

The Complete SPECTRUM ROM DISASSEMBLY

BY

Dr Ian Logan & Dr Frank O'Hara

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

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

Preface

The Sinclair ZX Spectrum is a worthy successor to the ZX 81 which in turn replaced the ZX 80.

The Spectrum has a 16K monitor program. This program has been developed directly from the 4K program of the ZX 80 although there are now so many new features that the differences outweigh the similarities.

We have both enjoyed producing this book. We have learnt a great deal about the techniques of Z80 machine code programming and now feel that between us we have unravelled the 'secrets of the Spectrum'.

We would like to thank:

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

January 1983

Ian Logan	Lincoln, U.K.
Frank O'Hara	London, U.K.

Contents

	page
Preface	
Introduction	
The DISASSEMBLY	
- The restart routines and tables	1
- The keyboard routines	5
- The loudspeaker routines	11
- The cassette handling routines	15
- The screen and printer handling routines	33
- The executive routines	59
- BASIC line and command interpretation	84
- Expression evaluation	127
- The arithmetic routines	164
- The floating-point calculator	190
Appendix	
- BASIC programs for the main series (SIN X, EXP X, LN X & ATN X)	222
- The 'DRAW' algorithm	228
- The 'CIRCLE' algorithm	228
- Note on small integers and -65536	229
Index to routines	231

Introduction

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

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

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

The restart routines and tables.

At the start of the monitor program are the various 'restart' routines that are called with the single byte 'RST' instructions. All of the restarts are used. For example 'restart 0008' is used for the reporting of syntax or run-time errors.

The tables in this part of the monitor program hold the expanded forms of the tokens and the 'key-codes'.

The keyboard routine.

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

The loudspeaker routines.

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

The cassette handling routines.

It was a very unfortunate feature of the ZX 81 that so little of the monitor program for that machine was devoted to the cassette handling.

However in the Spectrum there is an extensive block of code and now the high standard of cassette handling is one of the most successful features of the machine.

BASIC programs or blocks of data are both dealt with in the same manner of having a 'header' block (seventeen bytes) that is SAVED first. This 'header' describes the 'data block' that is SAVED after it.

One disadvantage of this system is that it is not possible to produce programs with any 'security' whatsoever.

The screen and printer handling routines.

All of the remaining input/output routines of the Spectrum are 'vectored' through the 'channel & stream information areas'.

In the standard Spectrum 'input' is only possible from the keyboard but 'output' can be directed to the printer, the upper part of the T.V. display or the lower part of the T.V. display.

The major 'input' routine in this part of the monitor program is the EDITOR that allows the user to enter characters into the lower part of the T.V. display.

The PRINT-OUT routine is a rather slow routine as the same routine is used for 'all possibilities'. For example, the adding of a single byte to the 'display area' involves considering the present status of OVER and INVERSE on every occasion.

The executive routines

In this part of the monitor program are to be found the INITIALISATION procedure and the 'main execution loop' of the BASIC interpreter.

In the Spectrum the BASIC line returned by the EDITOR is checked for the correctness of its syntax and then saved in the program area, if it was a line starting with a line number, or 'executed' otherwise.

This execution can in turn lead to further statements being considered. (Most clearly seen as in the case of - RUN.)

BASIC line and command interpretation.

This part of the monitor program considers a BASIC line as a set of statements and in its turn each statement as starting with a particular command. For each command there is a 'command routine' and it is the execution of the machine code in the appropriate 'command routine' that effects the 'interpretation'.

Expression evaluation

The Spectrum has a most comprehensive expression evaluator allowing for a wide range of variable types, functions and operations. Once again this part of the monitor is fairly slow as all the possible alternatives have to be considered.

The handling of strings is particularly well managed. All simple strings are managed 'dynamically' and old copies are 'reclaimed' once they are redundant. This means that there is no 'garbage collecting' to be done.

The arithmetic routines

The Spectrum has two forms for numbers. Integer values in the range -65535 to +65535 are in an 'integral' or 'short' form whilst all other numbers are in a five byte floating point form.

The present version of the monitor is unfortunately marred by two mistakes in this part.

- i. There is a mistake in 'division' whereby the 34th bit of a division is lost.
- ii. The value of -65536 is sometimes put in 'short' form and at other times in 'floating-point' and this leads to troubles.

The floating-point calculator

The CALCULATOR of the Spectrum handles numbers and strings and its operations are specified by 'literals'. It can therefore be considered that there is an internal 'stack operating' language in the CALCULATOR.

This part of the monitor program contains routines for all the mathematical functions. The approximations to SIN X, EXP X, LN X & ATN X are obtained by developing Chebyshev polynomials and full details are given in the appendix.

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

THE DISASSEMBLY

THE RESTART ROUTINES and THE TABLES

THE 'START'

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

0000	START	DI		Disable the 'keyboard interrupt'.
		XOR	A	+00 for start (but +FF for 'NEW').
		LD	DE,+FFFF	Top of possible RAM.
		JP	11CB,START/NEW	Jump forward.

THE 'ERROR' RESTART

The error pointer is made to point to the position of the error.

0008	ERROR-1	LD	HL,(CH-ADD)	The address reached by the
		LD	(X-PTR),HL	interpreter is copied to the error
		JP	0053,ERROR-2	pointer before proceeding.

THE 'PRINT A CHARACTER' RESTART

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

0010	PRINT-A-1	JP	15F2,PRINT-A-2	Jump forward immediately.
		DEFB	+FF,+FF,+FF,+FF,+FF	Unused locations.

THE 'COLLECT CHARACTER' RESTART

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

0018	GET-CHAR	LD	HL,(CH-ADD)	Fetch the value that is addressed
		LD	A,(HL)	by CH-ADD.
001C	TEST-CHAR	CALL	007D,SKIP-OVER	Find out if the character is
		RET	NC	printable. Return if it is so.

THE 'COLLECT NEXT CHARACTER' RESTART

As a BASIC line is interpreted, this routine is called repeatedly to step along the line.

0020	NEXT-CHAR	CALL	0074,CH-ADD+1	CH-ADD needs to be incremented.
		JR	001C,TEST-CHAR	Jump back to test the new value.
		DEFB	+FF,+FF,+FF	Unused locations.

THE 'CALCULATOR' RESTART

The floating point calculator is entered at 335B.

0028	FP-CALC	JP	335B,CALCULATE	Jump forward immediately.
		DEFB	+FF,+FF,+FF,+FF,+FF	Unused locations.

THE 'MAKE BC SPACES' RESTART

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

0030	BC-SPACES	PUSH	BC	Save the 'number'.
		LD	HL,(WORKSP)	Fetch the present address of the
		PUSH	HL	start of the work space and save
		JP	169E,RESERVE	that also before proceeding.

THE 'MASKABLE INTERRUPT' ROUTINE

The real time clock is incremented and the keyboard scanned whenever a maskable interrupt occurs.

0038	MASK-INT	PUSH	AF	Save the current values held in
		PUSH	HL	these registers.
		LD	HL,(FRAMES)	The lower two bytes of the

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

THE 'ERROR-2' ROUTINE

The return address to the interpreter points to the 'DEFB' that signifies which error has occurred. This 'DEFB' is fetched and transferred to ERR-NR. The machine stack is cleared before jumping forward to clear the calculator stack.

0053	ERROR-2	POP	HL	The address on the stack points
		LD	L,(HL)	to the error code.
0055	ERROR-3	LD	(ERR-NR),L	It is transferred to ERR-NR.
		LD	SP,(ERR-SP)	The machine is cleared before
		JP	16C5,SET-STK	exiting via SET-STK.
		DEFB	+FF,+FF,+FF,+FF	Unused locations.
		DEFB	+FF,+FF,+FF	

THE 'NON-MASKABLE INTERRUPT' ROUTINE

This routine is not used in the standard Spectrum but the code allows for a system reset to occur following activation of the NMI line. The system variable at 5CB0, named here NMIADD, has to have the value zero for the reset to occur.

0066	RESET	PUSH	AF	Save the current values held
		PUSH	HL	in these registers.
		LD	HL,(NMIADD)	The two bytes of NMIADD
		LD	A,H	must both be zero for the reset
		OR	L	to occur.
		JR	NZ,0070,NO-RESET	Note: This should have been
				'JR Z'!
		JP	(HL)	Jump to START.
0070	NO-RESET	POP	HL	Restore the current values to
		POP	AF	these registers and return.
		RET		

THE 'CH-ADD+1' SUBROUTINE

The address held in CH-ADD is fetched, incremented and restored. The contents of the location now addressed by CH-ADD is fetched. The entry points of TEMP-PTR1 and TEMP-PTR2 are used to set CH-ADD for a temporary period.

0074	CH-ADD+1	LD	HL,(CH-ADD)	Fetch the address.
0077	TEMP-PTR1	INC	HL	Increment the pointer.
0078	TEMP-PTR2	LD	(CH-ADD),HL	Set CH-ADD.
		LD	A,(HL)	Fetch the addressed value and
		RET		then return.

THE 'SKIP-OVER' SUBROUTINE

The value brought to the subroutine in the A register is tested to see if it is printable. Various special codes lead to HL being incremented once, or twice, and CH-ADD amended accordingly.

007D	SKIP-OVER	CP	+21	Return with the carry flag reset
		RET	NC	if ordinary character code.
		CP	+0D	Return if the end of the line
		RET	Z	has been reached.
		CP	+10	Return with codes +00 to +0F

		RET	C	but with carry set.
		CP	+18	Return with codes +18 to +20
		CCF		again with carry set.
		RET	C	
		INC	HL	Skip-over once.
		CP	+16	Jump forward with codes +10
		JR	C,0090,SKIPS	to +15 (INK to OVER).
		INC	HL	Skip-over once more (AT & TAB).
0090	SKIPS	SCF		Return with the carry flag set
		LD	(CH-ADD),HL	and CH-ADD holding the
		RET		appropriate address.

THE TOKEN TABLE

All the tokens used by the Spectrum are expanded by reference to this table. The last code of each token is 'inverted' by having its bit 7 set.

0095	BF	52	4E	C4	49	4E	4B	45	'?' R N 'D' I N K E
009D	59	A4	50	C9	46	CE	50	4F	Y '\$' P 'I' F 'N' P O
00A5	49	4E	D4	53	43	52	45	45	I N 'T' S C R E E
00AD	4E	A4	41	54	54	D2	41	D4	N '\$' A T T 'R' A 'T'
00B5	54	41	C2	56	41	4C	A4	43	T A 'B' V A L '\$' C
00BD	4F	44	C5	56	41	CC	4C	45	O D 'E' V A 'L' L E
00C5	CE	53	49	CE	43	4F	D3	54	'N' S I 'N' C O 'S' T
00CD	41	CE	41	53	CE	41	43	D3	A 'N' A S 'N' A C 'S'
00D5	41	54	CE	4C	CE	45	58	D0	A T 'N' L 'N' E X 'P'
00DD	49	4E	D4	53	51	D2	53	47	I N 'T' S Q 'R' S G
00E5	CE	41	42	D3	50	45	45	CB	'N' A B 'S' P E E 'K'
00ED	49	CE	55	53	D2	53	54	52	I 'N' U S 'R' S T R
00F5	A4	43	48	52	A4	4E	4F	D4	'\$' C H R '\$' N O 'T'
00FD	42	49	CE	4F	D2	41	4E	C4	B I 'N' O 'R' A N 'D'
0105	3C	BD	3E	BD	3C	BE	4C	49	< '=' > '=' < '>' L I
010D	4E	C5	54	48	45	CE	54	CF	N 'E' T H E 'N' T 'O'
0115	53	54	45	D0	44	45	46	20	S T E 'P' D E F
011D	46	CE	43	41	D4	46	4F	52	F 'N' C A 'T' F O R
0125	4D	41	D4	4D	4F	56	C5	45	M A 'T' M O V 'E' E
012D	52	41	53	C5	4F	50	45	4E	R A S 'E' O P E N
0135	20	A3	43	4C	4F	53	45	20	'#' C L O S E
013D	A3	4D	45	52	47	C5	56	45	'#' M E R G 'E' V E
0145	52	49	46	D9	42	45	45	D0	R I F 'Y' B E E 'P'
014D	43	49	52	43	4C	C5	49	4E	C I R C L 'E' I N
0155	CB	50	41	50	45	D2	46	4C	'K' P A P E 'R' F L
015D	41	53	C8	42	52	49	47	48	A S 'H' B R I G H
0165	D4	49	4E	56	45	52	53	C5	'T' I N V E R S 'E'
016D	4F	56	45	D2	4F	55	D4	4C	O V E 'R' O U 'T' L
0175	50	52	49	4E	D4	4C	4C	49	P R I N 'T' L L I
017D	53	D4	53	54	4F	D0	52	45	S 'T' S T O 'P' R E
0185	41	C4	44	41	54	C1	52	45	A 'D' D A T 'A' R E
018D	53	54	4F	52	C5	4E	45	D7	S T O R 'E' N E 'W'
0195	42	4F	52	44	45	D2	43	4F	B O R D E 'R' C O
019D	4E	54	49	4E	55	C5	44	49	N T I N U 'E' D I
01A5	CD	52	45	CD	46	4F	D2	47	'M' R E 'M' F O 'R' G
01AD	4F	20	54	CF	47	4F	20	53	O T 'O' G O S
01B5	55	C2	49	4E	50	55	D4	4C	U 'B' I N P U 'T' L
01BD	4F	41	C4	4C	49	53	D4	4C	O A 'D' L I S 'T' L
01C5	45	D4	50	41	55	53	C5	4E	E 'T' P A U S 'E' N
01CD	45	58	D4	50	4F	4B	C5	50	E X 'T' P O K 'E' P
01D5	52	49	4E	D4	50	4C	4F	D4	R I N 'T' P L O 'T'
01DD	52	55	CE	53	41	56	C5	52	R U 'N' S A V 'E' R
01E5	41	4E	44	4F	4D	49	5A	C5	A N D O M I Z 'E'
01ED	49	C6	43	4C	D3	44	52	41	I 'F' C L 'S' D R A
01F5	D7	43	4C	45	41	D2	52	45	'W' C L E A 'R' R E
01FD	54	55	52	CE	43	4F	50	D9	T U R 'N' C O P 'Y'

THE KEY TABLES

There are six separate key tables. The final character code obtained depends on the particular key pressed and the 'mode' being used.

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

0205	42	48	59	36	35	54	47	56	B	H	Y	6	5	T	G	V
020D	4E	4A	55	37	34	52	46	43	N	J	U	7	4	R	F	C
0215	4D	4B	49	38	33	45	44	58	M	K	I	8	3	E	D	X
021D	0E	4C	4F	39	32	57	53	5A	SYMBOL	L	0	9	2	W	S	Z
									SHIFT							
0225	20	0D	50	30	31	51	41		SPACE	ENTER	P	0	1	Q	A	

(b) Extended mode. Letter keys and unshifted

022C	E3	C4	E0	E4	READ	BIN	LPRINT	DATA
0230	B4	BC	BD	BB	TAN	SGN	ABS	SQR
0234	AF	B0	B1	C0	CODE	VAL	LEN	USR
0238	A7	A6	BE	AD	PI	INKEY\$	PEEK	TAB
023C	B2	BA	E5	A5	SIN	INT	RESTORE	RND
0240	C2	E1	B3	B9	CHR\$	LLIST	COS	EXP
0244	C1	B8			STR\$	LN		

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

0246	7E	DC	DA	5C	~	BRIGHT	PAPER	\
024A	B7	7B	7D	D8	ATN	{	}	CIRCLE
024E	BF	AE	AA	AB	IN	VAL\$	SCREEN\$	ATTR
0252	DD	DE	DF	7F	INVERSE	OVER	OUT	@
0256	B5	D6	7C	D5	ASN	VERIFY		MERGE
025A	5D	DB	B6	D9]	FLASH	ACS	INK
025E	5B	D7	0C	07	[BEEP		

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

0260	0C	07	06	04	DELETE	EDIT	CAPS LOCK	TRUE VIDEO
0264	05	08	0A	0B	INV VIDEO	Cursor left	Cursor down	Cursor up
0268	09	0F			Cursor right	GRAPHICS		

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

026A	E2	2A	3F	CD	STOP	*	?	STEP
026E	C8	CC	CB	5E	>=	TO	THEN	^
0272	AC	2D	2B	3D	AT	-	+	=
0276	2E	2C	3B	22	.	,	;	"
027A	C7	3C	C3	3E	<=	<	NOT	>
027E	C5	2F	C9	60	OR	/	<>	£
0282	C6	3A			AND	:		

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

0284	D0	CE	A8	CA	FORMAT	DEF FN	FN	LINE
0288	D3	D4	D1	D2	OPEN	CLOSE	MOVE	ERASE
028C	A9	CF			POINT	CAT		

THE KEYBOARD ROUTINES

THE 'KEYBOARD SCANNING' SUBROUTINE

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

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

If no keys is being pressed then the DE register pair is returned holding +FFFF.

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

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

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

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

Four tests are now made.

LD INC RET	A,D A Z	Accept any key value which still has the D register holding +FF. i.e. A single key pressed or 'no-key'.
CP RET	+28 Z	Accept the key value for a pair of keys if the 'D' key is CAPS SHIFT.

CP	+19	Accept the key value for a pair
RET	Z	of keys if the 'D' key is SYMBOL
		SHIFT.
LD	A,E	It is however possible for the 'E'
LD	E,D	key of a pair to be SYMBOL
LD	D,A	SHIFT - so this has to be
CP	+18	considered.
RET		Return with the zero flag set if
		it was SYMBOL SHIFT and
		'another key'; otherwise reset.

THE 'KEYBOARD' SUBROUTINE

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

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

A double system of 'KSTATE system variables' (KSTATE0 - KSTATE 3 and KSTATE4 - KSTATE7) is used from now on.

The two sets allow for the detection of a new key being pressed (using one set) whilst still within the 'repeat period' of the previous key to have been pressed (details in the other set).

A set will only become free to handle a new key if the key is held down for about 1/10 th. of a second. i.e. Five calls to KEYBOARD.

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

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

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

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

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

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

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

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

	BIT	7,(HL)	Consider the second set.
	JR	NZ,02F1,K-NEW	Jump forward if 'free'.
	EX	DE,HL	Now consider the first set.

BIT	7,(HL)	Continue if the set is 'free' but
RET	Z	exit from the KEYBOARD
		subroutine if not.

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

02F1	K-NEW	LD	E,A	The code is passed to the
		LD	(HL),A	E register and to KSTATE0/4.
		INC	HL	The '5 call counter' for this
		LD	(HL),+05	set is reset to '5'.
		INC	HL	The third system variable of
		LD	A,(REPDEL)	the set holds the REPDEL value
		LD	(HL),A	(normally 0.7 secs.).
		INC	HL	Point to KSTATE3/7.

The decoding of a 'main code' depends upon the present state of MODE, bit 3 of FLAGS and the 'shift byte'.

LD	C,(MODE)	Fetch MODE.
LD	D,(FLAGS)	Fetch FLAGS.
PUSH	HL	Save the pointer whilst the
CALL	0333,K-DECODE	'main code' is decoded.
POP	HL	
LD	(HL),A	The final code value is saved in
		KSTATE3/7; from where it is
		collected in case of a repeat.

The next three instruction lines are common to the handling of both 'new keys' and 'repeat keys'.

0308	K-END	LD	(LAST-K),A	Enter the final code value into
		SET	5,(FLAGS)	LAST-K and signal 'a new key'.
		RET		Finally return.

THE 'REPEATING KEY' SUBROUTINE

A key will 'repeat' on the first occasion after the delay period - REPDEL (normally 0.7 secs.) and on subsequent occasions after the delay period - REPPER (normally 0.1 secs.).

0310	K-REPEAT	INC	HL	Point to the '5 call counter'
		LD	(HL),+05	of the set being used and reset
				it to '5'.
		INC	HL	Point to the third system vari-
		DEC	(HL)	able - the REPDEL/REPPER
				value, and decrement it.
		RET	NZ	Exit from the KEYBOARD
				subroutine if the delay period
				has not passed.
		LD	A,(REPPER)	However once it has passed the
		LD	(HL),A	delay period for the next repeat
				is to be REPPER.
		INC	HL	The repeat has been accepted
		LD	A,(HL)	so the final code value is fetched
				from KSTATE3/7 and passed
		JR	0308,K-END	to K-END.

THE 'K-TEST' SUBROUTINE

The key value is tested and a return made if 'no-key' or 'shift-only'; otherwise the 'main code' for that key is found.

031E	K-TEST	LD	B,D	Copy the shift byte.
		LD	D,+00	Clear the D register for later.
		LD	A,E	Move the key number.
		CP	+27	Return now if the key was
		RET	NC	'CAPS SHIFT' only or 'no-key'.

CP	+18	Jump forward unless the 'E' key was SYMBOL SHIFT.
JR	NZ,032C,K-MAIN	However accept SYMBOL SHIFT and another key; return with SYMBOL SHIFT only.
BIT	7,B	
RET	NZ	

The 'main code' is found by indexing into the main key table.

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

THE 'KEYBOARD DECODING' SUBROUTINE

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

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

0333	K-DECODE	LD	A,E	Copy the 'main code'.
		CP	+3A	Jump forward if a digit key is being considered; also SPACE, ENTER & both shifts.
		JR	C,0367,K-DIGIT	Decrement the MODE value.
		DEC	C	Jump forward, as needed, for modes 'K', 'L', 'C' & 'E'.
		JP	M,034F,K-KLC-LET	
		JR	Z,0341,K-E-LET	

Only 'graphics' mode remains and the 'final code' for letter keys in graphics mode is computed from the 'main code'.

ADD	A,+4F	Add the offset.
RET		Return with the 'final code'.

Letter keys in extended mode are considered next.

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

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

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

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

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

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

0364	K-TOKENS	ADD RET	A,+A5	Add the required offset and return.
Next the digit keys; and SPACE, ENTER & both shifts; are considered.				
0367	K-DIGIT	CP RET	+30 C	Proceed only with the digit keys. i.e. Return with SPACE (+20), ENTER (+0D) & both shifts (+0E).
		DEC	C	Now separate the digit keys into three groups - according to the mode.
		JP JR	M,039D,K-KLC-DGT NZ,0389,K-GRA-DGT	Jump with 'K', 'L' & 'C' modes; and also with 'G' mode. Continue with 'E' mode.
		LD BIT JR	HL,+0254 5,B Z,034A,K-LOOK-UP	The base address for table 'f'. Use this table for SYMBOL SHIFT & a digit key in extended mode.
		CP JR	+38 NC,0382,K-8-&-9	Jump forward with digit keys '8' and '9'.

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

SUB	+20	Reduce the range +30 to +37 giving +10 to +17.
INC RET	B Z	Return with this 'paper colour code' if the CAPS SHIFT is not being used.
ADD RET	A,+08	But if it is then the range is to be +18 to +1F instead - indicat- ing an 'ink colour code'.

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

0382	K-8-&-9	SUB INC RET	+36 B Z	+38 & +39 go to +02 & +03. Return with these codes if CAPS SHIFT is not being used. (These are 'BRIGHT' codes.)
		ADD RET	A,+FE	Subtract '2' is CAPS SHIFT is being used; giving +00 & +01 (as 'FLASH' codes).

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

0389	K-GRA-DGT	LD CP JR CP JR AND ADD INC RET	HL,+0230 +39 Z,034A,K-LOOK-UP +30 Z,034A,K-LOOK-UP +07 A,+80 B Z	The base address of table 'd'. Use this table directly for both digit key '9' that is to give GRAPHICS, and digit key '0' that is to give DELETE. For keys '1' to '8' make the range +80 to +87. Return with a value from this range if neither shift key is being pressed.
		XOR RET	+0F	But if 'shifted' make the range +88 to +8F.

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

039D	K-KLC-DGT	INC RET	B Z	Return directly if neither shift key is being used. (Final codes +30 to +39.)
		BIT LD JR	5,B HL,+0230 NZ,034A,K-LOOK-UP	Use table 'd' if the CAPS SHIFT key is also being pressed.

The codes for the various digit keys and SYMBOL SHIFT can now be found.

		SUB	+10	Reduce the range to give +20 to +29.
		CP	+22	Separate the '@' character from the others.
		JR	Z,03B2,K-@-CHAR	The '-' character has also to be separated.
		CP	+20	
		RET	NZ	Return now with the 'final codes' +21, +23 to +29.
		LD	A,+5F	Give the '-' character a code of +5F.
03B2	K-@-CHAR	RET		
		LD	A,+40	Give the '@' character a code of +40.
		RET		

THE LOUDSPEAKER ROUTINES

The two subroutines in this section are the BEEPER subroutine, that actually controls the loudspeaker, and the BEEP command routine.

The loudspeaker is activated by having D4 low during an OUT instruction that is using port '254'. When D4 is high in a similar situation the loudspeaker is deactivated. A 'beep' can therefore be produced by regularly changing the level of D4.

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

THE 'BEEPER' SUBROUTINE

This subroutine is entered with the DE register pair holding the value 'f*t', where a note of given frequency 'f' is to have a duration of 't' seconds, and the HL register pair holding a value equal to the number of T states in the 'timing loop' divided by '4'.

i.e. For the note 'middle C' to be produced for one second DE holds +0105 (INT(261.3 * 1)) and HL holds +066A (derived from 6,689/4 - 30.125).

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

Now enter the sound generation loop. 'DE' complete passes are made, i.e. a pass for each cycle of the note.

The HL register holds the 'length of the timing loop' with '16' T states being used for each '1' in the L register and '1,024' T states for each '1' in the H register.

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

The loudspeaker is now alternately activated and deactivated.

XOR	+10	Flip bit 4.
-----	-----	-------------

OUT	(+FE),A	Perform the OUT operation; leaving the border unchanged.
LD	B,H	Reset the B register.
LD	C,A	Save the A register.
BIT	4,A	Jump if at the half-cycle
JR	NZ,03F2,BE-AGAIN	point.

After a full cycle the DE register pair is tested.

LD	A,D	Jump forward if the last
OR	E	complete pass has been
JR	Z,03D6,BE-END	made already.
LD	A,C	Fetch the saved value.
LD	C,L	Reset the C register.
DEC	DE	Decrease the pass counter.
JP	(IX)	Jump back to the required starting location of the loop.

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

03F2	BE-AGAIN	LD	C,L	Reset the C register.
		INC	C	Add '16' T states as this path is shorter.
		JP	(IX)	Jump back.

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

03F6	BE-END	EI	Enable interrupt.
		RET	Finally return.

THE 'BEEP' COMMAND ROUTINE

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

03F8	BEEP	RST	0028,FP-CALC	The floating-point calculator is used to manipulate the two values - t & P.
		DEFB	+31,duplicate	t,P,P
		DEFB	+27,int	t,P,I(where i = INT P)
		DEFB	+C0,st-mem-0	t,P,i (mem-0 holds i)
		DEFB	+03,subtract	t,P (where p is the fractional part of P)
		DEFB	+34,stk-data	Stack the decimal value 'K'.
		DEFB	+EC,exponent+7C	0.0577622606 (which is a little below $12\sqrt{2}-1$)
		DEFB	+6C,+98,+1F,+F5	t,pK
		DEFB	+04,multiply	t,pK,1
		DEFB	+A1,stk-one	t,pK+1
		DEFB	+0F,addition	
		DEFB	+38,end-calc	

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

LD	HL,+5C92	This is 'mem-0-1st' (MEMBOT).
LD	A,(HL)	Fetch the exponent of i.
AND	A	Give an error if i is not in the
JR	NZ,046C,REPORT-B	integral (short) form.
INC	HL	Copy the sign byte to the
LD	C,(HL)	C register.
INC	HL	Copy the low-byte to the
LD	B,(HL)	B register; and to the A
LD	A,B	register.
RLA		Again give report B if i does not
SBC	A,A	satisfy the test:
CP	C	-128<=i<=+127
JR	NZ,046C,REPORT-B	
INC	HL	
CP	(HL)	

JR	NZ,046C,REPORT-B	
LD	A,B	Fetch the low-byte and test it further.
ADD	A,+3C	
JP	P,0425,BE-i-OK	Accept -60<=i<=67.
JP	PO,046C,REPORT-B	Reject -128 to -61.

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

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

0425	BE-i-OK	LD	B,+FA	Start '6' octaves below middle C. Repeatedly reduce i in order to find the correct octave.
0427	BE-OCTAVE	INC	B	
		SUB	+0C	
		JR	NC,0427,BE-OCTAVE	
		ADD	A,+0C	Ass back the last subtraction.
		PUSH	BC	Save the octave number.
		LD	HL,+046E	The base address of the 'semi-tone table'.
		CALL	3406,LOC-MEM	Consider the table and pass the 'A th.' value to the calculator stack. (Call it C.)
		CALL	33B4,STACK-NUM	

Now the fractional part of the 'pitch' can be taken into consideration.

RST	0028,FP-CALC	t, pK+1, C
DEFB	+04,multiply	t, C(pK+1)
DEFB	+38,end-calc	

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

POP	AF	Fetch the octave number.
ADD	A,(HL)	Multiply the 'last value' by
LD	(HL),A	'2 to the power of the octave number'.
RST	0028,FP-CALC	t, f
DEFB	+C0,st-mem-0	The frequency is put aside for the moment in mem-0.
DEFB	+02,delete	

Attention is now turned to the 'duration'.

DEFB	+31,duplicate	t, t
DEFB	+38,end-calc	
CALL	1E94,FIND-INT1	The value 'INT t' must be in the range +00 to +0A.
CP	+0B	
JR	NC,046C,REPORT-B	

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

RST	0028,FP-CALC	t
DEFB	+E0,get-mem-0	t, f
DEFB	+04,multiply	f*t

The result is left on the calculator stack whilst the length of the 'timing loop' required for the 'beep' is computed;

DEFB	+E0,get-mem-0	f*t, f
DEFB	+34,stk-data	The value '3.5 * 10 ⁶ /8'
DEFB	+80,four bytes	is formed on the top of the calculator stack.
DEFB	+43,exponent +93	f*t, f, 437,500 (dec.)
DEFB	+55,+9F,+80,(+00)	f*t, 437,500, f
DEFB	+01,exchange	f*t, 437,500/f
DEFB	+05,division	
DEFB	+34,stk-data	
DEFB	+35,exponent +85	
DEFB	+71,(+00,+00,+00)	f*t, 437,500/f, 30.125 (dec.)
DEFB	+03,subtract	f*t, 437,500/f - 30.125
DEFB	+38,end-calc	

Note: The value '437,500/f' gives the 'half-cycle' length of the note and reducing it by '30.125' allows for '120.5' T states in which to actually produce the note and adjust the counters etc.
The values can now be transferred to the required registers.

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

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

CALL	1E99,FIND-INT2	The 'f*t' value is compressed into the BC register pair.
POP	HL	Move the 'timing loop' value to the HL register pair.
LD	D,B	Move the 'f*t' value to the DE register pair.
LD	E,C	

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

LD	A,D	Return if 'f*t' has given the result of 'no cycles' required.
OR	E	
RET	Z	
DEC	DE	Decrease the cycle number and jump to the BEEPER subroutine (making, at least, one pass).
JP	03B5,BEEPER	

Report B - integer out of range

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

THE 'SEMI-TONE' TABLE

This table holds the frequencies of the twelve semi-tones in an octave.

			frequency hz.	note
046E	DEFB	+89,+02,+D0,+12,+86	261.63	C
	DEFB	+89,+0A,+97,+60,+75	277.18	C#
	DEFB	+89,+12,+D5,+17,+1F	293.66	D
	DEFB	+89,+1B,+90,+41,+02	311.12	D#
	DEFB	+89,+24,+D0,+53,+CA	329.63	E
	DEFB	+89,+2E,+9D,+36,+B1	349.23	F
	DEFB	+89,+38,+FF,+49,+3E	369.99	F#
	DEFB	+89,+43,+FF,+6A,+73	392	G
	DEFB	+89,+4F,+A7,+00,+54	415.30	G#
	DEFB	+89,+5C,+00,+00,+00	440	A
	DEFB	+89,+69,+14,+F6,+24	466.16	A#
	DEFB	+89,+76,+F1,+10,+05	493.88	B

THE 'PROGRAM NAME' SUBROUTINE (ZX81)

The following subroutine applies to the ZX81 and was not removed when the program was rewritten for the SPECTRUM.

04AA	DEFB	+CD,+FB,+24,+3A
	DEFB	+3B,+5C,+87,+FA
	DEFB	+8A,+1C,+E1,+D0
	DEFB	+E5,+CD,+F1,+2B
	DEFB	+62,+6B,+0D,+F8
	DEFB	+09,+CB,+FE,+C9

THE CASSETTE HANDLING ROUTINES

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

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

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

THE 'SA-BYTES' SUBROUTINE

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

04C2	SA-BYTES	LD	HL,+053F	Pre-load the machine stack with the address - SA/LD-RET. This constant will give a leader of about 5 secs. for a 'header'. Jump forward if SAVEing a header. This constant will give a leader of about 2 secs. for a program/data block.
		PUSH	HL	
		LD	HL,+1F80	
		BIT	7,A	
		JR	Z,04D0,SA-FLAG	
04D0	SA-FLAG	LD	HL,+0C98	The flag is saved. The 'length' is incremented and the 'base address' reduced to allow for the flag. The maskable interrupt is disabled during the SAVE. Signal 'MIC on' and border to be RED. Give a value to B.
		EX	AF,A'F'	
		INC	DE	
		DEC	IX	
		DI		
		LD	A,+02	
		LD	B,A	

A loop is now entered to create the pulses of the leader. Both the 'MIC on' and the 'MIC off' pulses are 2,168 T states in length. The colour of the border changes from RED to CYAN with each 'edge'.

Note: An 'edge' will be a transition either from 'on' to 'off', or from 'off' to 'on'.

04D8	SA-LEADER	DJNZ	04D8,SA-LEADER	The main timing period. MIC on/off, border RED/CYAN, on each pass. The main timing constant. Decrease the low counter. Jump back for another pulse. Allow for the longer path (-reduce by 13 T states). Decrease the high counter. Jump back for another pulse until completion of the leader.
		OUT	(+FE),A	
		XOR	+0F	
		LD	B,+A4	
		DEC	L	
		JR	NZ,04D8,SA-LEADER	
		DEC	B	
		DEC	H	
		JP	P,04D8,SA-LEADER	

A sync pulse is now sent.

04EA	SA-SYNC-1	LD	B,+2F	MIC off for 667 T states from 'OUT to OUT'. MIC on and RED. Signal 'MIC off & CYAN'. MIC on for 735 T States from 'OUT to OUT'. Now MIC off & border CYAN.
		DJNZ	04EA,SA-SYNC-1	
		OUT	(+FE),A	
		LD	A,+0D	
04F2	SA-SYNC-2	LD	B,+37	
		DJNZ	04F2,SA-SYNC-2	
		OUT	(+FE),A	

The header v. program/data flag will be the first byte to be SAVED.

LD	BC,+3B0E	+3B is a timing constant; +0E signals 'MIC off & YELLOW'.
EX	AF,A'F'	Fetch the flag and pass it to the L register for 'sending'.
LD	L,A	
JP	0507,SA-START	Jump forward into the SAVEing loop.

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

04FE	SA-LOOP	LD OR JR LD	A,D E Z,050E,SA-PARITY L,(IX+00)	The 'length' counter is tested and the jump taken when it has reached zero. Fetch the next byte that is to be SAVED.
0505	SA-LOOP-P	LD	A,H	Fetch the current 'parity'.
		XOR	L	Include the present byte.
0507	SA-START	LD	H,A	Restore the 'parity'. Note that on entry here the 'flag' value initialises 'parity'.
		LD SCF	A,+01	Signal 'MIC on & BLUE'. Set the carry flag. This will act as a 'marker' for the 8 bits of a byte.
		JP	0525,SA-8-BITS	Jump forward.

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

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

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

0511	SA-BIT-2	LD	A,C	Come here on the second pass and fetch 'MIC off & YELLOW'.
		BIT	7,B	Set the zero flag to show 'second pass'.
0514	SA-BIT-1	DJNZ	0514,SA-BIT-1	The main timing loop; always 801 T states on a 2nd. pass.
		JR	NC,051C,SA-OUT	Jump, taking the shorter path, if SAVEing a '0'.
051A	SA-SET	LD	B,+42	However if SAVEing a '1' then add 855 T states.
051C	SA-OUT	DJNZ	051A,SA-SET	On the 1st. pass 'MIC on & BLUE' and on the 2nd. pass 'MIC off & YELLOW'.
		OUT	(+FE),A	
		LD	B,+3E	Set the timing constant for the second pass.
		JR	NZ,0511,SA-BIT-2	Jump back at the end of the first pass; otherwise reclaim 13 T states.
		DEC	B	
		XOR	A	Clear the carry flag and set A to hold +01 (MIC on & BLUE) before continuing into the '8 bit loop'.
		INC	A	

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

0525	SA-8-BITS	RL	L	Move bit 7 to the carry and the 'marker' leftwards.
------	-----------	----	---	---

		JP	NZ,0514,SA-BIT-1	SAVE the bit unless finished with the byte.
		DEC	DE	Decrease the 'counter'.
		INC	IX	Advance the 'base address'.
		LD	B,+31	Set the timing constant for the first bit of the next byte.
		LD	A,+7F	Return (to SA/LD-RET) if the BREAK key is being pressed.
		IN	A,(+FE)	
		RRA		
		RET	NC	
		LD	A,D	Otherwise test the 'counter' and jump back even if it has reached zero (so as to send the 'parity' byte).
		INC	A	
		JP	NZ,04FE,SA-LOOP	
053C	SA-DELAY	LD	B,+3B	Exit when the 'counter' reaches +FFFF. But first give a short delay.
		DJNZ	053C,SA-DELAY	
		RET		

Note: A reset bit will give a 'MIC off' pulse of 855 T states followed by a 'MIC on' pulse of 855 T states. Whereas a Set bit will give pulses of exactly twice as long. Note also that there are no gaps either between the sync pulse and the first bit of the flag, or between bytes.

THE 'SA/LD-RET' SUBROUTINE

This subroutine is common to both SAVEing and LOADING.

The border is set to its original colour and the BREAK key tested for a last time.

053F	SA/LD-RET	PUSH	AF	Save the carry flag. (It is reset after a LOADING error.)
		LD	A,(BORDCR)	Fetch the original border colour from its system variable.
		AND	+38	Move the border colour to bits 2, 1 & 0.
		RRCA		Set the border to its original colour.
		RRCA		
		OUT	(+FE),A	Read the BREAK key for a last time.
		LD	A,+7F	
		IN	A,(+FE)	
		RRA		
		EI		Enable the maskable interrupt.
		JR	C,0554,SA/LD-END	Jump unless a break is to be made.

Report D - BREAK-CONT repeats

0552	REPORT-D	RST	0008,ERROR-I	Call the error handling routine.
		DEFB	+0C	

Continue here.

0554	SA/LD-END	POP	AF	Retrieve the carry flag.
		RET		Return to the calling routine.

THE 'LD-BYTES' SUBROUTINE

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

0556	LD-BYTES	INC	D	This resets the zero flag. (D cannot hold +FF.)
		EX	AF,A'F'	The A register holds +00 for a header and +FF for a block of data.
				The carry flag is reset for VERIFYing and set for LOADING.
		DEC	D	Restore D to its original value.

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

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

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

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

0574	LD-WAIT	LD DJNZ DEC LD OR JR CALL JR	HL,+0415 0574,LD-WAIT HL A,H L NZ,0574,LD-WAIT 05E3,LD-EDGE-2 NC,056B,LD-BREAK	The length of this waiting period will be almost one second in duration.
				Continue only if two edges are found within the allowed time period.

Now accept only a 'leader signal'.

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

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

058F	LD-SYNC	LD CALL JR LD CP JR CALL RET	B,+C9 05E7,LD-EDGE-1 NC,056B,LD-BREAK A,B +D4 NC,058F,LD-SYNC 05E7,LD-EDGE-1 NC	The timing constant. Every edge is considered until two edges are found close together - these will be the start and finishing edges of the 'off' sync pulse. The finishing edge of the 'on' pulse must exist. (Return carry flag reset.)
------	---------	---	--	---

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

LD	A,C	The border colours from now
XOR	+03	on will be BLUE & YELLOW.

LD	C,A	
LD	H,+00	Initialise the 'parity matching' byte to zero.
LD	B,+B0	Set the timing constant for the flag byte.
JR	05C8,LD-MARKER	Jump forward into the byte LOADING loop.

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

05A9	LD-LOOP	EX	AF,A'F'	Fetch the flags.
		JR	NZ,05B3,LD-FLAG	Jump forward only when handling the first byte.
		JR	NC,05BD,LD-VERIFY	Jump forward if VERIFYing a tape.
		LD	(IX+00),L	Make the actual LOAD when required.
		JR	05C2,LD-NEXT	Jump forward to LOAD the next byte.
05B3	LD-FLAG	RL	C	Keep the carry flag in a safe place temporarily.
		XOR	L	Return now if the type flag does not match the first byte on the tape. (Carry flag reset.)
		RET	NZ	Restore the carry flag now.
		LD	A,C	
		RRA		
		LD	C,A	
		INC	DE	Increase the counter to compensate for its 'decrease' after the jump.
		JR	05CA,LD-DEC	

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

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

A new byte can now be collected from the tape.

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

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

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

The 'parity matching' byte has to be updated with each new byte.

	LD	A,H	Fetch the 'parity matching'
	XOR	L	byte and include the new byte.
	LD	H,A	Save it once again.
Passes round the loop are made until the 'counter' reaches zero. At that point the 'parity matching' byte should be holding zero.			
	LD	A,D	Make a further pass if the DE
	OR	E	register pair does not hold
	JR	NZ,05A9,LD-LOOP	zero.
	LD	A,H	Fetch the 'parity matching'
			byte.
	CP	+01	Return with the carry flat set
	RET		if the value is zero.
			(Carry flag reset if in error.)

THE 'LD-EDGE-2' AND 'LD-EDGE-1' SUBROUTINES

These two subroutines form the most important part of the LOAD/VERIFY operation.

The subroutines are entered with a timing constant in the B register, and the previous border colour and 'edge-type' in the C register.

The subroutines return with the carry flag set if the required number of 'edges' have been found in the time allowed; and the change to the value in the B register shows just how long it took to find the 'edge(s)'.

The carry flag will be reset if there is an error. The zero flag then signals 'BREAK pressed' by being reset, or 'time-up' by being set.

The entry point LD-EDGE-2 is used when the length of a complete pulse is required and LD-EDGE-1 is used to find the time before the next 'edge'.

05E3	LD-EDGE-2	CALL	05E7,LD-EDGE-1	In effect call LD-EDGE-1 twice;
		RET	NC	returning in between if there
				is an error.
05E7	LD-EDGE-1	LD	A,+16	Wait 358 T states before
05E9	LD-DELAY	DEC	A	entering the sampling loop.
		JR	NZ,05E9,LD-DELAY	
		AND	A	

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

05ED	LD-SAMPLE	INC	B	Count each pass.
		RET	Z	Return carry reset & zero set if
				'time-up'.
		LD	A,+7F	Read from port +7FFE.
		IN	A,(+FE)	i.e. BREAK & EAR.
		RRA		Shift the byte.
		RET	NC	Return carry reset & zero reset
				if BREAK was pressed.
		XOR	C	Now test the byte against the
		AND	+20	'last edge-type'; jump back
		JR	Z,05ED,LD-SAMPLE	unless it has changed.

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

	LD	A,C	Change the 'last edge-type'
	CPL		and border colour.
	LD	C,A	
	AND	+07	Keep only the border colour.
	OR	+08	Signal 'MIC off'.
	OUT	(+FE),A	Change the border colour (RED/
			CYAN or BLUE/YELLOW).
	SCF		Signal the successful search
	RET		before returning.

Note: The LD-EDGE-1 subroutine takes 465 T states, plus an additional 58 T states for each unsuccessful pass around the sampling loop.

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

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

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

0605	SAVE-ETC	POP	AF	Drop the address - SCAN-LOOP.
		LD	A,(T-ADDR-lo)	Reduce T-ADDR-lo by +E0;
		SUB	+E0	giving +00 for SAVE, +01 for
		LD	(T-ADDR-lo),A	LOAD, +02 for VERIFY and
				+03 for MERGE.
		CALL	1C8C,EXPT-EXP	Pass the parameters of the
				'name' to the calculator stack.
		CALL	2530,SYNTAX-Z	Jump forward if checking
		JR	Z,0652,SA-DATA	syntax.
		LD	BC,+0011	Allow seventeen locations
0621	SA-SPACE	LD	A,(T-ADDR-lo)	for the header of a SAVE but
		AND	A	thirty four for the other
		JR	Z,0621,SA-SPACE	commands.
		LD	C,+22	
		RST	0030,BC-SPACES	The required amount of space is
				made in the work space.
		PUSH	DE	Copy the start address to the
		POP	IX	IX register pair.
		LD	B,+0B	A program name can have
		LD	A,+20	up to ten characters but
0629	SA-BLANK	LD	(DE),A	first enter eleven space
		INC	DE	characters into the prepared
		DJNZ	0629,SA-BLANK	area.
		LD	(IX+01),+FF	A null name is +FF only.
		CALL	2BF1,STK-FETCH	The parameters of the name
				are fetched and its length is
				tested.
		LD	HL,+FFF6	This is '-10'.
		DEC	BC	In effect jump forward if the
		ADD	HL,BC	length of the name is not
		INC	BC	too long. (i.e. No more than
		JR	NC,064B,SA-NAME	ten characters.)
		LD	A,(T-ADDR-lo)	But allow for the LOADING,
		AND	A	VERIFYing and MERGEing of
		JR	NZ,0644,SA-NULL	programs with 'null' names or
				extra long names.

Report F - Invalid file name

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

Continue to handle the name of the program.

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

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

064B	SA-NAME	PUSH	IX	Copy the start address to the
		POP	HL	HL register pair.
		INC	HL	Step to the second location.
		EX	DE,HL	Switch the pointers over and
		LDIR		copy the name.

The many different parameters, if any, that follow the command are now considered. Start by handling 'xxx "name" DATA'.

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

Report 2 - Variable not found

0670	REPORT-2	RST	0008,ERROR-1	Call the error handling routine.
		DEFB	+01	

Continue with the handling of an existing array.

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

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

0685	SA-V-NEW	LD	(IX+0E),C	Copy the array's name.
		LD	A,+01	Assume an array of numbers.
		BIT	6,C	Jump if it is so.
		JR	Z,068F,SA-V-TYPE	
		INC	A	It is an array of characters.
068F	SA-V-TYPE	LD	(IX+00),A	Save the 'type' in the first location of the header area.

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

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

Now consider 'SCREEN\$'.

06A0	SA-SCR\$	CP	+AA	Is the present code the token SCREEN\$'.
------	----------	----	-----	--

JR	NZ,06C3,SA-CODE	Jump if not.
LD	A,(T-ADDR-lo)	However it is not possible to
CP	+03	have 'MERGE name SCREEN\$'.
JP	Z,1C8A,REPORT-C	
RST	0020,NEXT-CHAR	Advance CH-ADD.
CALL	1BEE,CHECK-END	Move on to the next statement
		if checking syntax.
LD	(IX+0B),+00	The display area and the
LD	(IX+0C),+1B	attribute area occupy +1800
		locations and these locations
LD	HL,+4000	start at +4000; these details
LD	(IX+0D),L	are passed to the header area
LD	(IX+0E),H	in the work space.
JR	0710,SA-TYPE-3	Jump forward.

Now consider 'CODE'.

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

Look for a 'starting address'.

06E1	SA-CODE-1	CALL	1C82,EXPT-1NUM	Fetch the first number.
		RST	0018,GET-CHAR	Is the present character a ','
		CP	+2C	or not?
		JR	Z,06F5,SA-CODE-3	Jump if it is - the number was
				a 'starting address'.
		LD	A,(T-ADDR-lo)	However refuse 'SAVE name
		AND	A	CODE' that does not have a
		JP	Z,1C8A,REPORT-C	'start' and a 'length'.
06F0	SA-CODE-2	CALL	1CE6,USE-ZERO	Put a zero on the calculator
				stack - for the 'length'.
		JR	06F9,SA-CODE-4	Jump forward.

Fetch the 'length' as it was specified.

06F5	SA-CODE-3	RST	0020,NEXT-CHAR	Advance CH-ADD.
		CALL	1C82,EXPT-1NUM	Fetch the 'length'.

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

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

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

0710	SA-TYPE-3	LD	(IX+00),+03	Enter the 'type' number.
------	-----------	----	-------------	--------------------------

		JR	075A,SA-ALL	Rejoin the other pathways.
Now consider 'LINE'; and 'no further parameters'.				
0716	SA-LINE	CP	+CA	Is the present code the token 'LINE'?
		JR	Z,0723,SA-LINE-1	Jump if it is.
		CALL	1BEE,CHECK-END	Move on to the next statement if checking syntax.
		LD	(IX+0E),+80	When there are no further parameters an +80 is entered.
		JR	073A,SA-TYPE-0	Jump forward.

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

0723	SA-LINE-1	LD	A,(T-ADDR-lo)	However only allow 'SAVE name LINE number'.
		AND	A	
		JP	NZ,1C8A,REPORT-C	
		RST	0020,NEXT-Char	Advance CH-ADD.
		CALL	1C82,EXPT-1NUM	Pass the number to the calculator stack.
		CALL	1BEE,CHECK-END	Move on to the next statement if checking syntax.
		CALL	1E99,FIND-INT2	Compress the 'line number' into the BC register pair and store it.
		LD	(IX+0D),C	
		LD	(IX+0E),B	

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

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

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

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

In all cases the header information has now been prepared.

The location 'IX+00' holds the type number.
Locations 'IX+01 to IX+0A' holds the name (+FF in 'IX+01' if null).
Locations 'IX+0B & IX+0C' hold the number of bytes that are to be found in the 'data block'.
Locations 'IX+0D to IX+10' hold a variety of parameters whose exact interpretation depends on the 'type'.

The routine continues with the first task being to separate SAVE from LOAD, VERIFY and MERGE.

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

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

	PUSH	HL	Save the 'destination' pointer.
--	------	----	---------------------------------

LD	BC,+0011	Form in the IX register pair
ADD	IX,BC	the base address of the 'second header area'.

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

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

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

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

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

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

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

PUSH	IX	Make the DE register pair point to the 'new type' and the HL register pair to the 'old name'.
POP	DE	
LD	HL,+FFF0	Ten characters are to be considered.
ADD	HL,DE	
LD	B,+0A	Jump forward if the match is to be against an actual name.
LD	A,(HL)	
INC	A	
JR	NZ,07A6,LD-NAME	But if the 'old name' is 'null' then signal 'ten characters already match'.
LD	A,C	
ADD	A,B	
LD	C,A	

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

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

LD	A,+0D	Follow the 'new name' with
RST	0010,PRINT-A-1	a 'carriage return'.

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

POP	HL	Fetch the pointer.
LD	A,(IX+00)	'SCREEN\$ and CODE' are
CP	+03	handled with VERIFY.
JR	Z,07CB,VR-CONTRL	
LD	A,(T-ADDR-lo)	Jump forward if using a
DEC	A	LOAD command.
JP	Z,0808,LD-CONTRL	
CP	+02	Jump forward if using a MERGE
JP	Z,08B6,ME-CONTRL	command; continue with a
		VERIFY command.

THE 'VERIFY' CONTROL ROUTINE

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

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

The routine continues by considering the 'destination pointer'.

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

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

07F4	VR-CONT-2	PUSH	HL	Move the 'pointer' to the
		POP	IX	IX register pair.
		LD	A,(T-ADDR-lo)	Jump forward unless using
		CP	+02	the VERIFY command; with
		SCF		the carry flag signalling
		JR	NZ,0800,VR-CONT-3	'LOAD'
		AND	A	Signal 'VERIFY'.
0800	VR-CONT-3	LD	A,+FF	Signal 'accept data block only'
				before LOADING the block.

THE 'LOAD A DATA BLOCK' SUBROUTINE

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

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

Report R - Tape loading error

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

THE 'LOAD' CONTROL ROUTINE

This routine controls the LOADing of a BASIC program, and its variables, or an array.

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

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

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

Make the actual test for room.

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

Now deal with the LOADing of arrays.

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

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

084C	LD-DATA-1	LD	HL,(E-LINE)	Find the pointer to the end-marker of the variables area - the '80-byte'.
		DEC	HL	
		LD	C,(IX+0B)	Fetch the 'length' of the new array.
		LD	B,(IX+0C)	Save this 'length'.
		PUSH	BC	

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

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

0873	LD-PROG	EX	DE,HL	Save the 'destination pointer'.
		LD	HL,(E-LINE)	Find the address of the end-marker of the current variables area - the '80-byte'.
		DEC	HL	Save IX temporarily.
		LD	(X-PTR),IX	Fetch the 'length' of the new data block.
		LD	C,(IX+0B)	Keep a copy of the 'length' whilst the present program and variables areas are reclaimed.
		LD	B,(IX+0C)	Save the pointer to the program area and the length of the new data block.
		PUSH	BC	Make sufficient room available for the new program and its variables.
		CALL	19E5,RECLAIM-1	Restore the IX register pair.
		POP	BC	The system variable VARS has also to be set for the new program.
		PUSH	HL	
		PUSH	BC	
		CALL	1655,MAKE-ROOM	
		LD	IX,(X-PTR)	
		INC	HL	
		LD	C,(IX+0F)	
		LD	B,(IX+10)	
		ADD	HL,BC	
		LD	(VARS),HL	
		LD	H,(IX+0E)	
		LD	A,H	
		AND	+C0	
		JR	NZ,08AD,LD-PROG-1	
		LD	L,(IX+0D)	
		LD	(NEWPPC),HL	
		LD	(NSPPC),+00	

The data block can now be LOAded.

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

THE 'MERGE' CONTROL ROUTINE

There are three main parts to this routine.

- i. LOAD the data block into the work space.
- ii. MERGE the lines of the new program into the old program.
- iii. MERGE the new variables into the old variables.

Start therefore with the LOADING of the data block.

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

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

POP	HL	Fetch the 'start' of the new program.
LD	DE,(PROG)	Initialise DE to the 'start' of the old program.

Enter a loop to deal with the lines of the new program.

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

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

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

In a similar manner the variables of the new program are MERGEed with the variables of the old program.

A loop is entered to deal with each of the new variables in turn.

08F0	ME-VAR-LP	LD	A,(HL)	Fetch each variable name in
------	-----------	----	--------	-----------------------------

LD	C,A
CP	+80
RET	Z
PUSH	HL
LD	HL,(VARS)

turn and test it.
Return when all the variables
have been considered.
Save the current new pointer.
Fetch VARS (for the old
program).

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

08F9	ME-OLD-VP	LD	A,(HL)
		CP	+80
		JR	Z,0923,ME-VAR-L2
		CP	C
		JR	Z,0909,ME-OLD-V2

Fetch each variable name and
test it.
Jump forward once the end
marker is found. (Make an
'addition'.)
Compare the names (1st. bytes).
Jump forward to consider it
further; returning here if it
proves not to match fully.
Save the new variable's name
whilst the next 'old variable'
is located.
Restore the pointer to the
DE register pair and go round
the loop again.

0901	ME-OLD-V1	PUSH	BC
		CALL	19B8,NEXT-ONE
		POP	BC
		EX	DE,HL
		JR	08F9,ME-OLD-VP

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

0909	ME-OLD-V2	AND	+E0
		CP	+A0
		JR	NZ,0921,ME-VAR-L1
		POP	DE
		PUSH	DE
		PUSH	HL

Consider bits 7, 6 & 5 only.
Accept all the variable types
except 'long named variables'.
Make DE point to the first
character of the 'new name'.
Save the pointer to the 'old
name'.

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

0912	ME-OLD-V3	INC	HL
		INC	DE
		LD	A,(DE)
		CP	(HL)
		JR	NZ,091E,ME-OLD-V4
		RLA	
		JR	NC,0912,ME-OLD-V3
		POP	HL
		JR	0921,ME-VAR-L1
091E	ME-OLD-V4	POP	HL
		JR	0901,ME-OLD-V1

Update both the 'old' and the
'new' pointers.
Compare the two letters

Jump forward if the match
fails.
Go round the loop until the
'last character' is found.
Fetch the pointer to the
start of the 'old' name and
jump forward - successful.
Fetch the pointer and jump
back - unsuccessful.

Come here if the match was found.

0921	ME-VAR-L1	LD	A,+FF
------	-----------	----	-------

Signal 'replace' variable.

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

0923	ME-VAR-L2	POP	DE
		EX	DE,HL
		INC	A
		SCF	
		CALL	092C,ME-ENTER
		JR	08F0,ME-VAR-LP

Fetch pointer to 'new' name.
Switch over the registers.
The zero flag is to be set if there
is to be a 'replacement'; reset
for an 'addition'.
Signal 'handling variables'.
Now make the entry.
Go round the loop to consider
the next new variable.

THE 'MERGE A LINE OR A VARIABLE' SUBROUTINE

This subroutine is entered with the following parameters:

Carry flag	reset
	set
Zero	reset
	set
HL register pair	
DE register pair	

- MERGE a BASIC line.
- MERGE a variable.
- It will be an 'addition'.
- It is a 'replacement'.
- Points to the start of the new entry.
- Points to where it is to MERGE.

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

The new entry can now be made.

093E	ME-ENT-1	EX	AF,A'F'	Save the flags.
		PUSH	DE	Make a copy of the
				'destination' pointer.
		CALL	19B8,NEXT-ONE	Find the length of the 'new'
				variable/line.
		LD	(X-PTR),HL	Save the pointer to the 'new'
				variable/line.
		LD	HL,(PROG)	Fetch PROG - to avoid
				corruption.
		EX	(SP),HL	Save PROG on the stack and
				fetch the 'new' pointer.
		PUSH	BC	Save the length.
		EX	AF,A'F'	Retrieve the flags.
		JR	C,0955,ME-ENT-2	Jump forward if adding a new
				variable.
		DEC	HL	A new line is added before the
				'destination' location.
		CALL	1655,MAKE-ROOM	Make the room for the new line.
		INC	HL	
		JR	0958,ME-ENT-3	Jump forward.
0955	ME-ENT-2	CALL	1655,MAKE-ROOM	Make the room for the new
				variable.
0958	ME-ENT-3	INC	HL	Point to the 1st new location.
		POP	BC	Retrieve the length.
		POP	DE	Retrieve PROG and store it
		LD	(PROG),DE	in its correct place.
		LD	DE,(X-PTR)	Also fetch the 'new' pointer.
		PUSH	BC	Again save the length and the
		PUSH	DE	'new' pointer.
		EX	DE,HL	Switch the pointers and copy
		LDIR		the 'new' variable/line into the
				room made for it.

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

	POP	HL	Fetch the 'new' pointer.
	POP	BC	Fetch the length.
	PUSH	DE	Save the 'old' pointer. (Points
			to the location after the 'added'
			variable/line.)
	CALL	19E8,RECLAIM-2	Remove the variable/line from
			the work space.
	POP	DE	Return with the 'old' pointer
	RET		in the DE register pair.

THE 'SAVE' CONTROL ROUTINE

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

0970	SA-CONTRL	PUSH	HL	Save the 'pointer'.
		LD	A,+FD	Ensure that channel 'K'
		CALL	1601,CHAN-OPEN	is open.
		XOR	A	Signal 'first message'.
		LD	DE,+09A1	Print the message – 'Start tape,
		CALL	0C0A,PO-MSG	then press any key.'
		SET	5,(TV-FLAG)	Signal 'screen will require to be
				cleared'.
		CALL	15D4,WAIT-KEY	Wait for a key to be pressed.

Upon receipt of a keystroke the 'header' is saved.

	PUSH	IX	Save the base address of the
			'header' on the machine stack.
	LD	DE,+0011	Seventeen bytes are to be
			SAVED.
	XOR	A	Signal 'it is a header'.
	CALL	04C2,SA-BYTES	Send the 'header'; with a leading
			'type' byte and a trailing 'parity'
			byte.

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

		POP	IX	Retrieve the pointer to the
				'header'.
		LD	B,+32	The delay is for fifty
0991	SA-1-SEC	HALT		interrupts, i.e. one second.
		DJNZ	0991,SA-1-SEC	
		LD	E,(IX+0B)	Fetch the length of the
		LD	D,(IX+0C)	data block that is to be SAVED.
		LD	A,+FF	Signal 'data block'.
		POP	IX	Fetch the 'start of block
		JP	04C2,SA-BYTES	pointer' and SAVE the block.

THE CASSETTE MESSAGES

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

09A1	DEFB +80	- Initial byte is stepped over.
09A2	DEFM	- Start tape, then press any key.
09C1	DEFM	- 'carriage return' - Program:
09CB	DEFM	- 'carriage return' - Number array:
09DA	DEFM	- 'carriage return' - Character array:
09EC	DEFM	- 'carriage return' - Bytes:

THE SCREEN & PRINTER HANDLING ROUTINES

THE 'PRINT-OUT' ROUTINES

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

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

THE 'CONTROL CHARACTER' TABLE

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

THE 'CURSOR LEFT' SUBROUTINE

The subroutine is entered with the B register holding the current line number and the C register with the current column number.

0A23	PO-BACK-1	INC	C	Move leftwards by one column.
		LD	A,+22	Accept the change unless
		CP	C	up against the lefthand side.
		JR	NZ,0A3A,PO-BACK-3	
		BIT	1,(FLAGS)	If dealing with the printer
		JR	NZ,0A38,PO-BACK-2	jump forward.
		INC	B	Go up one line.
		LD	C,+02	Set column value.
		LD	A,+18	Test against top line.
		CP	B	Note: This ought to be +19.
		JR	NZ,0A3A,PO-BACK-3	Accept the change unless at
				the top of the screen.
		DEC	B	Unacceptable so down a line.
0A38	PO-BACK-2	LD	C,+21	Set to lefthand column.
0A3A	PO-BACK-3	JP	0DD9,CL-SET	Make an indirect return via
				CL-SET & PO-STORE.

THE 'CURSOR RIGHT' SUBROUTINE

This subroutine performs an operation identical to the BASIC statement – PRINT OVER 1;CHR\$ 32; -.

0A3D	PO-RIGHT	LD	A,(P-FLAG)	Fetch P-FLAG and save it on
		PUSH	AF	the machine stack.

LD	(P-FLAG),+01	Set P-FLAG to OVER 1.
LD	A,+20	A 'space'.
CALL	0B65,PO-CHAR	Print the character.
POP	AF	Fetch the old value of
LD	(P-FLAG),A	P-FLAG.
RET		Finished.

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

THE 'CARRIAGE RETURN' SUBROUTINE

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

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

THE 'PRINT COMMA' SUBROUTINE

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

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

THE 'PRINT A QUESTION MARK' SUBROUTINE

A question mark is printed whenever an attempt is made to print an unprintable code.

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

THE 'CONTROL CHARACTERS WITH OPERANDS' ROUTINE

The control characters from INK to OVER require a single operand whereas the control characters AT & TAB are required to be followed by two operands.

The present routine leads to the control character code being saved in TVDATA-lo, the first operand in TVDATA-hi or the A register if there is only a single operand required, and the second operand in the A register.

0A6D	PO-TV-2	LD	DE,+0A87	Save the first operand in
		LD	(TVDATA-hi),A	TVDATA-hi and change the
		JR	0A80,PO-CHANGE	address of the 'output' routine
				to PO-CONT (+0A87).

Enter here when handling the characters AT & TAB.

0A75	PO-2-OPER	LD	DE,+0A6D	The character code will be
		JR	0A7D,PO-TV-1	saved in TVDATA-lo and the
				address of the 'output' routine
				changed to PO-TV-2 (+0A6D).

Enter here when handling the colour items - INK to OVER.

0A7A	PO-1-OPER	LD	DE,+0A87	The 'output' routine is to be
				changed to PO-CONT (+0A87).
0A7D	PO-TV-1	LD	(TVDATA-lo),A	Save the control character code.

The current 'output' routine address is changed temporarily.

0A80	PO-CHANGE	LD	HL,(CURCHL)	HL will point to the 'output' routine address.
		LD	(HL),E	Enter the new 'output' routine address and thereby
		INC	HL	force the next character code
		LD	(HL),D	to be considered as an operand.
		RET		

Once the operands have been collected the routine continues.

0A87	PO-CONT	LD	DE,+09F4	Restore the original address for PRINT-OUT (+09F4).
		CALL	0A80,PO-CHANGE	Fetch the control code and the first operand if there are indeed
		LD	HL,(TVDATA)	two operands.
		LD	D,A	The 'last' operand and the control code are moved.
		LD	A,L	Jump forward if handling
		CP	+16	INK to OVER.
		JP	C,2211,CO-TEMPS	Jump forward if handling TAB.
		JR	NZ,0AC2,PO-TAB	

Now deal with the AT control character.

		LD	B,H	The line number.
		LD	C,D	The column number.
		LD	A,+1F	Reverse the column number;
		SUB	C	i.e. +00 - +1F becomes +1F - +00.
		JR	C,0AAC,PO-AT-ERR	Must be in range.
		ADD	A,+02	Add in the offset to give
		LD	C,A	C holding +21 - +22.
		BIT	1,(FLAGS)	Jump forward if handling the printer.
		JR	NZ,0ABF,PO-AT-SET	Reverse the line number;
		LD	A,+16	i.e. +00 - +15 becomes +16 - +01.
		SUB	B	
0AAC	PO-AT-ERR	JP	C,1E9F,REPORT-B	If appropriate jump forward.
		INC	A	The range +16 - +01 becomes
		LD	B,A	+17 - +02.
		INC	B	And now +18 - +03.
		BIT	0,(TV-FLAG)	If printing in the lower part of the screen then consider whether scrolling is needed.
		JP	NZ,0C55,PO-SCR	Give report 5 - Out of screen, if required.
		CP	(DF-SZ)	Return via CL-SET & PO-STORE.
0ABF	PO-AT-SET	JP	C,0C86,REPORT-5	
		JP	0D09,CL-SET	

And the TAB control character.

