk-half,1A51       10**(8-n)*X', .5
                    DEFB  +0F,addition,1755       10**(8-n)*X'+.5
                    DEFB  +24,int,1C46            m
                    DEFB  +34,end-calc.,002B      m
```

iv Ten digits from m are now stored in mem-3 and mem-4 in reverse order. This means that up to 2 leading zeros are stored (since m has 8 to 10 digits) and this will ensure that the correct number of digits are printed before the decimal in X.

```
                    LD    HL,+406B                Address of last byte of mem-2.
                    LD    (HL),+90                Marker byte Hex.90 – see 1620 below.
                    LD    B,+0A                   B will count the 10 digits.
```

Perform the following loop 10 times.

```
1615  NXT-DGT-3     INC   HL                      Each byte of mem-3 and mem-4.
                    PUSH  HL                      Save the pointer.
                    PUSH  BC                      Save the digit-counter.
                    RST   0028,FP-CALC.           Use the calculator.
                    DEFB  +A4,stk-ten,1A51        m, 10
                    DEFB  +2E,n-mod-m,1C37        m mod 10, INT (m/10)
                    DEFB  +01,exchange,1A72       INT (m/10), m mod 10
                    DEFB  +34,end-calc.,002B
                    CALL  15CD,FP-TO-A            A will hold m mod 10.
1620                OR    +90                     Add left nibble of Hex.9 to each digit;
                                                  this ensures full carry on half carry
                                                  after DAA.
                    POP   BC                      Restore the digit-counter.
                    POP   HL                      Restore the pointer.
                    LD    (HL),A                  Store this digit in the buffer.
                    DJNZ  1615,NXT-DGT-3          Until 10 digits have been stored.
```

Pass over any leading zeros.

```
                    INC   HL                      Point one-past the end of mem-4.
                    LD    BC,+0008                Looking for 8 digits.
                    PUSH  HL                      Save the pointer.
162C  GET-FIRST     DEC   HL                      Pass over any leading zeros; the
                    LD    A,(HL)                  first non-zero digit will be the
                    CP    +90                     first digit of X to be printed.
                    JR    Z,162C,GET-FIRST        Jump back if digit is zero.
```

Round up the digits if necessary.

```
                    SBC   HL,BC                   Point to the 9th digit; use it to
                    PUSH  HL                      round up 8th digit; first save the
                    LD    A,(HL)                  pointer here, then add Hex.6B.
                    ADD   A,+6B                   (6B + 95 = 0100 Hex. & carry set)
                    PUSH  AF                      Save the carry flag.
1639  ROUND-UP      POP   AF                      Restore the carry inside loop.
                    INC   HL                      Increment the pointer.
                    LD    A,(HL)                  Get the digit and round it up by
                    ADC   A,+00                   adding in the carry.
                    DAA                           Set the carry if the digit becomes
                                                  10 decimal.
                    PUSH  AF                      Save the new carry.
                    AND   +0F                     Remove the left nibble of the digit.
                    LD    (HL),A                  Store the digit.
                    SET   7,(HL)                  This changes Hex.00 to Hex.80 and
                                                  prevents any final 0's after the decimal
                                                  from being printed. (see 164B,MARKERS)
```

```
                    JR    Z,1639,ROUND-UP       Go for any carry ripple or further
                                                final zeros.
                    POP   AF                    Discard the carry.
                    POP   HL                    Restore the pointer to the 9th digit.
```

Enter six marker bytes.

```
                    LD    B,+06                 These six markers will end output
164B  MARKERS       LD    (HL),+80              by setting the overflow flag after
                    DEC   HL                    DEC and INC – see 16C4 and 16CA
                    DJNZ  164B,MARKERS          below.
```

Note that the markers are in the 6 locations which precede the 8 significant digits of the number; so they will end the output even after 13 digits are printed; a marker will turn into a '0' when its left nibble is cleared.

v The digits can now be printed.

```
                    RST   0028,FP-CALC.         Use the calculator.
                    DEFB  +02,delete,19E3       Delete the '0' left on the stack.
                    DEFB  +E1,get-mem-1,1A45    Get the number n from mem-1.
                    DEFB  +34,end-calc.,002B
                    CALL  15CD,FP-TO-A          Put ABS n into the A register.
                    JR    Z,165B,SIGND-EXP      If n positive (Z flag set), jump.
                    NEG                         Else, negate A.
165B  SIGND-EXP     LD    E,A                   A now holds true n; copy to E.
                    INC   E
                    INC   E                     E now holds n+2.
                    POP   HL                    Get the pointer to one-past the end
                                                of mem-4.
165F  GET-FST-2     DEC   HL                    Find first non-zero digit of X
                    DEC   E                     again, thus passing over the 1 or 2
                    LD    A,(HL)                leading zeros that may be present;
                    AND   +0F                   decrease E to ensure that the correct
                    JR    Z,165F,GET-FST-2      number of digits before the decimal
                                                are printed.
                    LD    A,E                   Put count back into A; at this point
                                                -5 and 12 are the critical values of
                                                the counter.
                    SUB   +05                   Subtract 5; -10 and 7 are now the
                    CP    +08                   critical values; i.e. the jump to
                    JP    P,1682,E-NEEDED       E-NEEDED will now occur, unless A
                    CP    +F6                   is less than 8, or greater than 245.
                    JP    M,1682,E-NEEDED       (245 dec. is -11 in 2's comp.)
                    ADD   A,+06                 Add 6, giving the true critical values,
                                                i.e. -4 and 13.
```

Note that A now contains the correct number of digits before the decimal in X, and that these digits will be printed in full if they are not more than 13 decimal, while up to 4 initial zeros will be printed after the decimal if A is negative. Outside that range E-format will be needed.

```
                    JR    Z,16BF,OUT-ZERO       If A holds zero then go and print
                                                a '0' and continue into decimal part.
                    JP    M,16B2,EXP-MINUS      If A is minus then go and print the
                                                'decimal-point' and the digits.
                    LD    B,A                   A is positive, so transfer to B.
167B  OUT-B-CHS     CALL  16D0,OUT-NEXT         Print B characters.
                    DJNZ  167B,OUT-B-CHS        Then jump forward to test whether
                    JR    16C2,TEST-INT         just an integer, or a 'decimal-point'
                                                is needed.
```

E-format is required.

```
1682  E-NEEDED  LD    B,E                  B now contains the correct integer to
                                           follow 'E' of E-format.
                CALL  16D0,OUT-NEXT        Print the first digit.
                CALL  16C2,TEST-INT        Test whether there are more non-zero
                                           digits, in which case a 'decimal-point'
                                           will be needed.
                LD    A,+2A                Enter the character code for 'E'.
                RST   0010,PRINT-A         Print the 'E'.
                LD    A,B                  Transfer the 'exponent' integer to A.
                AND   A                    Set the flags.
                JP    P,1698,PLUS-SIGN     If positive, jump and print a '+'.
                NEG                        Else, change its sign.
                LD    B,A                  Transfer back to B, briefly.
                LD    A,+16                Enter the character code for '-'.
                JR    169A,OUT-SIGN        Jump forward.
1698  PLUS-SIGN LD    A,+15                Enter the character code for '+'.
169A  OUT-SIGN  RST   0010,PRINT-A         Print the sign character.
                LD    A,B                  Transfer the 'exponent' back to A.
                LD    B,+FF                Now reduce A mod 10 to give B equal
169E  TEN-MORE  INC   B                    to INT (A/10); initialise B to -1
                SUB   +0A                  (2's comp.) and increment it each
                JR    NC,169E,TEN-MORE     time A is decreased by 10.
                ADD   A,+0A                After the loop, restore the last 10
                LD    C,A                  to A; and store A in the C register.
                LD    A,B                  Transfer the 'tens' to A.
                AND   A                    Test to see if there are any 'tens'.
                JR    Z,16AD,BYTE-TWO      Jump forward if no 'tens'.
                CALL  07EB,OUT-CODE        Print the first digit.
16AD  BYTE-TWO  LD    A,C                  Fetch the 'unit' digit.
                CALL  07EB,OUT-CODE        Print the digit.
                RET                        Finished with E-format.
```

Decimal format is required.

```
16B2  EXP-MINUS NEG                        A was negative but in range for simple
                                           printing so the format is .000...dddd
                                           with up to 4 zeros.
                LD    B,A                  B will count out the zeros.
                LD    A,+1B                Enter the character code for '.' .
                RST   0010,PRINT-A         Now print the 'decimal-point'.
                LD    A,+1C                Enter the character code for '0' .
16BA  OUT-ZEROS RST   0010,PRINT-A         Print the '0' .
                DJNZ  16BA,OUT-ZEROS       Until B reaches zero.
                JR    16C8,TEST-DONE       Exit via TEST-DONE to print the
                                           digits until they also are finished.
```

The special case of the 'exponent' being zero.

```
16BF  OUT-ZERO  LD    A,+1C                Enter the character code for '0'.
                RST   0010,PRINT-A         Print the '0' and continue with
                                           TEST-INT to print the decimal part.
```

THE 'TEST-INT' SUBROUTINE

If the next digit to be printed is a 'marker' byte then the subroutine returns, otherwise the decimal point is printed and the subroutine enters TEST-DONE.

```
16C2  TEST-INT  DEC   (HL)                 This gives PE (overflow/parity flag
                                           set) if (HL) was Hex.80.
                INC   (HL)                 PE is kept, incrementing to Hex.80.
```

```
16C4                 RET   PE                 So a 'marker' byte forces a return.
                     LD    A,+1B              Enter the character code for '.' .
                     RST   0010,PRINT-A       Now print the 'decimal-point'.
```

Note that the decimal point is not printed if the number is an integer, all printed, or if there is just one digit to go before the 'E' of the exponent part.

THE 'TEST-DONE' SUBROUTINE

The digits in the *ad hoc* print buffer, mem-2 to mem-4, are printed in turn until a 'marker' byte is found.

```
16C8  TEST-DONE DEC   (HL)               Test the digit to see if it is a
                INC   (HL)               'marker' (see TEST-INT).
16CA            RET   PE                 Return when a 'marker' is found.
                CALL  16D0,OUT-NEXT      Print the digit.
                JR    16C8,TEST-DONE     Jump back to consider the next digit.
```

THE 'OUT-NEXT' SUBROUTINE

This subroutine prepares the current digit for printing, passes it to OUT-CODE and moves the pointer to the next digit.

```
16D0  OUT-NEXT  LD    A,(HL)             Fetch the present digit.
                AND   +0F                Mask off any unwanted bits.
                CALL  07EB,OUT-CODE      Pass the digit for actual printing.
                DEC   HL                 Move the pointer back an address.
                RET                      Finished.
```

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 (2'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.

```
16D8  PREP-ADD  LD    A,(HL)             Transfer the exponent to A.
                LD    (HL),+00           Presume a positive number.
                AND   A                  If the number is zero then the
                RET   Z                  preparation is already finished.
                INC   HL                 Now point to the sign byte.
                BIT   7,(HL)             Set the zero flag for positive number.
                SET   7,(HL)             Restore the true numeric bit.
                DEC   HL                 Point to the first byte again.
                RET   Z                  Positive numbers have been prepared,
                                         but negative numbers need to be 2's
                                         complemented.
                PUSH  BC                 Save any earlier exponent.
                LD    BC,+0005           There are 5 bytes to be handled.
                ADD   HL,BC              Point one-past the last byte.
                LD    B,C                Transfer the '5' to B.
                LD    C,A                Save the exponent in C.
                SCF                      Set carry flag for negation.
16EC  NEG-BYTE  DEC   HL                 Point to each byte in turn.
                LD    A,(HL)             Get each byte.
                CPL                      One's complement the byte.
```

```
          ADC   A,+00                    Add in carry for negation.
          LD    (HL),A                   Restore the byte.
          DJNZ  16EC,NEG-BYTE            Loop the '5' times.
          LD    A,C                      Restore the exponent to A.
          POP   BC                       Restore any earlier exponent.
          RET                            Finished.
```

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 registers, 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.

