```
;************************************************************************
;** An Assembly File Listing to generate a 16K ROM for the ZX Spectrum **
;************************************************************************

; -------------------------
; Last updated: 09-AUG-2003
; -------------------------

; TASM cross-assembler directives.
; ( comment out, perhaps, for other assemblers - see Notes at end.)

#define DEFB .BYTE
#define DEFW .WORD
#define DEFM .TEXT
#define ORG  .ORG
#define EQU  .EQU
#define equ  .EQU

;   It is always a good idea to anchor, using ORGs, important sections such as
;   the character bitmaps so that they don't move as code is added and removed.

;   Generally most approaches try to maintain main entry points as they are
;   often used by third-party software. The Sinclair Interface 1 ROM written
;   by Dr. Ian Logan and Martin Brennan calls numerous routines in this ROM.
;   Non-standard entry points have a label beginning with X.

; ORG 0000

;*****************************************
;** Part 1. RESTART ROUTINES AND TABLES **
;*****************************************

; -----------
; THE 'START'
; -----------
;   At switch on, the Z80 chip is in Interrupt Mode 0.
;   The Spectrum uses Interrupt Mode 1.
;   This location can also be 'called' to reset the machine.
;   Typically with PRINT USR 0.

;; START
L0000:  DI                      ; Disable Interrupts.
        XOR     A               ; Signal coming from START.
        LD      DE,$FFFF        ; Set pointer to top of possible physical RAM.
        JP      L11CB           ; Jump forward to common code at START-NEW.

; -------------------
; THE 'ERROR' RESTART
; -------------------
;   The error pointer is made to point to the position of the error to enable
;   the editor to highlight the error position if it occurred during syntax
;   checking.  It is used at 37 places in the program.  An instruction fetch
;   on address $0008 may page in a peripheral ROM such as the Sinclair
;   Interface 1 or Disciple Disk Interface.  This was not an original design
;   concept and not all errors pass through here.

;; ERROR-1
L0008:  LD      HL,($5C5D)      ; Fetch the character address from CH_ADD.
        LD      ($5C5F),HL      ; Copy it to the error pointer X_PTR.
        JR      L0053           ; Forward to continue at ERROR-2.

; -----------------------------
; THE 'PRINT CHARACTER' RESTART
; -----------------------------
```

```
;   The A register holds the code of the character that is to be sent to
;   the output stream of the current channel.  The alternate register set is
;   used to output a character in the A register so there is no need to
;   preserve any of the current main registers (HL, DE, BC).
;   This restart is used 21 times.

;; PRINT-A
L0010:  JP      L15F2           ; Jump forward to continue at PRINT-A-2.

; ---

        DEFB    $FF, $FF, $FF   ; Five unused locations.
        DEFB    $FF, $FF        ;

; -------------------------------
; 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 character that has
;   relevance to the BASIC parser. Otherwise CH_ADD is incremented and the
;   tests repeated. CH_ADD will be addressing somewhere -
;   1) in the BASIC program area during line execution.
;   2) in workspace if evaluating, for example, a string expression.
;   3) in the edit buffer if parsing a direct command or a new BASIC line.
;   4) in workspace if accepting input but not that from INPUT LINE.

;; GET-CHAR
L0018:  LD      HL,($5C5D)      ; fetch the address from CH_ADD.
        LD      A,(HL)          ; use it to pick up current character.

;; TEST-CHAR
L001C:  CALL    L007D           ; routine SKIP-OVER tests if the character is
                                ; relevant.
        RET     NC              ; Return if it is significant.

; ------------------------------------
; THE 'COLLECT NEXT CHARACTER' RESTART
; ------------------------------------
;   As the BASIC commands and expressions are interpreted, this routine is
;   called repeatedly to step along the line.  It is used 83 times.

;; NEXT-CHAR
L0020:  CALL    L0074           ; routine CH-ADD+1 fetches the next immediate
                                ; character.
        JR      L001C           ; jump back to TEST-CHAR until a valid
                                ; character is found.

; ---

        DEFB    $FF, $FF, $FF   ; unused

; -----------------------
; THE 'CALCULATE' RESTART
; -----------------------
;   This restart enters the Spectrum's internal, floating-point, stack-based,
;   FORTH-like language.
;   It is further used recursively from within the calculator.
;   It is used on 77 occasions.

;; FP-CALC
L0028:  JP      L335B           ; jump forward to the CALCULATE routine.

; ---
```

```
        DEFB    $FF, $FF, $FF   ; spare - note that on the ZX81, space being a
        DEFB    $FF, $FF        ; little cramped, these same locations were
                                ; used for the five-byte end-calc literal.

; ------------------------------
; THE 'CREATE BC SPACES' RESTART
; ------------------------------
;   This restart is used on only 12 occasions to create BC spaces
;   between workspace and the calculator stack.

;; BC-SPACES
L0030:  PUSH    BC              ; Save number of spaces.
        LD      HL,($5C61)      ; Fetch WORKSP.
        PUSH    HL              ; Save address of workspace.
        JP      L169E           ; Jump forward to continuation code RESERVE.

; --------------------------------
; THE 'MASKABLE INTERRUPT' ROUTINE
; --------------------------------
;   This routine increments the Spectrum's three-byte FRAMES counter fifty
;   times a second (sixty times a second in the USA ).
;   Both this routine and the called KEYBOARD subroutine use the IY register
;   to access system variables and flags so a user-written program must
;   disable interrupts to make use of the IY register.

;; MASK-INT
L0038:  PUSH    AF              ; Save the registers that will be used but not
        PUSH    HL              ; the IY register unfortunately.
        LD      HL,($5C78)      ; Fetch the first two bytes at FRAMES1.
        INC     HL              ; Increment lowest two bytes of counter.
        LD      ($5C78),HL      ; Place back in FRAMES1.
        LD      A,H             ; Test if the result was zero.
        OR      L               ;
        JR      NZ,L0048        ; Forward, if not, to KEY-INT

        INC     (IY+$40)        ; otherwise increment FRAMES3 the third byte.

;   Now save the rest of the main registers and read and decode the keyboard.

;; KEY-INT
L0048:  PUSH    BC              ; Save the other main registers.
        PUSH    DE              ;

        CALL    L02BF           ; Routine KEYBOARD executes a stage in the
                                ; process of reading a key-press.
        POP     DE              ;
        POP     BC              ; Restore registers.

        POP     HL              ;
        POP     AF              ;

        EI                      ; Enable Interrupts.
        RET                     ; Return.

; ---------------------
; THE 'ERROR-2' ROUTINE
; ---------------------
;   A continuation of the code at 0008.
;   The error code is stored and after clearing down stacks, an indirect jump
;   is made to MAIN-4, etc. to handle the error.

;; ERROR-2
L0053:  POP     HL              ; drop the return address - the location
                                ; after the RST 08H instruction.
```

```
        LD      L,(HL)          ; fetch the error code that follows.
                                ; (nice to see this instruction used.)

;   Note. this entry point is used when out of memory at REPORT-4.
;   The L register has been loaded with the report code but X-PTR is not
;   updated.

;; ERROR-3
L0055:  LD      (IY+$00),L      ; Store it in the system variable ERR_NR.
        LD      SP,($5C3D)      ; ERR_SP points to an error handler on the
                                ; machine stack. There may be a hierarchy
                                ; of routines.
                                ; To MAIN-4 initially at base.
                                ; or REPORT-G on line entry.
                                ; or  ED-ERROR when editing.
                                ; or   ED-FULL during ed-enter.
                                ; or  IN-VAR-1 during runtime input etc.

        JP      L16C5           ; Jump to SET-STK to clear the calculator stack
                                ; and reset MEM to usual place in the systems
                                ; variables area and then indirectly to MAIN-4,
                                ; etc.

; ---

        DEFB    $FF, $FF, $FF   ; Unused locations
        DEFB    $FF, $FF, $FF   ; before the fixed-position
        DEFB    $FF             ; NMI routine.

; ------------------------------------
; THE 'NON-MASKABLE INTERRUPT' ROUTINE
; ------------------------------------
;   New
;   There is no NMI switch on the standard Spectrum or its peripherals.
;   When the NMI line is held low, then no matter what the Z80 was doing at
;   the time, it will now execute the code at 66 Hex.
;   This Interrupt Service Routine will jump to location zero if the contents
;   of the system variable NMIADD are zero or return if the location holds a
;   non-zero address.   So attaching a simple switch to the NMI as in the book
;   "Spectrum Hardware Manual" causes a reset.  The logic was obviously
;   intended to work the other way.  Sinclair Research said that, since they
;   had never advertised the NMI, they had no plans to fix the error "until
;   the opportunity arose".
;
;   Note. The location NMIADD was, in fact, later used by Sinclair Research
;   to enhance the text channel on the ZX Interface 1.
;   On later Amstrad-made Spectrums, and the Brazilian Spectrum, the logic of
;   this routine was indeed reversed but not as at first intended.
;
;   It can be deduced by looking elsewhere in this ROM that the NMIADD system
;   variable pointed to L121C and that this enabled a Warm Restart to be
;   performed at any time, even while playing machine code games, or while
;   another Spectrum has been allowed to gain control of this one.
;
;   Software houses would have been able to protect their games from attack by
;   placing two zeros in the NMIADD system variable.

;; RESET
L0066:  PUSH    AF              ; save the
        PUSH    HL              ; registers.
        LD      HL,($5CB0)      ; fetch the system variable NMIADD.
        LD      A,H             ; test address
        OR      L               ; for zero.
```

```
        JR      NZ,L0070        ; skip to NO-RESET if NOT ZERO

        JP      (HL)            ; jump to routine ( i.e. L0000 )

;; NO-RESET
L0070:  POP     HL              ; restore the
        POP     AF              ; registers.
        RETN                    ; return to previous interrupt state.

; ---------------------------
; THE 'CH ADD + 1' SUBROUTINE
; ---------------------------
;   This subroutine is called from RST 20, and three times from elsewhere
;   to fetch the next immediate character following the current valid character
;   address and update the associated system variable.
;   The entry point TEMP-PTR1 is used from the SCANNING routine.
;   Both TEMP-PTR1 and TEMP-PTR2 are used by the READ command routine.

;; CH-ADD+1
L0074:  LD      HL,($5C5D)      ; fetch address from CH_ADD.

;; TEMP-PTR1
L0077:  INC     HL              ; increase the character address by one.

;; TEMP-PTR2
L0078:  LD      ($5C5D),HL      ; update CH_ADD with character address.

X007B:  LD      A,(HL)          ; load character to A from HL.
        RET                     ; and return.

; --------------------------
; THE 'SKIP OVER' SUBROUTINE
; --------------------------
;   This subroutine is called once from RST 18 to skip over white-space and
;   other characters irrelevant to the parsing of a BASIC line etc. .
;   Initially the A register holds the character to be considered
;   and HL holds its address which will not be within quoted text
;   when a BASIC line is parsed.
;   Although the 'tab' and 'at' characters will not appear in a BASIC line,
;   they could be present in a string expression, and in other situations.
;   Note. although white-space is usually placed in a program to indent loops
;   and make it more readable, it can also be used for the opposite effect and
;   spaces may appear in variable names although the parser never sees them.
;   It is this routine that helps make the variables 'Anum bEr5 3BUS' and
;   'a number 53 bus' appear the same to the parser.

;; SKIP-OVER
L007D:  CP      $21             ; test if higher than space.
        RET     NC              ; return with carry clear if so.

        CP      $0D             ; carriage return ?
        RET     Z               ; return also with carry clear if so.

                                ; all other characters have no relevance
                                ; to the parser and must be returned with
                                ; carry set.

        CP      $10             ; test if 0-15d
        RET     C               ; return, if so, with carry set.

        CP      $18             ; test if 24-32d
        CCF                     ; complement carry flag.
        RET     C               ; return with carry set if so.
```

```
                                ; now leaves 16d-23d

        INC     HL              ; all above have at least one extra character
                                ; to be stepped over.

        CP      $16             ; controls 22d ('at') and 23d ('tab') have two.
        JR      C,L0090         ; forward to SKIPS with ink, paper, flash,
                                ; bright, inverse or over controls.
                                ; Note. the high byte of tab is for RS232 only.
                                ; it has no relevance on this machine.

        INC     HL              ; step over the second character of 'at'/'tab'.

;; SKIPS
L0090:  SCF                     ; set the carry flag
        LD      ($5C5D),HL      ; update the CH_ADD system variable.
        RET                     ; return with carry set.


; ------------------
; THE 'TOKEN' TABLES
; ------------------
;   The tokenized characters 134d (RND) to 255d (COPY) are expanded using
;   this table. The last byte of a token is inverted to denote the end of
;   the word. The first is an inverted step-over byte.

;; TKN-TABLE
L0095:  DEFB    '?'+$80
        DEFM    "RN"
        DEFB    'D'+$80
        DEFM    "INKEY"
        DEFB    '$'+$80
        DEFB    'P','I'+$80
        DEFB    'F','N'+$80
        DEFM    "POIN"
        DEFB    'T'+$80
        DEFM    "SCREEN"
        DEFB    '$'+$80
        DEFM    "ATT"
        DEFB    'R'+$80
        DEFB    'A','T'+$80
        DEFM    "TA"
        DEFB    'B'+$80
        DEFM    "VAL"
        DEFB    '$'+$80
        DEFM    "COD"
        DEFB    'E'+$80
        DEFM    "VA"
        DEFB    'L'+$80
        DEFM    "LE"
        DEFB    'N'+$80
        DEFM    "SI"
        DEFB    'N'+$80
        DEFM    "CO"
        DEFB    'S'+$80
        DEFM    "TA"
        DEFB    'N'+$80
        DEFM    "AS"
        DEFB    'N'+$80
        DEFM    "AC"
        DEFB    'S'+$80
        DEFM    "AT"
        DEFB    'N'+$80
        DEFB    'L','N'+$80
```

```
        DEFM    "EX"
        DEFB    'P'+$80
        DEFM    "IN"
        DEFB    'T'+$80
        DEFM    "SQ"
        DEFB    'R'+$80
        DEFM    "SG"
        DEFB    'N'+$80
        DEFM    "AB"
        DEFB    'S'+$80
        DEFM    "PEE"
        DEFB    'K'+$80
        DEFB    'I','N'+$80
        DEFM    "US"
        DEFB    'R'+$80
        DEFM    "STR"
        DEFB    '$'+$80
        DEFM    "CHR"
        DEFB    '$'+$80
        DEFM    "NO"
        DEFB    'T'+$80
        DEFM    "BI"
        DEFB    'N'+$80

;   The previous 32 function-type words are printed without a leading space
;   The following have a leading space if they begin with a letter

        DEFB    'O','R'+$80
        DEFM    "AN"
        DEFB    'D'+$80
        DEFB    $3C,'='+$80             ; <=
        DEFB    $3E,'='+$80             ; >=
        DEFB    $3C,$3E+$80             ; <>
        DEFM    "LIN"
        DEFB    'E'+$80
        DEFM    "THE"
        DEFB    'N'+$80
        DEFB    'T','O'+$80
        DEFM    "STE"
        DEFB    'P'+$80
        DEFM    "DEF F"
        DEFB    'N'+$80
        DEFM    "CA"
        DEFB    'T'+$80
        DEFM    "FORMA"
        DEFB    'T'+$80
        DEFM    "MOV"
        DEFB    'E'+$80
        DEFM    "ERAS"
        DEFB    'E'+$80
        DEFM    "OPEN "
        DEFB    '#'+$80
        DEFM    "CLOSE "
        DEFB    '#'+$80
        DEFM    "MERG"
        DEFB    'E'+$80
        DEFM    "VERIF"
        DEFB    'Y'+$80
        DEFM    "BEE"
        DEFB    'P'+$80
        DEFM    "CIRCL"
        DEFB    'E'+$80
        DEFM    "IN"
        DEFB    'K'+$80
```

```
DEFM    "PAPE"
DEFB    'R'+$80
DEFM    "FLAS"
DEFB    'H'+$80
DEFM    "BRIGH"
DEFB    'T'+$80
DEFM    "INVERS"
DEFB    'E'+$80
DEFM    "OVE"
DEFB    'R'+$80
DEFM    "OU"
DEFB    'T'+$80
DEFM    "LPRIN"
DEFB    'T'+$80
DEFM    "LLIS"
DEFB    'T'+$80
DEFM    "STO"
DEFB    'P'+$80
DEFM    "REA"
DEFB    'D'+$80
DEFM    "DAT"
DEFB    'A'+$80
DEFM    "RESTOR"
DEFB    'E'+$80
DEFM    "NE"
DEFB    'W'+$80
DEFM    "BORDE"
DEFB    'R'+$80
DEFM    "CONTINU"
DEFB    'E'+$80
DEFM    "DI"
DEFB    'M'+$80
DEFM    "RE"
DEFB    'M'+$80
DEFM    "FO"
DEFB    'R'+$80
DEFM    "GO T"
DEFB    'O'+$80
DEFM    "GO SU"
DEFB    'B'+$80
DEFM    "INPU"
DEFB    'T'+$80
DEFM    "LOA"
DEFB    'D'+$80
DEFM    "LIS"
DEFB    'T'+$80
DEFM    "LE"
DEFB    'T'+$80
DEFM    "PAUS"
DEFB    'E'+$80
DEFM    "NEX"
DEFB    'T'+$80
DEFM    "POK"
DEFB    'E'+$80
DEFM    "PRIN"
DEFB    'T'+$80
DEFM    "PLO"
DEFB    'T'+$80
DEFM    "RU"
DEFB    'N'+$80
DEFM    "SAV"
DEFB    'E'+$80
DEFM    "RANDOMIZ"
DEFB    'E'+$80
```

```
        DEFB    'I','F'+$80
        DEFM    "CL"
        DEFB    'S'+$80
        DEFM    "DRA"
        DEFB    'W'+$80
        DEFM    "CLEA"
        DEFB    'R'+$80
        DEFM    "RETUR"
        DEFB    'N'+$80
        DEFM    "COP"
        DEFB    'Y'+$80

; ----------------
; THE 'KEY' TABLES
; ----------------
;   These six look-up tables are used by the keyboard reading routine
;   to decode the key values.
;
;   The first table contains the maps for the 39 keys of the standard
;   40-key Spectrum keyboard. The remaining key [SHIFT $27] is read directly.
;   The keys consist of the 26 upper-case alphabetic characters, the 10 digit
;   keys and the space, ENTER and symbol shift key.
;   Unshifted alphabetic keys have $20 added to the value.
;   The keywords for the main alphabetic keys are obtained by adding $A5 to
;   the values obtained from this table.

;; MAIN-KEYS
L0205:  DEFB    $42             ; B
        DEFB    $48             ; H
        DEFB    $59             ; Y
        DEFB    $36             ; 6
        DEFB    $35             ; 5
        DEFB    $54             ; T
        DEFB    $47             ; G
        DEFB    $56             ; V
        DEFB    $4E             ; N
        DEFB    $4A             ; J
        DEFB    $55             ; U
        DEFB    $37             ; 7
        DEFB    $34             ; 4
        DEFB    $52             ; R
        DEFB    $46             ; F
        DEFB    $43             ; C
        DEFB    $4D             ; M
        DEFB    $4B             ; K
        DEFB    $49             ; I
        DEFB    $38             ; 8
        DEFB    $33             ; 3
        DEFB    $45             ; E
        DEFB    $44             ; D
        DEFB    $58             ; X
        DEFB    $0E             ; SYMBOL SHIFT
        DEFB    $4C             ; L
        DEFB    $4F             ; O
        DEFB    $39             ; 9
        DEFB    $32             ; 2
        DEFB    $57             ; W
        DEFB    $53             ; S
        DEFB    $5A             ; Z
        DEFB    $20             ; SPACE
        DEFB    $0D             ; ENTER
        DEFB    $50             ; P
        DEFB    $30             ; 0
        DEFB    $31             ; 1
```

```
        DEFB    $51             ; Q
        DEFB    $41             ; A


;; E-UNSHIFT
;  The 26 unshifted extended mode keys for the alphabetic characters.
;  The green keywords on the original keyboard.
L022C:  DEFB    $E3             ; READ
        DEFB    $C4             ; BIN
        DEFB    $E0             ; LPRINT
        DEFB    $E4             ; DATA
        DEFB    $B4             ; TAN
        DEFB    $BC             ; SGN
        DEFB    $BD             ; ABS
        DEFB    $BB             ; SQR
        DEFB    $AF             ; CODE
        DEFB    $B0             ; VAL
        DEFB    $B1             ; LEN
        DEFB    $C0             ; USR
        DEFB    $A7             ; PI
        DEFB    $A6             ; INKEY$
        DEFB    $BE             ; PEEK
        DEFB    $AD             ; TAB
        DEFB    $B2             ; SIN
        DEFB    $BA             ; INT
        DEFB    $E5             ; RESTORE
        DEFB    $A5             ; RND
        DEFB    $C2             ; CHR$
        DEFB    $E1             ; LLIST
        DEFB    $B3             ; COS
        DEFB    $B9             ; EXP
        DEFB    $C1             ; STR$
        DEFB    $B8             ; LN


;; EXT-SHIFT
;  The 26 shifted extended mode keys for the alphabetic characters.
;  The red keywords below keys on the original keyboard.
L0246:  DEFB    $7E             ; ~
        DEFB    $DC             ; BRIGHT
        DEFB    $DA             ; PAPER
        DEFB    $5C             ; \
        DEFB    $B7             ; ATN
        DEFB    $7B             ; {
        DEFB    $7D             ; }
        DEFB    $D8             ; CIRCLE
        DEFB    $BF             ; IN
        DEFB    $AE             ; VAL$
        DEFB    $AA             ; SCREEN$
        DEFB    $AB             ; ATTR
        DEFB    $DD             ; INVERSE
        DEFB    $DE             ; OVER
        DEFB    $DF             ; OUT
        DEFB    $7F             ; (Copyright character)
        DEFB    $B5             ; ASN
        DEFB    $D6             ; VERIFY
        DEFB    $7C             ; |
        DEFB    $D5             ; MERGE
        DEFB    $5D             ; ]
        DEFB    $DB             ; FLASH
        DEFB    $B6             ; ACS
        DEFB    $D9             ; INK
        DEFB    $5B             ; [
        DEFB    $D7             ; BEEP
```

```
;; CTL-CODES
;  The ten control codes assigned to the top line of digits when the shift
;  key is pressed.
L0260:  DEFB    $0C             ; DELETE
        DEFB    $07             ; EDIT
        DEFB    $06             ; CAPS LOCK
        DEFB    $04             ; TRUE VIDEO
        DEFB    $05             ; INVERSE VIDEO
        DEFB    $08             ; CURSOR LEFT
        DEFB    $0A             ; CURSOR DOWN
        DEFB    $0B             ; CURSOR UP
        DEFB    $09             ; CURSOR RIGHT
        DEFB    $0F             ; GRAPHICS


;; SYM-CODES
;  The 26 red symbols assigned to the alphabetic characters of the keyboard.
;  The ten single-character digit symbols are converted without the aid of
;  a table using subtraction and minor manipulation.
L026A:  DEFB    $E2             ; STOP
        DEFB    $2A             ; *
        DEFB    $3F             ; ?
        DEFB    $CD             ; STEP
        DEFB    $C8             ; >=
        DEFB    $CC             ; TO
        DEFB    $CB             ; THEN
        DEFB    $5E             ; ^
        DEFB    $AC             ; AT
        DEFB    $2D             ; -
        DEFB    $2B             ; +
        DEFB    $3D             ; =
        DEFB    $2E             ; .
        DEFB    $2C             ; ,
        DEFB    $3B             ; ;
        DEFB    $22             ; "
        DEFB    $C7             ; <=
        DEFB    $3C             ; <
        DEFB    $C3             ; NOT
        DEFB    $3E             ; >
        DEFB    $C5             ; OR
        DEFB    $2F             ; /
        DEFB    $C9             ; <>
        DEFB    $60             ; pound
        DEFB    $C6             ; AND
        DEFB    $3A             ; :

;; E-DIGITS
;  The ten keywords assigned to the digits in extended mode.
;  The remaining red keywords below the keys.