0AC2	PO-TAB	LD	A,H	Fetch the first operand.
0AC3	PO-FILL	CALL	0B03,PO-FETCH	The current print position.
		ADD	A,C	Add the current column value.
		DEC	A	Find how many 'spaces', modulo
		AND	+1F	32, are required and return
		RET	Z	if the result is zero.
		LD	D,A	Use 0 as the counter.
		SET	0,(FLAGS)	Suppress 'leading space'.
0AD0	PO-SPACE	LD	A,+20	Print 'D number' of
		CALL	0C3B,PO-SAVE	spaces.
		DEC	D	
		JR	NZ,0AD0,PO-SPACE	
		RET		Now finished.

PRINTABLE CHARACTER CODES.

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

0AD9	PO-ABLE	CALL	0B24,PO-ANY	Print the character(s) and continue into PO-STORE.
------	---------	------	-------------	---

THE 'POSITION STORE' SUBROUTINE

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

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

THE 'POSITION FETCH' SUBROUTINE

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

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

THE 'PRINT ANY CHARACTER(S)' SUBROUTINE

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

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

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

0B38	PO-GR-1	LD	HL,+5C92	This is MEMBOT.
		CALL	0B3E,PO-GR-2	In effect call the following subroutine twice.
0B3E	PO-GR-2	RR	B	Determine bit 0 (and later bit 2) of the graphic code.
		SBC	A,A	The A register will hold +00 or +0F depending on the value of the bit in the code.
		AND	+0F	

0B4C	PO-GR-3	LD	C,A	Save the result in C. Determine bit 1 (and later bit 3) of the graphic code. The A register will hold +00 or +F0. The two results are combined. The A register holds half the character form and has to be used four times. This is done for the upper half of the character form and then the lower.
		RR	B	
		SBC	A,A	
		AND	+F0	
		OR	C	
		LD	C,+04	
		LD	(HL),A	
		INC	HL	
		DEC	C	
		JR	NZ,0B4C,PO-GR-3	
		RET		

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

0B52	PO-T&UDG	SUB	+A5	Jump forward with token codes
		JR	NC,0B5F,PO-T	
		ADD	A,+15	
		PUSH	BC	
0B5F	PO-T	LD	BC,(UDG)	UDG codes are now +00 - +0F. Save the current position values on the machine stack. Fetch the base address of the UDG area and jump forward. Now print the token and return via PO-FETCH.
		JR	0B6A,PO-CHAR-2	
		CALL	0C10,PO-TOKENS	
		JP	0B03,PO-FETCH	

The required character form is identified.

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

THE 'PRINT ALL CHARACTERS' SUBROUTINE

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

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

0B93 PR-ALL-1

PUSH	DE
CALL	Z,0C55,PO-SCR
POP	DE

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

Now consider the present state of INVERSE & OVER

		PUSH	BC	Save the position values
		PUSH	HL	and the destination address
				on the machine stack.
		LD	A,(P-FLAG)	Fetch P-FLAG and read bit 0.
		LD	B,+FF	Prepare the 'OVER-mask' in
		RRA		the B register; i.e. OVER 0
		JR	C,0BA4,PR-ALL-2	= +00 & OVER 1 - +FF.
0BA4	PR-ALL-2	INC	B	
		RRA		Read bit 2 of P-FLAG and
		RRA		prepare the 'INVERSE-mask'
		SBC	A,A	in the C register; i.e.
		LD	C,A	INVERSE 0 = +00 & INVERSE
				1 = +FF.
		LD	A,+08	Set the A register to hold
		AND	A	the 'pixel-line' counter and
				clear the carry flag.
		BIT	1,(FLAGS)	Jump forward if handling
		JR	Z,0BB6,PR-ALL-3	the screen.
		SET	1,(FLAGS2)	Signal 'printer buffer no longer
				empty'.
		SCF		Set the carry flag to show that
				the printer is being used.
0BB6	PR-ALL-3	EX	DE,HL	Exchange the destination
				address with the base address
				before entering the loop.

The character can now be printed. Eight passes of the loop are made - one for each 'pixel-line'.

0BB7	PR-ALL-4	EX	AF,A'F'	The carry flag is set when using
				the printer. Save this flag in F'.
		LD	A,(DE)	Fetch the existing 'pixel-line'.
		AND	B	Use the 'OVER-mask' and then
		XOR	(HL)	XOR the result with the 'pixel-
				line' of the character form.
		XOR	C	Finally consider the 'INVERSE-
				mask'.
		LD	(DE),A	Enter the result.
		EX	AF,A'F'	Fetch the printer flag and
		JR	C,0BD3,PR-ALL-6	jump forward if required.
		INC	D	Update the destination address
0BC1	PR-ALL-5	INC	HL	Update the 'pixel-line' of
				the character form.
		DEC	A	Decrease the counter and loop
		JR	NZ,0BB7,PR-ALL-4	back unless it is zero.

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

	EX	DE,HL	Make the H register hold a
	DEC	H	correct high-address for the
			character area.
	BIT	1,(FLAGS)	Set the attribute byte only if
	CALL	Z,0BDB,PO-ATTR	handling the screen.
	POP	HL	Restore the original
	POP	BC	destination address and the
			position values.
	DEC	C	Decrease the column number
	INC	HL	and increase the destination
	RET		address before returning.

When the printer is being used the destination address has to be updated in increments of +20.

0BD3	PR-ALL-6	EX	AF,A'F'	Save the printer flag again.
		LD	A,+20	The required increment value.
		ADD	A,E	Add the value and pass the
		LD	E,A	result back to the E register.
		EX	AF,A'F'	Fetch the flag.
		JR	0BC1,PR-ALL-5	Jump back into the loop.

THE 'SET ATTRIBUTE BYTE' SUBROUTINE

The appropriate attribute byte is identified and fetched. The new value is formed by manipulating the old value, ATTR-T, MASK-T and P-FLAG. Finally this new value is copied to the attribute area.

0BDB	PO-ATTR	LD	A,H	The high byte of the destination address is divided by eight and ANDed with +03 to determine which third of the screen is being addressed; i.e. 00, 01 or 02.
		RRCA		
		RRCA		
		RRCA		
		AND	+03	
		OR	+58	The high byte for the attribute area is then formed.
		LD	H,A	D holds ATTR-T, and E holds MASK-T.
		LD	DE,(ATTR-T)	The old attribute value.
		LD	A,(HL)	The values of MASK-T and ATTR-R are taken into account.
		XOR	E	
		AND	D	
		XOR	E	
		BIT	6,(P-FLAG)	Jump forward unless dealing with PAPER 9.
		JR	Z,0BFA,PO-ATTR-1	The old paper colour is ignored and depending on whether the ink colour is light or dark the new paper colour will be black (000) or white (111).
		AND	+C7	Jump forward unless dealing with INK 9.
		BIT	2,A	The old ink colour is ignored and depending on whether the paper colour is light or dark the new ink colour will be black (000) or white (111).
		JR	NZ,0BFA,PO-ATTR-1	
		XOR	+38	
0BFA	PO-ATTR-1	BIT	4,(P-FLAG)	Jump forward unless dealing with INK 9.
		JR	Z,0C08,PO-ATTR-2	
		AND	+F8	
		BIT	5,A	
		JR	NZ,0C08,PO-ATTR-2	
		XOR	+07	
0C08	PO-ATTR-2	LD	(HL),A	Enter the new attribute value and return.
		RET		

THE 'MESSAGE PRINTING' SUBROUTINE

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

0C0A	PO-MSG	PUSH	HL	The high byte of the last entry on the machine stack is made zero so as to suppress trailing spaces (see below).
		LD	H,+00	
		EX	(SP),HL	
		JR	0C14,PO-TABLE	Jump forward.

Enter here when expanding token codes.

0C10	PO-TOKENS	LD	DE,+0095	The base address of the token table.
		PUSH	AF	Save the code on the stack.
				(Range +00 - +5A; RND - COPY).

The table is searched and the correct entry printed.

0C14	PO-TABLE	CALL	0C41,PO-SEARCH	Locate the required entry.
		JR	C,0C22,PO-EACH	Print the message/token.
		LD	A,+20	A 'space' will be printed

BIT	0,(FLAGS)	before the message/token
CALL	Z,0C3B,PO-SAVE	if required.

The characters of the message/token are printed in turn.

0C22	PO-EACH	LD	A,(DE)	Collect a code.
		AND	+7F	Cancel any 'inverted bit'.
		CALL	0C3B,PO-SAVE	Print the character.
		LD	A,(DE)	Collect the code again.
		INC	DE	Advance the pointer.
		ADD	A,A	The 'inverted bit' goes to
		JR	NC,0C22,PO-EACH	the carry flag and signals
				the end of the message/token;
				otherwise jump back.

Now consider whether a 'trailing space' is required.

		POP	DE	For messages - D holds +00;
				for tokens - D holds +00 - +5A.
		CP	+48	Jump forward if the last
		JR	Z,0C35,PO-TRSP	character was a '\$'.
		CP	+82	Return if the last character
		RET	C	was any other before 'A'.
0C35	PO-TR-SP	LD	A,D	Examine the value in D and
		CP	+03	return if it indicates a
		RET	C	message, RND, INKEY\$ or PI.
		LD	A,+20	All other cases will require
				a 'trailing space'.

THE 'PO-SAVE' SUBROUTINE

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

0C3B	PO-SAVE	PUSH	DE	Save the DE register pair.
		EXX		Save HL & BC.
		RST	0010,PRINT-A-1	Print the single character.
		EXX		Restore HL & BC.
		POP	DE	Restore DE.
		RET		Finished.

THE 'TABLE SEARCH' SUBROUTINE

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

0C41	PO-SEARCH	PUSH	AF	Save the 'entry number'.
		EX	DE,HL	HL now holds the base address.
		INC	A	Make the range +01 - ?.
0C44	PO-STEP	BIT	7,(HL)	Wait for an 'inverted
		INC	HL	character'.
		JR	Z,0C44,PO-STEP	
		DEC	A	Count through the entries
		JR	NZ,0C44,PO-STEP	until the correct one is found.
		EX	DE,HL	DE points to the initial character.
		POP	AF	Fetch the 'entry number' and
		CP	+20	return with carry set for the
		RET	C	first thirty two entries.
		LD	A,(DE)	However if the initial
		SUB	+41	character is a letter then a
		RET		leading space may be needed.

THE 'TEST FOR SCROLL' SUBROUTINE

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

0C55	PO-SCR	BIT	1,(FLAGS)	Return immediately if the printer is being used. Pre-load the machine stack with the address of 'CL-SET'. Transfer the line number. Jump forward if considering 'INPUT ... AT ...'. Return, via CL-SET, if the line number is greater than the value of DF-SZ; give report 5 if it is less; otherwise continue. Jump forward unless dealing with an 'automatic listing'. Fetch the line counter. Decrease this counter. Jump forward if the listing is to be scrolled. Otherwise open channel 'K', restore the stack pointer, flag that the automatic listing has finished and return via CL-SET.
		RET	NZ	
		LD	DE,+0DD9	
		PUSH	DE	
		LD	A,B	
		BIT	0,(TV-FLAG)	
		JP	NZ,0D02,PO-SCR-4	
		CP	(DF-SZ)	
		JR	C,0C86,REPORT-6	
		RET	NZ	
		BIT	4,(TV-FLAG)	
		JR	Z,0C88,PO-SCR-2	
		LD	E,(BREG)	
		DEC	E	
		JR	Z,0CD2,PO-SCR-3	
		LD	A,+00	
		CALL	1601,CHAN-OPEN	
		LD	SP,(LIST-SP)	
		RES	4,(TV-FLAG)	
		RET		

Report 5 - Out of screen

0C86 REPORT-5	RST	0008,ERROR-1	Call the error handling routine.
	DEFB	+04	

Now consider if the prompt 'scroll?' is required.

0C88	PO-SCR-2	DEC	(SCR-CT)	Decrease the scroll counter and proceed to give the prompt only if it becomes zero.
		JR	NZ,0CD2,PO-SCR-3	

Proceed to give the prompt message.

LD	A,+18	The counter is reset.
SUB	B	
LD	(SCR-CT),A	The current values of ATTR-T and MASK-T are saved. The current value of P-FLAG is saved. Channel 'K' is opened.
LD	HL,(ATTR-T)	
PUSH	HL	The message 'scroll?' is message '0'. This message is now printed. Signal 'clear the lower screen after a keystroke'. This is FLAGS. Signal 'L mode'. Signal 'no key yet'. Note: DE should be pushed also. Fetch a single key code. Restore the registers. There is a jump forward to REPORT-D - 'BREAK - CONT repeats' - if the keystroke was 'BREAK', 'STOP', 'N' or 'n'; otherwise accept the keystroke as indicating the need to scroll the display. Open channel 'S'. Restore the value of
LD	A,(P-FLAG)	
PUSH	AF	
LD	A,+FD	
CALL	1601,CHAN-OPEN	
XOR	A	
LD	DE,+0CF8	
CALL	0C0A,PO-MSG	
SET	5,(TV-FLAG)	
LD	HL,+5C3B	Restore the value of
SET	3,(HL)	
RES	5,(HL)	
EXX		
CALL	15D4,WAIT-KEY	
EXX		
CP	+20	
JR	Z,0D00,REPORT-D	
CP	+E2	
JR	Z,0D00,REPORT-D	
OR	+20	
CP	+6E	
JR	Z,0D00,REPORT-D	
LD	A,+FE	
CALL	1601,CHAN-OPEN	
POP	AF	

LD	(P-FLAG),A	P-FLAG.
POP	HL	Restore the values of ATTR-T
LD	(ATTR-T),HL	and MASK-T.

The display is now scrolled.

0CD2	PO-SCR-3	CALL	0DFE,CL-SC-ALL	The whole display is scrolled. The line and column numbers for the start of the line above the lower part of the display are found and saved. The corresponding attribute byte for this character area is then found. The HL register pair holds the address of the byte.
		LD	B(DF-SZ)	
		INC	B	
		LD	C,+21	
		PUSH	BC	
		CALL	0E9B,CL-ADDR	
		LD	A,H	
		RRCA		
		RRCA		
		RRCA		
		AND	+03	
		OR	+58	
		LD	H,A	

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

		LD	DE,+5AE0	DE points to the first attribute byte of the bottom line.
		LD	A,(DE)	The value is fetched.
		LD	C,(HL)	The 'lower part' value.
		LD	B,+20	There are thirty two bytes.
		EX	DE,HL	Exchange the pointers.
0CF0	PO-SCR-3A	LD	(DE),A	Make the first exchange
		LD	(HL),C	and then proceed to use the
		INC	DE	same values for the thirty
		INC	HL	two attribute bytes of the
		DJNZ	0CF0,PO-SCR-3A	two lines being handled.
		POP	BC	The line and column numbers of the bottom line of the 'upper part' are fetched before returning.
		RET		

The 'scroll?' message

0CF8	DEFB	+80	Initial marker - stepped over.
	DEFB	+73,+63,+72,+6F	s-c-r-o
	DEFB	+6C,+6C,+BF	l - l - ? (inverted).

Report 0 - BREAK - CONT repeats

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

The lower part of the display is handled as follows:

0D02	PO-SCR-4	CP	+02	The 'out of screen' error is given if the lower part is going to be 'too large' and a return made if scrolling is unnecessary. The A register will now hold 'the number of scrolls to be made'.
		JR	C,0C86,REPORT-5	
		ADD	A,(DF-SZ)	
		SUB	+19	
		RET	NC	
		PUSH	BC	The line and column numbers are now saved.
		LD	B,A	The 'scroll number', ATTR-T
		LD	HL,(ATTR-T)	MASK-T & P-FLAG are all
		PUSH	HL	saved.
		LD	HL,(P-FLAG)	
		PUSH	HL	
		CALL	0D40,TEMPS	The 'permanent' colour items are to be used.

The 'scroll number' is fetched.

0D1C	PO-SCR-4A	PUSH	AF
		LD	HL,+5C6B
		LD	B,(HL)
		LD	A,B
		INC	A
		LD	(HL),A
		LD	HL,+5C89
		CP	(HL)
		JR	C.0D2D.PO-SCR-4B

Save the 'number'.
This is DF-SZ.
The value in DF-SZ is incremented; the B register set to hold the former value and the A register the new value.
This is S-POSN-hi.
The jump is taken if only the lower part of the display is to be scrolled. (B = old DF-SZ). Otherwise S-POSN-hi is incremented and the whole display scrolled. (B = +18)

0D2D	PO-SCR-4B	CALL	0E00,CL-SCROLL
		POP	AF
		DEC	A
		JR	NZ,0D1C,PO-SCR-4A
		POP	HL
		LD	(P-FLAG),L
		POP	HL
		LD	(ATTR-T),HL
		LD	BC,(S-POSN)
		RES	0,(TV-FLAG)
		CALL	0DD9,CL-SET
		SET	0,(TV-FLAG)
		POP	BC
		RET	

Scroll 'B' lines.
Fetch and decrement the
scroll number'.
Jump back until finished.
Restore the value of
P-FLAG.
Restore the values of ATTR-T
and MASK-T.
In case S-POSN has been
changed CL-SET is called to
give a matching value to DF-CC.
Reset the flag to indicate that
the lower screen is being
handled, fetch the line and
column numbers, and then
return

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

0D4D	TEMPS	XOR	A
		LD	HL,(ATTR-P)
		BIT	0,(TV-FLAG)
		JR	Z,0D5B TEMPS-1

A is set to hold +00.
The current values of ATTR-P
and MASK-P are fetched.
Jump forward if handling the
main part of the screen.

		LD	H,A
		LD	L,(BORDCR)
0D5B	TEMPS-1	LD	(ATTR-T).HL

Otherwise use +00 and the value in BORDCR instead.
Now set ATTR-T & MASK-T.

LD HL,+5C91
JR NZ,0D65,TEMPS-2

This is P-FLAG.
Jump forward if dealing with
the lower part of the screen
(A = +00).

LD A,(HL)
RRCA

Otherwise fetch the value of P-FLAG and move the odd bits to the even bits.

```
0D65    TEMPS-2    XOR        (HL)
                   AND        +55
                   XOR        (HL)
                   LD         (HL),A
                   RET
```

Proceed to copy the even bits of A to P-FLAG.

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

0D6B	CLS	CALL	0DAF,CL-ALL	The whole of the display is 'cleared'.
0D6E	CLS-LOWER	LD RES	HL,+5C3C 5,(HL)	This is TV-FLAG. Signal 'do not clear the lower screen after keystroke'.
		SET CALL	0,(HL) 0D4D,TEMPS	Signal 'lower part'. Use the permanent values. i.e. ATTR-T is copied from BORDCR.
		LD CALL	B,(DF-SZ) 0E44,CL-LINE	The lower part of the screen is now 'cleared' with these values.

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

		LD	HL,+5AC0	Attribute byte at start of line '22'.
		LD	A,(ATTR-P)	Fetch ATTR-P.
		DEC	B	The line counter.
0D87	CLS-1	JR	0D8E,CLS-3	Jump forward into the loop.
0D89	CLS-2	LD	C,+20	+20 characters per line.
		DEC	HL	Go back along the line setting the attribute bytes.
		LD	(HL),A	
		DEC	C	
0D8E	CLS-3	JR	NZ,0D89,CLS-2	
		DJNZ	0D87,CLS-1	Loop back until finished.

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

		LD	(DF-SZ),+02	It will be two lines in size.
--	--	----	-------------	-------------------------------

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

0D94	CL-CHAN	LD	A,+FD	Open channel 'K'.
		CALL	1601,CHAN-OPEN	
		LD	HL,(CURCHL)	Fetch the address of the current channel and make the output address +09F4 (= PRINT-OUT) and the input address +10A8 (= KEY-INPUT).
0DA0	CL-CHAN-A	LD	DE,+09F4	
		AND	A	
		LD	(HL),E	
		INC	HL	
		LD	(HL),D	
		INC	HL	
		LD	DE,+10A8	
		CCF		First the output address then the input address.
		JR	C,0DA0,CL-CHAN-A	As the lower part of the display is being handled the 'lower print line' will be line '23'.
		LD	BC,+1721	
		JR	0DD9,CL-SET	Return via CL-SET.

THE 'CLEARING THE WHOLE DISPLAY AREA' SUBROUTINE

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

0DAF	CL-ALL	LD	HL,+0000	The system variable C00RDS is reset to zero.
		LD	(C00RDS),HL	Signal 'the screen is clear'.
		RES	0,(FLAGS2)	Perform the 'house keeping' tasks.
		CALL	0D94,CL-CHAN	Open channel 'S'.
		LD	A,+FE	
		CALL	1601,CHAN-OPEN	
		CALL	0D4D,TEMPS	Use the 'permanent' values.
		LD	B,+18	Now 'clear' the 24 lines of the display.
		CALL	0E44,CL-LINE	Ensure that the current output address is +09F4
		LD	HL,(CURCHL)	
		LD	DE,+09F4	

LD	(HL),E	(PRINT-OUT).
INC	HL	
LD	(HL),D	
LD	(SCR-CT),+01	Reset the scroll counter.
LD	BC,+1821	As the upper part of the display is being handled the 'upper print line' will be Line '0'.
		Continue into CL-SET.

THE 'CL-SET' SUBROUTINE

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

0DD9	CL-SET	LD	HL,+5B00	The start of the printer buffer.
		BIT	1,(FLAGS)	Jump forward if handling the printer buffer.
		JR	NZ,0DF4,CL-SET-2	
		LD	A,B	Transfer the line number.
		BIT	0,(TV-FLAG)	Jump forward if handling the main part of the display.
		JR	Z,0DEE,CL-SET-1	
		ADD	A,(DF-SZ)	The top line of the lower part of the display is called 'line +18' and this has to be converted.
		SUB	+18	
0DEE	CL-SET-1	PUSH	BC	The line & column numbers are saved.
		LD	B,A	The line number is moved.
		CALL	0E9B,CL-ADDR	The address for the start of the line is formed in HL.
		POP	BC	The line & column numbers are fetched back.
0DF4	CL-SET-2	LD	A,+21	The column number is now reversed and transferred to the DE register pair.
		SUB	C	
		LD	E,A	
		LD	D,+00	
		ADD	HL,DE	The required address is now formed; and the address and the line and column numbers are stored by jumping to PO-STORE.
		JP	0ADC,PO-STORE	

THE 'SCROLLING' SUBROUTINE

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

0DFE	CL-SC-ALL	LD	B,+17	The entry point after 'scroll?'
------	-----------	----	-------	---------------------------------

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

0E00	CL-SCROLL	CALL	0E9B,CL-ADDR	Find the starting address of the line.
		LD	C,+08	There are eight pixel lines to a complete line.

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

0E05	CL-SCR-1	PUSH	BC	Save both counters.
		PUSH	HL	Save the starting address.
		LD	A,B	Jump forward unless dealing at the present moment with a 'third' of the display.
		AND	+07	
		LD	A,B	
		JR	NZ,0E19,CL-SCR-3	

The pixel lines of the top lines of the 'thirds' of the display have to be moved across the 2K boundaries. (Each 'third' = 2K.)

0E0D	CL-SCR-2	EX	DE,HL	The result of this manipulation is to leave HL unchanged and DE pointing to the required destination. There are +20 characters. Decrease the counter as one line is being dealt with. Now move the thirty two bytes.
		LD	HL,+F8E0	
		ADD	HL,DE	
		EX	DE,HL	
		LD	BC,+0020	
		DEC	A	
		LDIR		

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

0E19	CL-SCR-3	EX	DE,HL	Again DE is made to point to the required destination. This time only thirty two locations away. Save the line number in B. Now find how many characters there are remaining in the 'third'. Pass the 'character total' to the C register. Fetch the line number. BC holds the 'character total' and a pixel line from each of the characters is 'scrolled'. Now prepare to increment the address to jump across a 'third' boundary. Increase HL by +0700. Jump back if there are any 'thirds' left to consider.
		LD	HL,+FFE0	
		ADD	HL,DE	
		EX	DE,HL	
		LD	B,A	
		AND	+07	
		RRCA		
		RRCA		
		RRCA		
		LD	C,A	
		LD	A,B	
		LD	B,+00	
		LDIR		
		LD	B,+07	
		ADD	HL,BC	
		AND	+F8	
		JR	NZ,0E0D,CL-SCR-2	

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

		POP	HL	Fetch the original address. Address the next pixel line. Fetch the counters. Decrease the pixel line counter and jump back unless eight lines have been moved.
		INC	H	
		POP	BC	
		DEC	C	
		JR	NZ,0E05,CL-SCR-1	

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

		CALL	0E88,CL-ATTR	The required address in the attribute area and the number of characters in 'B' lines are found. The displacement for all the attribute bytes is thirty two locations away. The attribute bytes are 'scrolled'.
		LD	HL,+FFE0	
		ADD	HL,DE	
		EX	DE,HL	
		LDIR		

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

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

THE 'CLEAR LINES' SUBROUTINE

This subroutine will clear the bottom 'B' lines of the display.

0E44	CL-LINE	PUSH	BC	The line number is saved for the duration of the subroutine. The starting address for the line is formed in HL.
		CALL	0E9B,CL-ADDR	

LD C,+08

Again there are eight pixel lines to be considered.

Now enter a loop to clear all the pixel lines.

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

Now find if the loop has been used eight times.

POP	HL	Update the address for each pixel line.
INC	H	Fetch the counters.
POP	BC	Decrease the pixel line counter and jump back unless finished.
DEC	C	
JR	NZ,0E4A,CL-LINE-1	

Next the attribute bytes are set as required. The value in ATTR-P will be used when handling the main part of the display and the value in BORDCR when handling the lower part.

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

THE 'CL-ATTR' SUBROUTINE

This subroutine has two separate functions.

- For a given display area address the appropriate attribute address is returned in the DE register pair. Note that the value on entry points to the 'ninth' line of a character.
- For a given line number, in the B register, the number of character areas in the display from the start of that line onwards is returned in the BC register pair.

0E88	CL-ATTR	LD	A,H	Fetch the high byte.
		RRCA		Multiply this value by
		RRCA		thirty two.
		RRCA		
		DEC	A	Go back to the 'eight' line.
		OR	+50	Address the attribute area.
		LD	H,A	Restore to the high byte and
		EX	DE,HL	transfer the address to DE.
		LD	H,C	This is always zero.
		LD	L,B	The line number.
		ADD	HL,HL	Multiply by thirty two.
		ADD	HL,HL	
		ADD	HL,HL	
		ADD	HL,HL	
		ADD	HL,HL	
		LD	B,H	Move the result to the
		LD	C,L	BC register pair before
		RET		returning.

THE 'CL-ADDR' SUBROUTINE

For a given line number, in the B register, the appropriate display file address is formed in the HL register pair.

0E9B	CL-ADDR	LD	A,+18	The line number has to be
		SUB	B	reversed.
		LD	D,A	The result is saved in D.
		RRCA		In effect '(A mod 8) * 32'.
		RRCA		In a 'third' of the display
		RRCA		the low byte for the:
		AND	+E0	1st. line = +00,
				2nd. line = +20, etc.
		LD	L,A	The low byte goes into L.
		LD	A,D	The true line number is fetched.
		AND	+18	In effect '64 + 8 * INT (A/8)'
		OR	+40	For the upper 'third' of the
				display the high byte = +40,
				middle 'third' = +48, and the
				lower 'third' = +50.
		LD	H,A	The high byte goes to H.
		RET		Finished.

THE 'COPY' COMMAND ROUTINE

The one hundred and seventy six pixel lines of the display are dealt with one by one.

0EAC	COPY	DI		The maskable interrupt is
				disabled during COPY.
		LD	B,+B0	The '176' lines.
		LD	HL,+4000	The base address of the display.

The following loop is now entered.

0EB2	COPY-1	PUSH	HL	Save the base address and
		PUSH	BC	the number of the line.
		CALL	0EF4,COPY-LINE	It is called '176' times.
		POP	BC	Fetch the line number and
		POP	HL	the base address.
		INC	H	The base address is updated by
				'256' locations for each line of
				pixels.

LD	A,H	Jump forward and hence round
AND	+07	the loop again directly for the
JR	NZ,0EC9,COPY-2	eight pixel lines of a character
		line.

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

	LD	A,L	Fetch the low byte.
	ADD	A,+20	Update it by +20 bytes.
	LD	L,A	The carry flag will be reset when
			'within thirds' of the display.
	CCF		Change the carry flag.
	SBC	A,A	The A register will hold +F8
	AND	+F8	when within a 'third' but +00
			when a new 'third' is reached.
	ADD	A,H	The high byte of the
	LD	H,A	address is now updated.
0EC9	COPY-2	DJNZ	0EB2,COPY-1
			Jump back until '176' lines have
			been printed.
	JR	0EDA,COPY-END	Jump forward to the end
			routine.

THE 'COPY-BUFF' SUBROUTINE

This subroutine is called whenever the printer buffer is to have its contents passed to the printer.

0ECD	COPY-BUFF	DI		Disable the maskable interrupt.
		LD	HL,+5B00	The base address of the printer
				buffer.
				There are eight pixel lines.
0ED3	COPY-3	LD	B,+08	Save the line number.
		PUSH	BC	It is called '8' times.
		CALL	0EF4,COPY-LINE	Fetch the line number.
		POP	BC	Jump back until '8' lines
		DJNZ	0ED3,COPY-3	have been printed.

Continue into the COPY-END routine.

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

THE 'CLEAR PRINTER BUFFER' SUBROUTINE

The printer buffer is cleared by calling this subroutine.

0EDF	CLEAR-PRB	LD	HL,+5B00	The base address of the printer
				buffer.
		LD	(PR-CC-lo),L	Reset the printer 'column'.
		XOR	A	Clear the A register.
		LD	B,A	Also clear the B register (in
				effect B holds dec.256).
0EE7	PRB-BYTES	LD	(HL),A	The '256' bytes of the
		INC	HL	printer buffer are all
		DJNZ	0EE7,PRB-BYTES	cleared in turn.
		RES	1,(FLAGS2)	Signal 'the buffer is empty'.
		LD	C,+21	Set the printer position and
		JP	0DD9,CL-SET	return via CL-SET & P0-STORE.

THE 'COPY-LINE' SUBROUTINE

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

0EF4	COPY-LINE	LD	A,B	Copy the pixel-line number.
		CP	+03	The A register will hold
		SBC	A,A	+00 until the last two lines
		AND	+02	are being handled.

OUT	(+FB),A	Slow the motor for the last two pixel lines only.
LD	D,A	The D register will hold either +00 or +02.

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

0EFD	COPY-L-1	CALL	1F54,BREAK-KEY	Jump forward unless the BREAK key is being pressed. But if it is then; stop the motor, enable the maskable interrupt, clear the printer buffer and exit via the error handling routine - 'BREAK-CONT repeats'. Fetch the status of the printer. Make an immediate return if the printer is not present. Wait for the stylus. There are thirty two bytes.
		JR	C,0F0C,COPY-L-2	
		LD	A,+04	
		OUT	(+FB),A	
		EI		
		CALL	0EDF,CLEAR-PRB	
		RST	0008,ERROR-1	
		DEFB	+0C	
0F0C	COPY-L-2	IN	A,(+FB)	
		ADD	A,A	
		RET	M	
		JR	NC,0EFD,COPY-L-1	
		LD	C,+20	

Now enter a loop to handle these bytes.

0F14	COPY-L-3	LD	E,(HL)	Fetch a byte.
		INC	HL	Update the pointer.
		LD	B,+08	Eight bits per byte.
0F18	COPY-L-4	RL	D	Move D left.
		RL	E	Move each bit into the carry.
		RR	D	Move D back again, picking up the carry from E.
0F1E	COPY-L-5	IN	A,(+FB)	Again fetch the status of the printer and wait for the signal from the encoder.
		RRA		Now go ahead and pass the 'bit' to the printer
		JR	NC,0F1E,COPY-L-5	Note: bit 2 - low starts the motor, bit 1 - high slows the motor and bit 7 is high for the actual 'printing'.
		LD	A,D	'Print' each bit.
		OUT	(+FB),A	Decrease the byte counter.
				Jump back whilst there are still bytes; otherwise return.
		DJNZ	0F18,COPY-L-4	
		DEC	C	
		JR	NZ,0F14,COPY-L-3	
		RET		

THE 'EDITOR' ROUTINES

The editor is called on two occasions:

- From the main execution routine so that the user can enter a BASIC line into the system.
- From the INPUT command routine.

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

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

A loop is now entered to handle each keystroke.

0F38	ED-LOOP	CALL	15D4,WAIT-KEY	Return once a key has been pressed.
		PUSH	AF	Save the code temporarily.

LD	D,+00	Fetch the duration of the
LD	E,(PIP)	keyboard click.
LD	HL,+00C8	And the pitch.
CALL	03B5,BEEPER	Now make the 'pip'.
POP	AF	Restore the code.
LD	HL,+0F38	Pre-load the machine stack
PUSH	HL	with the address of ED-LOOP.

Now analyse the code obtained.

CP	+18	Accept all character codes,
JR	NC,0F81,ADD-CHAR	graphic codes and tokens.
CP	+07	Also accept ','.
JR	C,0F81,ADD-CHAR	
CP	+10	Jump forward if the code
JR	C,0F92,ED-KEYS	represents an editing key.

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

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

AT & TAB would be handled as follows:

INC	BC	Three locations required.
BIT	7,(FLAGX)	Jump forward unless dealing
JP	Z,101E,ED-IGNORE	with INPUT LINE....
CALL	15D4,WAIT-KEY	Get the second code.
LD	E,A	and put it in E.

The other bytes for the control characters are now fetched.

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

THE 'ADDCHAR' SUBROUTINE

This subroutine actually adds a code to the current EDIT or INPUT line.

0F81	ADD-CHAR	RES	0,(MODE)	Signal 'K mode'.
		LD	HL,(K-CUR)	Fetch the cursor position.
		CALL	1652,ONE-SPACE	Make a single space.
0F8B	ADD-CH-1	LD	(DE),A	Enter the code into the space
		INC	DE	and signal that the cursor is to
		LD	(K-CUR),DE	occur at the location after. Then
		RET		return indirectly to ED-LOOP.

The editing keys are dealt with as follows:

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

PUSH	HL	routine is saved on the machine stack.
LD	HL,(K-CUR)	The HL register pair is set and an indirect jump made to the required routine.
RET		

THE 'EDITING KEYS' TABLE

address	offset	character	address	offset	character
0FA0	09	EDIT	0FA5	70	DELETE
0FA1	66	cursor left	0FA6	7E	ENTER
0FA2	6A	cursor right	0FA7	CF	SYMBOL SHIFT
0FA3	50	cursor down	0FA8	D4	GRAPHICS
0FA4	85	cursor up			

THE 'EDIT KEY' SUBROUTINE

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

0FA9	ED-EDIT	LD	HL,(E-PPC)	Fetch the current line number.
		BIT	5,(FLAGX)	But jump forward if in 'INPUT mode'.
		JP	NZ,1097,CLEAR-SP	Find the address of the start of the current line and hence its number.
		CALL	196E,LINE-ADDR	If the line number returned is zero then simply clear the editing area.
		CALL	1695,LINE-NO	Save the address of the line.
		LD	A,D	Move on to collect the length of the line.
		OR	E	
		JP	Z,1097,CLEAR-SP	
		PUSH	HL	
		INC	HL	
		LD	C,(HL)	
		INC	HL	
		LD	B,(HL)	
		LD	HL,+000A	Add +0A to the length and test that there is sufficient room for a copy of the line.
		ADD	HL,BC	
		LD	B,H	
		LD	C,L	
		CALL	1F05,TEST-ROOM	
		CALL	1097,CLEAR-SP	Now clear the editing area.
		LD	HL,(CURCHL)	Fetch the current channel address and exchange it for the address of the line.
		EX	(SP),HL	Save it temporarily.
		PUSH	HL	Open channel 'R' so that the line will be copied to the editing area.
		LD	A,+FF	Fetch the address of the line.
		CALL	1601,CHAN-OPEN	Goto before the line.
		POP	HL	Decrement the current line number so as to avoid printing the cursor.
		DEC	HL	Print the BASIC line
		DEC	(E-PPC-lo)	Increment the current line number.
		CALL	1855,OUT-LINE	Note: The decrementing of the line number does not always stop the cursor from being printed.
		INC	(E-PPC-lo)	Fetch the start of the line in the editing area and step past the line number and the length to find the address for K-CUR.
		LD	HL,(E-LINE)	
		INC	HL	
		INC	HL	
		INC	HL	
		INC	HL	

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

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

THE 'CURSOR DOWN EDITING' SUBROUTINE

0FF3	ED-DOWN	BIT	5,(FLAGX)
		JR	NZ,1001,ED-STOP
		LD	HL,+5C49
		CALL	190F,LN-FETCH
		JR	106E,ED-LIST
1001	ED-STOP	LD	(ERR-NR),+10
		JR	1024,ED-ENTER

Jump forward if in 'INPUT mode'. This is E-PPC. The next line number is found and a new automatic listing produced. 'STOP in INPUT' report. Jump forward.

THE 'CURSOR LEFT EDITING' SUBROUTINE

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

The cursor is moved. Jump forward.

THE 'CURSOR RIGHT EDITING' SUBROUTINE

100C	ED-RIGHT	LD	A,(HL)
		CP	+0D
		RET	Z
		INC	HL
1011	ED-CUR	LD	(K-CUR),HL
		RET	

The current character is tested and if it is 'carriage return' then return. Otherwise make the cursor come after the character. Set the system variable K-CUR.

THE 'DELETE EDITING' SUBROUTINE

1015	ED-DELETE	CALL	1031,ED-EDGE
		LD	BC,+0001
		JP	19E8,RECLAIM-2

Move the cursor leftwards. Reclaim the current character.

THE 'ED-IGNORE' SUBROUTINE

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

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

THE 'ENTER EDITING' SUBROUTINE

1024	ED-ENTER	POP	HL
		POP	HL
1026	ED-END	POP	HL
		LD	(ERR-SP),HL
		BIT	7,(ERR-NR)
		RET	NZ
		LD	SP,HL
		RET	

The address of ED-LOOP and ED-ERROR are discarded. The old value of ERR-SP is restored. Now return if there were no errors. Otherwise make an indirect jump to the error routine.

THE 'ED-EDGE' SUBROUTINE

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

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

DE will hold either E-LINE (for editing) or WORKSP (for INPUTing). The carry flag will become set if the cursor is already to be at the start of the line. Correct for the subtraction.

POP	BC	Drop the return address.
RET	C	Return via ED-LOOP if the carry flag is set.
PUSH	BC	Restore the return address.
LD	B,H	Move the current address of the cursor to BC.
LD	C,L	

Now enter a loop to check that control characters are not split from their parameters.

103E	ED-EDGE-1	LD	H,D	HL will point to the character in the line after that addressed by DE.
		LD	L,E	Fetch a character code.
		INC	HL	Jump forward if the code does not represent INK to TAB.
		LD	A,(DE)	Allow for one parameter.
		AND	+F0	Fetch the code anew.
		CP	+10	Carry is reset for TAB.
		JR	NZ,1051,ED-EDGE-2	Note: This splits off AT & TAB but AT & TAB in this form are not implemented anyway so it makes no difference.
		INC	HL	Jump forward unless dealing with AT & TAB which would have two parameters, if used.
		LD	A,(DE)	Prepare for true subtraction.
		SUB	+17	The carry flag will be reset when the 'updated pointer' reaches K-CUR.
		ADC	A,+00	For the next loop use the 'updated pointer', but if exiting use the 'present pointer' for K-CUR.
		JR	NZ,1051,ED-EDGE-2	Note: It is the control character that is deleted when using DELETE.
		INC	HL	
1051	ED-EDGE-2	AND	A	
		SBC	HL,BC	
		ADD	HL,BC	
		EX	DE,HL	
		JR	C,103E,ED-EDGE-1	
		RET		

THE 'CURSOR UP EDITING' SUBROUTINE

1059	ED-UP	BIT	5,(FLAGX)	Return if in 'INPUT mode'.
		RET	NZ	
		LD	HL,(E-PPC)	Fetch the current line number and its start address.
		CALL	196E,LINE-ADDR	HL now points to the previous line.
		EX	DE,HL	
		CALL	1695,LINE-NO	This line's number is fetched.
		LD	HL,+5C4A	This is E-PPC-hi.
		CALL	1910,LN-STORE	The line number is stored.
106E	ED-LIST	CALL	1795,AUTO-LIST	A new automatic listing is now produced and channel 'K' re-opened before returning to ED-LOOP.
		LD	A,+00	
		JP	1601,CHAN-OPEN	

THE 'ED-SYMBOL' SUBROUTINE

If SYMBOL & GRAPHICS codes were used they would be handled as follows:

1076	ED-SYMBOL	BIT	7,(FLAGX)	Jump back unless dealing with INPUT LINE.
		JR	Z,1024,ED-ENTER	
107C	ED-GRAPH	JP	0F81,ADD-CHAR	Jump back.

THE 'ED-ERROR' SUBROUTINE

Come here when there has been some kind of error.

107F	ED-ERROR	BIT	4,(FLAGS2)	Jump back if using other than channel 'K'.
		JR	Z,1026,ED-END	

LD	(ERR-NR),+FF
LD	D,+00
LD	E,(RASP)
LD	HL,+1A90
CALL	0385,BEEPER
JP	0F30,ED-AGAIN

Cancel the error number and give a 'rasp' before going around the editor again.

THE 'CLEAR-SP' SUBROUTINE

The editing area or the work space is cleared as directed.

1097	CLEAR-SP	PUSH	HL
		CALL	1190,SET-HL
		DEC	HL
		CALL	19E5,RECLAIM-1
		LD	(K-CUR),HL
		LD	(MODE),+00
		POP	HL
		RET	

Save the pointer to the space. DE will point to the first character and HL the last. The correct amount is now reclaimed. The system variables K-CUR and MODE ('K mode') are initialised before fetching the pointer and returning.

THE 'KEYBOARD INPUT' SUBROUTINE

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

10A8	KEY-INPUT	BIT	3,(TV-FLAG)
		CALL	NZ,111D,ED-COPY
		AND	A
		BIT	5,(FLAGS)
		RET	Z
		LD	A,(LAST-K)
		RES	5,(FLAGS)
		PUSH	AF
		BIT	5,(TV-FLAG)
		CALL	NZ,0D6E,CLS-LOWER
		POP	AF
		CP	+20
		JR	NC,111B,KEY-DONE
		CP	+10
		JR	NC,10FA,KEY-CONTR
		CP	+06
		JR	NC,10DB,KEY=M&CL

Copy the edit-linear the INPUT-line to the screen if the mode has changed. Return with both carry and zero flags reset if no new key has been pressed. Otherwise fetch the code and signal that it has been taken. Save the code temporarily. Clear the lower part of the display if necessary; e.g. after 'scroll?'; Fetch the code. Accept all characters and token codes. Jump forward with most of the control character codes. Jump forward with the 'mode' codes and the CAPS LOCK code.

Now deal with the FLASH, BRIGHT& INVERSE codes.

LD	B,A
AND	+01
LD	C,A
LD	A,B
RRA	
ADD	A,+12
JR	1105,KEY-DATA

Save the code. Keep only bit 0. C holds +00 (= OFF) or C holds +01 (= ON). Fetch the code. Rotate it once (losing bit 0). Increase it by +12 giving for FLASH - +12, BRIGHT - +13 and INVERSE - +14.

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

10DB	KEY-M&CL	JR	NZ,10E6,KEY-MODE
		LD	HL,+5C6A
		LD	A,+08
		XOR	(HL)
		LD	(HL),A
		JR	10F4,KEY-FLAG
10E6	KEY-MODE	CP	+0E

Jump forward with 'mode' codes. This is FLAGS2. Flip bit 3 of FLAGS2. This is the CAPS LOCK flag.

Jump forward. Check the lower limit.

		RET	C	
		SUB	+0D	Reduce the range.
		LD	HL,+5C41	This is MODE.
		CP	(HL)	Has it been changed?
		LD	(HL),A	Enter the new 'mode' code.
		JR	NZ,10F4,KEY-FLAG	Jump if it has changed;
		LD	(HL),+00	otherwise make it 'L mode'.
10F4	KEY-FLAG	SET	3,(TV-FLAG)	Signal 'the mode might have changed.
		CP	A	Reset the carry flag and
		RET		return.

The control key codes (apart from FLASH, BRIGHT & INVERSE) are manipulated.

10FA	KEY-CONTR	LD	B,A	Save the code.
		AND	+07	Make the C register hold the
		LD	C,A	parameter. (+00 to +07)
		LD	A,+10	A now holds the INK code.
		BIT	3,B	But if the code was an
		JR	NZ,1105,KEY-DATA	'unshifted' code then make A
		INC	A	hold the PAPER code.

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

1105	KEY-DATA	LD	(K-DATA),C	Save the parameter.
		LD	DE,+110D	This is KEY-NEXT.
		JR	1113,KEY-CHAN	Jump forward.

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

110D	KEY-NEXT	LD	A,(K-DATA)	Fetch the parameter.
		LD	DE,+10A8	This is KEY-INPUT.

Now set the input address in the first channel area.

1113	KEY-CHAN	LD	HL,(CHANS)	Fetch the channel address.
		INC	HL	
		INC	HL	
		LD	(HL),E	Now set the input address.
		INC	HL	
		LD	(HL),D	

Finally exit with the required code in the A register.

111B	KEY-DONE	SCF		Show a code has been found
		RET		and return.

THE 'LOWER SCREEN COPYING' SUBROUTINE

This subroutine is called whenever the line in the editing area or the INPUT area is to be printed in the lower part of the screen.

111D	ED-COPY	CALL	0D4D,TEMPS	Use the permanent colours.
		RES	3,(TV-FLAG)	Signal that the 'mode is to be
		RES	5,(TV-FLAG)	considered unchanged' and the
				'lower screen does not need
				clearing'.
		LD	HL,(S-POSNL)	Save the current value of
		PUSH	HL	S-POSNL.
		LD	HL,(ERR-SP)	Keep the current value of
		PUSH	HL	ERR-SP.
		LD	HL,+1167	This is ED-FULL.
		PUSH	HL	Push this address on to the
		LD	(ERR-SP),SP	machine stack to make ED-FULL
				the entry point following an
				error.

LD	HL,(ECHO-E)	Push the value of ECHO-E
PUSH	HL	on to the stack.
SCF		Make HL point to the start
CALL	1195,SET-HL	of the space and DE the end.
EX	DE,HL	
CALL	187D,OUT-LINE2	Now print the line.
EX	DE,HL	Exchange the pointers and
CALL	18E1,OUT-CURS	print the cursor.
LD	HL,(S-POSNL)	Next fetch the Current value
EX	(SP),HL	of S-POSNL and exchange it
		with ECHO-E.
EX	DE,HL	Pass ECHO-E to DE.
CALL	0D4D,TEMPS	Again fetch the permanent
		colours.

The remainder of any line that has been started is now completed with spaces printed with the 'permanent' PAPER colour.

1150	ED-BLANK	LD	A,(S-POSNL-hi)	Fetch the current line number
		SUB	D	and subtract the old line number.
		JR	C,117C,ED-C-DONE	Jump forward if no 'blanking'
				of lines required.
		JR	NZ,115E,ED-SPACES	Jump forward if not on the
				same line.
		LD	A,E	Fetch the old column number
		SUB	(S-POSNL-lo)	and subtract the new column
				number.
		JR	NC,117C,ED-C-DONE	Jump if no spaces required.
115E	ED-SPACES	LD	A,+20	A 'space'.
		PUSH	DE	Save the old values,
		CALL	09F4,PRINT-OUT	Print it.
		POP	DE	Fetch the old values.
		JR	1150,ED-BLANK	Back again.

New deal with any errors.

1167	ED-FULL	LD	D,+00	Give out a 'rasp'.
		LD	E,(RASP)	
		LD	HL,+1A90	
		CALL	03B5,BEEPER	
		LD	(ERR-NR),+FF	Cancel the error number.
		LD	DE,(S-POSNL)	Fetch the current value of
		JR	117E,ED-C-END	S-POSNL and jump forward.

The normal exit upon completion of the copying over of the editor the INPUT line.

117C	ED-C-DONE	POP	DE	The new position value.
		POP	HL	The 'error address'.

But come here after an error.

117E	ED-C-END	POP	HL	The old value of ERR-SP is
		LD	(ERR-SP),HL	restored.
		POP	BC	Fetch the old value of
				S-POSNL.
		PUSH	DE	Save the new position values.
		CALL	0DD9,CL-SET	Set the system variables.
		POP	HL	The old value of S-POSNL
		LD	(ECHO-E),HL	goes into ECHO-E.
		LD	(X-PTR-hi),+00	X-PTR is cleared in a
		RET		suitable manner and the return
				made.

THE 'SET-HL' AND 'SET-DE' SUBROUTINES

These subroutines return with HL pointing to the first location and DE the 'last' location of either the editing area or the work space.

1190	SET-HL	LD	HL,(WORKSP)	Point to the last location
------	--------	----	-------------	----------------------------

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

THE 'REMOVE-FP' SUBROUTINE

This subroutine removes the hidden floating-point forms in a BASIC line.

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

THE EXECUTIVE ROUTINES

THE 'INITIALISATION' ROUTINE

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

THE 'NEW COMMAND' ROUTINE

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

The main entry point.

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

Now the memory is checked.

11DA	RAM-CHECK	LD	H,D	Transfer the value in DE (START = +FFFF, NEW = RAMTOP).
		LD	L,E	
11DC	RAM-FILL	LD	(HL),+02	Enter the value of +02 into every location above +3FFF.
		DEC	HL	
		CP	H	
		JR	NZ,11DC,RAM-FILL	
11E2	RAM-READ	AND	A	Prepare for true subtraction. The carry flag will become reset when the top is reached.
		SBC	HL,DE	Update the pointer.
		ADD	HL,DE	
		INC	HL	
		JR	NC,11EF,RAM-DONE	Jump when at top.
		DEC	(HL)	+02 goes to +01.
		JR	Z,11EF,RAM-DONE	But if zero then RAM is faulty. Use current HL as top.
		DEC	(HL)	+01 goes to +00.
		JR	Z,11E2,RAM-READ	Step to the next test unless it fails.
11EF	RAM-DONE	DEC	HL	HL points to the last actual location in working order.

Next restore the 'preserved' system variables. (Meaningless when coming from START.)

EXX			Switch registers.
LD	(P-RAMT),BC		Restore P-RAMT,RASP/PIP &UDG
LD	(RASP/PIP),DE		
LD	(UDG),HL		
EXX			
INC	B		Test the START/NEW flag.
JR	Z,1219,RAM-SET		Jump forward if coming from the NEW command routine.

Overwrite the system variables when coming from START and initialise the user-defined graphics area.

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

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

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

Next the machine stack is set up.

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

The initialisation routine continues with:

IM	1	Interrupt mode 1 is used.
LD	IY,+5C3A	IY holds +ERR-NR always.
EI		The maskable interrupt can now be enabled. The real-time clock will be updated and the keyboard scanned every 1/50th of a second.
LD	HL,+5CB6	The base address of the
LD	(CHANS),HL	channel information area.
LD	DE,15AF	The initial channel data
LD	BC,+0015	is moved from the table
EX	DE,HL	(15AF) to the channel
LDIR		information area.
EX	DE,HL	The system variable DATADD
DEC	HL	is made to point to the last
LD	(DATADD),HL	location of the channel data.
INC	HL	And PROG & VARS to the
LD	(PROG),HL	the location after that.
LD	(VARS),HL	
LD	(HL),+80	The end-marker of the
		variables area.
INC	HL	Move on one location to find
LD	(E-LINE),HL	the value for E-LINE.
LF	(HL),+0D	Make the edit-line be a single
INC	HL	'carriage return' character.
LD	(HL),+80	Now enter an end-marker.
INC	HL	Move on one location to find
LD	(WORKSP),HL	the value for WORKSP, STKBOT
LD	(STKBOT),HL	& STKEND.

LD	(STKEND),HL	
LD	A,+38	Initialise the colour system
LD	(ATTR-P),A	variables to : FLASH 0,
LD	(ATTR-T),A	BRIGHT 0, PAPER 7, & INK 0.
LD	(BORDCR),A	
LD	HL,+0523	Initialise the system
LD	(REPDEL),HL	variables REPDEL & REPPER.
DEC	(KSTATE-0)	Make KSTATE-0 hold +FF
DEC	(KSTATE-4)	Make KSTATE-4 hold +FF
LD	HL,+15C6	Next move the initial stream
LD	DE,+5C10	data from its table to the
LD	BC,+000E	streams area.
LDIR		
SET	1,(FLAGS)	Signal 'printer in use'
CALL	0EDF,CLEAR-PRB	and clear the printer buffer.
LD	(DF-SZ),+02	Set the size of the lower
CALL	0D6B,CLS	part of the display and clear
		the whole display.
XOR	A	Now print the message
LD	DE,+1538	'© 1982 Sinclair Research Ltd'
CALL	0C0A,PO-MSG	on the bottom line.
SET	5,(TV-FLAG)	Signal 'the lower part will
		required to be cleared.
JR	12A9,MAIN-1	Jump forward into the main
		execution loop.

THE 'MAIN EXECUTION' LOOP

The main loop extends from location 12A2 to location 15AE and it controls the 'editing mode', the execution of direct commands and the production of reports.

12A2	MAIN-EXEC	LD	(DF-SZ),+02	The lower part of the screen
				is to be two lines in size.
		CALL	1795,AUTO-LIST	Produce an automatic listing.
12A9	MAIN-1	CALL	16B0,SET-MIN	All the areas from E-LINE
				onwards are given their
				minimum configurations.
12AC	MAIN-2	LD	A,+00	Channel 'K' is opened before
		CALL	1601,CHAN-OPEN	calling the EDITOR.
		CALL	0F2C,EDITOR	The EDITOR is called to allow
				the user to build up a BASIC line.
		CALL	1B17,LINE-SCAN	The current line is scanned for
				correct syntax.
		BIT	7,(ERR-NR)	Jump forward if the syntax is
		JR	NZ,12CF,MAIN-3	correct.
		BIT	4,(FLAGS2)	Jump forward if other than
		JR	Z,1303,MAIN-4	channel 'K' is being used.
		LD	HL,(E-LINE)	Point to the start of the line
				with the error.
		CALL	11A7,REMOVE-FP	Remove the floating-point
				forms from this line.
		LD	(ERR-NR),+FF	Reset ERR-NR and jump back
		JR	12AC,MAIN-2	to MAIN-2 leaving the listing
				unchanged.

The 'edit-line' has passed syntax and the three types of line that are possible have to be distinguished from each other.

12CF	MAIN-3	LD	HL,(E-LINE)	Point to the start of the line.
		LD	(CH-ADD),HL	Set CH-ADD to the start also.
		CALL	19FB,E-LINE-NO	Fetch any line number into BC.
		LD	A,B	Is the line number a valid
		OR	C	one?
		JR	NZ,155D,MAIN-ADD	Jump if it is so, and add the new
				line to the existing program.

RST	0018	Fetch the first character of
CP	+0D	the line and see if the line is
JR	Z,12A2,MAIN-EXEC	'carriage return only'. If it is then jump back.

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

BIT	0,(FLAGS2)	Clear the whole display unless
CALL	NZ,0DAF,CL-ALL	the flag says it is unnecessary.
CALL	0D6E,CLS-LOWER	Clear the lower part anyway.
LD	A,+19	Set the appropriate value
SUB	(S-POSN-hi)	for the scroll counter.
LD	(SCR-CT),A	
SET	7,(FLAGS)	Signal 'line execution'.
LD	(ERR-NR),+FF	Ensure ERR-NR is correct.
LD	(NSPPC),+01	Deal with the first statement in
		the line.
CALL	1B8A,PROG-RUN	Now the line is interpreted.
		Note: The address 1303 goes on
		to the machine stack and is
		addressed by ERR-SP.

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

1303	MAIN-4	HALT		The maskable interrupt must be enabled.
		RES	5,(FLAGS)	Signal 'ready for a new key'.
		BIT	1,(FLAGS2)	Empty the printer buffer if
		CALL	NZ,0ECD,COPY-BUFF	it has been used.
		LD	A,(ERR-NR)	Fetch the error number and
		INC	A	increment it.
1313	MAIN-G	PUSH	AF	Save the new value.
		LD	HL,+0000	The system variables
		LD	(FLAGX),H	FLAGX, X-PTR-hi &
		LD	(X-PTR-hi),H	DEFADD are all set to zero.
		LD	(DEFADD),HL	
		LD	HL,+0001	Ensure that stream +00
		LD	(STRMS-6),HL	points to channel 'K'
		CALL	16B0,SET-MIN	Clear all the work areas and the
				calculator stack.
		RES	5,(FLAGX)	Signal 'editing mode'.
		CALL	0D6E,CLS-LOWER	Clear the lower screen.
		SET	5,(TV-FLAG)	Signal 'the lower screen will
				require clearing'.
		POP	AF	Fetch the report value.
		LD	B,A	Make a copy in B.
		CP	+0A	Jump forward with report
		JR	C,133C,MAIN-5	numbers '0 to 9'.
		ADD	A,+07	Add the ASCII letter
				offset value.
133C	MAIN-5	CALL	15EF,OUT-CODE	Print the report code and
		LD	A,+20	follow it with a 'space'.
		RST	0010,PRINT-A-1	
		LD	A,B	Fetch the report value and
		LD	DE,+1391	use it to identify the
		CALL	0C0A,PO-MSG	required report message.
		XOR	A	Print the message and follow
		LD	DE,+1536	it by a 'comma' and a 'space'.
		CALL	0C0A,PO-MSG	
		LD	BC,(PPC)	Now fetch the current line
		CALL	1A1B,OUT-NUM1	number and print it as well.
		LD	A,+3A	Follow it by a ':'
		RST	0010,PRINT-A-1	

		LD	C,(SUBPPC)	Fetch the current statement
		LD	B,+00	number into the BC register
		CALL	1A1B,OUT-NUM1	pair and print it.
		CALL	1097,CLEAR-SP	Clear the editing area.
		LD	A,(ERR-NR)	Fetch the error number again.
		INC	A	Increment it as usual.
		JR	Z,1386,MAIN-9	If the program was completed
				successfully there cannot be
				any 'CONTInuing' so jump.
		CP	+09	If the program halted with
		JR	Z,1373,MAIN-6	'STOP statement' or 'BREAK
		CP	+15	into program' CONTInuing will
		JR	NZ,1376,MAIN-7	be from the next statement;
1373	MAIN-6	INC	(SUBPPC)	otherwise SUBPPC is unchanged.
1376	MAIN-7	LD	BC,+0003	The system variables OLDPPC
		LD	DE,+5C70	& OSPCC have now to be made
				to hold the CONTInuing line
				and statement numbers.
		LD	HL,+5C44	The values used will be those in
		BIT	7,(NSPPC)	PPC & SUBPPC unless NSPPC
		JR	Z,1384,MAIN-8	indicates that the 'break'
		ADD	HL,BC	occurred before a 'jump'.
1384	MAIN-8	LDDR		(i.e. after a GO TO statement
				etc.)
1386	MAIN-9	LD	(NSPPC),+FF	NSPPC is reset to indicate
				'no jump'.
		RES	3,(FLAGS)	'K mode' is selected.
		JP	12AC,MAIN-2	And finally the jump back is
				made but no program listing
				will appear until requested.

THE REPORT MESSAGES

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

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

There are also the following two messages.

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

Report G - No room for line

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

THE 'MAIN-ADD' SUBROUTINE

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

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

		EX	DE,HL	Destination into HL & number into DE.
		POP	BC	Fetch the new line's length.
		LD	(HL),B	The high length byte.
		DEC	HL	
		LD	(HL),C	The low length byte.
		DEC	HL	
		LD	(HL),E	The low line number byte.
		DEC	HL	
		LD	(HL),D	The high line number byte.
15AB	MAIN-ADD2	POP	AF	Drop the address of REPORT-G.
		JP	12A2,MAIN-EXEC	Jump back and this time do produce and automatic listing.

THE 'INITIAL CHANNEL INFORMATION'

Initially there are four channels - 'K', 'S', 'R', & 'P' - for communicating with the 'keyboard', 'screen', 'work space' and 'printer'. For each channel the output routine address comes before the input routine address and the channel's code.

15AF	DEFB	F4 09	- PRINT-OUT
	DEFB	A8 10	- KEY-INPUT
	DEFB	4B	- 'K'
15B4	DEFB	F4 09	- PRINT-OUT
	DEFB	C4 15	- REPORT-J
	DEFB	53	- 'S'
15B9	DEFB	81 0F	- ADD-CHAR
	DEFB	C4 15	- REPORT-J
	DEFB	52	- 'R'
15BE	DEFB	F4 09	- PRINT-OUT
	DEFB	C4 15	- REPORT-J
	DEFB	50	- 'P'
15C3	DEFB	80	- End marker.

Report J - Invalid I/O device

15C4	REPORT-J	RST	0008,ERROR-1	Call the error handling
		DEFB	+12	routine

THE 'INITIAL STREAM DATA'

Initially there are seven streams - +FD to +03.

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

THE 'WAIT-KEY' SUBROUTINE

This subroutine is the controlling subroutine for calling the current input subroutine.

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

Report 8 - End of file

15E4	REPORT-8	RST	0008,ERROR-1	Call the error handling
		DEFB	+07	routine.

THE 'INPUT-AD' SUBROUTINE

The registers are saved and HL made to point to the input address.

15E6	INPUT-AD	EXX		Save the registers.
		PUSH	HL	
		LD	HL,(CURCHL)	Fetch the base address for the
				current channel information.
		INC	HL	Step past the output address.
		INC	HL	
		JR	15F7,CALL-SUB	Jump forward.

THE 'MAIN PRINTING' SUBROUTINE

The subroutine is called with either an absolute value or a proper character code in the A register.

15EF	OUT-CODE	LD	E,+30	Increase the value in the
		ADD	A,E	A register by +30.
15F2	PRINT-A-2	EXX		Again save the registers.
		PUSH	HL	
		LD	HL,(CURCHL)	Fetch the base address for the
				current channel. This will point
				to an output address.

Now call the actual subroutine. HL points to the output or the input address as directed.

15F7	CALL-SUB	LD	E,(HL)	Fetch the low byte.
		INC	HL	
		LD	D,(HL)	Fetch the high byte.
		EX	DE,HL	Move the address to the HL
				register pair.
		CALL	162C,CALL-JUMP	Call the actual subroutine.
		POP	HL	Restore the registers.
		EXX		
		RET		Return will be from here
				unless an error occurred.

THE 'CHAN-OPEN' SUBROUTINE

This subroutine is called with the A register holding a valid stream number - normally +FD to +03. Then depending on the stream data a particular channel will be made the current channel.

1601	CHAN-OPEN	ADD	A,A	The value in the A register is
		ADD	A,+16	doubled and then increased by
		LD	L,A	+16. The result is moved to L.
		LD	H,+5C	The address 5C16 is the base
				address for stream +00.
		LD	E,(HL)	Fetch the first byte of the
		INC	HL	required stream's data; then
		LD	D,(HL)	the second byte.
		LD	A,D	Give an error if both bytes
		OR	E	are zero; otherwise jump
		JR	NZ,1610,CHAN-OP-1	forward.

Report O - Invalid stream

160E	REPORT-O	RST	0008,ERROR-1	Call the error handling
		DEFB	+17	routine.

Using the stream data now find the base address of the channel information associated with that stream.

1610	CHAN-OP-1	DEC	DE	Reduce the stream data.
		LD	HL,(CHANS)	The base address of the whole
				channel information area.

ADD HL,DE

Form the required address in this area.

THE 'CHAN-FLAG' SUBROUTINE

The appropriate flags for the different channels are set by this subroutine.

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

THE 'CHANNEL CODE LOOK-UP' TABLE

162D	DEFB	4B 06	- channel 'K',	offset +06,	address 1634
162F	DEFB	53 12	- channel 'S',	offset +12,	address 1642
1631	DEFB	50 1B	- channel 'P',	offset +1B,	address 164D
1633	DEFB	00	- end marker.		

THE 'CHANNEL 'K' FLAG' SUBROUTINE

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

THE 'CHANNEL 'S' FLAG' SUBROUTINE

1642	CHAN-S	RES	0,(TV-FLAG)	Signal 'using main screen'.
1646	CHAN-S-1	RES	1,(FLAGS)	Signal 'printer not being used'.
		JP	0D4D,TEMPS	Exit via TEMPS so as to set the colour system variables.

THE 'CHANNEL 'P' FLAG' SUBROUTINE

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

THE 'MAKE-ROOM' SUBROUTINE

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

1652	ONE-SPACE	LD	BC,+0001	Just the single extra location is required.
1655	MAKE-ROOM	PUSH	HL	Save the pointer.
		CALL	1F05,TEST-ROOM	Make sure that there is sufficient memory available for the task being undertaken.
		POP	HL	Restore the pointer.

CALL	1664,POINTERS	Alter all the pointers before making the 'room'.
LD	HL,(STKEND)	Make HL hold the new STKEND.
EX	DE,HL	Switch 'old' and 'new'.
LDDR		Now make the 'room'
RET		and return.

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

THE 'POINTERS' SUBROUTINE

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

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

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

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

Now find the size of the block to be moved.

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

ADD	HL,DE	Reform the old value of
EX	DE,HL	STKEND and pass it to DE
RET		before returning.

THE 'COLLECT A LINE NUMBER' SUBROUTINE

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

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

The usual entry point is at LINE-NO.

1695	LINE-NO	LD	A,(HL)	Fetch the high byte and
		AND	+C0	test it.
		JR	NZ,1691,LINE-NO-A	Jump back if not suitable.
		LD	D,(HL)	Fetch the high byte.
		INC	HL	
		LD	E,(HL)	Fetch the low byte and
		RET		return.

THE 'RESERVE' SUBROUTINE

This subroutine is normally called by using RST 0030,BC-SPACES.

On entry here the last value on the machine stack is WORKSP and the value above it is the number of spaces that is to be 'reserved'. This subroutine always makes 'room' between the existing work space and the calculator stack.

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

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

THE 'SET-MIN' SUBROUTINE

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

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

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

16BF	SET-WORK	LD	HL,(WORKSP)	Fetch the WORKSP.
		LD	(STKBOT),HL	This clears the work space.

Entering here will 'clear' only the calculator stack.