```
16F7  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 for 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).

```
171A  SHIFT-FP   AND   A                        If the exponent difference is zero,
                 RET   Z                        the subroutine returns at once.
                 CP    +21                      If the difference is greater than
                 JR    NC,1736,ADDEND-0         Hex.20, jump forward.
                 PUSH  BC                       Save BC briefly.
                 LD    B,A                      Transfer the exponent difference to
                                                B to count the shifts right.
1722  ONE-SHIFT  EXX                            Arithmetic shift right for L',
                 SRA   L                        preserving the sign marker bits.
                 RR    D                        Rotate right with carry D', E',
                 RR    E                        D & E.
                 EXX                            Thereby shifting the whole five bytes
                 RR    D                        of the number to the right as
                 RR    E                        many times as B counts.
                 DJNZ  1722,ONE-SHIFT           Loop back until B reaches zero.
                 POP   BC                       Restore the original BC.
                 RET   NC                       Done if no carry to retrieve.
                 CALL  1741,ADD-BACK            Retrieve carry.
                 RET   NZ                       Return unless the carry rippled
                                                right back. (In this case there is nothing
                                                to add)
1736  ADDEND-0   EXX                            Fetch L', D' & E'.
                 XOR   A                        Clear the A register.
1738  ZEROS-4/5  LD    L,+00                    Set the addend to zero in D', E',
                 LD    D,A                      D & E, together with its marker byte
                 LD    E,L                      (sign indicator) L', which was
                 EXX                            Hex.00 for a positive number and
                 LD    DE,+0000                 Hex.FF for a negative number.
                                                ZEROS-4/5 produces only 4 zero bytes
                                                when called for near underflow at 1833.
                 RET                            Finished.
```

Note: The original 8K ROM had 3 further bytes in this subroutine, immediately after the EXX at the label ADDEND-0 (address 1733 in the old ROM), namely LD A,H; SUB L; & LD H,A. These bytes would seem to have been a mistaken attempt to counteract the effect of bytes 177D – 177F below. In fact they caused errors in subtraction and, through the LN function at byte 1D15, in exponentiation and SQR as well. These three bytes were simply omitted when the program was improved. It is interesting to note also that the hardware add-on, fitted to some 'unimproved' machines worked by changing the instruction LD H,A to a DAA instruction and thereby prevented any corruption of the H register.

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.

```
1741  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                           Get the next byte.
                  INC   E                       Increment it too.
                  JR    NZ,174A,ALL-ADDED       Jump if no overflow.
                  INC   D                       Increment the last byte.
174A  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 carries on into ADDITION. Note that HL points to the minuend and DE points to the subtrahend. (See ADDITION for more details.)

```
174C  SUBTRACT    LD    A,(DE)                  Get the exponent byte of subtrahend.
                  AND   A                       Test whether zero.
                  RET   Z                       If so, return.
                  INC   DE                      Point to the sign byte.
                  LD    A,(DE)                  Transfer the sign byte to A.
                  XOR   +80                     Change the sign bit.
                  LD    (DE),A                  Replace the byte.
                  DEC   DE                      Point to the exponent byte again.
                                                Continue on into ADDITION.
```

THE 'ADDITION' OPERATION (Offset OF – see CALCULATE below: 'addition')

The first of three major arithmetical subroutines, this subroutine carries out 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.

ADDITION first calls PREP-ADD for each number, then gets the 2 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 2's complemented if negative, and any arithmetic overflow is reported; otherwise the subroutine jumps to TEST-NORM to normalize the result and return it to the stack with the correct sign bit inserted into the second byte.

```
1755  addition    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  16D8,PREP-ADD           Prepare the augend.
                  LD    B,A                     Save its exponent in B.
                  EX    DE,HL                   Exchange the pointers.
                  CALL  16D8,PREP-ADD           Prepare the addend.
                  LD    C,A                     Save its exponent in C.
                  CP    B                       If the first exponent is smaller,
                  JR    NC,1769,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.
1769  SHIFT-LEN   PUSH  AF                      Save the larger exponent in A.
                  SUB   B                       The difference between the exponents
                                                is the length of the shift right.
```

```
                   CALL  16F7,FETCH-TWO     Get the two numbers from the stack.
                   CALL  171A,SHIFT-FP      Shift the addend right.
                   POP   AF                 Restore the larger exponent.
                   POP   HL                 HL is to point to the result.
                   LD    (HL),A             Store the exponent of the result.
                   PUSH  HL                 Save the pointer again.
                   LD    L,B                M4 to L & M5 to H,
                   LD    H,C                (see FETCH-TWO).
                   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 mechanism will ensure
                   LD    L,A                that a single shift right is called
                   RRA                      if the sum of 2 positive numbers
                   XOR   L                  has overflowed left, or the sum of 2
                   EXX                      negative numbers has not overflowed left.
                   EX    DE,HL              The result is now in DED'E'.
                   POP   HL                 Get the pointer to the exponent.
                   RRA                      The test for shift (H', L' were
                   JR    NC,1790,TEST-NEG   Hex.00 for positive numbers and
                                            Hex.FF for negative numbers).
                   LD    A,+01              A counts a single shift right.
                   CALL  171A,SHIFT-FP      The shift is called.
                   INC   (HL)               Add 1 to the exponent; this may
                   JR    Z,17B3,ADD-REP-6   lead to arithmetic overflow.
1790  TEST-NEG     EXX                      Test for negative result: get
                   LD    A,L                sign bit of L' into A (this now
                   AND   +80                correctly indicates the sign of
                   EXX                      the result).
                   INC   HL                 Store it in the second byte
                   LD    (HL),A             position of the result on
                   DEC   HL                 the calculator stack.
                   JR    Z,17B9,GO-NC-MLT   If it is zero, then do not
                                            2's complement the result.
                   LD    A,E                Get the first byte.
                   NEG                      Negate it.
                   CCF                      Complement the carry for continued
                   LD    E,A                negation, and store byte.
                   LD    A,D                Get the next byte.
                   CPL                      One's complement it.
                   ADC   A,+00              Add in the carry for negation.
                   LD    D,A                Store the byte.
                   EXX                      Proceed to get next byte into the
                   LD    A,E                A register.
                   CPL                      One's complement it.
                   ADC   A,+00              Add in the carry for negation.
                   LD    E,A                Store the byte.
                   LD    A,D                Get the last byte.
                   CPL                      One's complement it.
                   ADC   A,+00              Add in the carry for negation.
                   JR    NC,17B7,END-COMPL  Done if no carry.
                   RRA                      Else, get .5 into mantissa and add 1
                   EXX                      to the exponent; this will be needed
                   INC   (HL)               when two negative numbers add to give
                                            an exact power of 2, and it may lead to
                                            arithmetic overflow.
17B3  ADD-REP-6    JP    Z,1880,REPORT-6    Give the error if required.
                   EXX
```

```
17B7  END-COMPL LD    D,A              Store the last byte.
                EXX
17B9  GO-NC-MLT XOR   A                Clear the carry flag.
                JR    1828,TEST-NORM   Exit via TEST-NORM.
```

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.

```
17BC  PREP-M/D  SCF               Set the carry flag.
                DEC   (HL)        Test the exponent byte.
                INC   (HL)
                RET   Z           If the number is zero, return with both
                                  the zero and the carry flags 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
                                  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 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 normalized and returned to the calculator stack with the correct sign bit in the second byte.

```
17C6  multiply  XOR   A                A is set to Hex.00 so that the sign
                                       of the first number will go into A.
                CALL  17BC,PREP-M/D    Prepare the first number, and return
                RET   C                if zero. (Result already zero.)
                EXX                    Exchange the registers.
                PUSH  HL               Save the next literal address.
                EXX                    Exchange the registers.
                PUSH  DE               Save the pointer to the multiplicand.
                EX    DE,HL            Exchange the pointers.
                CALL  17BC,PREP-M/D    Prepare the 2nd number.
                EX    DE,HL            Exchange the pointers again.
                JR    C,1830,ZERO-RSLT Jump forward if 2nd number is zero.
                PUSH  HL               Save the pointer to the result.
                CALL  16F7,FETCH-TWO   Get the two numbers from the stack.
                LD    A,B              M5 to A (see FETCH-TWO).
                AND   A                Prepare for a subtraction.
                SBC   HL,HL            Initialise HL to zero for the result.
                EXX                    Exchange the registers.
                PUSH  HL               Save M1 & N1 (see FETCH-TWO).
                SBC   HL,HL            Also initialise H'L' for the result.
                EXX                    Exchange the registers.
                LD    B,+21            B counts 33 decimal, Hex.21, shifts.
                JR    17F8,STRT-MLT    Jump forward into the loop.
```

Now enter the multiplier loop.

```
17E7  MLT-LOOP   JR    NC,17EE,NO-ADD       Jump forward to NO-ADD if no carry,
                                            i.e. the multiplier bit was reset;
                 ADD   HL,DE                Else, add the multiplicand in
                 EXX                        D'E'DE (see FETCH-TWO) into the
                 ADC   HL,DE                result being built up in H'L'HL.
                 EXX
17EE  NO-ADD     EXX                        Whether multiplicand was added
                 RR    H                    or not, shift result right in
                 RR    L                    H'L'HL, i.e. the shift is done by
                 EXX                        rotating each byte with carry, so that
                 RR    H                    any bit that drops into the carry is
                 RR    L                    picked up by the next byte, and the
                                            shift continues into B'C'CA.
17F8  STRT-MLT   EXX                        Shift right the multiplier in
                 RR    B                    B'C'CA (see FETCH-TWO & above).
                 RR    C                    A final bit dropping into the carry
                 EXX                        will trigger another add of the
                 RR    C                    multiplicand to the result.
                 RRA
                 DJNZ  17E7,MLT-LOOP        Loop 33 times to get all the bits.
                 EX    DE,HL                Move the result from:
                 EXX
                 EX    DE,HL                H'L'HL to D'E'DE.
                 EXX
```

Next add the exponents together.

```
                 POP   BC                   Restore the exponents – M1 & N1.
                 POP   HL                   Restore the pointer to the exponent byte.
                 LD    A,B                  Get the sum of the two exponent
                 ADD   A,C                  bytes in A, and the correct carry.
                 JR    NZ,180E,MAKE-EXPT    If the sum equals zero then clear
                 AND   A                    the carry; else leave it unchanged.
180E  MAKE-EXPT  DEC   A                    Prepare to increase the exponent by
                 CCF                        Hex.80.
```

The rest of the subroutine is common to both MULTIPLICATION and DIVISION.

```
1810  DIVN-EXPT  RLA                        These few bytes very cleverly make
                 CCF                        the correct exponent byte.
                 RRA                        Rotating left then right gets the
                                            exponent byte (true exponent plus
                                            Hex.80) into A.
                 JP    P,1819,OFLW1-CLR     If the sign flag is reset, no report of
                                            arithmetic overflow needed.
                 JR    NC,1880,REPORT-6     Report the overflow if carry reset.
                 AND   A                    Clear the carry now.
1819  OFLW1-CLR  INC   A                    The exponent byte is now complete;
                 JR    NZ,1824,OFLW2-CLR    but if A is zero a further check for
                 JR    C,1824,OFLW2-CLR     overflow is needed.
                 EXX                        If there is no carry set and the
                 BIT   7,D                  result is already in normal form
                 EXX                        (bit 7 of D' set) then there is overflow to
                 JR    NZ,1880,REPORT-6     report; but if bit 7 of D' is reset, the
                                            result is just in range, i.e. just under
                                            2**127.
1824  OFLW2-CLR  LD    (HL),A               Store the exponent byte, at last.
                 EXX                        Pass the fifth result byte to A for the
                 LD    A,B                  normalization sequence, i.e.
                 EXX                        the overflow from L into B'.
```

The remainder of the subroutine that deals with normalization is common to all the arithmetic routines.

```
1828  TEST-NORM JR    NC,183F,NORMALIZE    If no carry then normalize now.
                LD    A,(HL)               Else, deal with underflow (zero result)
                AND   A                    or near underflow
182C  NEAR-ZERO LD    A,+80                (result 2** -128):
                JR    Z,1831,SKIP-ZERO     return exponent to A, test if A is
1830  ZERO-RSLT XOR   A                    zero (case 2**-128) and if so
1831  SKIP-ZERO EXX                        produce 2**-128 if number is normal;
                AND   D                    otherwise produce zero.
                CALL  1738,ZEROS-4/5       The exponent must then be set to
                RLCA                       zero (for zero) or 1 (for 2**-128).
                LD    (HL),A               Restore the exponent byte.
                JR    C,1868,OFLOW-CLR     Jump if case 2** -128.
                INC   HL                   Otherwise, put zero into second
                LD    (HL),A               byte of result on the calculator
                DEC   HL                   stack.
                JR    1868,OFLOW-CLR       Jump forward to transfer the result.
```

The actual normalization operation.

```
183F  NORMALIZE LD    B,+20                Normalize the result by up to 32
1841  SHIFT-ONE EXX                        decimal, Hex.20, shifts left of
                BIT   7,D                  D'E'DE (with A adjoined) until bit 7
                EXX                        of D' is set. A holds zero after
                JR    NZ,1859,NORML-NOW    addition, so no precision is
                RLCA                       gained or lost; A holds the fifth
                RL    E                    byte from B' after multiplication
                RL    D                    or division; but as only about 32
                EXX                        bits can be correct, no precision
                RL    E                    is lost. Note that A is rotated
                RL    D                    circularly, with branch at carry...
                EXX                        ..eventually a random process.
                DEC   (HL)                 The exponent is decremented on
                                           each shift.
                JR    Z,182C,NEAR-ZERO     If the exponent becomes zero, then
                                           numbers from 2** -129 are rounded
                                           up to 2** -128.
                DJNZ  1841,SHIFT-ONE       Loop back, up to 32 times.
                JR    1830,ZERO-RSLT       If bit 7 never became 1 then the
                                           whole result is to be zero.
```