L0284:  DEFB    $D0             ; FORMAT
        DEFB    $CE             ; DEF FN
        DEFB    $A8             ; FN
        DEFB    $CA             ; LINE
        DEFB    $D3             ; OPEN #
        DEFB    $D4             ; CLOSE #
        DEFB    $D1             ; MOVE
        DEFB    $D2             ; ERASE
        DEFB    $A9             ; POINT
        DEFB    $CF             ; CAT


;*******************************
```

```
;** Part 2. KEYBOARD ROUTINES **
;******************************

;   Using shift keys and a combination of modes the Spectrum 40-key keyboard
;   can be mapped to 256 input characters

; ---------------------------------------------------------------------------
;
;         0     1     2     3     4 -Bits-  4     3     2     1     0
; PORT                                                                    PORT
;
; F7FE  [ 1 ] [ 2 ] [ 3 ] [ 4 ] [ 5 ]  |  [ 6 ] [ 7 ] [ 8 ] [ 9 ] [ 0 ]   EFFE
;  ^                                   |                                   v
; FBFE  [ Q ] [ W ] [ E ] [ R ] [ T ]  |  [ Y ] [ U ] [ I ] [ O ] [ P ]   DFFE
;  ^                                   |                                   v
; FDFE  [ A ] [ S ] [ D ] [ F ] [ G ]  |  [ H ] [ J ] [ K ] [ L ] [ ENT ] BFFE
;  ^                                   |                                   v
; FEFE  [SHI] [ Z ] [ X ] [ C ] [ V ]  |  [ B ] [ N ] [ M ] [sym] [ SPC ] 7FFE
;  ^     $27                                                 $18           v
; Start                                                                   End
;        00100111                                            00011000
;
; ---------------------------------------------------------------------------
;   The above map may help in reading.
;   The neat arrangement of ports means that the B register need only be
;   rotated left to work up the left hand side and then down the right
;   hand side of the keyboard. When the reset bit drops into the carry
;   then all 8 half-rows have been read. Shift is the first key to be
;   read. The lower six bits of the shifts are unambiguous.

; -------------------------------
; THE 'KEYBOARD SCANNING' ROUTINE
; -------------------------------
;   From keyboard and s-inkey$
;   Returns 1 or 2 keys in DE, most significant shift first if any
;   key values 0-39 else 255

;; KEY-SCAN
L028E:  LD      L,$2F           ; initial key value
                                ; valid values are obtained by subtracting
                                ; eight five times.
        LD      DE,$FFFF        ; a buffer to receive 2 keys.

        LD      BC,$FEFE        ; the commencing port address
                                ; B holds 11111110 initially and is also
                                ; used to count the 8 half-rows
;; KEY-LINE
L0296:  IN      A,(C)           ; read the port to A - bits will be reset
                                ; if a key is pressed else set.
        CPL                     ; complement - pressed key-bits are now set
        AND     $1F             ; apply 00011111 mask to pick up the
                                ; relevant set bits.

        JR      Z,L02AB         ; forward to KEY-DONE if zero and therefore
                                ; no keys pressed in row at all.

        LD      H,A             ; transfer row bits to H
        LD      A,L             ; load the initial key value to A

;; KEY-3KEYS
L029F:  INC     D               ; now test the key buffer
        RET     NZ              ; if we have collected 2 keys already
                                ; then too many so quit.
```

```
;; KEY-BITS
L02A1:  SUB     $08             ; subtract 8 from the key value
                                ; cycling through key values (top = $27)
                                ; e.g. 2F>   27>1F>17>0F>07
                                ;      2E>   26>1E>16>0E>06
        SRL     H               ; shift key bits right into carry.
        JR      NC,L02A1        ; back to KEY-BITS if not pressed
                                ; but if pressed we have a value (0-39d)

        LD      D,E             ; transfer a possible previous key to D
        LD      E,A             ; transfer the new key to E
        JR      NZ,L029F        ; back to KEY-3KEYS if there were more
                                ; set bits - H was not yet zero.

;; KEY-DONE
L02AB:  DEC     L               ; cycles 2F>2E>2D>2C>2B>2A>29>28 for
                                ; each half-row.
        RLC     B               ; form next port address e.g. FEFE > FDFE
        JR      C,L0296         ; back to KEY-LINE if still more rows to do.

        LD      A,D             ; now test if D is still FF ?
        INC     A               ; if it is zero we have at most 1 key
                                ; range now $01-$28  (1-40d)
        RET     Z               ; return if one key or no key.

        CP      $28             ; is it capsshift (was $27) ?
        RET     Z               ; return if so.

        CP      $19             ; is it symbol shift (was $18) ?
        RET     Z               ; return also

        LD      A,E             ; now test E
        LD      E,D             ; but first switch
        LD      D,A             ; the two keys.
        CP      $18             ; is it symbol shift ?
        RET                     ; return (with zero set if it was).
                                ; but with symbol shift now in D

; ----------------------
; THE 'KEYBOARD' ROUTINE
; ----------------------
;   Called from the interrupt 50 times a second.
;

;; KEYBOARD
L02BF:  CALL    L028E           ; routine KEY-SCAN
        RET     NZ              ; return if invalid combinations

;   then decrease the counters within the two key-state maps
;   as this could cause one to become free.
;   if the keyboard has not been pressed during the last five interrupts
;   then both sets will be free.


        LD      HL,$5C00        ; point to KSTATE-0

;; K-ST-LOOP
L02C6:  BIT     7,(HL)          ; is it free ?  (i.e. $FF)
        JR      NZ,L02D1        ; forward to K-CH-SET if so

        INC     HL              ; address the 5-counter
        DEC     (HL)            ; decrease the counter
        DEC     HL              ; step back
```

```
        JR      NZ,L02D1        ; forward to K-CH-SET if not at end of count

        LD      (HL),$FF        ; else mark this particular map free.

;; K-CH-SET
L02D1:  LD      A,L             ; make a copy of the low address byte.
        LD      HL,$5C04        ; point to KSTATE-4
                                ; (ld l,$04 would do)
        CP      L               ; have both sets been considered ?
        JR      NZ,L02C6        ; back to K-ST-LOOP to consider this 2nd set

;   now the raw key (0-38d) is converted to a main key (uppercase).

        CALL    L031E           ; routine K-TEST to get main key in A

        RET     NC              ; return if just a single shift

        LD      HL,$5C00        ; point to KSTATE-0
        CP      (HL)            ; does the main key code match ?
        JR      Z,L0310         ; forward to K-REPEAT if so

;   if not consider the second key map.

        EX      DE,HL           ; save kstate-0 in de
        LD      HL,$5C04        ; point to KSTATE-4
        CP      (HL)            ; does the main key code match ?
        JR      Z,L0310         ; forward to K-REPEAT if so

;   having excluded a repeating key we can now consider a new key.
;   the second set is always examined before the first.

        BIT     7,(HL)          ; is the key map free ?
        JR      NZ,L02F1        ; forward to K-NEW if so.

        EX      DE,HL           ; bring back KSTATE-0
        BIT     7,(HL)          ; is it free ?
        RET     Z               ; return if not.
                                ; as we have a key but nowhere to put it yet.

;   continue or jump to here if one of the buffers was free.

;; K-NEW
L02F1:  LD      E,A             ; store key in E
        LD      (HL),A          ; place in free location
        INC     HL              ; advance to the interrupt counter
        LD      (HL),$05        ; and initialize counter to 5
        INC     HL              ; advance to the delay
        LD      A,($5C09)       ; pick up the system variable REPDEL
        LD      (HL),A          ; and insert that for first repeat delay.
        INC     HL              ; advance to last location of state map.

        LD      C,(IY+$07)      ; pick up MODE  (3 bytes)
        LD      D,(IY+$01)      ; pick up FLAGS (3 bytes)
        PUSH    HL              ; save state map location
                                ; Note. could now have used, to avoid IY,
                                ; ld l,$41; ld c,(hl); ld l,$3B; ld d,(hl).
                                ; six and two threes of course.

        CALL    L0333           ; routine K-DECODE

        POP     HL              ; restore map pointer
        LD      (HL),A          ; put the decoded key in last location of map.

;; K-END
```

```
L0308:  LD      ($5C08),A       ; update LASTK system variable.
        SET     5,(IY+$01)      ; update FLAGS  - signal a new key.
        RET                     ; return to interrupt routine.

; -----------------------
; THE 'REPEAT KEY' BRANCH
; -----------------------
;   A possible repeat has been identified. HL addresses the raw key.
;   The last location of the key map holds the decoded key from the first
;   context.  This could be a keyword and, with the exception of NOT a repeat
;   is syntactically incorrect and not really desirable.

;; K-REPEAT
L0310:  INC     HL              ; increment the map pointer to second location.
        LD      (HL),$05        ; maintain interrupt counter at 5.
        INC     HL              ; now point to third location.
        DEC     (HL)            ; decrease the REPDEL value which is used to
                                ; time the delay of a repeat key.

        RET     NZ              ; return if not yet zero.

        LD      A,($5C0A)       ; fetch the system variable value REPPER.
        LD      (HL),A          ; for subsequent repeats REPPER will be used.

        INC     HL              ; advance
                                ;
        LD      A,(HL)          ; pick up the key decoded possibly in another
                                ; context.
                                ; Note. should compare with $A5 (RND) and make
                                ; a simple return if this is a keyword.
                                ; e.g. cp $a5; ret nc; (3 extra bytes)
        JR      L0308           ; back to K-END

; ----------------------
; THE 'KEY-TEST' ROUTINE
; ----------------------
;   also called from s-inkey$
;   begin by testing for a shift with no other.

;; K-TEST
L031E:  LD      B,D             ; load most significant key to B
                                ; will be $FF if not shift.
        LD      D,$00           ; and reset D to index into main table
        LD      A,E             ; load least significant key from E
        CP      $27             ; is it higher than 39d   i.e. FF
        RET     NC              ; return with just a shift (in B now)

        CP      $18             ; is it symbol shift ?
        JR      NZ,L032C        ; forward to K-MAIN if not

;   but we could have just symbol shift and no other

        BIT     7,B             ; is other key $FF (ie not shift)
        RET     NZ              ; return with solitary symbol shift


;; K-MAIN
L032C:  LD      HL,L0205        ; address: MAIN-KEYS
        ADD     HL,DE           ; add offset 0-38
        LD      A,(HL)          ; pick up main key value
        SCF                     ; set carry flag
        RET                     ; return    (B has other key still)

; ----------------------------------
```

```
; THE 'KEYBOARD DECODING' SUBROUTINE
; ----------------------------------
;   also called from s-inkey$

;; K-DECODE
L0333:  LD      A,E             ; pick up the stored main key
        CP      $3A             ; an arbitrary point between digits and letters
        JR      C,L0367         ; forward to K-DIGIT with digits, space, enter.

        DEC     C               ; decrease MODE ( 0='KLC', 1='E', 2='G')

        JP      M,L034F         ; to K-KLC-LET if was zero

        JR      Z,L0341         ; to K-E-LET if was 1 for extended letters.

;   proceed with graphic codes.
;   Note. should selectively drop return address if code > 'U' ($55).
;   i.e. abort the KEYBOARD call.
;   e.g. cp 'V'; jr c,addit; pop af ;pop af ;;addit etc. (6 extra bytes).
;   (s-inkey$ never gets into graphics mode.)

;; addit
        ADD     A,$4F           ; add offset to augment 'A' to graphics A say.
        RET                     ; return.
                                ; Note. ( but [GRAPH] V gives RND, etc ).

; ---

;   the jump was to here with extended mode with uppercase A-Z.

;; K-E-LET
L0341:  LD      HL,L022C-$41    ; base address of E-UNSHIFT L022c.
                                ; ( $01EB in standard ROM ).
        INC     B               ; test B is it empty i.e. not a shift.
        JR      Z,L034A         ; forward to K-LOOK-UP if neither shift.

        LD      HL,L0246-$41    ; Address: $0205 L0246-$41 EXT-SHIFT base

;; K-LOOK-UP
L034A:  LD      D,$00           ; prepare to index.
        ADD     HL,DE           ; add the main key value.
        LD      A,(HL)          ; pick up other mode value.
        RET                     ; return.

; ---

;   the jump was here with mode = 0

;; K-KLC-LET
L034F:  LD      HL,L026A-$41    ; prepare base of sym-codes
        BIT     0,B             ; shift=$27 sym-shift=$18
        JR      Z,L034A         ; back to K-LOOK-UP with symbol-shift

        BIT     3,D             ; test FLAGS is it 'K' mode (from OUT-CURS)
        JR      Z,L0364         ; skip to K-TOKENS if so

        BIT     3,(IY+$30)      ; test FLAGS2 - consider CAPS LOCK ?
        RET     NZ              ; return if so with main code.

        INC     B               ; is shift being pressed ?
                                ; result zero if not
        RET     NZ              ; return if shift pressed.

        ADD     A,$20           ; else convert the code to lower case.
```

```
        RET                         ; return.

; ---

;   the jump was here for tokens

;; K-TOKENS
L0364:  ADD     A,$A5               ; add offset to main code so that 'A'
                                    ; becomes 'NEW' etc.

        RET                         ; return.

; ---

;   the jump was here with digits, space, enter and symbol shift (< $xx)

;; K-DIGIT
L0367:  CP      $30                 ; is it '0' or higher ?
        RET     C                   ; return with space, enter and symbol-shift

        DEC     C                   ; test MODE (was 0='KLC', 1='E', 2='G')
        JP      M,L039D             ; jump to K-KLC-DGT if was 0.

        JR      NZ,L0389            ; forward to K-GRA-DGT if mode was 2.

;   continue with extended digits 0-9.

        LD      HL,L0284-$30        ; $0254 - base of E-DIGITS
        BIT     5,B                 ; test - shift=$27 sym-shift=$18
        JR      Z,L034A             ; to K-LOOK-UP if sym-shift

        CP      $38                 ; is character '8' ?
        JR      NC,L0382            ; to K-8-&-9 if greater than '7'

        SUB     $20                 ; reduce to ink range $10-$17
        INC     B                   ; shift ?
        RET     Z                   ; return if not.

        ADD     A,$08               ; add 8 to give paper range $18 - $1F
        RET                         ; return

; ---

;   89

;; K-8-&-9
L0382:  SUB     $36                 ; reduce to 02 and 03  bright codes
        INC     B                   ; test if shift pressed.
        RET     Z                   ; return if not.

        ADD     A,$FE               ; subtract 2 setting carry
        RET                         ; to give 0 and 1    flash codes.

; ---

;   graphics mode with digits

;; K-GRA-DGT
L0389:  LD      HL,L0260-$30        ; $0230 base address of CTL-CODES

        CP      $39                 ; is key '9' ?
        JR      Z,L034A             ; back to K-LOOK-UP - changed to $0F, GRAPHICS.

        CP      $30                 ; is key '0' ?
```

```
        JR      Z,L034A         ; back to K-LOOK-UP - changed to $0C, delete.

;   for keys '0' - '7' we assign a mosaic character depending on shift.

        AND     $07             ; convert character to number. 0 - 7.
        ADD     A,$80           ; add offset - they start at $80

        INC     B               ; destructively test for shift
        RET     Z               ; and return if not pressed.

        XOR     $0F             ; toggle bits becomes range $88-$8F
        RET                     ; return.

; ---

;   now digits in 'KLC' mode

;; K-KLC-DGT
L039D:  INC     B               ; return with digit codes if neither
        RET     Z               ; shift key pressed.

        BIT     5,B             ; test for caps shift.

        LD      HL,L0260-$30    ; prepare base of table CTL-CODES.
        JR      NZ,L034A        ; back to K-LOOK-UP if shift pressed.

;   must have been symbol shift

        SUB     $10             ; for ASCII most will now be correct
                                ; on a standard typewriter.

        CP      $22             ; but '@' is not - see below.
        JR      Z,L03B2         ; forward to K-@-CHAR if so

        CP      $20             ; '_' is the other one that fails
        RET     NZ              ; return if not.

        LD      A,$5F           ; substitute ASCII '_'
        RET                     ; return.

; ---

;; K-@-CHAR
L03B2:  LD      A,$40           ; substitute ASCII '@'
        RET                     ; return.


; ------------------------------------------------------------------------
;   The Spectrum Input character keys. One or two are abbreviated.
;   From $00 Flash 0 to $FF COPY. The routine above has decoded all these.

;  | 00 Fl0| 01 Fl1| 02 Br0| 03 Br1| 04 In0| 05 In1| 06 CAP| 07 EDT|
;  | 08 LFT| 09 RIG| 0A DWN| 0B UP | 0C DEL| 0D ENT| 0E SYM| 0F GRA|
;  | 10 Ik0| 11 Ik1| 12 Ik2| 13 Ik3| 14 Ik4| 15 Ik5| 16 Ik6| 17 Ik7|
;  | 18 Pa0| 19 Pa1| 1A Pa2| 1B Pa3| 1C Pa4| 1D Pa5| 1E Pa6| 1F Pa7|
;  | 20 SP | 21  ! | 22  " | 23  # | 24  $ | 25  % | 26  & | 27  ' |
;  | 28  ( | 29  ) | 2A  * | 2B  + | 2C  , | 2D  - | 2E  . | 2F  / |
;  | 30  0 | 31  1 | 32  2 | 33  3 | 34  4 | 35  5 | 36  6 | 37  7 |
;  | 38  8 | 39  9 | 3A  : | 3B  ; | 3C  < | 3D  = | 3E  > | 3F  ? |
;  | 40  @ | 41  A | 42  B | 43  C | 44  D | 45  E | 46  F | 47  G |
;  | 48  H | 49  I | 4A  J | 4B  K | 4C  L | 4D  M | 4E  N | 4F  O |
;  | 50  P | 51  Q | 52  R | 53  S | 54  T | 55  U | 56  V | 57  W |
;  | 58  X | 59  Y | 5A  Z | 5B  [ | 5C  \ | 5D  ] | 5E  ^ | 5F  _ |
;  | 60 ukp| 61  a | 62  b | 63  c | 64  d | 65  e | 66  f | 67  g |
```

```
;  | 68  h | 69  i | 6A  j | 6B  k | 6C  l | 6D  m | 6E  n | 6F  o |
;  | 70  p | 71  q | 72  r | 73  s | 74  t | 75  u | 76  v | 77  w |
;  | 78  x | 79  y | 7A  z | 7B  { | 7C  | | 7D  } | 7E  ~ | 7F (c)|
;  | 80 128| 81 129| 82 130| 83 131| 84 132| 85 133| 86 134| 87 135|
;  | 88 136| 89 137| 8A 138| 8B 139| 8C 140| 8D 141| 8E 142| 8F 143|
;  | 90 [A]| 91 [B]| 92 [C]| 93 [D]| 94 [E]| 95 [F]| 96 [G]| 97 [H]|
;  | 98 [I]| 99 [J]| 9A [K]| 9B [L]| 9C [M]| 9D [N]| 9E [O]| 9F [P]|
;  | A0 [Q]| A1 [R]| A2 [S]| A3 [T]| A4 [U]| A5 RND| A6 IK$| A7 PI |
;  | A8 FN | A9 PNT| AA SC$| AB ATT| AC AT | AD TAB| AE VL$| AF COD|
;  | B0 VAL| B1 LEN| B2 SIN| B3 COS| B4 TAN| B5 ASN| B6 ACS| B7 ATN|
;  | B8 LN | B9 EXP| BA INT| BB SQR| BC SGN| BD ABS| BE PEK| BF IN |
;  | C0 USR| C1 ST$| C2 CH$| C3 NOT| C4 BIN| C5 OR | C6 AND| C7 <= |
;  | C8 >= | C9 <> | CA LIN| CB THN| CC TO | CD STP| CE DEF| CF CAT|
;  | D0 FMT| D1 MOV| D2 ERS| D3 OPN| D4 CLO| D5 MRG| D6 VFY| D7 BEP|
;  | D8 CIR| D9 INK| DA PAP| DB FLA| DC BRI| DD INV| DE OVR| DF OUT|
;  | E0 LPR| E1 LLI| E2 STP| E3 REA| E4 DAT| E5 RES| E6 NEW| E7 BDR|
;  | E8 CON| E9 DIM| EA REM| EB FOR| EC GTO| ED GSB| EE INP| EF LOA|
;  | F0 LIS| F1 LET| F2 PAU| F3 NXT| F4 POK| F5 PRI| F6 PLO| F7 RUN|
;  | F8 SAV| F9 RAN| FA IF | FB CLS| FC DRW| FD CLR| FE RET| FF CPY|

;   Note that for simplicity, Sinclair have located all the control codes
;   below the space character.
;   ASCII DEL, $7F, has been made a copyright symbol.
;   Also $60, '`', not used in BASIC but used in other languages, has been
;   allocated the local currency symbol for the relevant country -
;   ukp in most Spectrums.

; -------------------------------------------------------------------------

;*********************************
;** Part 3. LOUDSPEAKER ROUTINES **
;*********************************


; Documented by Alvin Albrecht.


; ------------------------------
; Routine to control loudspeaker
; ------------------------------
; Outputs a square wave of given duration and frequency
; to the loudspeaker.
;   Enter with: DE = #cycles - 1
;               HL = tone period as described next
;
; The tone period is measured in T states and consists of
; three parts: a coarse part (H register), a medium part
; (bits 7..2 of L) and a fine part (bits 1..0 of L) which
; contribute to the waveform timing as follows:
;
;                          coarse    medium       fine
; duration of low  = 118 + 1024*H + 16*(L>>2) + 4*(L&0x3)
; duration of hi   = 118 + 1024*H + 16*(L>>2) + 4*(L&0x3)
; Tp = tone period = 236 + 2048*H + 32*(L>>2) + 8*(L&0x3)
;                  = 236 + 2048*H + 8*L = 236 + 8*HL
;
; As an example, to output five seconds of middle C (261.624 Hz):
;   (a) Tone period = 1/261.624 = 3.822ms
;   (b) Tone period in T-States = 3.822ms*fCPU = 13378
;         where fCPU = clock frequency of the CPU = 3.5MHz
;   (c) Find H and L for desired tone period:
;         HL = (Tp - 236) / 8 = (13378 - 236) / 8 = 1643 = 0x066B
;   (d) Tone duration in cycles = 5s/3.822ms = 1308 cycles
;         DE = 1308 - 1 = 0x051B
```

```
;
; The resulting waveform has a duty ratio of exactly 50%.
;
;
;; BEEPER
L03B5:  DI                      ; Disable Interrupts so they don't disturb
timing
        LD      A,L             ;
        SRL     L               ;
        SRL     L               ; L = medium part of tone period
        CPL                     ;
        AND     $03             ; A = 3 - fine part of tone period
        LD      C,A             ;
        LD      B,$00           ;
        LD      IX,L03D1        ; Address: BE-IX+3
        ADD     IX,BC           ;   IX holds address of entry into the loop
                                ;   the loop will contain 0-3 NOPs, implementing
                                ;   the fine part of the tone period.
        LD      A,($5C48)       ; BORDCR
        AND     $38             ; bits 5..3 contain border colour
        RRCA                    ; border colour bits moved to 2..0
        RRCA                    ;   to match border bits on port #FE
        RRCA                    ;
        OR       $08            ; bit 3 set (tape output bit on port #FE)
                                ;   for loud sound output
;; BE-IX+3
L03D1:  NOP              ;(4)   ; optionally executed NOPs for small
                                ;   adjustments to tone period
;; BE-IX+2
L03D2:  NOP              ;(4)   ;

;; BE-IX+1
L03D3:  NOP              ;(4)   ;

;; BE-IX+0
L03D4:  INC     B        ;(4)   ;
        INC     C        ;(4)   ;

;; BE-H&L-LP
L03D6:  DEC     C        ;(4)   ; timing loop for duration of
        JR      NZ,L03D6 ;(12/7);   high or low pulse of waveform

        LD      C,$3F    ;(7)   ;
        DEC     B        ;(4)   ;
        JP      NZ,L03D6 ;(10)  ; to BE-H&L-LP

        XOR     $10      ;(7)   ; toggle output beep bit
        OUT     ($FE),A  ;(11)  ; output pulse
        LD      B,H      ;(4)   ; B = coarse part of tone period
        LD      C,A      ;(4)   ; save port #FE output byte
        BIT     4,A      ;(8)   ; if new output bit is high, go
        JR      NZ,L03F2 ;(12/7);   to BE-AGAIN

        LD      A,D      ;(4)   ; one cycle of waveform has completed
        OR      E        ;(4)   ;   (low->low). if cycle countdown = 0
        JR      Z,L03F6  ;(12/7);   go to BE-END

        LD      A,C      ;(4)   ; restore output byte for port #FE
        LD      C,L      ;(4)   ; C = medium part of tone period
        DEC     DE       ;(6)   ; decrement cycle count
        JP      (IX)     ;(8)   ; do another cycle

;; BE-AGAIN                     ; halfway through cycle
L03F2:  LD      C,L      ;(4)   ; C = medium part of tone period
```

```
        INC     C        ;(4)   ; adds 16 cycles to make duration of high =
duration of low
        JP      (IX)     ;(8)   ; do high pulse of tone

;; BE-END
L03F6:  EI                      ; Enable Interrupts
        RET                     ;


; ------------------
; THE 'BEEP' COMMAND
; ------------------
; BASIC interface to BEEPER subroutine.