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

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

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

THE 'RECLAIM THE EDIT-LINE' SUBROUTINE

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

THE 'INDEXER' SUBROUTINE

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

16DB	INDEXER-1	INC	HL	Move on to consider the next pair of entries.
16DC	INDEXER	LD	A,(HL)	Fetch the first of a pair of entries but return if it is zero - the end marker.
		AND	A	
		RET	Z	Compare it to the supplied code.
		CP	C	
		INC	HL	Point to the second entry.
		JR	NZ,16DB,INDEXER-1	Jump back if the correct entry has not been found.
		SCF		The carry flag is set upon a successful search.
		RET		

THE 'CLOSE #' COMMAND ROUTINE

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

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

THE 'CLOSE-2' SUBROUTINE

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

1701	CLOSE-2	PUSH	HL	Save the address of the stream's data.
------	---------	------	----	--

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

THE 'CLOSE STREAM LOOK-UP' TABLE

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

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

THE 'CLOSE STREAM' SUBROUTINE.

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

THE 'STREAM DATA' SUBROUTINE

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

171E	STR-DATA	CALL	1E94,STK-TO-A	The given stream number is
				taken off the calculator stack.
		CP	+10	Give an error if the stream
		JR	C,1727,STR-DATA1	number is greater than +0F.

Report O - Invalid stream

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

Continue with valid stream numbers.

1727	STR-DATA1	ADD	A,+03	Range now +03 to +12;
		RLCA		and now +06 to +24.
		LD	HL,+5C10	The base address of the
				stream data area.
		LD	C,A	Move the stream code to the
		LD	B,+00	BC register pair.
		ADD	HL,BC	Index into the data area
		LD	C,(HL)	and fetch the two data bytes
		INC	HL	into the BC register pair.
		LD	B,(HL)	
		DEC	HL	Make the pointer address the
		RET		first of the data bytes before
				returning.

THE 'OPEN #' COMMAND ROUTINE

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

1736	OPEN	RST	0028,FP-CALC	Use the CALCULATOR.
		DEFB	+01,exchange	Exchange the stream number

		DEFB	+38,end-calc	and the channel code.
		CALL	171E,STR-DATA	Fetch the data for the stream.
		LD	A,B	Jump forward if both bytes of
		OR	C	the data are zero, i.e. the
		JR	Z,1756,OPEN-1	stream was in a closed state.
		EX	DE,HL	Save DE.
		LD	HL,(CHANS)	Fetch CHANS - the base
		ADD	HL,BC	address of the channel
		INC	HL	information and find the
		INC	HL	code of the channel
		INC	HL	associated with the stream
		LD	A,(HL)	being OPENed.
		EX	DE,HL	Return DE.
		CP	+4B	The code fetched from the
		JR	Z,1756,OPEN-1	channel information area
		CP	+53	must be 'K', 'S' or 'P';
		JR	Z,1756,OPEN-1	give an error if it is not.
		CP	+50	
		JR	NZ,1725,REPORT-O	
1756	OPEN-1	CALL	175D,OPEN-2	Collect the appropriate data
		LD	(HL),E	in DE.
		INC	HL	Enter the data into the
		LD	(HL),D	two bytes in the stream
		RET		information area.
				Finally return.

THE 'OPEN-2' SUBROUTINE

The appropriate stream data bytes for the channel that is associated with the stream being OPENed are found.

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

Report F - Invalid file name

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

Continue if no error occurred.

1767	OPEN-3	PUSH	BC	The length of the expression
				is saved.
		LD	A,(DE)	Fetch the first character.
		AND	+DF	Convert lower case codes to
				upper case ones.
		LD	C,A	Move code to the C register.
		LD	HL,+177A	The base address of the
				'OPEN stream look-up' table.
		CALL	16DC,INDEXER	Index into this table and locate
				the required offset.
		JR	NC,1765,REPORT-F	Jump back if not found.
		LD	C,(HL)	Pass the offset to the BC
		LD	B,+00	register pair.
		ADD	HL,BC	Make HL point to the start of
				the appropriate subroutine.
		POP	BC	Fetch the length of the
		JP	(HL)	expression before jumping to
				the subroutine.

THE 'OPEN STREAM LOOK-UP' TABLE

177A	DEFB	4B 06	- channel 'K', offset +06, address 1781
------	------	-------	---

177C	DEFB	53 08	- channel 'S', offset +08, address 1785
177E	DEFB	50 0A	- channel 'P', offset +0A, address 1789
1780	DEFB	00	- end marker;

THE 'OPEN-K' SUBROUTINE

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

THE 'OPEN-S' SUBROUTINE

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

THE 'OPEN-P' SUBROUTINE

1789	OPEN-P	LD	E,+10	The data bytes will be +10
				& +00.
178B	OPEN-END	DEC	BC	Decrease the length of the
		LD	A,B	expression and give an error
		OR	C	if it was not a single
		JR	NZ,1765,REPORT-F	character; otherwise clear the
		LD	D,A	D register, fetch HL and
		POP	HL	return.
		RET		

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

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

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

THE 'LIST & LLIST' COMMAND ROUTINES

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

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

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

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

	PUSH	DE	Save the 'automatic' number.
	CALL	196E,LINE-ADDR	Find the address of the
	LD	DE,+02C0	start of the 'current' line
	EX	DE,HL	and produce an address roughly

SBC	HL,DE	a 'screen before it' (negated).
EX	(SP),HL	Save the 'result' on the machine
CALL	196E,LINE-ADDR	stack whilst the 'automatic' line
		address is also found (in HL).
POP	BC	The 'result' goes to the BC
		register pair.

A loop is now entered. The 'automatic' line number is increased on each pass until it is likely that the 'current' line will show on a listing.

17CE	AUTO-L-1	PUSH CALL	BC 19B8,NEXT-ONE	Save the 'result'. Find the address of the start of the line after the present 'automatic' line (in DE).
		POP ADD JR EX LD INC LD DEC LD JR	BC HL,BC C,17E4,AUTO-L-3 DE,HL D,(HL) HL E,(HL) HL (S-TOP),DE 17CE,AUTO-L-1	Restore the 'result'. Perform the computation and jump forward if finished. Move the next line's address to the HL register pair and collect its line number.
				Now S-TOP can be updated and the test repeated with the new line.

Now the 'automatic' listing can be made.

17E1	AUTO-L-2	LD	(S-TOP),HL	When E-PPC is less than S-TOP.
17E4	AUTO-L-3	LD CALL JR EX	HL,(S-TOP) 196E,LINE-ADDR Z,17ED,AUTO-L-4 DE,HL	Fetch the top line's number and hence its address. If the line cannot be found use DE instead.
17ED	AUTO-L-4	CALL RES RET	1833,LIST-ALL 4,(TV-FLAG)	The listing is produced. The return will be to here unless scrolling was needed to show the current line.

THE 'LLIST' ENTRY POINT

The printer channel will need to be opened.

17F5	LLIST	LD JR	A,+03 17FB,LIST-1	Use stream +03. Jump forward.
------	-------	----------	----------------------	----------------------------------

THE 'LIST' ENTRY POINT

The 'main screen' channel will need to be opened.

17F9	LIST	LD	A,+02	Use stream +02.
17FB	LIST-1	LD	(TV-FLAG),+00	Signal 'an ordinary listing in the main part of the screen'.
		CALL CALL RST CALL	2530,SYNTAX-Z NZ,1601,CHAN-OPEN 0018,GET-CHAR 2070,STR-ALTER	Open the channel unless checking syntax. With the present character in the A register see if the stream is to be changed.
		JR RST CP JR CP JR	C,181F,LIST-4 0018,GET-CHAR +3B Z,1814,LIST-2 +2C NZ,181A,LIST-3	Jump forward if unchanged. Is the present character a ','? Jump if it is. Is it a ','? Jump if it is not.
1814	LIST-2	RST CALL JR	0020,NEXT-CHAR 1C82,EXPT-1NUM 1822,LIST-5	A numeric expression must follow, e.g. LIST #5,20 Jump forward with it.
181A	LIST-3	CALL JR	1CE6,USE-ZERO 1822,LIST-5	Otherwise use zero and also jump forward.

Come here if the stream was unaltered.

181F	LIST-4	CALL	1CDE,FETCH-NUM	Fetch any line or use zero if none supplied.
1822	LIST-5	CALL	1BEE,CHECK-END	If checking the syntax of the edit-line move on to the next statement.
		CALL	1E99,FIND-INT	Line number to BC.
		LD	A,B	High byte to A.
		AND	+3F	Limit the high byte to the correct range and pass the whole line number to HL.
		LD	H,A	Set E-PPC and find the address of the start of this line or the first line after it if the actual line does not exist.
		LD	L,C	
		LD	(E-PPC),HL	
		CALL	196E,LINE-ADDR	
1833	LIST-ALL	LD	E,+01	Flag 'before the current line'.

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

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

THE 'PRINT A WHOLE BASIC LINE' SUBROUTINE

The HL register pair points to the start of the line - the location holding the high byte of the line number.

Before the line number is printed it is tested to determine whether it comes before the 'current' line, is the 'current' line or comes after.

1855	OUT-LINE	LD	BC,(E-PPC)	Fetch the 'current' line number and compare it.
		CALL	1980,CP-LINES	Pre-load the D register with the current line cursor.
		LD	D,+3E	Jump forward if printing the 'current' line.
		JR	Z,1865,OUT-LINE1	Load the D register with zero (it is not the cursor) and set E to hold +01 if the line is before the 'current' line and +00 if after. (The carry flag comes from CP-LINES.)
		LD	DE,+0000	Save the line marker.
		RL	E	Fetch the high byte of the line number and make a full return if the listing has been finished.
1865	OUT-LINE1	LD	(BREG),E	The line number can now be printed - with leading spaces.
		LD	A,(HL)	
		CP	+40	
		POP	BC	
		RET	NC	
		PUSH	BC	
		CALL	1A28,OUT-NUM-2	

		INC	HL	Move the pointer on to address
		INC	HL	the first command code in
		INC	HL	the line.
		RES	0,(FLAGS)	Signal 'leading space allowed'
		LD	A,D	Fetch the cursor code and
		AND	A	jump forward unless the
		JR	Z,1881,OUT-LINE3	cursor is to be printed.
		RST	0010,PRINT-A-1	So print the cursor now.
187D	OUT-LINE2	SET	0,(FLAGS)	Signal 'no leading space now'.
1881	OUT-LINE3	PUSH	DE	Save the registers.
		EX	DE,HL	Move the pointer to DE.
		RES	2,(FLAGS2)	Signal 'not in quotes'.
		LD	HL,+5C3B	This is FLAGS.
		RES	2,(HL)	Signal 'print in K-mode'.
		BIT	5,(FLAGX)	Jump forward unless in
		JR	1894,OUT-LINE4	INPUT mode.
		SET	2,(HL)	Signal 'print in L-mode'.

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

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

The line has now been printed.

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

THE 'NUMBER' SUBROUTINE

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

18B6	NUMBER	CP	+0E	Is the character a 'number
		RET	NZ	marker'. Return if not.
		INC	HL	Advance the pointer six
		INC	HL	times so as to step past the
		INC	HL	'number marker' and the five
		INC	HL	locations holding the
		INC	HL	floating-point form.
		INC	HL	
		LD	A,(HL)	Fetch the current code before
		RET		returning.

THE 'PRINT A FLASHING CHARACTER' SUBROUTINE

The 'error cursor' and the 'mode cursors' are printed using this subroutine.

18C1	OUT-FLASH	EXX		Save the current register.
		LD	HL,(ATTR-T)	Save the ATTR-T & MASK-T on
		PUSH	HL	the machine stack.
		RES	7,H	Ensure that FLASH is
		SET	7,L	active.
		LD	(ATTR-T),HL	Use these modified values
				for ATTR-T & MASK-T.
		LD	HL,+5C91	This is P-FLAG.
		LD	D,(HL)	Save P-FLAG also on the
		PUSH	DE	machine stack.
		LD	(HL),+00	Ensure INVERSE 0, OVER 0,
				and not PAPER 9 nor INK 9.
		CALL	09F4,PRINT-OUT	The character is printed.
		POP	HL	The former value of P-FLAG
		LD	(P-FLAG),H	is restored.
		POP	HL	The former values of ATTR-T
		LD	(ATTR-T),HL	& MASK-T are also restored
		EXX		before returning.
		RET		

THE 'PRINT THE CURSOR' SUBROUTINE

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

18E1	OUT-CURS	LD	HL,(K-CUR)	Fetch the address of the
		AND	A	cursor but return if the
		SBC	HL,DE	correct place is not being
		RET	NZ	considered.
		LD	A,(MODE)	The current value of MODE is
		RLC	A	fetches and doubled.
		JR	Z,18F3,OUT-C-1	Jump forward unless dealing with
				Extended mode or Graphics.
		ADD	A,+43	Add the appropriate offset to
				give 'E' or 'G'.
		JR	1909,OUT-C-2	Jump forward to print it.
18F3	OUT-C-1	LD	HL,+5C3B	This is FLAGS.
		RES	3,(HL)	Signal 'K-mode'.
		LD	A,+4B	The character 'K'.
		BIT	2,(HL)	Jump forward to print 'K'.
		JR	Z,1909,OUT-C-2	If 'the printing is to be in
				K-mode'.
		SET	3,(HL)	The 'printing is to be in L-mode'
				so signal 'L-MODE'.
		INC	A	Form the character 'L'.
		BIT	3,(FLAGS2)	Jump forward if not in
		JR	Z,1909,OUT-C-2	'C-mode'.
		LD	A,+43	The character 'C'.
1909	OUT-C-2	PUSH	DE	Save the DE register pair
		CALL	18C1,OUT-FLASH	whilst the cursor is printed
		POP	DE	- FLASHing.
		RET		Return once it has been done.

Note: It is the action of considering which cursor-letter is to be printed that determines the mode - 'K' vs. 'L/C'.

THE 'LN-FETCH' SUBROUTINE

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

190F	LN-FETCH	LD	E,(HL)	The line number held by the
		INC	HL	system variable is collected.

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

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

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

THE 'PRINTING CHARACTERS IN A BASIC LINE' SUBROUTINE

All of the character/token codes in a BASIC line are printed by repeatedly calling this subroutine.

The entry point OUT-SP-NO is used when printing line numbers which may require leading spaces.

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

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

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

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

1937	OUT-CHAR	CALL	2D1B,NUMERIC	Return carry reset if handling a digit code.
		JR	NC,196C,OUT-CH-3	Jump forward to print the digit.
		CP	+21	Also print the control
		JR	C,196C,OUT-CH-3	characters and 'space'.
		RES	2,(FLAGS)	Signal 'print in K-mode'.
		CP	+CB	Jump forward if dealing
		JR	Z,196C,OUT-CH-3	with the token 'THEN'.
		CP	+3A	Jump forward unless dealing
		JR	NZ,195A,OUT-CH-1	with ':'.
		BIT	5,(FLAGX)	Jump forward to print the
		JR	NZ,1968,OUT-CH-2	'.' if in 'INPUT mode'.
		BIT	2,(FLAGS2)	Jump forward if the ':'
		JR	Z,196C,OUT-CH-3	is 'not in quotes', i.e.
				an inter-statement marker.
		JR	1968,OUT-CH-2	The ':' is inside quotes and can now be printed.

195A	OUT-CH-1	CP JR PUSH	+22 NZ,1968,OUT-CH-2 AF	Accept for printing all characters except "". Save the character code whilst changing the 'quote mode'. Fetch FLAGS2 and flip bit 2.
		LD XOR LD POP	A,(FLAGS2) +04 (FLAGS2),A AF	Enter the amended value and restore the character code.
1968	OUT-CH-2	SET	2,(FLAGS)	Signal 'the next character is to be printed in L-mode'.
196C	OUT-CH-3	RST RET	0010,PRINT-A-1	The present character is printed before returning.

Note: It is the consequence of the tests on the present character that determines whether the next character is to be "printed in 'K' or 'L' mode".

Also note how the program does not cater for ':' in REM statements.

THE 'LINE-ADDR' SUBROUTINE

For a given line number, in the HL register pair, this subroutine returns the starting address of that line or the 'first line after', in the HL register pair, and the start of the previous line in the DE register pair.

If the line number is being used the zero flag will be set. However if the 'first line after' is substituted then the zero flag is returned reset.

196E	LINE-ADDR	PUSH LD LD LD	HL HL,(PROG) D,H E,L	Save the given line number. Fetch the system variable PROG and transfer the address to the DE register pair.
------	-----------	------------------------	-------------------------------	---

Now enter a loop to test the line number of each line of the program against the given line number until the line number is matched or exceeded.

1974	LINE-AD-1	POP CALL	BC 1980,CP-LINES	The given line number. Compare the given line number against the addressed line number. Return if carry reset; otherwise address the next line's number.
		RET PUSH CALL EX JR	NC BC 19B8,NEXT-ONE DE,HL 1974,LINE-AD-1	Switch the pointers and jump back to consider the next line of the program.

THE 'COMPARE LINE NUMBERS' SUBROUTINE

The given line number in the BC register pair is matched against the addressed line number.

1980	CP-LINES	LD CP RET	A,(HL) B NZ	Fetch the high byte of the addressed line number and compare it. Return if they do not match.
		INC LD DEC CP RET	HL A,(HL) HL C	Next compare the low bytes. Return with the carry flag set if the addressed line number has yet to reach the given line number.

THE 'FIND EACH STATEMENT' SUBROUTINE

This subroutine has two distinct functions.

- I. It can be used to find the 'D'th. statement in a BASIC line - returning with the HL register pair addressing the location before the start of the statement and the zero flag set.
- II. Also the subroutine can be used to find a statement, if any, that starts with a given token code (in the E register).

1988		INC	HL	Not used.
		INC	HL	
		INC	HL	
198B	EACH-STMT	LD	(CH-ADD),HL	Set CH-ADD to the current byte.
		LD	C,+00	Set a 'quotes off' flag.

Enter a loop to handle each statement in the BASIC line.

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

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

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

THE 'NEXT-ONE' SUBROUTINE

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

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

		INC	HL	long name is reached.
		LD	A,(HL)	Increment the pointer and
		JR	NC,19CE,NEXT-O-2	fetch the new code.
				Jump back unless the previous
				code was the last code of the
				variable's name.
		JR	19DB,NEXT-O-5	Now jump forward (BC =
19D5	NEXT-O-3	INC	HL	+0005 or +0012).
				Step past the low byte of the
19D6	NEXT-O-4	INC	HL	line number.
				Now point to the low byte
		LD	C,(HL)	of the length.
		INC	HL	Fetch the length into the
		LD	B,(HL)	BC register pair.
		INC	HL	
				Allow for the inclusive byte.

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

19DB	NEXT-O-5	ADD	HL,BC	Point to the first byte of the
				'next' line or variable.
		POP	DE	Fetch the address of the
				previous one and continue into
				the 'difference' subroutine.

THE 'DIFFERENCE' SUBROUTINE

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

19DD	DIFFER	AND	A	Prepare for a true subtraction.
		SBC	HL,DE	Find the length from one
		LD	B,H	'start' to the next and pass
		LD	C,L	it to the BC register pair.
		ADD	HL,DE	Reform the address and
		EX	DE,HL	exchange them before
		RET		returning.

THE 'RECLAIMING' SUBROUTINE

The entry point RECLAIM-1 is used when the address of the first location to be reclaimed is in the DE register pair and the address of the first location to be left alone is in the HL register pair. The entry point RECLAIM-2 is used when the HL register pair points to the first location to be reclaimed and the BC register pair holds the number of the bytes that are to be reclaimed.

19E5	RECLAIM-1	CALL	19DD,DIFFER	Use the 'difference' subroutine
				to develop the appropriate
				values.
19E8	RECLAIM-2	PUSH	BC	Save the number of bytes to be
				reclaimed.
		LD	A,B	All the system variable
		CPL		pointers above the area
		LD	B,A	have to be reduced by 'BC'
		LD	A,C	so this number is 2's
		CPL		complemented before the
		LD	C,A	pointers are altered.
		INC	BC	
		CALL	1664,POINTERS	
		EX	DE,HL	Return the 'first location'
		POP	HL	address to the DE register
		ADD	HL,DE	pair and reform the address of
				the first location to the left.
		PUSH	DE	Save the 'first location'
		LDIR		whilst the actual reclamation
		POP	HL	occurs.
		RET		Now return.

THE 'E-LINE-NO' SUBROUTINE

This subroutine is used to read the line number of the line in the editing area. If there is no line number, i.e. a direct BASIC line, then the line number is considered to be zero.

In all cases the line number is returned in the BC register pair.

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

THE 'REPORT AND LINE NUMBER PRINTING' SUBROUTINE

The entry point OUT-NUM-1 will lead to the number in the BC register pair being printed. Any value over '9,999' will not however be printed correctly.

The entry point OUT-NUM-2 will lead to the number indirectly addressed by the HL register pair being printed. This time any necessary leading spaces will appear. Again the limit of correctly printed numbers is '9,999'.

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

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

1A30	OUT-NUM-3	LD	BC,+FC18	This is '-1,000'.
		CALL	192A,OUT-SP-NO	Print a first digit.
		LD	BC,+FF9C	This is '-100'.
		CALL	192A,OUT-SP-NO	Print the second digit.
		LD	C,+F6	This is '-10'.
		CALL	192A,OUT-SP-NO	Print the third digit.

		LD	A,L	Move any remaining part of the number to the A register.
1A42	OUT-NUM-4	CALL	15EF,OUT-CODE	Print the digit.
		POP	HL	Restore the registers
		POP	DE	before returning.
		RET		

BASIC LINE AND COMMAND INTERPRETATION

THE SYNTAX TABLES

i. The offset table

There is an offset value for each of the fifty BASIC commands.

			command	address				command	address
1A48	DEFB	+B1	DEF FN	1AF9	1A61	DEFB	+94	BORDER	1AF5
1A49	DEFB	+CB	CAT	1B14	1A62	DEFB	+56	CONTINUE	1AB8
1A4A	DEFB	+BC	FORMAT	1B06	1A63	DEFB	+3F	DIM	1AA2
1A4B	DEFB	+BF	MOVE	1B0A	1A64	DEFB	+41	REM	1AA5
1A4C	DEFB	+C4	ERASE	1B10	1A65	DEFB	+2B	FOR	1A90
1A4D	DEFB	+AF	OPEN #	1AFC	1A66	DEFB	+17	GO TO	1A7D
1A4E	DEFB	+B4	CLOSE #	1B02	1A67	DEFB	+1F	GO SUB	1A86
1A4F	DEFB	+93	MERGE	1AE2	1A68	DEFB	+37	INPUT	1A9F
1A50	DEFB	+91	VERIFY	1AE1	1A69	DEFB	+77	LOAD	1AE0
1A51	DEFB	+92	BEEP	1AE3	1A6A	DEFB	+44	LIST	1AAE
1A52	DEFB	+95	CIRCLE	1AE7	1A6B	DEFB	+0F	LET	1A7A
1A53	DEFB	+98	INK	1AEB	1A6C	DEFB	+59	PAUSE	1AC5
1A54	DEFB	+98	PAPER	1AEC	1A6D	DEFB	+2B	NEXT	1A98
1A55	DEFB	+98	FLASH	1AED	1A6E	DEFB	+43	POKE	1AB1
1A56	DEFB	+98	BRIGHT	1AEE	1A6F	DEFB	+2D	PRINT	1A9C
1A57	DEFB	+98	INVERSE	1AEF	1A70	DEFB	+51	PLOT	1AC1
1A58	DEFB	+98	OVER	1AF0	1A71	DEFB	+3A	RUN	1AAB
1A59	DEFB	+98	OUT	1AF1	1A72	DEFB	+6D	SAVE	1ADF
1A5A	DEFB	+7F	LPRINT	1AD9	1A73	DEFB	+42	RANDOMIZE	1AB5
1A5B	DEFB	+81	LLIST	1ADC	1A74	DEFB	+0D	IF	1A81
1A5C	DEFB	+2E	STOP	1A8A	1A75	DEFB	+49	CLS	1ABE
1A5D	DEFB	+6C	READ	1AC9	1A76	DEFB	+5C	DRAW	1AD2
1A5E	DEFB	+6E	DATA	1ACC	1A77	DEFB	+44	CLEAR	1ABB
1A5F	DEFB	+70	RESTORE	1ACF	1A78	DEFB	+15	RETURN	1A8D
1A60	DEFB	+48	NEW	1AA8	1A79	DEFB	+5D	COPY	1AD6

ii. The parameter table

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

1A7A	P-LET	DEFB +01	CLASS-01
		DEFB +3D	'='
		DEFB +02	CLASS-02
1A7D	P-GO-TO	DEFB +06	CLASS-06
		DEFB +00	CLASS-00
		DEFB +67,+1E	GO-TO,1E67
1A81	P-IF	DEFB +06	CLASS-06
		DEFB +CB	'THEN'
		DEFB +05	CLASS-05
		DEFB +F0,+1C	IF,1CF0
1A86	P-GO-SUB	DEFB +06	CLASS-06
		DEFB +00	CLASS-00
		DEFB +ED,+1E	GO-SUB,1EED
1A8A	P-STOP	DEFB +00	CLASS-00
		DEFB +EE,+1C	STOP,1CEE
1A8D	P-RETURN	DEFB +00	CLASS-00
		DEFB +23,+1F	RETURN,1F23
1A90	P-FOR	DEFB +04	CLASS-04
		DEFB +3D	'='
		DEFB +06	CLASS-06
		DEFB +CC	'TO'
		DEFB +06	CLASS-06
		DEFB +05	CLASS-05
		DEFB +03,+1D	FOR,1D03

1A98	P-NEXT	DEFB +04	CLASS-04
		DEFB +00	CLASS-00
		DEFB +AB,+1D	NEXT,1DAB
1A9C	P-PRINT	DEFB +05	CLASS-05
		DEFB +CD,+1F	PRINT,1FCD
1A9F	P-INPUT	DEFB +05	CLASS-05
		DEFB +89,+20	INPUT,2089
1AA2	P-DIM	DEFB +05	CLASS-05
		DEFB +02,+2C	DIM,2C02
1AA5	P-REM	DEFB +05	CLASS-05
		DEFB +B2,+1B	REM,1BB2
1AA8	P-NEW	DEFB +00	CLASS-00
		DEFB +B7,+11	NEW,11B7
1AAB	P-RUN	DEFB +03	CLASS-03
		DEFB +A1,+1E	RUN,1EA1
1AAE	P-LIST	DEFB +05	CLASS-05
		DEFB +F9,+17	LIST,17F9
1AB1	P-POKE	DEFB +08	CLASS-08
		DEFB +00	CLASS-00
		DEFB +80,+1E	POKE,1E80
1AB5	P-RANDOM	DEFB +03	CLASS-03
		DEFB +4F,+1E	RANDOMIZE,1E4F
1AB8	P-CONT	DEFB +00	CLASS-00
		DEFB +5F,+1E	CONTINUE,1E5F
1ABB	P-CLEAR	DEFB +03	CLASS-03
		DEFB +AC,+1E	CLEAR,1EAC
1ABE	P-CLS	DEFB +00	CLASS-00
		DEFB +6B,+0D	CLS,0D6B
1AC1	P-PLOT	DEFB +09	CLASS-09
		DEFB +00	CLASS-00
		DEFB +DC,+22	PLOT,22DC
1AC5	P-PAUSE	DEFB +06	CLASS-06
		DEFB +00	CLASS-00
		DEFB +3A,+1F	PAUSE,1F3A
1AC9	P-READ	DEFB +05	CLASS-05
		DEFB +ED,+1D	READ,1DED
1ACC	P-DATA	DEFB +05	CLASS-05
		DEFB +27,+1E	DATA,1E27
1ACF	P-RESTORE	DEFB +03	CLASS-03
		DEFB +42,+1E	RESTORE,1E42
1AD2	P-DRAW	DEFB +09	CLASS-09
		DEFB +05	CLASS-05
		DEFB +82,+23	DRAW,2382
1AD6	P-COPY	DEFB +00	CLASS-00
		DEFB +AC+0E	COPY,0EAC
1AD9	P-LPRINT	DEFB +05	CLASS-05
		DEFB +C9,+1F	LPRINT,1FC9
1ADC	P-LLIST	DEFB +05	CLASS-05
		DEFB +F5,+17	LLIST,17F5
1ADF	P-SAVE	DEFB +0B	CLASS-0B
1AE0	P-LOAD	DEFB +0B	CLASS-0B
1AE1	P-VERIFY	DEFB +0B	CLASS-0B
1AE2	P-MERGE	DEFB +0B	CLASS-0B
1AE3	P-BEEP	DEFB +08	CLASS-08
		DEFB +00	CLASS-00
		DEFB +F8,+03	BEEP,03F8
1AE7	P-CIRCLE	DEFB +09	CLASS-09
		DEFB +05	CLASS-05
		DEFB +20,+23	CIRCLE,2320
1AEB	P-INK	DEFB +07	CLASS-07
1AEC	P-PAPER	DEFB +07	CLASS-07
1AED	P-FLASH	DEFB +07	CLASS-07
1AEE	P-BRIGHT	DEFB +07	CLASS-07

1AEF	P-INVERSE	DEFB +07	CLASS-07
1AF0	P-OVER	DEFB +07	CLASS-07
1AF1	P-OUT	DEFB +08	CLASS-08
		DEFB +00	CLASS-00
		DEFB +7A,+1E	OUT,1E7A
1AF5	P-BORDER	DEFB +06	CLASS-06
		DEFB +00	CLASS-00
		DEFB +94,+22	BORDER,2294
1AF9	P-DEF-FN	DEFB +05	CLASS-05
		DEFB +60,+1F	DEF-FN,1F60
1AFC	P-OPEN	DEFB +06	CLASS-06
		DEFB +2C	','
		DEFB +0A	CLASS-0A
		DEFB +00	CLASS-00
		DEFB +36,+17	OPEN,1736
1B02	P-CLOSE	DEFB +06	CLASS-06
		DEFB +00	CLASS-00
		DEFB +E5,+16	CLOSE,16E5
1B06	P-FORMAT	DEFB +0A	CLASS-0A
		DEFB +00	CLASS-00
		DEFB +93,+17	CAT-ETC,1793
1B0A	P-MOVE	DEFB +0A	CLASS-0A
		DEFB +2C	','
		DEFB +0A	CLASS-0A
		DEFB +00	CLASS-00
		DEFB +93,+17	CAT-ETC,1793
1B10	P-ERASE	DEFB +0A	CLASS-0A
		DEFB +00	CLASS-00
		DEFB +93,+17	CAT-ETC,1793
1B14	P-CAT	DEFB +00	CLASS-00
		DEFB +93,+17	CAT-ETC,1793

Note: The requirements for the different command classes are as follows:

CLASS-00 - No further operands.
 CLASS-01 - Used in LET. A variable is required.
 CLASS-02 - Used in LET. An expression, numeric or string, must follow.
 CLASS-03 - A numeric expression may follow. Zero to be used in case of default.
 CLASS-04 - A single character variable must follow.
 CLASS-05 - A set of items may be given.
 CLASS-06 - A numeric expression must follow.
 CLASS-07 - Handles colour items.
 CLASS-08 - Two numeric expressions, separated by a comma, must follow.
 CLASS-09 - As for CLASS-08 but colour items may precede the expressions.
 CLASS-0A - A string expression must follow.
 CLASS-0B - Handles cassette routines.

THE 'MAIN PARSER' OF THE BASIC INTERPRETER

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

Each statement is considered in turn and the system variable CH-ADD is used to point to each code of the statement as it occurs in the program area or the editing area.

1B17	LINE-SCAN	RES	7,(FLAGS)	Signal 'syntax checking'.
		CALL	19FB,E-LINE-NO	CH-ADD is made to point to the first code after any line number.
		XOR	A	The system variable SUBPPC
		LD	(SUBPPC),A	is initialised to +00 and
		DEC	A	ERR-NR to +FF.
		LD	(ERR-NR),A	
		JR	1B29,STMT-L-1	Jump forward to consider the first statement of the line.

THE STATEMENT LOOP.

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

1B28	STMT-LOOP	RST	0020,NEXT-CHAR	Advance CH-ADD along the line.
1B29	STMT-L-1	CALL	16BF,SET-WORK	The work space is cleared.
		INC	(SUBPPC)	Increase SUBPPC on each passage around the loop.
		JP	M,1C8A,REPORT-C	But only '127' statements are allowed in a single line.
		RST	0018,GET-CHAR	Fetch a character.
		LD	B,+00	Clear the register for later.
		CP	+0D	Is the character a 'carriage return'; jump if it is.
		JR	Z,1BB3,LINE-END	
		CP	+3A	Go around the loop again if it is a '.'
		JR	Z,1B28,STMT-LOOP	

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

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

Each of the command class routines applicable to the present command are executed in turn. Any required separators are also considered.

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

RST	0018,GET-CHAR	Before making an indirect jump to the command class routine pass the command code to the A register and set the B register to +FF.
DEC	B	
RET		

THE 'SEPARATOR' SUBROUTINE

The report - 'Nonsense in BASIC is given if the required separator is not present. But note that when syntax is being checked the actual report does not appear on the screen - only the 'error marker'.

1B6F	SEPARATOR	RST	0018,GET-CHAR	The current character is fetched and compared to the entry in the parameter table. Give the error report if there is not a match. Step past a correct character and return.
		CP	C	
		JP	NZ,1C8A,REPORT-C	
		RST	0020,NEXT-CHAR	
		RET		

THE 'STMT-RET' SUBROUTINE

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

1B76	STMT-RET	CALL	1F54,BREAK-KEY	The BREAK key is tested after every statement. Jump forward unless it has been pressed.
		JR	C,1B7D,STMT-R-1	

Report L - 'BREAK into program'

1B7B	REPORT-L	RST	0008,ERROR-1	Call the error handling routine.
		DEFB	+14	

Continue here as the BREAK key was not pressed.

1B7D	STMT-R-1	BIT	7,(NSPPC)	Jump forward if there is not a 'jump' to be made. Fetch the 'new line' number and jump forward unless dealing with a further statement in the editing area.
		JR	NZ,1BF4,STMT-NEXT	
		LD	HL,(NEWPPC)	
		BIT	7,H	
		JR	Z,1B9E,LINE-NEW	

THE 'LINE-RUN' ENTRY POINT

This entry point is used wherever a line in the editing area is to be 'run'. In such a case the syntax/run flag (bit 7 of FLAGS) will be set.

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

1B8A	LINE-RUN	LD	HL,+FFFE	A line in the editing area is considered as line '-2'. Make HL point to the end marker of the editing area and DE to the location before the start of that area. Fetch the number of the next statement to be handled before jumping forward.
		LD	(PPC),HL	
		LD	HL,(WORKSP)	
		DEC	HL	
		LD	DE,(E-LINE)	
		DEC	DE	
		LD	A,(NSPPC)	
		JR	1BD1,NEXT-LINE	

THE 'LINE-NEW' SUBROUTINE

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

1B9E	LINE-NEW	CALL	196E,LINE-ADDR	The starting address of the line, or the 'first line after' is found. Collect the statement number. Jump forward if the required line was found; otherwise
		LD	A,(NSPPC)	
		JR	Z,1BBF,LINE-USE	
		AND	A	

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

Report 0 - 'OK'

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

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

THE 'REM' COMMAND ROUTINE

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

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

THE 'LINE-END' ROUTINE

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

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

THE 'LINE-USE' ROUTINE

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

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

THE 'NEXT-LINE' ROUTINE

On entry the HL register pair points to the location after the end of the 'next' line to be handled and the DE register pair to the location before the first character of the line. This applies to lines in the program area and also to a line in the editing area - where the next line will be the same line again whilst there are still statements to be interpreted.

1BD1	NEXT-LINE	LD EX	(NXTLIN),HL DE,HL	Set NXTLIN for use once the current line has been completed. As usual CH-ADD points to the
------	-----------	--------------	--------------------------	--

LD	(CH-ADD),HL	location before the first character to be considered.
LD	D,A	The statement number is fetched.
LD	E,+00	The E register is cleared in case EACH-STMT is used.
LD	(NSPPC),+FF	Signal 'no jump'.
DEC	D	The statement number minus one goes into SUBPPC.
LD	(SUBPPC),D	A first statement can now be considered.
JP	Z,1B28,STMT-LOOP	However for later statements the 'starting address' has to be found.
INC	D	Jump forward unless the statement does not exist.
CALL	198B,EACH-STMT	
JR	Z,1BF4,STMT-NEXT	

Report N - 'Statement lost'

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

THE 'CHECK-END' SUBROUTINE

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

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

THE 'STMT-NEXT' ROUTINE

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

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

THE 'COMMAND CLASS' TABLE

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

THE 'COMMAND CLASSES - 00, 03 & 05'

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

1C0D	CLASS-03	CALL	1CDE,FETCH-NUM	A number is fetched but zero is used in cases of default.
------	----------	------	----------------	---

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

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

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

1C11	CLASS-05	POP	BC	In all cases drop the address - SCAN-LOOP.
		CALL	Z,1BEE,CHECK-END	If handling commands of classes 00 & 03 AND syntax is being checked move on now to consider the next statement.
		EX	DE,HL	Save the line pointer in the DE register pair.

THE 'JUMP-C-R' ROUTINE

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

1C16	JUMP-C-R	LD	HL,(T-ADDR)	Fetch the pointer to the
		LD	C,(HL)	entries in the parameter table
		INC	HL	and fetch the address of the
		LD	B,(HL)	required command routine.
		EX	DE,HL	Exchange the pointers back
		PUSH	BC	and make an indirect jump
		RET		to the command routine.

THE 'COMMAND CLASSES - 01, 02 & 04'

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

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

THE 'VARIABLE IN ASSIGNMENT' SUBROUTINE

This subroutine develops the appropriate values for the system variables DEST & STRLEN.

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

Report 2 - Variable not found

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

Continue with the handling of existing variables.

1C30	VAR-A-2	CALL	Z,2996,STK-VARS	The parameters of simple string variables and all array variables are passed to the calculator stack. (STK-VARS will 'slice' a string if required.)
		BIT	6,(FLAGS)	Jump forward if handling a numeric variable.
		JR	NZ,1C46,VAR-A-3	Clear the A register.
		XOR	A	The parameters of the string of string array variable are fetched unless syntax is being checked.
		CALL	2530,SYNTAX-Z	This is FLAGX.
		CALL	NZ,2BF1,STK-FETCH	
		LD	HL,+5C71	

OR	(HL)	Bit 0 is set only when handling complete 'simple strings' thereby signalling 'old copy to be deleted'.
LD	(HL),A	
EX	DE,HL	HL now points to the string or the element of the array.

The pathways now come together to set STRLEN & DEST as required. For all numeric variables and 'new' string & string array variables STRLEN-lo holds the 'letter' of the variable's name. But for 'old' string & string array variables whether 'sliced' or complete it holds the 'length' in 'assignment'.

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

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

LD	(DEST),HL	Set DEST as required and return.
RET		

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

1C4E CLASS-02 POP BC The address - SCAN-LOOP is dropped.

 CALL 1C56,VAL-FET-1 The assignment is made.

 CALL 1BEE,CHECK-END Move on to the next statement either via CHECK-END if checking syntax, or STMT-RET if in 'run-time'.

 RET

THE 'FETCH A VALUE' SUBROUTINE

This subroutine is used by LET, READ & INPUT statements to first evaluate and then assign values to the previously designated variable.

The entry point VAL-FET-1 is used by LET & READ and considers FLAGS whereas the entry point VAL-FET-2 is used by INPUT and considers FLAGX.

1C56 VAL-FET-1 LD A,(FLAGS) Use FLAGS.

1C59 VAL-FET-2 PUSH AF Save FLAGS or FLAGX.

 CALL 24FB,SCANNING Evaluate the next expression.

 POP AF Fetch the old FLAGS or FLAGX.

 LD D,(FLAGS) Fetch the new FLAGS.

 XOR D The nature - numeric or string of the variable and the expression must match.

 AND +40 Give report C if they do not.

 JR NZ,1C8A,REPORT-C Jump forward to make the actual assignment unless checking syntax when simply return.

 BIT 7,D

 JP NZ,2AFF,LET

 RET

THE 'COMMAND CLASS 04' ROUTINE

The command class 04 entry point is used by FOR & NEXT statements.

1C6C CLASS-04 CALL 28B2,LOOK-VARS Look in the variables area for the variable being used.

 PUSH AF Save the AF register pair whilst the discriminator byte is tested to ensure that the variable is a FOR-NEXT control variable.

 LD A,C

 OR +9F

 INC A

 JR NZ,1C8A,REPORT-C

 POP AF Restore the flags register and jump back to make the variable that has been found the 'variable in assignment'.

 JR 1C22,VAR-A-1

THE 'EXPECT NUMERIC/STRING EXPRESSIONS' SUBROUTINE

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

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

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

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

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

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

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

Report C - Nonsense in BASIC

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

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

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

THE 'SET PERMANENT COLOURS' SUBROUTINE (EQU. CLASS-07)

This subroutine allows for the current temporary colours to be made permanent. As command class 07 it is in effect the command routine for the six colour item commands.

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

The following instructions cleverly copy the even bits of the supplied byte to the odd bits. In effect making the permanent bits the same as the temporary ones.

RLCA		Move the mask leftwards.
XOR	(HL)	Impress onto the mask
AND	+AA	only the even bits of the
XOR	(HL)	other byte.
LD	(HL),A	Restore the result.
RET		

THE 'COMMAND CLASS 09' ROUTINE

This routine is used by PLOT, DRAW & CIRCLE statements in order to specify the default conditions of 'FLASH 8; BRIGHT 8; PAPER 8;' that are set up before any embedded colour items are considered.

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

THE 'COMMAND CLASS 0B' ROUTINE

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

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

THE 'FETCH A NUMBER' SUBROUTINE

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

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

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

1CE6	USE-ZERO	CALL	2530,SYNTAX-Z	Do not perform the operation
		RET	Z	if syntax is being checked.
		RST	0028,FP-CALC	Use the calculator.
		DEFB	+A0,stk-zero	The 'last value' is now zero.
		DEFB	+38,end-calc	
		RET		Return with zero added to the
				stack.

THE COMMAND ROUTINES

The section of the 16K monitor program from 1CEE to 23FA contains most of the command routines of the BASIC interpreter.

THE 'STOP' COMMAND ROUTINE

The command routine for STOP contains only a call to the error handling routine.

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

THE 'IF' COMMAND ROUTINE

On entry the value of the expression between the IF and the THEN is the 'last value' on the calculator stack. If this is logically true then the next statement is considered; otherwise the line is considered to have been finished.

1CF0	IF	POP	BC	Drop the return address - STMT-RET.
		CALL JR	2530,SYNTAX-Z Z,1D00,IF-1	Jump forward if checking syntax.

Now use the calculator to 'delete' the last value on the calculator stack but leave the DE register pair addressing the first byte of the value.

		RST DEFB DEFB EX CALL JP	0028,FP-CALC +02,delete +38,end-calc DE,HL 34E9,TEST-ZERO C,1BB3,LINE-END	Use the calculator. The present 'last value' is deleted. Make HL point to the first byte and call TEST-ZERO. If the value was 'FALSE' jump to the next line.
1D00	IF-1	JP	1B29,STMT-L-1	But if 'TRUE' jump to the next statement (after the THEN).

THE 'FOR' COMMAND ROUTINE

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

1D03	FOR	CP JR RST CALL CALL JR	+CD NZ,1D10,F-USE-1 0020,NEXT-CHAR 1C82,EXPT-1NUM 1BEE,CHECK-END 1D16,F-REORDER	Jump forward unless a 'STEP' is given. Advance CH-ADD and fetch the value of the STEP. Move on to the next statement if checking syntax; otherwise jump forward.
------	-----	---------------------------------------	--	--

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

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

The three values on the calculator stack are the VALUE (v), the LIMIT (l) and the STEP (s). These values now have to be manipulated.

1D16	F-REORDER	RST DEFB DEFB DEFB DEFB DEFB DEFB	0028,FP-CALC +C0,st-mem-0 +02,delete +01,exchange +E0,get-mem-0 +01,exchange +38,end-calc	v, l, s v, l, s (mem-0 = s) v, l l, v l, v, s l, s, v
------	-----------	---	---	--

A FOR control variable is now established and treated as a temporary calculator memory area.

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

The variable that has been found may be a simple numeric variable using only six locations in which case it will need extending.

DEC	HL	Fetch the variable's single
LD	A,(HL)	character name.
SET	7,(HL)	Ensure bit 7 of the name is set.
LD	BC,+0006	It will have six locations at least.
ADD	HL,BC	Make HL point after them.
RLCA		Rotate the name and jump if
JR	C,1D34,F-L&S	it was already a FOR variable.
LD	C,+0D	Otherwise create thirteen
CALL	1655,MAKE-ROOM	more locations.
INC	HL	Again make HL point to the
		LIMIT position.

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

1D34	F-L&S	PUSH	HL	The pointer is saved.
		RST	0028,FP-CALC	I, s
		DEFB	+02,delete	I
		DEFB	+02,delete	-
		DEFB	+38,end-calc	DE still points to 'I'.
		POP	HL	The pointer is restored and
		EX	DE,HL	both pointers exchanged.
		LD	C,+0A	The ten bytes of the LIMIT
		LDIR		and the STEP are moved.

The looping line number and statement number are now entered.

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

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

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

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

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

RST	0020,NEXT-CHAR	Advance past the NEXT that was found.
OR	+20	Allow for upper and lower case letters before the new variable name is tested.
CP	B	
JR	Z,1D7C,F-FOUND	Jump forward if it matches.
RST	0020,NEXT-CHAR	Advance CH-ADD again and
JR	1D64,F-LOOP	jump back if not the correct variable.

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

1D7C	F-FOUND	RST	0020,NEXT-CHAR	Advance CH-ADD.
		LD	A,+01	The statement counter in the D register counted statements back from zero so it has to be subtracted from '1'.
		SUB	D	The result is stored.
		LD	(NSPPC),A	Now return - to STMT-RET.
		RET		

REPORT I - FOR without NEXT

1D84	REPORT-I	RST	0008,ERROR-1	Call the error handling routine.
		DEFB	+11	

THE 'LOOK-PROG' SUBROUTINE

This subroutine is used to find occurrences of either DATA, DEF FN or NEXT. On entry the appropriate token code is in the E register and the HL register pair points to the start of the search area.

1D86	LOOK-PROG	LD	A,(HL)	Fetch the present character.
		CP	+3A	Jump forward if it is a ' : ' which will indicate there are more statements in the present line.
		JR	Z,1DA3,LOOK-P-2	

Now a loop is entered to examine each further line in the program.

1D8B	LOOK-P-1	INC	HL	Fetch the high byte of the line number and return with carry set if there are no further lines in the program.
		LD	A,(HL)	
		AND	+CO	
		SCF		
		RET	NZ	The line number is fetched and passed to NEWPPC.
		LD	B,(HL)	
		INC	HL	
		LD	C,(HL)	
		LD	(NEWPPC),BC	Then the length is collected.
		INC	HL	
		LD	C,(HL)	
		INC	HL	
		LD	B,(HL)	
		PUSH	HL	The pointer is saved whilst the address of the end of the line is formed in the BC register pair.
		ADD	HL,BC	The pointer is restored.
		LD	B,H	Set the statement counter to zero.
		LD	C,L	
		POP	HL	
		LD	D,+00	
1DA3	LOOK-P-2	PUSH	BC	The end-of-line pointer is saved whilst the statements of the line are examined.
		CALL	198B,EACH-STMT	Make a return if there was an 'occurrence'; otherwise consider the next line.
		POP	BC	
		RET	NC	
		JR	1D8B,LOOK-P-1	

THE 'NEXT' COMMAND ROUTINE

The 'variable in assignment' has already been determined (see CLASS-04,1C6C); and it remains to change the VALUE as required.

1DAB	NEXT	BIT	1,(FLAGX)	Jump to give the error report
		JP	NZ,1C2E,REPORT-2	if the variable was not found.
		LD	HL,(DEST)	The address of the variable
		BIT	7,(HL)	is fetched and the name
		JR	Z,1DD8,REPORT-1	tested further.

Next the variable's VALUE and STEP are manipulated by the calculator.

INC	HL	Step past the name.
LD	(MEM),HL	Make the variable a
		temporary 'memory area'.
RST	0028,FP-CALC	-
DEFB	+E0,get-mem-0	v
DEFB	+E2,get-mem-2	v, s
DEFB	+0F,addition	v+s
DEFB	+C0,st-mem-0	v+s
DEFB	+02,delete	-
DEFB	+38,end-calc	-

The result of adding the VALUE and the STEP is now tested against the LIMIT by calling NEXT-LOOP.

CALL	1DDA,NEXT-LOOP	Test the new VALUE against
		the LIMIT
RET	C	Return now if the FOR-NEXT
		loop has been completed.

Otherwise collect the 'looping' line number and statement.

LD	HL,(MEM)	Find the address of the
LD	DE,+000F	low byte of the looping
ADD	HL,DE	line number.
LD	E,(HL)	Now fetch this line number.
INC	HL	
LD	D,(HL)	
INC	HL	
LD	H,(HL)	Followed by the statement
		number.
EX	DE,HL	Exchange the numbers before
JP	1E73,GO-TO-2	jumping forward to treat them
		as the destination line of a
		GO TO command.

Report 1 - NEXT without FOR

1DD8	REPORT-1	RST	0008,ERROR-1	Call the error handling
		DEFB	+00	routine.

THE 'NEXT-LOOP' SUBROUTINE

This subroutine is used to determine whether the LIMIT has been exceeded by the present VALUE. Note has to be taken of the sign of the STEP.

The subroutine returns the carry flag set if the LIMIT is exceeded.

1DDA	NEXT-LOOP	RST	0028,FP-CALC	-
		DEFB	+E1,get-mem-1	I
		DEFB	+E0,get-mem-0	I, v
		DEFB	+E2,get-mem-2	I, v, s
		DEFB	+36,less-0	I, v,(1/0)
		DEFB	+00,jump-true	I, v,(1/0)
		DEFB	+02,to NEXT-1	I, v,(1/0)
		DEFB	+01,exchange	v, I
1DE2	NEXT-1	DEFB	+03,subtract	v-I or I-v
		DEFB	+37,greater-0	(1/0)
		DEFB	+00,jump-true	(1/0)

DEFB	+04,to NEXT-2	-
DEFB	+38,end-calc	-
AND	A	Clear the carry flag and
RET		return - loop is possible.

However if the loop is impossible the carry flag has to be set.

1DE9	NEXT-2	DEFB	+38,end-calc	-
		SCF		Set the carry flag and
		RET		return.

THE 'READ' COMMAND ROUTINE

The READ command allows for the reading of a DATA list and has an effect similar to a series of LET statements.

Each assignment within a single READ statement is dealt with in turn. The system variable X-PTR is used as a storage location for the pointer to the READ statement whilst CH-ADD is used to step along the DATA list.

1DEC	READ-3	RST	0020,NEXT-CHAR	Come here on each pass, after the first, to move along the READ statement.
1DED	READ	CALL	1C1F,CLASS-01	Consider whether the variable has been used before; find the existing entry if it has.
		CALL	2530,SYNTAX-Z	Jump forward if checking syntax.
		JR	Z,1E1E,READ-2	Save the current pointer CH-ADD in X-PTR.
		RST	0018,GET-CHAR	Fetch the current DATA list pointer and jump forward unless a new DATA statement has to be found.
		LD	(X-PTR),HL	The search is for 'DATA'.
		LD	HL,(DATADD)	Jump forward if the search is successful.
		LD	A,(HL)	
		CP	+2C	
		JR	Z,1E0A,READ-1	
		LD	E,+E4	
		CALL	1D86,LOOK-PROG	
		JR	NC,1E0A,READ-1	

Report E - Out of DATA

1E08	REPORT-E	RST	0008,ERROR-1	Call the error handling routine.
		DEFB	+0D	

Continue - picking up a value from the DATA list.

1E0A	READ-1	CALL	0077,TEMP-PTR1	Advance the pointer along the DATA list and set CH-ADD.
		CALL	1C56,VAL-FET-1	Fetch the value and assign it to the variable.
		RST	0018,GET-CHAR	Fetch the current value of CH-ADD and store it in DATADD.
		LD	(DATADD),HL	Fetch the pointer to the READ statement and clear X-PTR.
		LD	HL,(X-PTR)	Make CH-ADD once again point to the READ statement.
		LD	(X-PTR-hi),+00	GET the present character and see if it is a ','.
		CALL	0078,TEMP-PTR2	If it is then jump back as there are further items; otherwise return either via CHECK-END (if checking syntax) or the RET instruction (to STMT-RET).
1E1E	READ-2	RST	0018,GET-CHAR	
		CP	+2C	
		JR	Z,1DEC,READ-3	
		CALL	1BEE,CHECK-END	
		RET		

THE 'DATA' COMMAND ROUTINE

During syntax checking a DATA statement is checked to ensure that it contains a series of valid expressions, separated by commas. But in 'run-time' the statement is passed by.

1E27	DATA	CALL	2530,SYNTAX-Z	Jump forward unless checking syntax.
		JR	NZ,1E37,DATA-2	
A loop is now entered to deal with each expression in the DATA statement.				
1E2C	DATA-1	CALL	24FB,SCANNING	Scan the next expression.
		CP	+2C	Check for the correct separator - a ',';
		CALL	NZ,1BEE,CHECK-END	but move on to the next statement if not matched.
		RST	0020,NEXT-CHAR	Whilst there are still expressions to be checked go around the loop.
		JR	1E2C,DATA-1	

The DATA statement has to be passed-by in 'run-time'.

1E37	DATA-2	LD	A,+E4	It is a 'DATA' statement that is to be passed-by.
------	--------	----	-------	---

THE 'PASS-BY' SUBROUTINE

On entry the A register will hold either the token 'DATA' or the token 'DEF FN' depending on the type of statement that is being 'passed-by'.

1E39	PASS-BY	LD	B,A	Make the BC register pair hold a very high number.
		CPDR		Look back along the statement for the token.
		LD	DE,+0200	Now look along the line for the statement after. (The 'D-1'th statement from the current position).
		JP	198B,EACH-STMT	

THE 'RESTORE' COMMAND ROUTINE

The operand for a RESTORE command is taken as a line number, zero being used if no operand is given.

The REST-RUN entry point is used by the RUN command routine.

1E42	RESTORE	CALL	1E99,FIND-INT2	Compress the operand into the BC register pair.
1E45	REST-RUN	LD	H,B	Transfer the result to the HL register pair.
		LD	L,C	
		CALL	196E,LINE-ADDR	Now find the address of that line or the 'first line after'.
		DEC	HL	Make DATADD point to the location before.
		LD	(DATADD),HL	
		RET		Return once it is done.

THE 'RANDOMIZE' COMMAND ROUTINE

Once again the operand is compressed into the BC register pair and transferred to the required system variable. However if the operand is zero the value in FRAMES1 and FRAMES2 is used instead.

1E4F	RANDOMIZE	CALL	1E99,FIND-INT2	Fetch the operand.
		LD	A,B	Jump forward unless the value of the operand is zero.
		OR	C	
		JR	NZ,1E5A,RAND-1	
		LD	BC,(FRAMES1)	Fetch the two low order bytes of FRAMES instead.
1E5A	RAND-1	LD	(SEED),BC	Now enter the result into the system variable SEED before returning.
		RET		

THE 'CONTINUE' COMMAND ROUTINE

The required line number and statement number within that line are made the object of a jump.

1E5F	CONTINUE	LD	HL,(OLDPPC)	The line number.
		LD	D,(OSPPC)	The statement number.
		JR	1E73,GO-TO-2	Jump forward.

THE 'GO TO' COMMAND ROUTINE

The operand of a GO TO ought to be a line number in the range '1' to '9999' but the actual test is against an upper value of '61439'.

1E67	GO-TO	CALL	1E99,FIND-INT2	Fetch the operand and transfer
		LD	H,B	it to the HL register pair.
		LD	L,C	
		LD	D,+00	Set the statement number to
				zero.
		LD	A,H	Give the error message
		CP	+F0	- Integer out of range -
		JR	NC,1E9F,REPORT-B	with lines over '614139'

The entry point GO-TO-2 is used to determine the line number of the next line to be handled in several instances.

1E73	GO-TO-2	LD	(NEWPPC),HL	Enter the line number and
		LD	(NSPPC),D	then the statement number.
		RET		Return; - to STMT-RET.

THE 'OUT' COMMAND ROUTINE

The two parameters for the OUT instruction are fetched from the calculator stack and used as directed.

1E7A	OUT	CALL	1E85,TWO-PARAM	The operands are fetched.
		OUT	(C),A	The actual OUT instruction.
		RET		Return; - to STMT-RET.

THE 'POKE' COMMAND ROUTINE

In a similar manner the POKE operation is performed.

1E80	POKE	CALL	1E85,TWO-PARAM	The operands are fetched.
		LD	(BC),A	The actual POKE operation.
		RET		Return; - to STMT-RET.

THE 'TWO-PARAM' SUBROUTINE

The topmost parameter on the calculator stack must be compressible into a single register. It is two's complemented if it is negative.

The second parameter must be compressible into a register pair.

1E85	TWO-PARAM	CALL	2DD5,FP-TO-A	The parameter is fetched.
		JR	C,1E9F,REPORT-B	Give an error if it is too high
				a number.
		JR	Z,1E8E,TWO-P-1	Jump forward with positive
		NEG		numbers but two's complement
				negative numbers.
1E8E	TWO-P-1	PUSH	AF	Save the first parameter
		CALL	1E99,FIND-INT2	whilst the second is fetched.
		POP	AF	The first parameter is
		RET		restored before returning.

THE 'FIND INTEGERS' SUBROUTINE

The 'last value' on the calculator stack is fetched and compressed into a single register or a register pair by entering at FIND-INT1 AND FIND-INT2 respectively.

1E94	FIND-INT1	CALL	2DD5,FP-TO-A	Fetch the 'last value'.
		JR	1E9C,FIND-I-1	Jump forward.
1E99	FIND-INT2	CALL	2DA2,FP-TO-BC	Fetch the 'last value'.
1E9C	FIND-I-1	JR	C,1E9F,REPORT-B	In both cases overflow is

		RET	Z	indicated by a set carry flag. Return with all positive numbers that are in range.
Report B - Integer out of range				
1E9F	REPORT-B	RST	0008,ERROR-1	Call the error handling
		DEFB	+0A	routine.

THE 'RUN' COMMAND ROUTINE

The parameter of the RUN command is passed to NEWPPC by calling the GO TO command routine. The operations of 'RESTORE 0' and 'CLEAR 0' are then performed before a return is made.

1EA1	RUN	CALL	1E67,GO-TO	Set NEWPPC as required.
		LD	BC,+0000	Now perform a 'RESTORE 0'.
		CALL	1E45,REST-RUN	
		JR	1EAF,CLEAR-1	Exit via the CLEAR command routine.

THE 'CLEAR' COMMAND ROUTINE

This routine allows for the variables area to be cleared, the display area cleared and RAMTOP moved. In consequence of the last operation the machine stack is rebuilt thereby having the effect of also clearing the GO SUB stack.

1EAC	CLEAR	CALL	1E99,FIND-INT2	Fetch the operand - using zero by default.
1EAF	CLEAR-RUN	LD	A,B	Jump forward if the operand is other than zero. When called from RUN there is no jump.
		OR	C	
		JR	NZ,1EB7,CLEAR-1	If zero use the existing value in RAMTOP.
		LD	BC,(RAMTOP)	
1EB7	CLEAR-1	PUSH	BC	Save the value.
		LD	DE,(VARS)	Next reclaim all the bytes of the present variables area.
		LD	HL,(E-LINE)	
		DEC	HL	
		CALL	19E5,RECLAIM-1	
		CALL	0D6B,CLS	Clear the display area.

The value in the BC register pair which will be used as RAMTOP is tested to ensure it is neither too low nor too high.

LD	HL,(STKEND)	The current value of STKEND.
LD	DE,+0032	is increased by '50' before
ADD	HL,DE	being tested. This forms the
POP	DE	lower limit.
SBC	HL,DE	
JR	NC,1EDA,REPORT-M	RAMTOP will be too low.
LD	HL,(P-RAMT)	For the upper test the value
AND	A	for RAMTOP is tested against
SBC	HL,DE	P-RAMT.
JR	NC,1EDC,CLEAR-2	Jump forward if acceptable.

Report M - RAMTOP no good

1EDA	REPORT-M	RST	0008,ERROR-1	Call the error handling
		DEFB	+15	routine.

Continue with the CLEAR operation.

1EDC	CLEAR-2	EX	DE,HL	Now the value can actually be
		LD	(RAMTOP),HL	passed to RAMTOP.
		POP	DE	Fetch the address - STMT-RET.
		POP	BC	Fetch the 'error address'.
		LD	(HL),+3E	Enter a GO SUB stack end marker.
		DEC	HL	Leave one location.
		LD	SP,HL	Make the stack pointer point

PUSH	BC	to an empty GO SUB stack.
LD	(ERR-SP),SP	Next pass the 'error address'
		to the stack and save its
		address in ERR-SP.
EX	DE,HL	An indirect return is now
JP	(HL)	made to STMT-RET.

Note: When the routine is called from RUN the values of NEWPPC & NSPPC will have been affected and no statements coming after RUN can ever be found before the jump is taken.

THE 'GO SUB' COMMAND ROUTINE

The present value of PPC and the incremented value of SUBPPC are stored on the GO SUB stack.

1EED	GO-SUB	POP	DE	Save the address - STMT-RET.
		LD	H,(SUBPPC)	Fetch the statement number
		INC	H	and increment it.
		EX	(SP),HL	Exchange the 'error address'
				with the statement number.
		INC	SP	Reclaim the use of a location.
		LD	BC,(PPC)	Next save the present line
		PUSH	BC	number.
		PUSH	HL	Return the 'error address'
		LD	(ERR-SP),SP	to the machine stack and
				reset ERR-SP to point to it.
		PUSH	DE	Return the address -
				STMT-RET.
		CALL	1E67,GO-TO-1	Now set NEWPPC & NSPPC to
				the required values.
		LD	BC,+0014	But before making the jump
				make a test for room.

THE 'TEST-ROOM' SUBROUTINE

A series of tests is performed to ensure that there is sufficient free memory available for the task being undertaken.

1F05	TEST-ROOM	LD	HL,(STKEND)	Increase the value taken from
		ADD	HL,BC	STKEND by the value carried
				into the routine by the BC
				register pair.
		JR	C,1F15,REPORT-4	Jump forward if the result is
				over +FFFF.
		EX	DE,HL	Try it again allowing for a
		LD	HL,+0050	further eighty bytes.
		ADD	HL,DE	
		JR	C,1F15,REPORT-4	
		SBC	HL,SP	Finally test the value against the
				address of the machine stack.
		RET	C	Return if satisfactory.
Report 4 - Out of memory				
1F15	REPORT-4	LD	L,+03	This is a 'run-time' error and the
		JP	0055,ERROR-3	error marker is not to be used.

THE 'FREE MEMORY' SUBROUTINE

There is no BASIC command 'FRE' in the SPECTRUM but there is a subroutine for performing such a task.

An estimate of the amount of free space can be found at any time by using:

'PRINT 65536-USR 7962'

1F1A	FREE-MEM	LD	BC,+0000	Do not allow any overhead.
		CALL	1F05,TEST-ROOM	Make the test and pass the

LD	B,H	result to the BC register
LD	C,L	before returning.
RET		

THE 'RETURN' COMMAND ROUTINE

The line number and the statement number that are to be made the object of a 'return' are fetched from the GO SUB stack.

1F23	RETURN	POP	BC	Fetch the address - STMT-RET.
		POP	HL	Fetch the 'error address'.
		POP	DE	Fetch the last entry on the GO SUB stack.
		LD	A,D	The entry is tested to see if
		CP	+3E	it is the GO SUB stack end
		JR	Z,1F36,REPORT-7	marker; jump if it is.
		DEC	SP	The full entry uses three
				locations only.
		EX	(SP),HL	Exchange the statement number
				with the 'error address'.
		EX	DE,HL	Move the statement number.
		LD	(ERR-SP),SP	Reset the error pointer.
		PUSH	BC	Replace the address -
				STMT-RET.
		JP	1E73,GO-TO-2	Jump back to change NEWPPC
				& NSPPC.

Report 7 - RETURN without GOSUB

1F36	REPORT-7	PUSH	DE	Replace the end marker and
		PUSH	HL	the 'error address'.
		RST	0008,ERROR-1	Call the error handling
		DEFB	+06	routine.

THE 'PAUSE' COMMAND ROUTINE

The period of the PAUSE is determined by counting the number of maskable interrupts as they occur every 1/50 th. of a second.

A PAUSE is finished either after the appropriate number of interrupts or by the system Variable FLAGS indicating that a key has been pressed.

1F3A	PAUSE	CALL	1E99,FIND-INT2	Fetch the operand.
1F3D	PAUSE-1	HALT		Wait for a maskable interrupt.
		DEC	BC	Decrease the counter.
		LD	A,B	If the counter is thereby
		OR	C	reduced to zero the PAUSE
		JR	Z,1F4F,PAUSE-END	has come to an end.
		LD	A,B	If the operand was zero BC
		AND	C	will now hold +FFFF and this
		INC	A	value will be returned to
		JR	NZ,1F49,PAUSE-2	zero. Jump will all other
		INC	BC	operand values.
1F49	PAUSE-2	BIT	5,(FLAGS)	Jump back unless a key has
		JR	Z,1F3D,PAUSE-1	been pressed.

The period of the PAUSE has now finished.

1F4F	PAUSE-END	RES	5,(FLAGS)	Signal 'no key pressed'.
		RET		Now return; - to STMT-RET.

THE 'BREAK-KEY' SUBROUTINE

This subroutine is called in several instances to read the BREAK key. The carry flag is returned reset only if the SHIFT and the BREAK keys are both being pressed.

1F54	BREAK-KEY	LD	A,+7F	Form the port address
		IN	A,(+FE)	+7FFE and read in a byte.
		RRA		Examine only bit 0 by shifting
				it into the carry position.