Finish the normalization by considering the 'carry'.

```
1859  NORML-NOW RLA                        After normalization add back any
                JR    NC,1868,OFLOW-CLR    final carry that went into A.
                CALL  1741,ADD-BACK        Jump forward if the carry does not
                JR    NZ,1868,OFLOW-CLR    ripple right back.
                EXX                        If it should ripple right back then
                LD    D,+80                set mantissa to 0.5 and increment
                EXX                        the exponent.
                INC   (HL)                 This action may lead to arithmetic
                JR    Z,1880,REPORT-6      overflow (final case).
```

The final part of the subroutine involves passing the result to the bytes reserved for it on the calculator stack and resetting the pointers.

```
1868  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 present
                    PUSH  DE            registers, D'E'DE, to BCDE; and
                    EXX                 then to ACDE.
                    POP   BC
                    LD    A,B           The sign bit is retrieved from its
                    RLA                 temporary store and transferred to
                    RL    (HL)          its correct position of bit 7 of the
                    RRA                 first byte of the mantissa.
                    LD    (HL),A        The first byte is stored.
                    INC   HL            Next.
                    LD    (HL),C        The second byte is stored.
                    INC   HL            Next.
                    LD    (HL),D        The third byte is stored.
                    INC   HL            Next.
                    LD    (HL),E        The fourth byte is stored.
                    POP   HL            Restore the pointer to the result.
                    POP   DE            Restore the pointer to second number.
                    EXX                 Exchange the registers.
                    POP   HL            Restore the next literal address.
                    EXX                 Exchange the registers.
                    RET                 Finished.
```

REPORT-6 – Arithmetic overflow

```
1880  REPORT-6      RST   0008,ERROR-1
                    DEFB  +05
```

THE 'DIVISION' OPERATION (Offset 05 – see CALCULATE below: 'division')

This subroutine first prepares the divisor by calling PREP-M/D, reporting arithmetic overflow if it is zero; then it prepares the dividend by again calling PREP-M/D, returning if it is zero. Next it fetches the two numbers from the calculator stack and divides their mantissas 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.

```
1882  division      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  17BC,PREP-M/D     Prepare the divisor and give the
                    JR    C,1880,REPORT-6   report for arithmetic overflow if it
                                            is zero.
                    EX    DE,HL             Exchange the pointers.
                    CALL  17BC,PREP-M/D     Prepare the dividend and return if
                    RET   C                 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  16F7,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
                    LD    L,C               from registers B'C'CB (i.e. M2, M3,
                    EXX                     M4 & M5; see FETCH-TWO) to the
                    LD    H,C               registers H'L'HL.
                    LD    L,B
                    XOR   A                 Clear A and reset the carry flag.
```

```
                     LD    B,+DF               B will count upwards from -33 to -1,
                                               2's complement, Hex.DF to FF, looping
                                               on minus and will jump again on zero
                                               for extra precision.
                     JR    18B2,DIV-START      Jump forward into the division loop
                                               for the first trial subtraction.
```

Now enter the division loop.

```
18A2  DIV-LOOP   RLA                           Shift the result left into B'C'CA,
                 RL    C                       shifting out the bits already there,
                 EXX                           picking up 1 from the carry
                 RL    C                       whenever it is set, and rotating
                 RL    B                       left each byte with carry to
                 EXX                           achieve the 32 bit shift.
                 ADD   HL,HL                   Move what remains of the dividend
                 EXX                           left in H'L'HL before the next
                 ADC   HL,HL                   trial subtraction; if a bit drops into the
                 EXX                           carry, force no restore and a bit for the
                                               quotient, thus retrieving the lost bit and
                 JR    C,18C2,SUBN-ONLY        allowing a full 32-bit divisor.
18B2  DIV-START  SBC   HL,DE                   Trial subtract divisor in D'E'DE
                 EXX                           from rest of dividend in H'L'HL;
                 SBC   HL,DE                   there is no initial carry (see
                 EXX                           previous step).
                 JR    NC,18C9,NO-RSTORE       Jump forward if there is no carry.
                 ADD   HL,DE                   Otherwise restore, i.e. add back the
                 EXX                           divisor. Then clear the carry so that
                 ADC   HL,DE                   there will be no bit for the
                 EXX                           quotient (the divisor 'did not go').
                 AND   A
                 JR    18CA,COUNT-ONE          Jump forward to the counter.
18C2  SUBN-ONLY  AND   A                       Just subtract with no restore and
                 SBC   HL,DE                   go on to set the carry flag because
                 EXX                           the lost bit of the dividend is to
                 SBC   HL,DE                   be retrieved and used for the
                 EXX                           quotient.
18C9  NO-RSTORE  SCF                           One for the quotient in B'C'CA.
18CA  COUNT-ONE  INC   B                       Step the loop count up by one.
                 JP    M,18A2,DIV-LOOP         Loop 32 times for all bits.
                 PUSH  AF                      Save any 33rd bit for extra precision
                                               (the present carry).
                 JR    Z,18B2,DIV-START        Trial subtract yet again for any 34th bit;
                                               the PUSH AF above saves this bit too.
                 LD    E,A                     Now move the four bytes that form
                 LD    D,C                     the mantissa bytes of the result
                 EXX                           from B'C'CA to D'E'DE.
                 LD    E,C
                 LD    D,B
                 POP   AF                      Then put any 34th and 33rd bits into
                 RR    B                       B' to be picked up on normalization.
                 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
                 SUB   C                       exponent bytes into A and
                                               set the carry flag if required.
                 JP    1810,DIVN-EXPT          Exit via DIVN-EXPT.
```

THE 'INTEGER TRUNCATION TOWARDS ZERO' SUBROUTINE

(Offset 36 – 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 zero if the exponent byte of X is less than Hex.81 (mod X less than 1). It returns X if the exponent byte is Hex.A0 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.

```
18E4  truncate   LD    A,(HL)             Get the exponent byte of X into A.
                 CP    +81                Compare e, the exponent, to Hex.81.
                 JR    NC,18EF,X-LARGE    Jump if e is greater than Hex.80.
                 LD    (HL),+00           Else, set the exponent to zero;
                 LD    A,+20              enter 32 decimal, Hex.20, into A
                 JR    18F4,NIL-BYTES     and jump forward to NIL-BYTES to
                                          make all the bits of X be zero.
18EF  X-LARGE    SUB   +A0                Subtract 160 decimal, Hex.A0, from e.
                 RET   P                  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.

```
18F4  NIL-BYTES  PUSH  DE                 Save the current value of DE (STKEND).
                 EX    DE,HL              Make HL point one-past the 5th byte.
                 DEC   HL                 HL now points to the 5th byte of X.
                 LD    B,A                Get the number of bits to be set to
                 SRL   B                  zero into B and divide it by 8 to give
                 SRL   B                  the number of whole bytes
                 SRL   B                  implied.
                 JR    Z,1905,BITS-ZERO   Jump forward if the result is zero.
1900  BYTE-ZERO  LD    (HL),+00           Else, set the bytes to zero; B
                 DEC   HL                 counts them.
                 DJNZ  1900,BYTE-ZERO
1905  BITS-ZERO  AND   +07                Get A (mod 8): this is the number of
                                          bits still to be set to zero.
                 JR    Z,1912,IX-END      Jump to the end if nothing more to do.
                 LD    B,A                B will count the bits now.
                 LD    A,+FF              Prepare the mask.
190C  LESS-MASK  SLA   A                  With each loop a zero enters the
                 DJNZ  190C,LESS-MASK     mask from the right and thereby a mask
                                          of the correct 'length' is produced.
                 AND   (HL)               The unwanted bits of (HL) are lost
                 LD    (HL),A             as the masking if performed.
1912  IX-END     EX    DE,HL              Return the pointer to HL.
                 POP   DE                 Return the pointer to DE, (STKEND).
                 RET                      Finished.
```

THE CALCULATOR TABLES

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.

	data:	constant:	when expanded gives: exp. mantissa: (Hex.)
1915 stk-zero	DEFB +00 DEFB +B0 DEFB +00	zero	00 00 00 00 00
1918 stk-one	DEFB +31 DEFB +00	one	81 00 00 00 00
191A stk-half	DEFB +30 DEFB +00	a half	80 00 00 00 00
191C stk-pi/2	DEFB +F1 DEFB +49 DEFB +0F DEFB +DA DEFB +A2	a half of pi	81 49 0F DA A2
1921 stk-ten	DEFB +34 DEFB +20	ten	84 20 00 00 00

The table of addresses:

This second table is a look-up table of the addresses of the 61 decimal, operational subroutines of the calculator. The offsets used to index into the table are derived either from the operation codes used in SCANNING, see 10BC etc., or from the literals that follow an RST 0028 instruction.

	offset	label	address		offset	label	address		offset	label	address
1923	00	jump-true	2F 1C	194D	15	str-less	03 1B	1977	2A	strs	D5 1B
1925	01	exchange	72 1A	194F	16	strs-eql	03 1B	1979	2B	chrs	8F 1B
1927	02	delete	E3 19	1951	17	strs-add	62 1B	197B	2C	not	D5 1A
1929	03	subtract	4C 17	1953	18	negate	A0 1A	197D	2D	duplicate	F6 19
192B	04	multiply	C6 17	1955	19	code	06 1C	197F	2E	n-mod-m	37 1C
192D	05	division	82 18	1957	1A	val	A4 1B	1981	2F	jump	23 1C
192F	06	to-power	E2 1D	1959	1B	len	11 1C	1983	30	stk-data	FC 19
1931	07	or	ED 1A	195B	1C	sin	49 1D	1985	31	dec-jr-nz	17 1C
1933	08	no.-&-no.	F3 1A	195D	1D	cos	3E 1D	1987	32	less-0	DB 1A
1935	09	no.-l-eql	03 1B	195F	1E	tan	6E 1D	1989	33	greater-0	CE 1A
1937	0A	no.-gr-eq	03 1B	1961	1F	asn	C4 1D	198B	34	end-calc.	2B 00
1939	0B	nos.-neql	03 1B	1963	20	acs	D4 1D	198D	35	get-argt.	18 1D
193B	0C	no.-grtr	03 1B	1965	21	atn	76 1D	198F	36	truncate	E4 18
193D	0D	no.-less	03 1B	1967	22	ln	A9 1C	1991	37	fp-calc-2	E4 19
193F	0E	nos.-eql	03 1B	1969	23	exp	5B 1C	1993	38	e-to-fp	5A 15
1941	0F	addition	55 17	196B	24	int	46 1C	1995	39	series-06 etc.	7F 1A
1943	10	str-&-no.	F8 1A	196D	25	sqr	DB 1D	1997	3A	stk-zero etc.	51 1A
1945	11	str-l-eql	03 1B	196F	26	sgn	AF 1A	1999	3B	st-mem-0 etc.	63 1A
1947	12	str-gr-eq	03 1B	1971	27	abs	AA 1A	199B	3C	get-mem-0 etc.	45 1A
1949	13	strs-neql	03 1B	1973	28	peek	BE 1A				
194B	14	str-grtr	03 1B	1975	29	usr	C5 1A				

Note: The last four subroutines are multi-purpose subroutines and are entered with a parameter that is a copy of the righthand five bits of the original literal. The full set follows:–

Offset 39 : series-06, series-08 & series-0C.
Offset 3A : stk-zero, stk-one, stk-half, stk-pi/2 & stk-ten.
Offset 3B : st-mem-0, st-mem-1, st-mem-2, st-mem-3, st-mem-4 & st-mem-5.
Offset 3C : get-mem-0, get-mem-1, get-mem-2, get-mem-3, get-mem-4 & get-mem-5.

Note: TABLE-CON EQU 1915
TABLE-ADD EQU 1923

THE 'CALCULATOR' 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 '34' 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 '37', 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 BERG 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.

The offset is used to index into the calculator's table of addresses, see above, to find the required subroutine address.

The parameter is used when the multi-purpose subroutines are called.

Note: A floating-point number may in reality be a set of string parameters.