; Invoked in BASIC with:
;   BEEP dur, pitch
;   where dur   = duration in seconds
;         pitch = # of semitones above/below middle C
;
; Enter with: pitch on top of calculator stack
;             duration next on calculator stack
;
;; beep
L03F8:  RST     28H             ;; FP-CALC
        DEFB    $31             ;;duplicate                  ; duplicate pitch
        DEFB    $27             ;;int                        ; convert to
integer
        DEFB    $C0             ;;st-mem-0                   ; store integer
pitch to memory 0
        DEFB    $03             ;;subtract                   ; calculate
fractional part of pitch = fp_pitch - int_pitch
        DEFB    $34             ;;stk-data                   ; push constant
        DEFB    $EC             ;;Exponent: $7C, Bytes: 4    ; constant =
0.05762265
        DEFB    $6C,$98,$1F,$F5 ;;($6C,$98,$1F,$F5)
        DEFB    $04             ;;multiply                   ; compute:
        DEFB    $A1             ;;stk-one                    ; 1 + 0.05762265 *
fraction_part(pitch)
        DEFB    $0F             ;;addition
        DEFB    $38             ;;end-calc                   ; leave on calc
stack

        LD      HL,$5C92        ; MEM-0: number stored here is in 16 bit integer
format (pitch)
                                ;   0, 0/FF (pos/neg), LSB, MSB, 0
                                ;   LSB/MSB is stored in two's complement
                                ; In the following, the pitch is checked if it
is in the range -128<=p<=127
        LD      A,(HL)          ; First byte must be zero, otherwise
        AND     A               ;   error in integer conversion
        JR      NZ,L046C        ; to REPORT-B

        INC     HL              ;
        LD      C,(HL)          ; C = pos/neg flag = 0/FF
        INC     HL              ;
        LD      B,(HL)          ; B = LSB, two's complement
        LD      A,B             ;
        RLA                     ;
        SBC     A,A             ; A = 0/FF if B is pos/neg
        CP      C               ; must be the same as C if the pitch is
-128<=p<=127
        JR      NZ,L046C        ; if no, error REPORT-B

        INC     HL              ; if -128<=p<=127, MSB will be 0/FF if B is
pos/neg
```

```
        CP      (HL)            ; verify this
        JR      NZ,L046C        ; if no, error REPORT-B
                                ; now we know -128<=p<=127
        LD      A,B             ; A = pitch + 60
        ADD     A,$3C           ; if -60<=pitch<=67,
        JP      P,L0425         ;   goto BE-i-OK

        JP      PO,L046C        ; if pitch <= 67 goto REPORT-B
                                ;   lower bound of pitch set at -60

;; BE-I-OK                      ; here, -60<=pitch<=127
                                ; and A=pitch+60 -> 0<=A<=187

L0425:  LD      B,$FA           ; 6 octaves below middle C

;; BE-OCTAVE                    ; A=# semitones above 5 octaves below middle C
L0427:  INC     B               ; increment octave
        SUB     $0C             ; 12 semitones = one octave
        JR      NC,L0427        ; to BE-OCTAVE

        ADD     A,$0C           ; A = # semitones above C (0-11)
        PUSH    BC              ; B = octave displacement from middle C, 2's
complement: -5<=B<=10
        LD      HL,L046E        ; Address: semi-tone
        CALL    L3406           ; routine LOC-MEM
                                ;   HL = 5*A + $046E
        CALL    L33B4           ; routine STACK-NUM
                                ;   read FP value (freq) from semitone table
(HL) and push onto calc stack

        RST     28H             ;; FP-CALC
        DEFB    $04             ;;multiply   mult freq by 1 + 0.0576 *
fraction_part(pitch) stacked earlier
                                ;;             thus taking into account
fractional part of pitch.
                                ;;           the number 0.0576*frequency is the
distance in Hz to the next
                                ;;             note (verify with the frequencies
recorded in the semitone
                                ;;             table below) so that the
fraction_part of the pitch does
                                ;;             indeed represent a fractional
distance to the next note.
        DEFB    $38             ;;end-calc   HL points to first byte of fp num
on stack = middle frequency to generate

        POP     AF              ; A = octave displacement from middle C, 2's
complement: -5<=A<=10
        ADD     A,(HL)          ; increase exponent by A (equivalent to
multiplying by 2^A)
        LD      (HL),A          ;

        RST     28H             ;; FP-CALC
        DEFB    $C0             ;;st-mem-0          ; store frequency in memory
0
        DEFB    $02             ;;delete            ; remove from calc stack
        DEFB    $31             ;;duplicate         ; duplicate duration
(seconds)
        DEFB    $38             ;;end-calc

        CALL    L1E94           ; routine FIND-INT1 ; FP duration to A
        CP      $0B             ; if dur > 10 seconds,
        JR      NC,L046C        ;   goto REPORT-B
```

```
        ;;; The following calculation finds the tone period for HL and the cycle
count
        ;;; for DE expected in the BEEPER subroutine.  From the example in the
BEEPER comments,
        ;;;
        ;;; HL = ((fCPU / f) - 236) / 8 = fCPU/8/f - 236/8 = 437500/f -29.5
        ;;; DE = duration * frequency - 1
        ;;;
        ;;; Note the different constant (30.125) used in the calculation of HL
        ;;; below.  This is probably an error.

        RST     28H             ;; FP-CALC
        DEFB    $E0             ;;get-mem-0                 ; push frequency
        DEFB    $04             ;;multiply                  ; result1: #cycles =
duration * frequency
        DEFB    $E0             ;;get-mem-0                 ; push frequency
        DEFB    $34             ;;stk-data                  ; push constant
        DEFB    $80             ;;Exponent $93, Bytes: 3    ; constant = 437500
        DEFB    $43,$55,$9F,$80 ;;($55,$9F,$80,$00)
        DEFB    $01             ;;exchange                  ; frequency on top
        DEFB    $05             ;;division                  ; 437500 / frequency
        DEFB    $34             ;;stk-data                  ; push constant
        DEFB    $35             ;;Exponent: $85, Bytes: 1   ; constant = 30.125
        DEFB    $71             ;;($71,$00,$00,$00)
        DEFB    $03             ;;subtract                  ; result2:
tone_period(HL) = 437500 / freq - 30.125
        DEFB    $38             ;;end-calc

        CALL    L1E99           ; routine FIND-INT2
        PUSH    BC              ;   BC = tone_period(HL)
        CALL    L1E99           ; routine FIND-INT2, BC = #cycles to generate
        POP     HL              ; HL = tone period
        LD      D,B             ;
        LD      E,C             ; DE = #cycles
        LD      A,D             ;
        OR      E               ;
        RET     Z               ; if duration = 0, skip BEEP and avoid 65536
cycle
                                ;   boondoggle that would occur next
        DEC     DE              ; DE = #cycles - 1
        JP      L03B5           ; to BEEPER

; ---


;; REPORT-B
L046C:  RST     08H             ; ERROR-1
        DEFB    $0A             ; Error Report: Integer out of range



; ---------------------
; THE 'SEMI-TONE' TABLE
; ---------------------
;
;   Holds frequencies corresponding to semitones in middle octave.
;   To move n octaves higher or lower, frequencies are multiplied by 2^n.

;; semi-tone         five byte fp         decimal freq     note (middle)
L046E:  DEFB    $89, $02, $D0, $12, $86;  261.625565290         C
        DEFB    $89, $0A, $97, $60, $75;  277.182631135         C#
        DEFB    $89, $12, $D5, $17, $1F;  293.664768100         D
        DEFB    $89, $1B, $90, $41, $02;  311.126983881         D#
        DEFB    $89, $24, $D0, $53, $CA;  329.627557039         E
```

```
        DEFB    $89, $2E, $9D, $36, $B1;  349.228231549         F
        DEFB    $89, $38, $FF, $49, $3E;  369.994422674         F#
        DEFB    $89, $43, $FF, $6A, $73;  391.995436072         G
        DEFB    $89, $4F, $A7, $00, $54;  415.304697513         G#
        DEFB    $89, $5C, $00, $00, $00;  440.000000000         A
        DEFB    $89, $69, $14, $F6, $24;  466.163761616         A#
        DEFB    $89, $76, $F1, $10, $05;  493.883301378         B


;   "Music is the hidden mathematical endeavour of a soul unconscious it
;    is calculating" - Gottfried Wilhelm Liebnitz 1646 - 1716


;****************************************
;** Part 4. CASSETTE HANDLING ROUTINES **
;****************************************

;   These routines begin with the service routines followed by a single
;   command entry point.
;   The first of these service routines is a curiosity.

; -----------------------
; THE 'ZX81 NAME' ROUTINE
; -----------------------
;   This routine fetches a filename in ZX81 format and is not used by the
;   cassette handling routines in this ROM.

;; zx81-name
L04AA:  CALL    L24FB           ; routine SCANNING to evaluate expression.
        LD      A,($5C3B)       ; fetch system variable FLAGS.
        ADD     A,A             ; test bit 7 - syntax, bit 6 - result type.
        JP      M,L1C8A         ; to REPORT-C if not string result
                                ; 'Nonsense in BASIC'.

        POP     HL              ; drop return address.
        RET     NC              ; return early if checking syntax.

        PUSH    HL              ; re-save return address.
        CALL    L2BF1           ; routine STK-FETCH fetches string parameters.
        LD      H,D             ; transfer start of filename
        LD      L,E             ; to the HL register.
        DEC     C               ; adjust to point to last character and
        RET     M               ; return if the null string.
                                ; or multiple of 256!

        ADD     HL,BC           ; find last character of the filename.
                                ; and also clear carry.
        SET     7,(HL)          ; invert it.
        RET                     ; return.

; =========================================
;
; PORT 254 ($FE)
;
;                      spk mic { border  }
;          ___ ___ ___ ___ ___ ___ ___ ___
; PORT    |   |   |   |   |   |   |   |   |
; 254     |   |   |   |   |   |   |   |   |
; $FE     |___|___|___|___|___|___|___|___|
;           7   6   5   4   3   2   1   0
;

; ----------------------------------
; Save header and program/data bytes
```

```
; --------------------------------
;   This routine saves a section of data. It is called from SA-CTRL to save the
;   seventeen bytes of header data. It is also the exit route from that routine
;   when it is set up to save the actual data.
;   On entry -
;   HL points to start of data.
;   IX points to descriptor.
;   The accumulator is set to  $00 for a header, $FF for data.

;; SA-BYTES
L04C2:  LD      HL,L053F        ; address: SA/LD-RET
        PUSH    HL              ; is pushed as common exit route.
                                ; however there is only one non-terminal exit
                                ; point.

        LD      HL,$1F80        ; a timing constant H=$1F, L=$80
                                ; inner and outer loop counters
                                ; a five second lead-in is used for a header.

        BIT     7,A             ; test one bit of accumulator.
                                ; (AND A ?)
        JR      Z,L04D0         ; skip to SA-FLAG if a header is being saved.

;   else is data bytes and a shorter lead-in is used.

        LD      HL,$0C98        ; another timing value H=$0C, L=$98.
                                ; a two second lead-in is used for the data.


;; SA-FLAG
L04D0:  EX      AF,AF'          ; save flag
        INC     DE              ; increase length by one.
        DEC     IX              ; decrease start.

        DI                      ; disable interrupts

        LD      A,$02           ; select red for border, microphone bit on.
        LD      B,A             ; also does as an initial slight counter value.

;; SA-LEADER
L04D8:  DJNZ    L04D8           ; self loop to SA-LEADER for delay.
                                ; after initial loop, count is $A4 (or $A3)

        OUT     ($FE),A         ; output byte $02/$0D to tape port.

        XOR     $0F             ; switch from RED (mic on) to CYAN (mic off).

        LD      B,$A4           ; hold count. also timed instruction.

        DEC     L               ; originally $80 or $98.
                                ; but subsequently cycles 256 times.
        JR      NZ,L04D8        ; back to SA-LEADER until L is zero.

;   the outer loop is counted by H

        DEC     B               ; decrement count
        DEC     H               ; originally  twelve or thirty-one.
        JP      P,L04D8         ; back to SA-LEADER until H becomes $FF

;   now send a sync pulse. At this stage mic is off and A holds value
;   for mic on.
;   A sync pulse is much shorter than the steady pulses of the lead-in.

        LD      B,$2F           ; another short timed delay.
```

```
;; SA-SYNC-1
L04EA:  DJNZ    L04EA           ; self loop to SA-SYNC-1

        OUT     ($FE),A         ; switch to mic on and red.
        LD      A,$0D           ; prepare mic off - cyan
        LD      B,$37           ; another short timed delay.

;; SA-SYNC-2
L04F2:  DJNZ    L04F2           ; self loop to SA-SYNC-2

        OUT     ($FE),A         ; output mic off, cyan border.
        LD      BC,$3B0E        ; B=$3B time(*), C=$0E, YELLOW, MIC OFF.

;

        EX      AF,AF'          ; restore saved flag
                                ; which is 1st byte to be saved.

        LD      L,A             ; and transfer to L.
                                ; the initial parity is A, $FF or $00.
        JP      L0507           ; JUMP forward to SA-START     ->
                                ; the mid entry point of loop.

; -------------------------
;   During the save loop a parity byte is maintained in H.
;   the save loop begins by testing if reduced length is zero and if so
;   the final parity byte is saved reducing count to $FFFF.

;; SA-LOOP
L04FE:  LD      A,D             ; fetch high byte
        OR      E               ; test against low byte.
        JR      Z,L050E         ; forward to SA-PARITY if zero.

        LD      L,(IX+$00)      ; load currently addressed byte to L.

;; SA-LOOP-P
L0505:  LD      A,H             ; fetch parity byte.
        XOR     L               ; exclusive or with new byte.

; -> the mid entry point of loop.

;; SA-START
L0507:  LD      H,A             ; put parity byte in H.
        LD      A,$01           ; prepare blue, mic=on.
        SCF                     ; set carry flag ready to rotate in.
        JP      L0525           ; JUMP forward to SA-8-BITS            -8->

; ---

;; SA-PARITY
L050E:  LD      L,H             ; transfer the running parity byte to L and
        JR      L0505           ; back to SA-LOOP-P
                                ; to output that byte before quitting normally.

; ---

;   The entry point to save yellow part of bit.
;   A bit consists of a period with mic on and blue border followed by
;   a period of mic off with yellow border.
;   Note. since the DJNZ instruction does not affect flags, the zero flag is
;   used to indicate which of the two passes is in effect and the carry
;   maintains the state of the bit to be saved.
```

```
;; SA-BIT-2
L0511:  LD      A,C             ; fetch 'mic on and yellow' which is
                                ; held permanently in C.
        BIT     7,B             ; set the zero flag. B holds $3E.

;   The entry point to save 1 entire bit. For first bit B holds $3B(*).
;   Carry is set if saved bit is 1. zero is reset NZ on entry.

;; SA-BIT-1
L0514:  DJNZ    L0514           ; self loop for delay to SA-BIT-1

        JR      NC,L051C        ; forward to SA-OUT if bit is 0.

;   but if bit is 1 then the mic state is held for longer.

        LD      B,$42           ; set timed delay. (66 decimal)

;; SA-SET
L051A:  DJNZ    L051A           ; self loop to SA-SET
                                ; (roughly an extra 66*13 clock cycles)

;; SA-OUT
L051C:  OUT     ($FE),A         ; blue and mic on OR  yellow and mic off.

        LD      B,$3E           ; set up delay
        JR      NZ,L0511        ; back to SA-BIT-2 if zero reset NZ (first pass)

;   proceed when the blue and yellow bands have been output.

        DEC     B               ; change value $3E to $3D.
        XOR     A               ; clear carry flag (ready to rotate in).
        INC     A               ; reset zero flag i.e. NZ.

; -8->

;; SA-8-BITS
L0525:  RL      L               ; rotate left through carry
                                ; C<76543210<C
        JP      NZ,L0514        ; JUMP back to SA-BIT-1
                                ; until all 8 bits done.

;   when the initial set carry is passed out again then a byte is complete.

        DEC     DE              ; decrease length
        INC     IX              ; increase byte pointer
        LD      B,$31           ; set up timing.

        LD      A,$7F           ; test the space key and
        IN      A,($FE)         ; return to common exit (to restore border)
        RRA                     ; if a space is pressed
        RET     NC              ; return to SA/LD-RET.   - - >

;   now test if byte counter has reached $FFFF.

        LD      A,D             ; fetch high byte
        INC     A               ; increment.
        JP      NZ,L04FE        ; JUMP to SA-LOOP if more bytes.

        LD      B,$3B           ; a final delay.

;; SA-DELAY
L053C:  DJNZ    L053C           ; self loop to SA-DELAY

        RET                     ; return - - >
```

```
; ------------------------------
; THE 'SAVE/LOAD RETURN' ROUTINE
; ------------------------------
;   The address of this routine is pushed on the stack prior to any load/save
;   operation and it handles normal completion with the restoration of the
;   border and also abnormal termination when the break key, or to be more
;   precise the space key is pressed during a tape operation.
;
; - - >

;; SA/LD-RET
L053F:  PUSH    AF              ; preserve accumulator throughout.
        LD      A,($5C48)       ; fetch border colour from BORDCR.
        AND     $38             ; mask off paper bits.
        RRCA                    ; rotate
        RRCA                    ; to the
        RRCA                    ; range 0-7.

        OUT     ($FE),A         ; change the border colour.

        LD      A,$7F           ; read from port address $7FFE the
        IN      A,($FE)         ; row with the space key at outside.

        RRA                     ; test for space key pressed.
        EI                      ; enable interrupts
        JR      C,L0554         ; forward to SA/LD-END if not


;; REPORT-Da
L0552:  RST     08H             ; ERROR-1
        DEFB    $0C             ; Error Report: BREAK - CONT repeats

; ---

;; SA/LD-END
L0554:  POP     AF              ; restore the accumulator.
        RET                     ; return.

; ------------------------------------
; Load header or block of information
; ------------------------------------
;   This routine is used to load bytes and on entry A is set to $00 for a
;   header or to $FF for data.  IX points to the start of receiving location
;   and DE holds the length of bytes to be loaded. If, on entry the carry flag
;   is set then data is loaded, if reset then it is verified.

;; LD-BYTES
L0556:  INC     D               ; reset the zero flag without disturbing carry.
        EX      AF,AF'          ; preserve entry flags.
        DEC     D               ; restore high byte of length.

        DI                      ; disable interrupts

        LD      A,$0F           ; make the border white and mic off.
        OUT     ($FE),A         ; output to port.

        LD      HL,L053F        ; Address: SA/LD-RET
        PUSH    HL              ; is saved on stack as terminating routine.

;   the reading of the EAR bit (D6) will always be preceded by a test of the
;   space key (D0), so store the initial post-test state.

        IN      A,($FE)         ; read the ear state - bit 6.
```

```
        RRA                     ; rotate to bit 5.
        AND     $20             ; isolate this bit.
        OR      $02             ; combine with red border colour.
        LD      C,A             ; and store initial state long-term in C.
        CP      A               ; set the zero flag.

;

;; LD-BREAK
L056B:  RET     NZ              ; return if at any time space is pressed.

;; LD-START
L056C:  CALL    L05E7           ; routine LD-EDGE-1
        JR      NC,L056B        ; back to LD-BREAK with time out and no
                                ; edge present on tape.

;   but continue when a transition is found on tape.

        LD      HL,$0415        ; set up 16-bit outer loop counter for
                                ; approx 1 second delay.

;; LD-WAIT
L0574:  DJNZ    L0574           ; self loop to LD-WAIT (for 256 times)

        DEC     HL              ; decrease outer loop counter.
        LD      A,H             ; test for
        OR      L               ; zero.
        JR      NZ,L0574        ; back to LD-WAIT, if not zero, with zero in B.

;   continue after delay with H holding zero and B also.
;   sample 256 edges to check that we are in the middle of a lead-in section.

        CALL    L05E3           ; routine LD-EDGE-2
        JR      NC,L056B        ; back to LD-BREAK
                                ; if no edges at all.

;; LD-LEADER
L0580:  LD      B,$9C           ; set timing value.
        CALL    L05E3           ; routine LD-EDGE-2
        JR      NC,L056B        ; back to LD-BREAK if time-out

        LD      A,$C6           ; two edges must be spaced apart.
        CP      B               ; compare
        JR      NC,L056C        ; back to LD-START if too close together for a
                                ; lead-in.

        INC     H               ; proceed to test 256 edged sample.
        JR      NZ,L0580        ; back to LD-LEADER while more to do.

;   sample indicates we are in the middle of a two or five second lead-in.
;   Now test every edge looking for the terminal sync signal.

;; LD-SYNC
L058F:  LD      B,$C9           ; initial timing value in B.
        CALL    L05E7           ; routine LD-EDGE-1
        JR      NC,L056B        ; back to LD-BREAK with time-out.

        LD      A,B             ; fetch augmented timing value from B.
        CP      $D4             ; compare
        JR      NC,L058F        ; back to LD-SYNC if gap too big, that is,
                                ; a normal lead-in edge gap.

;   but a short gap will be the sync pulse.
;   in which case another edge should appear before B rises to $FF
```

```
        CALL    L05E7           ; routine LD-EDGE-1
        RET     NC              ; return with time-out.

; proceed when the sync at the end of the lead-in is found.
; We are about to load data so change the border colours.

        LD      A,C             ; fetch long-term mask from C
        XOR     $03             ; and make blue/yellow.

        LD      C,A             ; store the new long-term byte.

        LD      H,$00           ; set up parity byte as zero.
        LD      B,$B0           ; timing.
        JR      L05C8           ; forward to LD-MARKER
                                ; the loop mid entry point with the alternate
                                ; zero flag reset to indicate first byte
                                ; is discarded.

; --------------
;   the loading loop loads each byte and is entered at the mid point.

;; LD-LOOP
L05A9:  EX      AF,AF'          ; restore entry flags and type in A.
        JR      NZ,L05B3        ; forward to LD-FLAG if awaiting initial flag
                                ; which is to be discarded.

        JR      NC,L05BD        ; forward to LD-VERIFY if not to be loaded.

        LD      (IX+$00),L      ; place loaded byte at memory location.
        JR      L05C2           ; forward to LD-NEXT

; ---

;; LD-FLAG
L05B3:  RL      C               ; preserve carry (verify) flag in long-term
                                ; state byte. Bit 7 can be lost.

        XOR     L               ; compare type in A with first byte in L.
        RET     NZ              ; return if no match e.g. CODE vs. DATA.

;   continue when data type matches.

        LD      A,C             ; fetch byte with stored carry
        RRA                     ; rotate it to carry flag again
        LD      C,A             ; restore long-term port state.

        INC     DE              ; increment length ??
        JR      L05C4           ; forward to LD-DEC.
                                ; but why not to location after ?

; ---
;   for verification the byte read from tape is compared with that in memory.

;; LD-VERIFY
L05BD:  LD      A,(IX+$00)      ; fetch byte from memory.