RET	C	Return if the BREAK key is not being pressed.
LD	A,+FE	Form the port address +FEFE and read in a byte.
IN	A,(+FE)	Again examine bit 0.
RRA		Return with carry reset if both keys are being pressed.
RET		

THE 'DEF FN' COMMAND ROUTINE

During syntax checking a DEF FN statement is checked to ensure that it has the correct form. Space is also made available for the result of evaluating the function.

But in 'run-time' a DEF FN statement is passed-by.

1F60	DEF-FN	CALL	2530,SYNTAX-Z	Jump forward if checking syntax.
		JR	Z,1F6A,DEF-FN-1	Otherwise pass-by the 'DEF FN' statement.
		LD	A,+CE	
		JP	1E39,PASS-BY	

First consider the variable of the function.

1F6A	DEF-FN-1	SET	6,(FLAGS)	Signal 'a numeric variable'.
		CALL	2C8D,ALPHA	Check that the present code is a letter.
		JR	NC,1F89,DEF-FN-4	Jump forward if not.
		RST	0020,NEXT-CHAR	Fetch the next character.
		CP	+24	Jump forward unless it is a '\$'.
		JR	NZ,1F7D,DEF-FN-2	Change bit 6 as it is a string variable.
		RES	6,(FLAGS)	Fetch the next character.

1F7D	DEF-FN-2	RST	0020,NEXT-CHAR	A '(' must follow the variable's name.
		CP	+28	Fetch the next character.
		JR	NZ,1FBD,DEF-FN-7	Jump forward if it is a ')' as there are no parameters of the function.
		RST	0020,NEXT-CHAR	
		CP	+29	
		JR	Z,1FA6,DEF-FN-6	

A loop is now entered to deal with each parameter in turn.

1F86	DEF-FN-3	CALL	2C8D,ALPHA	The present code must be a letter.
1F89	DEF-FN-4	JP	NC,1C8A,REPORT-C	Save the pointer in DE.
		EX	DE,HL	Fetch the next character.
		RST	0020,NEXT-CHAR	Jump forward unless it is a '\$'.
		CP	+24	Otherwise save the new pointer in DE instead.
		JR	NZ,1F94,DEF-FN-5	Fetch the next character.
		EX	DE,HL	Move the pointer to the last character of the name to the HL register pair.
1F94	DEF-FN-5	RST	0020,NEXT-CHAR	Now make six locations after that last character and enter a 'number marker' into the first of the new locations.
		EX	DE,HL	If the present character is a ';' then jump back as there should be a further parameter; otherwise jump out of the loop.
		LD	BC,+0006	
		CALL	1655,MAKE-ROOM	
		INC	HL	
		INC	HL	
		LD	(HL),+0E	
		CP	+2C	
		JR	NZ,1FA6,DEF-FN-6	
		RST	0020,NEXT-CHAR	
		JR	1F86,DEF-FN-3	

Next the definition of the function is considered.

1FA6	DEF-FN-6	CP	+29	Check that the ')' does exist.
		JR	NZ,1FBD,DEF-FN-7	

		RST	0020,NEXT-CHAR	The next character is fetched.
		CP	+3D	It must be an '='.
		JR	NZ,1FBD,DEF-FN-7	
		RST	0020,NEXT-CHAR	Fetch the next character.
		LD	A,(FLAGS)	Save the nature - numeric or
		PUSH	AF	string - of the variable.
		CALL	2F4B,SCANNING	Now consider the definition
				as an expression.
		POP	AF	Fetch the nature of the
		XOR	(FLAGS)	variable and check that it
		AND	+40	is of the same type as found
1FBD	DEF-FN-7	JP	NZ,1C8A,REPORT-C	for the definition.
				Give an error report if it
		CALL	1BEE,CHECK-END	is required.
				Exit via the CHECK-END
				subroutine. (Thereby moving
				on to consider the next state-
				ment in the line.)

THE 'UNSTACK-Z' SUBROUTINE

This subroutine is called in several instances in order to 'return early' from a subroutine when checking syntax. The reason for this is to avoid actually printing characters or passing values to/from the calculator stack.

1FC3	UNSTACK-Z	CALL	2530,SYNTAX-Z	Is syntax being checked?
		POP	HL	Fetch the return address but
		RET	Z	ignore it in 'syntax-time'.
		JP	(HL)	In 'run-time' make a simple
				return to the calling routine.

THE 'LPRINT & PRINT' COMMAND ROUTINES

The appropriate channel is opened as necessary and the items to be printed are considered in turn.

1FC9	LPRINT	LD	A,+03	Prepare to open channel 'P'.
		JR	1FCF,PRINT-1	Jump forward.
1FCD	PRINT	LD	A,+02	Prepare to open channel 'S'.
1FCF	PRINT-1	CALL	2530,SYNTAX-Z	Unless syntax is being
		CALL	NZ,1601,CHAN-OPEN	checked open a channel.
		CALL	0D4D,TEMPS	Set the temporary colour
				system variables.
		CALL	1FDF,PRINT-2	Call the print controlling
				subroutine.
		CALL	1BEE,CHECK-END	Move on to consider the next
		RET		statement; via CHECK-END if
				checking syntax.

The print controlling subroutine is called by the PRINT, LPRINT and INPUT command routines.

1FDF	PRINT-2	RST	0018,GET-CHAR	Get the first character.
		CALL	2045,PR-END-Z	Jump forward if already at the
		JR	Z,1FF2,PRINT-4	end of the item list.
Now enter a loop to deal with the 'position controllers' and the print items.				
1FE5	PRINT-3	CALL	204E,PR-POSN-1	Deal with any consecutive
		JR	Z,1FE5,PRINT-3	position controllers.
		CALL	1FFC,PR-ITEM-1	Deal with a single print item.
		CALL	204E,PR-POSN-1	Check for further position
		JR	Z,1FE5,PRINT-3	controllers and print items
				until there are none left.
1FF2	PRINT-4	CP	+29	Return now if the present
		RET	Z	character is a ')'; otherwise
				consider performing a 'carriage
				return'.

THE 'PRINT A CARRIAGE RETURN' SUBROUTINE

1FF5	PRINT-CR	CALL	1FC3,UNSTACK-Z	Return if changing syntax.
		LD	A,+0D	Print a carriage return
		RST	0010,PRINT-A-1	character and then return.
		RET		

THE 'PRINT ITEMS' SUBROUTINE

This subroutine is called from the PRINT, LPRINT and INPUT command routines.

The various types of print item are identified and printed.

1FFC	PR-ITEM-1	RST	0018,GET-CHAR	The first character is fetched.
		CP	+AC	Jump forward unless it is
		JR	NZ,200E,PR-ITEM-2	an 'AT'.
Now deal with an 'AT'.				
		CALL	1C79,NEXT-2NUM	The two parameters are trans-
				ferred to the calculator stack.
		CALL	1FC3,UNSTACK-Z	Return now if checking syntax.
		CALL	2307,STK-TO-BC	The parameters are compressed
				into the BC register pair.
		LD	A,+16	The A register is loaded with
		JR	201E,PR-AT-TAB	the AT control character before
				the jump is taken.
Next look for a 'TAB'.				
200E	PR-ITEM-2	CP	+AD	Jump forward unless it is
		JR	NZ,2024,PR-ITEM-3	a 'TAB'.
Now deal with a 'TAB'.				
		RST	0020,NEXT-CHAR	Get the next character.
		CALL	1C82,EXPT-1NUM	Transfer one parameter to the
				calculator stack.
		CALL	1FC3,UNSTACK-Z	Return now if checking syntax.
		CALL	1E99,FIND-INT2	The value is compressed into the
				BC register pair.
		LD	A,+17	The A register is loaded with the
				TAB control character.
The 'AT' and the 'TAB' print items are printed by making three calls to PRINT-OUT.				
201E	PR-AT-TAB	RST	0010,PRINT-A-1	Print the control character.
		LD	A,C	Follow it with the first
		RST	0010,PRINT-A-1	value.
		LD	A,B	Finally print the second
		RST	0010,PRINT-A-1	value; then return.
		RET		
Next consider embedded colour items.				
2024	PR-ITEM-3	CALL	21F2,CO-TEMP-3	Return with carry reset if a
				colour items was found.
		RET	NC	Continue if none were found.
		CALL	2070,STR-ALTER	Next consider if the stream is
				to be changed.
		RET	NC	Continue unless it was altered.
The print item must now be an expression, either numeric or string.				
		CALL	24FB,SCANNING	Evaluate the expression but
		CALL	1FC3,UNSTACK-Z	return now if checking syntax.
		BIT	6,(FLAGS)	Test for the nature of the
				expression.
		CALL	Z,2BF1,STK-FETCH	If it is string then fetch the nec-
				essary parameters; but if it is
		JP	NZ,2DE3,PRINT-FP	numeric then exit via PRINT-FP.

A loop is now set up to deal with each character in turn of the string.

203C	PR-STRING	LD	A,B	Return now if there are
		OR	C	no characters remaining
		DEC	BC	in the string; otherwise
		RET	Z	decrease the counter.
		LD	A,(DE)	Fetch the code and increment
		INC	DE	the pointer.
		RST	0010,PRINT-A-1	The code is printed and a jump
		JR	203C,PR-STRING	taken to consider any further
				characters.

THE 'END OF PRINTING' SUBROUTINE

The zero flag will be set if no further printing is to be done.

2045	PR-END-Z	CP	+29	Return now if the character
		RET	Z	is a 'y'.
2048	PR-ST-END	CP	+0D	Return now if the character is
		RET	Z	a 'carriage return'.
		CP	+3A	Make a final test against ':'
		RET		before returning.

THE 'PRINT POSITION' SUBROUTINE

The various position controlling characters are considered by this subroutine.

204E	PR-POSN-1	RST	0018,GET-CHAR	Get the present character.
		CP	+3B	Jump forward if it is
		JR	Z,2067,PR-POSN-3	a ','.
		CP	+2C	Also jump forward with a
		JR	NZ,2061,PR-POSN-2	character other than a ',';
		CALL	2530,SYNTAX-Z	but do not actually print the
		JR	Z,2067,PR-POSN-3	character if checking syntax.
		LD	A,+06	Load the A register with
		RST	0010,PRINT-A-1	the 'comma' control code and
		JR	2067,PR-POSN-3	print it; then jump forward.
2061	PR-POSN-2	CP	+27	Is it a ""?
		RET	NZ	Return now if not any of the
				position controllers.
		CALL	1FF5,PR-CR	Print 'carriage return' unless
				checking syntax.
2067	PR-POSN-3	RST	0020,NEXT-CHAR	Fetch the next character.
		CALL	2045,PR-END-Z	If not at the end of a print
		JR	NZ,206E,PR-POSN-4	statement then jump forward;
		POP	BC	otherwise return to the
206E	PR-POSN-4	CP	A	calling routine.
		RET		The zero flag will be reset if the
				end of the print statement has
				not been reached.

THE 'ALTER STREAM' SUBROUTINE

This subroutine is called whenever there is the need to consider whether the user wishes to use a different stream.

2070	STR-ALTER	CP	+23	Unless the present character
		SCF		is a '#' return with the
		RET	NZ	carry flag set.
		RST	0020,NEXT-CHAR	Advance CH-ADD.
		CALL	1C82,EXPT-1NUM	Pass the parameter to the
				calculator stack.
		AND	A	Clear the carry flag.
		CALL	1FC3,UNSTACK-Z	Return now if checking syntax.
		CALL	1E94,FIND-INT1	The value is passed to the A
				register.
		CP	+10	Give report O if the value is

JP	NC,160E,REPORT-O	over +FF.
CALL	1601,CHAN-OPEN	Use the channel for the stream in question.
AND	A	Clear the carry flag and return.
RET		

THE 'INPUT' COMMAND ROUTINE

This routine allows for values entered from the keyboard to be assigned to variables. It is also possible to have print items embedded in the INPUT statement and these items are printed in the lower part of the display.

2089	INPUT	CALL	2530,SYNTAX-Z	Jump forward if syntax is being checked.
		JR	Z,2096,INPUT-1	
		LD	A,+01	Open channel 'K'.
		CALL	1601,CHAN-OPEN	
		CALL	0D6E,CLS-LOWER	The lower part of the display is cleared.
2096	INPUT-1	LD	(TV-FLAG),+01	Signal that the lower screen is being handled. Reset all other bits.
		CALL	20C1,IN-ITEM-1	Call the subroutine to deal with the INPUT items.
		CALL	1BEE,CHECK-END	Move on to the next statement if checking syntax.
		LD	BC,(S-POSN)	Fetch the current print position.
		LD	A,(DF-SZ)	Jump forward if the current position is above the lower screen.
		CP	B	
		JR	C,20AD,INPUT-2	
		LD	C,+21	Otherwise set the print position to the top of the lower screen.
20AD	INPUT-2	LD	B,A	Reset S-POSN.
		LD	(S-POSN),BC	Now set the scroll counter.
		LD	A,+19	
		SUB	B	
		LD	(SCR-CT),A	
		RES	0,(TV-FLAG)	Signal 'main screen'.
		CALL	0DD9,CL-SET	Set the system variables and exit via CLS-LOWER.
		JP	0D6E,CLS-LOWER	
The INPUT items and embedded PRINT items are dealt with in turn by the following loop.				
20C1	IN-ITEM-1	CALL	204E,PR-POSN-1	Consider first any position control characters.
		JR	Z,20C1,IN-ITEM-1	
		CP	+28	Jump forward if the present character is not a '('.
		JR	NZ,20D8,IN-ITEM-2	
		RST	0020,NEXT-CHAR	Fetch the next character.
		CALL	1FDF,PRINT-2	Now call the PRINT command routine to handle the items inside the brackets.
		RST	0018,GET-CHAR	Fetch the present character.
		CP	+29	Give report C unless the character is a ')'. Fetch the next character and jump forward to see if there are any further INPUT items.
		JP	NZ,1C8A,REPORT-C	
		RST	0020,NEXT-CHAR	
		JP	21B2,IN-NEXT-2	

Now consider whether INPUT LINE is being used.

20D8	IN-ITEM-2	CP	+CA	Jump forward if it is not 'LINE'.
		JR	NZ,20ED,IN-ITEM-3	
		RST	0020,NEXT-CHAR	Advance CH-ADD.
		CALL	1C1F,CLASS-01	Determine the destination address for the variable.
		SET	7,(FLAGX)	Signal 'using INPUT LINE'.
		BIT	6,(FLAGS)	Give report C unless using a string variable.
		JP	NZ,1C8A,REPORT-C	

		JR	20FA,IN-PROMPT	Jump forward to issue the prompt message.
Proceed to handle simple INPUT variables.				
20ED	IN-ITEM-3	CALL JP	2C8D,ALPHA NC,21AF-IN-NEXT-1	Jump to consider going round the loop again if the present character is not a letter.
		CALL	1C1F,CLASS-01	Determine the destination address for the variable.
		RES	7,(FLAGX)	Signal 'not INPUT LINE'.
The prompt message is now built up in the work space.				
20FA	IN-PROMPT	CALL JP	2530,SYNTAX-Z Z,21B2,IN-NEXT-2	Jump forward if only checking syntax.
		CALL	16BF,SET-WORK	The work space is set to null.
		LD	HL,+5C71	This is FLAGX.
		RES	6,(HL)	Signal 'string result'.
		SET	5,(HL)	Signal 'INPUT mode'.
		LD	BC,+0001	Allow the prompt message only a single location.
		BIT	7,(HL)	Jump forward if using 'LINE'.
		JR	NZ,211C,IN-PR-2	
		LD	A,(FLAGS)	Jump forward if awaiting a numeric entry.
		AND	+40	
		JR	NZ,211A,IN-PR-1	
		LD	C,+03	
211A	IN-PR-1	OR	(HL)	A string entry will need three locations.
		LD	(HL),A	Bit 6 of FLAGX will become set for a numeric entry.
211C	IN-PR-2	RST	0030,BC-SPACES	The required number of locations is made available.
		LD	(HL),+0D	A 'carriage return' goes into the last location.
		LD	A,C	Test bit 6 of the C register and jump forward if only one location was required.
		RRCA		
		RRCA		
		JR	NC,2129,IN-PR-3	
		LD	A,+22	A 'double quotes' character goes into the first and second locations.
		LD	(DE),A	
		DEC	HL	
		LD	(HL),A	
2129	IN-PR-3	LD	(K-CUR),HL	The position of the cursor can now be saved.
In the case of INPUT LINE the EDITOR can be called without further preparation but for other types of INPUT the error stack has to be changed so as to trap errors.				
		BIT	7,(FLAGX)	Jump forward with INPUT LINE'
		JR	NZ,215E,IN-VAR-3	
		LD	HL,(CH-ADD)	Save the current values of CH-ADD & ERR-SP on the machine stack.
		PUSH	HL	
		LD	HL,(ERR-SP)	
		PUSH	HL	
213A	IN-VAR-1	LD	HL,+213A	This will be the 'return point' in case of errors.
		PUSH	HL	Only change the error stack pointer if using channel 'K'.
		BIT	4,(FLAGS2)	
		JR	Z,2148,IN-VAR-2	
		LD	(ERR-SP),SP	
2148	IN-VAR-2	LD	HL,(WORKSP)	Set HL to the start of the INPUT line and remove any floating-point forms. (There will not be any except perhaps after an error.)
		CALL	11A7,REMOVE-FP	Signal 'no error yet'.
		LD	(ERR-NR),+FF	

		CALL	0F2C,EDITOR	Now get the INPUT and with
		RES	7,(FLAGS)	the syntax/run flag indicating
		CALL	21B9,IN-ASSIGN	syntax, check the INPUT for
		JR	2161,IN-VAR-4	errors; jump if in order; return
215E	IN-VAR-3	CALL	0F2C,EDITOR	to IN-VAR-1 if not.
All the system variables have to be reset before the actual assignment of a value can be made.				Get a 'LINE'.
2161	IN-VAR-4	LD	(K-CUR-hi),+00	The cursor address is reset.
		CALL	21D6,IN-CHAN-K	The jump is taken if using
		JR	NZ,2174,IN-VAR-5	other than channel 'K'.
		CALL	111D,ED-COPY	The input-line is copied to
		LD	BC,(ECHO-E)	the display and the position
		CALL	0DD9,CL-SET	in ECHO-E made the current
				position in the lower screen.
2174	IN-VAR-5	LD	HL,+5C71	This is FLAGX.
		RES	5,(HL)	Signal 'edit mode'.
		BIT	7,(HL)	Jump forward if handling an
		RES	7,(HL)	INPUT LINE.
		JR	NZ,219B,IN-VAR-6	
		POP	HL	Drop the address IN-VAR-1.
		POP	HL	Reset the ERR-SP to its
		LD	(ERR-SP),HL	original address.
		POP	HL	Save the original CH-ADD
		LD	(X-PTR),HL	address in X-PTR.
		SET	7,(FLAGS)	Now with the syntax/run flag
		CALL	21B9,IN-ASSIGN	indicating 'run' make the
				assignment.
		LD	HL,(X-PTR)	Restore the original address
		LD	(X-PTR-hi),+00	to CH-ADD and clear X-PTR.
		LD	(CH-ADD),HL	
		JR	21B2,IN-NEXT-2	Jump forward to see if there
				are further INPUT items.
219B	IN-VAR-6	LD	HL,(STKBOT)	The length of the 'LINE' in
		LD	DE,(WORKSP)	the work space is found.
		SCF		
		SBC,	HL,DE	
		LD	B,H	DE points to the start and
		LD	C,L	BC holds the length.
		CALL	2AB2,STK-ST-\$	These parameters are stacked
		CALL	2AFF,LET	and the actual assignment made.
		JR	21B2,IN-NEXT-2	Also jump forward to consider
				further items.
Further items in the INPUT statement are considered.				
21AF	IN-NEXT-1	CALL	1FFC,PR-ITEM-1	Handle any print items.
21B2	IN-NEXT-2	CALL	204E,PR-POSN-1	Handle any position controllers.
		JP	Z,20C1,IN-ITEM-1	Go around the loop again if
				there are further items;
		RET		otherwise return.

THE 'IN-ASSIGN' SUBROUTINE

This subroutine is called twice for each INPUT value. Once with the syntax/run flag reset (syntax) and once with it set (run).

21B9	IN-ASSIGN	LD	HL,(WORKSP)	Set CH-ADD to point to the
		LD	(CH-ADD),HL	first location of the work
		RST	0018,GET-CHAR	space and fetch the character.
		CP	+E2	Is it a 'STOP'?
		JR	Z,21D0,IN-STOP	Jump if it is.
		LD	A,(FLAGX)	Otherwise make the assignment
		CALL	1C59,VAL-FET-2	of the 'value' to the variable.

		RST	0018,GET-CHAR	Get the present character
		CP	+0D	and check it is a 'carriage
		RET	Z	return'. Return if it is.
Report C - Nonsense in BASIC				
21CE	REPORT-C	RST	0008,ERROR-1	Call the error handling
		DEFB	+0B	routine.
Come here if the INPUT line starts with 'STOP'.				
21D0	IN-STOP	CALL	2530,SYNTAX-Z	But do not give the error
		RET	Z	report on the syntax-pass.
Report H - STOP in INPUT				
21D4	REPORT-H	RST	0008,ERROR-1	Call the error handling
		DEFB	+10	routine.

THE 'IN-CHAN-K' SUBROUTINE

This subroutine returns with the zero flag reset only if channel 'K' is being used.

21D6	IN-CHAN-K	LD	HL,(CURCHL)	The base address of the
		INC	HL	channel information for the
		INC	HL	current channel is fetched
		INC	HL	and the channel code compared
		INC	HL	to the character 'K'.
		LD	A,(HL)	
		CP	+4B	
		RET		Return afterwards.

THE 'COLOUR ITEM' ROUTINES

This set of routines can be readily divided into two parts:

- i. The embedded colour item' handler.
 - ii. The 'colour system variable' handler.
- i. Embedded colour items are handled by calling the PRINT-OUT subroutine as required.
A loop is entered to handle each item in turn. The entry point is at CO-TEMP-2.

21E1	CO-TEMP-1	RST	0020,NEXT-CHAR	Consider the next character
				in the BASIC statement.
21E2	CO-TEMP-2	CALL	21F2,CO-TEMP-3	Jump forward to see if the
				present code represents an
				embedded 'temporary' colour
		RET	C	item. Return carry set if not a
				colour item.
		RST	0018,GET-CHAR	Fetch the present character.
		CP	+2C	Jump back if it is either a
		JR	Z,21E1,CO-TEMP-1	',' or a ';;' otherwise
		CP	+3B	there has been an error.
		JR	Z,21E1,CO-TEMP-1	
		JP	1C8A,REPORT-C	Exit via 'report C'.
21F2	CO-TEMP-3	CP	+D9	Return with the carry flag
		RET	C	Set if the code is not in the
		CP	+DF	range +D9 to +DE (INK to
				OVER).
		CCF		
		RET	C	
		PUSH	AF	The colour item code is
		RST	0020,NEXT-CHAR	preserved whilst CH-ADD is
		POP	AF	advanced to address the
				parameter that follows it.

The colour item code and the parameter are now 'printed' by calling PRINT-OUT on two occasions.

21FC	CO-TEMP-4	SUB	+C9	The token range (+D9 to +DE)
------	-----------	-----	-----	------------------------------

		PUSH	AF	is reduced to the control character range (+10 to +15).
		CALL	1C82,EXPT-1NUM	The control character code is preserved whilst the parameter is moved to the calculator stack.
		POP	AF	A return is made at this point if syntax is being checked.
		AND	A	The control character code is preserved whilst the parameter is moved to the D register.
		CALL	1FC3,UNSTACK-Z	
		PUSH	AF	
		CALL	1E94,FIND-INT1	
		LD	D,A	
		POP	AF	
		RST	0010,PRINT-A-1	The control character is sent out.
		LD	A,D	Then the parameter is fetched and sent out before returning.
		RST	0010,PRINT-A-1	
		RET		
ii. The colour system variables - ATTR-T, MASK-T & P-FLAG - are altered as required. This subroutine is called by PRINT-OUT. On entry the control character code is in the A register and the parameter is in the D register. Note that all changes are to the 'temporary' system variables.				
2211	CO-TEMP-5	SUB	+11	Reduce the range and jump forward with INK & PAPER.
		ADC	A,+00	
		JR	Z,2234,CO-TEMP-7	
		SUB	+02	Reduce the range once again and jump forward with FLASH & BRIGHT.
		ADC	A,+00	
		JR	Z,2273,CO-TEMP-C	
The colour control code will now be +01 for INVERSE and +02 for OVER and the system variable P-FLAG is altered accordingly.				
		CP	+01	Prepare to jump with OVER.
		LD	A,D	Fetch the parameter.
		LD	B,+01	Prepare the mask for OVER.
		JR	NZ,2228,CO-TEMP-6	Now jump.
		RLCA		Bit 2 of the A register is to be reset for INVERSE 0 and set for INVERSE 1; the mask is to have bit 2 set.
		RLCA		
		LD	B,+04	
2228	CO-TEMP-6	LD	C,A	Save the A register whilst the range is tested.
		LD	A,D	The correct range for INVERSE and OVER is only '0-1'.
		CP	+02	
		JR	NC,2244,REPORT-K	
		LD	A,C	Fetch the A register.
		LD	HL,+5C91	It is P-FLAG that is to be changed.
		JR	226C,CO-CHANGE	Exit via CO-CHANGE and alter P-FLAG using 'B' as a mask. i.e. Bit 0 for OVER & bit 2 for INVERSE'
PAPER & INK are dealt with by the following routine. On entry the carry flag is set for INK.				
2234	CO-TEMP-7	LD	A,D	Fetch the parameter.
		LD	B,+07	Prepare the mask for INK.
		JR	C,223E,CO-TEMP-8	Jump forward with INK.
		RLCA		Multiply the parameter for PAPER by eight.
		RLCA		
		RLCA		
		LD	B,+38	Prepare the mask for PAPER.
223E	CO-TEMP-8	LD	C,A	Save the parameter in the C register whilst the range of the parameter is tested.

		LD	A,D	Fetch the original value.
		CP	+0A	Only allow PAPER/INK a
		JR	C,2246,CO-TEMP-9	range of '0' to '9'.
Report K - Invalid colour				
2244	REPORT-K	RST	0008,ERROR-1	Call the error handling
		DEFB	+13	routine.
Continue to handle PAPER & INK;				
2246	CO-TEMP-9	LD	HL,+5C8F	Prepare to alter ATTR-T,
				MASK-T & P-FLAG.
		CP	+08	Jump forward with PAPER/INK
		JR	C,2258,CO-TEMP-B	'0' to '7'.
		LD	A,(HL)	Fetch the current value of
		JR	Z,2257,CO-TEMP-A	ATTR-T and use it unchanged,
				by jumping forward, with
				PAPER/INK '8'.
		OR	B	But for PAPER/INK '9' the
		CPL		PAPER and INK colours
		AND	+24	have to be black and white.
		JR	Z,2257,CO-TEMP-A	Jump for black INK/PAPER;
		LD	A,B	but continue for white INK/
				PAPER.
2257	CO-TEMP-A	LD	C,A	Move the value to the C register.
The mask (B) and the value (C) are now used to change ATTR-T.				
2258	CO-TEMP-B	LD	A,C	Move the value.
		CALL	226C,CO-CHANGE	Now change ATTR-T as needed.
Next MASK-T is considered.				
		LD	A,+07	The bits of MASK-T are set
		CP	D	only when using PAPER/INK
		SBC	A,A	'8' or '9'.
		CALL	226C,CO-CHANGE	Now change MASK-T as needed.
Next P-FLAG is considered.				
		RLCA		The appropriate mask is
		RLCA		built up in the B register
		AND	+50	is order to change bits 4 &
		LD	B,A	6 as necessary.
		LD	A,+08	The bits of P-FLAG are set
		CP	D	only when using PAPER/INK
		SBC	A,A	'9'.
				Continue into CO-CHANGE to
				manipulate P-FLAG.

THE 'CO-CHANGE' SUBROUTINE

This subroutine is used to 'impress' upon a system variable the 'nature' of the bits in the A register, The B register holds a mask that shows which bits are to be 'copied over' from A to (HL).

226C	CO-CHANGE	XOR	(HL)	The bits, specified by the
		AND	B	mask in the B register, are
		XOR	(HL)	changed in the value and the
		LD	(HL),A	result goes to form the
				system variable.
		INC	HL	Move on to address the next
				system variable.
		LD	A,B	Return with the mask in the
		RET		A register.

FLASH & BRIGHT are handled by the following routine.

2273	CO-TEMP-C	SBC	A,A	The zero flag will be set for BRIGHT.
		LD	A,D	The parameter is fetched and rotated.
		RRCA		
		LD	B,+80	Prepare the mask for FLASH.
		JR	NZ,227D,CO-TEMP-D	Jump forward with FLASH.
		RRCA		Rotate an extra time and prepare the mask for BRIGHT.
		LD	B,+40	
227D	CO-TEMP-D	LD	C,A	Save the value in the C register.
		LD	A,D	Fetch the parameter and test its range; only '0', '1' & '8' are allowable.
		CP	+08	
		JR	Z,2287,CO-TEMP-E	
		CP	+02	
		JR	NC,2244,REPORT-K	
The system variable ATTR-T can now be altered.				
2287	CO-TEMP-E	LD	A,C	Fetch the value.
		LD	HL,+5C8F	This is ATTR-T.
		CALL	226C,CO-CHANGE	Now change the system variable.
The value in MASK-T is now considered.				
		LD	A,C	The value is fetched anew.
		RRCA		The set bit of FLASH/BRIGHT '8' (bit 3) is moved to bit 7 (for FLASH) or bit 6 (for BRIGHT).
		RRCA		
		RRCA		
		JR	226C,CO-CHANGE	Exit via CO-CHANGE.

THE 'BORDER' COMMAND ROUTINE

The parameter of the BORDER command is used with an OUT command to actually alter the colour of the border. The parameter is then saved in the system variable BORDCR.

2294	BORDER	CALL	1E94,FIND-INT1	The parameter is fetched and its range is tested.
		CP	+08	
		JR	NC,2244,REPORT-K	
		OUT	(+FE),A	The OUT instruction is then used to set the border colour.
		RLCA		The parameter is then multiplied by eight.
		RLCA		
		RLCA		
		BIT	5,A	If the border colour is a 'light' colour then the INK colour in the editing area is to be black - make the jump.
		JR	NZ,22A6,BORDER-1	
		XOR	+07	Change the INK colour.
22A6	BORDER-1	LD	(BORDCR),A	Set the system variable as required and return.
		RET		

THE 'PIXEL ADDRESS' SUBROUTINE

This subroutine is called by the POINT subroutine and by the PLOT command routine. Is is entered with the co-ordinates of a pixel in the BC register pair and returns with HL holding the address of the display file byte which contains that pixel and A pointing to the position of the pixel within the byte.

22AA	PIXEL-ADD	LD	A,+AF	Test that the y co-ordinate (in B) is not greater than 175.
		SUB	B	
		JP	C,24F9,REPORT-B	
		LD	B,A	B now contains 175 minus y.
		AND	A	A holds b7b6b5b4b3b2b1b0, the bite of B. And now 0b7b6b5b4b3b2b1.
		RRA		
		SCF		
		RRA		Now 10b7b6b5b4b3b2.

AND	A	Now 010b7b6b5b4b3.
RRA		
XOR	B	
AND	+F8	Finally 010b7b6b2b1b0, so that
XOR	B	H becomes $64 + 8 * \text{INT}(B/64) +$
LD	H,A	B (mod 8), the high byte of the
LD	A,C	pixel address. C contains X.
RLCA		A starts as c7c6c5c4c3c2c1c0.
RLCA		
RLCA		And is now c2c1c0c7c6c5c4c3.
XOR	B	
AND	+C7	
XOR	B	Now c2c1b5b4b3c5c4c3.
RLCA		
RLCA		Finally b5b4b3c7c6c5c4c3, so
LD	L,A	that L becomes $32 * \text{INT}(B(\text{mod}$
LD	A,C	$64)/8) + \text{INT}(x/8)$, the low byte.
AND	+07	A holds $x(\text{mod } 8)$: so the pixel
RET		is bit (A - 7) within the byte.

THE 'POINT' SUBROUTINE

This subroutine is called by the POINT function in SCANNING. It is entered with the co-ordinates of a pixel on the calculator stack, and returns a last value of 1 if that pixel is ink colour, and 0 if it is paper colour.

22CB	POINT-SUB	CALL	2307,STK-TO-BC	Y co-ordinate to B, x to C.
		CALL	22AA,PIXEL-ADD	Pixel address to HL.
		LD	B,A	B will count A+1 loops to get
		INC	B	the wanted bit of (HL) to
		LD	A,(HL)	location 0.
22D4	POINT-LP	RLCA		The shifts.
		DJNZ	22D4,POINT-LP	
		AND	+01	The bit is 1 for ink, 0 for paper.
		JP	2D28,STACK-A	It is put on the calculator stack.

THE 'PLOT' COMMAND ROUTINE

This routine consists of a main subroutine plus one line to call it and one line to exit from it. The main routine is used twice by CIRCLE and the subroutine is called by DRAW. The routine is entered with the co-ordinates of a pixel on the calculator stack. It finds the address of that pixel and plots it, taking account of the status of INVERSE and OVER held in the P-FLAG.

22DC	PLOT	CALL	2307,STK-TO-BC	Y co-ordinate to B, x to C.
		CALL	22E5,PLOT-SUB	The subroutine is called.
		JP	0D4D,TEMPS	Exit, setting temporary colours.
22E5	PLOT-SUB	LD	(COORDS),BC	The system variable is set.
		CALL	22AA,PIXEL-ADD	Pixel address to HL.
		LD	B,A	B will count A+1 loops to get a
		INC	B	zero to the correct place in A.
		LD	A,+FE	The zero is entered.
22F0	PLOT-LOOP	RRCA		Then lined up with the pixel
		DJNZ	22F0,PLOT-LOOP	bit position in the byte.
		LD	B,A	Then copied to B.
		LD	A,(HL)	The pixel-byte is obtained in A.
		LD	C,(P-FLAG)	P-FLAG is obtained and first
		BIT	0,C	tested for OVER.
		JR	NZ,22FD,PL-TST-IN	Jump if OVER 1.
		AND	B	OVER 0 first makes the pixel
				zero.
22FD	PL-TST-IN	BIT	2,C	Test for INVERSE.
		JR	NZ,2303,PLOT-END	INVERSE 1 just leaves the pixel
				as it was (OVER 1) or zero
				(OVER 0).
		XOR	B	INVERSE 0 leaves the pixel

		CPL		complemented (OVER 1) or 1 (OVER 0).
2303	PLOT-END	LD	(HL),A	The byte is entered. Its other bits are unchanged in every case.
		JP	0BDB,PO-ATTR	Exit, setting attribute byte.

THE 'STK-TO-BC' SUBROUTINE

This subroutine loads two floating point numbers into the BC register pair. It is thus used to pick up parameters in the range +00-+FF. It also obtains in DE the 'diagonal move' values ($\pm 1, \pm 1$) which are used in the line drawing subroutine of DRAW.

2307	STK-TO-BC	CALL	2314,STK-TO-A	First number to A.
		LD	B,A	Hence to B.
		PUSH	BC	Save it briefly.
		CALL	2314,STK-TO-A	Second number to A.
		LD	E,C	Its sign indicator to E.
		POP	BC	Restore first number.
		LD	D,C	Its signs indicator to D.
		LD	C,A	Second number to C.
		RET		BC, DE are now as required.

THE 'STK-TO-A' SUBROUTINE

This subroutine loads the A register with the floating point number held at the top of the calculator stack. The number must be in the range 00-FF.

2314	STK-TO-A	CALL	2DD5,FP-TO-A	Modulus of rounded last value to
		JP	C,24F9,REPORT-B	A if possible; else, report error.
		LD	C,+01	One to C for positive last value.
		RET	Z	Return if value was positive.
		LD	C,+FF	Else change C to +FF (i.e. minus
		RET		one). Finished.

THE 'CIRCLE' COMMAND ROUTINE

This routine draws an approximation to the circle with centre co-ordinates X and Y and radius Z. These numbers are rounded to the nearest integer before use. Thus Z must be less than 87.5, even when (X,Y) is in the centre of the screen. The method used is to draw a series of arcs approximated by straight lines. It is illustrated in the BASIC program in the appendix. The notation of that program is followed here.

CIRCLE has four parts:

- i. Tests the radius. If its modulus is less than 1, just plot X,Y;
- ii. Calls CD-PRMS-1 at 2470-24B6, which is used to set the initial parameters for both CIRCLE and DRAW;
- iii. Sets up the remaining parameters for CIRCLE, including the initial displacement for the first 'arc' (a straight line in fact);
- iv. Jumps into DRAW to use the arc-drawing loop at 2420-24FA.

Parts i. to iii. will now be explained in turn.

i. 2320-23AA. The radius, say Z', is obtained from the calculator stack. Its modulus Z is formed and used from now on. If Z is less than 1, it is deleted from the stack and the point X,Y is plotted by a jump to PLOT.

2320	CIRCLE	RST	0017,GET-CHAR	Get the present character.
		CP	+2C	Test for comma.
		JP	NZ,1C8A,REPORT-C	If not so, report the error.
		RST	0020,NEXT-CHAR	Get next character (the radius).
		CALL	1C82,EXPT-1NUM	Radius to calculator stack.
		CALL	1BEE,CHECK-END	Move to consider next statement if checking syntax.
		RST	0028,FP-CALC	Use calculator: the stack holds:
		DEFB	+2A,abs	X, Y, Z
		DEFB	+3D,re-stack	Z is re-stacked; its exponent
		DEFB	+38,end-calc	is therefore available.
		LD	A,(HL)	Get exponent of radius.

CP	+81	Test whether radius less than 1.
JR	NC,233B,C-R-GRE-1	If not, jump.
RST	0028,FP-CALC	If less, delete it from the stack.
DEFB	+02,delete	The stack holds X, Y.
DEFB	+38,end-calc	
JR	22DC,PLOT	Just plot the point X, Y.

ii. 233B-2346 and the call to CD-PRMS1. 2π is stored in mem-5 and CD-PRMS1 is called. This subroutine stores in the B register the number of arcs required for the circle, viz. $A=4*\text{INT}(\pi*\text{SQR } Z/4)+4$, hence 4, 8, 12 ..., up to a maximum of 32. It also stores in mem-0 to mem-4 the quantities $2\pi/A$, $\sin(\pi/A)$, 0, $\cos(2\pi/A)$ and $\sin(2\pi/A)$.

233B	C-R-GRE-1	RST	0028,FP-CALC	
		DEFB	+A3,stk-pi/2	X, Y, Z, $\pi/2$.
		DEFB	+38,end-calc	Now increase exponent to 83
		LD	(HL),+83	hex, changing $\pi/2$ into 2π .
		RST	0028,FP-CALC	X, Y, Z, 2π .
		DEFB	+C5,st-mem-5	(2π is copied to mem-5).
		DEFB	+02,delete	X, Y, Z
		DEFB	+38,end-calc	
		CALL	247D,CD-PRMS1	Set the initial parameters.

iii. 2347-2381: the remaining parameters and the jump to DRAW. A test is made to see whether the initial 'arc' length is less than 1. If it is, a jump is made simply to plot X, Y. Otherwise, the parameters are set: $X+Z$ and $X-Z*\sin(\pi/A)$ are stacked twice as start and end point, and copied to COORDS as well; zero and $2*Z*\sin(\pi/A)$ are stored in mem-1 and mem-2 as initial increments, giving as first 'arc' the vertical straight line joining $X+Z$, $y-Z*\sin(\pi/A)$ and $X+Z$, $Y+Z*\sin(\pi/A)$. The arc-drawing loop of DRAW will ensure that all subsequent points remain on the same circle as these two points, with incremental angle $2\pi/A$. But it is clear that these 2 points in fact subtend this angle at the point $X+Z*(1-\cos(\pi/A))$, Y not at X, Y. Hence the end points of each arc of the circle are displaced right by an amount $2*(1-\cos(\pi/A))$, which is less than half a pixel, and rounds to one pixel at most.

2347	C-ARC-GE1	PUSH	BC	Save the arc-count in B.
		RST	0028,FP-CALC	X,Y,Z
		DEFB	+31,duplicate	X,Y,Z,Z
		DEFB	+E1,get-mem-1	X,Y,Z,Z, $\sin(\pi/A)$
		DEFB	+04,multiply	X,Y,Z,Z, $\sin(\pi/A)$
		DEFB	+38,end-calc	$Z*\sin(\pi/A)$ is half the initial
		LD	A,(HL)	'arc' length; it is tested to see
		CP	+80	whether it is less than 0.5.
		JR	NC,235A,C-ARC-GE1	If not, the jump is made.
		RST	0028,FP-CALC	Otherwise, Z is deleted from the
		DEFB	+02,delete	stack, with the half-arc too; the
		DEFB	+02,delete	machine stack is cleared; and a
		DEFB	+38,end-calc	jump is made to plot X, Y.
		POP	BC	
		JP	22DC,PLOT	
235A		RST	0028,FP-CALC	X,Y,Z,Z, $\sin(\pi/A)$
		DEFB	+C2,st-mem-2	($Z*\sin(\pi/A)$ to mem-2 for
				now).
		DEFB	+01,exchange	X,Y,Z, $\sin(\pi/A)$,Z
		DEFB	+C0,st-mem-0	X,Y,Z, $\sin(\pi/A)$,Z
		DEFB	+02,delete X,Y,Z, $\sin(\pi/A)$	
		DEFB	+03,subtract	X, Y - $Z*\sin(\pi/A)$
		DEFB	+01,exchange	Y - $Z*\sin(\pi/A)$, X
		DEFB	+E0,get-mem-0	Y - $Z*\sin(\pi/A)$, X, Z
		DEFB	+0F,addition	Y - $Z*\sin(\pi/A)$, X+Z
		DEFB	+C0,st-mem-0	(X+Z is copied to mem-0)
		DEFB	+01,exchange	X+Z, Y - $Z*\sin(\pi/A)$
		DEFB	+31,duplicate	X+Z, Y- $Z*\sin(\pi/A)$, Y- $Z*\sin$
				(π/A)
		DEFB	+E0,get-mem-0	sa,sb,sb,sa

DEFB	+01,exchange	sa,sb,sa,sb
DEFB	+31,duplicate	sa,sb,sa,sb,sb
DEFB	+E0,get-mem-0	sa,sb,sa,sb,sb,sa
DEFB	+A0,stk-zero	sa,sb,sa,sb,sb,sa,0
DEFB	+C1,st-mem-1	(mem-1 is set to zero)
DEFB	+02,delete	sa,sb,sa,sb,sb,sa
DEFB	+38,end-calc	

(Here sa denotes $X+Z$ and sb denotes $Y - Z \cdot \sin(\pi/A)$).

INC	(mem-2-1st)	Incrementing the exponent byte of mem-2 sets mem-2 to $2 \cdot Z \cdot \sin(\pi/A)$.
CALL	1E94,FIND-INT1	The last value $X+Z$ is moved from the stack to A and copied to L.
LD	L,A	It is saved in HL.
PUSH	HL	$Y - Z \cdot \sin(\pi/A)$ goes from the stack to A and is copied to H.
CALL	1E94,FIND-INT1	HL now holds the initial point.
POP	HL	It is copied to COORDS.
LD	H,A	The arc-count is restored.
LD	(COORDS),HL	The jump is made to DRAW.
POP	BC	
JP	2420,DRW-STEPS	

(The stack now holds $X+Z$, $Y - Z \cdot \sin(\pi/A)$, $Y - Z \cdot \sin(\pi/A)$, $X+Z$).

THE DRAW COMMAND ROUTINE

This routine is entered with the co-ordinates of a point X_0 , Y_0 , say, in COORDS. If only two parameters X , Y are given with the DRAW command, it draws an approximation to a straight line from the point X_0 , Y_0 to X_0+X , Y_0+Y . If a third parameter G is given, it draws an approximation to a circular arc from X_0 , Y_0 to X_0+X , Y_0+Y turning anti-clockwise through an angle G radians.

The routine has four parts:

- Just draws a line if only 2 parameters are given or if the diameter of the implied circle is less than 1;
- Calls CD-PRMS1 at 247D-24B6 to set the first parameters;
- Sets up the remaining parameters, including the initial displacements for the first arc;
- Enters the arc-drawing loop and draws the arc as a series of smaller arcs approximated by straight lines, calling the line-drawing subroutine at 24B7-24FA as necessary.

Two subroutines, CD-PRMS1 and DRAW-LINE, follow the main routine. The above 4 parts of the main routine will now be treated in turn.

- If there are only 2 parameters, a jump is made to LINE-DRAW at 2477. A line is also drawn if the quantity $Z = (\text{ABS } X + \text{ABS } Y) / \text{ABS } \sin(G/2)$ is less than 1. Z lies between 1 and 1.5 times the diameter of the implied circle. In this section mem-0 is set to $\sin(G/2)$, mem-1 to Y , and mem-5 to G .

2382	DRAW	RST	0018,GET-CHAR	Get the current character.
		CP	+2C	If it is a comma,
		JR	Z,238D,DR-3-PRMS	then jump.
		CALL	1BEE,CHECK-END	Move on to next statement if checking syntax.
		JP	2477,LINE-DRAW	Jump to just draw the line.
238D	DR-3-PRMS	RST	0020,NEXT-CHAR	Get next character (the angle).
		CALL	1C82,EXPT-1NUM	Angle to calculator stack.
		CALL	1BEE,CHECK-END	Move on to next statement if checking syntax.
		RST	0028,FP-CALC	X , Y , G are on the stack.
		DEFB	+C5,st-mem-5	(G is copied to mem-5)
		DEFB	+A2,stk-half	X , Y , G , 0.5
		DEFB	+04,multiply	X , Y , $G/2$
		DEFB	+1F,sin	X , Y , $\sin(G/2)$
		DEFB	+31,duplicate	X , Y , $\sin(G/2)$, $\sin(G/2)$

		DEFB	+30,not	X, Y, SIN (G/2), (0/1)
		DEFB	+30,not	X, Y, SIN (G/2), (1/0)
		DEFB	+00,jump-true	X, Y, SIN (G/2)
		DEFB	+06,to DR-SIN-NZ	(If SIN (G/2)=0 i.e. $G = 2 \cdot N \cdot \pi$
		DEFB	+02,delete	just draw a straight line).
		DEFB	+38,end-calc	X, Y
		JP	2477,LINE-DRAW	Line X0, Y0 to X0+X, Y0+Y.
23A3	DR-SIN-NZ	DEFB	+C0,st-mem-0	(SIN (G/2) is copied to mem-0)
		DEFB	+02,delete	X, Y are now on the stack.
		DEFB	+C1,st-mem-1	(Y is copied to mem-1).
		DEFB	+02,delete	X
		DEFB	+31,duplicate	X, X
		DEFB	+2A,abs	X, X' ($X' = \text{ABS } X$)
		DEFB	+E1,get-mem-1	X, X', Y
		DEFB	+01,exchange	X, Y, X'
		DEFB	+E1,get-mem-1	X, Y, X', Y
		DEFB	+2A,abs	X, Y, X', Y' ($Y' = \text{ABS } Y$)
		DEFB	+0F,addition	X, Y, X'+Y'
		DEFB	+E0,get-mem-0	X, Y, X'+Y', SIN (G/2)
		DEFB	+05,division	X, Y, (X'+Y')/SIN (G/2)=Z', say
		DEFB	+2A,abs	X, Y, Z ($Z = \text{ABS } Z'$)
		DEFB	+E0,get-mem-0	X, Y, Z, SIN (G/2)
		DEFB	+01,exchange	X, Y, SIN (G/2), Z
		DEFB	+3D,re-stack	(Z is re-stacked to make sure
		DEFB	+38,end-calc	that its exponent is available).
		LD	A,(HL)	Get exponent of Z.
		CP	+81	If Z is greater than or equal
		JR	NC,23C1,DR-PRMS	to 1, jump.
		RST	0028,FP-CALC	X, Y, SIN (G/2), Z
		DEFB	+02,delete	X, Y, SIN (G/2)
		DEFB	+02,delete	X, Y
		DEFB	+38,end-calc	Just draw the line from X0, Y0
		JP	2477,LINE-DRAW	to X0+X, Y0+Y.

ii. Just calls CD-PRMS1. This subroutine saves in the B register the number of shorter arcs required for the complete arc, viz. $A=4 \cdot \text{INT}(G \cdot \text{SQR } Z/8)+4$, where $G' = \text{mod } G$, or 252 if this expression exceeds 252 (as can happen with a large chord and a small angle). So A is 4, 8, 12, ..., up to 252. The subroutine also stores in mem-0 to mem-4 the quantities G/A , $\text{SIN}(G/2 \cdot A)$, 0, $\text{COS}(G/A)$, $\text{SIN}(G/A)$.

23C1 DR-PRMS CALL 247D,CD-PRMS1 The subroutine is called.

iii. Sets up the rest of the parameters as follow. The stack will hold these 4 items, reading up to the top: X_0+X and Y_0+Y as end of last arc; then X_0 and Y_0 as beginning of first arc. Mem-0 will hold X_0 and mem-5 Y_0 . Mem-1 and mem-2 will hold the initial displacements for the first arc, U and V; and mem-3 and mem-4 will hold $\text{COS}(G/A)$ and $\text{SIN}(G/A)$ for use in the arc-drawing loop.

The formulae for U and V can be explained as follows. Instead of stepping along the final chord, of length L, say, with displacements X and Y, we want to step along an initial chord (which may be longer) of length $L \cdot W$, where $W = \text{SIN}(G/2 \cdot A) / \text{SIN}(G/2)$, with displacements $X \cdot W$ and $Y \cdot W$, but turned through an angle $-(G/2 - G/2 \cdot A)$, hence with true displacements:

$$U = Y \cdot W \cdot \text{SIN}(G/2 - G/2 \cdot A) + X \cdot W \cdot \text{COS}(G/2 - G/2 \cdot A)$$

$$Y = Y \cdot W \cdot \text{COS}(G/2 - G/2 \cdot A) - X \cdot W \cdot \text{SIN}(G/2 - G/2 \cdot A)$$

These formulae can be checked from a diagram, using the normal expansion of $\text{COS}(P - Q)$ and $\text{SIN}(P - Q)$, where $Q = G/2 - G/2 \cdot A$

23C4	PUSH	BC	Save the arc-counter in B.
	RST	0028,FP-CALC	X,Y,SIN(G/2),Z
	DEFB	+02,delete	X,Y,SIN(G/2)
	DEFB	+E1,get-mem-1	X,Y,SIN(G/2),SIN(G/2*A)
	DEFB	+01,exchange	X,Y,SIN(G/2*A),SIN(G/2)
	DEFB	+05,division	X,Y,SIN(G/2*A)/SIN(G/2)=W
	DEFB	+C1,st-mem-1	(W is copied to mem-1).

DEFB	+02,delete	X,Y
DEFB	+01,exchange	Y,X
DEFB	+31,duplicate	Y,X,X
DEFB	+E1,get-mem-1	Y,X,X,W
DEFB	+04,multiply	Y,X,X*W
DEFB	+C2,st-mem-2	(X*W is copied to mem-2).
DEFB	+02,delete	Y,X
DEFB	+01,exchange	X,Y
DEFB	+31,duplicate	X,Y,Y
DEFB	+E1,get-mem-1	X,Y,Y,W
DEFB	+04,multiply	X,Y,Y*W
DEFB	+E2,get-mem-2	X,Y,Y*W,X*W
DEFB	+E5,get-mem-5	X,Y,Y*W,X*W,G
DEFB	+E0,get-mem-0	X,Y,Y*W,X*W,G,G/A
DEFB	+03,subtract	X,Y,Y*W,X*W,G - G/A
DEFB	+A2,stk-half	X,Y,Y*W,X*W,G - G/A, 1/2
DEFB	+04,multiply	X,Y,Y*W,X*W, G/2 - G/2*A=F
DEFB	+31,duplicate	X,Y,Y*W,X*W, F, F
DEFB	+1F,sin	X,Y,Y*W,X*W, F, SIN F
DEFB	+C5,st-mem-5	(SIN F is copied to mem-5).
DEFB	+02,delete	X,Y,Y*W,X*W,F
DEFB	+20,cos	X,Y,Y*W,X*W, COS F
DEFB	+C0,st-mem-0	(COS F is copied to mem-0).
DEFB	+02,delete	X,Y,Y*W,X*W
DEFB	+C2,st-mem-2	(X*W is copied to mem-2).
DEFB	+02,delete	X,Y,Y*W
DEFB	+C1,st-mem-1	(Y*W is copied to mem-1).
DEFB	+E5,get-mem-5	X,Y,Y*W,SIN F
DEFB	+04,multiply	X,Y,Y*W*SIN F
DEFB	+E0,get-mem-0	X,Y,Y*W*SIN F,X*W
DEFB	+E2,get-mem-2	X,Y,Y*W*SIN F,X*W, COS F
DEFB	+04,multiply	X,Y,Y*W*SIN F,X*W*COS F
DEFB	+0F,addition	X,Y,Y*W*SIN F+X*W*COS F=F
DEFB	+E1,get-mem-1	X,Y,U,Y*W
DEFB	+01,exchange	X,Y,Y*W,U
DEFB	+C1,st-mem-1	(U is copied to mem-1)
DEFB	+02,delete	X,Y,Y*W
DEFB	+E0,get-mem-0	X,Y,Y*W, COS F
DEFB	+04,multiply	X,Y,Y*W*COS F
DEFB	+E2,get-mem-2	X,Y,Y*W*COS F,X*W
DEFB	+E5,get-mem-5	X,Y,Y*W*COS F,X*W, SIN F
DEFB	+04,multiply	X,Y,Y*W*COS F,X*W*SIN F
DEFB	+03,subtract	X,Y,Y*W*COS F - X*W*SIN F = V
DEFB	+C2,st-mem-2	(V is copied to mem-2).
DEFB	+2A,abs	X, Y, V' (V' = ABS V)
DEFB	+E1,get-mem-1	X, Y, V', U
DEFB	+2A,abs	X, Y, V', U' (U' = ABS U)
DEFB	+0F,addition	X, Y, U' + V'
DEFB	+02,delete	X, Y
DEFB	+38,end-calc	(DE now points to U' + V').
LD	A,(DE)	Get exponent of U' + V'
CP	+81	If U' + V' is less than 1, just
POP	BC	tidy the stack and draw the line
JP	C,2477,LINE-DRAW	from X0, Y0 to X0+X, Y0+Y.
PUSH	BC	Otherwise, continue with the
RST	0028,FP-CALC	parameters: X, Y, on the stack.
DEFB	+01,exchange	Y, X
DEFB	+38,end-calc	
LD	A,(COORDS-lo)	Get X0 into A and so
CALL	2D28,STACK-A	on to the stack.
RST	0028,FP-CALC	Y, X, X0

DEFB	+C0,st-mem-0	(X0 is copied to mem-0).
DEFB	+0F,addition	Y, X0 + X
DEFB	+01,exchange	X0+X, Y
DEFB	+38,end-calc	
LD	A,(COORDS-hi)	Get Y0 into A and so
CALL	2D28,STACK-A	on to the stack.
RST	0028,FP-CALC	X0+X, Y, Y0
DEFB	+C5,st-mem-5	(Y0 is copied to mem-5).
DEFB	+0F,addition	X0+X, Y0+Y
DEFB	+E0,get-mem-0	X0+X, Y0+Y, X0
DEFB	+E5,get-mem-5	X0+X, Y0+Y, X0, Y0
DEFB	+38,end calc	
POP	BC	Restore the arc-counter in B.

iv. The arc-drawing loop. This is entered at 2439 with the co-ordinates of the starting point on top of the stack, and the initial displacements for the first arc in mem-1 and mem-2. It uses simple trigonometry to ensure that all subsequent arcs will be drawn to points that lie on the same circle as the first two, subtending the same angle at the centre. It can be shown that if 2 points X1, Y1 and X2, Y2 lie on a circle and subtend an angle N at the centre, which is also the origin of co-ordinates, then $X2 = X1 \cdot \cos N - Y1 \cdot \sin N$, and $Y2 = X1 \cdot \sin N + Y1 \cdot \cos N$. But because the origin is here at the increments, say $Un = Xn+1 - Xn$ and $Vn = Yn+1 - Yn$, thus achieving the desired result. The stack is shown below on the (n+1)th pass through the loop, as Xn and Yn are incremented by Un and Vn , after these are obtained from $Un-1$ and $Vn-1$. The 4 values on the top of the stack at 2425 are, in DRAW, reading upwards, $X0+X$, $Y0+Y$, Xn and Yn but to save space these are not shown until 2439. For the initial values in CIRCLE, see the end of CIRCLE, above. In CIRCLE too, the angle G must be taken to be $2^\circ P$.

2420	DRW-STEPS	DEC	B	B counts the passes through the loop.
		JR	Z,245F,ARC-END	Jump when B has reached zero.
		JR	2439,ARC-START	Jump into the loop to start.
2425	ARC-LOOP	RST	0028,FP-CALC	(See text above for the stack).
		DEFB	+E1,get-mem-1	$Un-1$
		DEFB	+31,duplicate	$Un-1, Un-1$
		DEBF	+E3,get-mem-3	$Un-1, Un-1, \cos(G/A)$
		DEFB	+04,multiply	$Un-1, Un-1 \cdot \cos(G/A)$
		DEFB	+E2,get-mem-2	$Un-1, Un-1 \cdot \cos(G/A), Vn-1$
		DEFB	+E4,get-mem-4	$Un-1, Un-1 \cdot \cos(G/A), Vn-1,$
				$\sin(G/A)$
		DEFB	+04,multiply	$Un-1, Un-1 \cdot \cos(G/A), Vn-1 \cdot$
				$\sin(G/A)$
		DEFB	+03,subtract	$Un-1, Un-1 \cdot \cos(G/A) - Vn-1 \cdot$
				$\sin(G/A) = Un$
		DEFB	+C1,st-mem-1	(Un is copied to mem-1).
		DEFB	+02,delete	$Un-1$
		DEFB	+E4,get-mem-4	$Un-1, \sin(G/A)$
		DEFB	+04,multiply	$Un-1 \cdot \sin(G/A)$
		DEFB	+E2,get-mem-2	$Un-1 \cdot \sin(G/A), Vn-1$
		DEFB	+E3,get-mem-3	$Un-1 \cdot \sin(G/A), Vn-1, \cos(G/A)$
		DEFB	+04,multiply	$Un-1 \cdot \sin(G/A), Vn-1 \cdot \cos(G/A)$
		DEFB	+0F,addition	$Un-1 \cdot \sin(G/A) + Vn-1 \cdot \cos$
				$(G/A) = Vn$
		DEFB	+C2,st-mem-2	(Vn is copied to mem-2).
		DEFB	+02,delete	(As noted in the text, the stack
		DEFB	+38,end-calc	in fact holds $X0+X, Y0+Y, Xn$
				and Yn).
2439	ARC-START	PUSH	BC	Save the arc-counter.
		RST	0028,FP-CALC	$X0+X, Y0+y, Xn, Yn$
		DEFB	+C0,st-mem-0	(Yn is copied to mem-0).
		DEFB	+02,delete	$X0+X, Y0+Y, Xn$
		DEFB	+E1,get-mem-1	$X0+X, Y0+Y, Xn, Un$
		DEFB	+0F,addition	$X0+X, Y0+Y, Xn+Un = Xn+1$

		DEFB	+31,duplicate	X0+X, Y0+Y, Xn+1, Xn+1
		DEFB	+38,end-calc	Next Xn', the approximate value of Xn reached by the line-drawing subroutine is copied to A and hence to the stack.
		LD	A,(COORDS-lo)	X0+X, Y0+Y, Xn+1, Xn'
		CALL	2D28,STACK-A	X0+X, Y0+Y, Xn+1, Xn'
		RST	0028,FP-CALC	X0+X, Y0+Y, Xn+1, Xn'
		DEFB	+03,subtract	- Xn' = Un'
		DEFB	+E0,get-mem-0	X0+X, Y0+Y, Xn+1, Un', Yn
		DEFB	+E2,get-mem-2	X0+X, Y0+Y, Xn+1, Un', Yn, Vn
		DEFB	+0F,addition	X0+X, Y0+Y, Xn+1, Un', Yn + Vn = Yn+1
		DEFB	+C0,st-mem-0	(Yn+1 is copied to mem-0).
		DEFB	+01,exchange	X0+X, Y0+Y, Xn+1, Yn+1, Un'
		DEFB	+E0,get-mem-0	X0+X, Y0+Y, Xn+1, Yn+1, Un', Yn+1
		DEFB	+38,end-calc	
		LD	A,(COORDS-hi)	Yn', approximate like Xn', is copied to A and hence to the stack.
		CALL	2D28,STACK-A	
		RST	0028,FP-CALC	X0+X, Y0+Y, Xn+1, Yn+1, Un', Yn+1, Yn'
		DEFB	+03,subtract	X0+X, Y0+Y, Xn+1, Yn+1, Un', Vn'
		DEFB	+38,end-calc	
		CALL	24B7,DRAW-LINE	The next 'arc' is drawn.
		POP	BC	The arc-counter is restored.
		DJNZ	2425,ARC-LOOP	Jump if more arcs to draw.
245F	ARC-END	RST	0028,FP-CALC	The co-ordinates of the end of the last arc that was drawn are now deleted from the stack.
		DEFB	+02,delete	Y0+Y, X0+X
		DEFB	+02,delete	
		DEFB	+01,exchange	
		DEFB	+38,end-calc	
		LD	A,(COORDS-lo)	The X-co-ordinate of the end of the last arc that was drawn, say Xz', is copied to the stack.
		CALL	2D28,STACK-A	Y0+Y, X0+X - Xz'
		RST	0028,FP-CALC	X0+X - Xz', Y0+Y
		DEFB	+03,subtract	
		DEFB	+01,exchange	
		DEFB	+38,end-calc	
		LD	A,(COORDS-hi)	The Y-co-ordinate is obtained.
		CALL	2D28,STACK-A	
		RST	0028,FP-CALC	X0+X - Xz', Y0+Y, Yz'
		DEFB	+03,subtract	X0+X - Xz', Y0+Y - Yz'
		DEFB	+38,end-calc	
2477	LINE-DRAW	CALL	24B7,DRAW-LINE	The final arc is drawn to reach X0+X, Y0+Y (or close the circle).
		JP	0D4D,TEMPS	Exit, setting temporary colours.

THE 'INITIAL PARAMETERS' SUBROUTINE

This subroutine is called by both CIRCLE and DRAW to set their initial parameters. It is called by CIRCLE with X, Y and the radius Z on the top of the stack, reading upwards. It is called by DRAW with its own X, Y, SIN (G/2) and Z, as defined in DRAW i. above, on the top of the stack. In what follows the stack is only shown from Z upwards.

The subroutine returns in B the arc-count A as explained in both CIRCLE and DRAW above, and in mem-0 to mem-5 the quantities G/A, SIN (G/2*A), 0, COS (G/A), SIN (G/A) and G. For a circle, G must be taken to be equal to 2*PI.

247D	CD-PRMS1	RST	0028,FP-CALC	Z
		DEFB	+31,duplicate	Z, Z
		DEFB	+28,sqr	Z, SQR Z
		DEFB	+34,stk-data	Z, SQR Z, 2

		DEFB	+32,exponent +82	
		DEFB	+00,(+00,+00,+00)	
		DEFB	+01,exchange	Z, 2, SQR Z
		DEFB	+05,division	Z, 2/SQR Z
		DEFB	+E5,get-mem-5	Z, 2/SQR Z, G
		DEFB	+01,exchange	Z, G, 2/SQR Z
		DEFB	+05,division	Z, G*SQR Z/2
		DEFB	+2A,abs	Z, G*SQR Z/2 (G' = mod G)
		DEFB	+38,end-calc	Z, G*SQR Z/2 = A1, say
		CALL	2DD5,FP-TO-A	A1 to A from the stack, if possible.
		JR	C,2495,USE-252	If A1 rounds to 256 or more, use 252.
		AND	+FC	4*INT (A1/4) to A.
		ADD	A,+04	Add 4, giving the arc-count A.
		JR	NC,2497,DRAW-SAVE	Jump if still under 256.
2495	USE-252	LD	A,+FC	Here, just use 252 decimal.
2497	DRAW-SAVE	PUSH	AF	Now save the arc-count.
		CALL	2D28,STACK-A	Copy it to calculator stack too.
		RST	0028,FP-CALC	Z, A
		DEFB	+E5,get-mem-5	Z, A, G
		DEFB	+01,exchange	Z, G, A
		DEFB	+05,division	Z, G/A
		DEFB	+31,duplicate	Z,G/A, G/A
		DEFB	+1F,sin	Z, G/A, SIN (G/A)
		DEFB	+C4,st-mem-4	(SIN (G/A) is copied to mem-4).
		DEFB	+02,delete	Z, G/A
		DEFB	+31,duplicate	Z, G/A, G/A
		DEFB	+A2,stk-half	Z, G/A, G/A, 0.5
		DEFB	+04,multiply	Z, G/A, G/2*A
		DEFB	+1F,sin	Z, G/A, SIN (G/2*A)
		DEFB	+C1,st-mem-1	(SIN (G/2*A) is copied to mem-1).
		DEFB	+01,exchange	Z, SIN (G/2*A), G/A
		DEFB	+C0,st-mem-0	(G/A is copied to mem-0).
		DEFB	+02,delete	Z, SIN (G/2*A) = S
		DEFB	+31,duplicate	Z, S, S
		DEFB	+04,multiply	Z, S*S
		DEFB	+31,duplicate	Z, S*S, S*S
		DEFB	+0F,addition	Z, 2*S*S
		DEFB	+A1,stk-one	Z, 2*S*S, 1
		DEFB	+03,subtract	Z, 2*S*S - 1
		DEFB	+1B,negate	Z, 1 - 2*S*S = COS (G/A)
		DEFB	+C3,st-mem-3	(COS (G/A) is copied to mem-3).
		DEFB	+02,delete	Z
		DEFB	+38,end-calc	
		POP	BC	Restore the arc-count to B.
		RET		Finished.

THE LINE-DRAWING SUBROUTINE

This subroutine is called by DRAW to draw an approximation to a straight line from the point X0, Y0 held in COORDS to the point X0+X, Y0+Y, where the increments X and Y are on the top of the calculator stack. The subroutine was originally intended for the ZX80 and ZX81 8K ROM, and it is described in a BASIC program on page 121 of the ZX81 manual. It is also illustrated here in the Circle program in the appendix.