```
199D  CALCULATE CALL 1B85,STK-PNTRS    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).
19A0  GEN-ENT-1 LD   A,B               Either, transfer a single operation
                LD   (BERG),A          offset to BERG temporarily, or, when
                                       using the subroutine recursively pass the
                                       parameter to BERG to be used as a
                                       counter.
19A4  GEN-ENT-2 EXX                    The return address of the subroutine is
                EX   (SP),HL           stored in H'L'. This saves the pointer
                EXX                    to the first literal. Entering the
                                       CALCULATOR at GEN-ENT-2 is used
                                       whenever BERG is in use as a counter
                                       and is not to be disturbed.
19A7  RE-ENTRY  LD   (STKEND),DE       A loop is now entered to handle each
                                       literal in the list that follows the calling
                                       instruction; so first, always set STKEND.
                EXX                    Go to the alternate register set, and
                LD   A,(HL)            fetch the literal for this loop.
                INC  HL                Make H'L' point to the next literal.
19AE  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,19C2,FIRST-38     Separate the simple literals from the
                                             multi-purpose literals. Jump with
                                             literals 00 – 38.
                    LD   D,A                 Save the literal in D.
                    AND  +60                 Continue only with bits 5 & 6.
                    RRCA                     Four right shifts make them
                    RRCA                     now bits 1 & 2.
                    RRCA
                    RRCA
                    ADD  A,+72               The offsets required are 39 – 3C,
                    LD   L,A                 and L will now hold double the
                                             required offset.
                    LD   A,D                 Now produce the parameter by
                    AND  +1F                 taking bits 0,1,2,3 & 4 of the
                                             literal; keep the parameter in A.
                    JR   19D0,ENT-TABLE      Jump forward to find the address of
                                             the required subroutine.
19C2  FIRST-38      CP   +18                 Jump forward if performing a
                    JR   NC,19CE,DOUBLE-A    unary operation.
                    EXX                      All of the subroutines that perform
                    LD   BC,+FFFB            binary operations require that
                    LD   D,H                 HL points to the first operand and
                    LD   E,L                 DE points to the second operand (the
                    ADD  HL,BC               'last value') as they appear on
                    EXX                      the calculator stack.
19CE  DOUBLE-A      RLCA                     As each entry in the table of addresses
                    LD L,A                   takes up two bytes the offset
                                             produced is doubled.
19D0  ENT-TABLE     LD   DE,+TABLE-ADD       The base address of the table.
                    LD   H,+00               The address of the required table
                    ADD  HL,DE               entry is formed in HL; and the
                    LD   E,(HL)              required subroutine address is
                    INC  HL                  loaded into the DE register pair.
                    LD   D,(HL)
                    LD   HL,+RE-ENTRY        The RE-ENTRY address of 19A7 is
                    EX   (SP),HL             put on the machine stack underneath
                    PUSH DE                  the subroutine address.
                    EXX                      Return to the main set of registers.
                    LD   BC,(STKEND-hi.)     The current value of BERG is
                                             transferred to the B register thereby
                                             returning the single operation offset.
                                             (See COMPARISON at 1B03)
19E3  delete        RET                      An indirect jump to the required
                                             subroutine.
```

THE 'DELETE' SUBROUTINE (Offset 02: 'delete')

This subroutine contains only the single RET instruction at 19E3, 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 37: 'fp-calc-2')

This subroutine is only called from SCANNING, see page 2, 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 BERG.

The effect of calling this subroutine is essentially to make a jump to the appropriate subroutine for the single operation.

19E4	fp-calc-2	POP	AF	Discard the RE-ENTRY address.
		LD	A,(BERG)	Transfer the offset to A.
		EXX		Enter the alternate register set.
		JR	19AE,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.

19EB	TEST-5-SP	PUSH	DE	Save DE briefly.
		PUSH	HL	Save HL briefly.
		LD	BC,+0005	Specify the test is for 5 bytes.
		CALL	0EC5,TEST-ROOM	Make the test.
		POP	HL	Restore HL.
		POP	DE	Restore DE.
		RET		Finished.

THE 'MOVE A FLOATING-POINT NUMBER' SUBROUTINE (Offset 2D: '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.

19F6	MOVE-FP	CALL	19EB,TEST-5-SP	A test is made for room.
		LDIR		Move the five bytes involved.
		RET		Finished.

THE 'STACK LITERALS' SUBROUTINE (Offset 30: '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 '30' the literals follow the '30' in the list of literals; when called by the SERIES GENERATOR, see below, the literals are supplied by the subroutine 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 (1915–1922).

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.

19FC	STK-DATA	LD	H,D	This subroutine performs the manipulatory operation of adding a 'last value' to the calculator stack; hence HL is set to point one-past the present 'last value' and hence point to the result.
		LD	L,E	

```
19FE STK-CONST CALL 19EB,TEST-5-SP       Now test that there is indeed room.
               EXX                       Go to the alternate register set and
               PUSH HL                   stack the pointer to the next
               EXX                       literal.
               EX   (SP),HL              Switch over the result pointer and the
                                         next literal pointer.
               PUSH BC                   Save BC briefly.
               LD   A,(HL)               The first literal is put into A
               AND  +C0                  and divided by Hex.40 to give the
               RLCA                      integer values 0, 1, 2 or 3.
               RLCA
               LD   C,A                  The integer value is transferred to
               INC  C                    C and incremented, thereby giving
                                         the range 1, 2, 3 or 4 for the number
                                         of literals that will be needed.
               LD   A,(HL)               The literal is fetched anew, reduced
               AND  +3F                  mod Hex.40 and discarded as
               JR   NZ,1A14,FORM-EXP     inappropriate if the remainder is
               INC  HL                   zero; in which case the next literal
               LD   A,(HL)               is fetched and used unreduced.
1A14 FORM-EXP  ADD  A,+50                The exponent, e, is formed by the
               LD   (DE),A               addition of Hex.50 and passed to the
                                         calculator stack as the first of the
                                         five bytes of the result.
               LD   A,+05                The number of literals specified
               SUB  C                    in C are taken from the source
               INC  HL                   and entered into the bytes of the
               INC  DE                   result.
               LD   B,+00
               LDIR
               POP  BC                   Restore BC.
               EX   (SP),HL              Return the result pointer to HL
               EXX                       and the next literal pointer to
               POP  HL                   its usual position in H' & L'.
               EXX
               LD   B,A                  The number of zero bytes required
               XOR  A                    at this stage is given by 5-C-1;
1A27 STK-ZEROS DEC  B                    and this number of zeros is added
               RET  Z                    to the result to make up the
               LD   (DE),A               required five bytes.
               INC  DE
               JR   1A27,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.

```
1A2D SKIP-CONS AND  A                    The subroutine returns if the
1A2E SKIP-NEXT RET  Z                    parameter is zero, or when the requested
                                         constant has been reached.
               PUSH AF                   Save the parameter.
```

```
          PUSH  DE                Save the result pointer.
          LD    DE,+0000          The dummy address.
          CALL  19FE,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    1A2E,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 (part A, pp.23-25).

```
1A3C LOC-MEM   LD    C,A          Copy the parameter to C.
               RLCA               Double the parameter.
               RLCA               Double that result.
               ADD   A,C          Add the value of the parameter to give
                                  five times the original value.
               LD    C.A          This result is wanted in the
               LD    B,+00        BC register pair.
               ADD   HL,BC        Produce the new base address.
               RET                Finished.
```

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'.

```
1A45 get-mem-0 PUSH  DE               Save the result pointer.
     etc.      LD    HL,(MEM)         Fetch the pointer to the current
                                      memory area (see above).
               CALL  1A3C,LOC-MEM     The base address is found.
               CALL  19F6,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 constant 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.

```
1A51 stk-zero  LD    H,D              Set HL to hold the result pointer.
     etc.      LD    L,E
               EXX                    Go to the alternate register set and
               PUSH  HL               save the next literal pointer.
               LD    HL,+TABLE-CON    The base address of the calculator's
                                      table of constants.
```

```
                EXX                             Back to the main set of registers.
                CALL  1A2D,SKIP-CONS            Find the requested base address.
                CALL  19FE,STK-CONST            Expand the constant.
                EXX
                POP   HL                        Restore the next literal pointer.
                EXX
                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.

```
1A63  st-mem-0   PUSH  HL                   Save the result pointer.
        etc.     EX    DE,HL                Source to DE briefly.
                 LD    HL,(MEM)             Fetch the pointer to the current
                                            memory area.
                 CALL  1A3C,LOC-MEM         The base address is found.
                 EX    DE,HL                Exchange source and destination
                                            pointers.
                 CALL  19F6,MOV-FP          The five bytes are moved.
                 EX    DE,HL                'Last value' +5, i.e. STKEND to DE.
                 POP   HL                   Result pointer to HL.
                 RET                        Finished.
```

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.

```
1A72  EXCHANGE  LD    B,+05              There are five bytes involved.
1A74  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  1A74,SWAP-BYTE     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:

$$T_{n+1}(z) = 2zT_n(z) - T_{n-1}(z)\text{ , where } T_n(z) \text{ is the nth Chebyshev polynomial in z.}$$

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 parameter in the A register for use as a counter. The calculator is entered at GEN-ENT-1 so that the counter can be set.

```
1A7F  series-06   LD    B,A                  Move the parameter to B.
      etc.        CALL  19A0,GEN-ENT-1       In effect an RST 0028 instruction
                                             but sets the counter.
```

ii. The handling of the 'last value', Z:
The loop of the generator requires 2*Z to be placed in mem-0, zero to be placed in mem-2 and the 'last value' to be zero.

```
                                             calculator stack
                  DEFB  +2D,duplicate,19F6   Z, Z
                  DEFB  +0F,addition,1755    2*Z
                  DEFB  +C0,st-mem-0,1A63    2*Z          mem-0 holds 2*Z
                  DEFB  +02,delete,19E3      –
                  DEFB  +A0,stk-zero,1A51    0
                  DEFB  +C2,st-mem-2,1A63    0            mem-2 holds 0
```

iii. The main loop:
The series is generated by looping, using BERG 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 BERG; and the series is built up in the form:

B(R) = 2*Z*B(R-1) - B(R-2) + A(R), for R = 1, 2,...,N, where A(1), A(2),..., A(N) are the constants supplied by the calling subroutine (SIN, ATN, LN and EXP) and B(0) = 0 = B(-1).

The (R+1)th loop starts with B(R) on the stack and with 2*Z, B(R-2) and B(R-1) in mem-0, mem-1 and mem-2 respectively.

```
1A89  G-LOOP      DEFB  +2D,duplicate,19F6   B(R),  B(R)
                  DEFB  +E0,get-mem-0,1A45   B(R),  B(R),  2*Z
                  DEFB  +04,multiply,17C6    B(R),  2*B(R)*Z
                  DEFB  +E2,get-mem-2,1A45   B(R),  2*B(R)*Z,  B(R-1)
                  DEFB  +C1,st-mem-1,1A63    mem-1 holds  B(R-1)
                  DEFB  +03,subtract,174C    B(R),  2*B(R)*Z-B(R-1)
                  DEFB  +34,end-calc.,0028
```

The next constant is placed on the calculator stack.

```
                  CALL 19FC,STK-DATA         B(R),  2*B(R)*Z-B(R-1), A(R+1)
```

The Calculator is re-entered without disturbing BERG.

```
              CALL  19A4,GEN-ENT-2
              DEFB  +0F,addition,1755       B(R),  2*B(R)*Z-B(R-1)+A(R+1)
              DEFB  +01,exchange,1A72       2*B(R)*Z-B(R-1)+A(R+1),  B(R)
              DEFB  +C2,st-mem-2,1A63       mem-2 holds B(R)
              DEFB  +02,delete,19E3         2*B(R)*Z-B(R-1)+A(R+1) = B(R+1)
              DEFB  +31,dec-jr-nz,1C17      B(R+1)
              DEFB  +EE, to 1A89,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).

```
              DEFB  +E1,get-mem-1,1A45      B(N),  B(N-2)
              DEFB  +03,subtract,174C       B(N)-B(N-2)
              DEFB  +34,end-calc.,002B
              RET                           Finished
```

THE 'UNARY MINUS' OPERATION (Offset 18: 'negate')

This subroutine performs its unary operation by changing the sign of the 'last value' on the calculator stack.

```
1AA0 negate   LD    A,(HL)                  Fetch the exponent, e.
              AND   A                       Test it.
              RET   Z                       Return if the 'last value' is zero.
              INC   HL                      Point to the sign byte.
              LD    A,(HL)                  Fetch the sign byte.
              XOR   +80                     Change the sign bit.
              LD    (HL),A                  Return the sign byte.
              DEC   HL                      Set the result pointer.
              RET                           Finished.
```

THE 'ABSOLUTE MAGNITUDE' FUNCTION (Offset 27: 'abs')

This subroutine performs its unary operation by ensuring that the sign bit of a floating-point number is reset.

```
1AAA abs      INC   HL                      Point to the sign bit of the 'last value'.
              RES   7,(HL)                  The bit must be reset always.
              DEC   HL                      Set the result pointer.
              RET                           Finished.
```

THE 'SIGNUM' FUNCTION (Offset 26: '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.

```
1AAF sgn      INC   HL                      Point to the sign byte of the present
                                            'last value'.
              LD    A,(HL)                  Fetch the sign byte.
              DEC   HL                      Point to the exponent.
              DEC   (HL)                    Test the exponent byte; the zero
              INC   (HL)                    flag is set for zero.
              SCF                           Set the carry flag.
              CALL  NZ,1AE0,FP-0/1          If the value is not zero then call
                                            FP-0/1 with carry set to give a
                                            'last value' of 1.
```

```
                INC   HL                  Point to the sign byte again.
                RLCA                      The sign bit of X is passed into the
                RR    (HL)                carry, and hence into the result.
                DEC   HL                  Set the result pointer.
                RET                       Finished.
```

THE 'PEEK' FUNCTION (Offset 28: 'peek')

This subroutine handles the function PEEK X. The 'last value' is unstacked by calling FIND-INT. and replaced by the value of the contents of the required location.