        XOR     L               ; compare with that on tape
        RET     NZ              ; return if not zero.

;; LD-NEXT
L05C2:  INC     IX              ; increment byte pointer.

;; LD-DEC
L05C4:  DEC     DE              ; decrement length.
```

```
        EX      AF,AF'          ; store the flags.
        LD      B,$B2           ; timing.

;   when starting to read 8 bits the receiving byte is marked with bit at right.
;   when this is rotated out again then 8 bits have been read.

;; LD-MARKER
L05C8:  LD      L,$01           ; initialize as %00000001

;; LD-8-BITS
L05CA:  CALL    L05E3           ; routine LD-EDGE-2 increments B relative to
                                ; gap between 2 edges.
        RET     NC              ; return with time-out.

        LD      A,$CB           ; the comparison byte.
        CP      B               ; compare to incremented value of B.
                                ; if B is higher then bit on tape was set.
                                ; if <= then bit on tape is reset.

        RL      L               ; rotate the carry bit into L.

        LD      B,$B0           ; reset the B timer byte.
        JP      NC,L05CA        ; JUMP back to LD-8-BITS

;   when carry set then marker bit has been passed out and byte is complete.

        LD      A,H             ; fetch the running parity byte.
        XOR     L               ; include the new byte.
        LD      H,A             ; and store back in parity register.

        LD      A,D             ; check length of
        OR      E               ; expected bytes.
        JR      NZ,L05A9        ; back to LD-LOOP
                                ; while there are more.

;   when all bytes loaded then parity byte should be zero.

        LD      A,H             ; fetch parity byte.
        CP      $01             ; set carry if zero.
        RET                     ; return
                                ; in no carry then error as checksum disagrees.

; -------------------------
; Check signal being loaded
; -------------------------
;   An edge is a transition from one mic state to another.
;   More specifically a change in bit 6 of value input from port $FE.
;   Graphically it is a change of border colour, say, blue to yellow.
;   The first entry point looks for two adjacent edges. The second entry point
;   is used to find a single edge.
;   The B register holds a count, up to 256, within which the edge (or edges)
;   must be found. The gap between two edges will be more for a '1' than a '0'
;   so the value of B denotes the state of the bit (two edges) read from tape.

; ->

;; LD-EDGE-2
L05E3:  CALL    L05E7           ; call routine LD-EDGE-1 below.
        RET     NC              ; return if space pressed or time-out.
                                ; else continue and look for another adjacent
                                ; edge which together represent a bit on the
                                ; tape.

; ->
```

```
;   this entry point is used to find a single edge from above but also
;   when detecting a read-in signal on the tape.

;; LD-EDGE-1
L05E7:  LD      A,$16           ; a delay value of twenty two.

;; LD-DELAY
L05E9:  DEC     A               ; decrement counter
        JR      NZ,L05E9        ; loop back to LD-DELAY 22 times.

        AND      A              ; clear carry.

;; LD-SAMPLE
L05ED:  INC     B               ; increment the time-out counter.
        RET     Z               ; return with failure when $FF passed.

        LD      A,$7F           ; prepare to read keyboard and EAR port
        IN      A,($FE)         ; row $7FFE. bit 6 is EAR, bit 0 is SPACE key.
        RRA                     ; test outer key the space. (bit 6 moves to 5)
        RET     NC              ; return if space pressed.  >>>

        XOR     C               ; compare with initial long-term state.
        AND     $20             ; isolate bit 5
        JR      Z,L05ED         ; back to LD-SAMPLE if no edge.

;   but an edge, a transition of the EAR bit, has been found so switch the
;   long-term comparison byte containing both border colour and EAR bit.

        LD      A,C             ; fetch comparison value.
        CPL                     ; switch the bits
        LD      C,A             ; and put back in C for long-term.

        AND     $07             ; isolate new colour bits.
        OR      $08             ; set bit 3 - MIC off.
        OUT     ($FE),A         ; send to port to effect the change of colour.

        SCF                     ; set carry flag signaling edge found within
                                ; time allowed.
        RET                     ; return.

; ---------------------------------
; Entry point for all tape commands
; ---------------------------------
;   This is the single entry point for the four tape commands.
;   The routine first determines in what context it has been called by examining
;   the low byte of the Syntax table entry which was stored in T_ADDR.
;   Subtracting $EO (the present arrangement) gives a value of
;   $00 - SAVE
;   $01 - LOAD
;   $02 - VERIFY
;   $03 - MERGE
;   As with all commands the address STMT-RET is on the stack.

;; SAVE-ETC
L0605:  POP     AF              ; discard address STMT-RET.
        LD      A,($5C74)       ; fetch T_ADDR

;   Now reduce the low byte of the Syntax table entry to give command.
;   Note. For ZASM use SUB $E0 as next instruction.

L0609:  SUB     L1ADF + 1 % 256 ; subtract the known offset.
                                ; ( is SUB $E0 in standard ROM )

        LD      ($5C74),A       ; and put back in T_ADDR as 0,1,2, or 3
```

```
                                ; for future reference.

        CALL    L1C8C           ; routine EXPT-EXP checks that a string
                                ; expression follows and stacks the
                                ; parameters in run-time.

        CALL    L2530           ; routine SYNTAX-Z
        JR      Z,L0652         ; forward to SA-DATA if checking syntax.

        LD      BC,$0011        ; presume seventeen bytes for a header.
        LD      A,($5C74)       ; fetch command from T_ADDR.
        AND     A               ; test for zero - SAVE.
        JR      Z,L0621         ; forward to SA-SPACE if so.

        LD      C,$22           ; else double length to thirty four.

;; SA-SPACE
L0621:  RST     30H             ; BC-SPACES creates 17/34 bytes in workspace.

        PUSH    DE              ; transfer the start of new space to
        POP     IX              ; the available index register.

;   ten spaces are required for the default filename but it is simpler to
;   overwrite the first file-type indicator byte as well.

        LD      B,$0B           ; set counter to eleven.
        LD      A,$20           ; prepare a space.

;; SA-BLANK
L0629:  LD      (DE),A          ; set workspace location to space.
        INC     DE              ; next location.
        DJNZ    L0629           ; loop back to SA-BLANK till all eleven done.

        LD      (IX+$01),$FF    ; set first byte of ten character filename
                                ; to $FF as a default to signal null string.

        CALL    L2BF1           ; routine STK-FETCH fetches the filename
                                ; parameters from the calculator stack.
                                ; length of string in BC.
                                ; start of string in DE.

        LD      HL,$FFF6        ; prepare the value minus ten.
        DEC     BC              ; decrement length.
                                ; ten becomes nine, zero becomes $FFFF.
        ADD     HL,BC           ; trial addition.
        INC     BC              ; restore true length.
        JR      NC,L064B        ; forward to SA-NAME if length is one to ten.

;   the filename is more than ten characters in length or the null string.

        LD      A,($5C74)       ; fetch command from T_ADDR.
        AND     A               ; test for zero - SAVE.
        JR      NZ,L0644        ; forward to SA-NULL if not the SAVE command.

;   but no more than ten characters are allowed for SAVE.
;   The first ten characters of any other command parameter are acceptable.
;   Weird, but necessary, if saving to sectors.
;   Note. the golden rule that there are no restriction on anything is broken.

;; REPORT-Fa
L0642:  RST     08H             ; ERROR-1
        DEFB    $0E             ; Error Report: Invalid file name

;   continue with LOAD, MERGE, VERIFY and also SAVE within ten character limit.
```

```
;; SA-NULL
L0644:  LD      A,B             ; test length of filename
        OR      C               ; for zero.
        JR      Z,L0652         ; forward to SA-DATA if so using the 255
                                ; indicator followed by spaces.

        LD      BC,$000A        ; else trim length to ten.

;   other paths rejoin here with BC holding length in range 1 - 10.

;; SA-NAME
L064B:  PUSH    IX              ; push start of file descriptor.
        POP     HL              ; and pop into HL.

        INC     HL              ; HL now addresses first byte of filename.
        EX      DE,HL           ; transfer destination address to DE, start
                                ; of string in command to HL.
        LDIR                    ; copy up to ten bytes
                                ; if less than ten then trailing spaces follow.

;   the case for the null string rejoins here.

;; SA-DATA
L0652:  RST     18H             ; GET-CHAR
        CP      $E4             ; is character after filename the token 'DATA' ?
        JR      NZ,L06A0        ; forward to SA-SCR$ to consider SCREEN$ if
                                ; not.

;   continue to consider DATA.

        LD      A,($5C74)       ; fetch command from T_ADDR
        CP      $03             ; is it 'VERIFY' ?
        JP      Z,L1C8A         ; jump forward to REPORT-C if so.
                                ; 'Nonsense in BASIC'
                                ; VERIFY "d" DATA is not allowed.

;   continue with SAVE, LOAD, MERGE of DATA.

        RST     20H             ; NEXT-CHAR
        CALL    L28B2           ; routine LOOK-VARS searches variables area
                                ; returning with carry reset if found or
                                ; checking syntax.
        SET     7,C             ; this converts a simple string to a
                                ; string array. The test for an array or string
                                ; comes later.
        JR      NC,L0672        ; forward to SA-V-OLD if variable found.

        LD      HL,$0000        ; set destination to zero as not fixed.
        LD      A,($5C74)       ; fetch command from T_ADDR
        DEC     A               ; test for 1 - LOAD
        JR      Z,L0685         ; forward to SA-V-NEW with LOAD DATA.
                                ; to load a new array.

;   otherwise the variable was not found in run-time with SAVE/MERGE.

;; REPORT-2a
L0670:  RST     08H             ; ERROR-1
        DEFB    $01             ; Error Report: Variable not found

;   continue with SAVE/LOAD  DATA

;; SA-V-OLD
L0672:  JP      NZ,L1C8A        ; to REPORT-C if not an array variable.
```

```
                                ; or erroneously a simple string.
                                ; 'Nonsense in BASIC'


        CALL    L2530           ; routine SYNTAX-Z
        JR      Z,L0692         ; forward to SA-DATA-1 if checking syntax.

        INC     HL              ; step past single character variable name.
        LD      A,(HL)          ; fetch low byte of length.
        LD      (IX+$0B),A      ; place in descriptor.
        INC     HL              ; point to high byte.
        LD      A,(HL)          ; and transfer that
        LD      (IX+$0C),A      ; to descriptor.
        INC     HL              ; increase pointer within variable.

;; SA-V-NEW
L0685:  LD      (IX+$0E),C      ; place character array name in  header.
        LD      A,$01           ; default to type numeric.
        BIT     6,C             ; test result from look-vars.
        JR      Z,L068F         ; forward to SA-V-TYPE if numeric.

        INC     A               ; set type to 2 - string array.

;; SA-V-TYPE
L068F:  LD      (IX+$00),A      ; place type 0, 1 or 2 in descriptor.

;; SA-DATA-1
L0692:  EX      DE,HL           ; save var pointer in DE

        RST     20H             ; NEXT-CHAR
        CP      $29             ; is character ')' ?
        JR      NZ,L0672        ; back if not to SA-V-OLD to report
                                ; 'Nonsense in BASIC'

        RST     20H             ; NEXT-CHAR advances character address.
        CALL    L1BEE           ; routine CHECK-END errors if not end of
                                ; the statement.

        EX      DE,HL           ; bring back variables data pointer.
        JP      L075A           ; jump forward to SA-ALL

; ---
;   the branch was here to consider a 'SCREEN$', the display file.

;; SA-SCR$
L06A0:  CP      $AA             ; is character the token 'SCREEN$' ?
        JR      NZ,L06C3        ; forward to SA-CODE if not.

        LD      A,($5C74)       ; fetch command from T_ADDR
        CP      $03             ; is it MERGE ?
        JP       Z,L1C8A        ; jump to REPORT-C if so.
                                ; 'Nonsense in BASIC'

;   continue with SAVE/LOAD/VERIFY SCREEN$.

        RST     20H             ; NEXT-CHAR
        CALL    L1BEE           ; routine CHECK-END errors if not at end of
                                ; statement.

;   continue in runtime.

        LD      (IX+$0B),$00    ; set descriptor length
        LD      (IX+$0C),$1B    ; to $1b00 to include bitmaps and attributes.
```

```
        LD      HL,$4000        ; set start to display file start.
        LD      (IX+$0D),L      ; place start in
        LD      (IX+$0E),H      ; the descriptor.
        JR      L0710           ; forward to SA-TYPE-3

; ---
;   the branch was here to consider CODE.

;; SA-CODE
L06C3:  CP      $AF             ; is character the token 'CODE' ?
        JR      NZ,L0716        ; forward if not to SA-LINE to consider an
                                ; auto-started BASIC program.

        LD      A,($5C74)       ; fetch command from T_ADDR
        CP      $03             ; is it MERGE ?
        JP      Z,L1C8A         ; jump forward to REPORT-C if so.
                                ; 'Nonsense in BASIC'


        RST     20H             ; NEXT-CHAR advances character address.
        CALL    L2048           ; routine PR-ST-END checks if a carriage
                                ; return or ':' follows.
        JR      NZ,L06E1        ; forward to SA-CODE-1 if there are parameters.

        LD      A,($5C74)       ; else fetch the command from T_ADDR.
        AND     A               ; test for zero - SAVE without a specification.
        JP      Z,L1C8A         ; jump to REPORT-C if so.
                                ; 'Nonsense in BASIC'

;   for LOAD/VERIFY put zero on stack to signify handle at location saved from.

        CALL    L1CE6           ; routine USE-ZERO
        JR      L06F0           ; forward to SA-CODE-2

; ---

;   if there are more characters after CODE expect start and possibly length.

;; SA-CODE-1
L06E1:  CALL    L1C82           ; routine EXPT-1NUM checks for numeric
                                ; expression and stacks it in run-time.

        RST     18H             ; GET-CHAR
        CP      $2C             ; does a comma follow ?
        JR      Z,L06F5         ; forward if so to SA-CODE-3

;   else allow saved code to be loaded to a specified address.

        LD      A,($5C74)       ; fetch command from T_ADDR.
        AND     A               ; is the command SAVE which requires length ?
        JP      Z,L1C8A         ; jump to REPORT-C if so.
                                ; 'Nonsense in BASIC'

;   the command LOAD code may rejoin here with zero stacked as start.

;; SA-CODE-2
L06F0:  CALL    L1CE6           ; routine USE-ZERO stacks zero for length.
        JR      L06F9           ; forward to SA-CODE-4

; ---
;   the branch was here with SAVE CODE start,

;; SA-CODE-3
L06F5:  RST     20H             ; NEXT-CHAR advances character address.
```

```
        CALL    L1C82           ; routine EXPT-1NUM checks for expression
                                ; and stacks in run-time.

;   paths converge here and nothing must follow.

;; SA-CODE-4
L06F9:  CALL    L1BEE           ; routine CHECK-END errors with extraneous
                                ; characters and quits if checking syntax.

;   in run-time there are two 16-bit parameters on the calculator stack.

        CALL    L1E99           ; routine FIND-INT2 gets length.
        LD      (IX+$0B),C      ; place length
        LD      (IX+$0C),B      ; in descriptor.
        CALL    L1E99           ; routine FIND-INT2 gets start.
        LD      (IX+$0D),C      ; place start
        LD      (IX+$0E),B      ; in descriptor.
        LD      H,B             ; transfer the
        LD      L,C             ; start to HL also.

;; SA-TYPE-3
L0710:  LD      (IX+$00),$03    ; place type 3 - code in descriptor.
        JR      L075A           ; forward to SA-ALL.

; ---
;   the branch was here with BASIC to consider an optional auto-start line
;   number.

;; SA-LINE
L0716:  CP      $CA             ; is character the token 'LINE' ?
        JR      Z,L0723         ; forward to SA-LINE-1 if so.

;   else all possibilities have been considered and nothing must follow.

        CALL    L1BEE           ; routine CHECK-END

;   continue in run-time to save BASIC without auto-start.

        LD      (IX+$0E),$80    ; place high line number in descriptor to
                                ; disable auto-start.
        JR      L073A           ; forward to SA-TYPE-0 to save program.

; ---
;   the branch was here to consider auto-start.

;; SA-LINE-1
L0723:  LD      A,($5C74)       ; fetch command from T_ADDR
        AND     A               ; test for SAVE.
        JP      NZ,L1C8A        ; jump forward to REPORT-C with anything else.
                                ; 'Nonsense in BASIC'

;

        RST     20H             ; NEXT-CHAR
        CALL    L1C82           ; routine EXPT-1NUM checks for numeric
                                ; expression and stacks in run-time.
        CALL    L1BEE           ; routine CHECK-END quits if syntax path.
        CALL    L1E99           ; routine FIND-INT2 fetches the numeric
                                ; expression.
        LD      (IX+$0D),C      ; place the auto-start
        LD      (IX+$0E),B      ; line number in the descriptor.

;   Note. this isn't checked, but is subsequently handled by the system.
;   If the user typed 40000 instead of 4000 then it won't auto-start
```

```
;   at line 4000, or indeed, at all.

;   continue to save program and any variables.

;; SA-TYPE-0
L073A:  LD      (IX+$00),$00    ; place type zero - program in descriptor.
        LD      HL,($5C59)      ; fetch E_LINE to HL.
        LD      DE,($5C53)      ; fetch PROG to DE.
        SCF                     ; set carry flag to calculate from end of
                                ; variables E_LINE -1.
        SBC     HL,DE           ; subtract to give total length.

        LD      (IX+$0B),L      ; place total length
        LD      (IX+$0C),H      ; in descriptor.
        LD      HL,($5C4B)      ; load HL from system variable VARS
        SBC     HL,DE           ; subtract to give program length.
        LD      (IX+$0F),L      ; place length of program
        LD      (IX+$10),H      ; in the descriptor.
        EX      DE,HL           ; start to HL, length to DE.

;; SA-ALL
L075A:  LD      A,($5C74)       ; fetch command from T_ADDR
        AND     A               ; test for zero - SAVE.
        JP      Z,L0970         ; jump forward to SA-CONTRL with SAVE  ->

; ---
;   continue with LOAD, MERGE and VERIFY.

        PUSH    HL              ; save start.
        LD      BC,$0011        ; prepare to add seventeen
        ADD     IX,BC           ; to point IX at second descriptor.

;; LD-LOOK-H
L0767:  PUSH    IX              ; save IX
        LD      DE,$0011        ; seventeen bytes
        XOR     A               ; reset zero flag
        SCF                     ; set carry flag
        CALL    L0556           ; routine LD-BYTES loads a header from tape
                                ; to second descriptor.
        POP     IX              ; restore IX.
        JR      NC,L0767        ; loop back to LD-LOOK-H until header found.

        LD      A,$FE           ; select system channel 'S'
        CALL    L1601           ; routine CHAN-OPEN opens it.

        LD      (IY+$52),$03    ; set SCR_CT to 3 lines.

        LD      C,$80           ; C has bit 7 set to indicate type mismatch as
                                ; a default startpoint.

        LD      A,(IX+$00)      ; fetch loaded header type to A
        CP      (IX-$11)        ; compare with expected type.
        JR      NZ,L078A        ; forward to LD-TYPE with mis-match.

        LD      C,$F6           ; set C to minus ten - will count characters
                                ; up to zero.

;; LD-TYPE
L078A:  CP      $04             ; check if type in acceptable range 0 - 3.
        JR      NC,L0767        ; back to LD-LOOK-H with 4 and over.

;   else A indicates type 0-3.

        LD      DE,L09C0        ; address base of last 4 tape messages
```

```
        PUSH    BC                ; save BC
        CALL    L0C0A             ; routine PO-MSG outputs relevant message.
                                  ; Note. all messages have a leading newline.
        POP     BC                ; restore BC

        PUSH    IX                ; transfer IX,
        POP     DE                ; the 2nd descriptor, to DE.
        LD      HL,$FFF0          ; prepare minus seventeen.
        ADD     HL,DE             ; add to point HL to 1st descriptor.
        LD      B,$0A             ; the count will be ten characters for the
                                  ; filename.

        LD      A,(HL)            ; fetch first character and test for
        INC     A                 ; value 255.
        JR      NZ,L07A6          ; forward to LD-NAME if not the wildcard.

;   but if it is the wildcard, then add ten to C which is minus ten for a type
;   match or -128 for a type mismatch. Although characters have to be counted
;   bit 7 of C will not alter from state set here.

        LD      A,C               ; transfer $F6 or $80 to A
        ADD     A,B               ; add $0A
        LD      C,A               ; place result, zero or -118, in C.

;   At this point we have either a type mismatch, a wildcard match or ten
;   characters to be counted. The characters must be shown on the screen.

;; LD-NAME
L07A6:  INC     DE                ; address next input character
        LD      A,(DE)            ; fetch character
        CP      (HL)              ; compare to expected
        INC     HL                ; address next expected character
        JR      NZ,L07AD          ; forward to LD-CH-PR with mismatch

        INC     C                 ; increment matched character count

;; LD-CH-PR
L07AD:  RST     10H               ; PRINT-A prints character
        DJNZ    L07A6             ; loop back to LD-NAME for ten characters.

;   if ten characters matched and the types previously matched then C will
;   now hold zero.

        BIT     7,C               ; test if all matched
        JR      NZ,L0767          ; back to LD-LOOK-H if not

;   else print a terminal carriage return.

        LD      A,$0D             ; prepare carriage return.
        RST     10H               ; PRINT-A outputs it.

;   The various control routines for LOAD, VERIFY and MERGE are executed
;   during the one-second gap following the header on tape.

        POP     HL                ; restore xx
        LD      A,(IX+$00)        ; fetch incoming type
        CP      $03               ; compare with CODE
        JR      Z,L07CB           ; forward to VR-CONTROL if it is CODE.

;  type is a program or an array.

        LD      A,($5C74)         ; fetch command from T_ADDR
        DEC     A                 ; was it LOAD ?
        JP      Z,L0808           ; JUMP forward to LD-CONTRL if so to
```

```
                                ; load BASIC or variables.

        CP      $02             ; was command MERGE ?
        JP      Z,L08B6         ; jump forward to ME-CONTRL if so.

;   else continue into VERIFY control routine to verify.

; ----------------------------
; THE 'VERIFY CONTROL' ROUTINE
; ----------------------------
;   There are two branches to this routine.
;   1) From above to verify a program or array
;   2) from earlier with no carry to load or verify code.

;; VR-CONTROL
L07CB:  PUSH    HL              ; save pointer to data.
        LD      L,(IX-$06)      ; fetch length of old data
        LD      H,(IX-$05)      ; to HL.
        LD      E,(IX+$0B)      ; fetch length of new data
        LD      D,(IX+$0C)      ; to DE.
        LD      A,H             ; check length of old
        OR      L               ; for zero.
        JR      Z,L07E9         ; forward to VR-CONT-1 if length unspecified
                                ; e.g. LOAD "x" CODE

;   as opposed to, say, LOAD 'x' CODE 32768,300.

        SBC     HL,DE           ; subtract the two lengths.
        JR      C,L0806         ; forward to REPORT-R if the length on tape is
                                ; larger than that specified in command.
                                ; 'Tape loading error'

        JR      Z,L07E9         ; forward to VR-CONT-1 if lengths match.

;   a length on tape shorter than expected is not allowed for CODE

        LD      A,(IX+$00)      ; else fetch type from tape.
        CP      $03             ; is it CODE ?
        JR      NZ,L0806        ; forward to REPORT-R if so
                                ; 'Tape loading error'

;; VR-CONT-1
L07E9:  POP     HL              ; pop pointer to data
        LD      A,H             ; test for zero
        OR      L               ; e.g. LOAD 'x' CODE
        JR      NZ,L07F4        ; forward to VR-CONT-2 if destination specified.

        LD      L,(IX+$0D)      ; else use the destination in the header
        LD      H,(IX+$0E)      ; and load code at address saved from.

;; VR-CONT-2
L07F4:  PUSH    HL              ; push pointer to start of data block.
        POP     IX              ; transfer to IX.
        LD      A,($5C74)       ; fetch reduced command from T_ADDR
        CP      $02             ; is it VERIFY ?