The method is to intersperse as many horizontal or vertical steps as are needed among a basic set of diagonal steps, using an algorithm that spaces the horizontal or vertical steps as evenly as possible.

24B7	DRAW-LINE	CALL	2307,STK-TO-BC	ABS Y to B; ABS X to C; SGN Y to D; SGN X to E.
		LD	A,C	Jump if ABS X is greater than
		CP	B	or equal to ABS Y, so that the

24C4	DL-X-GE-Y	JR	NC,24C4,DL-X-GE-Y	smaller goes to L, and the
		LD	L,C	larger (later) goes to H.
		PUSH	DE	Save diag. step ($\pm 1, \pm 1$) in DE.
		XOR	A	Insert a vertical step ($\pm 1, 0$)
		LD	E,A	into DE (D holds SGN Y).
		JR	24CB,DL-LARGER	Now jump to set H.
		OR	C	Return if ABS X and ABS Y
		RET	Z	are both zero.
		LD	L,B	The smaller (ABS Y here) goes
				to L.
		LD	B,C	ABS X to B here, for H.
		PUSH	DE	Save the diagonal step here too.
24CB	DL-LARGER	LD	D,+00	Hor. step ($0, \pm 1$) to DE here.
		LD	H,B	Larger of ABS X, ABS Y to H
				now.

The algorithm starts here. The larger of ABS X and ABS Y, say H, is put into A and reduced to INT (H/2). The H - L horizontal or vertical steps and L diagonal steps are taken (where L is the smaller of ABS X and ABS Y) in this way: L is added to A; if A now equals or exceeds H, it is reduced by H and a diagonal step is taken; otherwise a horizontal or vertical step is taken. This is repeated H times (B also holds H). Note that meanwhile the exchange registers H' and L' are used to hold COORDS.

24CE	D-L-LOOP	LD	A,B	B to A as well as to H.
		RRA		A starts at INT (H/2).
		ADD	A,L	L is added to A.
		JR	C,24D4,D-L-DIAG	If 256 or more, jump - diag.
				step.
		CP	H	If A is less than H, jump for
		JR	C,24DB,D-L-HR-VT	horizontal or vertical step.
		SUB	H	Reduce A by H.
		LD	C,A	Restore it to C.
		EXX		Now use the exchange registers.
		POP	BC	Diag. step to B'C'.
		PUSH	BC	Save it too.
		JR	24DF,D-L-STEP	Jump to take the step.
		LD	C,A	Save A (unreduced) in C.
		PUSH	DE	Step to stack briefly.
		EXX		Get exchange registers.
		POP	BC	Step to B'C' now.
		LD	HL,(COORDS)	Now take the step: first,
				COORDS to H'L' as the start
				point.
		LD	A,B	Y-step from B' to A.
		ADD	A,H	Add in H'.
		LD	B,A	Result to B'.
		LD	A,C	Now the X-step; it will be tested
		INC	A	for range (Y will be tested in
				PLOT).
		ADD	A,L	Add L' to C' in A, jump on
		JR	C,24F7,D-L-RANGE	carry for further test.
		JR	Z,24F9,REPORT-B	Zero after no carry denotes
				X-position -1, out of range.
		DEC	A	Restore true value to A.
		LD	C,A	Value to C' for plotting.
		CALL	22E5,PLOT-SUB	Plot the step.
		EXX		Restore main registers.
		LD	A,C	C back to A to continue
				algorithm.
		DJNZ	24CE,D-L-LOOP	Loop back for 8 steps (i.e. H
				steps).
		POP	DE	Clear machine stack.
		RET		Finished.

24F7	D-L-RANGE	JR	Z,24EC,D-L-PLOT	Zero after carry denotes X. position 255, in range.
Report B - Integer out of range				
24F9	REPORT-B	RST DEFB	0008,ERROR-1 +0A	Call the error handling routine.

EXPRESSION EVALUATION

THE 'SCANNING' SUBROUTINE

This subroutine is used to produce an evaluation result of the 'next expression'.

The result is returned as the 'last value' on the calculator stack. For a numerical result, the last value will be the actual floating point number. However, for a string result the last value will consist of a set of parameters. The first of the five bytes is unspecified, the second and third bytes hold the address of the start of the string and the fourth and fifth bytes hold the length of the string.

Bit 6 of FLAGS is set for a numeric result and reset for a string result.

When a next expression consists of only a single operand, e.g. ... A ..., ... RND ..., ... A\$ (4, 3 TO 7) ... , then the last value is simply the value that is obtained from evaluating the operand.

However when the next expression contains a function and an operand, e.g. ... CHR\$ A..., ... NOT A ... , SIN 1 ..., the operation code of the function is stored on the machine stack until the last value of the operand has been calculated. This last value is then subjected to the appropriate operation to give a new last value.

In the case of there being an arithmetic or logical operation to be performed, e.g. ... A+B ..., A*B ..., ... A=B ... , then both the last value of the first argument and the operation code have to be kept until the last value of the second argument has been found. Indeed the calculation of the last value of the second argument may also involve the storing of last values and operation codes whilst the calculation is being performed.

It can therefore be shown that as a complex expression is evaluated, e.g. ... CHR\$ (T+A - 26*INT ((T+A)/26)+65)... , a hierarchy of operations yet to be performed is built up until the point is reached from which it must be dismantled to produce the final last value.

Each operation code has associated with it an appropriate priority code and operations of higher priority are always performed before those of lower priority.

The subroutine begins with the A register being set to hold the first character of the expression and a starting priority marker - zero - being put on the machine stack.

24FB	SCANNING	RST	0018,GET-CHAR	The first character is fetched.
		LD	B,+00	The starting priority marker.
		PUSH	BC	It is stacked.
24FF	S-LOOP-1	LD	C,A	The main re-entry point.
		LD	HL,+2596	Index into scanning function
		CALL	16DC,INDEXER	table with the code in C.
		LD	A,C	Restore the code to A.
		JP	NC,2684,S-ALPHNUM	Jump if code not found in table.
		LD	B,+00	Use the entry found in the table
		LD	C,(HL)	to build up the required address
		ADD	HL,BC	in HL, and jump to it.
		JP	(HL)	

Four subroutines follow; they are called by routines from the scanning function table. The first one, the 'scanning quotes subroutine', is used by S-QUOTE to check that every string quote is matched by another one.

250F	S-QUOTE-S	CALL	0074,CH-ADD+1	Point to the next character.
		INC	BC	Increase the length count by
				one.
		CP	+0D	Is it a carriage return?
		JP	Z,1C8A,REPORT-C	Report the error if so.
		CP	+22	Is it another ""?
		JR	NZ,250F,S-QUOTE-S	Loop back if it is not.
		CALL	0074,CH-ADD+1	Point to next character; set zero
		CP	+22	flag if it is another "".
		RET		Finished.

The next subroutine, the 'scanning: two co-ordinates' subroutine, is called by S-SCREEN\$, S-ATTR and S-POINT to make sure the required two co-ordinates are given in their proper form.

2522	S-2-COORD	RST	0020, NEXT-CHAR	Fetch the next character.
		CP	+28	Is it a '('?

		JR	NZ,252D,S-RPORT-C	Report the error if it is not.
		CALL	1C79,NEXT-2NUM	Co-ordinates to calculator stack.
		RST	0018,GET-CHAR	Fetch the current character.
		CP	+29	Is it a ')?
252D	S-RPORT-C	JP	NZ,1C8A,REPORT-C	Report the error if it is not.

THE 'SYNTAX-Z' SUBROUTINE

At this point the 'SYNTAX-Z' subroutine is interpolated. It is called 32 times, with a saving of just one byte each call. A simple test of bit 7 of FLAGS will give the zero flag reset during execution and set during syntax checking.
i.e. SYNTAX gives Z set.

2530	SYNTAX-Z	BIT	7,(FLAGS)	Test bit 7 of FLAGS.
		RET		Finished.

The next subroutine is the 'scanning SCREEN\$ subroutine', which is used by S-SCREENS\$ to find the character that appears at line x, column y of the screen. It only searches the character set 'pointed to' to CHARS.

Note: This is normally the characters +20 (space) to +7F (©) although the user can alter CHARS to match for other characters, including user-defined graphics.

2535	S-SCRN\$-S	CALL	2307,STK-TO-BC	x to C, y to B; 0<=x<=23 decimal; 0<=y<=31 decimal.
		LD	HL,(CHARS)	CHARS plus 256 decimal gives HL pointing to the character set.
		LD	DE,+0100	x is copied to A.
		ADD	HL,DE	The number 32 (decimal) * (x mod 8) + y is formed in A and copied to E.
		LD	A,C	This is the low byte of the required screen address.
		RRCA		
		RRCA		
		AND	+E0	
		XOR	B	
		LD	E,A	
		LD	A,C	x is copied to A again
		AND	+18	Now the number 64 (decimal) + 8*INT (x/8) is inserted into D.
		XOR	+40	DE now holds the screen address.
		LD	D,A	B counts the 96 characters.
		LD	B,+60	Save the count.
254F	S-SCRN-LP	PUSH	BC	And the screen pointer.
		PUSH	DE	And the character set pointer.
		PUSH	HL	Get first row of screen character.
		LD	A,(DE)	Match with row from character set.
		XOR	(HL)	Jump if direct match found.
		JR	Z,255A,S-SC-MTCH	Now test for match with inverse character (get +00 in A from +FF).
		INC	A	Jump if neither match found.
		JR	NZ,2573,S-SCR-NXT	Restore +FF to A.
255A	S-SC-MTCH	DEC	A	Inverse status (+00 or +FF) to C.
		LD	C,A	B counts through the other 7 rows.
		LD	B,+07	Move DE to next row (add 256 dec.).
255D	S-SC-ROWS	INC	D	Move HL to next row (i.e. next byte).
		INC	HL	Get the screen row.
		LD	A,(DE)	Match with row from the ROM.
		XOR	(HL)	Include the inverse status.
		XOR	C	Jump if row fails to match.
		JR	NZ,2573,S-SCR-NXT	Jump back till all rows done.
		DJNZ	255D,S-SC-ROWS	Discard character set pointer.
		POP	BC	And screen pointer.
		POP	BC	

		POP	BC	Final count to BC.
		LD	A,+80	Last character code in set plus one.
		SUB	B	A now holds required code.
		LD	BC,+0001	One space is now needed in the work space.
		RST	0030,BC-SPACES	Make the space.
		LD	(DE),A	Put the character into it.
		JR	257D,S-SCR-STO	Jump to stack the character.
2573	S-SCR-NXT	POP	HL	Restore character set pointer.
		LD	DE,+0008	Move it on 8 bytes, to the next character in the set.
		ADD	HL,DE	
		POP	DE	Restore the screen pointer.
		POP	BC	And the counter.
		DJNZ	254F,S-SCRN-LP	Loop back for the 96 characters.
		LD	C,B	Stack the empty string (Length zero).
257D	S-SCR-STO	JP	2AB2,STK-STO-\$	Jump to stack the matching character, or the null string if no match is found.

Note: This exit, via STK-STO-\$, is a mistake as it leads to 'double storing' of the string result (see S-STRING, 25DB). The instruction line should be 'RET'.

The last of these four subroutines is the 'scanning attributes subroutine'. It is called by S-ATTR to return the value of ATTR (x,y) which codes the attributes of line x, column y on the television screen.

2580	S-ATTR-S	CALL	2307,STK-TO-BC	x to C, y to B. Again, $0 \leq x \leq 23$ decimal; $0 \leq y \leq 31$ decimal.
		LD	A,C	x is copied to A and the number
		RRCA		$32 \text{ (decimal)} * x \text{ (mod } 8) + y$ is
		RRCA		formed in A and copied to L.
		RRCA		$32 * x \text{ (mod } 8) + \text{INT}(x/8)$ is also
		LD	C,A	copied to C.
		AND	+E0	
		XOR	B	
		LD	L,A	L holds low byte of attribute address.
		LD	A,C	$32 * x \text{ (mod } 8) + \text{INT}(x/8)$ is
				copied to A.
		AND	+03	$88 \text{ (decimal)} + \text{INT}(x/8)$ is
		XOR	+58	formed in A and copied to H.
		LD	H,A	H holds high byte of attribute address.
		LD	A,(HL)	The attribute byte is copied to A.
		JP	2D28,STACK-A	Exit, stacking the required byte.

THE SCANNING FUNCTION TABLE

This table contains 8 functions and 4 operators. It thus incorporates 5 new Spectrum functions and provides a neat way of accessing some functions and operators which already existed on the ZX81.

location	code	offset	name	address of handling routine
2596	22	1C	S-QUOTE	25B3
2598	28	4F	S-BRACKET	25E8
259A	2E	F2	S-DECIMAL	268D
259C	2B	12	S-U-PLUS	25AF
259E	A8	56	S-FN	25F5
25A0	A5	57	S-AND	25F8
25A2	A7	84	S-PI	2627
25A4	A6	8F	S-INKEY\$	2634
25A6	C4	E6	S-BIN (EQU. S-DECIMAL)	268D
25A8	AA	BF	S-SCREEN\$	2668
25AA	AB	C7	S-ATTR	2672

25AC	A9	CE	S-POINT	267B
25AE	00		End-marker	

THE SCANNING FUNCTION ROUTINES

25AF	S-U-PLUS	RST JP	0020,NEXTCHAR 24FF,S-LOOP-1	For unary plus, simply move on to the next character and jump back to the main re-entry of SCANNING.
------	----------	-----------	--------------------------------	--

The 'scanning QUOTE routine': This routine deals with string quotes, whether simple like "name" or more complex like "a ""white"" lie" or the seemingly redundant VAL\$ ""a"".

25B3	S-QUOTE	RST INC PUSH LD CALL JR	0018,GET-CHAR HL HL BC,+0000 250F,S-QUOTE-S NZ,25D9,S-Q-PRMS	Fetch the current character. Point to the start of the string. Save the start address. Set the length to zero. Call the "matching" subroutine. Jump if zero reset - no more quotes.
25BE	S-Q-AGAIN	CALL JR CALL JR RST POP PUSH	250F,S-QUOTE-S Z,25BE,S-Q-AGAIN 2530,SYNTAX-Z Z,25D9,S-Q-PRMS 0030,BC-SPACES HL DE	Call it again for a third quote. And again for the fifth, seventh etc. If testing syntax, jump to reset bit 6 of FLAGS and to continue scanning. Make space in the work space for the string and the terminating quote. Get the pointer to the start. Save the pointer to the first space.
25CB	S-Q-COPY	LD INC LD INC CP JR LD INC CP JR	A,(HL) HL (DE),A DE +22 NZ,25CB,S-Q-COPY A,(HL) HL +22 Z,25CB,S-Q-COPY	Get a character from the string. Point to the next one. Copy last one to work space. Point to the next space. Is last character a ""? If not, jump to copy next one. But if it was, do not copy next one; if next one is a "", jump to copy the one after it; otherwise, finished with copying.
25D9	S-Q-PRMS	DEC	BC	Get true length to BC.

Note that the first quote was not counted into the length; the final quote was, and is discarded now. Inside the string, the first, third, fifth, etc., quotes were counted in but the second, fourth, etc., were not.

25DB	S-STRING	POP LD RES BIT CALL JP	DE HL,+5C3B 6,(HL) 7,(HL) NZ,2AB2,STK-STO-S 2712,S-CONT-2	Restore start of copied string. This is FLAGS; this entry point is used whenever bit 6 is to be reset and a string stacked if executing a line. This is done now. Jump to continue scanning the line.
------	----------	---------------------------------------	--	---

Note that in copying the string to the work space, every two pairs of string quotes (""") have been reduced to one pair of string quotes(").

25E8	S-BRACKET	RST CALL CP JP RST	0020,NEXT-CHAR 24FB,SCANNING +29 NZ,1C8A,REPORT-C 0020,NEXT-CHAR	The 'scanning BRACKET routine' simply gets the character and calls SCANNING recursively. Report the error if no matching bracket; then continue scanning.
------	-----------	------------------------------------	--	--

25F5	S-FN	JP JP	2712,S-CONT-2 27BD,S-FN-SBRN The 'scanning FN routine'.	
This routine, for user-defined functions, just jumps to the 'scanning FN subroutine'.				
25F8	S-RND	CALL JR LD CALL RST DEFB DEFB DEFB DEFB DEFB DEFB DEFB DEFB DEFB DEFB DEFB DEFB DEFB DEFB DEFB CALL LD LD AND JR SUB LD	2530,SYNTAX-Z Z,2626,S-RND-END BC,(SEED) 2D2B,STACK-BC 0028,FP-CALC +A1,stk-one +0F,addition +34,stk-data +37,exponent+87 +16,(+00,+00,+00) +04,multiply +34,stk-data +80,(four bytes) +41,exponent +91 +00,+00,+80,(+00) +32,n-mod-m +02,delete +A1,stk-one +03,subtract +31,duplicate +38,end-calc 2DA2,FP-TO-BC (SEED),BC A,(HL) A Z,2625,S-RND-END +10 (HL),A	Unless syntax is being checked, jump to calculate a random number. Fetch the current value of SEED. Put it on the calculator stack. Now use the calculator, The 'last value' is now SEED+1. Put the decimal number 75 on the calculator stack. 'last value' (SEED+1)*75. See STACK LITERALS to see how bytes are expanded so as to put the decimal number 65537 on the calculator stack. Divide (SEED+1)*75 by 65537 to give a 'remainder' and an 'answer'. Discard the 'answer'. The 'last value' is now 'remainder' - 1. Make a copy of the 'last value'. The calculation is finished. Use the 'last value' to give the new value for SEED. Fetch the exponent of 'last value'. Jump forward if the exponent is zero. Reduce the exponent, i.e. divide 'last value' by 65536 to give the required 'last value'. Jump past the 'PI' routine.
2625	S-RND-END	JR	2630,S-PI-END	

The 'scanning-PI routine': unless syntax is being checked the value of 'PI' is calculated and forms the 'last value' on the calculator stack.

2627	S-PI	CALL JR RST DEFB DEFB INC	2530,SYNTAX-Z Z,2630,S-PI-END 0028,FP-CALC +A3,stk-pi/2 +3B,end-calc (HL)	Test for syntax checking. Jump if required. Now use the calculator. The value of PI/2 is put on the calculator stack as the 'last value'. The exponent is incremented thereby doubling the 'last value' giving PI.
2630	S-PI-END	RST JP	0020,NEXT-CHAR 26C3,S-NUMERIC	Move on to the next character. Jump forward.
2634	S-INKEY\$	LD RST CP JP LD RES BIT	BC,+105A 0020,NEXT-CHAR +23 Z,270D,S-PUSH-PO HL,+5C3B 6,(HL) 7,(HL)	Priority +10 hex, operation code +5A for the 'read-in' subroutine. If next char. is '#', jump. There will be a numerical argument. This is FLAGS. Reset bit 6 for a string result. Test for syntax checking.

		JR	Z,2665,S-INK\$-EN	Jump if required.
		CALL	028E,KEY-SCAN	Fetch a key-value in DE.
		LD	C,+00	Prepare empty string; stack it if
		JR	NZ,2660,S-IK\$-STK	too many keys pressed.
		CALL	031E,K-TEST	Test the key value; stack empty
		JR	NC,2660,S-IK\$-STK	string if unsatisfactory.
		DEC	D	+FF to D for L made (bit 3 set).
		LD	E,A	Key-value to E for decoding.
		CALL	0333,K-DECODE	Decode the key-value.
		PUSH	AF	Save the ASCII value briefly.
		LD	BC,+0001	One space is needed in the work
				space.
		RST	0030,BC-SPACES	Make it now.
		POP	AF	Restore the ASCII value.
		LD	(DE),A	Prepare to stack it as a string.
		LD	C,+01	Its length is one.
2660	S-IK\$-STK	LD	B,+00	Complete the length parameter.
		CALL	2AB2,STK-STO-\$	Stack the required string.
2665	S-INK\$-EN	JP	2712,S-CONT-2	Jump forward.
2668	S-SCREEN\$	CALL	2522,S-2-COORD	Check that 2 co-ordinates are
				given.
		CALL	NZ,2535,S-SCRN\$-S	Call the subroutine unless
		RST	0020,NEXT-CHAR	checking syntax; then get next
		JP	25DB,S-STRING	character and jump back.
2672	S-ATTR	CALL	2522,5-2-COORD	Check that 2 co-ordinates are
				given.
		CALL	NZ,2580,S-ATTR-S	Call the subroutine unless
		RST	0020,NEXT-CHAR	checking syntax; then get the
		JR	26C3,S-NUMERIC	next character and jump
				forward.
267B	S-POINT	CALL	2522,S-2-COORD	Check that 2 co-ordinates are
				given.
		CALL	NZ,22CB,POINT-SUB	Call the subroutine unless
		RST	0020,NEXT-CHAR	checking syntax; then get the
		JR	26C3,S-NUMERIC	next character and jump
				forward.
2684	S-ALPHNUM	CALL	2C88,ALPHANUM	Is the character alphanumeric?
		JR	NC,26DF,S-NEGATE	Jump if not a letter or a digit.
		CP	+41	Now jump if it a letter;
		JR	NC,26C9,S-LETTER	otherwise continue on into
				S-DECIMAL.

The 'scanning DECIMAL routine' which follows deals with a decimal point or a number that starts with a digit. It also takes care of the expression 'BIN', which is dealt with in the 'decimal to floating-point' subroutine.

268D	S-DECIMAL	CALL	2530,SYNTAX-Z	Jump forward if a line is
	(EQU. S-BIN)	JR	NZ,2685,S-STK-DEC	being executed.

The action taken is now very different for syntax checking and line execution. If syntax is being checked then the floating-point form has to be calculated and copied into the actual BASIC line. However when a line is being executed the floating-point form will always be available so it is copied to the calculator stack to form a 'last value'.

During syntax checking:

CALL	2C9B,DEC-TO-FP	The floating-point form is
		found.
RST	0018,GET-CHAR	Set HL to point one past the
		last digit.
LD	BC,+0006	Six locations are required.
CALL	1655,MAKE-ROOM	Make the room in the BASIC
		line.
INC	HL	Point to the first free space.
LD	(HL),+0E	Enter the number marker code.
INC	HL	Point to the second location.
EX	DE,HL	This pointer is wanted in DE.

LD	HL,(STKEND)	Fetch the 'old' STKEND.
LD	C,+05	There are 5 bytes to move.
AND	A	Clear the carry flag.
SBC	HL,BC	The 'new' STKEND='old' STKEND -5.
LD	(STKEND),HL	Move the floating-point number from the calculator stack to the line.
EX	DE,HL	Put the line pointer in HL.
DEC	HL	Point to the last byte added.
CALL	0077,TEMP-PTR1	This sets CH-ADD.
JR	26C3,S-NUMERIC	Jump forward.

During line execution:

26B5	S-STK-DEC	RST	0018,GET-CHAR	Get the current character.
26B6	S-SD-SKIP	INC	HL	Now move on to the next
		LD	A,(HL)	character in turn until
		CP	+0E	the number marker code
		JR	NZ,26B6,S-SD-SKIP	is found.
		INC	HL	Point to the first byte of the
				number.
		CALL	33B4,STACK-NUM	Move the floating-point number.
		LD	(CH-ADD),HL	Set CH-ADD.

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

26C3	S-NUMERIC	SET	6,(FLAGS)	Set the numeric marker flag.
		JR	26DD,S-CONT-1	Jump forward.

THE SCANNING VARIABLE ROUTINE

When a variable name has been identified a call is made to LOOK-VARS which looks through those variables that already exist in the variables area (or in the program area at DEF FN statements for a user-defined function FN). If an appropriate numeric value is found then it is copied to the calculator stack using STACK-NUM. However a string or string array entry has to have the appropriate parameters passed to the calculator stack by the STK-VAR subroutine (or in the case of a user-defined function, by the STK-F-ARG subroutine as called from LOOK-VARS).

26C9	S-LETTER	CALL	28B2,LOOK-VARS	Look in the existing variables for the matching entry.
		JP	C,1C2E,REPORT-2	An error is reported if there is no existing entry.
		CALL	Z,2996,STK-VARS	Stack the parameters of the string entry/return numeric element base address.
		LD	A,(FLAGS)	Fetch FLAGS.
		CP	+C0	Test bits 6 and 7 together.
		JR	C,26DD,S-CONT-1	One or both bits are reset.
		INC	HL	A numeric value is to be stacked.
		CALL	33B4,STACK-NUM	Move the number.
26DD	S-CONT-1	JR	2712,S-CONT-2	Jump forward.

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

Before the actual test the B register is set to hold the priority +09 and the C register the operation code +D8 that are required for this operation.

26DF	S-NEGATE	LD	BC,+09DB	Priority +09, operation code +D8.
		CP	+2D	Is it a '-'?
		JR	Z,270D,S-PUSH-PO	Jump forward if it is 'unary minus'.

Next the character is tested against the code for 'VAL\$', with priority 16 decimal and operation code 18 hex.

LD	BC,+1018	Priority 16 dec, operation code +18 hex.
CP	+AE	Is it 'VAL\$'?
JR	Z,270D,S-PUSH-PO	Jump forward if it is 'VAL\$'.

The present character must now represent one of the functions CODE to NOT, with codes +AF to +C3.

SUB	+AF	The range of the functions is changed from +AF to +C3 to range +00 to +14 hex.
JP	C,1C8A,REPORT-C	Report an error if out of range.

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

LD	BC,+04F0	Priority +04, operation code +F0.
CP	+14	Is it the function 'NOT'?
JR	Z,270D,S-PUSH-PO	Jump if it is so.
JP	NC,1C8A,REPORT-C	Check the range again.

The remaining functions have priority 16 decimal. The operation codes for these functions are now calculated. Functions that operate on strings need bit 6 reset and functions that give string results need bit 7 reset in their operation codes.

	LD	B,+10	Priority 16 decimal.
	ADD	A,+DC	The function range is now +DC +EF.
	LD	C,A	Transfer the operation code.
	CP	+DF	Separate CODE, VAL and LEN
	JR	NC,2707.S-NO-TO-S	which operate on strings to give
	RES	6,C	numerical results.
2707	S-NO-TO-S	CP	+EE
	JR	C,2700,S-PUSH-PO	Separate STR\$ and CHR\$
			which operate on numbers to
			give string results.
	RES	7,C	Mark the operation codes.
			The other operation codes have
			bits 6 and 7 both set.

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

270D	S-PUSH-PO	PUSH	BC	Stack the priority and operation
		RST	0020,NEXT-CHAR	codes before moving on to
		JP	24FF,S-LOOP-1	consider the next part of the
				expression.

The scanning of the line now continues. The present argument may be followed by a '(', a binary operator or, if the end of the expression has been reached, then e.g. a carriage return character or a colon, a separator or a 'THEN'.

2712	S-CONT-2	RST	0018,GET-CHAR	Fetch the present character.
2713	S-CONT-3	CP	+28	Jump forward if it is not a '(',
		JR	NZ,2723,S-OPERTR	which 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
JR	NZ,2734,S-LOOP	numeric parenthesised
		expression.
CALL	2A52,SLICING	Modify the parameters of the
		'last value'.
RST	0020,NEXT-CHAR	Move on to consider the next
JR	2713,S-CONT-3	character.

If the present character is indeed a binary operator it will be given an operation code in the range +C3 - +CF hex, and the appropriate priority code.

2723	S-OPERTR	LD	B,+00	Original code to BC to index
		LD	C,A	into table of operators.
		LD	HL,+2795	The pointer to the table.
		CALL	16DC,INDEXER	Index into the table.
		JR	NC,2734,SLOOP	Jump forward if no operation found.
		LD	C,(HL)	Get required code from the table.
		LD	HL,+26ED	The pointer to the priority table: i.e. 26ED +C3 gives 27B0 as the first address.
		ADD	HL,BC	Index into the table.
		LD	B,(HL)	Fetch the appropriate priority.

The main loop of this subroutine is now entered. At this stage there are:

- A 'last value' on the calculator stack.
- The starting priority market on the machine stack below a hierarchy, of unknown size, of function and binary operation codes. This hierarchy may be null.
- The BC register pair holding the 'present' operation and priority, which if the end of an expression has been reached will be priority zero.

Initially the 'last' operation and priority are taken off the machine stack and compared against the 'present' operation and priority.

If the 'present' priority is higher than the 'last' priority then an exit is made from the loop as the 'present' priority is considered to bind tighter than the 'last' priority.

However, if the present priority is less binding, then the operation specified as the 'last' operation is performed. The 'present' operation and priority go back on the machine stack to be carried round the loop again. In this manner the hierarchy of functions and binary operations that have been queued are dealt with in the correct order.

2734	S-LOOP	POP	DE	Get the 'last' operation and priority.
		LD	A,D	The priority goes to the A register.
		CP	B	Compare 'last' against 'present'.
		JR	C,2773,S-TIGHTER	Exit to wait for the argument.
		AND	A	Are both priorities zero?
		JP	Z,0018,GET-CHAR	Exit via GET-CHAR thereby making 'last value' the required result.

Before the 'last' operation is performed, the 'USR' function is separated into 'USR number' and 'USR string' according as bit 6 of FLAGS was set or reset when the argument of the function was stacked as the 'last value'.

		PUSH	BC	Stack the 'present' values.
		LO	HL,+5C3B	This is FLAGS.
		LD	A,E	The 'last' operation is compared
		CP	+ED	with the code for USR, which
		JR	NZ,274C,S-STK-LST	will give 'USR number' unless modified; jump if not 'USR'.
		BIT	6,(HL)	Test bit 6 of FLAGS.
		JR	NZ,274C,S-STK-LST	Jump if it is set ('USR number').
		LD	E,+99	Modify the 'last' operation code: 'offset' 19, +80 for string input and numerical result ('USR string').
274C	S-STK-LST	PUSH	DE	Stack the 'last' values briefly.
		CALL	2530,SYNTAX-Z	Do not perform the actual
		JR	Z,275B,S-SYNTEST	operation if syntax is being checked.

LD	A,E	The 'last' operation code.
AND	+3F	Strip off bits 6 and 7 to convert
LD	B,A	the operation code to a
		calculator offset.
RST	0028,FP-CALC	Now use the calculator.
DEFB	+3B,fp-calc-2	Perform the actual operation
DEFB	+38,end-calc	It has been done.
JR	2764,S-RUNTEST	Jump forward.

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

275B	S-SYNTEST	LD	A,E	Get the 'last' operation code.
		XOR	(FLAGS)	This tests the nature of the 'last
		AND	+40	value' against the requirement
				of the operation. They are to be
2761	S-RPORT-C	JP	NZ,1C8A,REPORT-C	the same for correct syntax.
				Jump if syntax fails.

Before jumping back to go round the loop again the nature of the 'last value' must be recorded in FLAGS.

2764	S-RUNTEST	POP	DE	Get the 'last' operation code.
		LD	HL,+5C3B	This is FLAGS.
		SET	6,(HL)	Assume result to be numeric.
		BIT	7,E	Jump forward if the nature of
		JR	NZ,2770,S-LOOPEND	'last value' is numeric.
		RES	6,(HL)	It is string.
2770	S-LOOPEND	POP	BC	Get the 'present' values into BC:
		JR	2734,S-LOOP	Jump back.

Whenever the 'present' operation binds tighter, the 'last' and the 'present' values go back on the machine stack. However if the 'present' operation requires a string as its operand then the operation code is modified to indicate this requirement.

2773	S-TIGHTER	PUSH	DE	The 'last' values go on the stack.
		LD	A,C	Get the 'present' operation
				code.
		BIT	6,(FLAGS)	Do not modify the operation
		JR	NZ,2790,S-NEXT	code if dealing with a numeric
				operand.
		AND	+3F	Clear bits 6 and 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,2788,S-NOT-AND	Jump if it is not so.
		SET	6,C	'AND' requires a numeric
				operand.
		JR	2790,S-NEXT	Jump forward.
2788	S-NOT-AND	JR	C,2761,S-RPORT-C	The operations -, *, /, ^ and OR
				are not possible between strings.
		CP	+17	Is the operation a '+'?
		JR	Z,2790,S-NEXT	Jump if it is so.
		SET	7,C	The other operations yield a
				numeric result.
2790	S-NEXT	PUSH	BC	The 'present' values go on the
				machine stack.
		RST	0020,NEXT-CHAR	Consider the next character.
		JP	24FF,S-LOOP-1	Go around the loop again.

THE TABLE OF OPERATORS

location	code	operator code	operator	location	code	operator code	operator
2795	2B	CF	+	27A3	3C	CD	<
2797	2D	C3	-	27A5	C7	C9	<=
2799	2A	C4	*	27A7	C8	CA	>=
279B	2F	C5	/	27A9	C9	CB	<>
279D	5E	C6	^	27AB	C5	C7	OR
279F	3D	CE	=	27AD	C6	C8	AND
27A1	3E	CC	>	27AF	00		End marker

THE TABLE OF PRIORITIES (precedence table)

location	priority	operator	location	priority	operator
27B0	06	-	27B7	05	>=
27B1	08	*	27B8	05	<>
27B2	08	/	27B9	05	>
27B3	0A	^	27BA	05	<
27B4	02	OR	27BB	05	=
27B5	03	AND	27BC	06	+
27B6	05	<=			

THE 'SCANNING FUNCTION' SUBROUTINE

This subroutine is called by the 'scanning FN routine' to evaluate a user defined function which occurs in a BASIC line. The subroutine can be considered in four stages:

- The syntax of the FN statement is checked during syntax checking.
- During line execution, a search is made of the program area for a DEF FN statement, and the names of the functions are compared, until a match is found - or an error is reported.
- The arguments of the FN are evaluated by calls to SCANNING.
- The function itself is evaluated by calling SCANNING, which in turn calls LOOK-VARS and so the 'STACK FUNCTION ARGUMENT' subroutine.

27BD	S-FN-SBRN	CALL	2530,SYNTAX-Z	Unless syntax is being checked, a jump is made to SF-RUN. Get the first character of the name. If it is not alphabetic, then report the error. Get the next character. Is it a '\$'? Save the zero flag on the stack. Jump if it was not a '\$'. But get the next character if it was.
		JR	NZ,27F7,SF-RUN	
		RST	0020,NEXT-CHAR	
		CALL	2C8D,ALPHA	
		JP	NC,1C8A,REPORT-C	
27D0	SF-BRKT-1	RST	0020,NEXT-CHAR	If the character is not a '(', then report the error. Get the next character. Is it a ')'? Jump if it is; there are no arguments.
		CP	+24	
		PUSH	AF	
		JR	NZ,27D0,SF-BRKT-1	
		RST	0020,NEXT-CHAR	
27D9	SF-ARGMTS	CP	+28	Within the loop, call SCANNING to check the syntax of each argument and to insert floating-point numbers. Get the character which follows the argument; if it is not a ',' then jump - no more arguments. Get the first character in the next argument.
		JR	NZ,27E6,SF-RPRT-C	
		RST	0020,NEXT-CHAR	
		CP	+29	
		JR	Z,27E9,SF-FLAG-6	
		CALL	24FB,SCANNING	
		RST	0018,GET-CHAR	
		CP	+2C	
		JR	NZ,27E4,SF-BRKT-2	
		RST	0020,NEXT-CHAR	

		JR	27D9,SF-ARGMTS	Loop back to consider this argument.
27E4	SF-BRKT-2	CP	+29	Is the current character a ')'? Report the error if it is not.
27E6	SF-RPRT-C	JP	NZ,1C8A,REPORT-C	Point to the next character in the BASIC line.
27E9	SF-FLAG-6	RST	0020,NEXT-CHAR	This is FLAGS; assume a string-valued function and reset bit 6 of FLAGS.
		LD	HL,+5C3B	Restore the zero flag, jump if the FN is indeed string valued.
		RES	6,(HL)	Otherwise, set bit 6 of FLAGS
		POP	AF	Jump back to continue scanning the line.
		JR	Z,27F4,SF-SYN-EN	
		SET	6,(HL)	
27F4	SF-SYN-EN	JP	2712,S-CONT-2	

ii. During line execution, a search must first be made for a DEF FN statement.

27F7	SF-RUN	RST	0020,NEXT-CHAR	Get the first character of the name.
		AND	+DF	Reset bit 5 for upper case.
		LD	B,A	Copy the name to B.
		RST	0020,NEXT-CHAR	Get the next character.
		SUB	+24	Subtract 24 hex, the code for '\$'.
		LD	C,A	Copy the result to C (zero for a string, non-zero for a numerical function).
		JR	NZ,2802,SF-ARGMT1	Jump if non-zero: numerical function.
2802	SF-ARGMT1	RST	0020,NEXT-CHAR	Get the next character, the '('.
		RST	0020,NEXT-CHAR	Get 1st character of 1st argument.
		PUSH	HL	Save the pointer to it on the stack.
		LD	HL,(PROG)	Point to the start of the program.
2808	SF-FND-DF	DEC	HL	Go back one location.
		LD	DE,+00CE	The search will be for 'DEF FN'.
		PUSH	BC	Save the name and 'string status'.
		CALL	1D86,LOOK-PROG	Search the program now.
		POP	BC	Restore the name and status.
		JR	NC,2814,SF-CP-DEF	Jump if a DEF FN statement found.

REPORT P - FN without DEF.

2812	REPORT-P	RST	0008,ERROR-1	Call the error handling routine.
		DEFB	+18	

When a DEF FN statement is found, the name and status of the two functions are compared: if they do not match, the search is resumed.

2814	SF-CP-DEF	PUSH	HL	Save the pointer to the DEF FN character in case the search has to be resumed.
		CALL	28AB,FN-SKPOVR	Get the name of the DEF FN function.
		AND	+DF	Reset bit 5 for upper case.
		CP	B	Does it match the FN name?
		JR	NZ,2825,SF-NOT-FD	Jump if it does not match.
		CALL	28AB,FN-SKPOVR	Get the next character in the DEF FN.
		SUB	+24	Subtract 24 hex, the code for '\$'.
		CP	C	Compare the status with that of FN.

		JR	Z,2831,SF-VALUES	Jump if complete match now found.
2825	SF-NOT-FD	POP	HL	Restore the pointer to the 'DEF FN'.
		DEC	HL	Step back one location.
		LD	DE,+0200	Use the search routine to find the end of the DEF FN statement, preparing for the next search; save the name and status meanwhile.
		PUSH	BC	
		CALL	198B,EACH-STMT	
		POP	BC	
		JR	2808,SF-FND-DF	Jump back for a further search.

ii. The correct DEF FN statement has now been found. The arguments of the FN statement will be evaluated by repeated calls of SCANNING, and their 5 byte values (or parameters, for strings) will be inserted into the DEF FN statement in the spaces made there at syntax checking. HL will be used to point along the DEF FN statement (calling FN-SKPOVR as needed) while CH-ADD points along the FN statement (calling RST 0020, NEXT-CHAR, as needed).

2831	SF-VALUES	AND	A	If HL is now pointing to a '\$', move on to the '('.
		CALL	Z,28AB,FN-SKPOVR	Discard the pointer to 'DEF FN'.
		POP	DE	
		POP	DE	Get the pointer to the first argument of FN, and copy it to CH-ADD.
		LD	(CH-ADD),DE	
		CALL	28AB,FN-SKPOVR	Move past the '(' now.
		PUSH	HL	Save this pointer on the stack.
		CP	+29	Is it pointing to a ')'? If so, jump: FN has no arguments.
		JR	Z,2885,SF-R-BR-2	
2843	SF-ARG-LP	INC	HL	Point to the next code.
		LD	A,(HL)	Put the code into A.
		CP	+0E	Is it the 'number marker' code, 0E hex?
		LD	D,+40	Set bit 6 of D for a numerical argument.
		JR	Z,2852,SF-ARG-VL	Jump on zero: numerical argument.
		DEC	HL	Now ensure that HL is pointing to the '\$' character (not e.g. to a control code).
		CALL	28AB,FN-SKPOVR	HL now points to the 'number marker'.
		INC	HL	
		LD	D,+00	Bit 6 of D is reset: string argument.
2852	SF-ARG-VL	INC	HL	Point to the 1st of the 5 bytes in DEF FN.
		PUSH	HL	Save this pointer on the stack.
		PUSH	DE	Save the 'string status' of the argument.
		CALL	24FB,SCANNING	Now evaluate the argument.
		POP	AF	Get the no./string flag into A.
		XOR	(FLAGS)	Test bit 6 of it against the result of SCANNING.
		AND	+40	Give report Q if they did not match.
		JR	NZ,288B,REPORT-Q	
		POP	HL	Get the pointer to the first of the 5 spaces in DEF FN into DE.
		EX	DE,HL	
		LD	HL,(STKEND)	Point HL at STKEND.
		LD	BC,+0005	BC will count 5 bytes to be moved.
		SBC	HL,BC	First, decrease STKEND by 5,

		LD	(STKEND),HL	so deleting the 'last value' from the stack.
		LDIR		Copy the 5 bytes into the spaces in DEF FN.
		EX	DE,HL	Point HL at the next code.
		DEC	HL	Ensure that HL points to the character after the 5 bytes.
		CALL	28AB,FN-SKPOVR	Is it a ')?
		CP	+29	Jump if it is: no more arguments in the DEF FN statement.
		JR	Z,2885,SF-R-BR-2	It is a '': save the pointer to it.
		PUSH	HL	Get the character after the last argument that was evaluated from FN.
		RST	0018,GET-CHAR	If it is not a ',', jump: mismatched arguments of FN and DEF FN.
		CP	+2C	Point CH-ADD to the next argument of FN.
		JR	NZ,288B,REPORT-Q	Point HL to the ',' in DEF FN again.
		RST	0020,NEXT-CHAR	Move HL on to the next argument in DEF FN.
		POP	HL	Jump back to consider this argument.
		CALL	28AB,FN-SKPOVR	Save the pointer to the ')' in DEF FN.
		JR	2843,SF-ARG-LP	Get the character after the last argument in FN.
2885	SF-R-BR-2	PUSH	HL	Is it a ')?
		RST	0018,GET-CHAR	If so, jump to evaluate the function; but if not, give report Q.
		CP	+29	
		JR	Z,288D,SF-VALUE	
REPORT Q - Parameter error.				
288B	REPORT-Q	RST	0008,ERROR-1	Call the error handling routine.
		DEFB	+19	

iv. Finally, the function itself is evaluated by calling SCANNING, after first setting DEFADD to hold the address of the arguments as they occur in the DEF FN statement. This ensures that LOOK-VARS, when called by SCANNING, will first search these arguments for the required values, before making a search of the variables area.

288D	SF-VALUE	POP	DE	Restore pointer to ')' in DEF FN.
		EX	DE,HL	Get this pointer into HL.
		LD	(CH-ADD),HL	Insert it into CH-ADD.
		LD	HL,(DEFADD)	Get the old value of DEFADD.
		EX	(SP),HL	Stack it, and get the start address of the arguments area of DEF FN into DEFADD.
		LD	(DEFADD),HL	Save address of ')' in FN.
		PUSH	DE	Move CH-ADD on past ')' and '=' to the start of the expression for the function in DEF FN.
		RST	0020,NEXT-CHAR	Now evaluate the function.
		RST	0020,NEXT-CHAR	Restore the address of ')' in FN.
		CALL	24FB,SCANNING	Store it in CH-ADD.
		POP	HL	Restore original value of DEFADD.
		LD	(CH-ADD),HL	Put it back into DEFADD.
		POP	HL	
		LD	(DEFADD),HL	

RST	0020,NEXT-CHAR	Get the next character in the BASIC line.
JP	2712,S-CONT-2	Jump back to continue scanning.

THE 'FUNCTION SKIPOVER' SUBROUTINE

This subroutine is used by FN and by STK-F-ARG to move HL along the DEF FN statement while leaving CM-ADD undisturbed, as it points along the FN statement.

28AB	FN-SKPOVR	INC	HL	Point to the next code in the statement.
		LD	A,(HL)	Copy the code to A.
		CP	+21	Jump back to skip over it if it is a control code or a space.
		JR	C,28AB,FN-SKPOVR	
		RET		Finished.

THE 'LOOK-VARS' SUBROUTINE

This subroutine is called whenever a search of the variables area or of the arguments of a DEF FN statement is required. The subroutine is entered with the system variable CH-ADD pointing to the first letter of the name of the variable whose location is being sought. The name will be in the program area or the work space. The subroutine initially builds up a discriminator byte, in the C register, that is based on the first letter of the variable's name. Bits 5 & 6 of this byte indicate the type of the variable that is being handled.

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

28B2	LOOK-VARS	SET	6,(FLAGS)	Presume a numeric variable.
		RST	0018,GET-CHAR	Get the first character into A.
		CALL	2C8D,ALPHA	Is it alphabetic?
		JP	NC,1C8A,REPORT-C	Give an error report if it is not so.
		PUSH	HL	Save the pointer to the first letter.
		AND	+1F	Transfer bits 0 to 4 of the letter to the C register; bits 5 & 7 are always reset.
		LD	C,A	
		RST	0020,NEXT-CHAR	Get the 2nd character into A.
		PUSH	HL	Save this pointer also.
		CP	+28	is the 2nd character a '('?
		JR	Z,28EF,V-RUN/SYN	Separate arrays of numbers.
		SET	6,C	Now set bit 6.
		CP	+24	Is the 2nd character a '\$'?
		JR	Z,28DE,V-STR-VAR	Separate all the strings.
		SET	5,C	Now set bit 5.
		CALL	2C88,ALPHANUM	If the variable's name has only one character then jump forward.
		JR	NC,28E3,V-TEST-FN	

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

28D4	V-CHAR	CALL	2C88,ALPHANUM	Is the character alphanumeric?
		JR	NC,28EF,V-RUN/SYN	Jump out of the loop when the end of the name is found.
		RES	6,C	Mark the discriminator byte.
		RST	0020,NEXT-CHAR	Get the next character.
		JR	28D4,V-CHAR	Go back to test it.

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

28DE	V-STR-VAR	RST	0020,NEXT-CHAR	Step CH-ADD past the '\$'.
		RES	6,(FLAGS)	Reset the bit 6 to indicate a string.

If DEFADD-hi is non-zero, indicating that a 'function' (a 'FN') is being evaluated, and if in 'run-time', a search will be made of the arguments in the DEF FN statement.

28E3	V-TEST-FN	LD	A,(DEFADD-hi)	Is DEFADD-hi zero?
------	-----------	----	---------------	--------------------

AND	A	
JR	Z,28EF,V-RUN/SYN	If so, jump forward.
CALL	2530,SYNTAX-Z	In 'run-time'?
JP	NZ,2951,STK-F-ARG	If so, jump forward to search the DEF FN statement.

Otherwise (or if the variable was not found in the DEF FN statement) a search of variables area will be made, unless syntax is being checked.

28EF	V-RUN/SYN	LD	B,C	Copy the discriminator bytes to the B register.
		CALL	2530,SYNTAX-Z	Jump forward if in 'run-time'.
		JR	NZ,28FD,V-RUN	
		LD	A,C	Move the discriminator to A.
		AND	+E0	Drop the character code part.
		SET	7,A	Indicate syntax by setting bit 7.
		LD	C,A	Restore the discriminator.
		JR	2934,V-SYNTAX	Jump forward to continue.

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

28FD	V-RUN	LD	HL,(VARS)	Pick up the VARS pointer.
------	-------	----	-----------	---------------------------

Now enter a loop to consider the names of the existing variables.

2900	V-EACH	LD	A,(HL)	The 1st. letter of each existing variable.
		AND	+7F	Match on bits 0 to 6.
		JR	Z,2932,V-80-BYTE	Jump when the '80-byte' is reached.
		CP	C	The actual comparison.
		JR	NZ,292A,V-NEXT	Jump forward if the 1st characters do not match.
		RLA		Rotate A leftwards and then double it to test bits 5 & 6.
		ADD	A,A	Strings and array variables.
		JP	P,293F,V-FOUND-2	Simple numeric and FOR-NEXT variables.
		JR	C,293F,V-FOUND-2	

Long names are required to be matched fully.

		POP	DE	Take a copy of the pointer to the 2nd. character.
		PUSH	DE	Save the 1st letter pointer.
		PUSH	HL	Consider the next character.
2912	V-MATCHES	INC	HL	Fetch each character in turn.
2913	V-SPACES	LD	A,(DE)	Point to the next character.
		INC	DE	Is the character a 'space'?
		CP	+20	Ignore the spaces.
		JR	Z,2913,V-SPACES	Set bit 5 so as to match lower and upper case letters.
		OR	+20	Make the comparison.
		CP	(HL)	Back for another character if it does match.
		JR	Z,2912,V-MATCHES	Will it match with bit 7 set?
		OR	+80	Try it.
		CP	(HL)	Jump forward if the 'last characters' do not match.
		JR	NZ,2929,V-GET-PTR	Check that the end of the name has been reached before jumping forward.
		LD	A,(DE)	
		CALL	2C88,ALPHANUM	
		JR	NC,293E,V-FOUND-1	

In all cases where the names fail to match the HL register pair has to be made to point to the next variable in the variables area.

2929	V-GET-PTR	POP	HL	Fetch the pointer.
292A	V-NEXT	PUSH	BC	Save B & C briefly.
		CALL	19B8,NEXT-ONE	DE is made to point to the next variable.

EX	DE,HL	Switch the two pointers.
POP	BC	Get B & C back.
JR	2900,V-EACH	Go around the loop again.

Come here if no entry was found with the correct name.

2932	V-80-BYTE	SET	7,B	Signal 'variable not found'.
------	-----------	-----	-----	------------------------------

Come here if checking syntax.

2934	V-SYNTAX	POP	DE	Drop the pointer to the 2nd. character.
		RST	0018,GET-CHAR	Fetch the present character.
		CP	+28	Is it a '('?
		JR	Z,2943,V-PASS	Jump forward.
		SET	5,B	Indicate not dealing with an array and jump forward.
		JR	294B,V-END	

Come here when an entry with the correct name was found.

293E	V-FOUND-1	POP	DE	Drop the saved variable pointer.
293F	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-CHAR	Fetch the current character.

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

Note: This appears to have been done already at V-CHAR.

2943	V-PASS	CALL	2C88,ALPHANUM	Is it alphanumeric?
		JR	NC,294B,V-END	Jump when the end of the name has been found.
		RST	0020,NEXT-CHAR	Fetch the next character.
		JR	2943,V-PASS	Go back and test it.

The exit-parameters are now set.

294B	V-END	POP	HL	HL holds the pointer to the letter of a short name or the 'last' character of a long name.
		RL	B	Rotate the whole register.
		BIT	6,B	Specify the state of bit 6.
		RET		Finished.

The exit-parameters for the subroutine can be summarised as follows: The system variable CH-ADD points to the first location after the name of the variable as it occurs in the BASIC line.

When 'variable not found':

- The carry flag is set.
- The zero flag is set only when the search was for an array variable.
- The HL register pair points to the first letter of the name of the variable as it occurs in the BASIC line.

When 'variable found':

- The carry flag is reset.
- The zero flag is set for both simple string variables and all array variables.
- The HL register pair points to the letter of a 'short' name, or the last character of a 'long' name, of the existing entry that was found in the variables area.

In all cases bits 5 & 6 of the C register indicate the type of variable being handled. Bit 7 is the complement of the SYNTAX/RUN flag. But only when the subroutine is used in 'runtime' will bits 0 to 4 hold the code of the variable's letter.

In syntax time the return is always made with the carry flag reset. The zero flag is set for arrays and reset for all other variables, except that a simple string name incorrectly followed by a '\$' sets the zero flag and, in the case of SAVE "name" DATA a\$(), passes syntax as well.

THE 'STACK FUNCTION ARGUMENT' SUBROUTINE

This subroutine is called by LOOK-VARS when DEFADD-hi is non-zero, to make a search of the arguments area of a DEF FN statement, before searching in the variables area. If the variable is found in the DEF FN statement, then the parameters of a string variable are stacked and a signal is given that there is no need to call STK/VAR. But it is left to SCANNING to stack the value of a numerical variable at 26DA in the usual way.

2951	STK-F-ARG	LD	HL,(DEFADD)	Point to the 1st character in the arguments area and put it into A.
		LD	A,(HL)	Is it a ')?
		CP	+29	
		JP	Z,28EF,V-RUN/SYN	Jump to search the variables area.
295A	SFA-LOOP	LD	A,(HL)	Get the next argument in the loop.
		OR	+60	Set bits 5 & 6, assuming a simple numeric variable; copy it to B.
		LD	B,A	
		INC	HL	Point to the next code.
		LD	A,(HL)	Put it into the A register.
		CP	+0E	Is it the 'number marker' code 0E hex?
		JR	Z,296B,SFA-CP-VR	Jump if so: numeric variable.
		DEC	HL	Ensure that HL points to the character, not to a space or control code.
		CALL	28AB,FN-SKPOVR	HL now points to the 'number marker'.
		INC	HL	
296B	SFA-CP-VR	RES	5,B	Reset bit 5 of B: string variable.
		LD	A,B	Get the variable name into A.
		CP	C	Is it the one we are looking for?
		JR	Z,2981,SFA-MATCH	Jump if it matches.
		INC	HL	Now pass over the 5 bytes of the floating-point number or string parameters to get to the next argument.
		INC	HL	
		INC	HL	
		INC	HL	
		INC	HL	
		CALL	28AB,FN-SKPOVR	Pass on to the next character.
		CP	+29	Is it a ')?
		JP	Z,28EF,V-RUN/SYN	If so, jump to search the variables area.
		CALL	28AB,FN-SKPOVR	Point to the next argument.
		JR	295A,SFA-LOOP	Jump back to consider it.

A match has been found. The parameters of a string variable are stacked, avoiding the need to call the STK-VAR subroutine.

2981	SFA-MATCH	BIT	5,C	Test for a numeric variable.
		JR	NZ,2991,SFA-END	Jump if the variable is numeric; SCANNING will stack it.
		INC	HL	Point to the first of the 5 bytes to be stacked.
		LD	DE,(STKEND)	Point DE to STKEND.
		CALL	33C0,MOVE-FP	Stack the 5 bytes.
		EX	DE,HL	Point HL to the new position of STKEND, and reset the system variable.
		LD	(STKEND),HL	
2991	SFA-END	POP	DE	Discard the LOOK-VARS pointers (2nd & 1st character pointers).
		POP	DE	
		XOR	A	Return from the search with both the carry and zero flags reset - signalling that a call STK-VAR is not required.
		INC	A	
		RET		Finished.

THE 'STK-VAR' SUBROUTINE

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

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

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

2996	STK-VAR	XOR	A	Clear the array flag.
		LD	B,A	Clear the B register for later.
		BIT	7,C	Jump forward if syntax is
		JR	NZ,29E7,SV-COUNT	being checked.

Next, simple strings are separated from array variables.

		BIT	7,(HL)	Jump forward if dealing with
		JR	NZ,29AE,SV-ARRAYS	an array variable.

The parameters for a simple string are readily found.

29A1	SV-SIMPLE\$	INC	A	Signal 'a simple string'.
		INC	HL	Move along the entry.
		LD	C,(HL)	Pick up the low length counter.
		INC	HL	Advance the pointer.
		LD	B,(HL)	Pick up the high length
				pointer.
		INC	HL	Advance the pointer.
		EX	DE,HL	Transfer the pointer to the
				actual string.
		CALL	2AB2,STK-STORE	Pass these parameters to the
				calculator stack.
		RST	0018,GET-CHAR	Fetch the present character
				and jump forward to see if a
		JP	2A49,SV-SLICE?	'slice' is required.

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

29AE	SV-ARRAYS	INC	HL	Step past the length bytes.
		INC	HL	
		INC	HL	
		LD	B,(HL)	Collect the 'number of
				dimensions'.
		BIT	6,C	Jump forward if handling an
		JR	Z,29C0,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
		JR	Z,29A1,SV-SIMPLE\$	dimensions' and jump if the
				number is now zero.

Next a check is made to ensure that in the BASIC line the variable is followed by a subscript.

		EX	DE,HL	Save the pointer in DE.
		RST	0018,GET-CHAR	Get the present character.
		CP	+28	Is it a '('?
		JR	NZ,2A20,REPORT-3	Report the error if it is not so.
		EX	DE,HL	Restore the pointer.

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

29C0	SV-PTR	EX	DE,HL	Pass the pointer to DE.
		JR	29E7,SV-COUNT	Jump forward.

The following loop is used to find the parameters of a specified element within an array. The loop is entered at the mid-point - SV-COUNT -, where the element count is set to zero.

The loop is accessed 'B' times, this being, for a numeric array, equal to the number of dimensions that are being used, but for an array of strings 'B' is one less than the number of dimensions in use as the last subscript is used to specify a 'slice' of the string.

29C3	SV-COMMA	PUSH	HL	Save the counter.
		RST	0018,GET-CHAR	Get the present character.
		POP	HL	Restore the counter.
		CP	+2C	Is the present character a ','?
		JR	Z,29EA,SV-LOOP	Jump forward to consider another subscript.
		BIT	7,C	If a line is being executed then there is an error.
		JR	Z,2A20,REPORT-3	Jump forward if dealing with an array of strings.
		BIT	6,C	Is the present character a ')'? Report an error if not so.
		JR	NZ,29D8,SV-CLOSE	Advance CH-ADD.
		CP	+29	Return as the syntax is correct.
		JR	NZ,2A12,SV-RPT-C	
		RST	0020,NEXT-CHAR	
		RET		

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.

29D8	SV-CLOSE	CP	+29	Is the present character a ')'? Jump forward and check whether there is another subscript.
		JR	Z,2A48,SV-DIM	
29E0	SV-CH-ADD	CP	+CC	Is the present character a 'TO'? It must not be otherwise.
		JR	NZ,2A12,SV-RPT-C	Get the present character.
		RST	0018,GET-CHAR	Point to the preceding character and set CH-ADD.
		DEC	HL	Evaluate the 'slice'.
		LD	(CH-ADD),HL	
		JR	2A45,SV-SLICE	

Enter the loop here.

29E7	SV-COUNT	LD	HL,+0000	Set the counter to zero.
29EA	SV-LOOP	PUSH	HL	Save the counter briefly.
		RST	0020,NEXT-CHAR	Advance CH-ADD.
		POP	HL	Restore the counter.
		LD	A,C	Fetch the discriminator byte.
		CP	+C0	Jump unless checking the syntax for an array of strings.
		JR	NZ,29FB,SV-MULT	Get the present character.
		RST	0018,GET-CHAR	Is it a ')'? Jump forward as finished counting elements.
		CP	+29	Is to 'TO'? Jump back if dealing with a 'slice'.
		JR	Z,2A48,SV-DIM	Save the dimension-number counter and the discriminator byte.
		CP	+CC	Save the element-counter.
		JR	Z,29E0,SV-CH-ADD	Get a dimension-size into DE. The counter moves to HL and the variable pointer is stacked.
				The counter moves to DE and the dimension-size to HL.
29FB	SV-MULT	PUSH	BC	Evaluate the next subscript.
				Give an error if out of range.
		PUSH	HL	The result of the evaluation is decremented as the counter is to
		CALL	2AEE,DE,(DE+1)	
		EX	(SP),HL	
		EX	DE,HL	
		CALL	2ACC,INT-EXP1	
		JR	C,2A20,REPORT-3	
		DEC	BC	

CALL	2AF4,GET-HL*DE	count the elements occurring before the specified element. Multiply the counter by the dimension-size.
ADD	HL,BC	Add the result of 'INT-EXP1' to the present counter.
POP	DE	Fetch the variable pointer.
POP	BC	Fetch the dimension-number and the discriminator byte.
DJNZ	29C3,SV-COMMA	Keep going round the loop until 'B' equals zero.

The SYNTAX/RUN flag is checked before arrays of strings are separated from arrays of numbers.

2A12	SV-RPT-C	BIT JR PUSH BIT JR	7,C NZ,2A7A,SL-RPT-C HL 6,C NZ,2A2C,SV-ELEM\$	Report an error if checking syntax at this point. Save the counter. Jump forward if handling an array of strings.
------	----------	--------------------------------	---	---

When dealing with an array of numbers the present character must be a ')'.

LD	B,D	Transfer the variable pointer to the BC register pair.
LD	C,E	Fetch the present character.
RST	0018,GET-CHAR	Is it a ')'?
CP	+29	
JR	Z,2A22,SV-NUMBER	Jump past the error report unless it is needed.

Report 3 - Subscript out of range

2A20	REPORT-3	RST DEFB	0008,ERROR-1 +02	Call the error handling routine.
------	----------	-------------	---------------------	----------------------------------

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

2A22	SV-NUMBER	RST POP LD	0020,NEXT-CHAR HL DE,+0005	Advance CH-ADD. Fetch the counter. There are 5 bytes to each element in an array of numbers.
		CALL	2AF4,GET-HL*DE	Compute the total number of bytes before the required element.
		ADD	HL,BC	Make HL point to the location before the required element.
		RET		Return with this address.

When dealing with an array of strings the length of an element is given by the last 'dimension-size'. The appropriate parameters are calculated and then passed to the calculator stack.

2A2C	SV-ELEM\$	CALL EX	2AEE,DE,(DE+1) (SP),HL	Fetch the last dimension-size. The variable printer goes on the stack and the counter to HL.
		CALL	2AF4,GET-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 string.
		INC	HL	So point to the actual 'start'.
		LD	B,D	Transfer the last dimension-size to BC to form the 'length'.
		LD	C,E	
		EX	DE,HL	Move the 'start' to DE.
		CALL	2AB1,STK-ST-0	Pass these parameters to the calculator stack. Note: The first parameter is zero indicating a string from an 'array of strings'

and hence the existing entry is not to be reclaimed.

There are three possible forms of the last subscript. The first is illustrated by - A\$(2,4 TO 8) -, the second by - A\$(2)(4 TO 8) - and the third by - A\$(2) - which is the default form and indicates that the whole string is required.

		RST	0018,GET-CHAR	Get the present character.
		CP	+29	Is it a ')'?
		JR	Z,2A48,SV-DIM	Jump if it is so.
		CP	+2C	Is it a ','?
		JR	NZ,2A20,REPORT-3	Report the error if not so.
2A45	SV-SLICE	CALL	2A52,SLICING	Use SLICING to modify the set of parameters.
2A48	SV-DIM	RST	0020,NEXT-CHAR	Fetch the next character.
2A49	SV-SLICE?	CP	+28	Is It a '('?
		JR	Z,2A45,SV-SLICE	Jump back if there is a 'slice' to be considered.

When finished considering the last subscript a return can be made.

RES	6,(FLAGS)	Signal - string result.
RET		Return with the parameters of the required string forming a 'last value' on the calculator stack.

THE 'SLICING' SUBROUTINE

The present string can be sliced using this subroutine. The subroutine is entered with the parameters of the string being present on the top of the calculator stack and in the registers A, B, C, D & E. Initially the SYNTAX/RUN flag is tested and the parameters of the string are fetched only if a line is being executed.

2A52	SLICING	CALL	2530,SYNTAX-Z	Check the flag.
		CALL	NZ,2BF1,STK-FETCH	Take the parameters off the stack in 'run-time'.

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

RST	0020,NEXT-CHAR	Get the next character.
CP	+29	Is it a ')'?
JR	Z,2AAD,SL-STORE	Jump forward if it is so.

Before proceeding the registers are manipulated as follows:

PUSH	DE	The 'start' goes on the machine stack.
XOR	A	The A register is cleared and saved.
PUSH	AF	The 'length' is saved briefly.
PUSH	BC	Presume that the 'slice' is to begin with the first character.
LD	DE,+0001	Get the first character.
RST	0018,GET-CHAR	Pass the 'length' to HL.
POP	HL	

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

CP	+CC	Is the present character a 'TO'?
JR	Z,2A81,SL-SECOND	The first parameter, by default, will be '1' if the jump is taken.
POP	AF	At this stage A is zero.
CALL	2ACD,INT-EXP2	BC is made to hold the first parameter. A will hold +FF if there has been an 'out of range' error.
PUSH	AF	Save the value anyway.
LD	D,B	Transfer the first parameter to DE.
LD	E,C	
PUSH	HL	Save the 'length' briefly.

		RST	0018,GET-CHAR	Get the present character.
		POP	HL	Restore the 'length'.
		CP	+CC	Is the present character a 'TO'?
		JR	Z,2A81,SL-SECOND	Jump forward to consider the second parameter if it is so;
2A7A	SL-RPT-C	JP	NZ,1C8A,REPORT-C	CP +29 otherwise show that there is a closing bracket.

At this point a 'slice' of a single character has been identified. e.g. - A\$(4).

LD	H,D	The last character of the 'slice'
LD	L,E	is also the first character.
JR	2A94,SL-DEFINE	Jump forward.

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

2A81	SL-SECOND	PUSH	HL	Save the 'length' briefly.
		RST	0020,NEXT-CHAR	Get the next character.
		POP	HL	Restore the 'length'.
		CP	+29	Is the present character a 'J'?
		JR	Z,2A94,SL-DEFINE	Jump if there is not a second parameter.
		POP	AF	If the first parameter was in range A will hold zero; otherwise +FF.
		CALL	2ACD,INT-EXP2	Make BC hold the second parameter.
		PUSH	AF	Save the 'error register'.
		RST	0018,GET-CHAR	Get the present character.
		LD	H,B	Pass the result obtained from INT-EXP2 to the HL register pair.
		LD	L,C	
		CP	+29	Check that there is a closing bracket now.
		JR	NZ,2A7A,SL-RPT-C	

The 'new' parameters are now defined.

2A94	SL-DEFINE	POP	AF	Fetch the 'error register'.
		EX	(SP),HL	The second parameter goes on the stack and the 'start' goes to HL.
		ADD	HL,DE	The first parameter is added to the 'start'.
		DEC	HL	Go back a location to get it correct.
		EX	(SP),HL	The 'new start' goes on the stack and the second parameter goes to HL.
		AND	A	Subtract the first parameters from the second to find the length of the 'slice'.
		SBC	HL,DE	Initialise the 'new length'.
		LD	BC,+0000	A negative 'slice' is a 'null string' rather than an error condition. (See manual.)
		JR	C,2AA8,SL-OVER	Allow for the inclusive byte.
		INC	HL	Only now test the 'error register'.
		AND	A	Jump if either parameter was out of range for the string.
		JP	M,2A20,REPORT-3	Transfer the 'new length' to BC.
		LD	B,H	Get the 'new start'.
		LD	C,L	Ensure that a string is still indicated.
2AA8	SL-OVER	POP	DE	
		RES	6,(FLAGS)	

2AAD	SL-STORE	CALL RET	2530,SYNTAX-Z Z	Return at this point if checking syntax; otherwise continue into the STK-STORE subroutine.
------	----------	-------------	--------------------	--

THE 'STK-STORE' SUBROUTINE

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

The subroutine is normally used to transfer the parameters of strings but it is also used by STACK-BC and LOG (2^A) to transfer 'small integers' to the stack.