```
1ABE  peek      CALL  0EA7,FIND-INT.      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.
                JP    151D,STACK-A        Exit by jumping to STACK-A.
```

THE 'USR' FUNCTION (Offset 29: 'usr')

This subroutine handles the function USR X. The value of X is evaluated, a return address is stacked and the machine code executed from location X.

```
1AC5  usr       CALL  0EA7,FIND-INT.      Evaluate the 'last value', rounded to the
                                          nearest integer; test that it is in range
                                          and return it in BC.
                LD    HL,+STACK-BC        Make the return address be that of the
                PUSH  HL                  subroutine STACK-BC.
                PUSH  BC                  Make an indirect jump to the
                RET                       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 H'L' that holds the next literal pointer is not restored should it have been disturbed.

THE 'GREATER THAN ZERO' OPERATION (Offset 33: 'greater-0')

This subroutine returns a 'last value' of 1 if the present 'last value' is greater than zero and zero otherwise. It is also used by other subroutines to 'jump on plus'.

```
1ACE  GREATER-0 LD    A,(HL)              Fetch the exponent byte.
                AND   A                   Test it.
                RET   Z                   Return if the 'last value' is zero.
                LD    A,+FF               Jump forward to LESS THAN ZERO
                JR    1ADC,SIGN-TO-C      but signal the opposite action is needed.
```

THE 'NOT' FUNCTION (Offset 2C: 'not')

This subroutine returns a 'last value' of 1 if the present 'last value' is zero and zero otherwise. It is also used by other subroutines to 'jump on zero'.

```
1AD5  NOT       LD    A,(HL)              Fetch the exponent byte.
                NEG                       Negating and complementing ensure
                CCF                       that the carry is set only if the 'last value'
                                          is zero; this gives the correct return.
                JR    1AE0,FP-0/1         Jump forward.
```

THE 'LESS THAN ZERO' OPERATION (Offset 32: 'less-0')

This subroutine returns a 'last value' of 1 if the present 'last value' is less than zero and zero otherwise. It is also used by other subroutines to 'jump on minus'.

```
1ADB  less-0      XOR   A              Clear the A register.
1ADC  SIGN-TO-C   INC   HL             Point to the sign byte.
                  XOR   (HL)           The sign bit is collected and stored
                  DEC   HL             in the carry; when entered from
                  RLCA                 GREATER-0 the opposite sign goes to
                                       the carry.
```

THE 'ZERO OR ONE' SUBROUTINE

This subroutine gives the 'last value' as zero if the carry flag is reset and the value 1 if it is set.

```
1AE0  FP-0/1      PUSH  HL             Save the result pointer.
                  LD    B,+05          There are five bytes.
1AE3  FP-ZERO     LD    (HL),+00       Enter zero on each loop.
                  INC   HL             Move to next byte.
                  DJNZ  1AE3,FP-ZERO   Until the five bytes are done.
                  POP   HL             Restore the result pointer.
                  RET   NC             Return the zero if carry reset.
                  LD    (HL),+81       Return 1 if the carry flag is
                  RET                  set.
```

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.

```
1AED  or          LD    A,(DE)         Fetch the exponent of the second
                  AND   A              number; test it and return with the
                  RET   Z              first number as the 'last value' if it
                                       is zero.
                  SCF                  Set the carry flag and jump back to
                  JR    1AE0,FP-0/1    give the 'last value' as 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.

```
1AF3  no.-&-no.   LD    A,(DE)         Fetch the exponent of the second
                  AND   A              number; test it and return with the
                  RET   NZ             first number as the 'last value' if it
                                       is not zero.
                  JR    1AE0,FP-0/1    With the carry flag reset, jump back to
                                       give the 'last value' as zero.
```

THE 'STRING AND NUMBER' OPERATION (Offset 10: 'str-&-no.')

This subroutine performs the binary operation 'A$ AND Y' and returns A$ if Y is non-zero and a null string otherwise.

```
1AF8  str-&-no.   LD    A,(DE)         Fetch the exponent of the number;
                  AND   A              test it and return with the string as the
                  RET   NZ             'last value' if it is not zero.
```

```
                    PUSH DE                    Save the pointer to the number.
                    DEC  DE                    Point to the 5th byte of the string
                                               parameters i.e. length-high.
                    XOR  A                     Clear the A register.
                    LD   (DE),A                Length-high is now set to zero.
                    DEC  DE                    Point to length-low.
                    LD   (DE),A                Length-low is now set to zero.
                    POP  DE                    Restore the pointer.
                    RET                        Return with the string parameters
                                               being the 'last value'.
```

THE 'COMPARISON' OPERATIONS (Offsets 09 to 0E & 11 to 16: 'no.-l-eql', 'no.-gr-eq', 'nos.-neql', 'no.-grtr', 'no.-less', 'nos.-eql', 'str-l-eql', 'str-gr-eql', 'strs-neql', 'str-grtr', 'str-less' & 'strs-eql')

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.

```
1B03 no.-l-eql      LD   A,B                   The single operation offset goes to
     etc.                                      the A register.
                    SUB  +08                   The range is now 01-06 & 09-0E.
                    BIT  2,A                   This range is changed to:
                    JR   NZ,1B0B,EX-OR-NOT     00-02, 04-06, 08-0A &
                    DEC  A                     0C-0E.
1B0B EX-OR-NOT      RRCA                       Then reduced to 00-07 with carry set
                                               for 'greater than or equal to' &
                                               'less than'; the operations with
                    JR   NC,1B16,NU-OR-STR     carry set are then treated as their
                    PUSH AF                    complementary operations once the
                    PUSH HL                    values have been exchanged.
                    CALL 1A72,EXCHANGE
                    POP  DE
                    EX   DE,HL
                    POP  AF
1B16 NU-OR-STR      BIT  2,A                   The numerical comparisons are now
                    JR   NZ,1B21,STRINGS       separated from the string comparisons
                                               by testing bit 2.
                    RRCA                       The numerical operations now have
                                               the range 00-01 with carry set for
                                               'equal' and 'not equal'.
                    PUSH AF                    Save the offset.
                    CALL 174C,SUBTRACT         The numbers are subtracted for the
                    JR   1B54,END-TESTS        final tests.
1B21 STRINGS        RRCA                       The string comparisons now have the
                                               range 02-03 with carry set for 'equal'
                                               and 'not equal'.
                    PUSH AF                    Save the offset.
                    CALL 13F8,STK-FETCH        The lengths and starting addresses
                    PUSH DE                    of the strings are fetched from the
                    PUSH BC                    calculator stack.
                    CALL 13F8,STK-FETCH
                    POP  HL                    The length of the second string.
1B2C BYTE-COMP      LD   A,H
                    OR   L
                    EX   (SP),HL
                    LD   A,B
                    JR   NZ,1B3D,SEC-PLUS      Jump unless the second string is null.
                    OR   C
1B33 SECND-LOW      POP  BC                    Here the second string is either null
                                               or less than the first.
                    JR   Z,1B3A,BOTH-NULL
```

```
                    POP   AF
                    CCF                             The carry is complemented to give
                    JR    1B50,STR-TEST             the correct test results.
1B3A  BOTH-NULL     POP   AF                        Here the carry is used as it
                    JR    1B50,STR-TEST             stands.
1B3D  SEC-PLUS      OR    C
                    JR    Z,1B4D,FRST-LESS          The first string is now null, the
                                                    second not.
                    LD    A,(DE)                    Neither string is null, so their
                    SUB   (HL)                      next bytes are compared.
                    JR    C,1B4D,FRST-LESS          The first byte is less.
                    JR    NZ,1B33,SECND-LOW         The second byte is less.
                    DEC   BC                        The bytes are equal; so the lengths,
                    INC   DE                        are decremented and a jump is
                    INC   HL                        made to BYTE-COMP to compare the
                    EX    (SP),HL                   next bytes of the reduced strings.
                    DEC   HL
                    JR    1B2C,BYTE-COMP
1B4D  FRST-LESS     POP   BC
                    POP   AF
                    AND   A                         The carry is cleared here for the
                                                    correct test results.
1B50  STR-TEST      PUSH  AF                        For the string tests, a zero is
                    RST   0028,FP-CALC.             put on to the calculator stack.
                    DEFB  +A0,stk-zero,1A51
                    DEFB  +34,end-calc.,002B
1B54  END-TESTS     POP   AF                        These three tests, called as needed,
                    PUSH  AF                        give the correct results for all
                    CALL  C,1AD5,NOT                twelve comparisons. The initial
                    CALL  1ACE,GREATER-0            carry is set for 'not equal' and
                    POP   AF                        'equal', and the final carry is set
                    RRCA                            for 'greater than', 'less than' and
                    CALL  NC,1AD5,NOT               'equal'.
                    RET                             Finished.
```

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.

```
1B62  strs-add      CALL  13F8,STK-FETCH            The parameters of the second string
                    PUSH  DE                        are fetched and saved.
                    PUSH  BC
                    CALL  13F8,STK-FETCH            The parameters of the first string
                                                    are fetched.
                    POP   HL
                    PUSH  HL                        The lengths are now in HL and BC.
                    PUSH  DE                        The parameters of the first string
                    PUSH  BC                        are saved.
                    ADD   HL,BC                     The total length of the two strings is
                    LD    B,H                       calculated and passed to BC.
                    LD    C,L
                    RST   0030,BC-SPACES            Sufficient room is made available.
                    CALL  12C3,STK-STORE            The parameters of the new string are
                                                    passed to the calculator stack.
                    POP   BC                        The parameters of the first string are
                    POP   HL                        retrieved and the string copied to
                    LD    A,B                       the work space as long as it is not
                    OR    C                         a null string.
                    JR    Z,1B7D,OTHER-STR
                    LDIR
```

```
1B7D  OTHER-STR POP   BC            Exactly the same procedure is followed
                POP   HL            for the second string thereby
                LD    A,B           giving 'A$+B$'.
                OR    C
                JR    Z,1B85,STK-PNTRS
                LDIR
```

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.

```
1B85  STK-PNTRS LD    HL,(STKEND)       Fetch the current value of STKEND.
                LD    DE,+FFFB          Set DE to -5, 2's complement.
                PUSH  HL                Stack the value for STKEND.
                ADD   HL,DE             Calculate STKEND-5.
                POP   DE                DE now holds STKEND and HL holds
                RET                     STKEND-5.
```

THE 'CHR$' FUNCTION (Offset 2B: 'chrs')

This subroutine handles the function CHR$ X and creates a single character string in the work space.

```
1B8F  chrs      CALL  15CD,FP-TO-A      The 'last value' is compressed into the
                                        A register.
                JR    C,1BA2,REPORT-B2  Give the error report if X was
                                        greater than than 255 decimal, or
                JR    NZ,1BA2,REPORT-B2 X was a negative number.
                PUSH  AF                Save the compressed value of X.
                LD    BC,+0001          Make one space available in the
                RST   0030,BC-SPACES    work space.
                POP   AF                Fetch the value.
                LD    (DE),A            Copy the value to the work space.
                CALL  12C3,STK-STORE    Pass the parameters of the new string
                                        to the calculator stack.
                EX    DE,HL             Reset the pointers.
                RET                     Finished.
```

REPORT-B2 – integer out of range

```
1BA2  REPORT-B2 RST   0008,ERROR-1
                DEFB  +0A
```

THE 'VAL' FUNCTION (Offset 1A: 'val')

This subroutine handles the function VAL A$ and returns a 'last value' that is the result of evaluating the string as an arithmetical expression.

```
1BA4  val       LD    HL,(CH-ADD)       The current value of CH-ADD is
                PUSH  HL                preserved on the machine stack.
                CALL  13F8,STK-FETCH    The parameters of the string are fetched;
                PUSH  DE                the starting address is saved; one byte
                INC   BC                is added to the length and room made
                RST   0030,BC-SPACES    available for the string (+1) in the work
                                        space.
                POP   HL                The starting address of the string goes
                                        to HL as a source address.
```

```
                LD    (CH-ADD),DE         The pointer to the 2nd new space goes
                PUSH  DE                  to CH-ADD and the machine stack.
                LDIR                      The string is copied to the work space,
                                          together with an extra byte.
                EX    DE,HL               Switch the pointers.
                DEC   HL                  The extra byte is replaced by a
                LD    (HL),+76            NEWLINE character.
                RES   7,(FLAGS)           The syntax flag is reset and the string
                CALL  0D92,CLASS-6        scanned for correct syntax.
                CALL  0D22,CHECK-2        A check is made that the end of a line
                                          has been reached.
                POP   HL                  The starting address of the string is
                LD    (CH-ADD),HL         fetched and copied to CH-ADD.
                SET   7,(FLAGS)           The flag is set for line execution.
                CALL  0F55,SCANNING       The string is treated as a 'next expression'
                                          and a 'last value' produced.
                POP   HL                  The original value of CH-ADD is
                LD    (CH-ADD),HL         restored.
                JR    1B85,STK-PNTRS      The subroutine exits via STK-PNTRS
                                          which resets the pointers.
```

THE 'STR$' FUNCTION (Offset 2A: 'strs')

This subroutine handles the function STR$ X and returns a 'last value' which is a set of parameters that defines a string containing what would appear on the screen if X were displayed by a PRINT command.