        SCF                     ; prepare a set carry flag
        JR      NZ,L0800        ; skip to VR-CONT-3 if not

        AND     A               ; clear carry flag for VERIFY so that
                                ; data is not loaded.

;; VR-CONT-3
L0800:  LD      A,$FF           ; signal data block to be loaded
```

```
; -----------------
; Load a data block
; -----------------
;   This routine is called from 3 places other than above to load a data block.
;   In all cases the accumulator is first set to $FF so the routine could be
;   called at the previous instruction.

;; LD-BLOCK
L0802:  CALL    L0556           ; routine LD-BYTES
        RET     C               ; return if successful.


;; REPORT-R
L0806:  RST     08H             ; ERROR-1
        DEFB    $1A             ; Error Report: Tape loading error

; --------------------------
; THE 'LOAD CONTROL' ROUTINE
; --------------------------
;   This branch is taken when the command is LOAD with type 0, 1 or 2.

;; LD-CONTRL
L0808:  LD      E,(IX+$0B)      ; fetch length of found data block
        LD      D,(IX+$0C)      ; from 2nd descriptor.
        PUSH    HL              ; save destination
        LD      A,H             ; test for zero
        OR      L               ;
        JR      NZ,L0819        ; forward if not to LD-CONT-1

        INC     DE              ; increase length
        INC     DE              ; for letter name
        INC     DE              ; and 16-bit length
        EX      DE,HL           ; length to HL,
        JR      L0825           ; forward to LD-CONT-2

; ---

;; LD-CONT-1
L0819:  LD      L,(IX-$06)      ; fetch length from
        LD      H,(IX-$05)      ; the first header.
        EX      DE,HL           ;
        SCF                     ; set carry flag
        SBC     HL,DE           ;
        JR      C,L082E         ; to LD-DATA

;; LD-CONT-2
L0825:  LD      DE,$0005        ; allow overhead of five bytes.
        ADD     HL,DE           ; add in the difference in data lengths.
        LD      B,H             ; transfer to
        LD      C,L             ; the BC register pair
        CALL    L1F05           ; routine TEST-ROOM fails if not enough room.

;; LD-DATA
L082E:  POP     HL              ; pop destination
        LD      A,(IX+$00)      ; fetch type 0, 1 or 2.
        AND     A               ; test for program and variables.
        JR      Z,L0873         ; forward if so to LD-PROG

;   the type is a numeric or string array.

        LD      A,H             ; test the destination for zero
        OR      L               ; indicating variable does not already exist.
        JR      Z,L084C         ; forward if so to LD-DATA-1
```

```
;   else the destination is the first dimension within the array structure

        DEC     HL                  ; address high byte of total length
        LD      B,(HL)              ; transfer to B.
        DEC     HL                  ; address low byte of total length.
        LD      C,(HL)              ; transfer to C.
        DEC     HL                  ; point to letter of variable.
        INC     BC                  ; adjust length to
        INC     BC                  ; include these
        INC     BC                  ; three bytes also.
        LD      ($5C5F),IX          ; save header pointer in X_PTR.
        CALL    L19E8               ; routine RECLAIM-2 reclaims the old variable
                                    ; sliding workspace including the two headers
                                    ; downwards.
        LD      IX,($5C5F)          ; reload IX from X_PTR which will have been
                                    ; adjusted down by POINTERS routine.

;; LD-DATA-1
L084C:  LD      HL,($5C59)          ; address E_LINE
        DEC     HL                  ; now point to the $80 variables end-marker.
        LD      C,(IX+$0B)          ; fetch new data length
        LD      B,(IX+$0C)          ; from 2nd header.
        PUSH    BC                  ; * save it.
        INC     BC                  ; adjust the
        INC     BC                  ; length to include
        INC     BC                  ; letter name and total length.
        LD      A,(IX-$03)          ; fetch letter name from old header.
        PUSH    AF                  ; preserve accumulator though not corrupted.

        CALL    L1655               ; routine MAKE-ROOM creates space for variable
                                    ; sliding workspace up. IX no longer addresses
                                    ; anywhere meaningful.
        INC     HL                  ; point to first new location.

        POP     AF                  ; fetch back the letter name.
        LD      (HL),A              ; place in first new location.
        POP     DE                  ; * pop the data length.
        INC     HL                  ; address 2nd location
        LD      (HL),E              ; store low byte of length.
        INC     HL                  ; address next.
        LD      (HL),D              ; store high byte.
        INC     HL                  ; address start of data.
        PUSH    HL                  ; transfer address
        POP     IX                  ; to IX register pair.
        SCF                         ; set carry flag indicating load not verify.
        LD      A,$FF               ; signal data not header.
        JP      L0802               ; JUMP back to LD-BLOCK

; -----------------
;   the branch is here when a program as opposed to an array is to be loaded.

;; LD-PROG
L0873:  EX      DE,HL               ; transfer dest to DE.
        LD      HL,($5C59)          ; address E_LINE
        DEC     HL                  ; now variables end-marker.
        LD      ($5C5F),IX          ; place the IX header pointer in X_PTR
        LD      C,(IX+$0B)          ; get new length
        LD      B,(IX+$0C)          ; from 2nd header
        PUSH    BC                  ; and save it.

        CALL    L19E5               ; routine RECLAIM-1 reclaims program and vars.
                                    ; adjusting X-PTR.

        POP     BC                  ; restore new length.
```

```
        PUSH    HL              ; * save start
        PUSH    BC              ; ** and length.

        CALL    L1655           ; routine MAKE-ROOM creates the space.

        LD      IX,($5C5F)      ; reload IX from adjusted X_PTR
        INC     HL              ; point to start of new area.
        LD      C,(IX+$0F)      ; fetch length of BASIC on tape
        LD      B,(IX+$10)      ; from 2nd descriptor
        ADD     HL,BC           ; add to address the start of variables.
        LD      ($5C4B),HL      ; set system variable VARS

        LD      H,(IX+$0E)      ; fetch high byte of autostart line number.
        LD      A,H             ; transfer to A
        AND     $C0             ; test if greater than $3F.
        JR      NZ,L08AD        ; forward to LD-PROG-1 if so with no autostart.

        LD      L,(IX+$0D)      ; else fetch the low byte.
        LD      ($5C42),HL      ; set system variable to line number NEWPPC
        LD      (IY+$0A),$00    ; set statement NSPPC to zero.

;; LD-PROG-1
L08AD:  POP     DE              ; ** pop the length
        POP     IX              ; * and start.
        SCF                     ; set carry flag
        LD      A,$FF           ; signal data as opposed to a header.
        JP      L0802           ; jump back to LD-BLOCK

; --------------------------
; THE 'MERGE CONTROL' ROUTINE
; --------------------------
;   the branch was here to merge a program and its variables or an array.
;

;; ME-CONTRL
L08B6:  LD      C,(IX+$0B)      ; fetch length
        LD      B,(IX+$0C)      ; of data block on tape.
        PUSH    BC              ; save it.
        INC     BC              ; one for the pot.

        RST     30H             ; BC-SPACES creates room in workspace.
                                ; HL addresses last new location.
        LD      (HL),$80        ; place end-marker at end.
        EX      DE,HL           ; transfer first location to HL.
        POP     DE              ; restore length to DE.
        PUSH    HL              ; save start.

        PUSH    HL              ; and transfer it
        POP     IX              ; to IX register.
        SCF                     ; set carry flag to load data on tape.
        LD      A,$FF           ; signal data not a header.
        CALL    L0802           ; routine LD-BLOCK loads to workspace.
        POP     HL              ; restore first location in workspace to HL.
X08CE   LD      DE,($5C53)      ; set DE from system variable PROG.

;   now enter a loop to merge the data block in workspace with the program and
;   variables.

;; ME-NEW-LP
L08D2:  LD      A,(HL)          ; fetch next byte from workspace.
        AND     $C0             ; compare with $3F.
        JR      NZ,L08F0        ; forward to ME-VAR-LP if a variable or
                                ; end-marker.
```

```
;   continue when HL addresses a BASIC line number.

;; ME-OLD-LP
L08D7:  LD      A,(DE)          ; fetch high byte from program area.
        INC     DE              ; bump prog address.
        CP      (HL)            ; compare with that in workspace.
        INC     HL              ; bump workspace address.
        JR      NZ,L08DF        ; forward to ME-OLD-L1 if high bytes don't match

        LD      A,(DE)          ; fetch the low byte of program line number.
        CP      (HL)            ; compare with that in workspace.

;; ME-OLD-L1
L08DF:  DEC     DE              ; point to start of
        DEC     HL              ; respective lines again.
        JR      NC,L08EB        ; forward to ME-NEW-L2 if line number in
                                ; workspace is less than or equal to current
                                ; program line as has to be added to program.

        PUSH    HL              ; else save workspace pointer.
        EX      DE,HL           ; transfer prog pointer to HL
        CALL    L19B8           ; routine NEXT-ONE finds next line in DE.
        POP     HL              ; restore workspace pointer
        JR      L08D7           ; back to ME-OLD-LP until destination position
                                ; in program area found.

; ---
;   the branch was here with an insertion or replacement point.

;; ME-NEW-L2
L08EB:  CALL    L092C           ; routine ME-ENTER enters the line
        JR      L08D2           ; loop back to ME-NEW-LP.

; ---
;   the branch was here when the location in workspace held a variable.

;; ME-VAR-LP
L08F0:  LD      A,(HL)          ; fetch first byte of workspace variable.
        LD      C,A             ; copy to C also.
        CP      $80             ; is it the end-marker ?
        RET     Z               ; return if so as complete.  >>>>>

        PUSH    HL              ; save workspace area pointer.
        LD      HL,($5C4B)      ; load HL with VARS - start of variables area.

;; ME-OLD-VP
L08F9:  LD      A,(HL)          ; fetch first byte.
        CP      $80             ; is it the end-marker ?
        JR      Z,L0923         ; forward if so to ME-VAR-L2 to add
                                ; variable at end of variables area.

        CP      C               ; compare with variable in workspace area.
        JR      Z,L0909         ; forward to ME-OLD-V2 if a match to replace.

;   else entire variables area has to be searched.

;; ME-OLD-V1
L0901:  PUSH    BC              ; save character in C.
        CALL    L19B8           ; routine NEXT-ONE gets following variable
                                ; address in DE.
        POP     BC              ; restore character in C
        EX      DE,HL           ; transfer next address to HL.
        JR      L08F9           ; loop back to ME-OLD-VP
```

```
; ---
;   the branch was here when first characters of name matched.

;; ME-OLD-V2
L0909:  AND     $E0             ; keep bits 11100000
        CP      $A0             ; compare   10100000 - a long-named variable.

        JR      NZ,L0921        ; forward to ME-VAR-L1 if just one-character.

;   but long-named variables have to be matched character by character.

        POP     DE              ; fetch workspace 1st character pointer
        PUSH    DE              ; and save it on the stack again.
        PUSH    HL              ; save variables area pointer on stack.

;; ME-OLD-V3
L0912:  INC     HL              ; address next character in vars area.
        INC     DE              ; address next character in workspace area.
        LD      A,(DE)          ; fetch workspace character.
        CP      (HL)            ; compare to variables character.
        JR      NZ,L091E        ; forward to ME-OLD-V4 with a mismatch.

        RLA                     ; test if the terminal inverted character.
        JR      NC,L0912        ; loop back to ME-OLD-V3 if more to test.

;   otherwise the long name matches in its entirety.

        POP     HL              ; restore pointer to first character of variable
        JR      L0921           ; forward to ME-VAR-L1

; ---
;   the branch is here when two characters don't match

;; ME-OLD-V4
L091E:  POP     HL              ; restore the prog/vars pointer.
        JR      L0901           ; back to ME-OLD-V1 to resume search.

; ---
;   branch here when variable is to replace an existing one

;; ME-VAR-L1
L0921:  LD      A,$FF           ; indicate a replacement.

;   this entry point is when A holds $80 indicating a new variable.

;; ME-VAR-L2
L0923:  POP     DE              ; pop workspace pointer.
        EX      DE,HL           ; now make HL workspace pointer, DE vars pointer
        INC     A               ; zero flag set if replacement.
        SCF                     ; set carry flag indicating a variable not a
                                ; program line.
        CALL    L092C           ; routine ME-ENTER copies variable in.
        JR      L08F0           ; loop back to ME-VAR-LP

; ------------------------
; Merge a Line or Variable
; ------------------------
;   A BASIC line or variable is inserted at the current point. If the line
;   number or variable names match (zero flag set) then a replacement takes
;   place.

;; ME-ENTER
L092C:  JR      NZ,L093E        ; forward to ME-ENT-1 for insertion only.
```

```
;   but the program line or variable matches so old one is reclaimed.

        EX      AF,AF'          ; save flag??
        LD      ($5C5F),HL      ; preserve workspace pointer in dynamic X_PTR
        EX      DE,HL           ; transfer program dest pointer to HL.
        CALL    L19B8           ; routine NEXT-ONE finds following location
                                ; in program or variables area.
        CALL    L19E8           ; routine RECLAIM-2 reclaims the space between.
        EX      DE,HL           ; transfer program dest pointer back to DE.
        LD      HL,($5C5F)      ; fetch adjusted workspace pointer from X_PTR
        EX      AF,AF'          ; restore flags.

;   now the new line or variable is entered.

;; ME-ENT-1
L093E:  EX      AF,AF'          ; save or re-save flags.
        PUSH    DE              ; save dest pointer in prog/vars area.
        CALL    L19B8           ; routine NEXT-ONE finds next in workspace.
                                ; gets next in DE, difference in BC.
                                ; prev addr in HL
        LD      ($5C5F),HL      ; store pointer in X_PTR
        LD      HL,($5C53)      ; load HL from system variable PROG
        EX      (SP),HL         ; swap with prog/vars pointer on stack.
        PUSH    BC              ; ** save length of new program line/variable.
        EX      AF,AF'          ; fetch flags back.
        JR      C,L0955         ; skip to ME-ENT-2 if variable

        DEC     HL              ; address location before pointer
        CALL    L1655           ; routine MAKE-ROOM creates room for BASIC line
        INC     HL              ; address next.
        JR      L0958           ; forward to ME-ENT-3

; ---

;; ME-ENT-2
L0955:  CALL    L1655           ; routine MAKE-ROOM creates room for variable.

;; ME-ENT-3
L0958:  INC     HL              ; address next?

        POP     BC              ; ** pop length
        POP     DE              ; * pop value for PROG which may have been
                                ; altered by POINTERS if first line.
        LD      ($5C53),DE      ; set PROG to original value.
        LD      DE,($5C5F)      ; fetch adjusted workspace pointer from X_PTR
        PUSH    BC              ; save length
        PUSH    DE              ; and workspace pointer
        EX      DE,HL           ; make workspace pointer source, prog/vars
                                ; pointer the destination
        LDIR                    ; copy bytes of line or variable into new area.
        POP     HL              ; restore workspace pointer.
        POP     BC              ; restore length.
        PUSH    DE              ; save new prog/vars pointer.
        CALL    L19E8           ; routine RECLAIM-2 reclaims the space used
                                ; by the line or variable in workspace block
                                ; as no longer required and space could be
                                ; useful for adding more lines.
        POP     DE              ; restore the prog/vars pointer
        RET                     ; return.

; --------------------------
; THE 'SAVE CONTROL' ROUTINE
; --------------------------
;   A branch from the main SAVE-ETC routine at SAVE-ALL.
```

```
;   First the header data is saved. Then after a wait of 1 second
;   the data itself is saved.
;   HL points to start of data.
;   IX points to start of descriptor.

;; SA-CONTRL
L0970:  PUSH    HL                ; save start of data

        LD      A,$FD             ; select system channel 'S'
        CALL    L1601             ; routine CHAN-OPEN

        XOR     A                 ; clear to address table directly
        LD      DE,L09A1          ; address: tape-msgs
        CALL    L0C0A             ; routine PO-MSG -
                                  ; 'Start tape then press any key.'

        SET     5,(IY+$02)        ; TV_FLAG  - Signal lower screen requires
                                  ; clearing
        CALL    L15D4             ; routine WAIT-KEY

        PUSH    IX                ; save pointer to descriptor.
        LD      DE,$0011          ; there are seventeen bytes.
        XOR     A                 ; signal a header.
        CALL    L04C2             ; routine SA-BYTES

        POP     IX                ; restore descriptor pointer.

        LD      B,$32             ; wait for a second - 50 interrupts.

;; SA-1-SEC
L0991:  HALT                      ; wait for interrupt
        DJNZ    L0991             ; back to SA-1-SEC until pause complete.

        LD      E,(IX+$0B)        ; fetch length of bytes from the
        LD      D,(IX+$0C)        ; descriptor.

        LD      A,$FF             ; signal data bytes.

        POP     IX                ; retrieve pointer to start
        JP      L04C2             ; jump back to SA-BYTES


;   Arrangement of two headers in workspace.
;   Originally IX addresses first location and only one header is required
;   when saving.
;
;   OLD     NEW         PROG   DATA  DATA  CODE
;   HEADER  HEADER             num   chr          NOTES.
;   ------  ------      ----   ----  ----  ----   -----------------------------
;   IX-$11  IX+$00      0      1     2     3      Type.
;   IX-$10  IX+$01      x      x     x     x      F  ($FF if filename is null).
;   IX-$0F  IX+$02      x      x     x     x      i
;   IX-$0E  IX+$03      x      x     x     x      l
;   IX-$0D  IX+$04      x      x     x     x      e
;   IX-$0C  IX+$05      x      x     x     x      n
;   IX-$0B  IX+$06      x      x     x     x      a
;   IX-$0A  IX+$07      x      x     x     x      m
;   IX-$09  IX+$08      x      x     x     x      e
;   IX-$08  IX+$09      x      x     x     x      .
;   IX-$07  IX+$0A      x      x     x     x      (terminal spaces).
;   IX-$06  IX+$0B      lo     lo    lo    lo     Total
;   IX-$05  IX+$0C      hi     hi    hi    hi     Length of datablock.
;   IX-$04  IX+$0D      Auto   -     -     Start  Various
;   IX-$03  IX+$0E      Start  a-z   a-z   addr   ($80 if no autostart).
```

```
;   IX-$02  IX+$0F      lo     -     -     -      Length of Program
;   IX-$01  IX+$10      hi     -     -     -      only i.e. without variables.
;


; ------------------------
; Canned cassette messages
; ------------------------
;   The last-character-inverted Cassette messages.
;   Starts with normal initial step-over byte.

;; tape-msgs
L09A1:  DEFB    $80
        DEFM    "Start tape, then press any key"
L09C0:  DEFB    '.'+$80
        DEFB    $0D
        DEFM    "Program:"
        DEFB    ' '+$80
        DEFB    $0D
        DEFM    "Number array:"
        DEFB    ' '+$80
        DEFB    $0D
        DEFM    "Character array:"
        DEFB    ' '+$80
        DEFB    $0D
        DEFM    "Bytes:"
        DEFB    ' '+$80


;**************************************************
;** Part 5. SCREEN AND PRINTER HANDLING ROUTINES **
;**************************************************

; --------------------------
; THE 'PRINT OUTPUT' ROUTINE
; --------------------------
;   This is the routine most often used by the RST 10 restart although the
;   subroutine is on two occasions called directly when it is known that
;   output will definitely be to the lower screen.

;; PRINT-OUT
L09F4:  CALL    L0B03           ; routine PO-FETCH fetches print position
                                ; to HL register pair.
        CP      $20             ; is character a space or higher ?
        JP      NC,L0AD9        ; jump forward to PO-ABLE if so.

        CP      $06             ; is character in range 00-05 ?
        JR      C,L0A69         ; to PO-QUEST to print '?' if so.

        CP      $18             ; is character in range 24d - 31d ?
        JR      NC,L0A69        ; to PO-QUEST to also print '?' if so.

        LD      HL,L0A11 - 6    ; address 0A0B - the base address of control
                                ; character table - where zero would be.
        LD      E,A             ; control character 06 - 23d
        LD      D,$00           ; is transferred to DE.

        ADD     HL,DE           ; index into table.

        LD      E,(HL)          ; fetch the offset to routine.
        ADD     HL,DE           ; add to make HL the address.
        PUSH    HL              ; push the address.

        JP      L0B03           ; Jump forward to PO-FETCH,
```

```
                                ; as the screen/printer position has been
                                ; disturbed, and then indirectly to the PO-STORE
                                ; routine on stack.

; -----------------------------
; THE 'CONTROL CHARACTER' TABLE
; -----------------------------
;   For control characters in the range 6 - 23d the following table
;   is indexed to provide an offset to the handling routine that
;   follows the table.

;; ctlchrtab
L0A11:  DEFB    L0A5F - $       ; 06d offset $4E to Address: PO-COMMA
        DEFB    L0A69 - $       ; 07d offset $57 to Address: PO-QUEST
        DEFB    L0A23 - $       ; 08d offset $10 to Address: PO-BACK-1
        DEFB    L0A3D - $       ; 09d offset $29 to Address: PO-RIGHT
        DEFB    L0A69 - $       ; 10d offset $54 to Address: PO-QUEST
        DEFB    L0A69 - $       ; 11d offset $53 to Address: PO-QUEST
        DEFB    L0A69 - $       ; 12d offset $52 to Address: PO-QUEST
        DEFB    L0A4F - $       ; 13d offset $37 to Address: PO-ENTER
        DEFB    L0A69 - $       ; 14d offset $50 to Address: PO-QUEST
        DEFB    L0A69 - $       ; 15d offset $4F to Address: PO-QUEST
        DEFB    L0A7A - $       ; 16d offset $5F to Address: PO-1-OPER
        DEFB    L0A7A - $       ; 17d offset $5E to Address: PO-1-OPER
        DEFB    L0A7A - $       ; 18d offset $5D to Address: PO-1-OPER
        DEFB    L0A7A - $       ; 19d offset $5C to Address: PO-1-OPER
        DEFB    L0A7A - $       ; 20d offset $5B to Address: PO-1-OPER
        DEFB    L0A7A - $       ; 21d offset $5A to Address: PO-1-OPER
        DEFB    L0A75 - $       ; 22d offset $54 to Address: PO-2-OPER
        DEFB    L0A75 - $       ; 23d offset $53 to Address: PO-2-OPER


; -------------------------
; THE 'CURSOR LEFT' ROUTINE
; -------------------------
;   Backspace and up a line if that action is from the left of screen.
;   For ZX printer backspace up to first column but not beyond.

;; PO-BACK-1
L0A23:  INC     C               ; move left one column.
        LD      A,$22           ; value $21 is leftmost column.
        CP      C               ; have we passed ?
        JR      NZ,L0A3A        ; to PO-BACK-3 if not and store new position.

        BIT     1,(IY+$01)      ; test FLAGS  - is printer in use ?
        JR      NZ,L0A38        ; to PO-BACK-2 if so, as we are unable to
                                ; backspace from the leftmost position.


        INC     B               ; move up one screen line
        LD      C,$02           ; the rightmost column position.
        LD      A,$18           ; Note. This should be $19
                                ; credit. Dr. Frank O'Hara, 1982

        CP      B               ; has position moved past top of screen ?
        JR      NZ,L0A3A        ; to PO-BACK-3 if not and store new position.

        DEC     B               ; else back to $18.

;; PO-BACK-2
L0A38:  LD      C,$21           ; the leftmost column position.

;; PO-BACK-3
L0A3A:  JP      L0DD9           ; to CL-SET and PO-STORE to save new
```

```
                                ; position in system variables.

; --------------------------
; THE 'CURSOR RIGHT' ROUTINE
; --------------------------
;   This moves the print position to the right leaving a trail in the
;   current background colour.
;   "However the programmer has failed to store the new print position
;   so CHR$ 9 will only work if the next print position is at a newly
;   defined place.
;   e.g. PRINT PAPER 2; CHR$ 9; AT 4,0;
;   does work but is not very helpful"
;   - Dr. Ian Logan, Understanding Your Spectrum, 1982.

;; PO-RIGHT
L0A3D:  LD      A,($5C91)       ; fetch P_FLAG value
        PUSH    AF              ; and save it on stack.