Note that when storing the parameters of a string the first value stored (coming from the A register) will be a zero if the string comes from an array of strings or is a 'slice' of a string. The value will be '1' for a complete simple string. This 'flag' is used in the 'LET' command routine when the '1' signals that the old copy of the string is to be 'reclaimed'.

2AB1	STK-ST-0	XOR	A	Signal - a string from an array of strings or a 'sliced' string.
2AB2	STK-STO-\$	RES	6,(FLAGS)	Ensure the flag Indicates a string result.
2AB6	STK-STORE	PUSH CALL	BC 33A9,TEST-5-SP	Save B & C briefly. Is there room for 5 bytes? Do not return here unless there is room available.
		POP LD	BC HL,(STKEND)	Restore B & C. Fetch the address of the first location above the present stack.
		LD INC LD INC LD INC LD INC LD INC	(HL),A HL (HL),E HL (HL),D HL (HL),C HL (HL),B HL	Transfer the first byte. Step on. Transfer the second and third bytes; for a string these will be the 'start'. Step on. Transfer the fourth and fifth bytes; for a string these will be the 'length'.
		LD RET	(STKEND),HL	Step on so as to point to the location above the stack. Save this address In STKEND and return.

THE 'INT-EXP' SUBROUTINE

This subroutine returns the result of evaluating the 'next expression' as an integer value held in the BC register pair. The subroutine also tests this result against a limit-value supplied in the HL register pair. The carry flag becomes set if there is an 'out of range' error. The A register is used as an 'error register' and holds +00 if there is no 'previous error' and +FF if there has been one.

2ACC	INT-EXP1	XOR	A	Clear the 'error register'.
2ACD	INT-EXP2	PUSH PUSH PUSH	DE HL AF	Save both the DE & HL register pairs throughout. Save the 'error register' briefly.
		CALL	1C82,EXPT-1NUM	The 'next expression' is evaluated to give a 'last value' on the calculator stack.
		POP CALL	AF 2530,SYNTAX-Z	Restore the 'error register'. Jump forward if checking

JR	Z,2AEB,I-RESTORE	syntax.
PUSH	AF	Save the error register again.
CALL	1E99,FIND-INT2	The 'last value' is compressed into BC.
POP	DE	Error register to D.
LD	A,B	A 'next expression' that gives zero is always in error so jump forward if it is so.
OR	C	Take a copy of the limit-value. This will be a 'dimension-size' a 'DIM-limit' or a 'string length'.
SCF		Now compare the result of evaluating the expression against the limit.
JR	Z,2AE8,I-CARRY	
POP	HL	
PUSH	HL	
AND	A	
SBC	HL,BC	

The state of the carry flag and the value held in the D register are now manipulated so as to give the appropriate value for the 'error register'.

2AE8	I-CARRY	LD	A,D	Fetch the 'old error value'
		SBC	A,+00	Form the 'new error value';
				+00 if no error at anytime/
				+FF or less if an 'out of range' error on this pass or on previous ones.

Restore the registers before returning.

2AEB	I-RESTORE	POP	HL	Restore HL & DE.
		POP	DE	
		RET		Return; 'error register' is the A register.

THE 'DE,(DE+1)' SUBROUTINE

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

2AEE	DE,(DE+1)	EX	DE,HL	Use HL for the construction.
		INC	HL	Point to 'DE+1'.
		LD	E,(HL)	In effect - LD E,(DE+1).
		INC	HL	Point to 'DE+2'.
		LD	D,(HL)	In effect - LD D,(DE+2).
		RET		Finished.

THE 'GET-HL*DE' SUBROUTINE

Unless syntax is being checked this subroutine calls 'HL=HL*DE' which performs the implied construction.

Overflow of the 16 bits available in the HL register pair gives the report 'out of memory'. This is not exactly the true situation but it implies that the memory is not large enough for the task envisaged by the programmer.

2AF4	GET-HL*DE	CALL	2530,SYNTAX-Z	Return directly if syntax is being checked.
		RET	Z	
		CALL	30A9,HL=HL*DE	Perform the multiplication.
		JP	C,1F15,REPORT-4	Report 'Out of memory'.
		RET		Finished.

THE 'LET' COMMAND ROUTINE

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

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

However if the destination variable 'exists already' then bit 1 of FLAGX will be reset and DEST will point for a numeric variable to the location before the five bytes of the

'old number'; and for a string variable to the first location of the 'old string'. The use of DEST in this manner applies to simple variables and to elements of arrays.

Bit 0 of FLAGX is set if the destination variable is a 'complete' simple string variable. (Signalling - delete the old copy.)

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

2AFF	LET	LD	HL,(DEST)	Fetch the present address in DEST.
		BIT	1,(FLAGX)	Jump if handling a variable
		JR	Z,2B66,L-EXISTS	that 'exists already'.

A 'newly declared variable' is being used. So first the length of its name is found.

LD	BC,+0005	Presume dealing with a numeric variable - 5 bytes.
----	----------	--

Enter a loop to deal with the characters of a long name. Any spaces or colour codes in the name are ignored.

2B0B	L-EACH-CH	INC	BC	Add '1' to the counter for each character of a name.
2B0C	L-NO-SP	INC	HL	Move along the variable's name.
		LD	A,(HL)	Fetch the 'present code'.
		CP	+20	Jump back if it is a 'space';
		JR	Z,2B0C,L-NO-SP	thereby ignoring spaces.
		JR	NC,2B1F,L-TEST-CH	Jump forward if the code is +21 to +FF.
		CP	+10	Accept, as a final code, those
		JR	C,2B29,L-SPACES	in the range +00 to +0F.
		CP	+16	Also accept the range
		JR	NC,2B29,L-SPACES	+16 to +1F.
		INC	HL	Step past the control code
		JR	2B0C,L-NO-SP	after any of INK to OVER.
				Jump back as these control codes are treated as spaces.

Separate 'numeric' and 'string' names.

2B1F	L-TEST -CH	CALL	2C88,ALPHANUM	Is the code alphanumeric?
		JR	C,2B0B,L-EACH-CH	If It is so then accept it as a character of a 'long' name.
		CP	+24	Is the present code a 'S'?
		JP	Z,2BC0,L-NEWS	Jump forward as handling a 'newly declared' simple string.

The 'newly declared numeric variable' presently being handled will require 'BC' spaces in the variables area for its name and its value. The room is made available and the name of the variable is copied over with the characters being 'marked' as required.

2B29	L-SPACES	LD	A,C	Copy the 'length' to A.
		LD	HL,(E-LINE)	Make HL point to the
		DEC	HL	'80-byte' at the end of the variables area.
		CALL	1655,MAKE-ROOM	Now open up the variables area.
				Note: In effect 'BC' spaces are made before the displaced '80-byte'.
		INC	HL	Point to the first 'new' byte.
		INC	HL	Make DE point to the second
		EX	DE,HL	'new' byte.
		PUSH	DE	Save this pointer.
		LD	HL,(DEST)	Fetch the pointer to the start of the name.
		DEC	DE	Make DE point to the first 'new' byte.
		SUB	+06	Make B hold the 'number of extra letters' that are found
		LD	B,A	in a 'long name'.

JR	Z,2B4F,L-SINGLE	Jump forward if dealing with a variable with a 'short name'.
----	-----------------	--

The 'extra' codes of a long name are passed to the variables area.

2B3E	L-CHAR	INC LD CP JR OR	HL A,(HL) +21 C,2B3E,L-CHAR +20	Point to each 'extra' code. Fetch the code. Accept codes from +21 to +FF; ignore codes +00 to +20. Set bit 5, as for lower case letters.
		INC LD	DE (DE),A	Transfer the codes in turn to the 2nd 'new' byte onwards.
		DJNZ	2B3E,L-CHAR	Go round the loop for all the 'extra' codes.

The last code of a 'long' name has to be ORed with +80.

OR	+80	Mark the code as required
LD	(DE),A	and overwrite the last code.

The first letter of the name of the variable being handled is now considered.

		LD	A,+C0	Prepare the mark the letter of a 'long' name.
2B4F	L-SINGLE	LD XOR	HL,(DEST) (HL)	Fetch the pointer to the letter. A holds +00 for a 'short' name and +C0 for a 'long' name.
		OR	+20	Set bit 5, as for lower case letters.
		POP	HL	Drop the pointer now.

The subroutine L-FIRST is now called to enter the 'letter' into its appropriate location.

CALL	2BEA,L-FIRST	Enter the letter and return with HL pointing to 'new 80-byte'.
------	--------------	--

The 'last value' can now be transferred to the variables area. Note that at this point HL always points to the location after the five locations allotted to the number.

A 'RST 0028' instruction is used to call the CALCULATOR and the 'last value' is deleted. However this value is not overwritten.

2B59	L-NUMERIC	PUSH RST DEFB DEFB POP LD	HL 0028,FP-CALC +02,delete +38,end-calc HL BC,+0005	Save the 'destination' pointer. Use the calculator. This moves STKEND back five bytes. Restore the pointer. Give the number a 'length' of five bytes.
		AND SBC JR	A HL,BC 2BA6,L-ENTER	Make HL point to the first of the five locations and jump forward to make the actual transfer.

Come here if considering a variable that 'exists already'. First bit 6 of FLAGS is tested so as to separate numeric variables from string or array of string variables.

2B66	L-EXISTS	BIT JR	6,(FLAGS) Z,2B72,L-DELETES	Jump forward if handling any kind of string variable.
------	----------	-----------	-------------------------------	---

For numeric variables the 'new' number overwrites the 'old' number. So first HL has to be made to point to the location after the five bytes of the existing entry. At present HL points to the location before the five bytes.

LD	DE,+0006	The five bytes of a number '+1'.
ADD	HL,DE	HL now points 'after'.
JR	2B59,L-NUMERIC	Jump back to make the actual transfer.

The parameters of the string variable are fetched and complete simple strings separated from 'sliced' strings and array strings.

2B72	L-DELETE\$	LD	HL,(DEST)	Fetch the 'start'. Note: This line is redundant.
		LD	BC,(STRLEN)	Fetch the 'length'.
		BIT	0,(FLAGX)	Jump if dealing with a complete
		JR	NZ,2BAF,L-ADD\$	simple string; the old string will need to be 'deleted' in this case only.

When dealing with a 'slice' of an existing simple string, a 'slice' of a string from an array of strings or a complete string from an array of strings there are two distinct stages involved. The first is to build up the 'new' string in the work space, lengthening or shortening it as required. The second stage is then to copy the 'new' string to its allotted room in the variables area. However do nothing if the string has no 'length'.

LD	A,B	Return if the string is
OR	C	a null string.
RET	Z	

Then make the required number of spaces available in the work space.

PUSH	HL	Save the 'start' (DEST).
RST	0030,BC-SPACES	Make the necessary amount of room in the work space.
PUSH	DE	Save the pointer to the first location.
PUSH	BC	Save the 'length' for use later on.
LD	D,H	Make DE point to the last location.
LD	E,L	Make HL point 'one past' the new locations.
INC	HL	Enter a 'space' character.
LD	(HL),+20	Copy this character into all the new locations. Finish with HL pointing to the first new location.
LDDR		

The parameters of the string being handled are now fetched from the calculator stack.

PUSH	HL	Save the pointer briefly.
CALL	2BF1,STK-FETCH	Fetch the 'new' parameters.
POP	HL	Restore the pointer.

Note: At this point the required amount of room has been made available in the work space for the 'variable in assignment'. e.g. For statement - LET A\$(4 to 8)="abcdefg" - five locations have been made.

The parameters fetched above as a 'last value' represent the string that is to be copied into the new locations with Procrustean lengthening or shortening as required.

The length of the 'new' string is compared to the length of the room made available for it.

	EX	(SP),HL	'Length' of new area to HL.
	AND	A	'Pointer' to new area to stack.
	SBC	HL,BC	Compare the two 'lengths'
	ADD	HL,BC	and jump forward if the 'new' string will fit into the room.
	JR	NC,2B9B,L-LENGTH	i.e. No shortening required.
	LD	B,H	However modify the 'new' length if it is too long.
	LD	C,L	
2B9B	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 work space. Procrustean lengthening is achieved automatically if the 'new' string is shorter than the room available for it.

EX	DE,HL	'Start' of new string to HL.
LD	A,B	'Pointer' to new area to DE.
OR	C	Jump forward if the
JR	Z,2BA3,L-IN-W/S	'new' string is a 'null'
LDIR		string.
		Otherwise move the 'new'
		string to the work space.

The values that have been saved on the machine stack are restored.

2BA3	L-IN-W/S	POP	BC	'Length' of new area.
		POP	DE	'Pointer' to new area.
		POP	HL	The start - the pointer
				to the 'variable in assignment'
				which was originally in DEST.
				L-ENTER is now used to pass
				the 'new' string to the variables
				area.

THE 'L-ENTER' SUBROUTINE

This short subroutine is used to pass either a numeric value, from the calculator stack, or a string, from the work space, to its appropriate position in the variables area.

The subroutine is therefore used for all except 'newly declared' simple strings and 'complete & existing' simple strings.

2BA6	L-ENTER	EX	DE,HL	Change the pointers over.
		LD	A,B	Check once again that the
		OR	C	length is not zero.
		RET	Z	
		PUSH	DE	Save the destination pointer.
		LDIR		Move the numeric value or the
				string
		POP	HL	Return with the HL register
		RET		pair pointing to the first byte
				of the numeric value or the
				string.

THE LET SUBROUTINE CONTINUES HERE

When handling a 'complete & existing' simple string the new string is entered as if it were a 'newly declared' simple string before the existing version is 'reclaimed'.

2BAF	L-ADD\$	DEC	HL	Make HL point to the letter
		DEC	HL	of the variable's name.
		DEC	HL	i.e. DEST - 3.
		LD	A,(HL)	Pick up the letter.
		PUSH	HL	Save the pointer to the 'existing
				version'.
		PUSH	BC	Save the 'length' of the
				'existing string'.
		CALL	2BC6,L-STRING	Use L-STRING to add the new
				string to the variables area.
		POP	BC	Restore the 'length'.
		POP	HL	Restore the pointer.
		INC	BC	Allow one byte for the letter
		INC	BC	and two bytes for the length.
		INC	BC	
		JP	19E8,RECLAIM-2	Exit by jumping to RECLAIM-2
				which will reclaim the whole
				of the existing version.

'Newly declared' simple strings are handled as follows:

2BC0	L-NEW\$	LD	A,+DF	Prepare for the marking of
				the variable's letter.

LD	HL,(DEST)	Fetch the pointer to the letter.
AND	(HL)	Mark the letter as required. L-STRING is now used to add the new string to the variables area.

THE 'L-STRING' SUBROUTINE

The parameters of the 'new' string are fetched, sufficient room is made available for it and the string is then transferred.

2BC6	L-STRING	PUSH CALL	AF 2BF1,STK-FETCH	Save the variable's letter Fetch the 'start' and the 'length' of the 'new' string.
		EX ADD	DE,HL HL,BC	Move the 'start' to HL. Make HL point 'one-past' the string.
		PUSH DEC	BC HL	Save the 'length'. Make HL point to the end of the string.
		LD INC INC INC LD DEC	(DEST),HL BC BC BC HL,(E-LINE) HL	Save the pointer briefly. Allow one byte for the letter and two bytes for the length.
		CALL	1655,MAKE-ROOM	Make HL point to the '80-byte' at the end of the variables area. Now open up the variables area. Note: In effect 'BC' spaces are made before the displaced '80-byte'.
		LD	HL,(DEST)	Restore the pointer to the end of the 'new' string.
		POP PUSH INC	BC BC BC	Make a copy of the length of the 'new' string. Add one to the length in case the 'new' string is a 'null' string.
		LDDR		Now copy the 'new' string + one byte.
		EX INC POP LD DEC LD POP	DE,HL HL BC (HL),B HL (HL),C AF	Make HL point to the byte that is to hold the high-length. Fetch the 'length'. Enter the high-length. Back one. Enter the low-length. Fetch the variable's letter.

THE 'L-FIRST' SUBROUTINE

This subroutine is entered with the letter of the variable, suitably marked, in the A register. The letter overwrites the 'old 80-byte' in the variables area. The subroutine returns with the HL register pair pointing to the 'new 80-byte'.

2BEA	L-FIRST	DEC	HL	Make HL point to the 'old 80-byte'.
		LD	(HL),A	It is overwritten with the letter of the variable.
		LD	HL,(E-LINE)	Make HL point to the 'new 80-byte'.
		DEC RET	HL	Finished with all the 'newly declared variables'.

THE 'STK-FETCH' SUBROUTINE

This important subroutine collects the 'last value' from the calculator stack. The five bytes can be either a floating-point number, in 'short' or 'long' form, or set of parameters that define a string.

2BF1	STK-FETCH	LD	HL,(STKEND)	Get STKEND.
		DEC	HL	Back one;
		LD	B,(HL)	The fifth value.
		DEC	HL	Back one.
		LD	C,(HL)	The fourth one.
		DEC	HL	Back one.
		LD	D,(HL)	The third value.
		DEC	HL	Back one.
		LD	E,(HL)	The second value.
		DEC	HL	Back one.
		LD	A,(HL)	The first value.
		LD	(STKEND),HL	Reset STKEND to its new position
		RET		Finished.

THE 'DIM' COMMAND ROUTINE

This routine establishes new arrays in the variables area. The routine starts by searching the existing variables area to determine whether there is an existing array with the same name. If such an array is found then it is 'reclaimed' before the new array is established.

A new array will have all its elements set to zero, if it is a numeric array, or to 'spaces', if it is an array of strings.

2C02	DIM	CALL	28B2,LOOK-VARS	Search the variables area.
2C05	D-RPORT-C	JP	NZ,1C8A,REPORT-C	Give report C as there has been an error.
		CALL	2530,SYNTAX-Z	Jump forward if in 'run time'.
		JR	NZ,2C15,D-RUN	Test the syntax for string arrays as if they were numeric.
		RES	6,C	Check the syntax of the parenthesised expression.
		CALL	2996,STK-VAR	Move on to consider the next statement as the syntax was satisfactory.
		CALL	1BEE,CHECK-END	

An 'existing array' is reclaimed.

2C15	D-RUN	JR	C,2C1F,D-LETTER	Jump forward if there is no 'existing array'.
		PUSH	BC	Save the discriminator byte.
		CALL	19B8,NEXT-ONE	Find the start of the next variable
		CALL	19E8,RECLAIM-2	Reclaim the 'existing array'.
		POP	BC	Restore the discriminator byte.

The initial parameters of the new array are found.

2C1F	D-LETTER	SET	7,C	Set bit 7 in the discriminator byte.
		LD	B,+00	Make the dimension counter zero.
		PUSH	BC	Save the counter and the discriminator byte.
		LD	HL,+0001	The HL register pair is to hold the size of the elements in the array, '1' for a string array/ '5' for a numeric array.
		BIT	6,C	Element size DE.
		JR	NZ,2C2D,D-SIZE	
		LD	L,+05	
2C2D	D-SIZE	EX	DE,HL	

The following loop is accessed for each dimension that is specified in the parenthesised expression of the DIM statement. The total number of bytes required for the elements of the array is built up in the DE register pair.

2C2E	D-NO-LOOP	RST	0020,NEXT-CHAR	Advance CH-ADD on each pass..
		LD	H,+FF	Set a 'limit value'.
		CALL	2ACC,INT-EXP1	Evaluate a parameter.
		JP	C,2A20,REPORT-3	Give an error if 'out of range'.
		POP	HL	Fetch the dimension-counter and the discriminator byte.
		PUSH	BC	Save the parameter on each pass through the loop.
		INC	H	Increase the dimension counter on each pass also.
		PUSH	HL	Restack the dimension-counter and the discriminator byte.
		LD	H,B	The parameter is moved to the HL register pair.
		LD	L,C	The byte total is built up in HL and the transferred to DE.
		CALL	2AF4,GET-HL*DE	
		EX	DE,HL	
		RST	0018,GET-CHAR	Get the present character and go around the loop again if there is another dimension.
		CP	+2C	
		JR	Z,2C2E,D-NO-LOOP	

Note: At this point the DE register pair indicates the number of bytes required for the elements of the new array and the size of each dimension is stacked, on the machine stack.

Now check that there is indeed a closing bracket to the parenthesised expression.

CP	+29	Is it a ')?
JR	NZ,2C05,D-REPORT-C	Jump back if not so.
RST	0020,NEXT-CHAR	Advance CH-ADD past it.

Allowance is now made for the dimension-sizes.

POP	BC	Fetch the dimension-counter and the discriminator byte.
LD	A,C	Pass the discriminator byte to the A register for later.
LD	L,B	Move the counter to L.
LD	H,+00	Clear the H register.
INC	HL	Increase the dimension-counter by two and double the result and form the correct overall length for the variable by adding the element byte total.
INC	HL	
ADD	HL,HL	
ADD	HL,DE	
JP	C,1F15,REPORT-4	Give the report 'Out of memory' if required.
PUSH	DE	Save the element byte total.
PUSH	BC	Save the dimension counter and the discriminator byte.
PUSH	HL	Save the overall length also.
LD	B,H	Move the overall length to BC.
LD	C,L	

The required amount of room is made available for the new array at the end of the variables area.

LD	HL,(E-LINE)	Make the HL register pair point to the '80-byte'.
DEC	HL	The room is made available.
CALL	1655,MAKE-ROOM	HL is made to point to the first new location.
INC	HL	

The parameters are now entered.

LD	(HL),A	The letter, suitably marked, is entered first.
POP	BC	The overall length is fetched and decreased by '3'.
DEC	BC	
DEC	BC	
DEC	BC	
INC	HL	Advance HL.
LD	(HL),C	Enter the low length.
INC	HL	Advance HL.
LD	(HL),B	Enter the high length.
POP	BC	Fetch the dimension counter.
LD	A,B	Move it to the A register.
INC	HL	Advance HL.
LD	(HL),A	Enter the dimension count.

The elements of the new array are now 'cleared'.

	LD	H,D	HL is made to point to the last location of the array
	LD	L,E	and DE to the location before that one.
	DEC	DE	
	LD	(HL),+00	Enter a zero into the last location but overwrite it
	BIT	6,C	with 'space' if dealing with an array of strings.
	JR	Z,2C7C,DIM-CLEAR	Fetch the element byte total.
2C7C	DIM-CLEAR	LD	(HL),+20
	POP	BC	Clear the array + one extra location.
	LDDR		

The 'dimension-sizes' are now entered.

2C7F	DIM-SIZES	POP	BC	Get a dimension-size.
		LD	(HL),B	Enter the high byte.
		DEC	HL	Back one.
		LD	(HL),C	Enter the low byte.
		DEC	HL	Back one.
		DEC	A	Decrease the dimension counter.
		JR	NZ,2C7F,DIM-SIZES	Repeat the operation until all the dimensions have been considered; then return.
		RET		

THE 'ALPHANUM' SUBROUTINE

This subroutine returns with the carry flag set if the present value of the A register denotes a valid digit or letter.

2C88	ALPHANUM	CALL	2D1B,NUMERIC	Test for a digit; carry will be reset for a digit.
		CCF		Complement the carry flag.
		RET	C	Return if a digit; otherwise continue on into 'ALPHA'.

THE 'ALPHA' SUBROUTINE

This subroutine returns with the carry flag set if the present value of the A register denotes a valid letter of the alphabet.

2C8D	ALPHA	CP	+41	Test against 41 hex, the code for 'A'
		CCF		Complement the carry flag.
		RET	NC	Return if not a valid character code.
		CP	+5B	Test against 5B hex, 1 more than code for 'Z'.
		RET	C	Return if an upper case letter.
		CP	+61	Test against 61 hex, the code for 'a'.

CCF			Complement the carry flag.
RET	NC		Return if not a valid character code.
CP	+7B		Test against 7B hex, 1 more than the code for 'z'.
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. But first it deals with the alternative notation BIN, which introduces a sequence of 0's and 1's giving the binary representation of the required number.

2C9B	DEC-TO-FP	CP	+C4	Is the character a 'BIN'?
		JR	NZ,2CB8,NOT-BIN	Jump if it is not 'BIN'.
		LD	DE,+0000	Initialise result to zero in DE.
2CA2	BIN-DIGIT	RST	0020,NEXT-CHAR	Get the next character.
		SUB	+31	Subtract the character code for '1'.
		ADC	A,+00	0 now gives 0 with carry set; 1 gives 0 with carry reset.
		JR	NZ,2CB3,BIN-END	Any other character causes a jump to BIN-END and will be checked for syntax during or after scanning.
		EX	DE,HL	Result so far to HL now.
		CCF		Complement the carry flag.
		ADC	HL,HL	Shift the result left, with the carry going to bit 0.
		JP	C,31AD,REPORT-6	Report overflow if more than 65535.
		EX	DE,HL	Return the result so far to DE.
		JR	2CA2,BIN-DIGIT	Jump back for next 0 or 1.
2CB3	BIN-END	LD	B,D	Copy result to BC for stacking.
		LD	C,E	
		JP	2D2B,STACK-BC	Jump forward to stack the result.

For other numbers, first any integer part is converted; if the next character is a decimal, then the decimal fraction is considered.

2CB8	NOT-BIN	CP	+2E	Is the first character a '.'?
		JR	Z,2CCB,DECIMAL	If so, jump forward.
		CALL	2D3B,INT-TO-FP	Otherwise, form a 'last value' of the integer.
		CP	+2E	Is the next character a '.'?
		JR	NZ,2CEB,E-FORMAT	Jump forward to see if it is an 'E'.
		RST	0020,NEXT-CHAR	Get the next character.
		CALL	2D1B,NUMERIC	Is it a digit?
		JR	C,2CEB,E-FORMAT	Jump if not (e.g. 1.E4 is allowed).
		JR	2CD5,DEC-STO-1	Jump forward to deal with the digits after the decimal point.
2CCB	DECIMAL	RST	0020,NEXT-CHAR	If the number started with a decimal, see if the next character is a digit.
		CALL	2D1B,NUMERIC	
2CCF	DEC-RPT-C	JP	C,1C8A,REPORT-C	Report the error if it is not.
		RST	0028,FP-CALC	Use the calculator to stack zero as the integer part of such numbers.
		DEFB	+A0,stk-zero	
		DEFB	+38,end-calc	
2CD5	DEC-STO-1	RST	0028,FP-CALC	Use the calculator again.
		DEFB	+A1,stk-one	Find the floating-point form of the decimal number '1', and
		DEFB	+C0,st-mem-0	

2CDA	NXT-DGT-1	DEFB	+02,delete	save it in the memory area.
		DEFB	+38,end-calc	
		RST	0018,GET-CHAR	Get the present character.
		CALL	2D22,STK-DIGIT	If it is a digit then stack it.
		JR	C,2CEB,E-FORMAT	If not jump forward.
		RST	0028,FP-CALC	Now use the calculator.
		DEFB	+E0,get-mem-0	For each passage of the loop,
		DEFB	+A4,stk-ten	the number saved in the memory
		DEFB	+05,division	area is fetched, divided by 10
		DEFB	+C0,st-mem-0	and restored: i.e. going from .1
		DEFN	+04,multiply	to .01 to .001 etc.
		DEFB	+0F,addition	The present digit is multiplied
		DEFB	+38,end-calc	by the 'saved number' and
		RST	0020,NEXT-CHAR	added to the 'last value'.
		JR	2CDA,NXT-DGT-1	Get the next character.
				Jump back (one more byte than
				needed) to consider it.

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

2CEB	E-FORMAT	CP	+45	Is the present character an 'E'?
		JR	Z,2CF2,SIGN-FLAG	Jump forward if it is.
		CP	+65	Is it an 'e'?
2CF2	SIGN-FLAG	RET	NZ	Finished unless it is so.
		LD	B,+FF	Use B as a sign flag, FF for '+'.
		RST	0020,NEXT-CHAR	Get the next character.
		CP	+2B	Is it a '+'?
		JR	Z,2CFE,SIGN-DONE	Jump forward.
2CFE	SIGN-DONE	CP	+2D	Is it a '-'?
		JR	NZ,2CFF,ST-E-PART	Jump if neither '+' not '-'.
		INC	B	Change the sign of the flag.
		RST	0020,NEXT-CHAR	Point to the first digit.
		CALL	2D1B,NUMERIC	Is it indeed a digit?
2CFF	ST-E-PART	JR	C,2CCF,DEC-RPT-C	Report the error if not.
		PUSH	BC	Save the flag in B briefly.
		CALL	2D3B,INT-TO-FP	Stack ABS m, where m is the
				exponent.
		CALL	2DD5,FP-TO-A	Transfer ABS m to A.
		POP	BC	Restore the sign flag to B.
		JP	C,31AD,REPORT-6	Report the overflow now if
		AND	A	ABS m is greater than 255 or
		JP	M,31AD,REPORT-6	indeed greater than 127 (other
				values greater than about 39 will
				be detected later).
		INC	B	Test the sign flag in B; '+'
				(i.e. +FF) will now set the zero
				flag.
		JR	Z,2D18,E-FP-JUMP	Jump if sign of m is '+'.
2D18	E-FP-JUMP	NEG		Negate m if sign is '-'.
		JP	2D4F,E-TOO-FP	Jump to assign to the 'last value'
				the result of $x \cdot 10^m$.

THE 'NUMERIC' SUBROUTINE

This subroutine returns with the carry flag reset if the present value of the A register denotes a valid digit.

2D1B	NUMERIC	CP	+30	Test against 30 hex, the code
				for '0'.
		RET	C	Return if not a valid character
				code.
		CP	+3A	Test against the upper limit.
		CCF		Complement the carry flag.
		RET		Finished.

THE 'STK DIGIT' SUBROUTINE

This subroutine simply returns if the current value held in the A register does not represent a digit but if it does then the floating-point form for the digit becomes the 'last value' on the calculator stack.

2D22	STK-DIGIT	CALL	2D1B,NUMERIC	Is the character a digit?
		RET	C	Return if not in range.
		SUB	+30	Replace the code by the actual digit.

THE 'STACK-A' SUBROUTINE

This subroutine gives the floating-point form for the absolute binary value currently held in the A register.

2D28	STACK-A	LD	C,A	Transfer the value to the C register.
		LD	B,+00	Clear the B register

THE 'STACK-BC' SUBROUTINE

This subroutine gives the floating-point form for the absolute binary value currently held in the BC register pair.

The form used in this and hence in the two previous subroutines as well is the one reserved in the Spectrum for small integers n, where -65535 <= n <= 65535. The first and fifth bytes are zero; the third and fourth bytes are the less significant and more significant bytes of the 16 bit integer n in two's complement form (if n is negative, these two bytes hold 65536+n); and the second byte is a sign byte, 00 for '+' and FF for '-'.

2D2B	STACK-BC	LD	IY,+5C3A	Re-initialise IY to ERR-NR.
		XOR	A	Clear the A register.
		LD	E,A	And the E register, to indicate '+'.
		LD	D,C	Copy the less significant byte to D.
		LD	C,B	And the more significant byte to C.
		LD	B,A	Clear the B register.
		CALL	2AB6,STK-STORE	Now stack the number.
		RST	0028,FP-CALC	Make HL point to
		DEFB	+38,end-calc	STKEND-5.
		AND	A	Clear the carry flag.
		RET		Finished.

THE 'INTEGER TO FLOATING-POINT' SUBROUTINE

This subroutine returns a 'last value' on the calculator stack that is the result of converting an integer in a BASIC line, i.e. the integer part of the decimal number or the line number, to its floating-point form.

Repeated calls to CH-ADD+1 fetch each digit of the integer in turn. An exit is made when a code that does not represent a digit has been fetched.

2D3B	INT-TO-FP	PUSH	AF	Save the first digit - in A.
		RST	0028,FP-CALC	Use the calculator.
		DEFB	+A0,stk-zero	The 'last value' is now zero.
		DEFB	+38,end-calc	
		POP	AF	Restore the first digit.

Now a loop is set up. As long as the code represents a digit then the floating-point form is found and stacked under the 'last value'. The 'last value' is then multiplied by decimal 10 and added to the 'digit' to form a new 'last value' which is carried back to the start of the loop.

2D40	NXT-DGT-2	CALL	2D22,STK-DIGIT	<p>If the code represents a digit then stack the floating-point form. Use the calculator. 'Digit' goes under 'last value'. Define decimal 10. 'Last value' = 'last value' *10. 'Last value' = 'last value'+ 'digit'. The next code goes into A. Loop back with this code.</p>
		RET	C	
		RST	0028,FP-CALC	
		DEFB	+01,exchange	
		DEFB	+A4,stk-ten	
		DEFB	+04,multiply	
		DEFB	+0F,addition	
		DEFB	+38,end-calc	
		CALL	0074,CH-ADD+1	
		JR	2D40,NXT-DGT-2	

THE ARITHMETIC ROUTINES

THE 'E-FORMAT TO FLOATING-POINT' SUBROUTINE

(Offset 3C - see CALCULATE below: 'e-to-fp')

This subroutine gives a 'last value' on the top of the calculator stack that is the result of converting a number given in the form xEm, where m is a positive or negative integer. The subroutine is entered with x at the top of the calculator stack and m in the A register.

The method used is to find the absolute value of m, say p, and to multiply or divide x by 10^p according to whether m is positive or negative.

To achieve this, p is shifted right until it is zero, and x is multiplied or divided by $10^{(2^n)}$ for each set bit b(n) of p. Since p is never much more than decimal 39, bits 6 and 7 of p will not normally be set.

2D4F	E-TO-FP	RLCA RRCA		Test the sign of m by rotating bit 7 of A into the carry without changing A.
		JR	NC,2D55,E-SAVE	Jump if m is positive.
		CPL		Negate m in A without disturbing the carry flag.
		INC	A	Save m in A briefly.
2D55	E-SAVE	PUSH	AF	This is MEMBOT: a sign flag is now stored in the first byte of mem-0, i.e. 0 for '+' and 1 for
		LD	HL,+5C92	The stack holds x.
		CALL	350B,FP-0/1	x, 10 (decimal)
		RST	0028,FP-CALC	x, 10
		DEFB	+A4,stk-ten	Restore m in A.
		DEFB	+38,end-calc	In the loop, shift out the next bit of m, modifying the carry and zero flags appropriately; jump if carry reset.
		POP	AF	Save the rest of m and the flags.
2D60	E-LOOP	SRL	A	The stack holds x' and $10^{(2^n)}$, where x' is an interim stage in the multiplication of x by 10^m , and n= 0,1,2,3,4 or 5.
		JR	NC,2D71,E-TST-END	($10^{(2^n)}$ is copied to mem-1).
		PUSH	AF	x', $10^{(2^n)}$, (1/0)
		RST	0028,FP-CALC	x', $10^{(2^n)}$
		DEFB	+C1,st-mem-1	x', $10^{(2^n)}$
		DEFB	+E0,get-mem-0	x'* $10^{(2^n)}$ = x"
		DEFB	+00,jump-true	x"
		DEFB	+04,to E-DIVSN	x/10 ^(2^n) =x" (x" is N*10 ^(2^n) or x'/10 ^(2^n) according as m is '+' or '-').
		DEFB	+04,multiply	x", 10 ^(2^n)
		DEFB	+33,jump	Restore the rest of m in A, and the flags.
		DEFB	+02,to E-FETCH	Jump if m has been reduced to zero.
		DEFB	+05,division	Save the rest of m in A.
2D6D	E-DIVSN	DEFB		x", 10 ^(2^n)
		DEFB		x", 10 ^(2^n) , 10 ^(2^n)
		DEFB		x", 10 ^{(2^(n+1))}
		DEFB		x", 10 ^{(2^(n+1))}
		DEFB		Restore the rest of m in A.
		DEFB		Jump back for all bits of m.
2D6E	E-FETCH	DEFB	+E1,get-mem-1	Use the calculator to delete the
		DEFB	+38,end-calc	
		POP	AF	
2D71	E-TST-END	JR	Z,2D7B,E-END	
		PUSH	AF	
		RST	0028,FP-CALC	
		DEFB	+31,duplicate	
		DEFB	+04,multiply	
		DEFB	+38,end-calc	
		POP	AF	
		JR	2D60,E-LOOP	
2D7B	E-END	RST	0028,FP-CALC	

DEFB	+02,delete	final power of 10 reached,
DEFB	+28,end-calc	leaving the 'last value' x*10^m
RET		on the stack

THE 'INT-FETCH' SUBROUTINE

This subroutine collects in DE a small integer n ($-65535 \leq n \leq 65535$) from the location addressed by HL: i.e. n is normally the first (or second) number at the top of the calculator stack; but HL can also access (by exchange with DE) a number which has been deleted from the stack. The subroutine does not itself delete the number from the stack or from memory; it returns HL pointing to the fourth byte of the number in its original position.

2D7F	INT-FETCH	INC	HL	Point to the sign byte of the number.
		LD	C,(HL)	Copy the sign byte to C.

The following mechanism will two's complement the number if it is negative (C is FF) but leave it unaltered if it is positive (C is 00)

INC	HL	Point to the less significant byte.
LD	A,(HL)	Collect the byte in A.
XOR	C	Ones complement it if negative
SUB	C	This adds 1 for negative numbers; it sets the carry unless the byte was 0.
LD	E,A	Less significant byte to E now.
INC	HL	Point to the more significant byte.
LD	A,(HL)	Collect it in A.
ADC	A,C	Finish two complementing in the case of a negative number; note that the carry is always left reset.
LD	D,A	More significant byte to D now.
RET		Finished.

THE 'INT-STORE' SUBROUTINE

This subroutine stores a small integer n ($-65535 \leq n \leq 65535$) in the location addressed by HL and the four following locations: i.e. n replaces the first (or second) number at the top of the calculator stack. The subroutine returns HL pointing to the first byte of n on the stack.

2D8C	P-INT-STO	LD	C,+00	This entry point would store a number known to be positive
2D8E	INT-STORE	PUSH	HL	The pointer to the first location is saved.
		LD	(HL),+00	The first byte is set to zero.
		INC	HL	Point to the second location.
		LD	(HL),C	Enter the second byte.

The same mechanism is now used as in 'INT-FETCH' to two's complement negative numbers. This is needed e.g. before and after the multiplication of small integers. Addition is however performed without any further two's complementing before or afterwards.

INC	HL	Point to the third location.
LD	A,E	Collect the less significant byte.
XOR	C	Two's complement it if the number is negative
SUB	C	Store the byte.
LD	(HL),A	Point to the fourth location.
INC	HL	Collect the more significant byte.
LD	A,D	Two's complement it if the number is negative
ADC	A,C	
XOR	C	

LD	(HL),A	Store the byte.
INC	HL	Point to the fifth location.
LD	(HL),+00	The fifth byte is set to zero.
POP	HL	Return with HL pointing to the first byte of n on the stack
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 65536 decimal, then the subroutine returns with the carry flag set. If the 'last value' is negative then the zero flag is reset. The low byte of the result is also copied to the A register.

2DA2	FP-TO-BC	RST	0028,FP-CALC	Use the calculator to make HL point to STKEND-5
		DEFB	+38,end-calc	Collect the exponent byte of the 'last value'; jump if it is
		LD	A,(HL)	'small integer'.
		AND	A	
		JR	Z,2DAD,FP-DELETE	zero, indicating a 'small integer'.
		RST	0028,FP-CALC	Now use the calculator to round the 'last value' to the nearest integer, which also changes it to 'small integer' form on the calculator stack if that is possible, i.e. if $-65535.5 \leq x < 65535.5$
		DEFB	+A2,stk-half	
		DEFB	+0F,addition	
		DEFB	+27,int	
		DEFB	+38,end-calc	
2DAD	FP-DELETE	RST	0028,FP-CALC	Use the calculator to delete the integer from the stack; DE still points to it in memory (at STKEND).
		DEFB	+92,delete	
		DEFB	+38,end-calc	Save both stack pointers.
		PUSH	HL	
		PUSH	DE	
		EX	DE,HL	HL now points to the number.
		LD	B,(HL)	Copy the first byte to B.
		CALL	2D7F,INT-FETCH	Copy bytes 2, 3 and 4 to C, E and D.
		XOR	A	Clear the A register.
		SUB	B	This sets the carry unless B is zero.
		BIT	7,C	This sets the zero flag if the number is positive (NZ denotes negative).
		LD	B,D	Copy the high byte to B.
		LD	C,E	And the low byte to C.
		LD	A,E	Copy the low byte to A too.
		POP	DE	Restore the stack pointers.
		POP	HL	
		RET		Finished.

THE 'LOG (2^A)' SUBROUTINE

This subroutine is called by the 'PRINT-FP' subroutine to calculate the approximate number of digits before the decimal in x, the number to be printed, or, if there are no digits before the decimal, then the approximate number of leading zeros after the decimal. It is entered with the A register containing e', the true exponent of x, or e'-2, and calculates $z = \log$ to the base 10 of (2^A) . It then sets A equal to $\text{ABS INT}(z + 0.5)$, as required, using FP-TO-A for this purpose.

2DC1	LOG(2^A)	LD	D,A	The integer A is stacked, either as 00 00 A 00 00 (for positive A) or as 00 FF A FF 00 (for negative A).
		RLA		
		SBC	A,A	
		LD	E,A	These bytes are first loaded into A, E, D, C, B and then STK-STORE is called to put the number on the calculator stack.
		LD	C,A	
		XOR	A	
		LD	B,A	

CALL	2AB6,STK-STORE	
RST	0028,FP-CALC	The calculator is used
DEFB	+34,stk-data	Log 2 to the base 10 is now stacked.
DEFB	+EF,exponent +7F	The stack now holds A, log 2.
DEFB	+1A,+20,+9A,+85	
DEFB	+04,multiply	A*log 2 i.e. log (2^A)
DEFB	+27,int	INT log (2^A)
DEFB	+38,end-calc	

The subroutine continues on into FP-TO-A to complete the calculation.

THE 'FLOATING-POINT TO A' SUBROUTINE

This short but vital subroutine is called at least 8 times for various purposes. It uses the last but one subroutine, FP-TO-BC, to get the 'last value' into the A register where this is possible. It therefore tests whether the modulus of the number rounds to more than 255 and if it does the subroutine returns with the carry flag set. Otherwise it returns with the modulus of the number, rounded to the nearest integer, in the A register, and the zero flag set to imply that the number was positive, or reset to imply that it was negative.

2DD5	FP-TO-A	CALL	2DA2,FP-TO-BC	Compress the 'last value' into BC.
		RET	C	Return if out of range already.
		PUSH	AF	Save the result and the flags.
		DEC	B	Again it will be out of range
		INC	B	if the B register does not hold zero.
		JR	Z,2DE1,FP-A-END	Jump if in range.
		POP	AF	Fetch the result and the flags
		SCF		Signal the result is out of range.
		RET		Finished - unsuccessful.
2DE1	FP-A-END	POP	AF	Fetch the result and the flags.
		RET		Finished - successful.

THE 'PRINT A FLOATING-POINT NUMBER' SUBROUTINE

This subroutine is called by the PRINT command routine at 2039 and by STR\$ at 3630, which converts to a string the number as it would be printed. The subroutine prints x, the 'last value' on the calculator stack. The print format never occupies more than 14 spaces. The 8 most significant digits of x, correctly rounded, are stored in an ad hoc print buffer in mem-3 and mem-4. Small numbers, numerically less than 1, and large numbers, numerically greater than 2^{27} , are dealt with separately. The former are multiplied by 10^n , where n is the approximate number of leading zeros after the decimal, while the latter are divided by $10^{(n-7)}$, where n is the approximate number of digits before the decimal. This brings all numbers into the middle range, and the numbers of digits required before the decimal is built up in the second byte of mem-5. Finally the printing is done, using E-format if there are more than 8 digits before the decimal or, for small numbers, more than 4 leading zeros after the decimal.

The following program shows the range of print formats:

```
10 FOR a=-11 TO 12: PRINT SGN a*9^a,: NEXT a
```

i. First the sign of x is taken care of:

If X is negative, the subroutine jumps to PF-NEGATIVE, takes ABS x and prints the minus sign.

If x is zero, x is deleted from the calculator stack, a '0' is printed and a return is made from the subroutine.

If x is positive, the subroutine just continues.

2DE3	PRINT-FP	RST	0028,FP-CALC	Use the calculator
		DEFB	+31,duplicate	x,x
		DEFB	+36,less-0	x, (1/0) Logical value of x.
		DEFB	+00,jump-true	x
		DEFB	+0B,to PF-NEGTV	x
		DEFB	+31,duplicate	x,x
		DEFB	+37,greater-0	x, (1/0) Logical value of X.

		DEFB	+00,jump-true	x
		DEFB	+0D,to PF-POSTVE	x Hereafter x'=ABS x.
		DEFB	+02,delete	-
		DEFB	+38,end-calc	-
		LD	A,+30	Enter the character code for '0'.
		RST	0010,PRINT-A-1	Print the '0'.
		RET		Finished as the 'last value' is zero.
2DF2	PF-NEGTV	DEFB	+2A,abs	x' x'=ABS x.
		DEFB	+38,end-calc	x'
		LD	A,+2D	Enter the character code for '-'.
		RST	0010,PRINT-A-1	Print the '-'.
		RST	0028,FP-CALC	Use the calculator again.
2DF8	PF-POSTVE	DEFB	+A0,stk-zero	The 15 bytes of mem-3, mem-4 and mem-5 are now initialised to zero to be used for a print buffer and two counters.
		DEFB	+C3,st-mem-3	
		DEFB	+C4,st-mem-4	
		DEFB	+C5,st-mem-5	
		DEFB	+02,delete	The stack is cleared, except for x'.
		DEFB	+38,end-calc	x'
		EXX		H'L', which is used to hold calculator offsets, (e.g. for 'STR\$') is saved on the machine stack.
		PUSH	HL	
		EXX		

ii. This is the start of a loop which deals with large numbers. However every number x is first split into its integer part i and the fractional part f. If i is a small integer, i.e. if $-65535 \leq i \leq 65535$, it is stored in D'E' for insertion into the print buffer.

2E01	PF-LOOP	RST	0028,FP-CALC	Use the calculator again.
		DEFB	+31,duplicate	x', x'
		DEFB	+27,int	x', INT (x')=i
		DEFB	+C2,st-mem-2	(i is stored in mem-2).
		DEFB	+03,subtract	x'-i=f
		DEFB	+E2,get-mem-2	f,i
		DEFB	+01,exchange	i,f
		DEFB	+C2,st-mem-2	(f is stored in mem-2).
		DEFB	+03,delete	i
		DEFB	+38,end-calc	i
		LD	A,(HL)	Is i a small integer (first byte zero) i.e. is $ABS\ i \leq 65535$?
		AND	A	
		JR	NZ,2E56,PF-LARGE	Jump if it is not
		CALL	2D7F,INT-FETCH	i is copied to DE (i, like x', ≥ 0).
		LD	B,+10	B is set to count 16 bits.
		LD	A,D	D is copied to A for testing:
		AND	A	Is it zero?
		JR	NZ,2E1E,PF-SAVE	Jump if it is not zero.
		OR	E	Now test E.
		JR	Z,2E24,PF-SMALL	Jump if DE zero: x is a pure fraction.
		LD	D,E	Move E to D and set B for 8 bits: D was zero and E was not.
2E1E	PF-SAVE	LD	B,+08	
		PUSH	DE	Transfer DE to D'E', via the machine stack, to be moved into the print buffer at PF-BITS.
		EXX		
		POP	DE	
		EXX		
		JR	2E78,PF-BITS	Jump forward.

iii. Pure fractions are multiplied by 10^n , where n is the approximate number of leading zeros after the decimal; and -n is added to the second byte of mem-5, which holds the number of digits needed before the decimal; a negative number here indicates leading zeros after the decimal;

2E24	PF-SMALL	RST	0028,FP-CALC	i (i=zero here),
		DEFB	+E2,get-mem-2	i,f

DEFB +38,end-calc i, f

Note that the stack is now unbalanced. An extra byte 'DEFB +02, delete' is needed at 2E25, immediately after the RST 0028. Now an expression like "2" +STR\$ 0.5 is evaluated incorrectly as 0.5; the zero left on the stack displaces the "2" and is treated as a null string. Similarly all the string comparisons can yield incorrect values if the second string takes the form STR\$ x where x is numerically less than 1; e.g. the expression "50"<STR\$ 0 .1 yields the logical value "true"; once again "" is used instead of "50".

LD	A,(HL)	The exponent byte e of f is copied to A.
SUB	+7E	A becomes e - 126 dec i.e. e'+2, where e' is the true exponent of f.
CALL	2DC1,LOG (2^A)	The construction A = ABS INT (LOG (2^A)) is performed (LOG is to base 10); i.e. A=n, say: n is copied from A to D.
LD	D,A	The current count is collected from the second byte of mem-5 and n is subtracted from it.
LD	A,(mem-5-2nd),A	n is copied from D to A.
LD	A,D	y=f*10^n is formed and stacked.
CALL	2D4F,E-TO-FP	
RST	0028,FP-CALC	i, y
DEFB	+31,duplicate	i, y, y
DEFB	+27,int	i, y, (INT (y) = i2
DEFB	+C1,st-mem-1	(i2 is copied to mem-1).
DEFB	+03,subtract	i, y - i2
DEFB	+E1,get-mem-1	i, y - i2, i2
DEFB	+38,end-calc	i, f2, i2 (f2 = y - i2)
CALL	2DD5,FP-TO-A	i2 is transferred from the stack to A.
PUSH	HL	The pointer to f2 is saved.
LD	(mem-3-1st),A	i2 is stored in the first byte of mem-3: a digit for printing.
DEC	A	i2 will not count as a digit for printing if it is zero; A is manipulated so that zero will produce zero but a non-zero digit will produce 1.
RLA	A,A	The zero or one is inserted into the first byte of mem-5 (the no. of digits for printing) and added to the second byte of mem-5 (the number of digits before the decimal).
SBC	A,A	
INC	A	
LD	HL,+5CAB	
LD	(HL),A	
INC	HL	
ADD	A,(HL)	
LD	(HL),A	
POP	HL	The pointer to f2 is restored.
JP	2ECF,PF-FRACTN	Jump to store f2 in buffer (HL now points to f2, DE to i2).

iv. Numbers greater than 2^{27} are similarly multiplied by $2^{-(n+7)}$, reducing the number of digits before the decimal to 8, and the loop is re-entered at PF-LOOP.

2E56	PF-LARGE	SUB	+80	e - 80 hex = e', the true exponent of i.
		CP	+1C	Is e' less than 28 decimal?
		JR	C,2E6F,PF-MEDIUM	Jump if it is less.
		CALL	2DC1,LOG (2^A)	n is formed in A.
		SUB	+07	And reduced to n - 7.
		LD	B,A	Then copied to B.
		LD	HL,+5CAC	n - 7 is added in to the second
		ADD	A,(HL)	byte of mem-5, the number of
		LD	(HL),A	digits required before the

LD	A,B	decimal in x. Then i is multiplied by $10^{-(n+7)}$
NEG		This will bring it into medium range for printing.
CALL	2D4F,E-TO-FP	Round the loop again to deal with the now medium-sized number.
JR	2E01,PF-LOOP	

v. The integer part of x is now stored in the print buffer in mem-3 and mem-4.

2E6F	PF-MEDIUM	EX CALL	DE,HL 2FBA,FETCH-TWO	DE now points to i, HL to f. The mantissa of i is now in D',E',D,E.
		EXX SET	7,D	Get the exchange registers. True numerical bit 7 to D'.
		LD	A,L	Exponent byte e of i to A.
		EXX		Back to the main registers.
		SUB	+80	True exponent e'=e - 80 hex to A.
		LD	B,A	This gives the required bit count.

Note that the case where i is a small integer (less than 65536) re-enters here.

2E7B	PF-BITS	SLA RL EXX RL	E D E	The mantissa of i is now rotated left and all the bits of i are thus shifted into mem-4 and each byte of mem-4 is decimal adjusted at each shift.
		RL EXX	D	All four bytes of i.
		LD	HL,+5CAA	Back to the main registers.
		LD	C,+05	Address of fifth byte of mem-4 to HL; count of 5 bytes to C.
2E8A	PF-BYTES	LD ADC	A,(HL) A,A	Get the byte of mem-4. Shift it left, taking in the new bit.
		DAA		Decimal adjust the byte.
		LD	(HL),A	Restore it to mem-4.
		DEC	HL	Point to next byte of mem-4.
		DEC	C	Decrease the byte count by one.
		JR	NZ,2E8A,PF-BYTES	Jump for each byte of mem-4.
		DJNZ	2E7B,PF-BITS	Jump for each bit of INT (x).

Decimal adjusting each byte of mem-4 gave 2 decimal digits per byte, there being at most 9 digits. The digits will now be re-packed, one to a byte, in mem-3 and mem-4, using the instruction RLD.

		XOR	A	A is cleared to receive the digits.
		LD	HL,+5CA6	Source address: first byte of mem-4.
		LD	DE,+5CA1	Destination: first byte of mem-3.
		LD	B,+09	There are at most 9 digits.
		RLD		The left nibble of mem-4 is discarded.
		LD	C,+FF	FF in C will signal a leading zero, 00 will signal a non-leading zero.
2EA1	PF-DIGITS	RLD		Left nibble of (HL) to A, right nibble of (HL) to left.
		JR	NZ,2EA9,PF-INSERT	Jump if digit in A is not zero.
		DEC	C	Test for a leading zero:
		INC	C	it will now give zero reset.
		JR	NZ,2EB3,PF-TEST-2	Jump if it was a leading zero.
2EA9	PF-INSERT	LD	(DE),A	Insert the digit now.

		INC INC INC LD	DE (mem-5-1st) (mem-5-2nd) C,+00	Point to next destination. One more digit for printing, and one more before the decimal. Change the flag from leading zero to other zero.
2EB3	PF-TEST-2	BIT JR INC	0,B Z,2EB8,PF,ALL-9 HL	The source pointer needs to be incremented on every second passage through the loop, when B is odd.
2EB8	PF-ALL-9	DJNZ LD SUB JR DEC	2EA1,PF-DIGITS A,(mem-5-1st) +09 C,2ECB,PF-MORE (mem-5-1st)	Jump back for all 9 digits. Get counter: were there 9 digits excluding leading zeros? If not, jump to get more digits. Prepare to round: reduce count to 8.
		LD CP	A,+04 (mem-4-4th)	Compare 9th digit, byte 4 of mem-4, with 4 to set carry for rounding up.
2ECB	PF-MORE	JR RST DEFB DEFB DEFB	2F0C,PF-ROUND 0028,FP-CALC +02,delete +E2,get-mem-2 +38,end-calc	Jump forward to round up. Use the calculator again. - (i is now deleted). f f

vi. The fractional part of x is now stored in the print buffer.

2ECF	PF-FRACTN	EX CALL	DE,HL 2FBA,FETCH-TWO	DE now points to f. The mantissa of f is now in D',E',D,E.
		EXX LD SUB LD	A,+80 L L,+00	Get the exchange registers. The exponent of f is reduced to zero, by shifting the bits of f 80 hex - e places right, where L' contained e.
		SET	7,D	True numerical bit to bit 7 of D'.
2EDF	PF-FRN-LP	EXX CALL LP CP JR EXX RL	2FDD,SHIFT-FP A,(mem-5-1st) +08 C,2EEC,PR-FR-DGT D	Restore the main registers. Now make the shift. Get the digit count. Are there already 8 digits? If not, jump forward. If 8 digits, just use f to round i up, rotating D' left to set the carry.
2EEC	PF-FR-DGT	EXX JR	2F0C,PF-ROUND	Restore main registers and jump forward to round up.
2EEF	PF-FR-EXX	LD LD CALL LD LD CALL LD PUSH EXX POP DJNZ	BC,+0200 A,E 2F8B,CA=10*A+C E,A A,D 2F8B,CA=10*A+C D,A BC BC 2EEF,PF-FR-EXX	Initial zero to C, count of 2 to B. D'E'DE is multiplied by 10 in 2 stages, first DE then D'E', each byte by byte in 2 steps, and the integer part of the result is obtained in C to be passed into the print buffer.
		LD LD LD	HL,+5CA1 A,C C,(mem-5-1st)	The count and the result alternate between BC and B'C'.
		ADD LD	HL,BC (HL),A	Look back once through the exchange registers. The start - 1st byte of mem-3. Result to A for storing. Count of digits so far in number to C. Address the first empty byte. Store the next digit.

INC	(mem-5-1st)	Step up the count of digits.
JR	2EDF,PF-FRN-LP	Loop back until there are 8 digits.

vii. The digits stored in the print buffer are rounded to a maximum of 8 digits for printing.

2F0C	PF-ROUND	PUSH	AF	Save the carry flag for the rounding.
		LD	HL,+5CA1	Base address of number: mem-3, byte 1.
		LD	C,(mem-5-1st)	Offset (number of digits in number) to BC.
		LD	B,+00	Address the last byte of the number.
		ADD	HL,BC	Copy C to B as the counter.
		LD	B,C	Restore the carry flag.
2F18	PF-RND-LP	POP	AF	This is the last byte of the number.
		DEC	HL	Get the byte into A.
		LD	A,(HL)	Add in the carry i.e. round up.
		ADC	A,+00	Store the rounded byte in the buffer.
		LD	(HL),A	If the byte is 0 or 10, B will be decremented and the final zero (or the 10) will not be counted for printing.
		AND	A	Reset the carry for a valid digit.
		JR	Z,2F25,PF-R-BACK	Jump if carry reset.
		CP	+0A	Jump back for more rounding or more final zeros.
		CCF		There is overflow to the left; an extra 1 is needed here.
2F25	PF-R-BACK	JR	NC,2F2D,PF-COUNT	It is also an extra digit before the decimal.
		DJNZ	2F18,PF-RND-LP	B now sets the count of the digits to be printed (final zeros will not be printed).
		LD	(HL),+01	f is to be deleted.
		INC	B	
		INC	(mem-5-2nd)	
2F2D	PF-COUNT	LD	(mem-5-1st),B	
		RST	0028,FP-CALC	
		DEFB	+02,delete	-
		DEFB	+38,end-calc	-
		EXX		The calculator offset saved on the stack is restored to H'L'.
		POP	HL	
		EXX		

viii. The number can now be printed. First C will be set to hold the number of digits to be printed, not counting final zeros, while B will hold the number of digits required before the decimal.

		LD	BC,(mem-5-1st)	The counters are set.
		LD	HL,+5CA1	The start of the digits.
		LD	A,B	If more than 9, or fewer than minus 4, digits are required before the decimal, then E-format will be needed.
		CP	+09	Fewer than 4 means more than 4 leading zeros after the decimal.
		JR	C,2F46,PF-NOT-E	Are there no digits before the decimal? If so, print an initial zero.
		CP	+FC	
2F46	PF-NOT-E	JR	C,2F6C,PF-E-FRMT	
		AND	A	
		CALL	Z,15EF,OUT-CODE	

The next entry point is also used to print the digits needed for E-format printing.

2F4A	PF-E-SBRN	XOR	A	Start by setting A to zero.
------	-----------	-----	---	-----------------------------

		SUB JR	B M,2F52,PF-OUT-LP	Subtract B: minus will mean there are digits before the decimal; jump forward to print them.
		LD JR	B,A 2F5E,PF-DC-OUT	A is now required as a counter. Jump forward to print the decimal part.
2F52	PF-OUT-LP	LD AND JR	A,C A Z,2F59,PF-OUT-DT	Copy the number of digits to be printed to A. If A is 0, there are still final zeros to print (B is non-zero), so jump.
2F59	PF-OUT-DT	LD INC DEC CALL DJNZ	A,(HL) HL C 15EF,OUT-CODE 2F52,PF-OUT-LP	Get a digit from the print buffer. Point to the next digit. Decrease the count by one. Print the appropriate digit. Loop back until B is zero.
2F5E	PF-DC-OUT	LD AND RET INC	A,C A Z B	It is time to print the decimal, unless C is now zero; in that case, return - finished. Add 1 to B - include the decimal.
2F64	PF-DEC-0S	LD RST LD	A,+2E 0010,PRINT-A-1 A,+30	Put the code for '.' into A. Print the '.'.
		DJNZ	2F64,PF-DEC-0S	Enter the character code for '0'. Loop back to print all needed zeros.
		LD	B,C	Set the count for all remaining digits.
2F6C	PF-E-FRMT	JR LD	2F52,PF-OUT-LP D,B	Jump back to print them. The count of digits is copied to D.
		DEC	D	It is decremented to give the exponent.
		LD	B,+01	One digit is required before the decimal in E-format.
		CALL	2F4A,PF-E-SBRN	All the part of the number before the 'E' is now printed.
		LD	A,+45	Enter the character code for 'E'.
		RST LD LD AND JP NEG LD	0010,PRINT-A-1 C,D A,C A P,2F83,PF-E-POS C,A	Print the 'E'. Exponent to C now for printing. And to A for testing. Its sign is tested. Jump if it is positive. Otherwise, negate it in A. Then copy it back to C for printing.
2F83	PF-E-POS	LD JR LD	A,+2D 2F85,PF-E-SIGN A,+2B	Enter the character code for '-'. Jump to print the sign. Enter the character code for '+'.
2F85	PF-E-SIGN	RST LD	0010,PRINT-A-1 B,+00	Now print the sign: '+' or '-'. BC holds the exponent for printing.
		JP	1A1B,OUT-NUM	Jump back to print it and finish.

THE 'CA=10*A+C' SUBROUTINE'

This subroutine is called by the PRINT-FP subroutine to multiply each byte of D'E'DE by 10 and return the integer part of the result in the C register. On entry, the A register contains the byte to be multiplied by 10 and the C register contains the carry over from the previous byte. On return, the A register contains the resulting byte and the C register the carry forward to the next byte.

2F8B	CA=10*A+C	PUSH	DE	Save whichever DE pair is in use.
		LD	L,A	Copy the multiplicand from A
		LD	H,+00	to HL.
		LD	E,L	Copy it to DE too.
		LD	D,H	
		ADD	HL,HL	Double HL.
		ADD	HL,HL	Double it again.
		ADD	HL,DE	Add in DE to give HL=5*A.
		ADD	HL,HL	Double again: now HL=10*A.
		LD	E,C	Copy C to DE (D is zero) for
				addition.
		ADD	HL,DE	Now HL=10*A+C.
		LD	C,H	H is copied to C.
		LD	A,L	L is copied to A, completing
				the task.
		POP	DE	The DE register pair is restored.
		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 (two's complement) if it is negative. The exponent is returned in the A register and the first byte is set to Hex.00 for a positive number and Hex.FF for a negative number.

2F9B	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
		PUSH	BC	need to be twos complemented.
		LD	BC,+0005	Save any earlier exponent.
		ADD	HL,BC	There are 5 bytes to be handled.
		LD	B,C	Point one-past the last byte.
		LD	C,A	Transfer the '5' to B.
		SCF		Save the exponent in C.
				Set carry flag for negation.
2FAF	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	2FAF,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 register, including the exchange registers.

On entry to the subroutine the HL register pair points to the first byte of the first number and the DE register pair points to the first byte of the second number.

When the subroutine is called from MULTIPLICATION or DIVISION the sign of the result is saved in the second byte of the first number.

2FB4	FETCH-TWO	PUSH PUSH	HL AF	HL is preserved. AF is preserved.
Call the five bytes of the first number and the second number			- M1, M2, M3, M4 & M5. - N1, N2, N3, N4 & N5.	
		LD	C,(HL)	M1 to C.
		INC	HL	Next.
		LD	B,(HL)	M2 to B.
		LD	(HL),A	Copy the sign of the result to (HL).
		INC	HL	Next.
		LD	A,C	M1 to A.
		LD	C,(HL)	M3 to C.
		PUSH	BC	Save M2 & M3 on the machine stack.
		INC	HL	Next.
		LD	C,(HL)	M4 to C.
		INC	HL	Next.
		LD	B,(HL)	M5 to B.
		EX	DE,HL	HL now points to N1.
		LD	D,A	M1 to D.
		LD	E,(HL)	N1 to E.
		PUSH	DE	Save M1 & N1 on the machine stack.
		INC	HL	Next.
		LD	D,(HL)	N2 to D.
		INC	HL	Next.
		LD	E,(HL)	N3 to E.
		PUSH	DE	Save N2 & N3 on the machine stack.
		EXX		Get the exchange registers.
		POP	DE	N2 to D' & N3 to E'.
		POP	HL	M1 to H' & N1 to L'.
		POP	BC	M2 to B' & M3 to C'.
		EXX		Get the original set of registers.
		INC	HL	Next.
		LD	D,(HL)	N4 to D.
		INC	HL	Next.
		LD	E,(HL)	N5 to E.
		POP	AF	Restore the original AF.
		POP	HL	Restore the original HL.
		RET		Finished.

Summary: M1 - M5 are in H', B', C', C, B.
N1 - N5 are in: L', D', E', D, E.
HL points to the first byte of the first number.

THE 'SHIFT ADDEND' SUBROUTINE

This subroutine shifts a floating-point number up to 32 decimal, Hex.20, places right to line it up properly for addition. The number with the smaller exponent has been put in the addend position before this subroutine is called. Any overflow to the right, into the carry, is added back into the number. If the exponent difference is greater than 32 decimal, or the carry ripples right back to the beginning of the number then the number is set to zero so that the addition will not alter the other number (the augend).

2FDD	SHIFT-FP	AND RET CP JR PUSH LD	A Z +21 NC,2FF9,ADDEND-0 BC B,A	If the exponent difference is zero, the subroutine returns at once. If the difference is greater than Hex.20, jump forward. Save BC briefly. Transfer the exponent difference to B to count the shifts right.
2FE5	ONE-SHIFT	EXX SRA	L	Arithmetic shift right for L', preserving the sign marker bits.