```
1BD5  strs      LD    BC,+0001            One space is made in the work
                RST   0030,BC-SPACES      space and a NEWLINE character put
                LD    (HL),+76            into the location after it.
                LD    HL,(S-POSN)         The current value of S-POSN is
                PUSH  HL                  preserved on the machine stack.
                LD    L,+FF               The column number of the PRINT
                LD    (S-POSN),HL         position is set to a high value.
                LD    HL,(DF-CC)          The current value of DF-CC is
                PUSH  HL                  preserved on the machine stack.
                LD    (DF-CC),DE          The pointer to the NEWLINE
                PUSH  DE                  character becomes the destination
                                          pointer of the PRINT operation. A copy
                                          is saved on the machine stack.
                CALL  15DB,PRINT-FP       The 'last value', X, is now printed out in
                                          the work space and the work space is
                                          expanded with each character as DF-CC
                                          points to a NEWLINE character.
                POP   DE                  In effect now the start address.
                LD    HL,(DF-CC)          Now the NEWLINE character is one-past
                AND   A                   the end of the string and hence the
                SBC   HL,DE               difference is the length.
                LD    B,H                 Transfer the length to BC.
                LD    C,L
                POP   HL                  Restore the original value of
                LD    (DF-CC),HL          DF-CC.
                POP   HL                  Restore the original value of
                LD    (S-POSN),HL         S-POSN.
                CALL  12C3,STK-STORE      Pass the parameters of the new string to
                                          the calculator stack.
                EX    DE,HL               Reset the pointers.
                RET                       Finished.
```

THE 'CODE' FUNCTION (Offset 19: 'code')

This subroutine handles the function CODE A$ and returns the ZX81 code of the first character in A$, or zero if A$ should be null.

```
1C06  code       CALL  13F8,STK-FETCH     The parameters of the string are fetched.
                 LD    A,B                The length is tested and the A
                 OR    C                  register holding zero is carried forward
                 JR    Z,1C0E,STK-CODE    if A$ is a null string.
                 LD    A,(DE)             The code of the first character is put
                                          into A otherwise.
1C0E  STK-CODE   JP    151D,STACK-A       The subroutine exits via STACK-A
                                          which gives the correct 'last value'.
```

THE 'LEN' FUNCTION (Offset 18: 'len')

This subroutine handles the function LEN A$ and returns a 'last value' that is equal to the length of the string.

```
1C11  len        CALL  13F8,STK-FETCH     The parameters of the string are fetched.
                 JP    1520,STACK-BC      The subroutine exits via STACK-BC
                                          which gives the correct 'last value'.
```

THE 'DECREASE THE COUNTER' SUBROUTINE (Offset 31: '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, BERG, rather than the B register.

```
1C17  dec-jr-nz  EXX                      Go to the alternate register set and
                 PUSH  HL                 save the next literal pointer on the
                                          machine stack.
                 LD    HL,+BERG           Make HL point to BERG.
                 DEC   (HL)               Decrease BERG.
                 POP   HL                 Restore the next literal pointer.
                 JR    NZ,1C24,JUMP-2     The jump is made on non-zero.
                 INC   HL                 The next literal is passed over.
                 EXX                      Return to the main register set.
                 RET                      Finished.
```

THE 'JUMP' SUBROUTINE (Offset 2F: 'jump')

This subroutine executes an unconditional jump when called by the literal '2F'. It is also used by the subroutines DECREASE THE COUNTER and JUMP ON TRUE.

```
1C23  JUMP       EXX                      Go to the alternate register set.
1C24  JUMP-2     LD    E,(HL)             The next literal (jump length) is put
                                          in the E' register.
                 XOR   A                  The A register is cleared.
                 BIT   7,E                If E' is negative, indicating a backwards
                 JR    Z,1C2B,NEW-ADDR.   jump then Hex.FF is formed in the
                 CPL                      A register instead of the Hex.00.
1C2B  NEW-ADDR.  LD    D,A                Hex.00 or Hex.FF goes to D.
                 ADD   HL,DE              The registers H' & L' now hold the
                 EXX                      new next literal pointer.
                 RET                      Finished.
```

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.

```
1C2F  jump-true  LD    A,(DE)              Fetch the exponent.
                 AND   A                   Test it.
                 JR    NZ,1C23,JUMP        Make the jump on true, or more
                                           precisely, on 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 'MODULUS' SUBROUTINE (Offset 2E: 'n-mod-m')

This subroutine calculates N (mod M), where M is a positive integer held at the top of the calculator stack, the 'last value', and N is an 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 by PRINT-FP to reduce N mod 10 decimal, and during the calculation of a random number to reduce N mod 65537 decimal.

```
1C37  n-mod-m    RST   0028,FP-CALC.            N, M
                 DEFB  +C0,st-mem-0,1A63        N, M            mem-0 holds M
                 DEFB  +02,delete,19E3          N
                 DEFB  +2D,duplicate,19F6       N, N
                 DEFB  +E0,get-mem-0,1A45       N, N, M
                 DEFB  +05,division,1882        N, N/M
                 DEFB  +24,int,1C46             N, INT (N/M)
                 DEFB  +E0,get-mem-0,1A45       N, INT (N/M), M
                 DEFB  +01,exchange,1A72        N, M, INT (N/M)
                 DEFB  +C0,st-mem-0,1A63        N, M, INT (N/M) mem-0 holds INT (N/M)
                 DEFB  +04,multiply,17C6        N, M*INT (N/M)
                 DEFB  +03,subtract,174C        N-M*INT (N/M)
                 DEFB  +E0,get-mem-0,1A45       N-M*INT (N/M), INT (N/M)
                 DEFB  +34,end-calc.,002B
                 RET                            Finished.
```

THE 'INT' FUNCTION (Offset 24: '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 18E4 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).

```
1C46  int        RST   0028,FP-CALC.            X
                 DEFB  +2D,duplicate,19F6       X, X
                 DEFB  +32,less-0,1ADB          X, (1/0)
                 DEFB  +00,jump-true,1C2F       X
                 DEFB  +04, to 1C4E,X-NEG       X
```

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.

```
                DEFB  +36,truncate,18E4          I (X)
                DEFB  +34,end-calc.,002B
                RET                              Finished.
```

When X is a negative integer I (X) is returned, otherwise I (X)-1 is returned.

```
1C4E  X-NEG     DEFB  +2D,duplicate,19F6         X, X
                DEFB  +36,truncate,18E4          X, I (X)
                DEFB  +C0,st-mem-0,1A63          X, I (X)        mem-0 holds I (X)
                DEFB  +03,subtract,174C          X-I (X)
                DEFB  +E0,get-mem-0,1A45         X-I (X), I (X)
                DEFB  +01,exchange,1A72          I (X), X-I (X)
                DEFB  +2C,not,1AD5               I (X), (1/0)
                DEFB  +00,jump-true,1C2F         I (X)
                DEFB  +03, to 1C59,EXIT          I (X)
```

The jump is made for values of X that are negative integers, otherwise there is no jump and I (X)-1 is calculated.

```
                DEFB  +A1,stk-one,1A51           I (X), 1
                DEFB  +03,subtract,174C          I (X)-1
```

In either case the subroutine finishes with;

```
1C59  EXIT      DEFB  +34,end-calc.,002B         I (X) or I (X)-1
                RET
```

THE 'EXPONENTIAL' FUNCTION (Offset 23: 'exp')

This subroutine handles the function EXP X and is the first of the 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.
iv. The argument Z is 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.

```
1C5B  EXP       RST   0028,FP-CALC.              X
```

Perform step i.

```
                DEFB  +30,stk-data,19FC          X, 1/LN 2
                DEFB  +F1,exponent 81
                DEFB  +38,+AA,+3B,+29
                DEFB  +04,multiply,17C6          X/LN 2 = Y
```

Perform step ii.

```
                DEFB  +2D,duplicate,19F6         Y, Y
                DEFB  +24,int,1C46               Y, INT Y = N
                DEFB  +C3,st-mem-3, 1A63         Y, N            mem-3 holds N
```

Perform step iii.

```
                DEFB  +03,subtract,174C          Y-N = W
```

Perform step iv.

```
                    DEFB  +2D,duplicate,19F6          W, W
                    DEFB  +0F,addition,1755           2*W
                    DEFB  +A1,stk-one,1A51            2*W, 1
                    DEFB  +03,subtract,174C           2*W-1 = Z
```

Perform step v, passing to the SERIES GENERATOR the parameter '8' and the eight constants required.

```
                    DEFB  +88,series-08,1A7F          Z
              1.    DEFB  +13,exponent 63
                    DEFB  +36,(+00,+00,+00)
              2.    DEFB  +58,exponent 68
                    DEFB  +65,+66,(+00,+00)
              3.    DEFB  +9D,exponent 6D
                    DEFB  +78,+65,+40,(+00)
              4.    DEFB  +A2,exponent 72
                    DEFB  +60,+32,+C9,(+00)
              5.    DEFB  +E7,exponent 77
                    DEFB  +21,+F7,+AF,+24
              6.    DEFB  +EB,exponent 7B
                    DEFB  +2F,+B0,+B0,+14
              7.    DEFB  +EE,exponent 7E
                    DEFB  +7E,+BB,+94,+58
              8.    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-3,1A45          2**W, N
                    DEFB  +34,end-calc.,002B
                    CALL  15CD,FP-TO-A                The absolute value of N mod 256
                                                      decimal, is put into the A register.
                    JR    NZ,1C9B,N-NEGTV             Jump forward if N was negative.
                    JR    C,1C99,REPORT6-2            Error if ABS N greater than 255 dec.
                    ADD   A,(HL)                      Now add ABS N to the exponent.
                    JR    NC,1CA2,RESULT-OK           Jump unless e greater than 255 dec.
1C99  REPORT6-2     RST   0008,ERROR-1                Otherwise report the overflow.
                    DEFB  +05
1C9B  N-NEGTV       JR    C,1CA4,RSLT-ZERO            The result is to be zero if N is less
                                                      than -255 decimal.
                    SUB   (HL)                        Subtract ABS N from the exponent as
                                                      N was negative.
                    JR    NC,1CA4,RSLT-ZERO           Zero result if e less than zero.
                    NEG                               Minus e is changed to e.
1CA2  RESULT-OK     LD    (HL),A                      The exponent, e, is entered.
                    RET                               Finished: 'last value' is EXP X.
1CA4  RSLT-ZERO     RST   0028,FP-CALC.               Use the calculator to make the
                    DEFB  +02,delete,19E3             'last value' zero.
                    DEFB  +A0,stk-zero,1A51
                    DEFB  +34,end-calc.,002B
                    RET                               Finished, with EXP X = 0.
```

THE 'NATURAL LOGARITHM' FUNCTION (Offset 22: '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 simple multiplication and addition leads to LN X being returned as the 'last value'.

```
1CA9  ln        RST   0028,FP-CALC.          X
```

Perform step i.

```
                DEFB  +2D,duplicate,19F6      X, X
                DEFB  +33,greater-0,1ACE      X, (1/0)
                DEFB  +00,jump-true,1C2F      X
                DEFB  +04, to 1CB1,VALID      X
                DEFB  +34,end-calc.,002B      X
                RST   0008,ERROR-1
                DEFB  +09                     Give report A – invalid argument.
```

Perform step ii.

```
1CB1  VALID     DEFB  +A0,stk-zero,1A51       X, 0      The deleted 1 is overwritten
                DEFB  +02,delete,19E3         X         with zero.
                DEFB  +34,end-calc.,002B      X
                LD    A,(HL)                  The exponent, e, goes into A.
                LD    (HL),+80                X is reduced to X'.
                CALL  151D,STACK-A            The stack holds: X', e.
                RST   0028,FP-CALC.           X', e
                DEFB  +30,stk-data,19FC       X', e, 128 (decimal)
                DEFB  +38,exponent 88
                DEFB  +00,(+00,+00,+00)
                DEFB  +03,subtract,174C       X', e'
```

Perform step iii.

```
                DEFB  +01,exchange,1A72       e', X'
                DEFB  +2D,duplicate, 19F6     e', X', X'
                DEFB  +30,stk-data,19FC       e', X', X',0.8 (decimal)
                DEFB  +F0,exponent 80
                DEFB  +4C,+CC,+CC,+CD
                DEFB  +03,subtract,174C       e', X', X'-0.8
                DEFB  +33,greater-0,1ACE      e', X', (1/0)
                DEFB  +00,jump-true,1C2F      e', X'
                DEFB  +08, to 1CD2,GRE.8      e', X'
                DEFB  +01,exchange,1A72       X', e'
                DEFB  +A1,stk-one,1A51        X', e', 1
                DEFB  +03,subtract,174C       X', e'-1
                DEFB  +01,exchange,1A72       e'-1, X'
                DEFB  +34,end-calc.,002B      e'-1, X'
                INC   (HL)                    Double X' to give 2*X'.
                RST   0028,FP-CALC.           e'-1,2*X'
1CD2  GRE.8     DEFB  +01,exchange, 1A72      X', e'          – X' large.
                                              2*X', e'-1      – X' small.
```

```
        DEFB  +30,stk-data,19FC           X', e', LN 2
                                          2*X', e'-1, LN 2
        DEFB  +F0,exponent 80
        DEFB  +31,+72,+17,+F8
        DEFB  +04,multiply,17C6           X', e'*LN 2 = Y1
                                          2*X', (e'-1)*LN 2 = Y2
```