        LD      (IY+$57),$01    ; temporarily set P_FLAG 'OVER 1'.
        LD      A,$20           ; prepare a space.
        CALL    L0B65           ; routine PO-CHAR to print it.
                                ; Note. could be PO-ABLE which would update
                                ; the column position.

        POP     AF              ; restore the permanent flag.
        LD      ($5C91),A       ; and restore system variable P_FLAG

        RET                     ; return without updating column position

; -----------------------
; Perform carriage return
; -----------------------
; A carriage return is 'printed' to screen or printer buffer.

;; PO-ENTER
L0A4F:  BIT     1,(IY+$01)      ; test FLAGS  - is printer in use ?
        JP      NZ,L0ECD        ; to COPY-BUFF if so, to flush buffer and reset
                                ; the print position.

        LD      C,$21           ; the leftmost column position.
        CALL    L0C55           ; routine PO-SCR handles any scrolling required.
        DEC     B               ; to next screen line.
        JP      L0DD9           ; jump forward to CL-SET to store new position.

; -----------
; Print comma
; -----------
; The comma control character. The 32 column screen has two 16 character
; tabstops.  The routine is only reached via the control character table.

;; PO-COMMA
L0A5F:  CALL    L0B03           ; routine PO-FETCH - seems unnecessary.

        LD      A,C             ; the column position. $21-$01
        DEC     A               ; move right. $20-$00
        DEC     A               ; and again   $1F-$00 or $FF if trailing
        AND     $10             ; will be $00 or $10.
        JR      L0AC3           ; forward to PO-FILL

; -------------------
; Print question mark
; -------------------
; This routine prints a question mark which is commonly
; used to print an unassigned control character in range 0-31d.
```

```
; there are a surprising number yet to be assigned.

;; PO-QUEST
L0A69:  LD      A,$3F           ; prepare the character '?'.
        JR      L0AD9           ; forward to PO-ABLE.

; --------------------------------
; Control characters with operands
; --------------------------------
; Certain control characters are followed by 1 or 2 operands.
; The entry points from control character table are PO-2-OPER and PO-1-OPER.
; The routines alter the output address of the current channel so that
; subsequent RST $10 instructions take the appropriate action
; before finally resetting the output address back to PRINT-OUT.

;; PO-TV-2
L0A6D:  LD      DE,L0A87        ; address: PO-CONT will be next output routine
        LD      ($5C0F),A       ; store first operand in TVDATA-hi
        JR      L0A80           ; forward to PO-CHANGE >>

; ---

; -> This initial entry point deals with two operands - AT or TAB.

;; PO-2-OPER
L0A75:  LD      DE,L0A6D        ; address: PO-TV-2 will be next output routine
        JR      L0A7D           ; forward to PO-TV-1

; ---

; -> This initial entry point deals with one operand INK to OVER.

;; PO-1-OPER
L0A7A:  LD      DE,L0A87        ; address: PO-CONT will be next output routine

;; PO-TV-1
L0A7D:  LD      ($5C0E),A       ; store control code in TVDATA-lo

;; PO-CHANGE
L0A80:  LD      HL,($5C51)      ; use CURCHL to find current output channel.
        LD      (HL),E          ; make it
        INC     HL              ; the supplied
        LD      (HL),D          ; address from DE.
        RET                     ; return.

; ---

;; PO-CONT
L0A87:  LD      DE,L09F4        ; Address: PRINT-OUT
        CALL    L0A80           ; routine PO-CHANGE to restore normal channel.
        LD      HL,($5C0E)      ; TVDATA gives control code and possible
                                ; subsequent character
        LD      D,A             ; save current character
        LD      A,L             ; the stored control code
        CP      $16             ; was it INK to OVER (1 operand) ?
        JP      C,L2211         ; to CO-TEMP-5

        JR      NZ,L0AC2        ; to PO-TAB if not 22d i.e. 23d TAB.

                                ; else must have been 22d AT.
        LD      B,H             ; line to H   (0-23d)
        LD      C,D             ; column to C (0-31d)
        LD      A,$1F           ; the value 31d
        SUB     C               ; reverse the column number.
```

```
        JR      C,L0AAC         ; to PO-AT-ERR if C was greater than 31d.

        ADD     A,$02           ; transform to system range $02-$21
        LD      C,A             ; and place in column register.

        BIT     1,(IY+$01)      ; test FLAGS  - is printer in use ?
        JR      NZ,L0ABF        ; to PO-AT-SET as line can be ignored.

        LD      A,$16           ; 22 decimal
        SUB     B               ; subtract line number to reverse
                                ; 0 - 22 becomes 22 - 0.

;; PO-AT-ERR
L0AAC:  JP      C,L1E9F         ; to REPORT-B if higher than 22 decimal
                                ; Integer out of range.

        INC     A               ; adjust for system range $01-$17
        LD      B,A             ; place in line register
        INC     B               ; adjust to system range  $02-$18
        BIT     0,(IY+$02)      ; TV_FLAG  - Lower screen in use ?
        JP      NZ,L0C55        ; exit to PO-SCR to test for scrolling

        CP      (IY+$31)        ; Compare against DF_SZ
        JP      C,L0C86         ; to REPORT-5 if too low
                                ; Out of screen.

;; PO-AT-SET
L0ABF:  JP      L0DD9           ; print position is valid so exit via CL-SET

; ---

; Continue here when dealing with TAB.
; Note. In BASIC, TAB is followed by a 16-bit number and was initially
; designed to work with any output device.

;; PO-TAB
L0AC2:  LD      A,H             ; transfer parameter to A
                                ; Losing current character -
                                ; High byte of TAB parameter.


;; PO-FILL
L0AC3:  CALL    L0B03           ; routine PO-FETCH, HL-addr, BC=line/column.
                                ; column 1 (right), $21 (left)
        ADD     A,C             ; add operand to current column
        DEC     A               ; range 0 - 31+
        AND     $1F             ; make range 0 - 31d
        RET     Z               ; return if result zero

        LD      D,A             ; Counter to D
        SET     0,(IY+$01)      ; update FLAGS  - signal suppress leading space.

;; PO-SPACE
L0AD0:  LD      A,$20           ; space character.

        CALL    L0C3B           ; routine PO-SAVE prints the character
                                ; using alternate set (normal output routine)

        DEC     D               ; decrement counter.
        JR      NZ,L0AD0        ; to PO-SPACE until done

        RET                     ; return

; ----------------------
```

```
; Printable character(s)
; ----------------------
; This routine prints printable characters and continues into
; the position store routine

;; PO-ABLE
L0AD9:  CALL    L0B24           ; routine PO-ANY
                                ; and continue into position store routine.

; ----------------------------
; THE 'POSITION STORE' ROUTINE
; ----------------------------
;   This routine updates the system variables associated with the main screen,
;   the lower screen/input buffer or the ZX printer.

;; PO-STORE
L0ADC:  BIT     1,(IY+$01)      ; Test FLAGS - is printer in use ?
        JR      NZ,L0AFC        ; Forward, if so, to PO-ST-PR

        BIT     0,(IY+$02)      ; Test TV_FLAG - is lower screen in use ?
        JR      NZ,L0AF0        ; Forward, if so, to PO-ST-E

;   This section deals with the upper screen.

        LD      ($5C88),BC      ; Update S_POSN - line/column upper screen
        LD      ($5C84),HL      ; Update DF_CC - upper display file address

        RET                     ; Return.

; ---

;   This section deals with the lower screen.

;; PO-ST-E
L0AF0:  LD      ($5C8A),BC      ; Update SPOSNL line/column lower screen
        LD      ($5C82),BC      ; Update ECHO_E line/column input buffer
        LD      ($5C86),HL      ; Update DFCCL  lower screen memory address
        RET                     ; Return.

; ---

;   This section deals with the ZX Printer.

;; PO-ST-PR
L0AFC:  LD      (IY+$45),C      ; Update P_POSN column position printer
        LD      ($5C80),HL      ; Update PR_CC - full printer buffer memory
                                ; address
        RET                     ; Return.

;   Note. that any values stored in location 23681 will be overwritten with
;   the value 91 decimal.
;   Credit April 1983, Dilwyn Jones. "Delving Deeper into your ZX Spectrum".

; ----------------------------
; THE 'POSITION FETCH' ROUTINE
; ----------------------------
;   This routine fetches the line/column and display file address of the upper
;   and lower screen or, if the printer is in use, the column position and
;   absolute memory address.
;   Note. that PR-CC-hi (23681) is used by this routine and if, in accordance
;   with the manual (that says this is unused), the location has been used for
;   other purposes, then subsequent output to the printer buffer could corrupt
;   a 256-byte section of memory.
```

```
;; PO-FETCH
L0B03:  BIT     1,(IY+$01)      ; Test FLAGS - is printer in use ?
        JR      NZ,L0B1D        ; Forward, if so, to PO-F-PR

;   assume upper screen in use and thus optimize for path that requires speed.

        LD      BC,($5C88)      ; Fetch line/column from S_POSN
        LD      HL,($5C84)      ; Fetch DF_CC display file address

        BIT     0,(IY+$02)      ; Test TV_FLAG - lower screen in use ?
        RET     Z               ; Return if upper screen in use.

;   Overwrite registers with values for lower screen.

        LD      BC,($5C8A)      ; Fetch line/column from SPOSNL
        LD      HL,($5C86)      ; Fetch display file address from DFCCL
        RET                     ; Return.

; ---

;   This section deals with the ZX Printer.

;; PO-F-PR
L0B1D:  LD      C,(IY+$45)      ; Fetch column from P_POSN.
        LD      HL,($5C80)      ; Fetch printer buffer address from PR_CC.
        RET                     ; Return.

; ---------------------------------
; THE 'PRINT ANY CHARACTER' ROUTINE
; ---------------------------------
;   This routine is used to print any character in range 32d - 255d
;   It is only called from PO-ABLE which continues into PO-STORE

;; PO-ANY
L0B24:  CP      $80             ; ASCII ?
        JR      C,L0B65         ; to PO-CHAR is so.

        CP      $90             ; test if a block graphic character.
        JR      NC,L0B52        ; to PO-T&UDG to print tokens and UDGs

; The 16 2*2 mosaic characters 128-143 decimal are formed from
; bits 0-3 of the character.

        LD      B,A             ; save character
        CALL    L0B38           ; routine PO-GR-1 to construct top half
                                ; then bottom half.
        CALL    L0B03           ; routine PO-FETCH fetches print position.
        LD      DE,$5C92        ; MEM-0 is location of 8 bytes of character
        JR      L0B7F           ; to PR-ALL to print to screen or printer

; ---

;; PO-GR-1
L0B38:  LD      HL,$5C92        ; address MEM-0 - a temporary buffer in
                                ; systems variables which is normally used
                                ; by the calculator.
        CALL    L0B3E           ; routine PO-GR-2 to construct top half
                                ; and continue into routine to construct
                                ; bottom half.

;; PO-GR-2
L0B3E:  RR      B               ; rotate bit 0/2 to carry
        SBC     A,A             ; result $00 or $FF
        AND     $0F             ; mask off right hand side
```

```
        LD      C,A             ; store part in C
        RR      B               ; rotate bit 1/3 of original chr to carry
        SBC     A,A             ; result $00 or $FF
        AND     $F0             ; mask off left hand side
        OR      C               ; combine with stored pattern
        LD      C,$04           ; four bytes for top/bottom half

;; PO-GR-3
L0B4C:  LD      (HL),A          ; store bit patterns in temporary buffer
        INC     HL              ; next address
        DEC     C               ; jump back to
        JR      NZ,L0B4C        ; to PO-GR-3 until byte is stored 4 times

        RET                     ; return

; ---

; Tokens and User defined graphics are now separated.

;; PO-T&UDG
L0B52:  SUB     $A5             ; the 'RND' character
        JR      NC,L0B5F        ; to PO-T to print tokens

        ADD     A,$15           ; add 21d to restore to 0 - 20
        PUSH    BC              ; save current print position
        LD      BC,($5C7B)      ; fetch UDG to address bit patterns
        JR      L0B6A           ; to PO-CHAR-2 - common code to lay down
                                ; a bit patterned character

; ---

;; PO-T
L0B5F:  CALL    L0C10           ; routine PO-TOKENS prints tokens
        JP      L0B03           ; exit via a JUMP to PO-FETCH as this routine
                                ; must continue into PO-STORE.
                                ; A JR instruction could be used.

; This point is used to print ASCII characters  32d - 127d.

;; PO-CHAR
L0B65:  PUSH    BC              ; save print position
        LD      BC,($5C36)      ; address CHARS

; This common code is used to transfer the character bytes to memory.

;; PO-CHAR-2
L0B6A:  EX      DE,HL           ; transfer destination address to DE
        LD      HL,$5C3B        ; point to FLAGS
        RES     0,(HL)          ; allow for leading space
        CP      $20             ; is it a space ?
        JR      NZ,L0B76        ; to PO-CHAR-3 if not

        SET     0,(HL)          ; signal no leading space to FLAGS

;; PO-CHAR-3
L0B76:  LD      H,$00           ; set high byte to 0
        LD      L,A             ; character to A
                                ; 0-21 UDG or 32-127 ASCII.
        ADD     HL,HL           ; multiply
        ADD     HL,HL           ; by
        ADD     HL,HL           ; eight
        ADD     HL,BC           ; HL now points to first byte of character
        POP     BC              ; the source address CHARS or UDG
        EX      DE,HL           ; character address to DE
```

```
; ----------------------------------
; THE 'PRINT ALL CHARACTERS' ROUTINE
; ----------------------------------
;   This entry point entered from above to print ASCII and UDGs but also from
;   earlier to print mosaic characters.
;   HL=destination
;   DE=character source
;   BC=line/column

;; PR-ALL
L0B7F:  LD      A,C             ; column to A
        DEC     A               ; move right
        LD      A,$21           ; pre-load with leftmost position
        JR      NZ,L0B93        ; but if not zero to PR-ALL-1

        DEC     B               ; down one line
        LD      C,A             ; load C with $21
        BIT     1,(IY+$01)      ; test FLAGS  - Is printer in use
        JR      Z,L0B93         ; to PR-ALL-1 if not

        PUSH    DE              ; save source address
        CALL    L0ECD           ; routine COPY-BUFF outputs line to printer
        POP     DE              ; restore character source address
        LD      A,C             ; the new column number ($21) to C

;; PR-ALL-1
L0B93:  CP      C               ; this test is really for screen - new line ?
        PUSH    DE              ; save source

        CALL    Z,L0C55         ; routine PO-SCR considers scrolling

        POP     DE              ; restore source
        PUSH    BC              ; save line/column
        PUSH    HL              ; and destination
        LD      A,($5C91)       ; fetch P_FLAG to accumulator
        LD      B,$FF           ; prepare OVER mask in B.
        RRA                     ; bit 0 set if OVER 1
        JR      C,L0BA4         ; to PR-ALL-2

        INC     B               ; set OVER mask to 0

;; PR-ALL-2
L0BA4:  RRA                     ; skip bit 1 of P_FLAG
        RRA                     ; bit 2 is INVERSE
        SBC     A,A             ; will be FF for INVERSE 1 else zero
        LD      C,A             ; transfer INVERSE mask to C
        LD      A,$08           ; prepare to count 8 bytes
        AND     A               ; clear carry to signal screen
        BIT     1,(IY+$01)      ; test FLAGS  - is printer in use ?
        JR      Z,L0BB6         ; to PR-ALL-3 if screen

        SET     1,(IY+$30)      ; update FLAGS2  - signal printer buffer has
                                ; been used.
        SCF                     ; set carry flag to signal printer.

;; PR-ALL-3
L0BB6:  EX      DE,HL           ; now HL=source, DE=destination

;; PR-ALL-4
L0BB7:  EX      AF,AF'          ; save printer/screen flag
        LD      A,(DE)          ; fetch existing destination byte
        AND     B               ; consider OVER
        XOR     (HL)            ; now XOR with source
```

```
        XOR     C                ; now with INVERSE MASK
        LD      (DE),A           ; update screen/printer
        EX      AF,AF'           ; restore flag
        JR      C,L0BD3          ; to PR-ALL-6 - printer address update

        INC     D                ; gives next pixel line down screen

;; PR-ALL-5
L0BC1:  INC     HL               ; address next character byte
        DEC     A                ; the byte count is decremented
        JR      NZ,L0BB7         ; back to PR-ALL-4 for all 8 bytes

        EX      DE,HL            ; destination to HL
        DEC     H                ; bring back to last updated screen position
        BIT     1,(IY+$01)       ; test FLAGS  - is printer in use ?
        CALL    Z,L0BDB          ; if not, call routine PO-ATTR to update
                                 ; corresponding colour attribute.
        POP     HL               ; restore original screen/printer position
        POP     BC               ; and line column
        DEC     C                ; move column to right
        INC     HL               ; increase screen/printer position
        RET                      ; return and continue into PO-STORE
                                 ; within PO-ABLE

; ---

;   This branch is used to update the printer position by 32 places
;   Note. The high byte of the address D remains constant (which it should).

;; PR-ALL-6
L0BD3:  EX      AF,AF'           ; save the flag
        LD      A,$20            ; load A with 32 decimal
        ADD     A,E              ; add this to E
        LD      E,A              ; and store result in E
        EX      AF,AF'           ; fetch the flag
        JR      L0BC1            ; back to PR-ALL-5

; -----------------------------------
; THE 'GET ATTRIBUTE ADDRESS' ROUTINE
; -----------------------------------
;   This routine is entered with the HL register holding the last screen
;   address to be updated by PRINT or PLOT.
;   The Spectrum screen arrangement leads to the L register holding the correct
;   value for the attribute file and it is only necessary to manipulate H to
;   form the correct colour attribute address.

;; PO-ATTR
L0BDB:  LD       A,H             ; fetch high byte $40 - $57
        RRCA                     ; shift
        RRCA                     ; bits 3 and 4
        RRCA                     ; to right.
        AND     $03              ; range is now 0 - 2
        OR      $58              ; form correct high byte for third of screen
        LD      H,A              ; HL is now correct
        LD      DE,($5C8F)       ; make D hold ATTR_T, E hold MASK-T
        LD      A,(HL)           ; fetch existing attribute
        XOR     E                ; apply masks
        AND     D                ;
        XOR     E                ;
        BIT     6,(IY+$57)       ; test P_FLAG  - is this PAPER 9 ??
        JR      Z,L0BFA          ; skip to PO-ATTR-1 if not.

        AND     $C7              ; set paper
        BIT     2,A              ; to contrast with ink
```

```
        JR      NZ,L0BFA        ; skip to PO-ATTR-1

        XOR     $38             ;

;; PO-ATTR-1
L0BFA:  BIT     4,(IY+$57)      ; test P_FLAG  - Is this INK 9 ??
        JR      Z,L0C08         ; skip to PO-ATTR-2 if not

        AND     $F8             ; make ink
        BIT     5,A             ; contrast with paper.
        JR      NZ,L0C08        ; to PO-ATTR-2

        XOR     $07             ;

;; PO-ATTR-2
L0C08:  LD      (HL),A          ; save the new attribute.
        RET                     ; return.

; ---------------------------------
; THE 'MESSAGE PRINTING' SUBROUTINE
; ---------------------------------
;   This entry point is used to print tape, boot-up, scroll? and error messages.
;   On entry the DE register points to an initial step-over byte or the
;   inverted end-marker of the previous entry in the table.
;   Register A contains the message number, often zero to print first message.
;   (HL has nothing important usually P_FLAG)

;; PO-MSG
L0C0A:  PUSH    HL              ; put hi-byte zero on stack to suppress
        LD      H,$00           ; trailing spaces
        EX      (SP),HL         ; ld h,0; push hl would have done ?.
        JR      L0C14           ; forward to PO-TABLE.

; ---

;   This entry point prints the BASIC keywords, '<>' etc. from alt set

;; PO-TOKENS
L0C10:  LD      DE,L0095        ; address: TKN-TABLE
        PUSH    AF              ; save the token number to control
                                ; trailing spaces - see later *

; ->

;; PO-TABLE
L0C14:  CALL    L0C41           ; routine PO-SEARCH will set carry for
                                ; all messages and function words.

        JR      C,L0C22         ; forward to PO-EACH if not a command, '<>' etc.

        LD      A,$20           ; prepare leading space
        BIT     0,(IY+$01)      ; test FLAGS  - leading space if not set

        CALL    Z,L0C3B         ; routine PO-SAVE to print a space without
                                ; disturbing registers.

;; PO-EACH
L0C22:  LD      A,(DE)          ; Fetch character from the table.
        AND     $7F             ; Cancel any inverted bit.

        CALL    L0C3B           ; Routine PO-SAVE to print using the alternate
                                ; set of registers.

        LD      A,(DE)          ; Re-fetch character from table.
```

```
        INC     DE              ; Address next character in the table.

        ADD     A,A             ; Was character inverted ?
                                ; (this also doubles character)
        JR      NC,L0C22        ; back to PO-EACH if not.

        POP     DE              ; * re-fetch trailing space byte to D

        CP      $48             ; was the last character '$' ?
        JR      Z,L0C35         ; forward to PO-TR-SP to consider trailing
                                ; space if so.

        CP      $82             ; was it < 'A' i.e. '#','>','=' from tokens
                                ; or ' ','.' (from tape) or '?' from scroll

        RET     C               ; Return if so as no trailing space required.

;; PO-TR-SP
L0C35:  LD      A,D             ; The trailing space flag (zero if an error msg)

        CP      $03             ; Test against RND, INKEY$ and PI which have no
                                ; parameters and therefore no trailing space.

        RET     C               ; Return if no trailing space.

        LD      A,$20           ; Prepare the space character and continue to
                                ; print and make an indirect return.

; -----------------------------------
; THE 'RECURSIVE PRINTING' SUBROUTINE
; -----------------------------------
;   This routine which is part of PRINT-OUT allows RST $10 to be used
;   recursively to print tokens and the spaces associated with them.
;   It is called on three occasions when the value of DE must be preserved.

;; PO-SAVE
L0C3B:  PUSH    DE              ; Save DE value.
        EXX                     ; Switch in main set

        RST     10H             ; PRINT-A prints using this alternate set.

        EXX                     ; Switch back to this alternate set.
        POP     DE              ; Restore the initial DE value.

        RET                     ; Return.

; ------------
; Table search
; ------------
; This subroutine searches a message or the token table for the
; message number held in A. DE holds the address of the table.

;; PO-SEARCH
L0C41:  PUSH    AF              ; save the message/token number
        EX      DE,HL           ; transfer DE to HL
        INC     A               ; adjust for initial step-over byte

;; PO-STEP
L0C44:  BIT     7,(HL)          ; is character inverted ?
        INC     HL              ; address next
        JR      Z,L0C44         ; back to PO-STEP if not inverted.

        DEC     A               ; decrease counter
        JR      NZ,L0C44        ; back to PO-STEP if not zero
```

```
        EX      DE,HL           ; transfer address to DE
        POP     AF              ; restore message/token number
        CP      $20             ; return with carry set
        RET     C               ; for all messages and function tokens

        LD      A,(DE)          ; test first character of token
        SUB     $41             ; and return with carry set
        RET                     ; if it is less that 'A'
                                ; i.e. '<>', '<=', '>='

; ---------------
; Test for scroll
; ---------------
; This test routine is called when printing carriage return, when considering
; PRINT AT and from the general PRINT ALL characters routine to test if
; scrolling is required, prompting the user if necessary.
; This is therefore using the alternate set.
; The B register holds the current line.

;; PO-SCR
L0C55:  BIT     1,(IY+$01)      ; test FLAGS  - is printer in use ?
        RET     NZ              ; return immediately if so.

        LD      DE,L0DD9        ; set DE to address: CL-SET
        PUSH    DE              ; and push for return address.

        LD      A,B             ; transfer the line to A.
        BIT     0,(IY+$02)      ; test TV_FLAG - lower screen in use ?
        JP      NZ,L0D02        ; jump forward to PO-SCR-4 if so.

        CP      (IY+$31)        ; greater than DF_SZ display file size ?