		RR	D	Rotate right with carry D', E',
		RR	E	D & E.
		EXX		Thereby shifting the whole five
		RR	D	bytes of the number to the right
		RR	E	as many times as B counts.
		DJNZ	2FE5,ONE-SHIFT	Loop back until B reaches zero.
		POP	BC	Restore the original BC.
		RET	NC	Done if no carry to retrieve.
		CALL	3004,ADD-BACK	Retrieve carry.
		RET	NZ	Return unless the carry rippled
				right back. (In this case there is
				nothing to add).
2FF9	ADDEND-0	EXX		Fetch L', D' & E'.
		XOR	A	Clear the A register.
2FFB	ZEROS-4/5	LD	L,+00	Set the addend to zero in D',E',
		LD	D,A	D & E, together with its marker
		LD	E,L	byte (sign indicator) L', which
		EXX		was Hex.00 for a positive
		LD	DE,+0000	number and Hex.FF for a
				negative number. ZEROS-4/5
				produces only 4 zero bytes
				when called for near underflow
				at 3160.
		RET		Finished.

THE 'ADD-BACK' SUBROUTINE

This subroutine adds back into the number any carry which has overflowed to the right. In the extreme case, the carry ripples right back to the left of the number.

When this subroutine is called during addition, this ripple means that a mantissa of 0.5 was shifted a full 32 places right, and the addend will now be set to zero; when called from MULTIPLICATION, it means that the exponent must be incremented, and this may result in overflow.

3004	ADD-BACK	INC	E	Add carry to rightmost byte.
		RET	NZ	Return if no overflow to left.
		INC	D	Continue to the next byte.
		RET	NZ	Return if no overflow to left.
		EXX		Get the next byte.
		INC	E	Increment it too.
		JR	NZ,300D,ALL-ADDED	Jump if no overflow.
		INC	D	Increment the last byte.
300D	ALL-ADDED	EXX		Restore the original
				registers.
		RET		Finished.

THE 'SUBTRACTION' OPERATION

(Offset 03 - see CALCULATE below: 'subtract')

This subroutine simply changes the sign of the subtrahend and carried on into ADDITION.

Note that HL points to the minuend and DE points to the subtrahend. (See ADDITION for more details.)

300F	SUBTRACT	EX	DE,HL	Exchange the pointers.
		CALL	346E,NEGATE	Change the sign of the
				subtrahend.
		EX	DE,HL	Exchange the pointers back and
				continue into ADDITION.

THE 'ADDITION' OPERATION

(Offset 0F - see CALCULATE below: 'addition')

The first of three major arithmetical subroutines, this subroutine carries out the floating-point addition of two numbers, each with a 4-byte mantissa and a 1-byte exponent. In these three subroutines, the two numbers at the top of the calculator stack are added/multiplied/divided to give one number at the top of the calculator stack, a 'last value'.

HL points to the second number from the top, the augend/multiplier/dividend. DE points to the number at the top of the calculator stack, the addend/multiplicand/divisor. Afterwards HL points to the resultant 'last value' whose address can also be considered to be STKEND - 5.

But the addition subroutine first tests whether the 2 numbers to be added are 'small integers'. If they are, it adds them quite simply in HL and BC, and puts the result directly on the stack. No twos complementing is needed before or after the addition, since such numbers are held on the stack in twos complement form, ready for addition.

3014	addition	LD	A,(DE)	Test whether the first bytes of
		OR	(HL)	both numbers are zero.
		JR	NZ,303E,FULL-ADDN	If not, jump for full addition.
		PUSH	DE	Save the pointer to the second
				number.
		INC	HL	Point to the second byte of the
		PUSH	HL	first number and save that
				pointer too.
		INC	HL	Point to the less significant
				byte.
		LD	E,(HL)	Fetch it in E.
		INC	HL	Point to the more significant
				byte.
		LD	D,(HL)	Fetch it in D.
		INC	HL	Move on to the second byte of
		INC	HL	the second number.
		INC	HL	
		LD	A,(HL)	Fetch it in A (this is the sign
				byte).
		INC	HL	Point to the less significant
				byte.
		LD	C,(HL)	Fetch it in C.
		INC	HL	Point to the more significant
				byte.
		LD	B,(HL)	Fetch it in B.
		POP	HL	Fetch the pointer to the sign
		EX	DE,HL	byte of the first number; put it
				in DE, and the number in HL.
ADD	HL,BC	Perform the addition: result in		
		HL.		
EX	DE,HL	Result to DE, sign byte to HL.		
ADC	A,(HL)	Add the sign bytes and the carry		
RRCA		into A; this will detect any		
		overflow.		
ADC	A,+00	A non-zero A now indicates		
		overflow.		
JR	NZ,303C,ADDN-OFLW	Jump to reset the pointers and		
		to do full addition.		
SBC	A,A	Define the correct sign byte for		
		the result.		
3032	LD	(HL),A	Store it on the stack.	
	INC	HL	Point to the next location.	
	LD	(HL),E	Store the low byte of the result.	
	INC	HL	Point to the next location.	
	LD	(HL),D	Store the high byte of the	
			result.	
	DEC	HL	Move the pointer back to	
	DEC	HL	address the first byte of the	
	DEC	HL	result.	
	POP	DE	Restore STKEND to DE.	
RET		Finished.		

Note that the number -65536 decimal can arise here in the form 00 FF 00 00 00 as the result of the addition of two smaller negative integers, e.g. -65000 and -536. It is simply stacked in this form. This is a mistake. The Spectrum system cannot handle this number.

Most functions treat it as zero, and it is printed as -1E-38, obtained by treating it as 'minus zero' in an illegitimate format. One possible remedy would be to test for this number at about byte 3032 and, if it is present, to make the second byte 80 hex and the first byte 91 hex, so producing the full five byte floating-point form of the number, i.e. 91 80 00 00 00, which causes no problems. See also the remarks in 'truncate' below, before byte 3225, and the Appendix.

303C	ADDN-OFLW	DEC	HL	Restore the pointer to the first number.
		POP	DE	Restore the pointer to the second number.
303E	FULL-ADDN	CALL	3293,RE-ST-TWO	Re-stack both numbers in full five byte floating-point form.

The full ADDITION subroutine first calls PREP-ADD for each number, then gets the two numbers from the calculator stack and puts the one with the smaller exponent into the addend position. It then calls SHIFT-FP to shift the addend up to 32 decimal places right to line it up for addition. The actual addition is done in a few bytes, a single shift is made for carry (overflow to the left) if needed, the result is twos complemented if negative, and any arithmetic overflow is reported; otherwise the subroutine jumps to TEST-NORM to normalise the result and return it to the stack with the correct sign bit inserted into the second byte.

3055	SHIFT-LEN	EXX		Exchange the registers.
		PUSH	HL	Save the next literal address.
		EXX		Exchange the registers.
		PUSH	DE	Save pointer to the addend.
		PUSH	HL	Save pointer to the augend.
		CALL	2F9B,PREP-ADD	Prepare the augend.
		LD	B,A	Save its exponent in B.
		EX	DE,HL	Exchange its pointers.
		CALL	2F9B,PREP-ADD	Prepare the addend.
		LD	C,A	Save its exponent in C.
		CP	B	If the first exponent is smaller,
		JR	NC,3055,SHIFT-LEN	keep the first number in the
		LD	A,B	addend position; otherwise
		LD	B,C	change the exponents and the
		EX	DE,HL	pointers back again.
		PUSH	AF	Save the larger exponent in A.
		SUB	B	The difference between the
				exponents is the length of the
				shift right.
		CALL	2FBA,FETCH-TWO	Get the two numbers from the
				stack.
		CALL	2FDD,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 H & M5 to L,
		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 mechanisms 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
		EXX		of 2 negative numbers has not
				overflowed left.

		EX POP	DE,HL HL	The result is now in DED'E'. Get the pointer to the exponent.
		RRA JR	NC,307C,TEST-NEG	The test for shift (H', L' were Hex. 00 for positive numbers and Hex.FF for negative numbers).
		LD CALL INC JR	A,+01 2FDD,SHIFT-FP (HL) Z,309F,ADD-REP-6	A counts a single shift right. The shift is called. Add 1 to the exponent; this may lead to arithmetic overflow.
307C	TEST-NEG	EXX LD AND EXX INC LD DEC JR	A,L +80 HL (HL),A HL Z,30A5,GO-NC-MLT	Test for negative result: get sign bit of L' into A (this now correctly indicates the sign of the result). Store it in the second byte position of the result on the calculator stack. If it is zero, then do not twos complement the result.
		LD NEG CCF LD	A,E E,A	Get the first byte. Negate it. Complement the carry for continued negation, and store byte.
		LD CPL ADC LD EXX LD CPL ADC LD LD CPL ADC JR RRA EXX INC	A,D A,+00 D,A A,E A,+00 E,A A,D A,+00 NC,30A3,END-COMPL RRA EXX (HL)	Get the next byte. Ones complement it. Add in the carry for negation. Store the byte. Proceed to get next byte into the A register. Ones complement it. Add in the carry for negation. Store the byte. Get the last byte. Ones complement it. Add in the carry for negation. Done if no carry. Else, get .5 into mantissa and add 1 to the exponent; this will be needed when two negative numbers add to give an exact power of 2, and it may lead to arithmetic overflow.
309F	ADD-REP-6	JP EXX	Z,31AD,REPORT-6	Give the error if required.
30A3	END-COMPL	LD EXX	D,A	Store the last byte.
30A5	GO-NC-MLT	XOR JP	A 3155,TEST-NORM	Clear the carry flag. Exit via TEST-NORM.

THE 'HL=HL*DE' SUBROUTINE

This subroutine is called by 'GET-HL*DE' and by 'MULTIPLICATION' to perform the 16-bit multiplication as stated. Any overflow of the 16 bits available is dealt with on return from the subroutine.

30A9	HL=HL*DE	PUSH LD	BC B,+10	BC is saved. It is to be a 16 bit multiplication.
		LD LD LD	A,H C,L HL,+0000	A holds the high byte. C holds the low byte. Initialise the result to zero.
30B1	HL-LOOP	ADD	HL,HL	Double the result.

		JR	C,30BE,HL-END	Jump if overflow.
		RL	C	Rotate bit 7 of C into the carry.
		RLA		Rotate the carry bit into bit 0 and bit 7 into the carry flag.
		JR	NC,30BC,HL-AGAIN	Jump if the carry flag is reset.
		ADD	HL,DE	Otherwise add DE in once.
30BC	HL-AGAIN	JR	C,30BE,HL-END	Jump if overflow.
30BE	HL-END	DJNZ	30B1,HL-LOOP	Until 16 passes have been made.
		POP	BC	Restore BC.
		RET		Finished.

THE 'PREPARE TO MULTIPLY OR DIVIDE' SUBROUTINE

This subroutine prepares a floating-point number for multiplication or division, returning with carry set if the number is zero, getting the sign of the result into the A register, and replacing the sign bit in the number by the true numeric bit, 1.

30C0	PREP-M/D	CALL	34E9,TEST-ZERO	If the number is zero, return with the carry flag set.
		RET	C	Point to the sign byte.
		INC	HL	Get sign for result into A (like signs give plus, unlike give minus); also reset the carry flag.
		XOR	(HL)	Set the true numeric bit.
		SET	7,(HL)	Point to the exponent again.
		DEC	HL	Return with carry flag reset.
		RET		

THE 'MULTIPLICATION' OPERATION

(Offset 04 - see CALCULATE below: 'multiply')

This subroutine first tests whether the two numbers to be multiplied are 'small integers'. If they are, it uses INT-FETCH to get them from the stack, HL=HL*DE to multiply them and INT-STORE to return the result to the stack. Any overflow of this 'short multiplication' (i.e. if the result is not itself a 'small integer') causes a jump to multiplication in full five byte floating-point form (see below).

30CA	multiply	LD	A,(DE)	Test whether the first bytes of both numbers are zero.
		OR	(HL)	
		JR	NZ,30F0,MULT-LONG	If not, jump for 'long' multiplication.
		PUSH	DE	Save the pointers: to the second number.
		PUSH	HL	And to the first number.
		PUSH	DE	And to the second number yet again.
		CALL	2D7F,INT-FETCH	Fetch sign in C, number in DE.
		EX	DE,HL	Number to HL now.
		EX	(SP),HL	Number to stack, second pointer to HL.
		LD	B,C	Save first sign in B.
		CALL	2D7F,INT-FETCH	Fetch second sign in C, number in DE.
		LD	A,B	Form sign of result in A: like signs give plus (00), unlike give minus (FF).
		XOR	C	
		LD	C,A	Store sign of result in C.
		POP	HL	Restore the first number to HL.
		CALL	30A9,HL=HL*DE	Perform the actual multiplication.
		EX	DE,HL	Store the result in DE.
		POP	HL	Restore the pointer to the first number.
		JR	C,30EF,MULT-OFLW	Jump on overflow to 'full' multiplication.
30E5		LD	A,D	These 5 bytes ensure that

		OR	E	00 FF 00 00 00 is replaced by
		JR	NZ,30EA,MULT-RSLT	zero; that they should not be
		LD	C,A	needed if this number were
30EA	MULT-RSLT	CALL	2D8E,INT-STORE	excluded from the system (see
		POP	DE	after 303B) above).
		RET		Now store the result on the
		POP	DE	stack.
30EF	MULT-OFLW	POP	DE	Restore STKEND to DE.
				Finished.
30F0	MULT-LONG	CALL	3293,RE-ST-TWO	Restore the pointer to the
				second number.
				Re-stack both numbers in full
				five byte floating-point form.

The full MULTIPLICATION subroutine prepares the first number for multiplication by calling PREP-M/D, returning if it is zero; otherwise the second number is prepared by again calling PREP-M/D, and if it is zero the subroutine goes to set the result to zero. Next it fetches the two numbers from the calculator stack and multiplies their mantissas in the usual way, rotating the first number (treated as the multiplier) right and adding in the second number (the multiplicand) to the result whenever the multiplier bit is set. The exponents are then added together and checks are made for overflow and for underflow (giving the result zero). Finally, the result is normalised and returned to the calculator stack with the correct sign bit in the second byte.

		XOR	A	A is set to Hex.00 so that the
				sign of the first number will go
				into A.
		CALL	30C0,PREP-M/D	Prepare the first number, and
		RET	C	return 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 multi-
				plicand.
		EX	DE,HL	Exchange the pointers.
		CALL	30C0,PREP-M/D	Prepare the 2nd number.
		EX	DE,HL	Exchange the pointers again.
		JR	C,315D,ZERO-RSLT	Jump forward if 2nd number is
				zero.
		PUSH	HL	Save the pointer to the result.
		CALL	2FBA,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	3125,STRT-MLT	Jump forward into the loop.

Now enter the multiplier loop.

3114	MLT-LOOP	JR	NC,311B,NO-ADD	Jump forward to NO-ADD if no
				carry, i.e. the multiplier bit was
				reset.
		ADD	HL,DE	Else, add the multiplicand in
		EXX		D'E'DE (see FETCH-TWO) into
		ADC	HL,DE	the result being built up on

311B	NO-ADD	EXX		H'L'HL.
		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
3125	STRT-MLT	EXX		rotating each byte with carry, so
		RR		that any bit that drops into the
		RR	H	carry is picked up by the next
		RR	L	byte, and the shift continued
				into B'C'CA.
		EXX		Shift right the multiplier in
		RR	B	B'C'CA (see FETCH-TWO &
				above).
		RR	C	A final bit dropping into the
		EXX		carry will trigger another add of
		RR	C	the multiplicand to the result.
		RRA		
		DJNZ	3114,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		

Now 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
		ADD	A,C	exponent bytes in A, and the
				correct carry.
		JR	NZ,313B,MAKE-EXPT	If the sum equals zero then clear
313B	MAKE-EXPT	AND	A	the carry; else leave it
				unchanged.
		DEC	A	Prepare to increase the
		CCF		exponent by Hex.80.

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

313D	DIVN-EXPT	RLA		These few bytes very cleverly
		CCF		make the correct exponent
				byte.
		RRA		Rotating left then right gets the
3146	OFLW1-CLR			exponent byte (true exponent
				plus Hex.80) into A.
		JP	P,3146,OFLW1-CLR	If the sign flag is reset, no report
				of arithmetic overflow needed.
		JR	NC,31AD,REPORT-6	Report the overflow if carry
				reset.
		AND	A	Clear the carry now.
		INC	A	The exponent byte is now com-
		JR	NZ,3151,OFLW2-CLR	plete; but if A is zero a further
		JR	C,3151,OFLW2-CLR	check for overflow is needed.
3151	OFLW2-CLR	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
		JR	NZ,31AD,REPORT-6	overflow to report; but if bit 7
				of D' is reset, the result is just in
				range, i.e. just under 2**127.
		LD	(HL),A	Store the exponent byte, at last.
		EXX		Pass the fifth result byte to A
		LD	A,B	for the normalisation sequence,
		EXX		i.e. the overflow from L into B'.

The remainder of the subroutine deals with normalisation and is common to all the arithmetic routines.

3155	TEST-NORM	JR	NC,316C,NORMALISE	If no carry then normalise now. Else, deal with underflow (zero result) or near underflow (result $2^{**}-128$): return exponent to A, test if A is zero (case $2^{**}-128$) and if so produce $2^{**}-128$ if number is normal; otherwise produce zero. The exponent must then be set to zero (for zero) or 1 (for $2^{**}-128$). Restore the exponent byte. Jump if case $2^{**}-128$. Otherwise, put zero into second byte of result on the calculator stack. Jump forward to transfer the result.
		LD	A,(HL)	
		AND	A	
3159	NEAR-ZERO	LD	A,+80	
		JR	Z,315E,SKIP-ZERO	
315D	ZERO-RSLT	XOR	A	
315E	SKIP-ZERO	EXX		
		AND	D	
		CALL	2FFB,ZEROS-4/5	
		RLCA		
		LD	(HL),A	
		JR	C,3195,OFLOW-CLR	
		INC	HL	
		LD	(HL),A	
		DEC	HL	
		JR	3195,OFLOW-CLR	

The actual normalisation operation.

316C	NORMALISE	LD	B,+20	Normalise the result by up to 32 decimal, Hex.20, shifts left of D'E'DE (with A adjoined) until bit 7 of D' is set. A holds zero after addition so no precision is gained or lost; A holds the fifth byte from B' after multiplication or division; but as only about 32 bits can be correct, no precision is lost. Note that A is rotated circularly, with branch at carry eventually a random process. The exponent is decremented on each shift. If the exponent becomes zero, then number from $2^{**}-129$ are rounded up to $2^{**}-128$. Loop back, up to 32 times. If bit 7 never became 1 then the whole result is to be zero.
316E	SHIFT-ONE	EXX		
		BIT	7,D	
		EXX		
		JR	NZ,3186,NORML-NOW	
		RLCA		
		RL	E	
		RL	D	
		EXX		
		RL	E	
		RL	D	
		EXX		
		DEC	(HL)	
		JR	Z,3159,NEAR-ZERO	
		DJNZ	316E,SHIFT-ONE	
		JR	315D,ZERO-RSLT	

Finish the normalisation by considering the 'carry'.

3186	NORML-NOW	RLA		After normalisation add back any final carry that went into A. Jump forward if the carry does not ripple right back. If it should ripple right back then set mantissa to 0.5 and increment the exponent. This action may lead to arithmetic overflow (final case).
		JR	NC,3195,OFLOW-CLR	
		CALL	3004,ADD-BACK	
		JR	NZ,3195,OFLOW-CLR	
		EXX		
		LD	D,+80	
		EXX		
		INC	(HL)	
		JR	Z,31AD,REPORT-6	

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

3195	OFLOW-CLR	PUSH	HL	Save the result pointer. Point to the sign byte in the result. The result is moved from its present registers, D'E'DE, to BCDE; and then to ACDE.
		INC	HL	
		EXX		
		PUSH	DE	
		EXX		

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

Report 6 - Arithmetic overflow

31AD	REPORT-6	RST	0008,ERROR-1	Call the error handling routine.
		DEFB	+05	

THE 'DIVISION' OPERATION

(Offset 05 - see CALCULATE below: 'division')

This subroutine first prepared the divisor by calling PREP-M/D, reporting arithmetic overflow if it is zero; then it prepares the dividend again calling PREP-M/D, returning if it is zero. Next fetches the two numbers from the calculator stack and divides their mantissa by means of the usual restoring division, trial subtracting the divisor from the dividend and restoring if there is carry, otherwise adding 1 to the quotient. The maximum precision is obtained for a 4-byte division, and after subtracting the exponents the subroutine exits by joining the later part of MULTIPLICATION.

31AF	division	CALL	3293,RE-ST-TWO	Use full floating-point forms.
		EX	DE,HL	Exchange the pointers.
		XOR	A	A is set to Hex.00, so that the sign of the first number will go into A.
		CALL	30C0,PREP-M/D	Prepare the divisor and give the report for arithmetic overflow if it is zero.
		JR	C,31AD,REPORT-6	Exchange the pointers.
		EX	DE,HL	Prepare the dividend and return if it is zero (result already zero).
		CALL	30C0,PREP-M/D	Exchange the pointers.
		RET	C	Save the next literal address.
		EXX		Exchange the registers.
		PUSH	HL	Save pointer to divisor.
		EXX		Save pointer to dividend.
		PUSH	DE	Get the two numbers from the stack.
		PUSH	HL	Exchange the registers.
		CALL	2FBA,FETCH-TWO	Save M1 & N1 on the machine stack.
		EXX		Copy the four bytes of the dividend from registers B'C'CB (i.e. M2, M3, M4 & M5; see FETCH-TWO) to the registers H'L'HL.
		PUSH	HL	
		LD	H,B	
		LD	L,C	
		EXX		
		LD	H,C	
		LD	L,B	

XOR	A	Clear A and reset the carry flag. B will count upwards from -33 to -1, twos complement, Hex. DF to FF, looping on minus and will jump again on zero for extra precision. Jump forward into the division loop for the first trial subtraction.
LD	B,+DF	
JR	31E2,DIV-START	

Now enter the division loop.

31D2	DIV-LOOP	RLA RL EXX RL RL EXX	C C B	Shift the result left into B'C'CA, shifting out the bits already there, picking up 1 from the carry whenever it is set, and rotating left each byte with carry to achieve the 32 bit shift.
31DB	DIV-34TH	ADD EXX ADC EXX	HL,HL HL,HL	
		JR	C,31F2,SUBN-ONLY	Move what remains of the dividend left in H'L'HL before the next trial subtraction; if a bit drops into the carry, force no restore and a bit for the quotient, thus retrieving the lost bit and allowing a full 32-bit divisor.
31E2	DIV-START	SBC EXX SBC EXX JR	HL,DE HL,DE NC,31F9,NO-RSTORE	Trial subtract divisor in D'E'DE from rest of dividend in H'L'HL; there is no initial carry (see previous step). Jump forward if there is no carry.
		ADD EXX ADC EXX AND JR	HL,DE HL,DE A 31FA,COUNT-ONE	Otherwise restore, i.e. add back the divisor. Then clear the carry so that there will be no bit for the quotient (the divisor 'did not go'). Jump forward to the counter.
31F2	SUBN-ONLY	AND SBC EXX SBC EXX	A HL,DE HL,DE	Just subtract with no restore and go on to set the carry flag because the lost bit of the dividend is to be retrieved and used for the quotient.
31F9	NO-RSTORE	SCF	One for the quotient in B'C'CA.	
31FA	COUNT-ONE	INC JP PUSH	B M,31D2,DIV-LOOP AF	Step the loop count up by one. Loop 32 times for all bits. Save any 33rd bit for extra precision (the present carry). Trial subtract yet again for any 34th bit; the PUSH AF above saves this bit too.
		JR	Z,31E2,DIV-START	

Note: This jump is made to the wrong place. No 34th bit will ever be obtained without first shifting the dividend. Hence important results like 1/10 and 1/1000 are not rounded up as they should be. Rounding up never occurs when it depends on the 34th bit. The jump should have been to 31DB DIV-34TH above: i.e. byte 3200 hex in the ROM should read DA hex (128 decimal) instead of E1 hex (225 decimal).

LD	E,A	Now move the four bytes that form the mantissa bytes of the result from B'C'CA to D'E'DE.
LD	D,C	
EXX		
LD	E,C	Then put the 34th and 33rd bits
LD	D,B	
POP	AF	

RR	B	into 'B' to be picked up on
POP	AF	normalisation.
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
SUB	C	two exponent bytes into A and
		set the carry flag if required.
JP	313D,DIVN-EXPT	Exit via DIVN-EXPT.

THE 'INTEGER TRUNCATION TOWARDS ZERO' SUBROUTINE

(Offset 3A - see CALCULATE below: 'truncate')

This subroutine (say I(x)) returns the result of integer truncation of x, the 'last value', towards zero. Thus I(2.4) is 2 and I(-2.4) is -2. The subroutine returns at once if x is in the form of a 'short integer'. It returns zero if the exponent byte of x is less than 81 hex (ABS x is less than 1). If I(x) is a 'short integer' the subroutine returns it in that form. It returns x if the exponent byte is A0 hex or greater (x has no significant non-integral part). Otherwise the correct number of bytes of x are set to zero and, if needed, one more byte is split with a mask.

3214	truncate	LD	A,(HL)	Get the exponent byte of X into A.
		AND	A	If A is zero, return since x is
		RET	Z	already a small integer.
		CP	+81	Compare e, the exponent, to 81 hex.
		JR	NC,3221,T-GR-ZERO	Jump if e is greater than 80 hex.
		LD	(HL),+00	Else, set the exponent to zero;
		LD	A,+20	enter 32 decimal, 20 hex, into A
		JR	3272,NIL-BYTES	and jump forward to NIL-BYTES to make all the bits of
				x be zero.
3221	T-GR-ZERO	CP	+91	Compare e to 91 hex, 145 decimal.
3223		JR	NZ,323F,T-SMALL	Jump if e not 91 hex.

The next 26 bytes seem designed to test whether x is in fact -65536 decimal, i.e. 91 80 00 00 00, and if it is, to set it to 00 FF 00 00 00. This is a mistake. As already stated at byte 303B above, the Spectrum system cannot handle this number. The result here is simply to make INT (-65536) return the value -1. This is a pity, since the number would have been perfectly all right if left alone. The remedy would seem to be simply to omit the 28 bytes from 3223 above to 323E inclusive from the program.

3225		INC	HL	HL is pointed at the fourth byte
		INC	HL	of x, where the 17 bits of the
		INC	HL	integer part of x end after the
				first bit.
		LD	A,+80	The first bit is obtained in A.
		AND	(HL)	using 80 hex as a mask.
		DEC	HL	That bit and the previous 8 bits
		OR	(HL)	are tested together for zero.
		DEC	HL	HL is pointed at the second
				byte of x.
		JR	NZ,3233,T-FIRST	If already non-zero, the test can
				end.
		LD	A,+80	Otherwise, the test for -65536 is
		XOR	(HL)	now completed: 91 80 00 00 00
				will leave the zero flag set now.
3233	T-FIRST	DEC	HL	HL is pointed at the first byte of
				x.
		JR	NZ,326C,T-EXPONENT	If zero reset, the jump is made.
		LD	(HL),A	The first byte is set to zero.
		INC	HL	HL points to the second byte.

LD	(HL),+FF	The second byte is set to FF.
DEC	HL	HL again points to the first byte.
LD	A,+18	The last 24 bits are to be zero.
JR	3272,NIL-BYTES	The jump to NIL-BYTES completes the number 00 FF 00 00 00.

If the exponent byte of x is between 81 and 90 hex (129 and 144 decimal) inclusive, l(x) is a 'small integer', and will be compressed into one or two bytes. But first a test is made to see whether x is, after all, large.

323F	T-SMALL	JR	NC,326D,X-LARGE	Jump with exponent byte 92 or more (it would be better to jump with 91 too). Save STKEND in DE. Range 129 <= A <= 144 becomes 126 >= A >= 111. Range is now 15 dec >= A >= 0. Point HL at second byte. Second byte to D. Point HL at third byte. Third byte to E. Point HL at first byte again.
		PUSH	DE	
		CPL		
		ADD	A,+91	
		INC	HL	
		LD	D,(HL)	
		INC	HL	
		LD	E,(HL)	
		DEC	HL	
		DEC	HL	
		LD	C,+00	Assume a positive number.
		BIT	7,D	Now test for negative (bit 7 set).
		JR	Z,3252,T-NUMERIC	Jump if positive after all.
		DEC	C	Change the sign.
3252	T-NUMERIC	SET	7,D	Insert true numeric bit, 1, in D.
		LD	B,+08	Now test whether A >= 8 (one byte only) or two bytes needed.
		SUB	B	Leave A unchanged.
		ADD	A,B	
		JR	C,325E,T-TEST	Jump if two bytes needed.
		LD	E,D	Put the one byte into E.
		LD	D,+00	And set D to zero.
		SUB	B	Now 1 <= A <= 7 to count the shifts needed.
325E	T-TEST	JR	Z,3267,T-STORE	Jump if no shift needed.
		LD	B,A	B will count the shifts.
3261	T-SHIFT	SRL	D	Shift D and E right B times to produce the correct number.
		RR	E	
		DJNZ	3261,T-SHIFT	Loop until B is zero.
3267	T-STORE	CALL	2D8E,INT-STORE	Store the result on the stack.
		POP	DE	Restore STKEND to DE.
		RET		Finished.

Large values of x remains to be considered.

326C	T-EXPONENT	LD	A,(HL)	Get the exponent byte of x into A.
326D	X-LARGE	SUB	+A0	Subtract 160 decimal, A0 hex, from e.
		RET	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.

3272	NIL-BYTES	PUSH	DE	Save the current value of DE (STKEND).
		EX	DE,HL	Make HL point one past the fifth byte.
		DEC	HL	HL now points to the fifth byte of x.
		LD	B,A	Get the number of bits to be set to zero in B and divide it by B to give the number of whole bytes implied.
		SRL	B	
		SRL	B	
327E	BYTE-ZERO	JR	Z,3283,BITS-ZERO	Jump forward if the result is zero.
		LD	(HL),+00	Else, set the bytes to zero; B counts them.
		DEC	HL	
		DJNZ	327E,BYTE-ZERO	
3283	BITS-ZERO	AND	+07	Get A (mod 8); this is the number of bits still to be set to zero.
		JR	Z,3290,IX-END	Jump to the end if nothing more to do.
		LD	B,A	B will count the bits now.
328A	LESS-MASK	LD	A,+FF	Prepare the mask.
		SLA	A	With each loop a zero enters the mask from the right and thereby a mask of the correct length is produced.
		DJNZ	328A,LESS-MASK	
		AND	(HL)	The unwanted bits of (HL) are lost as the masking is performed.
3290	IX-END	LD	(HL),A	Return the pointer to HL.
		EX	DE,HL	Return STKEND to DE.
		POP	DE	
		RET		Finished.

THE 'RE-STACK TWO' SUBROUTINE

This subroutine is called to re-stack two 'small integers' in full five byte floating-point form for the binary operations of addition, multiplication and division. It does so by calling the following subroutine twice.

3293	RE-ST-TWO	CALL	3269,RESTK-SUB	Call the subroutine, and then continue into it for the second call.
3296	RESTK-SUB	EX	DE,HL	Exchange the pointers at each call.

THE 'RE-STACK TWO' SUBROUTINE

(Offset 3D - see CALCULATE below: 're-stack')

This subroutine is called to re-stack one number (which could be a 'small integer') in full five byte floating-point form. It is used for a single number by ARCTAN and also, through the calculator offset, by EXP, LN and 'get-argt'.

3297	RE-STACK	LD	A,(HL)	If the first byte is not zero, return - the number cannot be a 'small integer'.
		AND	A	
		RET	NZ	Save the 'other' pointer in DE.
		PUSH	DE	Fetch the sign in C and the number in DE.
		CALL	2D7F,INT-FETCH	
		XOR	A	Clear the A register.
		INC	HL	Point to the fifth location.
		LD	(HL),A	Set the fifth byte to zero.
		DEC	HL	Point to the fourth location.
		LD	(HL),A	Set the fourth byte to zero: bytes 2 and 3 will hold the mantissa.

		LD	B,+91	Set B to 145 dec for the exponent i.e. for up to 16 bits in the integer.
		LD	A,D	Test whether D is zero so that at most 8 bits would be needed.
		AND	A	Jump if more than 8 bits needed.
		JR	NZ,32B1,RS-NRMLSE	Now test E too.
		OR	E	Save the zero in B (it will give zero exponent if E too is zero).
		LD	B,D	Jump if E is indeed zero.
		JR	Z,32BD,RS-STORE	Move E to D (D was zero, E not).
		LD	D,E	Set E to zero now.
		LD	E,B	Set B to 137 dec for the exponent - no more than 8 bits now.
		LD	B,+89	Pointer to DE, number to HL.
32B1	RS-NRMLSE	EX	DE,HL	Decrement the exponent on each shift.
32B2	RSTK-LOOP	DEC	B	Shift the number right one position.
		ADD	HL,HL	Until the carry is set.
		JR	NC,32B2,RSTK-LOOP	Sign bit to carry flag now.
		RRC	C	Insert it in place as the number is shifted back one place - normal now.
		RR	H	Pointer to byte 4 back to HL.
		RR	L	Point to the third location.
		EX	DE,HL	Store the third byte.
32BD	RS-STORE	DEC	HL	Point to the second location.
		LD	(HL),E	Store the second byte.
		DEC	HL	Point to the first location.
		LD	(HL),D	Store the exponent byte.
		DEC	HL	Restore the 'other' pointer to DE.
		LD	(HL),B	DE.
		POP	DE	Finished.
		RET		

THE FLOATING-POINT CALCULATOR

THE TABLE OF CONSTANTS

This first table holds the five useful and frequently needed numbers zero, one, a half, a half of pi and ten. The numbers are held in a condensed form which is expanded by the STACK LITERALS subroutine, see below, to give the required floating-point form.

		data:	constant	when expanded gives: exp. mantissa: (Hex.)
32C5	stk-zero	DEFB +00 DEFB +B0 DEFB +00	zero	00 00 00 00 00
32C8	stk-one	DEFB +40 DEFB +B0 DEFB +00 DEFB +01	one	00 00 01 00 00
32CC	stk-half	DEFB +30 DEFB +00	a half	80 00 00 00 00
32CE	stk-pi/2	DEFB +F1 DEFB +49 DEFB +0F DEFB +DA DEFB +A2	a half of pi	81 49 0F DA A2
32D3	stk-ten	DEFB +40 DEFB +B0 DEFB +00 DEFB +0A	ten	00 00 0A 00 00

THE TABLE OF ADDRESSES:

This second table is a look-up table of the addresses of the sixty-six operational subroutines of the calculator. The offsets used to index into the table are derived either from the operation codes used in SCANNING, see 2734, etc., or from the literals that follow a RST 0028 instruction.

offset	label	address	offset	label	address
32D7 00	jump-true	8F 36	3319 21	tan	DA 37
32D9 01	exchange	3C 34	331B 22	asn	33 38
32DB 02	delete	A1 33	331D 23	acs	43 38
32DD 03	subtract	0F 30	331F 24	atn	E2 37
32DF 04	multiply	CA 30	3321 25	ln	13 37
32E1 05	division	AF 31	3323 26	exp	C4 36
32E3 06	to-power	51 38	3325 27	int	AF 36
32E5 07	or	1B 35	3327 28	sqr	4A 38
32E7 08	no-&-no	24 35	3329 29	sgn	92 34
32E9 09	no-l-eql	3B 35	332B 2A	abs	6A 34

32EB	0A	no-gr-eq	3B	332D	2B	peek	AC
			35				34
32ED	0B	nos-neql	3B	332F	2C	in	A5
			35				34
32EF	0C	no-grtr	3B	3331	2D	usr-no	B3
			35				34
32F1	0D	no-less	3B	3333	2E	str\$	1F
			35				36
32F3	0E	nos-eql	3B	3335	2F	chr\$	C9
			35				35
32F5	0F	addition	14	3337	30	not	01
			30				35
32F7	10	str-&-no	2D	3339	31	duplicate	C0
			35				33
32F9	11	str-l-eql	3B	333B	32	n-mod-m	A0
			35				36
32FB	12	str-gr-eq	3B	333D	33	jump	86
			35				36
32FD	13	strs-neql	3B	333F	34	stk-data	C6
			35				33
32FF	14	str-grtr	3B	3341	35	dec-jr-nz	7A
			35				36
3301	15	str-less	3B	3343	36	less-0	06
			35				35
3303	16	strs-eql	3B	3345	37	greater-0	F9
			35				34
3305	17	strs-add	9C	3347	38	end-calc	9B
			35				36
3307	18	val\$	DE	3349	39	get-argt	83
			35				37
3309	19	usr-\$	BC	334B	3A	truncate	14
			34				32
330B	1A	read-in	45	334D	3B	fp-calc-2	A2
			36				33
330D	1B	negate	6E	334F	3C	e-to-fp	4F
			34				2D
330F	1C	code	69	3351	3D	re-stack	97
			36				32
3311	1D	val	DE	3353	3E	series-06	49
			45			etc.	34
3313	1E	len	74	3355	3F	stk-zero	1B
			36			etc.	34
3315	1F	sin	B5	3357	40	st-mem-0	2D
			37			etc.	34
3317	20	cos	AA	3359	41	get-mem-0	0F
			37			etc.	34

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

Offset 3E: series-06, series-08, & series-0C; literals 86,88 & 8C.

Offset 3F: stk-zero, stk-one, stk-half, stk-pi/2 & stk-ten; literals A0 to A4.

Offset 40: st-mem-0, st-mem-1, st-mem-2, st-mem-3, st-mem-4 & st-mem-5;
literals C0 to C5.

Offset 41: get-mem-0, get-mem-1, get-mem-2, get-mem-3, get-mem-4 & get-mem-5;
literals E0 to E5.

THE 'CALCULATE' SUBROUTINE

This subroutine is used to perform floating-point calculations. These can be considered to be of three types:

- i. Binary operations, e.g. addition, where two numbers in floating-point form are added together to give one 'last value'.
- ii. Unary operations, e.g. sin, where the 'last value' is changed to give the appropriate function result as a new 'last value'.
- iii. Manipulatory operations, e.g. st-mem-0, where the 'last value' is copied to the first five bytes of the calculator's memory area.

The operations to be performed are specified as a series of data-bytes, the literals, that follow an RST 0028 instruction that calls this subroutine. The last literal in the list is always '38' which leads to an end to the whole operation.

In the case of a single operation needing to be performed, the operation offset can be passed to the CALCULATOR in the B register, and operation '3B', the SINGLE CALCULATION operation, performed.

It is also possible to call this subroutine recursively, i.e. from within itself, and in such a case it is possible to use the system variable BREG as a counter that controls how many operations are performed before returning.

The first part of this subroutine is complicated but essentially it performs the two tasks of setting the registers to hold their required values, and to produce an offset, and possibly a parameter, from the literal that is currently being considered.

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.

335B	CALCULATE	CALL	35BF,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).
335E	GEN-ENT-1	LD LD	A,B (BREG),A	Either, transfer a single operation offset to BREG temporarily, or, when using the subroutine recursively pass the parameter to BREG to be used as a counter.
3362	GEN-ENT-2	EXX EX EXX	(SP),HL	The return address of the subroutine is store in H'L'. This saves the pointer to the first literal. Entering the CALCULATOR at GEN-ENT-2 is used whenever BREG is in use as a counter and is not to be disturbed.
3365	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 to STKEND.
		EXX LD	A,(HL)	Go to the alternate register set, and fetch the literal for this loop.

		INC	HL	Make H'L' point to the next literal.
336C	SCAN-ENT	PUSH	HL	This pointer is saved briefly on the machine stack. SCAN-ENT is used by the SINGLE CALCULATION subroutine to find the subroutine that is required. Test the A register.
		AND JP	A P,3380,FIRST-3D	Separate the simple literals from the multi-purpose literals. Jump with literals 00 - 3D.
		LD AND RRCA RRCA RRCA ADD LD	D,A +60 A,+7C L,A	Save the literal in D. Continue only with bits 5 & 6. Four right shifts make them now bits 1 & 2.
		LD AND JR	A,D +1F 338E,ENT-TABLE	The offsets required are 3E-41. and L will now hold double the required offset.
3380	FIRST-3D	CP JR EXX LD LD LD ADD EXX RLCA LD	+18 NC,338C,DOUBLE-A BC, +FFFB D,H E,L HL,BC L,A	Now produce the parameter by taking bits 0,1,2,3 & 4 of the literal; keep the parameter in A. Jump forward to find the address of the required subroutine.
338C	DOUBLE-A			Jump forward if performing a unary operation.
338E	ENT-TABLE	LD LD ADD LD INC LD LD EX PUSH EXX LD	DE,+32D7 H,+00 HL,DE E,(HL) HL D,(HL) HL,+3365 (SP),HL DE BC,(STKEND-hi)	All of the subroutines that perform binary operations require that HL points to the first operand and DE points to the second operand (the 'last value') as they appear on the calculator stack. As each entry in the table of addresses takes up two bytes the offset produced is doubled. The base address of the table. The address of the required table entry is formed in HL; and the required subroutine address is loaded into the DE register pair.
				The RE-ENTRY address of 3365 is put on the machine stack underneath the subroutine address.
				Return to the main set of registers.
				The current value of BREG is transferred to the B register thereby returning the single operation offset.
33A1	delete	RET		(See COMPARISON at 353B) An indirect jump to the required subroutine.

THE 'DELETE' SUBROUTINE

(Offset 02: 'delete')

This subroutine contains only the single RET instruction at 33A1, above. The literal '02' results in this subroutine being considered as a binary operation that is to be entered with a first number addressed by the HL register pair and a second number addressed by the DE register pair, and the result produced again addressed by the HL register pair.

The single RET instruction thereby leads to the first number being considered as the resulting 'last value' and the second number considered as being deleted. Of course the number has not been deleted from the memory but remains inactive and will probably soon be overwritten.

THE 'SINGLE OPERATION' SUBROUTINE

(Offset 3B: 'fp-calc-2')

This subroutine is only called from SCANNING at 2757 hex and is used to perform a single arithmetic operation. The offset that specifies which operation is to be performed is supplied to the calculator in the B register and subsequently transferred to the system variable BREG.

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

33A2	fp-calc-2	POP	AF	Discard the RE-ENTRY address.
		LD	A,(BREG)	Transfer the offset to A.
		EXX		Enter the alternate register set.
		JR	336C,SCAN-ENT	Jump back to find the required address; stack the RE-ENTRY address and jump to the subroutine for the operation.

THE 'TEST 5-SPACES' SUBROUTINE

This subroutine tests whether there is sufficient room in memory for another 5-byte floating-point number to be added to the calculator stack.

33A9	TEST-5-SP	PUSH	DE	Save DE briefly.
		PUSH	HL	Save HL briefly.
		LD	BC,+0005	Specify the test is for 5 bytes.
		CALL	1F05,TEST-ROOM	Make the test.
		POP	HL	Restore HL.
		POP	DE	Restore DE.
		RET		Finished.

THE 'STACK NUMBER' SUBROUTINE

This subroutine is called by BEEP and SCANNING twice to copy STKEND to DE, move a floating-point number to the calculator stack, and reset STKEND from DE. It calls 'MOVE-FP' to do the actual move.

33B4	STACK-NUM	LD	DE,(STKEND)	Copy STKEND to DE as destination address.
		CALL	33C0,MOVE-FP	Move the number.
		LD	(STKEND),DE	Reset STKEND from DE.
		RET		Finished.

THE 'MOVE A FLOATING-POINT NUMBER' SUBROUTINE

(Offset 31: 'duplicate')

This subroutine moves a floating-point number to the top of the calculator stack (3 cases) or from the top of the stack to the calculator's memory area (1 case). It is also called through the calculator when it simply duplicates the number at the top of the calculator stack, the 'last value', thereby extending the stack by five bytes.

33C0	MOVE-FP	CALL	33A9,TEST-5-SP	A test is made for room.
		LDIR		Move the five bytes involved.
		RET		Finished.

THE 'STACK LITERALS' SUBROUTINE

(Offset 34: 'stk-data')

This subroutine places on the calculator stack, as a 'last value', the floating-point number supplied to it as 2, 3, 4 or 5 literals. When called by using offset '34' the literals follow the '34' in the list of literals; when called by the SERIES GENERATOR, see below, the literals are supplied by the sub-routine that called for a series to be generated; and when called by SKIP CONSTANTS & STACK A CONSTANT the literals are obtained from the calculator's table of constants (32C5-32D6). In each case, the first literal supplied is divided by Hex.40, and the integer quotient plus 1 determines whether 1, 2, 3 or 4 further literals will be taken from the source to form the mantissa of the number. Any unfilled bytes of the five bytes that go to form a 5-byte floating-point number are set to zero. The first literal is also used to determine the exponent, after reducing mod Hex.40, unless the remainder is zero, in which case the second literal is used, as it stands, without reducing mod Hex.40. In either case, Hex.50 is added to the literal, giving the augmented exponent byte, e (the true exponent e' plus Hex.80). The rest of the 5 bytes are stacked, including any zeros needed, and the subroutine returns.

33C6	STK-DATA	LD	H,D	This subroutine performs the 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	
33C8	STK-CONST	CALL	33A9,TEST-5-SP	Now test that there is indeed room.
		EXX		Go to the alternate register set
		PUSH	HL	and stack the pointer to the next literal.
		EXX		Switch over the result pointer and the next literal pointer.
		EX	(SP),HL	Save BC briefly.
		PUSH	BC	The first literal is put into A
		LD	A,(HL)	and divided by Hex.40 to give the integer values 0, 1, 2 or 3.
		AND	+C0	
		RLCA		
		RLCA		
		LD	C,A	The integer value is transferred to C and incremented, thereby giving the range 1, 2, 3 or 4 for the number of literals that will be needed.
		INC	C	
		LD	A,(HL)	The literal is fetch anew,
		AND	+3F	reduced mod Hex.40 and discarded as inappropriate if the remainder is zero; in which case
		JR	NZ,33DE,FORM-EXP	
		INC	HL	

		LD	A,(HL)	the next literal is fetched and used unreduced.
33DE	FORM-EXP	ADD LD	A,+50 (DE),A	The exponent, e, is formed by the addition of Hex.50 and passed to the calculator stack as the first of the five bytes of the result.
		LD SUB INC INC LD LDIR POP EX EXX POP EXX LD XOR DEC RET LD INC JR	A,+05 C HL DE B,+00 BC (SP),HL HL B,A A B Z (DE),A DE 33F1, STK-ZEROS	The number of literals specified in C are taken from the source and entered into the bytes of the result. Restore BC. Return the result pointer to HL and the next literal pointer to its usual position in H' & L'. The number of zero bytes required at this stage is given by 5-C-1; and this number of zeros is added to the result to make up the required five bytes.
33F1	STK-ZEROS			

THE 'SKIP CONSTANTS' SUBROUTINE

This subroutine is entered with the HL register pair holding the base address of the calculator's table of constants and the A register holding a parameter that shows which of the five constants is being requested.

The subroutine performs the null operations of loading the five bytes of each unwanted constant into the locations 0000, 0001, 0002, 0003 and 0004 at the beginning of the ROM until the requested constant is reached.

The subroutine returns with the HL register pair holding the base address of the requested constant within the table of constants.

33F7	SKIP-CONS	AND	A	The subroutine returns if the parameter is zero, or when the requested constant has been reached.
33F8	SKIP-NEXT	RET	Z	
		PUSH PUSH LD CALL	AF DE DE,+0000 33C8,STK-CONST	Save the parameter. Save the result pointer. The dummy address. Perform imaginary stacking of an expanded constant.
		POP POP DEC JR	DE AF A 33F8,SKIP-NEXT	Restore the result pointer. Restore the parameter. Count the loops. Jump back to consider the value of the counter.

THE 'MEMORY LOCATION' SUBROUTINE

This subroutine finds the base address for each five byte portion of the calculator's memory area to or from which a floating-point number is to be moved from or to the calculator stack. It does this operation by adding five times the parameter supplied to the base address for the area which is held in the HL register pair.

Note that when a FOR-NEXT variable is being handled then the pointers are changed so that the variable is treated as if it were the calculator's memory area (see address 1D20).

3406	LOC-MEM	LD RLCA RLCA ADD	C,A A,C	Copy the parameter to C. Double the parameter. Double the result. Add the value of the parameter to give five times the original
------	---------	---------------------------	--------------------	---

LD	C,A	value.
LD	B,+00	This result is wanted in the
ADD	HL,BC	BC register pair.
RET		Produce the new base address.
		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'.

340F	get-mem-0 etc.	PUSH LD	DE HL,(MEM)	Save the result pointer. Fetch the pointer to the current memory area (see above).
		CALL	3406,LOC-MEM	The base address is found.
		CALL	33C0,MOVE-FP	The five bytes are moved.
		POP	HL	Set the result pointer.
		RET		Finished.

THE 'STACK A CONSTANT' SUBROUTINE

(offsets A0 to A4: 'stk-zero','stk-one','stk-half','stk-pi/2' & 'stk-ten')

This subroutine uses SKIP CONSTANTS to find the base address of the requested constants from the calculator's table of constants and then calls STACK LITERALS, entering at STK-CONST, to make the expanded form of the constant the 'last value' on the calculator stack.

341B	stk-zero etc.	LD	H,D	Set HL to hold the result pointer.
		LD	L,E	
		EXX		Go to the alternate register set
		PUSH	HL	and save the next literal pointer.
		LD	HL,+32C5	The base address of the calcul- ator's table of constants.
		EXX		Back to the main set of registers.
		CALL	33F7,SKIP-CONS	Find the requested base address.
		CALL	33C8,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.

342D	st-mem-0 etc.	PUSH	HL	Save the result pointer.
		EX	DE,HL	Source to DE briefly.
		LD	HL,(MEM)	Fetch the pointer to the current memory area.
		CALL	3406,LOC-MEM	The base address is found.
		EX	DE,HL	Exchange source and destination pointers.
		CALL	33C0,MOVE-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.

343C	EXCHANGE	LD	B,+05	There are five bytes involved.
343E	SWAP-BYTE	LD	A,(DE)	Each byte of the second number.
		LD	C,(HL)	Each byte of the first number.
		EX	DE,HL	Switch source and destination.
		LD	(DE),A	Now to the first number.
		LD	(HL),C	Now to the second number
		INC	HL	Move to consider the next pair
		INC	DE	of bytes.
		DJNZ	343E,SWAP-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 $n=1,2,\dots$, by the recurrence relation:

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

The series in fact generates:

$T_0, 2T_1, 2T_2, \dots, 2T_{n-1}$, where n is 6 for SIN, 8 for EXP and 12 decimal, for LN and ATN.

The coefficients of the powers of z in these polynomials may be found in the Handbook of **Mathematical Functions by M. Abramowitz and I.A. Stegun** (Dover 1965), page 795.

BASIC programs showing the generation of each of the four functions are given here in the Appendix.

In simple terms this subroutine is called with the 'last value' on the calculator stack, say Z , being a number that bears a simple relationship to the argument, say X , when the task is to evaluate, for instance, $\sin X$. The calling subroutine also supplies the list of constants that are to be required (six constants for SIN). The SERIES GENERATOR then manipulates its data and returns to the calling routine a 'last value' that bears a simple relationship to the requested function, for instance, $\sin X$.

This subroutine can be considered to have four major parts:

i. The setting of the loop counter:

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

3449	series-06	LD	B,A	Move the parameter to B.
	etc.	CALL	335E,GEN-ENT-1	In effect a RST 0028
				instruction but sets the counter.

ii. The handling of the 'last value', Z :

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

DEFB	+31,duplicate	Z,Z	
DEFB	+0F,addition	2*Z	
DEFB	+C0,st-mem-0	2*Z	mem-0 holds 2*Z
DEFB	+02,delete	-	

DEFB	+A0,stk-zero	0	
DEFB	+C2,st-mem-2	0	mem-2 holds 0

iii. The main loop:

The series is generated by looping, using BREG as a counter; the constants in the calling subroutine are stacked in turn by calling STK-DATA; the calculator is re-entered at GEN-ENT-2 so as not to disturb the value of BREG; and the series is built up in the form: $B(R) = 2*Z*B(R-1) - B(R-2) + A(R)$, for $R = 1, 2, \dots, N$, where $A(1), A(2), \dots, A(N)$ 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.

3453	G-LOOP	DEFB	+31,duplicate	B(R),B(R)
		DEFB	+E0,get-mem-0	B(R),B(R), $2*Z$
		DEFB	+04,multiply	B(R), $2*B(R)*Z$
		DEFB	+E2,get-mem-2	B(R), $2*B(R)*Z$,B(R-1)
		DEFB	+C1,st-mem-1	mem-1 holds B(R-1)
DEFB	+38,end-calc	DEFB	+03,subtract	B(R), $2*B(R)*Z-B(R-1)$

The next constant is placed on the calculator stack.

CALL	33C6,STK-DATA	B(R), $2*B(R)*Z-B(R-1)$,A(R+1)
------	---------------	---------------------------------

The Calculator is re-entered without disturbing BREG.

CALL	3362,GEN-ENT-2	
DEFB	+0F,addition	B(R), $2*B(R)*Z-B(R-1)+A(R+1)$
DEFB	+01,exchange	$2*B(R)*Z-B(R-1)+A(R+1)$,B(R)
DEFB	+C2,st-mem-2	mem-2 holds B(R)
DEFB	+02,delete	$2*B(R)*Z-B(R-1)+A(R+1) =$
		B(R!1)
DEFB	+35,dec-jr-nz	B(R+1)
DEFB	+EE,to 3453,G-LOOP	

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	B(N),B(N-2)
DEFB	+03,subtract	B(N)-B(N-2)
DEFB	+38,end-calc	
RET		Finished

THE 'ABSOLUTE MAGNITUDE' FUNCTION

(Offset 2A: 'abs')

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

'Small integers' have to be treated separately. Most of the work is shared with the 'unary minus' operation.

346A	abs	LD	B,+FF	B is set to FF hex.
		JR	3474,NEG-TEST	The jump is made into 'unary minus'.

THE 'UNARY MINUS' OPERATION

(Offset 1B: 'negate')

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

Zero is simply returned unchanged. Full five byte floating-point numbers have their sign bit manipulated so that it ends up reset (for 'abs') or changed (for 'negate'). 'Small integers' have their sign byte set to zero (for 'abs') or changed (for 'negate').

346E	NEGATE	CALL	34E9,TEST-ZERO	If the number is zero, the
		RET	C	subroutine returns leaving
				00 00 00 00 00 unchanged.

		LD	B,+00	B is set to +00 hex for 'negate'.
 'ABS' enters here.				
3474	NEG-TEST	LD	A,(HL)	If the first byte is zero, the jump is made to deal with a 'small integer'. Point to the second byte. Get +FF for 'abs', +00 for 'negate'. Now +80 for 'abs', +00 for 'negate'. This sets bit 7 for 'abs', but changes nothing for 'negate'. Now bit 7 is changed, leading to bit 7 of byte 2 reset for 'abs', and simply changed for 'negate'. The new second byte is stored. HL points to the first byte again. Finished.
		AND	A	
		JR	Z,3483,INT-CASE	
		INC	HL	
		LD	A,B	
		AND	+80	
		OR	(HL)	
		RLA		
		CCF		
		RRA		
		LD	(HL),A	
		DEC	HL	
		RET		

The 'integer case' does a similar operation with the sign byte.

3483	INT-CASE	PUSH	DE	Save STKEND in DE. Save pointer to the number in HL. Fetch the sign in C, the number in DE. Restore the pointer to the number in HL. Get +FF for 'abs', +00 for 'negate'. Now +FF for 'abs', no change for 'negate'. Now +00 for 'abs', and a changed byte for 'negate': store it in C. Store result on the stack. Return STKEND to DE.
		PUSH	HL	
		CALL	2D7F,INT-FETCH	
		POP	HL	
		LD	A,B	
		OR	C	
		CPL		
		LD	C,A	
		CALL	2D8E,INT-STORE	
		POP	DE	
		RET		

THE 'SIGNUM' FUNCTION

(Offset 29: 'sgn')

This subroutine handles the function SGN X and therefore returns a 'last value' of 1 if X is positive, zero if X is zero and -1 if X is negative.

3492	sgn	CALL 34E9,TEST-ZERO	If X is zero, just return with zero as the 'last value'. Save the pointer to STKEND. Store 1 in DE. Point to the second byte of X. Rotate bit 7 into the carry flag. Point to the destination again. Set C to zero for positive X and to FF hex for negative X. Stack 1 or -1 as required. Restore the pointer to STKEND. Finished.
		RET	
		PUSH	
		LD	
		INC	
		RL	
		DEC	
		SBC	
		LD	
		CALL	
		POP	
		RET	

THE 'IN' FUNCTION

(Offset 2C: 'in')

This subroutine handles the function IN X. It inputs at processor level from port X, loading BC with X and performing the instruction IN A,(C).

34A5	in	CALL	1E99,FIND-INT2	The 'last value', X, is compressed into BC.
		IN	A,(C)	The signal is received.
		JR	34B0,IN-PK-STK	Jump to stack the result.

THE 'PEEK' FUNCTION

(Offset 2B: 'peek')

This subroutine handles the function PEEK X. The 'last value' is unstacked by calling FIND-INT2 and replaced by the value of the contents of the required location.

34AC	peek	CALL	1E99,FIND-INT2	Evaluate the 'last value', rounded to the nearest integer; test that it is in range and return it in BC.
34B0	IN-PK-STK	LD JP	A,(BC) 2D28,STACK-A	Fetch the required byte. Exit by jumping to STACK-A.

THE 'USR' FUNCTION

(Offset 2D: 'usr-no')

This subroutine ('USR number' as distinct from 'USR string') handles the function USR X, where X is a number. The value of X is obtained in BC, a return address is stacked and the machine code is executed from location X.

34B3	usr-no	CALL	1E99,FIND-INT2	Evaluate the 'last value', rounded to the nearest integer; test that it is in range and return it in BC.
		LD	HL,+2D2B	Make the return address be that of the subroutine STACK-BC.
		PUSH	HL	Make an indirect jump to the required location.
		PUSH	BC	
		RET		

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. For a successful return to BASIC, H'L' must on exit from the machine code contain the address in SCANNING of the 'end-calc' instruction, 2758 hex (10072 decimal).

THE 'USR-STRING' FUNCTION

(Offset 19: 'usr-\$')

This subroutine handles the function USR X\$, where X\$ is a string. The subroutine returns in BC the address of the bit pattern for the user-defined graphic corresponding to X\$. It reports error A if X\$ is not a single letter between a and u or a user-defined graphic.

34BC	usr-\$	CALL	2BF1,STK-FETCH	Fetch the parameters of the string X\$.
		DEC	BC	Decrease the length by 1 to test it.
		LD	A,B	If the length was not 1, then
		OR	C	jump to give error report A.
		JR	NZ,34E7,REPORT-A	
		LD	A,(DE)	Fetch the single code of the string.
		CALL	2C8D,ALPHA	Does it denote a letter?
		JR	C,34D3,USR-RANGE	If so, jump to gets its address.
		SUB	+90	Reduce range for actual user-defined graphics to 0 - 20 decimal.
		JR	C,34E7,REPORT-A	Give report A if out of range.
		CP	+15	Test the range again.
		JR	NC,34E7,REPORT-A	Give report A if out of range.
		INC	A	Make range of user-defined graphics 1 to 21 decimal, as for a to u.

34D3	USR-RANGE	DEC	A	Now make the range 0 to 20 decimal in each case. Multiply by 8 to get an offset for the address.
		ADD	A,A	
		ADD	A,A	
		ADD	A,A	
		CP	+A8	Test the range of the offset.
		JR	NC,34E7,REPORT-A	Give report A if out of range.
		LD	BC,(UDG)	Fetch the address of the first user-defined graphic in BC.
		ADD	A,C	Add C to the offset.
		LD	C,A	Store the result back in C.
		JR	NC,34E4,USR-STACK	Jump if there is no carry.
		INC	B	Increment B to complete the address.
34E4	USR-STACK	JP	2D2B,STACK-BC	Jump to stack the address.
REPORT A - Invalid argument.				
34E7	REPORT-A	RST	0008,ERROR-1	Call the error handling routine.
		DEFB	+09	

THE 'TEST-ZERO' SUBROUTINE

This subroutine is called at least nine times to test whether a floating-point number is zero. This test requires that the first four bytes of the number should each be zero. The subroutine returns with the carry flag set if the number was in fact zero.

34E9	TEST-ZERO	PUSH	HL	Save HL on the stack.
		PUSH	BC	Save BC on the stack.
		LD	B,A	Save the value of A in B.
		LD	A,(HL)	Get the first byte.
		INC	HL	Point to the second byte.
		OR	(HL)	OR first byte with second.
		INC	HL	Point to the third byte.
		OR	(HL)	OR the result with the third byte.
		INC	HL	Point to the fourth byte.
		OR	(HL)	OR the result with the fourth byte.
		LD	A,B	Restore the original value of A.
		POP	BC	And of BC.
		POP	HL	Restore the pointer to the number to HL.
		RET	NZ	Return with carry reset if any of the four bytes was non-zero.
		SCF		Set the carry flag to indicate that the number was zero, and return.
		RET		

THE 'GREATER THAN ZERO' OPERATION

(Offset 37: 'greater-0')

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

34F9	GREATER-0	CALL	34E9,TEST-ZERO	Is the 'last-value' zero?
		RET	C	If so, return.
		LD	A,+FF	Jump forward to LESS THAN
		JR	3507,SIGN-TO-C	ZERO but signal the opposite action is needed.

THE 'NOT' FUNCTION

(Offset 30: 'not')

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

3501	NOT	CALL	34E9,TEST-ZERO	The carry flag will be set only if the 'last value' is zero; this gives the correct result. Jump forward.
		JR	350B,FP-0/1	

THE 'LESS THAN ZERO' OPERATION

(Offset 36: 'less-0')

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

3506	less-0	XOR	A	Clear the A register. Point to the sign byte. The carry is reset for a positive number and set for a negative number; when entered from GREATER-0 the opposite sign goes to the carry.
3507	SIGN-TO-C	INC	HL	
		XOR	(HL)	
		DEC	HL	
		RLCA		

THE 'ZERO OR ONE' SUBROUTINE

This subroutine sets the 'last value' to zero if the carry flag is reset and to one if it is set. When called from 'E-TO-FP' however it creates the zero or one not on the stack but in mem-0.

350B	FP-0/1	PUSH	HL	Save the result pointer. Clear A without disturbing the carry. Set the first byte to zero. Point to the second byte. Set the second byte to zero. Point to the third byte. Rotate the carry into A, making A one if the carry was set, but zero if the carry was reset. Set the third byte to one or zero. Ensure that A is zero again. Point to the fourth byte. Set the fourth byte to zero. Point to the fifth byte. Set the fifth byte to zero. Restore the result pointer.
		LD	A,+00	
		LD	(HL),A	
		INC	HL	
		LD	(HL),A	
		INC	HL	
		RLA		
		LD	(HL),A	
		RRA		
		INC	HL	
		LD	(HL),A	
		INC	HL	
		LD	(HL),A	
		POP	HL	
		RET		

THE 'OR' OPERATION

(Offset 07: 'or')

This subroutine performs the binary operation 'X OR Y' and returns X if Y is zero and the value 1 otherwise.

351B	or	EX	DE,HL	Point HL at Y, the second number. Test whether Y is zero. Restore the pointers. Return if Y was zero; X is now the 'last value'. Set the carry flag and jump back to set the 'last value' to 1.
		CALL	34E9,TEST-ZERO	
		EX	DE,HL	
		RET	C	
		SCF		
		JR	350B,FP-0/1	

THE 'NUMBER AND NUMBER' OPERATION

(Offset 08: 'no-&-no')

This subroutine performs the binary operation 'X AND Y' and returns X if Y is non-zero and the value zero otherwise.

3524	no-&-no	EX CALL EX RET	DE,HL 34E9,TEST-ZERO DE,HL NC	Point HL at Y, DE at X. Test whether Y is zero. Swap the pointers back. Return with X as the 'last value' if Y was non-zero.
		AND JR	A 350B,FP-0/1	Reset the carry flag and jump back to set the 'last value' to zero.

THE 'STRING AND NUMBER' OPERATION

(Offset 10: 'str-&-no')

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

352D	str-&-no	EX CALL EX RET	DE,HL 34E9,TEST-ZERO DE,HL NC	Point HL at Y, DE at X\$ Test whether Y is zero. Swap the pointers back. Return with X\$ as the 'last value' if Y was non-zero.
		PUSH DEC	DE DE	Save the pointer to the number. Point to the fifth byte of the string parameters i.e. length-high.
		XOR LD DEC LD POP RET	A (DE),A DE (DE),A DE	Clear the A register. Length-high is now set to zero. Point to length-low. Length-low is now set to zero. Restore the pointer. 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-eq', '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.