Perform step iv.

```
        DEFB  +01,exchange,1A72           Y1, X'              - X' large.
                                          Y2, 2*X'            - X' small.
        DEFB  +A2,stk-half,1A51           Y1, X', .5 (decimal)
                                          Y2, 2*X', .5
        DEFB  +03,subtract,174C           Y1, X'-.5
                                          Y2, 2*X'-.5
        DEFB  +A2,stk-half,1A51           Y1, X'-.5, .5
                                          Y2, 2*X'-.5, .5
        DEFB  +03,subtract,174C           Y1, X'-1
                                          Y2,2*X'-1
```

Perform step v.

```
        DEFB  +2D,duplicate,19F6          Y, X'-1, X'-1
                                          Y2, 2*X'-1, 2*X'-1
        DEFB  +30,stk-data,19FC           Y1, X'-1, X'-1, 2.5 (decimal)
                                          Y2, 2*X'-1, 2*X'-1, 2.5
        DEFB  +32,exponent 82
        DEFB  +20,(+00,+00,+00)
        DEFB  +04,multiply,17C6           Y1, X'-1, 2.5*X'-2.5
                                          Y2, 2*X'-1, 5*X'-2.5
        DEFB  +A2,stk-half,1A51           Y1, X'-1, 2.5*X'-2.5, .5
                                          Y2, 2*X'-1, 5*X'-2.5, .5
        DEFB  +03,subtract,174C           Y1, X'-1, 2.5*X'-3 = Z
                                          Y2, 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,1A7F          Y1, X'-1, Z or Y2, 2*X'-1, Z
   1.   DEFB  +11,exponent 61
        DEFB  +AC,(+00,+00,+00)
   2.   DEFB  +14,exponent 64
        DEFB  +09,(+00,+00,+00)
   3.   DEFB  +56,exponent 66
        DEFB  +DA,+A5,(+00,+00)
   4.   DEFB  +59,exponent 69
        DEFB  +30,+C5,(+00,+00)
   5.   DEFB  +5C,exponent 6C
        DEFB  +90,+AA,(+00,+00)
   6.   DEFB  +9E,exponent 6E
        DEFB  +70,+6F,+61,(+00)
   7.   DEFB  +A1,exponent 71
        DEFB  +CB,+DA,+96,(+00)
   8.   DEFB  +A4,exponent 74
        DEFB  +31,+9F,+B4,(+00)
   9.   DEFB  +E7,exponent 77
        DEFB  +A0,+FE,+5C,+FC
```

```
10.     DEFB  +EA,exponent 7A
        DEFB  +1B,+43,+CA,+36
11.     DEFB  +ED,exponent 7D
        DEFB  +A7,+9C,+7E,+5E
12.     DEFB  +F0,exponent 80
        DEFB  +6E,+23,+80,+93
```

At the end of the last loop the 'last value' is:

either LN X'/(X'-1) for the larger values of X'
or LN (2*X')/(2*X'-1) for the smaller values of X'.

Perform step vii.

```
        DEFB  +04,multiply,17C6        Y1=LN (2**e'), LN X'
                                       Y2=LN (2**(e'-1)), LN (2*X')
        DEFB  +0F,addition,1755        LN ((2**e')*X')       = LN X
                                       LN(2**(e'-1)*2*X')    = LN X
        DEFB  +34,end-calc.,002B       LN X
        RET                            Finished: 'last value' is LN X.
```

THE 'REDUCE ARGUMENT' SUBROUTINE (Offset 35: '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 = X/(2*PI) – INT (X/(2*PI) + 0.5) , where Y is greater than, or equal to, -.5 but less than +.5.

The subroutine returns with:

```
     V = 4*Y    if  -1<=4*Y<=1   – case i.
or,  V = 2-4*Y  if   1<4*Y<2     – case ii.
or,  V = -4*Y-2 if  -2<=4*Y<-1.  – case iii.
```

In each case, -1<=V<=1 and SIN (PI*V/2) = SIN X.

```
1D18  get-argt.   RST   0028,FP-CALC.          X
                  DEFB  +30,stk-data,19FC      X, 1/(2*PI)
                  DEFB  +EE,exponent 7E
                  DEFB  +22,+F9,+83,+6E
                  DEFB  +04,multiply,17C6      X/(2*PI)
                  DEFB  +2D,duplicate,19F6     X/(2*PI), X/(2*PI)
                  DEFB  +A2,stk-half,1A51      X/(2*PI), X/(2*PI), 0.5
                  DEFB  +0F,addition,1755      X/(2*PI), X/(2*PI)+0.5
                  DEFB  +24,int,1C46           X/(2*PI), INT (X/(2*PI)+0.5)
                  DEFB  +03,subtract,174C      X/(2*PI)-INT (X/(2*PI)+0.5) = Y
```

Note: Adding 0.5 and taking INT rounds the result to the nearest integer.

```
                  DEFB  +2D,duplicate,19F6     Y, Y
                  DEFB  +0F,addition,1755      2*Y
                  DEFB  +2D,duplicate,19F6     2*Y, 2*Y
                  DEFB  +0F,addition,1755      4*Y
                  DEFB  +2D,duplicate,19F6     4*Y, 4*Y
                  DEFB  +27,abs,1AAA           4*Y, ABS (4*Y)
                  DEFB  +A1,stk-one,1A51       4*Y, ABS (4*Y), 1
                  DEFB  +03,subtract,174C      4*Y, ABS (4*Y)-1 = Z
                  DEFB  +2D,duplicate,19F6     4*Y, Z, Z
                  DEFB  +33,greater-0,1ACE     4*Y, Z, (1/0)
                  DEFB  +C0,st-mem-0,1A63      Mem-0 holds the result of the test.
```

```
                  DEFB +00,jump-true,1C2F       4*Y, Z
                  DEFB +04, to 1D35,ZPLUS       4*Y, Z
                  DEFB +02,delete,19E3          4*Y
                  DEFB +34,end-calc.,002B       4*Y = V  -  case i.
                  RET                           Finished.
```

If the jump was made then continue.

```
1D35  ZPLUS       DEFB +A1,stk-one,1A51         4*Y, Z, 1
                  DEFB +03,subtract,174C        4*Y, Z-1
                  DEFB +01,exchange,1A72        Z-1, 4*Y
                  DEFB +32,less-0,1ADB          Z-1, (1/0)
                  DEFB +00,jump-true,1C2F       Z-1
                  DEFB +02, to 1D3C,YNEG        Z-1
                  DEFB +18,negate,1AA0          1-Z
1D3C  YNEG        DEFB +34,end-calc.,002B       1-Z = V  -  case ii.
                                                Z-1 = V  -  case iii.
                  RET                           Finished.
```

THE 'COSINE' FUNCTION (Offset 1D: '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 from 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.

```
1D3E  cos         RST  0028,FP-CALC.            X
                  DEFB +35,get-argt.,1D18       V
                  DEFB +27,abs,1AAA             ABS V
                  DEFB +A1,stk-one,1A51         ABS V, 1
                  DEFB +03,subtract,174C        ABS V-1
                  DEFB +E0,get-mem-0,1A45       ABS V-1, (1/0)
                  DEFB +00,jump-true,1C2F       ABS V-1
                  DEFB +06, to 1D4B, C-ENT      ABS V-1 = W
```

If the jump was not made then continue.

```
                  DEFB +18,negate,1AA0          1-ABS V
                  DEFB +2F,jump,1C23            1-ABS V
                  DEFB +03, to 1D4B,C-ENT       1-ABS V = W
```

THE 'SINE' FUNCTION (Offset 1C: '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<= W<=1, as required for the series to converge.
ii. The argument Z is formed, such that Z=2*W*W-1.
iii. The SERIES GENERATOR is used to return (SIN (PI*W/2))/W.
iv. Finally a simple multiplication gives SIN X.

```
1D49  sin         RST  0028,FP-CALC.            X
```

Perform step i.

```
                    DEFB  +35,get-argt.,1D18        W
```

Perform step ii. The subroutine from now on is common to both the SINE and COSINE functions.

```
1D4B  C-ENT         DEFB  +2D,duplicate,19F6        W, W,
                    DEFB  +2D,duplicate,19F6        W, W, W
                    DEFB  +04,multiply,17C6         W, W*W
                    DEFB  +2D,duplicate,19F6        W, W*W, W*W
                    DEFB  +0F,addition,1755         W, 2*W*W
                    DEFB  +A1,stk-one,1A51          W, 2*W*W, 1
                    DEFB  +03,subtract,174C         W, 2*W*W-1 = Z
```

Perform step iii, passing to the SERIES GENERATOR the parameter '6' and the six constants required.

```
                    DEFB  +86,series-06,1A7F        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  +8F,+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.

```
                    DEFB  +04,multiply,17C6         SIN (PI*W/2) = SIN X (or = COS X)
                    DEFB  +34,end-calc.,002B
                    RET                             Finished: 'last value' = SIN X.
                                                                   or
                                                         ('last value' = COS X ).
```

THE 'TAN' FUNCTION (Offset 1E: 'tan')

This subroutine handles the function TAN X. The subroutine simply returns SIN X/COS X, with arithmetic overflow if COS X=0 .

```
1D6E  tan           RST   0028,FP-CALC.             X
                    DEFB  +2D,duplicate,19F6        X, X
                    DEFB  +1C,sin,1D49              X, SIN X
                    DEFB  +01,exchange,1A72         SIN X, X
                    DEFB  +1D,cos,1D3E              SIN X, COS X
                    DEFB  +05,division,1882         SIN X/COS X = TAN X
                                                    Report arithmetic overflow if needed.
                    DEFB  +34,end-calc.,002B        TAN X
                    RET                             Finished: 'last value' = TAN X.
```

THE 'ARCTAN' FUNCTION (Offset 21: '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       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, as required for the series to 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  =  2/(X*X)-1     – case ii.
if     X<=-1  then  Z = 2*Y*Y-1  =  2/(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.

```
1D76  atn       LD    A,(HL)                  Fetch the exponent of X.
                CP    +81
                JR    C,1D89,SMALL            Jump forward for case i: Y = X.
                RST   0028,FP-CALC.           X
                DEFB  +A1,stk-one,1A51        X, 1
                DEFB  +18,negate,1AA0         X, -1
                DEFB  +01,exchange,1A72       -1, X
                DEFB  +05,division,1882       -1/X
                DEFB  +2D,duplicate,19F6      -1/X, -1/X
                DEFB  +32,less-0,1ADB         -1/X, (1/0)
                DEFB  +A3,stk-pi/2,1A51       -1/X, (1/0), PI/2
                DEFB  +01,exchange,1A72       -1/X, PI/2, (1/0)
                DEFB  +00,jump-true,1C2F      -1/X, PI/2
                DEFB  +06, to 1D8B,CASES      Jump forward for case ii: Y = -1/X
                                                                        W = PI/2
                DEFB  +18,negate,1AA0         -1/X, -PI/2
                DEFB  +2F,jump,1C23           -1/X, -PI/2
                DEFB  +03, to 1D8B,CASES      Jump forward for case iii: Y = -1/X
                                                                         W = -PI/2
1D89  SMALL     RST   0028,FP-CALC.           Y
                DEFB  +A0,stk-zero,1A51       Y, 0
                                              Continue for case i: W = 0
```

Perform step ii.

```
1D8B  CASES     DEFB  +01,exchange,1A72       W, Y
                DEFB  +2D,duplicate,19F6      W, Y, Y
                DEFB  +2D,duplicate,19F6      W, Y, Y, Y
                DEFB  +04,multiply,17C6       W, Y, Y*Y
                DEFB  +2D,duplicate,19F6      W, Y, Y*Y, Y*Y
                DEFB  +0F,addition,1755       W, Y, 2*Y*Y
                DEFB  +A1,stk-one,1A51        W, Y, 2*Y*Y, 1
                DEFB  +03,subtract,174C       W, Y, 2*Y*Y-1 = Z
```

Perform step iii, passing to the SERIES GENERATOR the parameter '12' decimal, and the twelve constants required.