        JR      C,L0C86         ; forward to REPORT-5 if less.
                                ; 'Out of screen'

        RET     NZ              ; return (via CL-SET) if greater

        BIT     4,(IY+$02)      ; test TV_FLAG  - Automatic listing ?
        JR      Z,L0C88         ; forward to PO-SCR-2 if not.

        LD      E,(IY+$2D)      ; fetch BREG - the count of scroll lines to E.
        DEC     E               ; decrease and jump
        JR      Z,L0CD2         ; to PO-SCR-3 if zero and scrolling required.

        LD      A,$00           ; explicit - select channel zero.
        CALL    L1601           ; routine CHAN-OPEN opens it.

        LD      SP,($5C3F)      ; set stack pointer to LIST_SP

        RES     4,(IY+$02)      ; reset TV_FLAG  - signal auto listing finished.
        RET                     ; return ignoring pushed value, CL-SET
                                ; to MAIN or EDITOR without updating
                                ; print position                    >>

; ---


;; REPORT-5
L0C86:  RST     08H             ; ERROR-1
        DEFB    $04             ; Error Report: Out of screen

; continue here if not an automatic listing.

;; PO-SCR-2
```

```
L0C88:  DEC     (IY+$52)        ; decrease SCR_CT
        JR      NZ,L0CD2        ; forward to PO-SCR-3 to scroll display if
                                ; result not zero.

; now produce prompt.

        LD      A,$18           ; reset
        SUB     B               ; the
        LD      ($5C8C),A       ; SCR_CT scroll count
        LD      HL,($5C8F)      ; L=ATTR_T, H=MASK_T
        PUSH    HL              ; save on stack
        LD      A,($5C91)       ; P_FLAG
        PUSH    AF              ; save on stack to prevent lower screen
                                ; attributes (BORDCR etc.) being applied.
        LD      A,$FD           ; select system channel 'K'
        CALL    L1601           ; routine CHAN-OPEN opens it
        XOR     A               ; clear to address message directly
        LD      DE,L0CF8        ; make DE address: scrl-mssg
        CALL    L0C0A           ; routine PO-MSG prints to lower screen
        SET     5,(IY+$02)      ; set TV_FLAG  - signal lower screen requires
                                ; clearing
        LD      HL,$5C3B        ; make HL address FLAGS
        SET     3,(HL)          ; signal 'L' mode.
        RES     5,(HL)          ; signal 'no new key'.
        EXX                     ; switch to main set.
                                ; as calling chr input from alternative set.
        CALL    L15D4           ; routine WAIT-KEY waits for new key
                                ; Note. this is the right routine but the
                                ; stream in use is unsatisfactory. From the
                                ; choices available, it is however the best.

        EXX                     ; switch back to alternate set.
        CP      $20             ; space is considered as BREAK
        JR      Z,L0D00         ; forward to REPORT-D if so
                                ; 'BREAK - CONT repeats'

        CP      $E2             ; is character 'STOP' ?
        JR      Z,L0D00         ; forward to REPORT-D if so

        OR      $20             ; convert to lower-case
        CP      $6E             ; is character 'n' ?
        JR      Z,L0D00         ; forward to REPORT-D if so else scroll.

        LD      A,$FE           ; select system channel 'S'
        CALL    L1601           ; routine CHAN-OPEN
        POP     AF              ; restore original P_FLAG
        LD      ($5C91),A       ; and save in P_FLAG.
        POP     HL              ; restore original ATTR_T, MASK_T
        LD      ($5C8F),HL      ; and reset ATTR_T, MASK-T as 'scroll?' has
                                ; been printed.

;; PO-SCR-3
L0CD2:  CALL    L0DFE           ; routine CL-SC-ALL to scroll whole display
        LD      B,(IY+$31)      ; fetch DF_SZ to B
        INC     B               ; increase to address last line of display
        LD      C,$21           ; set C to $21 (was $21 from above routine)
        PUSH    BC              ; save the line and column in BC.

        CALL    L0E9B           ; routine CL-ADDR finds display address.

        LD      A,H             ; now find the corresponding attribute byte
        RRCA                    ; (this code sequence is used twice
        RRCA                    ; elsewhere and is a candidate for
        RRCA                    ; a subroutine.)
```

```
        AND     $03             ;
        OR      $58             ;
        LD      H,A             ;

        LD      DE,$5AE0        ; start of last 'line' of attribute area
        LD      A,(DE)          ; get attribute for last line
        LD      C,(HL)          ; transfer to base line of upper part
        LD      B,$20           ; there are thirty two bytes
        EX      DE,HL           ; swap the pointers.

;; PO-SCR-3A
L0CF0:  LD      (DE),A          ; transfer
        LD      (HL),C          ; attributes.
        INC     DE              ; address next.
        INC     HL              ; address next.
        DJNZ    L0CF0           ; loop back to PO-SCR-3A for all adjacent
                                ; attribute lines.

        POP     BC              ; restore the line/column.
        RET                     ; return via CL-SET (was pushed on stack).

; ---

; The message 'scroll?' appears here with last byte inverted.

;; scrl-mssg
L0CF8:  DEFB    $80             ; initial step-over byte.
        DEFM    "scroll"
        DEFB    '?'+$80

;; REPORT-D
L0D00:  RST     08H             ; ERROR-1
        DEFB    $0C             ; Error Report: BREAK - CONT repeats

; continue here if using lower display - A holds line number.

;; PO-SCR-4
L0D02:  CP      $02             ; is line number less than 2 ?
        JR      C,L0C86         ; to REPORT-5 if so
                                ; 'Out of Screen'.

        ADD     A,(IY+$31)      ; add DF_SZ
        SUB     $19             ;
        RET     NC              ; return if scrolling unnecessary

        NEG                     ; Negate to give number of scrolls required.
        PUSH    BC              ; save line/column
        LD      B,A             ; count to B
        LD      HL,($5C8F)      ; fetch current ATTR_T, MASK_T to HL.
        PUSH    HL              ; and save
        LD      HL,($5C91)      ; fetch P_FLAG
        PUSH    HL              ; and save.
                                ; to prevent corruption by input AT

        CALL    L0D4D           ; routine TEMPS sets to BORDCR etc
        LD      A,B             ; transfer scroll number to A.

;; PO-SCR-4A
L0D1C:  PUSH    AF              ; save scroll number.
        LD      HL,$5C6B        ; address DF_SZ
        LD      B,(HL)          ; fetch old value
        LD      A,B             ; transfer to A
        INC     A               ; and increment
        LD      (HL),A          ; then put back.
```

```
        LD      HL,$5C89        ; address S_POSN_hi - line
        CP      (HL)            ; compare
        JR      C,L0D2D         ; forward to PO-SCR-4B if scrolling required

        INC     (HL)            ; else increment S_POSN_hi
        LD      B,$18           ; set count to whole display ??
                                ; Note. should be $17 and the top line will be
                                ; scrolled into the ROM which is harmless on
                                ; the standard set up.
                                ; credit P.Giblin 1984.

;; PO-SCR-4B
L0D2D:  CALL    L0E00           ; routine CL-SCROLL scrolls B lines
        POP     AF              ; restore scroll counter.
        DEC     A               ; decrease
        JR      NZ,L0D1C        ; back to PO-SCR-4A until done

        POP     HL              ; restore original P_FLAG.
        LD      (IY+$57),L      ; and overwrite system variable P_FLAG.

        POP     HL              ; restore original ATTR_T/MASK_T.
        LD      ($5C8F),HL      ; and update system variables.

        LD      BC,($5C88)      ; fetch S_POSN to BC.
        RES     0,(IY+$02)      ; signal to TV_FLAG  - main screen in use.
        CALL    L0DD9           ; call routine CL-SET for upper display.

        SET     0,(IY+$02)      ; signal to TV_FLAG  - lower screen in use.
        POP     BC              ; restore line/column
        RET                     ; return via CL-SET for lower display.

; ----------------------
; Temporary colour items
; ----------------------
; This subroutine is called 11 times to copy the permanent colour items
; to the temporary ones.

;; TEMPS
L0D4D:  XOR     A               ; clear the accumulator
        LD      HL,($5C8D)      ; fetch L=ATTR_P and H=MASK_P
        BIT     0,(IY+$02)      ; test TV_FLAG  - is lower screen in use ?
        JR      Z,L0D5B         ; skip to TEMPS-1 if not

        LD      H,A             ; set H, MASK P, to 00000000.
        LD      L,(IY+$0E)      ; fetch BORDCR to L which is used for lower
                                ; screen.

;; TEMPS-1
L0D5B:  LD      ($5C8F),HL      ; transfer values to ATTR_T and MASK_T

; for the print flag the permanent values are odd bits, temporary even bits.

        LD      HL,$5C91        ; address P_FLAG.
        JR      NZ,L0D65        ; skip to TEMPS-2 if lower screen using A=0.

        LD      A,(HL)          ; else pick up flag bits.
        RRCA                    ; rotate permanent bits to temporary bits.

;; TEMPS-2
L0D65:  XOR     (HL)            ;
        AND     $55             ; BIN 01010101
        XOR     (HL)            ; permanent now as original
        LD      (HL),A          ; apply permanent bits to temporary bits.
        RET                     ; and return.
```

```
; -----------------
; THE 'CLS' COMMAND
; -----------------
;    This command clears the display.
;    The routine is also called during initialization and by the CLEAR command.
;    If it's difficult to write it should be difficult to read.

;; CLS
L0D6B:  CALL    L0DAF           ; Routine CL-ALL clears the entire display and
                                ; sets the attributes to the permanent ones
                                ; from ATTR-P.

;   Having cleared all 24 lines of the display area, continue into the
;   subroutine that clears the lower display area.  Note that at the moment
;   the attributes for the lower lines are the same as upper ones and have
;   to be changed to match the BORDER colour.

; --------------------------
; THE 'CLS-LOWER' SUBROUTINE
; --------------------------
;   This routine is called from INPUT, and from the MAIN execution loop.
;   This is very much a housekeeping routine which clears between 2 and 23
;   lines of the display, setting attributes and correcting situations where
;   errors have occurred while the normal input and output routines have been
;   temporarily diverted to deal with, say colour control codes.

;; CLS-LOWER
L0D6E:  LD      HL,$5C3C        ; address System Variable TV_FLAG.
        RES     5,(HL)          ; TV_FLAG - signal do not clear lower screen.
        SET     0,(HL)          ; TV_FLAG - signal lower screen in use.

        CALL    L0D4D           ; routine TEMPS applies permanent attributes,
                                ; in this case BORDCR to ATTR_T.
                                ; Note. this seems unnecessary and is repeated
                                ; within CL-LINE.

        LD      B,(IY+$31)      ; fetch lower screen display file size DF_SZ

        CALL    L0E44           ; routine CL-LINE clears lines to bottom of the
                                ; display and sets attributes from BORDCR while
                                ; preserving the B register.

        LD      HL,$5AC0        ; set initial attribute address to the leftmost
                                ; cell of second line up.

        LD      A,($5C8D)       ; fetch permanent attribute from ATTR_P.

        DEC     B               ; decrement lower screen display file size.

        JR      L0D8E           ; forward to enter the backfill loop at CLS-3
                                ; where B is decremented again.

; ---

;   The backfill loop is entered at midpoint and ensures, if more than 2
;   lines have been cleared, that any other lines take the permanent screen
;   attributes.

;; CLS-1
L0D87:  LD      C,$20           ; set counter to 32 character cells per line

;; CLS-2
L0D89:  DEC     HL              ; decrease attribute address.
```

```
        LD      (HL),A          ; and place attributes in next line up.
        DEC     C               ; decrease the 32 counter.
        JR      NZ,L0D89        ; loop back to CLS-2 until all 32 cells done.

;; CLS-3
L0D8E:  DJNZ    L0D87           ; decrease B counter and back to CLS-1
                                ; if not zero.

        LD      (IY+$31),$02    ; now set DF_SZ lower screen to 2

; This entry point is also called from CL-ALL below to
; reset the system channel input and output addresses to normal.

;; CL-CHAN
L0D94:  LD      A,$FD           ; select system channel 'K'

        CALL    L1601           ; routine CHAN-OPEN opens it.

        LD      HL,($5C51)      ; fetch CURCHL to HL to address current channel
        LD      DE,L09F4        ; set address to PRINT-OUT for first pass.
        AND     A               ; clear carry for first pass.

;; CL-CHAN-A
L0DA0:  LD      (HL),E          ; Insert the output address on the first pass
        INC     HL              ; or the input address on the second pass.
        LD      (HL),D          ;
        INC     HL              ;

        LD      DE,L10A8        ; fetch address KEY-INPUT for second pass
        CCF                     ; complement carry flag - will set on pass 1.

        JR      C,L0DA0         ; back to CL-CHAN-A if first pass else done.

        LD      BC,$1721        ; line 23 for lower screen
        JR      L0DD9           ; exit via CL-SET to set column
                                ; for lower display

; ---------------------------
; Clearing whole display area
; ---------------------------
; This subroutine called from CLS, AUTO-LIST and MAIN-3
; clears 24 lines of the display and resets the relevant system variables.
; This routine also recovers from an error situation where, for instance, an
; invalid colour or position control code has left the output routine addressing
; PO-TV-2 or PO-CONT.

;; CL-ALL
L0DAF:  LD      HL,$0000        ; Initialize plot coordinates.
        LD      ($5C7D),HL      ; Set system variable COORDS to 0,0.

        RES     0,(IY+$30)      ; update FLAGS2  - signal main screen is clear.

        CALL    L0D94           ; routine CL-CHAN makes channel 'K' 'normal'.

        LD      A,$FE           ; select system channel 'S'
        CALL    L1601           ; routine CHAN-OPEN opens it.

        CALL    L0D4D           ; routine TEMPS applies permanent attributes,
                                ; in this case ATTR_P, to ATTR_T.
                                ; Note. this seems unnecessary.

        LD      B,$18           ; There are 24 lines.

        CALL    L0E44           ; routine CL-LINE clears 24 text lines and sets
```

```
                                ; attributes from ATTR-P.
                                ; This routine preserves B and sets C to $21.

        LD      HL,($5C51)      ; fetch CURCHL make HL address output routine.

        LD      DE,L09F4        ; address: PRINT-OUT
        LD      (HL),E          ; is made
        INC     HL              ; the normal
        LD      (HL),D          ; output address.

        LD      (IY+$52),$01    ; set SCR_CT - scroll count - to default.

;   Note. BC already contains $1821.

        LD      BC,$1821        ; reset column and line to 0,0
                                ; and continue into CL-SET, below, exiting
                                ; via PO-STORE (for the upper screen).

; --------------------
; THE 'CL-SET' ROUTINE
; --------------------
; This important subroutine is used to calculate the character output
; address for screens or printer based on the line/column for screens
; or the column for printer.

;; CL-SET
L0DD9:  LD      HL,$5B00        ; the base address of printer buffer
        BIT     1,(IY+$01)      ; test FLAGS  - is printer in use ?
        JR      NZ,L0DF4        ; forward to CL-SET-2 if so.

        LD      A,B             ; transfer line to A.
        BIT     0,(IY+$02)      ; test TV_FLAG  - lower screen in use ?
        JR      Z,L0DEE         ; skip to CL-SET-1 if handling upper part

        ADD     A,(IY+$31)      ; add DF_SZ for lower screen
        SUB     $18             ; and adjust.

;; CL-SET-1
L0DEE:  PUSH    BC              ; save the line/column.
        LD      B,A             ; transfer line to B
                                ; (adjusted if lower screen)

        CALL    L0E9B           ; routine CL-ADDR calculates address at left
                                ; of screen.
        POP     BC              ; restore the line/column.

;; CL-SET-2
L0DF4:  LD      A,$21           ; the column $01-$21 is reversed
        SUB     C               ; to range $00 - $20
        LD      E,A             ; now transfer to DE
        LD      D,$00           ; prepare for addition
        ADD     HL,DE           ; and add to base address

        JP      L0ADC           ; exit via PO-STORE to update the relevant
                                ; system variables.
; ----------------
; Handle scrolling
; ----------------
; The routine CL-SC-ALL is called once from PO to scroll all the display
; and from the routine CL-SCROLL, once, to scroll part of the display.

;; CL-SC-ALL
L0DFE:  LD      B,$17           ; scroll 23 lines, after 'scroll?'.
```

```
;; CL-SCROLL
L0E00:  CALL    L0E9B           ; routine CL-ADDR gets screen address in HL.
        LD      C,$08           ; there are 8 pixel lines to scroll.

;; CL-SCR-1
L0E05:  PUSH    BC              ; save counters.
        PUSH    HL              ; and initial address.
        LD      A,B             ; get line count.
        AND     $07             ; will set zero if all third to be scrolled.
        LD      A,B             ; re-fetch the line count.
        JR      NZ,L0E19        ; forward to CL-SCR-3 if partial scroll.

; HL points to top line of third and must be copied to bottom of previous 3rd.
; ( so HL = $4800 or $5000 ) ( but also sometimes $4000 )

;; CL-SCR-2
L0E0D:  EX      DE,HL           ; copy HL to DE.
        LD      HL,$F8E0        ; subtract $08 from H and add $E0 to L -
        ADD     HL,DE           ; to make destination bottom line of previous
                                ; third.
        EX      DE,HL           ; restore the source and destination.
        LD      BC,$0020        ; thirty-two bytes are to be copied.
        DEC     A               ; decrement the line count.
        LDIR                    ; copy a pixel line to previous third.

;; CL-SCR-3
L0E19:  EX      DE,HL           ; save source in DE.
        LD      HL,$FFE0        ; load the value -32.
        ADD     HL,DE           ; add to form destination in HL.
        EX      DE,HL           ; switch source and destination
        LD      B,A             ; save the count in B.
        AND     $07             ; mask to find count applicable to current
        RRCA                    ; third and
        RRCA                    ; multiply by
        RRCA                    ; thirty two (same as 5 RLCAs)

        LD      C,A             ; transfer byte count to C ($E0 at most)
        LD      A,B             ; store line count to A
        LD      B,$00           ; make B zero
        LDIR                    ; copy bytes (BC=0, H incremented, L=0)
        LD      B,$07           ; set B to 7, C is zero.
        ADD     HL,BC           ; add 7 to H to address next third.
        AND     $F8             ; has last third been done ?
        JR      NZ,L0E0D        ; back to CL-SCR-2 if not.

        POP     HL              ; restore topmost address.
        INC     H               ; next pixel line down.
        POP     BC              ; restore counts.
        DEC     C               ; reduce pixel line count.
        JR      NZ,L0E05        ; back to CL-SCR-1 if all eight not done.

        CALL    L0E88           ; routine CL-ATTR gets address in attributes
                                ; from current 'ninth line', count in BC.

        LD      HL,$FFE0        ; set HL to the 16-bit value -32.
        ADD     HL,DE           ; and add to form destination address.
        EX      DE,HL           ; swap source and destination addresses.
        LDIR                    ; copy bytes scrolling the linear attributes.
        LD      B,$01           ; continue to clear the bottom line.

; ------------------------------
; THE 'CLEAR TEXT LINES' ROUTINE
; ------------------------------
; This subroutine, called from CL-ALL, CLS-LOWER and AUTO-LIST and above,
```

```
; clears text lines at bottom of display.
; The B register holds on entry the number of lines to be cleared 1-24.

;; CL-LINE
L0E44:  PUSH    BC              ; save line count
        CALL    L0E9B           ; routine CL-ADDR gets top address
        LD      C,$08           ; there are eight screen lines to a text line.

;; CL-LINE-1
L0E4A:  PUSH    BC              ; save pixel line count
        PUSH    HL              ; and save the address
        LD      A,B             ; transfer the line to A (1-24).

;; CL-LINE-2
L0E4D:  AND     $07             ; mask 0-7 to consider thirds at a time
        RRCA                    ; multiply
        RRCA                    ; by 32  (same as five RLCA instructions)
        RRCA                    ; now 32 - 256(0)
        LD      C,A             ; store result in C
        LD      A,B             ; save line in A (1-24)
        LD      B,$00           ; set high byte to 0, prepare for ldir.
        DEC     C               ; decrement count 31-255.
        LD      D,H             ; copy HL
        LD      E,L             ; to DE.
        LD      (HL),$00        ; blank the first byte.
        INC     DE              ; make DE point to next byte.
        LDIR                    ; ldir will clear lines.
        LD      DE,$0701        ; now address next third adjusting
        ADD     HL,DE           ; register E to address left hand side
        DEC     A               ; decrease the line count.
        AND     $F8             ; will be 16, 8 or 0  (AND $18 will do).
        LD      B,A             ; transfer count to B.
        JR      NZ,L0E4D        ; back to CL-LINE-2 if 16 or 8 to do
                                ; the next third.

        POP     HL              ; restore start address.
        INC     H               ; address next line down.
        POP     BC              ; fetch counts.
        DEC     C               ; decrement pixel line count
        JR      NZ,L0E4A        ; back to CL-LINE-1 till all done.

        CALL    L0E88           ; routine CL-ATTR gets attribute address
                                ; in DE and B * 32 in BC.

        LD      H,D             ; transfer the address
        LD      L,E             ; to HL.

        INC     DE              ; make DE point to next location.

        LD      A,($5C8D)       ; fetch ATTR_P - permanent attributes
        BIT     0,(IY+$02)      ; test TV_FLAG  - lower screen in use ?
        JR      Z,L0E80         ; skip to CL-LINE-3 if not.

        LD      A,($5C48)       ; else lower screen uses BORDCR as attribute.

;; CL-LINE-3
L0E80:  LD      (HL),A          ; put attribute in first byte.
        DEC     BC              ; decrement the counter.
        LDIR                    ; copy bytes to set all attributes.
        POP     BC              ; restore the line $01-$24.
        LD      C,$21           ; make column $21. (No use is made of this)
        RET                     ; return to the calling routine.

; ------------------
```

```
; Attribute handling
; ------------------
; This subroutine is called from CL-LINE or CL-SCROLL with the HL register
; pointing to the 'ninth' line and H needs to be decremented before or after
; the division. Had it been done first then either present code or that used
; at the start of PO-ATTR could have been used.
; The Spectrum screen arrangement leads to the L register already holding
; the correct value for the attribute file and it is only necessary
; to manipulate H to form the correct colour attribute address.

;; CL-ATTR
L0E88:  LD      A,H             ; fetch H to A - $48, $50, or $58.
        RRCA                    ; divide by
        RRCA                    ; eight.
        RRCA                    ; $09, $0A or $0B.
        DEC     A               ; $08, $09 or $0A.
        OR      $50             ; $58, $59 or $5A.
        LD      H,A             ; save high byte of attributes.

        EX      DE,HL           ; transfer attribute address to DE
        LD      H,C             ; set H to zero - from last LDIR.
        LD      L,B             ; load L with the line from B.
        ADD     HL,HL           ; multiply
        ADD     HL,HL           ; by
        ADD     HL,HL           ; thirty two
        ADD     HL,HL           ; to give count of attribute
        ADD     HL,HL           ; cells to the end of display.

        LD      B,H             ; transfer the result
        LD      C,L             ; to register BC.

        RET                     ; return.

; -------------------------------
; Handle display with line number
; -------------------------------
; This subroutine is called from four places to calculate the address
; of the start of a screen character line which is supplied in B.

;; CL-ADDR
L0E9B:  LD      A,$18           ; reverse the line number
        SUB     B               ; to range $00 - $17.
        LD      D,A             ; save line in D for later.
        RRCA                    ; multiply
        RRCA                    ; by
        RRCA                    ; thirty-two.

        AND     $E0             ; mask off low bits to make
        LD      L,A             ; L a multiple of 32.

        LD      A,D             ; bring back the line to A.

        AND     $18             ; now $00, $08 or $10.

        OR      $40             ; add the base address of screen.

        LD      H,A             ; HL now has the correct address.
        RET                     ; return.

; -------------------
; Handle COPY command
; -------------------
; This command copies the top 176 lines to the ZX Printer
; It is popular to call this from machine code at point
```

```
; L0EAF with B holding 192 (and interrupts disabled) for a full-screen
; copy. This particularly applies to 16K Spectrums as time-critical
; machine code routines cannot be written in the first 16K of RAM as
; it is shared with the ULA which has precedence over the Z80 chip.