353B	no-l-eql etc.	LD SUB	A,B +08	The single offset goes to the A register. The range is now 01-06 & 09-0E.
		BIT JR DEC	2,A NZ,3543,EX-OR-NOT A	This range is changed to: 00-02, 04-06, 08-0A & 0C-0E.
3543	EX-OR-NOT	RRCA		Then reduced to 00-07 with carry set for 'greater than or equal to' & 'less than'; the operations with carry set are then treated as their complementary operation once their values have been exchanged.
		JR PUSH PUSH CALL POP EX POP BIT JR	NC,354E,NU-OR-STR AF HL 343C,EXCHANGE DE DE,HL AF 2,A NZ,3559,STRINGS	
354E	NU-OR-STR	RRCA		The numerical comparisons are now separated from the string comparisons by testing bit 2.
		PUSH CALL	AF 300F,SUBTRACT	The numerical operations now have the range 00-01 with carry set for 'equal' and 'not equal'. Save the offset. The numbers are subtracted for

3559	STRINGS	JR RRCA	358C,END-TESTS	the final tests. The string comparisons now have the range 02-03 with carry set for 'equal' and 'not equal'. Save the offset. The lengths and starting addresses of the strings are fetched from the calculator stack. The length of the second string.
3564	BYTE-COMP	PUSH CALL PUSH PUSH CALL POP LD OR EX LD JR OR POP	AF 2BF1,STK-FETCH DE BC 2BF1,STK-FETCH HL A,H L (SP),HL A,B NZ,3575,SEC-PLUS C BC	
356B	SECND-LOW			Jump unless the second string is null. Here the second string is either null or less than the first.
3572	BOTH-NULL	JR POP CCF JR POP	Z,3572,BOTH-NULL AF 3588,STR-TEST AF	The carry is complemented to give the correct test results. Here the carry is used as it stands.
3575	SEC-PLUS	JR OR JR	3588,STR-TEST C Z,3585,FRST-LESS	
3585	FRST-LESS	LD SUB JR JR DEC INC INC EX DEC JR POP POP AND	A,(DE) (HL) C,3585,FRST-LESS NZ,356B,SECND-LOW BC DE HL (SP),HL HL 3564,BYTE-COMP BC AF A	The first string is now null, the second not. Neither string is null, so their next bytes are compared. The first byte is less. The second byte is less. The bytes are equal; so the lengths are decremented and a jump is made to BYTE-COMP to compare the next bytes of the reduced strings.
3588	STR-TEST	PUSH RST DEFB DEFB	AF 0028,FP-CALC +A0,stk-zero +38,end-calc	The carry is cleared here for the correct test results. For the string tests, a zero is put on to the calculator stack.
358C	END-TESTS	POP PUSH CALL POP PUSH CALL POP RRCA CALL RET	AF AF C,3501,NOT AF AF NC,34F9,GREATER-0 AF NC,3501,NOT	These three tests, called as needed, give the correct results for all twelve comparisons. The initial carry is set for 'not equal' and 'equal', and the final carry is set for 'greater than', 'less than' and 'equal'. 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.

359C	strs-add	CALL	2BF1,STK-FETCH	The parameters of the second string are fetched and saved.
		PUSH	DE	
		PUSH	BC	
		CALL	2BF1,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 are saved.
		PUSH	BC	The total length of the two strings is calculated and passed to BC.
		ADD	HL,BC	
		LD	B,H	
		LD	C,L	
		RST	0030,BC-SPACES	Sufficient room is made available.
		CALL	2AB2,STK-STORE	The parameters of the new string are passed to the calculator stack.
		POP	BC	The parameters of the first string are retrieved and the string copied to the work space as long as it is not a null string.
35B7	OTHER-STR	POP	HL	
		LD	A,B	
		OR	C	
		JR	Z,35B7,OTHER-STR	
		LDIR		
		POP	BC	Exactly the same procedure is followed for the second string thereby giving 'A\$+B\$'.
		POP	HL	
		LD	A,B	
		OR	C	
		JR	Z,35BF,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.

35BF	STK-PNTRS	LD	HL,(STKEND)	Fetch the current value of STKEND.
		LD	DE,+FFFB	Set DE to -5, twos complement.
		PUSH	HL	Stack the value for STKEND.
		ADD	HL,DE	Calculate STKEND-5.
		POP	DE	DE now holds STKEND and HL
		RET		

THE 'CHR\$' FUNCTION

(Offset 2F: 'chrs')

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

35C9	chrs	CALL	2DD5,FP-TO-A	The 'last value' is compressed into the A register.
		JR	C,35DC,REPORT-B	Give the error report if X was greater than 255 decimal, or X was a negative number.
		JR	NZ,35DC,REPORT-B	
		PUSH	AF	Save the compressed value of X.
		LD	BC,+0001	Make one space available in the
		POP	AF	Fetch the value.
		LD	(DE),A	Copy the value to the work space.
		CALL	2AB2,STK-STORE	Pass the parameters of the new string to the calculator stack.
		EX	DE,HL	Reset the pointers.
		RET		Finished.

REPORT-B - Integer out of range

35DC REPORT-B RST 0008,ERROR-1
 DEFB +0A

Call the error handling routine.

THE 'VAL' AND 'VAL\$' FUNCTION

(Offsets 1D: 'val' and 18: 'val\$')

This subroutine handles the functions VAL X\$ and VAL\$ X\$. When handling VAL X\$, it return a 'last value' that is the result of evaluating the string (without its bounding quotes) as a numerical expression. when handling VAL\$ X\$, it evaluates X\$ (without its bounding quotes) as a string expression, and returns the parameters of that string expression as a 'last value' on the calculator stack.

35DE	val (also val\$)	LD PUSH LD	HL,(CH-ADD) HL A,B	The current value of CH-ADD is preserved on the machine stack. The 'offset' for 'val' or 'val\$' must be in the B register; it is now copied to A.
		ADD	A,+E3	Produce +00 and carry set for 'val', +FB and carry reset for 'val\$'.
		SBC	A,A	Produce +FF (bit 6 therefore set) for 'val', but +00 (bit 6 reset) for 'val\$'.
		PUSH	AF	Save this 'flag' on the machine stack.
		CALL PUSH INC RST	2BF1,STK-FETCH DE BC 0030,BC-SPACES	The parameters of the string are fetched; the starting address is saved; one byte is added to the length and room made 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 PUSH	(CH-ADD),DE DE	The pointer to the first new space goes to CH-ADD and to the machine stack.
		LDIR		The string is copied to the work space, together with an extra byte.
		EX DEC LD RES CALL	DE,HL HL (HL),+0D 7,(FLAGS) 24FB,SCANNING	Switch the pointers. The extra byte is replaced by a 'carriage return' character. The syntax flag is reset and the string is scanned for correct syntax.
		RST	0018,GET-CHAR	The character after the string is fetched.
		CP	+0D	A check is made that the end of the expression has been reached.
		JR POP	NZ,360C,V-RPORT-C HL	If not, the error is reported. The starting address of the string is fetched.
		POP XOR AND	AF (FLAGS) +40	The 'flag' for 'val/val\$' is fetched and bit 6 is compared with bit 6 of the result of the syntax scan.
360C	V-RPORT-C	JP	NZ,1C8A,REPORT-C	Report the error if they do not match.
		LD SET	(CH-ADD),HL 7,(FLAGS)	Start address to CH-ADD again. The flag is set for line execution.

CALL	24FB,SCANNING	The string is treated as a 'next expression' and a 'last value' produced.
POP	HL	The original value of CH-ADD is restored.
LD	(CH-ADD),HL	
JR	35BF,STK-PNTRS	The subroutine exits via STK-PNTRS which resets the pointers.

THE 'STR\$' FUNCTION

(Offset 2E: 'str\$')

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

361F	str\$	LD	BC,+0001	One space is made in the work space and its address is copied to K-CUR, the address of the cursor.
		RST	0030,BC-SPACES	
		LD	(K-CUR),HL	
		PUSH	HL	This address is saved on the stack too.
		LD	HL,(CURCHL)	The current channel address is saved on the machine stack.
		PUSH	HL	Channel 'R' is opened, allowing the string to be 'printed' out into the work space.
		LD	A,+FF	
		CALL	1601,CHAN-OPEN	The 'last value', X, is now printed out in the work space and the work space is expanded with each character.
		CALL	2DE3,PRINT-FP	Restore CURCHL to HL and restore the flags that are appropriate to it.
		POP	HL	Restore the start address of the string.
		CALL	1615,CHAN-FLAG	Now the cursor address is one past the end of the string and hence the difference is the length.
		POP	DE	Transfer the length to BC.
		LD	HL,(K-CUR)	
		AND	A	
		SBC	HL,DE	
		LD	B,H	
		LD	C,L	
		CALL	2AB2,STK-STO-\$	Pass the parameters of the new string to the calculator stack.
		EX	DE,HL	Reset the pointers.
		RET		Finished.

Note: See PRINT-FP for an explanation of the 'PRINT "A"+STR\$ 0.1' error.

THE 'READ-IN' SUBROUTINE

(Offset 1A: 'read-in')

This subroutine is called via the calculator offset through the first line of the S-INKEY\$ routine in SCANNING. It appears to provide for the reading in of data through different streams from those available on the standard Spectrum. Like INKEY\$ the subroutine returns a string.

3645	read-in	CALL	1E94,FIND-INT1	The numerical parameter is compressed into the A register.
		CP	+10	Is it smaller than 16 decimal?
		JP	NC,1E9F,REPORT-B	If not, report the error.
		LD	HL,(CURCHL)	The current channel address is saved on the machine stack.
		PUSH	HL	The channel specified by the parameter is opened.
		CALL	1601,CHAN-OPEN	

		CALL	15E6,INPUT-AD	The signal is now accepted, like a 'key-value'.
		LD	BC,+0000	The default length of the resulting string is zero.
		JR	NC,365F,R-I-STORE	Jump if there was no signal.
		INC	C	Set the length to 1 now.
		RST	0030,BC-SPACES	Make a space in the work space.
		LD	(DE),A	Put the string into it.
365F	R-I-STORE	CALL	2AB2,STK-STO-\$	Pass the parameters of the string to the calculator stack.
		POP	HL	Restore CURCHL and the appropriate flags.
		CALL	1615,CHAN-FLAG	Exit, setting the pointers.
		JP	35BF,STK-PNTRS	

THE 'CODE' FUNCTION

(Offset 1C: 'code')

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

3669	code	CALL	2BF1,STK-FETCH	The parameters of the string are fetched.
		LD	A,B	The length is tested and the A register holding zero is carried forward is A\$ is a null string.
		OR	C	The code of the first character is put into A otherwise.
		JR	Z,3671,STK-CODE	The subroutine exits via STACK-A which gives the correct 'last value'.
		LD	A,(DE)	
3671	STK-CODE	JP	2D28,STACK-A	

THE 'LEN' FUNCTION

(Offset 1E: 'len')

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

3674	len	CALL	2BF1,STK-FETCH	The parameters of the string are fetched.
		JP	2D2B,STACK-BC	The subroutine exits via STACK-BC which gives the correct 'last value'.

THE 'DECREASE THE COUNTER' SUBROUTINE

(Offset 35: 'dec-jr-nz')

This subroutine is only called by the SERIES GENERATOR subroutine and in effect is a 'DJNZ' operation but the counter is the system variable, BREG, rather than the B register.

367A	dec-jr-nz	EXX	HL	Go to the alternative register set and save the next literal pointer on the machine stack.
		PUSH	HL	Make HL point to BREG.
		LD	HL,+5C67	Decrease BREG.
		DEC	(HL)	Restore the next literal pointer.
		POP	HL	The jump is made on non-zero.
		JR	NZ,3687,JUMP-2	The next literal is passed over.
		INC	HL	Return to the main register set.
		EXX		Finished.
		RET		

THE 'JUMP' SUBROUTINE

(Offset 33: 'jump')

This subroutine executes an unconditional jump when called by the literal '33'. It is also used by the subroutines DECREASE THE COUNTER and JUMP ON TRUE.

3686	JUMP	EXX		Go to the next alternate register set.
3687	JUMP-2	LD	E,(HL)	The next literal (jump length) is put in the E' register.
		LD	A,E	The number 00 hex or FF hex is formed in A according as E' is positive or negative, and is then copied to D'.
		RLA		The registers H' & L' now hold the next literal pointer.
		SBC	A,A	Finished.
		LD	D,A	
		ADD	HL,DE	
		EXX		
		RET		

THE 'JUMP ON TRUE' SUBROUTINE

(Offset 00: 'jump-true')

This subroutine executes a conditional jump if the 'last value' on the calculator stack, or more precisely the number addressed currently by the DE register pair, is true.

368F	jump-true	INC	DE	Point to the third byte, which is zero or one.
		INC	DE	Collect this byte in the A register.
		LD	A,(DE)	Point to the first byte once again.
		DEC	DE	Test the third byte: is it zero?
		DEC	DE	Make the jump if the byte is non-zero, i.e. if the number is not-false.
		AND	A	Go to the alternate register set.
		JR	NZ,3686,JUMP	Pass over the jump length.
		EXX		Back to the main set of registers.
		INC	HL	Finished.
		EXX		
		RET		

THE 'END-CALC' SUBROUTINE

(Offset 38: 'end-calc')

This subroutine ends a RST 0028 operation.

369B	end-calc	POP	AF	The return address to the calculator ('RE-ENTRY') is discarded.
		EXX		Instead, the address in H'L' is put on the machine stack and an indirect jump is made to it.
		EX	(SP).HL	H'L' will now hold any earlier address in the calculator chain of addresses.
		EXX		Finished.
		RET		

THE 'MODULUS' SUBROUTINE

(Offset 32: 'n-mod-m')

This subroutine calculates $M \pmod{M}$, where M is a positive integer held at the top of the calculator stack, the 'last value', and N is the integer held on the stack beneath M.

The subroutine returns the integer quotient $\text{INT}(N/M)$ at the top of the calculator stack, the 'last value', and the remainder $N - \text{INT}(N/M)$ in the second place on the stack.

This subroutine is called during the calculation of a random number to reduce $N \pmod{65537}$ decimal.

36A0	n-mod-m	RST	0028,FP-CALC	N,M
		DEFB	+C0,st-mem-0	N,M mem-0 holds M
		DEFB	+02,delete	N
		DEFB	+31,duplicate	N, N
		DEFB	+E0,get-mem-0	N, N, M
		DEFB	+05,division	N, N/M
		DEFB	+27,int	N, INT (N/M)

DEFB	+E0,get-mem-0	N, INT (N/M),M
DEFB	+01,exchange	N, M, INT (N/M)
DEFB	+C0,st-mem-0	N, M, INT (N/M) mem-0 holds
		INT (N/M)
DEFB	+04,multiply	N, M*INT (N/M)
DEFB	+03,subtract	n-M*INT (N/M)
DEFB	+E0,get-mem-0	n-M*INT (N/M), INT (N/M)
DEFB	+38,end-calc	
RET		Finished.

THE 'INT' FUNCTION

(Offset 27: 'int')

This subroutine handles the function INT X and returns a 'last value' that is the 'integer part' of the value supplied. Thus INT 2.4 gives 2 but as the subroutine always rounds the result down INT -2.4 gives -3.

The subroutine uses the INTEGER TRUNCATION TOWARDS ZERO subroutine at 3214 to produce I (X) such that I (2.4) gives 2 and I (-2.4) gives -2. Thus, INT X is given by I (X) for values of X that are greater than or equal to zero, and I (X)-1 for negative values of X that are not already integers, when the result is, of course, I (X).

36AF	int	RST	0028,FP-CALC	X
		DEFB	+31,duplicate	X, X
		DEFB	+36,less-0	X, (1/0)
		DEFB	+00,jump-true	X
		DEFB	+04, to 36B7,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	+3A,truncate	I (X)
DEFB	+38,end-calc	
RET		Finished.

when X is a negative integer I (X) is returned, otherwise I (X)-1 is returned.

36B7	X-NEG	DEFB	+31,duplicate	X, X
		DEFB	+3A,truncate	X, I (X)
		DEFB	+C0,st-mem-0	X, I (X) mem-0 holds I (X)
		DEFB	+03,subtract	X-I (X)
		DEFB	+E0,get-mem-0	X-I (X), I (X)
		DEFB	+01,exchange	I (X), X-I (X)
		DEFB	+30,not	I (X), (1/0)
		DEFB	+00,jump-true	I (X)
		DEFB	+03,to 36C2,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	I (X), 1
DEFB	+03,subtract	I (X)-1

In either case the subroutine finishes with;

36C2	EXIT	DEFB	+38,end-calc	I (X) or I (X)-1
	RET			

THE 'EXPONENTIAL' FUNCTION

(Offset 26: 'exp')

This subroutine handles the function EXP X and is the first of four routines that use SERIES GENERATOR to produce Chebyshev polynomials.

The approximation to EXP X is found as follows:

- i. X is divided by LN 2 to give Y, so that 2 to the power Y is now the required result.
- ii. The value N is found, such that N=INT Y.
- iii. The value W is found, such that W=Y-N, where 0 <=W <=1, as required for the series to converge.

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.

36C4	EXP	RST	0028,FP-CALC	X
Perform step i.				
	DEFB	+3D, re-stack		X (in full floating-point form)
	DEFB	+34, stk-data		X, 1/LN 2
	DEFB	+F1, exponent+81		
	DEFB	+38, +AA, +3B, +29		
	DEFB	+04, multiply		X/LN 2 = Y
Perform step ii.				
	DEFB	+31, duplicate		Y, Y
	DEFB	+27, int, 1C46		Y, INT Y = N
	DEFB	+C3, st-mem-3		Y, N mem-3 holds N
Perform step iii.				
	DEFB	+03, subtract		Y-N = W
Perform step iv.				
	DEFB	+31, duplicate		W, W
	DEFB	+0F, addition		2^*W
	DEFB	+A1, stk-one		$2^*W, 1$
	DEFB	+03, subtract		$2^*W-1 = Z$

Perform step v, passing to the SERIES GENERATOR the parameter '8' and the eight constants required.

	DEFB	+88, series-08	Z
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	$2^{**}W, N$
DEFB	+38, end-calc	
CALL	2DD5, FP-TO-A	The absolute value of N mod 256 decimal, is put into the A register.
JR	NZ, 3705, N-NEGTV	Jump forward if N was negative.
JR	C, 3703, REPORT-6	Error if ABS N greater than 255 dec.
ADD	A, (HL)	Now add ABS N to the exponent.
JR	NC, 370C, RESULT-OK	Jump unless e greater than 255 dec.

Report 6 - Number too big

3703	REPORT-6	RST DEFB	0008,ERROR-1 +05	Call the error handling routine.
3705	N-NEGTV	JR SUB JR NEG	C,370E,RSLT-ZERO (HL) NC,370E,RSLT-ZERO	The result is to be zero if N is less than -255 decimal. Subtract ABS N from the exponent as N was negative. Zero result if e less than zero. Minus e is changed to e.
370C	RESULT-OK	LD RET	(HL),A	The exponent, e, is entered. Finished: 'last value' is EXP X
370E	RSLT-ZERO	RST DEFB DEFB DEFB RET	0028,FP-CALC +02,delete +A0,stk-zero +38,end-calc	Use the calculator to make the 'last value' zero.
				Finished, with EXP X = 0.

THE 'NATURAL LOGARITHM' FUNCTION

(Offset 25: 'ln')

This subroutine handles the function LN X and is the second of the four routines that use SERIES GENERATOR to produce Chebyshev polynomials.

The approximation to LN X is found as follows:

- I. X is tested and report A is given if X is not positive.
- II. X is then split into its true exponent, e', and its mantissa $X' = X/(2^{**}e')$, where X' is greater than, or equal to, 0.5 but still less than 1.
- III. The required value Y1 or Y2 is formed. If X' is greater than 0.8 then $Y1 = e'^{**}LN\ 2$ and if otherwise $Y2 = (e'-1)*LN\ 2$.
- IV. If X' is greater than 0.8 then the quantity X'^{-1} is stacked; otherwise $2*X'^{-1}$ is stacked.
- V. Now the argument Z is formed, being if X' is greater than 0.8, $Z = 2.5*X'^{-3}$; otherwise $Z = 5*X'^{-3}$. In each case, $-1 \leq Z \leq 1$, as required for the series to converge.
- VI. The SERIES GENERATOR is used to produce the required function.
- VII. Finally a simply multiplication and addition leads to LN X being returned as the 'last value'.

3713	ln	RST	0028,FP-CALC	X
------	----	-----	--------------	---

Perform step i.

DEFB	+3D,re-stack	X (in full floating-point form)
DEFB	+31,duplicate	X, X
DEFB	+37,greater-0	X, (1/0)
DEFB	+00,jump-true	X
DEFB	+04,to 371C, VALID	X
DEFB	+38,end-calc	X

Report A - Invalid argument

371A	REPORT-A	RST DEFB	0008,ERROR-1 +09	Call the error handling routine.
------	----------	-------------	---------------------	----------------------------------

Perform step ii.

371C	VALID	DEFB	+A0,stk-zero	X,0	The deleted 1 is
		DEFB	+02,delete	X	overwritten with zero.
		DEFB	+38,end-calc	X	
		LD	A,(HL)		The exponent, e, goes into A.
		LD	(HL),+80		X is reduced to X'.
		CALL	2D28,STACK-A		The stack holds: X', e.
		RST	0028,FP-CALC	X', e	
		DEFB	+34,stk-data	X', e, 128 (decimal)	
		DEFB	+38,exponent+88		

	DEFB	+00,(+00,+00,+00)	
	DEFB	+03,subtract	X', e'
Perform step iii.			
	DEFB	+01,exchange	e', X'
	DEFB	+31,duplicate	e', X', X'
	DEFB	+34,stk-data	e', X', X', 0.8 (decimal)
	DEFB	+F0,exponent+80	
	DEFB	+4C,+CC,+CC,+CD	
	DEFB	+03,subtract	e', X', X'-0.8
	DEFB	+37,greater-0	e', X', (1/0)
	DEFB	+00,jump-true	e', X'
	DEFB	+08,to 373D, GRE.8	e', X'
	DEFB	+01,exchange	X', e'
	DEFB	+A1,stk-one	X', e', 1
	DEFB	+03,subtract	X', e'-1
	DEFB	+01,exchange	e'-1, X'
	DEFB	+38,end-calc	e'-1, X'
	INC	(HL)	Double X' to give 2*X'.
	RST	0028,FP-CALC	e'-1,2*X'
373D	DEFB	+01,exchange	X',e' - X' large.
GRE.8			2*X',e'-1 - X' small.
	DEFB	+34,stk-data	X',e',LN 2
	DEFB	+F0,exponent+80	2*X',e'-1, LN 2
	DEFB	+31,+72,+17,+F8	
	DEFB	+04,multiply	X',e'*LN 2 = Y1
			2*X', (e'-1)*LN 2 = Y2
Perform step iv.			
	DEFB	+01,exchange	Y1, X' - X' large.
			Y2, 2*X' - X' small.
	DEFB	+A2,stk-half	Y1, X', .5 (decimal)
			Y2, 2*X', .5
	DEFB	+03,subtract	Y1, X'-.5
			Y2, 2*X'-.5
	DEFB	+A2,stk-half	Y1, X'-.5, .5
			Y2, 2*X'-.5, .5
	DEFB	+03,subtract	Y1, X'-1
			Y2, 2*X'-1
Perform step v.			
	DEFB	+31,duplicate	Y, X'-1, X'-1
			Y2, 2*X'-1, 2*X'-1
	DEFB	+34,stk-data	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	Y1, X'-1, 2.5*X'-2.5
			Y2, 2*X'-1, 5*X'-2.5
	DEFB	+A2,stk-half	Y1,X'-1, 2.5*X'-2.5, .5
			Y2, 2*X'-1, 5*X'-2.5, .5
	DEFB	+03,subtract	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	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	Y1=LN (2**e'), LN X'
		Y2=LN (2**(e'-1)), LN (2*X')
DEFB	+0F,addition	LN (2**e')*X' = LN X
		LN (2**(e'-1)*2*X') = LN X
DEFB	+38,end-calc	LN X
RET		Finished: 'last value' is LN X.

THE 'REDUCE ARGUMENT' SUBROUTINE

(Offset 39: 'get-argt')

This subroutine transforms the argument X of SIN X or COS X into a value V.

The subroutine first finds a value Y such that:

$Y = X/(2*PI) - INT(X/(2*PI) + 0.5)$, 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 $SIN(PI*V/2) = SIN X$

3783	get-argt	RST	0028,FP-CALC	X
		DEFB	+3D,re-stack	X (in full floating-point form)
		DEFB	+34,stk-data	X, 1/(2*PI)
		DEFB	+EE,exponent+7E	
		DEFB	+22,+F9,+83,+6E	
		DEFB	+04,multiply	X/(2*PI)
		DEFB	+31,duplicate	X/(2*PI), X/(2*PI)
		DEFB	+A2,stk-half	X/(2*PI), X/(2*PI), 0.5
		DEFB	+0F,addition	X/(2*PI), X/(2*PI)+0.5
		DEFB	+27,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	+31,duplicate	Y, Y
DEFB	+0F,addition	2*Y
DEFB	+31,duplicate	2*Y, 2*Y
DEFB	+0F,addition	4*Y
DEFB	+31,duplicate	4*Y, 4*Y
DEFB	+2A,abs	4*Y, ABS (4*Y)
DEFB	+A1,stk-one	4*Y, ABS (4*Y), 1
DEFB	+03,subtract	4*Y, ABS (4*Y)-1 = Z
DEFB	+31,duplicate	4*Y, Z, Z
DEFB	+37,greater-0	4*Y, Z, (1/0)
DEFB	+C0,stk-mem-0	Mem-0 holds the result of the test.
DEFB	+00,jump-true	4*Y, Z
DEFB	+04, to 37A1,ZPLUS	4*Y, Z
DEFB	+02,delete	4*Y
DEFB	+38,end-calc	4*Y = V - case i.
RET		Finished.

If the jump was made then continue.

37A1	ZPLUS	DEFB	+A1,stk-one	4*Y, Z, 1
		DEFB	+03,subtract	4*Y, Z-1
		DEFB	+01,exchange	Z-1,4*Y
		DEFB	+36,less-0	Z-1,(1/0)
		DEFB	+00,jump-true	Z-1
		DEFB	+02,to 37A8,YNEG	Z-1
		DEFB	+1B,negate	1-Z
37A8	YNEG	DEFB	+38,end-calc	1-Z = V - case ii.
				Z-1 = V - case iii.
		RET		Finished.

THE 'COSINE' FUNCTION

(Offset 20: 'cos')

This subroutine handles the function COS X and returns a 'last value' that is an approximation to COS X.

The subroutine uses the expression:

$\text{COS } X = \text{SIN } (\text{PI} * W / 2)$, where $-1 \leq W \leq 1$.

In deriving W for X the subroutine uses the test result obtained in the previous subroutine and stored for this purpose in mem-0. It then jumps to the SINE, subroutine, entering at C-ENT, to produce a 'last value' of COS X.

37AA	cos	RST	0028,FP-CALC.	X
		DEFB	+39,get-argt	V
		DEFB	+2A,abs	ABS V
		DEFB	+A1,stk-one	ABS V, 1
		DEFB	+03,subtract	ABS V-1
		DEFB	+E0,get-mem-0	ABS V-1, (1/0)
		DEFB	+00,jump-true	ABS V-1
		DEFB	+06, to 37B7,C-ENT	ABS V-1 = W

If the jump was not made then continue.

DEFB	+1B,negate	1-ABS V
DEFB	+33,jump	1-ABS V
DEFB	+03, to 37B7,C-ENT	1-ABS V = W

THE 'SINE' FUNCTION

(Offset 1F: 'sin')

This subroutine handles the function SIN X and is the third of the four routines that use SERIES GENERATOR to produce Chebyshev polynomials.

The approximation to SIN X is found as follows:

i. The argument X is reduced and in this case $W = V$ directly.

Note that $-1 \leq W \leq 1$, as required for the series to converge.
 ii. The argument Z is formed, such that $Z = 2 * W * W - 1$.
 iii. The SERIES GENERATOR is used to return $(\sin(\pi * W / 2)) / W$
 iv. Finally a simple multiplication gives $\sin X$.

37B5	sin	RST	0028 FP-CALC	X
------	-----	-----	--------------	---

Perform step i.

DEFB	+39,get-argt	W
------	--------------	---

Perform step ii. The subroutine from now on is common to both the SINE and COSINE functions.

37B7	C-ENT	DEFB	+31,duplicate	W, W
		DEFB	+31,duplicate	W, W, W
		DEFB	+04,multiply	W, W*W
		DEFB	+31,duplicate	W, W*W, W*W
		DEFB	+0F,addition	W, 2*W*W
		DEFB	+A1,stk-one	W, 2*W*W, 1
		DEFB	+03,subtract	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	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	$\sin(\pi * W / 2) = \sin X$ (or =
		$\cos X)$
DEFB	+38,end-calc	
RET		Finished: 'last value' = $\sin X$.
		or ('last value' = $\cos X)$

THE 'TAN' FUNCTION

(Offset 21: 'tan')

This subroutine handles the function $\tan X$. The subroutine simply returns $\sin X / \cos X$, with arithmetic overflow if $\cos X = 0$.

37DA	tan	RST	0028,FP-CALC	X
		DEFB	+31,duplicate	X, X
		DEFB	+1F,sin	X, $\sin X$
		DEFB	+01,exchange	$\sin X$, X
		DEFB	+20,cos	$\sin X$, $\cos X$
		DEFB	+05,division	$\sin X / \cos X = \tan X$
				Report arithmetic overflow if
				needed.
		DEFB	+38,end-calc	$\tan X$
		RET		Finished: 'last value' = $\tan X$.

THE 'ARCTAN' FUNCTION

(Offset 24: 'atn')

This subroutine handles the function ATN X and is the last of the four routines that use SERIES GENERATOR to produce Chebyshev polynomials. It returns a real number between $-\pi/2$ and $\pi/2$, which is equal to the value in radians of the angle whose tan is X. The approximation to ATN X is found as follows:

i. The values W and Y are found for three cases of X, such that:

if $-1 < X < 1$ then $W = 0$ & $Y = X$ - case i.
 if $-1 \leq X$ then $W = \pi/2$ & $Y = -1/X$ - case ii.
 if $X \leq -1$ then $W = -\pi/2$ & $Y = -1/X$ - case iii.

In each case, $-1 \leq Y \leq 1$, 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 $X \leq -1$ 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.

37E2	atn	CALL	3297,RE-STACK	Use the full floating-point form of X.
		LD	A,(HL)	Fetch the exponent of X.
		CP	+81	
		JR	C,37F8,SMALL	Jump forward for case i: $Y = X$.
		RST	0028,FP-CALC	X
		DEFB	+A1,stk-one	X, 1
		DEFB	+1B,negate	X,-1
		DEFB	+01,exchange	-1, X
		DEFB	+05,division	-1/X
		DEFB	+31,duplicate	-1/X, -1/X
		DEFB	+36,less-0	-1/X, (1/0)
		DEFB	+A3,stk-pi/2	-1/X, (1/0), $\pi/2$
		DEFB	+01,exchange	-1/X, $\pi/2$, (1/0)
		DEFB	+00,jump-true	-1/X, $\pi/2$
		DEFB	+06, to 37FA,CASES	Jump forward for case ii:
				$Y = -1/X$ $W = \pi/2$
		DEFB	+1B,negate	-1/X, - $\pi/2$
		DEFB	+33,jump	-1/X, - $\pi/2$
		DEFB	+03,to 37FA,CASES	Jump forward for case iii:
				$Y = -1/X$ $W = -\pi/2$
37F8	SMALL	RST	0028,FP-CALC	Y
		DEFB	+A0,stk-zero	Y, 0
				Continue for case i: $W = 0$

Perform step ii.

37FA	CASES	DEFB	+01,exchange	W, Y
		DEFB	+31,duplicate	W, Y, Y
		DEFB	+31,duplicate	W, Y, Y, Y
		DEFB	+04,multiply	W, Y, $Y*Y$
		DEFB	+31,duplicate	W, Y, $Y*Y$, $Y*Y$
		DEFB	+0F,addition	W, Y, $2*Y*Y$
		DEFB	+A1,stk-one	W, Y, $2*Y*Y$, 1
		DEFB	+03,subtract	W, Y, $2*Y*Y-1 = Z$

Perform step iii, passing to the SERIES GENERATOR the parameter '12' decimal, and the twelve constants required.

		DEFB	+8C,series-0C	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
DEFB	+0F,addition
DEFB	+38,end-calc
RET	

W, ATN X- case i.
W, ATN (-1/X) - case ii.
W, ATN (-1/X) - case iii.
ATN X - all cases now.

Finished: 'last value' = ATN X.

THE 'ARCSIN' FUNCTION

(Offset 22: 'asn')

This subroutine handles the function ASN X and return a real real number from -PI/2 to PI/2 inclusive which is equal to the value in radians of the angle whose sine is X. Thereby if Y = ASN X then X = SIN Y.

This subroutine uses the trigonometric identity:

$$\text{TAN}(Y/2) = \text{SIN } Y / (1 + \text{COS } Y)$$

to obtain TAN (Y/2) and hence (using ATN) Y/2 and finally Y.

3833	asn	RST	0028,FP-CALC	X
		DEFB	+31,duplicate	X, X
		DEFB	+31,duplicate	X, X, X
		DEFB	+04,multiply	X, X*X
		DEFB	+A1,stk-one	X, X*X, 1
		DEFB	+03,subtract	X, X*X-1
		DEFB	+1B,negate	X, 1-X*X
		DEFB	+28,sqr	X, SQR (1-X*X)
		DEFB	+A1,stk-one	X, SQR (1-X*X), 1
		DEFB	+0F,addition	X, 1+SQR (1-X*X)
		DEFB	+05,division	X/(1+SQR (1-X*X)) = TAN
				(Y/2)
		DEFB	+24,atn	Y/2
		DEFB	+31,duplicate	Y/2, Y/2
		DEFB	+0F,addition	Y = ASN X
		DEFB	+38,end-calc	
		RET		Finished: 'last value' = ASN X.

THE 'ARCCOS' FUNCTION

(Offset 23: 'acs')

This subroutine handles the function ACS X and returns a real number from zero to PI inclusive which is equal to the value in radians of the angle whose cosine is X.

This subroutine uses the relation:

$$\text{ACS } X = \text{PI}/2 - \text{ASN } X$$

3843	acs	RST	0028,FP-CALC	X
		DEFB	+22,asn	ASN X
		DEFB	+A3,stk-pi/2	ASN X,PI/2
		DEFN	+03,subtract	ASN X-PI/2
		DEFB	+1B,negate	PI/2-ASN X = ACS X
		DEFB	+38,end-calc	
		RET		Finished: 'last value' = ACS X.

THE 'SQUARE ROOT' FUNCTION

(Offset 28: 'sqr')

This subroutine handles the function SQR X and returns the positive square root of the real number X if X is positive, and zero if X is zero. A negative value of X gives rise to report A - invalid argument (via Ln in the EXPONENTIATION subroutine).

This subroutine treats the square root operation as being $X^{.5}$ and therefore stacks the value .5 and proceeds directly into the EXPONENTIATION subroutine.

384A	sqr	RST	0028,FP-CALC	X
		DEFB	+31,duplicate	X,X
		DEFB	+30,not	X,(1/0)
		DEFB	+00,jump-true	X
		DEFB	+1E,to 386C,LAST	X

The jump is made if X = 0, otherwise continue with:

DEFB	+A2,stk-half	X,.5
DEFB	+38,end-calc	

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 $\text{EXP}(Y \cdot \text{LN } X)$. 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.

3851	to-power	RST	0028,FP-CALC	X,Y
		DEFB	+01,exchange	Y,X
		DEFB	+31,duplicate	Y,X,X
		DEFB	+30,not	Y,X,(1/0)
		DEFB	+00,jump-true	Y,X
		DEFB	+07,to 385D,XIS0	Y,X

The jump is made if X = 0, otherwise $\text{EXP}(Y \cdot \text{LN } X)$ is formed.

DEFB	+25,ln	Y, LN X
------	--------	---------

Giving report A if X is negative.

DEFB	+04,multiply	Y*LN X
DEFB	+38,end-calc	
JP	36C4,EXP	Exit via EXP to form EXP (Y*LN X).

The value of X is zero so consider the three possible cases involved.

385D	XIS0	DEFB	+02,delete	Y
		DEFB	+31,duplicate	Y,Y
		DEFB	+30,not	Y,(1/0)
		DEFB	+00,jump-true	Y

DEFB	+09,to 386A,ONE	Y
------	-----------------	---

The jump is made if X = 0 and Y = 0, otherwise proceed.

DEFB	+A0,stk-zero	Y,0
DEFB	+01,exchange	0,Y
DEFB	+37,greater-0	0,(1/0)
DEFB	+00,jump-true	0
DEFB	+06,to 386C,LAST	0

The jump is made if X = 0 and Y is positive, otherwise proceed.

DEFB	+A1,stk-one	0,1
DEFB	+01,exchange	1,0
DEFB	+05,division	Exit via 'division' as dividing by zero gives 'arithmetic overflow'.

The result is to be 1 for the operation.

386A	ONE	DEFB +02,delete	-
		DEFB +A1,stk-one	1

Now return with the 'last value' on the stack being 0**Y.

386C	LAST	DEFB +38,end-calc	(1/0)
		RET	Finished: 'last value' is 0 or 1.

386E - 3CFF These locations are 'spare'. They all hold +FF.

3D00 - 3FFF These locations hold the 'character set'. There are 8 byte representations for all the characters with codes +20 (space) to +7F (©).

e.g. the letter 'A' has the representation 00 3C 42 42 7E 42 42 00 and thereby the form:

```

00000000
00111100
01000010
01000010
01111110
01000010
01000010
01000010
00000000

```

APPENDIX

BASIC PROGRAMS FOR THE MAIN SERIES

The following BASIC programs have been included as they give a good illustration of how Chebyshev polynomials are used to produce the approximations to the functions SIN, EXP, LN and ATN.

The series generator:

This subroutine is called by all the 'function' programs.

```
500 REM SERIES GENERATOR, ENTER
510 REM USING THE COUNTER BREG
520 REM AND ARRAY-A HOLDING THE
530 REM CONSTANTS.
540 REM FIRST VALUE IN Z.
550 LET M0=2*Z
560 LET M2=0
570 LET T=0
580 FOR I=BREG TO 1 STEP -1
590 LET M1=M2
600 LET U=T*M0-M2+A(BREG+1-I)
610 LET M2=T
620 LET T=U
630 NEXT I
640 LET T=T-M1
650 RETURN
660 REM LAST VALUE IN T.
```

In the above subroutine the variable are:

Z -	the entry value.
T -	the exit value.
M0 -	mem-0
M1 -	mem-1
M2 -	mem-2
I -	the counter for BREG.
U -	a temporary variable for T.
A(1) to	
A(BREG) -	the constants.
BREG -	the number of constants to be used.

To see how the Chebyshev polynomials are generated, record on paper the values of U, M1, M2 and T through the lines 550 to 630, passing, say, 6 times through the loop, and keeping the algebraic expressions for A(1) to A(6) without substituting numerical values. Then record T-M1. The multipliers of the constants A(1) to A(6) will then be the required Chebyshev polynomials. More precisely, the multiplier of A(1) will be $2*T_5(Z)$, 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 $n \geq 2$, $T_n(Z)=2*Z*T_{n-1}(Z)-T_{n-2}(Z)$.

SIN X

```
10 REM DEMONSTRATION FOR SIN X
20 REM USING THE 'SERIES GENERATOR'.
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 BREG=6
260 REM USE 'SERIES GENERATOR'
270 GO SUB 550
280 PRINT TAB 6; "SIN ";C;" DEGREES"
290 PRINT
300 PRINT T*W,SIN (PI*C/180)
310 PRINT
320 NEXT J
330 GO TO 100
```

NOTES:

- I. When C is entered this program calculates and prints SIN C degrees, SIN (C+10) degrees, SIN (C+20) degrees and SIN (C+30) degrees. It also prints the values obtained by using the ROM program. For a specimen of results, try entering these values in degrees: 0; 5; 100; -80; -260; 3600; -7200.
- II. The constants A(1) to A(6) in lines 40 to 90 are given (apart from a factor of 1/2) in Abramowitz and Stegun *Handbook of Mathematical Functions* (Dover 1965) page 76. They can be checked by integrating $(\sin(\pi X/2))/X$ over the interval $U=0$ to π , after first multiplying by $\cos(N*U)$ for each constant (i.e. $N=1,2,\dots,6$) and substituting $\cos U=2*X*X-1$. Each result should then be divided by π . (This integration can be performed by approximate methods e.g. using Simpson's Rule if there is a reasonable computer or programmable calculator to hand.)

EXP X

```
10 REM DEMONSTRATION FOR EXP X
20 REM USING THE 'SERIES GENERATOR'
30 LET T=0 (This makes T the first variable.)
40 DIM A(8)
50 LET A(1)=0.000000001
60 LET A(2)=0.000000053
70 LET A(3)=0.000001851
80 LET A(4)=0.000053453
90 LET A(5)=0.001235714
100 LET A(6)=0.021446556
110 LET A(7)=0.248762434
120 LET A(8)=1.45699875
130 PRINT
140 PRINT "ENTER START VALUE"
150 INPUT C
160 CLS
170 LET C=C-10
180 PRINT "BASIC PROGRAM", "ROM PROGRAM"
190 PRINT "-----", "-----"
200 PRINT
210 FOR J=1 TO 4
220 LET C=C+10
230 LET D=C*1.442695041 (D=C*(1/LN 2);EXP C=2**D).
240 LET N=INT D
250 LET Z=D-N (2**(N+Z) is now required).
260 LET Z=2*Z-1
270 LET BREG=8
280 REM USE "SERIES GENERATOR"
290 GO SUB 550
300 LET V=PEEK 23627+256*PEEK 23628+1 (V=(VARS)+1)
310 LET N=N+PEEK V
320 IF N > 255 THEN STOP (STOP with arithmetic overflow).
330 IF N < 0 THEN GO TO 360
340 POKE V,N
350 GO TO 370
360 LET T=0
370 PRINT TAB 11;"EXP ";C
380 PRINT
390 PRINT T,EXP C
400 PRINT
410 NEXT J
420 GO TO 130
```

NOTES:

- I. When C is entered this program calculates and prints EXP C, EXP (C+10), EXP (C+20) and EXP (C+30). It also prints the values obtained by using the ROM program. For a specimen of results, try entering these values: 0; 15; 65 (with overflow at the end); -100; -40.
- II. The exponent is tested for overflow and for a zero result in lines 320 and 330. These tests are simpler in BASIC than in machine code, since the variable N, unlike the A register, is not confined to one byte.
- III. The constants A(1) to A(8) in lines 50 to 120 can be obtained by integrating 2^{**X} over the interval $U=0$ to π , after first multiplying the $\cos(N*U)$ for each constant (i.e. for $N=1,2,\dots,8$) and substituting $\cos U = 2^{*X}-1$. Each result should then be divided by π .

LN X:

```
10 REM DEMONSTRATION FOR LN X
20 REM USING THE 'SERIES GENERATOR'
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 STOP (STOP with 'invalid argument'.)
280 LET D=C
290 LET V=PEEK 23627+256*PEEK 23628+1
300 LET N=PEEK V-128 (N holds e').
310 POKE V,128
320 IF D<=0.8 THEN GO TO 360 (D holds X').
330 LET S=D-1
340 LET Z=2.5*D-3
350 GO TO 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 BREG=12
410 REM USE 'SERIES GENERATOR'
420 GO SUB 550
430 PRINT TAB 8;"LN ";C
440 PRINT
450 PRINT S*T+R, LN C
460 PRINT
470 NEXT J
480 GO TO 170
```

NOTES:

- I. When C is entered this program calculates and prints LN C, LN (C**2), LN (C**4) and LN (C**8). It also prints the values obtained by using the ROM program. For a specimen of results, try entering these values: 1.1; 0.9; 300; 0.004; 1E5 (for overflow) and 1E-5 (STOP as 'invalid argument').
- II. The constants A(1) to A(12) in lines 50 to 160 can be obtained by integrating $5 \cdot \ln(4 \cdot (X+1)^5 / (4 \cdot X - 1))$ over the interval U=0 to PI, after first multiplying by COS (N*U) for each constant (i.e. for N=1,2,...,12) and substituting COS U=2*X-1. Each result should then be divided by PI.

ATN X:

```
10 REM DEMONSTRATION FOR ATN X
20 REM USING THE 'SERIES GENERATOR'
30 DIM A(12)
40 LET A(1)= -.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 BREG=12
290 REM USE "SERIES GENERATOR"
300 GO SUB 550
310 LET T=D*T
320 IF B > =1 THEN LET T=T+PI/2
330 IF B < =-1 THEN LET T=T-PI/2
340 PRINT TAB 8;"ATN ";B
350 PRINT
360 PRINT T,ATN B          (or PRINT T*180/PI,ATN B*180/PI
370 PRINT                  to obtain the answers in degrees)
380 NEXT J
390 GO TO 160
```

NOTES:

- I. When C is entered this program calculates and prints ATN C, ATN (C*2), ATN (C*3) and ATN (C*4). For a specimen of results, try entering these values: 0.2; -1; 10 and -100. The results may be found more interesting if converted to yield degrees by multiplying the answers in line 360 by 180/PI.
- II. The constants A(1) to A(12) in lines 40 to 150 are given (apart from a factor of 1/2) in Abramowitz and Stegun *Handbook of Mathematical Functions* (Dover 1965) page 82. They can be checked by integrating ATN X/X over the interval U=0 to PI, after first multiplying by COS (N*U) for each parameter (i.e. for n=1,2,...,12) and substituting COS U=2*X*X-1. Each result should then be divided by PI.

An alternative subroutine for SIN X:

It is straightforward to produce the full expansion of the Chebyshev polynomials and this can be written in BASIC as follows:

```

550 LET T =(32*Z*Z*Z*Z*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

```

The expansion for LN X and A TN X, given algebraically, will be:

	$(2048z^{11}-5632z^9+5632z^7-2464z^5+440z^3-22z) * A(1)$
+	$(1024z^{10}-2560z^8+2240z^6-800z^4+100z^2-2) * A(2)$
+	$(512z^9-1152z^7+864z^5-240z^3+18z) * A(3)$
+	$(256z^8-512z^6+320z^4-64z^2+2) * A(4)$
+	$(128z^7-224z^5+112z^3-14z) * A(5)$
+	$(64z^6-96z^4+36z^2-2) * A(6)$
+	$(32z^5-40z^3+10z) * A(7)$
+	$(16z^4-16z^2+2) * A(8)$
+	$(8z^3-6z) * A(9)$
+	$(4z^2-2) * A(10)$
+	$(2z) * A(11)$
+	$A(12)$

THE 'DRAW' ALGORITHM

The following BASIC program illustrates the essential parts of the DRAW operation when being used to produce a straight line. The program in its present form only allows for lines where $X > Y$.

```
10 REM DRAW 255,175 PROGRAM
20 REM SET ORIGIN
30 LET PLOTx=0: LET PLOTy=0
40 REM SET LIMITS
50 LET X=255: LET Y=175
60 REM SET INCREMENT,i
70 LET i=X/2
80 REM ENTER LOOP
90 FOR B=X TO 1 STEP -1
100 LET A=Y+i
110 IF X> A THEN GO TO 160
120 REM UP A PIXEL ON THIS PASS
130 LET A=A-X
140 LET PLOTy=PLOTy+1
150 REM RESET INCREMENT,i
160 LET i=A
170 REM ALWAYS ALONG ONE PIXEL
180 LET PLOTx=PLOTx+1
190 REM NOW MAKE A PLOT
200 PLOT PLOTx,PLOTy
210 NEXT B
```

A complete algorithm is to found in the following program, as a subroutine that will 'DRAW A LINE' from the last position to X,Y.

THE 'CIRCLE' ALGORITHM

The following BASIC program illustrates how the CIRCLE command produces its circles.

Initially the number of arcs required is calculated. Then a set of parameters is prepared in the 'memory area' and the 'calculator stack'.

The arcs are then drawn by repeated calls to the line drawing subroutine that on each call draws a single line from the 'last position' to the position 'X,Y'.

Note: In the ROM program there is a final 'closing' line but this feature has not been included here.

```
10 REM A CIRCLE PROGRAM
20 LET X=127: LET Y=87: LET Z=87
30 REM How many arcs?
40 LET Arcs=4*INT (INT (ABS (PI*SQR Z)+0.5)/4)+4
50 REM Set up memory area; M0-M5
60 LET M0=X+Z
70 LET M1=0
80 LET M2=2*Z*SIN (PI/Arcs)
90 LET M3=1-2*(SIN (PI/Arcs)) ^ 2
100 LET M4=SIN (2*PI/Arcs)
110 LET M5=2*PI
120 REM Set up stack; Sa-Sd
130 LET Sa=X+Z
140 LET Sb=Y-Z*SIN (PI/Arcs)
150 LET Sc=Sa
160 LET Sd=Sb
170 REM Initialise COORDS
180 POKE 23677,Sa: POKE 23678,Sb
190 LET M0=Sd
200 REM 'DRAW THE ARCS'
210 LET M0=M0+M2
220 LET Sc=Sc+M1
230 LET X=Sc-PEEK 23677
240 LET Y=M0-PEEK 23678
```

```

250 GO SUB 510
260 LET Arcs=Arcs-1: IF Arcs=0 THEN STOP
270 LET MM1=M1
280 LET M1=M1*M3-M2*M4
290 LET M2=MM1*M4+M2*M3
300 GO TO 210

500 REM 'DRAW A LINE' from last position to X,Y
510 LET PLOTx=PEEK 23677: LET PLOTy=PEEK 23678
520 LET dx=SGN X: LET dy=SGN Y
530 LET X=ABS X: LET Y=ABS Y
540 IF X>=Y THEN GO TO 580
550 LET L=X: LET B=Y
560 LET ddx=0: LET ddy=dy
570 GO TO 610
580 IF X+Y=0 THEN STOP
590 LET L=Y: LET B=X
600 LET ddx=dx: LET ddy=0
610 LET H=B
620 LET i=INT (B/2)
630 FOR N=B TO 1 STEP -1
640 LET i=i+L
650 IF i < H THEN GO TO 690
660 LET i=i-H
670 LET ix=dx: LET iy=dy
680 GO TO 700
690 LET ix=ddx: LET iy=ddy
700 LET PLOTy=PLOTy+iy
710 IF PLOTy <0 OR PLOTy > 175 THEN STOP
720 LET PLOTx=PLOTx+ix
730 IF PLOTx <0 OR PLOTx > 255 THEN STOP
740 PLOT PLOTx,PLOTy
750 NEXT N
760 RETURN

```

NOTE ON SMALL INTEGERS AND -65536.

1. Small integers n are those for which -65535 is less than or equal to n which is less than or equal to 65535. The form in which they are held is described in 'STACK-BC'. Note that the manual is inaccurate when it says that the third and fourth bytes hold n plus 131072 if n is negative. Since the range of n is then -1 to -65535, the two bytes can only hold n plus 131072 if it is taken mod 65536; i.e. they hold n plus 65536. The manual is fudging the issue. The fact is that this is not a true twos complement form (as the form n plus 131072, in other circumstances, could be). Here the same number can stand for two different numbers according to the sign byte: e.g. 00 01 stands for 1 if the sign byte is 00 and for -65535 if the sign byte is FF; similarly FF FF stands for 65535 if the sign byte is 00 and for -1 if the sign byte is FF.

2. Accepting that negative numbers are given a special 'twos complement' form, the main feature about this method of holding numbers is that they are ready for 'short addition' without any further twos complementing. They are simply fetched and stored direct by the addition subroutine. But for multiplication they need to be fetched by INT-FETCH and stored afterwards by INT-STORE. These subroutines twos complement the number when fetching or storing it. The calls to INT-STORE are from 'multiply' (after 'short multiplication'), from 'truncate' (after forming a 'small integer' between -65535 and 65535 inclusive), from 'negate'/abs' for the 'integer case' and from 'sgn' to store 1 or -1. The calls to INT-FETCH are from PRINT-FP to fetch the integer part of the number when it is 'small', from 'multiply' twice to fetch two 'small integers', from 'RE-STACK' to fetch a 'small integer' for re-stacking, from 'negate'/abs' to fetch a 'small integer' for manipulation and from FP-TO-BC to fetch the integer for transfer to BC.

The Number -65536.

3. The number -65536 can fit into the 'small integer' format as 00 FF 00 00 00. It is then the 'limiting number', the one which when twos complemented overflows (cf. 80 hex in a simple one byte or 7 bit system, i.e. -128 decimal, which when twos complemented still gives 80 hex i.e. -128 decimal since the positive number 128 decimal does not fit into the system).

4. Some awareness of this may have inspired the abortive attempt to create 00 FF 00 00 00 in 'truncate'. It is abortive since it does not even survive the INT routine of which 'truncate' is a part. It just leads to the mistake INT (-65536) equals -1.

5. But the main error is that this number has been allowed to arise from 'short addition' of two smaller negative integers and then simply put on the stack as 00 FF 00 00 00. The system cannot cope with this number. The solution proposed in 'addition' is to form the full five byte floating-point form at once; i.e. test for the number first, at about byte 3032, as follows:

3032		PUSH	AF	Save the sign byte in A.
3033		INC	A	Make any FF in A into 00.
3034		OR	E	Test all 3 bytes now for zero.
3035		OR	D	
3036		JR	NZ,3040,ADD-STORE	Jump if not -65536.
3038		POP	AF	Clear the stack.
3039		LD	(HL),+80	Enter 80 hex into second byte.
303B		DEC	HL	Point to the first byte.
303C		LD	(HL),+91	Enter 91 hex into the first byte.
303E		JR	3049,ADD-RSTOR	Jump to set the pointer and exit.
3040	ADD-STORE	POP	AF	Restore the sign byte in A.
3041		LD	(HL),A	Store it on the stack.
3042		INC	HL	Point to the next location.
3043		LD	(HL),E	Store the low byte of the result.
3044		INC	HL	Point to the next location.
3045		LD	(HL),D	Store the high byte of the result.
3046		DEC	HL	Move the pointer back to
3047		DEC	HL	address the first byte of the
3048		DEC	HL	result.
3049	ADD-RSTOR	POP	DE	Restore STKEND to DE.
304A		RET		Finished.

6. The above amendment (i.e. 15 extra bytes) with the omission of bytes 3223 to 323E inclusive from 'truncate' should solve the problems. It would be nice to be able to test this. The calls of INT-STORE should not lead to 00 FF 00 00 00 being stacked. In 'multiply' the number will lead to overflow if it occurs, since 65536 will set the carry flag; so 'long' multiplication will be used. As noted at 30E5, the 5 bytes starting there could probably be omitted if the above amendments were made. 'Negate' avoids stacking 00 FF 00 00 00 by treating zero separately and returning it unaltered. Truncate deals separately with -65536, as noted above. SGN stores only 1 and -1.

INDEX TO ROUTINES

address routine page

THE RESTART ROUTINES and TABLES

0000	START	1
0008	Error	1
0010	Print a character	1
0018	Collect character	1
0020	Collect next character	1
0028	Calculator	1
0030	Make BC spaces	1
0038	Maskable interrupt	1
0053	ERROR-2	2
0066	Non-maskable Interrupt	2
0074	CH-ADD+1	2
007D	SKIP-OVER	2
0095	Token tables	3
0205	Key tables	4

THE KEYBOARD ROUTINES

028E	Keyboard scanning	5
02BF	KEYBOARD	6
0310	Repeating key	7
031E	K-TEST	7
0333	Keyboard decoding	8

THE LOUDSPEAKER ROUTINES

03B5	BEEPER	11
03F8	BEEP	12
046E	Semi-tone table	14

THE CASSETTE HANDLING ROUTINES

04C2	SA-BYTES	15
053F	SA/LD-RET	17
0556	LD-BYTES	17
05E3	LD-EDGE-2	20
0605	SAVE-ETC	21
07CB	VERIFY control	26
0802	Load a data block	26
0808	LOAD control	27
08B6	MERGE control	29
0970	SAVE control	32
09A1	Cassette messages	32

THE SCREEN & PRINTER HANDLING ROUTINES

09F4	PRINT-OUT	33
0A11	Control character table	33
0A23	Cursor left	33
0A3D	Cursor right	33
0A4F	Carriage return	34
0A5F	Print comma	34
0A69	Print a question mark	34
0A6D	Control characters with operands	34
0AD9	PO-ABLE	35
0ADC	Position store	36
0B03	Position fetch	36
0B24	Print any character	36
0B7F	Print all characters	37
0BDB	Set attribute byte	39
0C0A	Message printing	39
0C3B	PO-SAVE	40
0C41	Table search	40

address	routine	page
0C55	Test for scroll	40
0CF8	'scroll?' message	42
0D4D	Temporary colour items	43
0D6B	CLS command	43
0DAF	Clearing the whole display area	44
0DD9	CL-SET	45
0DFE	Scrolling	45
0E44	Clear lines	46
0E88	CL-ATTR	48
0E9B	CL-ADDR	48
0EAC	COPY command	48
0ECD	COPY-BUFF	49
0EF4	COPY-LINE	49
0F2C	EDITOR	50
0F81	ADD-CHAR	51
0FA0	Editing keys table	52
0FA9	EDIT key	52
0FF3	Cursor down editing	53
1007	Cursor left editing	53
100C	Cursor right editing	53
1015	DELETE editing	53
101E	ED-IGNORE	53
1024	ENTER editing	53
1031	ED-EDGE	53
1059	Cursor up editing	54
1076	ED-SYMBOL	54
107F	ED-ERROR	54
1097	CLEAR-SP	55
10A8	Keyboard input	55
111D	Lower screen copying	56
1190	SET-HL	57
11A7	REMOVE-FP	58

THE EXECUTIVE ROUTINES

11B7	NEW command	59
11CB	Main entry (Initialisation)	59
11DA	RAM-CHECK	59
12A2	Main execution loop	61
1391	Report messages	63
155D	MAIN-ADD	64
15AF	Initial channel information	65
15C6	Initial stream data	65
15D4	WAIT-KEY	65
15E6	INPUT-AD	66
15EF	Main printing	66
1601	CHAN-OPEN	66
1615	CHAN-FLAG	67
162D	Channel code look-up table	67
1634	Channel K flag	67
1642	Channel S flag	67
164D	Channel P flag	67
1652	ONE-SPACE	67
1655	MAKE-ROOM	67
1664	POINTERS	68
168F	Collect a line number	69
169E	RESERVE	69
16B0	SET-MIN	69
16D4	Reclaim the edit-line	70
16DB	INDEXER	70
16E5	CLOSE # commend	70

address	routine	page
1716	CLOSE stream look-up table	71
171E	Stream data	71
1736	OPEN # command	71
177A	OPEN stream look-up table	72
1793	CAT, ERASE, FORMAT & MOVE commands	73
1795	LIST & LLIST commands	73
1795	AUTO-LIST	73
17F5	LLIST	74
17F9	LIST	74
1855	Print a whole BASIC line	75
18B6	NUMBER	76
18C1	Print a flashing character	77
18E1	Print the cursor	77
190F	LN-FETCH	77
1925	Printing characters in a BASIC line	78
196E	LINE-ADDR	79
1980	Compare line numbers	79
1988	Find each statement	79
19B8	NEXT-ONE	80
19DD	Difference	81
19E5	Reclaiming	81
19FB	E-LINE-NO	82
1A1B	Report and line number printing	82

BASIC LINE AND COMMAND INTERPRETATION

1A48	Syntax tables	84
1B17	Main parser (BASIC interpreter)	86
1B28	Statement loop	87
1B52	SCAN-LOOP	87
1B6F	SEPARATOR	88
1B76	STMT-RET	88
1B8A	LINE-RUN	88
1B9E	LINE-NEW	88
1BB2	REM command	89
1BB3	LINE-END	89
1BBF	LINE-USE	89
1BD1	NEXT-LINE	89
1BEE	CHECK-END	90
1BF4	STMT-NEXT	90
1C01	Command class table	90
1C0D	Command classes - 00, 03 & 05	90
1C16	JUMP-C-R	91
1C1F	Command classes - 01, 02 & 04	91
1C22	Variable In assignment	91
1C56	Fetch a value	92
1C79	Expect numeric/string expressions	93
1C96	Set permanent colours (class 07)	93
1CBE	Command class - 09	94
1CDB	Command class - 0B	94
1CDE	Fetch a number	94
1CEE	STOP command	95
1CF0	IF command	95
1D03	FOR command	95
1D86	LOOK-PROG	96
1DAB	NEXT command	97
1DDA	NEXT-LOOP	97
1DEC	READ command	99
1E27	DATA command	100
1E39	PASS-BY	100
1E42	RESTORE command	100

address	routine	page
1E4F	RANDOMIZE command	100
1E5F	CONTINUE command	101
1E67	GO TO command	101
1E7A	OUT command	101
1E80	POKE command	101
1E85	TWO-PARAM	101
1E94	Find integers	101
1EA1	RUN command	102
1EAC	CLEAR command	102
1EED	GO SUB command	103
1F05	TEST-ROOM	103
1F1A	Free memory	103
1F23	RETURN command	104
1F3A	PAUSE command	104
1F54	BREAK-KEY	104
1F60	DEF FN command	105
1FC3	UNSTACK-Z	106
1FC9	LPRINT command	106
1FCF	PRINT command	106
1FF5	Print a carriage return	107
1FFC	Print items	107
2045	End of printing	108
204E	Print position	108
2070	Alter stream	108
2089	INPUT command	109
21B9	IN-ASSIGN	111
21D6	IN-CHAN-K	112
21E1	Colour item routines	112
226C	CO-CHANGE	114
2294	BORDER command	115
22AA	Pixel address	115
22CB	Point	116
22DC	PLOT command	116
2307	STK-TO-BC	117
2314	STK-TO-A	117
2320	CIRCLE command	117
2382	DRAW command	119
247D	Initial parameters	123
24B7	Line drawing	124

EXPRESSION EVALUATION

24FB	SCANNING	127
2530	SYNTAX-Z	128
2535	Scanning SCREEN\$	128
2580	Scanning ATTR	129
2596	Scanning function table	129
25AF	Scanning function routines	130
26C9	Scanning variable routine	133
2734	Scanning main loop	135
2795	Table of operators	137
27B0	Table of priorities	137
27BD	Scanning function (FN)	137
28AB	FN-SKPOVR	141
28B2	LOOK-VARS	141
2951	Stack function argument	144
2996	STK-VAR	145
2A52	SLICING	148
2AB6	STK-STORE	150
2ACC	INT-EXP	150
2AEE	DE,(DE+1)	151

address	routine	page
2AF4	GET-HL*DE	151
2AFF	LET command	151
2BF1	STK-FETCH	157
2C02	DIM command	157
2C88	ALPHANUM	159
2C8D	ALPHA	159
2C9B	Decimal to floating-point	160
2D1B	NUMERIC	161
2D22	STK-DIGIT	162
2D28	STACK-A	162
2D2B	STACK-BC	162
2D3B	Integer to floating-point	162

THE ARITHMETIC ROUTINES

2D4F	E-format to floating-point	164
2D7F	INT-FETCH	165
2D8E	INT-STORE	165
2DA2	Floating-point to BC	166
2DC1	LOG (2^A)	166
2DD5	Floating-point to A	167
2DE3	Print a floating-point number	167
2F8B	CA=10*A+C	173
2F9B	Prepare to add	174
2FBA	Fetch two numbers	174
2FDD	Shift addend	175
3004	ADD-BACK	176
300F	Subtraction (03)	176
3014	Addition (0F)	176
30A9	HL=HL*DE	179
30C0	Prepare to multiply or divide	180
30CA	Multiplication (04)	180
31AF	Division (05)	184
3214	Integer truncation towards zero (3A)	186
3293	Re-stack two	188
3297	RE-STACK (3D)	188

THE FLOATING-POINT CALCULATOR

32C5	Table of constants	190
32D7	Table of addresses	190
335B	CALCULATE	192
33A1	Delete (02)	194
33A2	Single operation (3B)	194
33A9	Test 5-spaces	194
33B4	Stack number	194
33C0	Move a floating-point number (31)	195
33C6	Stack literals (34)	195
33F7	Skip constants	196
3406	Memory location	196
340F	Get from memory area (E0 etc.)	197
341B	Stack a constant (A0 etc.)	197
342D	Store in memory area (C0 etc.)	197
343C	EXCHANGE (01)	198
3449	Series generator (86 etc.)	198
346A	Absolute magnitude (2A)	199
346E	Unary minus (1B)	199
3492	Signum (29)	200
34A5	IN (2C)	200
34AC	PEEK (2B)	201
34B3	USR number (2D)	201
34BC	USR string (19)	201

address	routine	page
34E9	TEST-ZERO	202
34F9	Greater than zero (37)	202
3501	NOT (30)	202
3506	Less than zero (36)	203
350B	Zero or one	203
351B	OR (07)	203
3524	Number AND number (08)	203
352D	String AND number (10)	204
353B	Comparison (09-0E, 11-16)	204
359C	String concatenation (17)	205
35BF	STK-PNTRS	206
35C9	CHR\$ (2F)	206
35DE	VAL and VAL\$ (1D,18)	207
361F	STR\$ (2E)	208
3645	Read-in (1A)	208
3669	CODE (1C)	209
3674	LEN (1E)	209
367A	Decrease the counter (35)	209
3686	Jump (33)	209
368F	Jump on true (00)	210
369B	END-CALC (38)	210
36A0	Modulus (32)	210
36AF	INT (27)	211
36C4	Exponential (26)	211
3713	Natural logarithm (25)	213
3783	Reduce argument (39)	215
37AA	Cosine (20)	216
37B5	SINE (1 F)	216
37DA	Tan (21)	217
37E2	ARCTAN (24)	218
3833	Arcsin (22)	219
3843	Arccos (23)	220
384A	Square root (28)	220
3851	Exponentiation (06)	220

APPENDIX

BASIC programs for the main series.

- Series generator	222
- SIN X	223
- EXP X	224
- LN X	226
- ATN X	228
The 'DRAW' algorithm	228
The 'CIRCLE' algorithm	229
Note on Small Integers and -65536	229

Spectrum ROM Disassembly

Customer Registration Card

Please fill out this page (or a photocopy of it) and return it so that we may keep you informed of new books, software and special offers. Post to the appropriate address on the back.

Date..... 19.....

Name.....

Street & No.....

City.....Postcode/Zipcode.....

Model of computer owned.....

Where did you learn of this book:

☐ FRIEND

☐ RETAIL SHOP

☐ MAGAZINE (give name)

☐ OTHER (specify)

Age?

☐ 10-15

☐ 16-19

☐ 20-24

☐ 25 and over

How would you rate this book?

QUALITY: ☐ Excellent ☐ Good

☐ Poor

VALUE: ☐ Overpriced

☐ Good

☐ Underpriced

What other books and software would you like to see produced for your computer?

.....

.....

.....

EDITION 7 6 5 4 3 2 1

Melbourne House addresses

Put this Registration Card (or photocopy) in an envelope and post it to the appropriate address:

United Kingdom

Melbourne House (Publishers) Ltd
Castle Yard House
Castle Yard
Richmond, TW10 6TF

Australia and New Zealand

Melbourne House (Australia) Pty Ltd
2nd Floor, 70 Park Street
South Melbourne, Victoria 3205