```
                DEFB  +8C,series-0C,1A7F      W, Y, Z
            1.  DEFB  +10,exponent 60
                DEFB  +B2,(+00,+00,+00)
```

```
   2.    DEFB  +13,exponent 63
         DEFB  +0E,(+00,+00,+00)
   3.    DEFB  +55,exponent 65
         DEFB  +E4,+8D,(+00,+00)
   4.    DEFB  +58,exponent 68
         DEFB  +39,+BC,(+00,+00)
   5.    DEFB  +5B,exponent 6B
         DEFB  +98,+FD,(+00,+00)
   6.    DEFB  +9E,exponent 6E
         DEFB  +00,+36,+75,(+00)
   7.    DEFB  +A0,exponent 70
         DEFB  +DB,+E8,+B4,(+00)
   8.    DEFB  +63,exponent 73
         DEFB  +42,+C4,(+00,+00)
   9.    DEFB  +E6,exponent 76
         DEFB  +B5,+09,+36,+BE
  10.    DEFB  +E9,exponent 79
         DEFB  +36,+73,+1B,+5D
  11.    DEFB  +EC,exponent 7C
         DEFB  +D8,+DE,+63,+BE
  12.    DEFB  +F0,exponent 80
         DEFB  +61,+A1,+B3,+0C
```

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.

```
DEFB  +04,multiply,17C6        W, ATN X        – case i.
                               W, ATN (-1/X)   – case ii.
                               W, ATN (-1/X)   – case iii.
DEFB  +0F,addition,1755        ATN X – all cases now.
DEFB  +34,end-calc.,002B
RET                            Finished: 'last value' = ATN X.
```

THE 'ARCSIN' FUNCTION (Offset 1F; 'asn')

This subroutine handles the function ASN X and returns a 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.

```
1DC4  asn    RST   0028,FP-CALC.         X
             DEFB  +2D,duplicate,19F6    X, X
             DEFB  +2D,duplicate,19F6    X, X, X
             DEFB  +04,multiply,17C6     X, X*X
             DEFB  +A1,stk-one,1A51      X, X*X, 1
             DEFB  +03,subtract,174C     X, X*X-1
             DEFB  +18,negate,1AA0       X,1-X*X
             DEFB  +25,sqr,1DDB          X, SQR (1-X*X)
             DEFB  +A1,stk-one,1A51      X, SQR (1-X*X), 1
             DEFB  +0F,addition,1755     X, 1+SQR (1-X*X)
             DEFB  +05,division,1882     X/(1+SQR (1-X*X)) = TAN (Y/2)
```

```
                  DEFB  +21,atn,1D76            Y/2
                  DEFB  +2D,duplicate,19F6      Y/2, Y/2
                  DEFB  +0F,addition,1755       Y = ASN X
                  DEFB  +34,end-calc.,002B
                  RET                           Finished: 'last value' = ASN X.
```

THE 'ARCCOS' FUNCTION (Offset 20: '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:

ACS X = PI/2 - ASN X

```
1DD4  acs         RST   0028,FP-CALC.           X
                  DEFB  +1F,asn,1DC4            ASN X
                  DEFB  +A3,stk-pi/2,1A51       ASN X, PI/2
                  DEFB  +03,subtract,174C       ASN X-PI/2
                  DEFB  +18,negate,1AA0         PI/2-ASN X = ACS X
                  DEFB  +34,end-calc.,002B
                  RET                           Finished: 'last value' = ACS X.
```

THE 'SQUARE ROOT' FUNCTION (Offset 25: '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**.5 and therefore stacks the value .5 and proceeds directly into the EXPONENTIATION subroutine.

```
1DDB  sqr         RST   0028,FP-CALC.           X
                  DEFB  +2D,duplicate,19F6      X, X
                  DEFB  +2C,not,1AD5            X, (1/0)
                  DEFB  +00,jump-true,1C2F      X
                  DEFB  +1E, to 1DFD,LAST       X
```

The jump is made if X = 0, otherwise continue with:

```
                  DEFB  +A2,stk-half,1A51       X, .5
                  DEFB  +34,end-calc.,002B
```

and then find the result of X**.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 (Y*LN 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.

```
1DE2  to-power    RST   0028,FP-CALC.           X, Y
                  DEFB  +01,exchange,1A72       Y, X
                  DEFB  +2D,duplicate,19F6      Y, X, X
                  DEFB  +2C,not,1AD5            Y, X, (1/0)
                  DEFB  +00,jump-true,1C2F      Y, X
                  DEFB  +07, to 1DEE,XIS0       Y, X
```

The jump is made if X = 0, otherwise EXP (Y*LN X) is formed.

```
                DEFB +22,ln,1CA9            Y, LN X
                                            Giving report A if X is negative.
                DEFB +04,multiply,17C6      Y*LN X
                DEFB +34,end-calc.,002B
                JP   1C5B,EXP               Exit via EXP to form EXP (Y*LN X).
```

The value of X is zero so consider the three possible cases involved.

```
1DEE  XISO      DEFB +02,delete,19E3        Y
                DEFB +2D,duplicate,19F6     Y, Y
                DEFB +2C,not,1AD5           Y, (1/0)
                DEFB +00,jump-true,1C2F     Y
                DEFB +09, to 1DFB,ONE       Y
```

The jump is made if X = 0 and Y = 0, otherwise proceed.

```
                DEFB +A0,stk-zero,1A51      Y, 0
                DEFB +01,exchange,1A72      0, Y
                DEFB +33,greater-0,1ACE     0, (1/0)
                DEFB +00,jump-true,1C2F     0
                DEFB +06, to 1DFD,LAST      0
```

The jump is made if X = 0 and Y is positive, otherwise proceed.

```
                DEFB +A1,stk-one,1A51       0, 1
                DEFB +01,exchange,1A72      1, 0
                DEFB +05,division,1882      Exit via 'division' as dividing by zero
                                            gives 'arithmetic overflow'.
```

The result is to be 1 for the operation.

```
1DFB  ONE       DEFB +02,delete,19E3        -
                DEFB +A1,stk-one,1A51       1
```

Now return with the 'last value' on the stack being 0**Y.

```
1DFD  LAST      DEFB +34,end-calc.,002B     (1/0)
                RET                         Finished: 'last value' is 0 or 1.
```

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.

```
5ØØ REM SERIES GENERATOR, ENTER
51Ø REM USING THE COUNTER BERG
52Ø REM AND ARRAY-A HOLDING THE
53Ø REM CONTANTS.
54Ø REM FIRST VALUE IN Z.
55Ø LET MØ=2*Z
56Ø LET M2=Ø
57Ø LET T=Ø
58Ø FOR I=BERG TO 1 STEP -1
59Ø LET M1=M2
6ØØ LET U=T*MØ-M2+A(BERG+1-I)
61Ø LET M2=T
62Ø LET T=U
63Ø NEXT I
64Ø LET T=T-M1
65Ø RETURN
66Ø REM LAST VALUE IN T.
```

In the above subroutine the variable are:

Z	–	the entry value.
T	–	the exit value.
MØ	–	mem-0
M1	–	mem-1
M2	–	mem-2
I	–	the counter for BERG.
U	–	a temporary variable for T.
A(1) to A(BERG)	–	the constants.
BERG	–	the number of constants to be used.

To see how the Chebyshev polynomials are generated, record on paper the values of U, M1, M2 and T through the lines 55Ø to 63Ø, 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 $2*T_5(Z)$, for A(2) it will be $2*T_4(Z)$ and so on to $2*T_1(Z)$ for A(5) and finally $T_0(Z)$ for A(6).

Note that $T_0(Z)=1$, $T_1(Z)=Z$ and, for $n>=2$, $T_n(Z)=2*Z*T_{n-1}(Z)-T_{n-2}(Z)$.

SIN X

```
 10  REM DEMONSTRATION FOR SIN X
 20  SLOW
 30  DIM A(6)
 40  LET A(1)=-.000000003
 50  LET A(2)=0.000000592
 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*W-1
250  LET BERG=6
260  REM USE "SERIES GENERATOR"
270  GOSUB 550
280  PRINT TAB 6;"SIN ";C;" DEGREES"
290  PRINT
300  PRINT T*W,SIN (PI*C/180)
310  PRINT
320  NEXT J
330  GOTO 100
```

NOTES:

i. As it stands the above program requires more than 1K of RAM.

ii. 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.

iii. The constants A(1) to A(6) in lines 40 to 90 are given (apart from a factor of ½) 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 (ie. 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  SLOW
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 BERG=8
280 REM USE "SERIES GENERATOR"
290 GOSUB 550
300 LET V=PEEK 16400+256*PEEK 16401+1    (V=(VARS)+1)
310 LET N=N+PEEK V
320 IF N>255 THEN POKE 16384,5   (Gives report 6, arithmetic overflow;
330 IF N<0 THEN GOTO 360          program stops).
340 POKE V,N
350 GOTO 370
360 LET T=0
370 PRINT TAB 11;"EXP ";C
380 PRINT
390 PRINT T,EXP C
400 PRINT
410 NEXT J
420 GOTO 130
```

NOTES:

i. The above program requires more than 1K of RAM.

ii. 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.

iii. 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.

iv. 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 by 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.

LN X:

```
 10  REM DEMONSTRATION FOR LN X
 20  SLOW
 30  LET D=0                        (This makes D the first variable).
 40  DIM A(12)
 50  LET A(1)=-.0000000003
 60  LET A(2)=0.0000000020
 70  LET A(3)=-.0000000127
 80  LET A(4)=0.0000000823
 90  LET A(5)=-.0000005389
100  LET A(6)=0.0000035828
110  LET A(7)=-.0000243013
120  LET A(8)=0.0001693953
130  LET A(9)=-.0012282837
140  LET A(10)=0.0094766116
150  LET A(11)=-.0818414567
160  LET A(12)=0.9302292213
170  PRINT
180  PRINT "ENTER START VALUE"
190  INPUT C
200  CLS
210  PRINT "BASIC PROGRAM", "ROM PROGRAM"
220  PRINT "------------", "-----------"
230  PRINT
240  LET C=SQR C
250  FOR J=1 TO 4
260  LET C=C*C
270  IF C=0 THEN POKE 16384,9      (Gives report A, invalid argument;
280  LET D=C                        program stops).
290  LET V=PEEK 16400+256*PEEK 16401+1
300  LET N=PEEK V-128               (N holds e').
310  POKE, V,128
320  IF D<=0.8 THEN GOTO 360        (D holds X').
330  LET S=D-1
340  LET Z=2.5*D-3
350  GOTO 390
360  LET N=N-1
370  LET S=2*D-1
380  LET Z=5*D-3
390  LET R=N*0.6931471806           (R holds N*LN 2).
400  LET BERG=12
410  REM USE "SERIES GENERATOR"
420  GOSUB 550
430  PRINT TAB 8;"LN ";C
440  PRINT
450  PRINT S*T+R,LN C
460  PRINT
470  NEXT J
480  GOTO 170
```

NOTES:

i The above program requires more than 1K of RAM.

ii. 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 (for report A).

iii. 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  SLOW
30  DIM A(12)
40  LET A(1)=-.0000000002
50  LET A(2)=0.0000000010
60  LET A(3)=-.0000000066
70  LET A(4)=0.0000000432
80  LET A(5)=-.0000002850
90  LET A(6)=0.0000019105
100 LET A(7)=-.0000131076
110 LET A(8)=0.0000928715
120 LET A(9)=-.0006905975
130 LET A(10)=0.0055679210
140 LET A(11)=-.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 BERG=12
290 REM USE "SERIES GENERATOR"
300 GOSUB 550
310 LET T=D*T
320 IF B>=I 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
370 PRINT                        to obtain the answers in degrees)
380 NEXT J
390 GOTO 160
```

NOTES:

i. The above program requires more than 1K of RAM.

ii. 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.
For those readers who are using an unimproved ROM it is interesting to note the results given by entering 4.2E9 and then 4.3E9.

iii. The constants A(1) to A(12) in lines 40 to 150 are given (apart from a factor of ½) 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 fairly 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*Z-40*Z*Z*Z+10*Z)*A(1)
          +(16*Z*Z*Z*Z-16*Z*Z+2)*A(2)
          +(8*Z*Z*Z-6*Z)*A(3)
          +(4*Z*Z-2)*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:

```
550  LET  T=(128*Z*Z*Z*Z*Z*Z*Z-224*Z*Z*Z*Z*Z+112*Z*Z*Z-14*Z)*A(1)
          +(64*Z*Z*Z*Z*Z*Z-96*Z*Z*Z*Z+36*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-2)*A(6)
          +(2*Z)*A(7)
          +A(8)
560  RETURN
```

It is left as an exercise for the reader to produce the alternative subroutine for LN X and ATN X.

INDEX — Functions, Operations, Subroutines, and Tables.

The book for the programmer that needs those answers about the Timex TS1000/Sinclair ZX81 ROM.

Dr. Logan and Dr. Frank O'Hara have examined all routines in the ROM and comment on each one. This book is a must for the experienced programmer.

Part A covers all functions that can be used except for the floating point calculator.

Part B covers all the routines involved in the 'evaluation of an expression' and a detailed explanation of the 'floating-point calculator'.

MELBOURNE HOUSE PUBLISHERS

ISBN 0 86161 113 6