;; COPY
L0EAC:  DI                      ; disable interrupts as this is time-critical.

        LD      B,$B0           ; top 176 lines.
L0EAF:  LD      HL,$4000        ; address start of the display file.

; now enter a loop to handle each pixel line.

;; COPY-1
L0EB2:  PUSH    HL              ; save the screen address.
        PUSH    BC              ; and the line counter.

        CALL    L0EF4           ; routine COPY-LINE outputs one line.

        POP     BC              ; restore the line counter.
        POP     HL              ; and display address.
        INC     H               ; next line down screen within 'thirds'.
        LD      A,H             ; high byte to A.
        AND     $07             ; result will be zero if we have left third.
        JR      NZ,L0EC9        ; forward to COPY-2 if not to continue loop.

        LD      A,L             ; consider low byte first.
        ADD     A,$20           ; increase by 32 - sets carry if back to zero.
        LD      L,A             ; will be next group of 8.
        CCF                     ; complement - carry set if more lines in
                                ; the previous third.
        SBC     A,A             ; will be FF, if more, else 00.
        AND     $F8             ; will be F8 (-8) or 00.
        ADD     A,H             ; that is subtract 8, if more to do in third.
        LD      H,A             ; and reset address.

;; COPY-2
L0EC9:  DJNZ    L0EB2           ; back to COPY-1 for all lines.

        JR      L0EDA           ; forward to COPY-END to switch off the printer
                                ; motor and enable interrupts.
                                ; Note. Nothing else is required.

; ------------------------------
; Pass printer buffer to printer
; ------------------------------
; This routine is used to copy 8 text lines from the printer buffer
; to the ZX Printer. These text lines are mapped linearly so HL does
; not need to be adjusted at the end of each line.

;; COPY-BUFF
L0ECD:  DI                      ; disable interrupts
        LD      HL,$5B00        ; the base address of the Printer Buffer.
        LD      B,$08           ; set count to 8 lines of 32 bytes.

;; COPY-3
L0ED3:  PUSH    BC              ; save counter.

        CALL    L0EF4           ; routine COPY-LINE outputs 32 bytes

        POP     BC              ; restore counter.
        DJNZ    L0ED3           ; loop back to COPY-3 for all 8 lines.
                                ; then stop motor and clear buffer.
```

```
; Note. the COPY command rejoins here, essentially to execute the next
; three instructions.

;; COPY-END
L0EDA:  LD      A,$04           ; output value 4 to port
        OUT     ($FB),A         ; to stop the slowed printer motor.
        EI                      ; enable interrupts.

; --------------------
; Clear Printer Buffer
; --------------------
; This routine clears an arbitrary 256 bytes of memory.
; Note. The routine seems designed to clear a buffer that follows the
; system variables.
; The routine should check a flag or HL address and simply return if COPY
; is in use.
; (T-ADDR-lo would work for the system but not if COPY called externally.)
; As a consequence of this omission the buffer will needlessly
; be cleared when COPY is used and the screen/printer position may be set to
; the start of the buffer and the line number to 0 (B)
; giving an 'Out of Screen' error.
; There seems to have been an unsuccessful attempt to circumvent the use
; of PR_CC_hi.

;; CLEAR-PRB
L0EDF:  LD      HL,$5B00        ; the location of the buffer.
        LD      (IY+$46),L      ; update PR_CC_lo - set to zero - superfluous.
        XOR     A               ; clear the accumulator.
        LD      B,A             ; set count to 256 bytes.

;; PRB-BYTES
L0EE7:  LD      (HL),A          ; set addressed location to zero.
        INC     HL              ; address next byte - Note. not INC L.
        DJNZ    L0EE7           ; back to PRB-BYTES. repeat for 256 bytes.

        RES     1,(IY+$30)      ; set FLAGS2 - signal printer buffer is clear.
        LD      C,$21           ; set the column position .
        JP      L0DD9           ; exit via CL-SET and then PO-STORE.

; -----------------
; Copy line routine
; -----------------
; This routine is called from COPY and COPY-BUFF to output a line of
; 32 bytes to the ZX Printer.
; Output to port $FB -
; bit 7 set - activate stylus.
; bit 7 low - deactivate stylus.
; bit 2 set - stops printer.
; bit 2 reset - starts printer
; bit 1 set - slows printer.
; bit 1 reset - normal speed.

;; COPY-LINE
L0EF4:  LD      A,B             ; fetch the counter 1-8 or 1-176
        CP      $03             ; is it 01 or 02 ?.
        SBC     A,A             ; result is $FF if so else $00.
        AND     $02             ; result is 02 now else 00.
                                ; bit 1 set slows the printer.
        OUT     ($FB),A         ; slow the printer for the
                                ; last two lines.
        LD      D,A             ; save the mask to control the printer later.

;; COPY-L-1
L0EFD:  CALL    L1F54           ; call BREAK-KEY to read keyboard immediately.
```

```
        JR      C,L0F0C         ; forward to COPY-L-2 if 'break' not pressed.

        LD      A,$04           ; else stop the
        OUT     ($FB),A         ; printer motor.
        EI                      ; enable interrupts.
        CALL    L0EDF           ; call routine CLEAR-PRB.
                                ; Note. should not be cleared if COPY in use.

;; REPORT-Dc
L0F0A:  RST     08H             ; ERROR-1
        DEFB    $0C             ; Error Report: BREAK - CONT repeats

;; COPY-L-2
L0F0C:  IN      A,($FB)         ; test now to see if
        ADD     A,A             ; a printer is attached.
        RET     M               ; return if not - but continue with parent
                                ; command.

        JR      NC,L0EFD        ; back to COPY-L-1 if stylus of printer not
                                ; in position.

        LD      C,$20           ; set count to 32 bytes.

;; COPY-L-3
L0F14:  LD      E,(HL)          ; fetch a byte from line.
        INC     HL              ; address next location. Note. not INC L.
        LD      B,$08           ; count the bits.

;; COPY-L-4
L0F18:  RL      D               ; prepare mask to receive bit.
        RL      E               ; rotate leftmost print bit to carry
        RR      D               ; and back to bit 7 of D restoring bit 1

;; COPY-L-5
L0F1E:  IN      A,($FB)         ; read the port.
        RRA                     ; bit 0 to carry.
        JR      NC,L0F1E        ; back to COPY-L-5 if stylus not in position.

        LD      A,D             ; transfer command bits to A.
        OUT     ($FB),A         ; and output to port.
        DJNZ    L0F18           ; loop back to COPY-L-4 for all 8 bits.

        DEC     C               ; decrease the byte count.
        JR      NZ,L0F14        ; back to COPY-L-3 until 256 bits done.

        RET                     ; return to calling routine COPY/COPY-BUFF.


; -----------------------------------
; Editor routine for BASIC and INPUT
; -----------------------------------
; The editor is called to prepare or edit a BASIC line.
; It is also called from INPUT to input a numeric or string expression.
; The behaviour and options are quite different in the various modes
; and distinguished by bit 5 of FLAGX.
;
; This is a compact and highly versatile routine.

;; EDITOR
L0F2C:  LD      HL,($5C3D)      ; fetch ERR_SP
        PUSH    HL              ; save on stack

;; ED-AGAIN
L0F30:  LD      HL,L107F        ; address: ED-ERROR
```

```
        PUSH    HL                ; save address on stack and
        LD      ($5C3D),SP        ; make ERR_SP point to it.

; Note. While in editing/input mode should an error occur then RST 08 will
; update X_PTR to the location reached by CH_ADD and jump to ED-ERROR
; where the error will be cancelled and the loop begin again from ED-AGAIN
; above. The position of the error will be apparent when the lower screen is
; reprinted. If no error then the re-iteration is to ED-LOOP below when
; input is arriving from the keyboard.

;; ED-LOOP
L0F38:  CALL    L15D4             ; routine WAIT-KEY gets key possibly
                                  ; changing the mode.
        PUSH    AF                ; save key.
        LD      D,$00             ; and give a short click based
        LD      E,(IY-$01)        ; on PIP value for duration.
        LD      HL,$00C8          ; and pitch.
        CALL    L03B5             ; routine BEEPER gives click - effective
                                  ; with rubber keyboard.
        POP     AF                ; get saved key value.
        LD      HL,L0F38          ; address: ED-LOOP is loaded to HL.
        PUSH    HL                ; and pushed onto stack.

; At this point there is a looping return address on the stack, an error
; handler and an input stream set up to supply characters.
; The character that has been received can now be processed.

        CP      $18               ; range 24 to 255 ?
        JR      NC,L0F81          ; forward to ADD-CHAR if so.

        CP      $07               ; lower than 7 ?
        JR      C,L0F81           ; forward to ADD-CHAR also.
                                  ; Note. This is a 'bug' and chr$ 6, the comma
                                  ; control character, should have had an
                                  ; entry in the ED-KEYS table.
                                  ; Steven Vickers, 1984, Pitman.

        CP      $10               ; less than 16 ?
        JR      C,L0F92           ; forward to ED-KEYS if editing control
                                  ; range 7 to 15 dealt with by a table

        LD      BC,$0002          ; prepare for ink/paper etc.
        LD      D,A               ; save character in D
        CP      $16               ; is it ink/paper/bright etc. ?
        JR      C,L0F6C           ; forward to ED-CONTR if so

                                  ; leaves 22d AT and 23d TAB
                                  ; which can't be entered via KEY-INPUT.
                                  ; so this code is never normally executed
                                  ; when the keyboard is used for input.

        INC     BC                ; if it was AT/TAB - 3 locations required
        BIT     7,(IY+$37)        ; test FLAGX  - Is this INPUT LINE ?
        JP      Z,L101E           ; jump to ED-IGNORE if not, else

        CALL    L15D4             ; routine WAIT-KEY - input address is KEY-NEXT
                                  ; but is reset to KEY-INPUT
        LD      E,A               ; save first in E

;; ED-CONTR
L0F6C:  CALL    L15D4             ; routine WAIT-KEY for control.
                                  ; input address will be key-next.

        PUSH    DE                ; saved code/parameters
```

```
        LD      HL,($5C5B)      ; fetch address of keyboard cursor from K_CUR
        RES     0,(IY+$07)      ; set MODE to 'L'

        CALL    L1655           ; routine MAKE-ROOM makes 2/3 spaces at cursor

        POP     BC              ; restore code/parameters
        INC     HL              ; address first location
        LD      (HL),B          ; place code (ink etc.)
        INC     HL              ; address next
        LD      (HL),C          ; place possible parameter. If only one
                                ; then DE points to this location also.
        JR      L0F8B           ; forward to ADD-CH-1

; ------------------------
; Add code to current line
; ------------------------
; this is the branch used to add normal non-control characters
; with ED-LOOP as the stacked return address.
; it is also the OUTPUT service routine for system channel 'R'.

;; ADD-CHAR
L0F81:  RES     0,(IY+$07)      ; set MODE to 'L'

X0F85:  LD      HL,($5C5B)      ; fetch address of keyboard cursor from K_CUR

        CALL    L1652           ; routine ONE-SPACE creates one space.

; either a continuation of above or from ED-CONTR with ED-LOOP on stack.

;; ADD-CH-1
L0F8B:  LD      (DE),A          ; load current character to last new location.
        INC     DE              ; address next
        LD      ($5C5B),DE      ; and update K_CUR system variable.
        RET                     ; return - either a simple return
                                ; from ADD-CHAR or to ED-LOOP on stack.

; ---

; a branch of the editing loop to deal with control characters
; using a look-up table.

;; ED-KEYS
L0F92:  LD      E,A             ; character to E.
        LD      D,$00           ; prepare to add.
        LD      HL,L0FA0 - 7    ; base address of editing keys table. $0F99
        ADD     HL,DE           ; add E
        LD      E,(HL)          ; fetch offset to E
        ADD     HL,DE           ; add offset for address of handling routine.
        PUSH    HL              ; push the address on machine stack.
        LD      HL,($5C5B)      ; load address of cursor from K_CUR.
        RET                     ; an make an indirect jump forward to routine.

; ------------------
; Editing keys table
; ------------------
; For each code in the range $07 to $0F this table contains a
; single offset byte to the routine that services that code.
; Note. for what was intended there should also have been an
; entry for chr$ 6 with offset to ed-symbol.

;; ed-keys-t
L0FA0:  DEFB    L0FA9 - $  ; 07d offset $09 to Address: ED-EDIT
        DEFB    L1007 - $  ; 08d offset $66 to Address: ED-LEFT
        DEFB    L100C - $  ; 09d offset $6A to Address: ED-RIGHT
```

```
        DEFB    L0FF3 - $  ; 10d offset $50 to Address: ED-DOWN
        DEFB    L1059 - $  ; 11d offset $B5 to Address: ED-UP
        DEFB    L1015 - $  ; 12d offset $70 to Address: ED-DELETE
        DEFB    L1024 - $  ; 13d offset $7E to Address: ED-ENTER
        DEFB    L1076 - $  ; 14d offset $CF to Address: ED-SYMBOL
        DEFB    L107C - $  ; 15d offset $D4 to Address: ED-GRAPH

; ---------------
; Handle EDIT key
; ---------------
; The user has pressed SHIFT 1 to bring edit line down to bottom of screen.
; Alternatively the user wishes to clear the input buffer and start again.
; Alternatively ...

;; ED-EDIT
L0FA9:  LD      HL,($5C49)      ; fetch E_PPC the last line number entered.
                                ; Note. may not exist and may follow program.
        BIT     5,(IY+$37)      ; test FLAGX  - input mode ?
        JP      NZ,L1097        ; jump forward to CLEAR-SP if not in editor.

        CALL    L196E           ; routine LINE-ADDR to find address of line
                                ; or following line if it doesn't exist.
        CALL    L1695           ; routine LINE-NO will get line number from
                                ; address or previous line if at end-marker.
        LD      A,D             ; if there is no program then DE will
        OR      E               ; contain zero so test for this.
        JP      Z,L1097         ; jump to CLEAR-SP if so.

; Note. at this point we have a validated line number, not just an
; approximation and it would be best to update E_PPC with the true
; cursor line value which would enable the line cursor to be suppressed
; in all situations - see shortly.

        PUSH    HL              ; save address of line.
        INC     HL              ; address low byte of length.
        LD      C,(HL)          ; transfer to C
        INC     HL              ; next to high byte
        LD      B,(HL)          ; transfer to B.
        LD      HL,$000A        ; an overhead of ten bytes
        ADD     HL,BC           ; is added to length.
        LD      B,H             ; transfer adjusted value
        LD      C,L             ; to BC register.
        CALL    L1F05           ; routine TEST-ROOM checks free memory.
        CALL    L1097           ; routine CLEAR-SP clears editing area.
        LD      HL,($5C51)      ; address CURCHL
        EX      (SP),HL         ; swap with line address on stack
        PUSH    HL              ; save line address underneath

        LD      A,$FF           ; select system channel 'R'
        CALL    L1601           ; routine CHAN-OPEN opens it

        POP     HL              ; drop line address
        DEC     HL              ; make it point to first byte of line num.
        DEC     (IY+$0F)        ; decrease E_PPC_lo to suppress line cursor.
                                ; Note. ineffective when E_PPC is one
                                ; greater than last line of program perhaps
                                ; as a result of a delete.
                                ; credit. Paul Harrison 1982.

        CALL    L1855           ; routine OUT-LINE outputs the BASIC line
                                ; to the editing area.
        INC     (IY+$0F)        ; restore E_PPC_lo to the previous value.
        LD      HL,($5C59)      ; address E_LINE in editing area.
        INC     HL              ; advance
```

```
        INC     HL                ; past space
        INC     HL                ; and digit characters
        INC     HL                ; of line number.

        LD      ($5C5B),HL        ; update K_CUR to address start of BASIC.
        POP     HL                ; restore the address of CURCHL.
        CALL    L1615             ; routine CHAN-FLAG sets flags for it.
        RET                       ; RETURN to ED-LOOP.

; -------------------
; Cursor down editing
; -------------------
;   The BASIC lines are displayed at the top of the screen and the user
;   wishes to move the cursor down one line in edit mode.
;   With INPUT LINE, this key must be used instead of entering STOP.

;; ED-DOWN
L0FF3:  BIT     5,(IY+$37)        ; test FLAGX  - Input Mode ?
        JR      NZ,L1001          ; skip to ED-STOP if so

        LD      HL,$5C49          ; address E_PPC - 'current line'
        CALL    L190F             ; routine LN-FETCH fetches number of next
                                  ; line or same if at end of program.
        JR      L106E             ; forward to ED-LIST to produce an
                                  ; automatic listing.

; ---

;; ED-STOP
L1001:  LD      (IY+$00),$10      ; set ERR_NR to 'STOP in INPUT' code
        JR      L1024             ; forward to ED-ENTER to produce error.

; -------------------
; Cursor left editing
; -------------------
; This acts on the cursor in the lower section of the screen in both
; editing and input mode.

;; ED-LEFT
L1007:  CALL    L1031             ; routine ED-EDGE moves left if possible
        JR      L1011             ; forward to ED-CUR to update K-CUR
                                  ; and return to ED-LOOP.

; --------------------
; Cursor right editing
; --------------------
; This acts on the cursor in the lower screen in both editing and input
; mode and moves it to the right.

;; ED-RIGHT
L100C:  LD      A,(HL)            ; fetch addressed character.
        CP      $0D               ; is it carriage return ?
        RET     Z                 ; return if so to ED-LOOP

        INC     HL                ; address next character

;; ED-CUR
L1011:  LD      ($5C5B),HL        ; update K_CUR system variable
        RET                       ; return to ED-LOOP

; --------------
; DELETE editing
; --------------
; This acts on the lower screen and deletes the character to left of
```

```
; cursor. If control characters are present these are deleted first
; leaving the naked parameter (0-7) which appears as a '?' except in the
; case of chr$ 6 which is the comma control character. It is not mandatory
; to delete these second characters.

;; ED-DELETE
L1015:  CALL    L1031           ; routine ED-EDGE moves cursor to left.
        LD      BC,$0001        ; of character to be deleted.
        JP      L19E8           ; to RECLAIM-2 reclaim the character.

; ------------------------------------------
; Ignore next 2 codes from key-input routine
; ------------------------------------------
; Since AT and TAB cannot be entered this point is never reached
; from the keyboard. If inputting from a tape device or network then
; the control and two following characters are ignored and processing
; continues as if a carriage return had been received.
; Here, perhaps, another Spectrum has said print #15; AT 0,0; "This is yellow"
; and this one is interpreting input #15; a$.

;; ED-IGNORE
L101E:  CALL    L15D4           ; routine WAIT-KEY to ignore keystroke.
        CALL    L15D4           ; routine WAIT-KEY to ignore next key.

; -------------
; Enter/newline
; -------------
; The enter key has been pressed to have BASIC line or input accepted.

;; ED-ENTER
L1024:  POP     HL              ; discard address ED-LOOP
        POP     HL              ; drop address ED-ERROR

;; ED-END
L1026:  POP     HL              ; the previous value of ERR_SP
        LD      ($5C3D),HL      ; is restored to ERR_SP system variable
        BIT     7,(IY+$00)      ; is ERR_NR $FF (= 'OK') ?
        RET     NZ              ; return if so

        LD      SP,HL           ; else put error routine on stack
        RET                     ; and make an indirect jump to it.

; -----------------------------
; Move cursor left when editing
; -----------------------------
; This routine moves the cursor left. The complication is that it must
; not position the cursor between control codes and their parameters.
; It is further complicated in that it deals with TAB and AT characters
; which are never present from the keyboard.
; The method is to advance from the beginning of the line each time,
; jumping one, two, or three characters as necessary saving the original
; position at each jump in DE. Once it arrives at the cursor then the next
; legitimate leftmost position is in DE.

;; ED-EDGE
L1031:  SCF                     ; carry flag must be set to call the nested
        CALL    L1195           ; subroutine SET-DE.
                                ; if input   then DE=WORKSP
                                ; if editing then DE=E_LINE
        SBC     HL,DE           ; subtract address from start of line
        ADD     HL,DE           ; and add back.
        INC     HL              ; adjust for carry.
        POP     BC              ; drop return address
        RET     C               ; return to ED-LOOP if already at left
```

```
                                ; of line.

        PUSH    BC              ; resave return address - ED-LOOP.
        LD      B,H             ; transfer HL - cursor address
        LD      C,L             ; to BC register pair.
                                ; at this point DE addresses start of line.

;; ED-EDGE-1
L103E:  LD      H,D             ; transfer DE - leftmost pointer
        LD      L,E             ; to HL
        INC     HL              ; address next leftmost character to
                                ; advance position each time.
        LD      A,(DE)          ; pick up previous in A
        AND     $F0             ; lose the low bits
        CP      $10             ; is it INK to TAB $10-$1F ?
                                ; that is, is it followed by a parameter ?
        JR      NZ,L1051        ; to ED-EDGE-2 if not
                                ; HL has been incremented once

        INC     HL              ; address next as at least one parameter.

; in fact since 'tab' and 'at' cannot be entered the next section seems
; superfluous.
; The test will always fail and the jump to ED-EDGE-2 will be taken.

        LD      A,(DE)          ; reload leftmost character
        SUB     $17             ; decimal 23 ('tab')
        ADC     A,$00           ; will be 0 for 'tab' and 'at'.
        JR      NZ,L1051        ; forward to ED-EDGE-2 if not
                                ; HL has been incremented twice

        INC     HL              ; increment a third time for 'at'/'tab'

;; ED-EDGE-2
L1051:  AND     A               ; prepare for true subtraction
        SBC     HL,BC           ; subtract cursor address from pointer
        ADD     HL,BC           ; and add back
                                ; Note when HL matches the cursor position BC,
                                ; there is no carry and the previous
                                ; position is in DE.
        EX      DE,HL           ; transfer result to DE if looping again.
                                ; transfer DE to HL to be used as K-CUR
                                ; if exiting loop.
        JR      C,L103E         ; back to ED-EDGE-1 if cursor not matched.

        RET                     ; return.

; -----------------
; Cursor up editing
; -----------------
; The main screen displays part of the BASIC program and the user wishes
; to move up one line scrolling if necessary.
; This has no alternative use in input mode.

;; ED-UP
L1059:  BIT     5,(IY+$37)      ; test FLAGX  - input mode ?
        RET     NZ              ; return if not in editor - to ED-LOOP.

        LD      HL,($5C49)      ; get current line from E_PPC
        CALL    L196E           ; routine LINE-ADDR gets address
        EX      DE,HL           ; and previous in DE
        CALL    L1695           ; routine LINE-NO gets prev line number
        LD      HL,$5C4A        ; set HL to E_PPC_hi as next routine stores
                                ; top first.
```

```
        CALL    L191C           ; routine LN-STORE loads DE value to HL
                                ; high byte first - E_PPC_lo takes E

; this branch is also taken from ed-down.

;; ED-LIST
L106E:  CALL    L1795           ; routine AUTO-LIST lists to upper screen
                                ; including adjusted current line.
        LD      A,$00           ; select lower screen again
        JP      L1601           ; exit via CHAN-OPEN to ED-LOOP

; --------------------------------
; Use of symbol and graphics codes
; --------------------------------
; These will not be encountered with the keyboard but would be handled
; otherwise as follows.
; As noted earlier, Vickers says there should have been an entry in
; the KEYS table for chr$ 6 which also pointed here.
; If, for simplicity, two Spectrums were both using #15 as a bi-directional
; channel connected to each other:-
; then when the other Spectrum has said PRINT #15; x, y
; input #15; i ; j  would treat the comma control as a newline and the
; control would skip to input j.
; You can get round the missing chr$ 6 handler by sending multiple print
; items separated by a newline '.

; chr$14 would have the same functionality.

; This is chr$ 14.
;; ED-SYMBOL
L1076:  BIT     7,(IY+$37)      ; test FLAGX - is this INPUT LINE ?
        JR      Z,L1024         ; back to ED-ENTER if not to treat as if
                                ; enter had been pressed.
                                ; else continue and add code to buffer.

; Next is chr$ 15
; Note that ADD-CHAR precedes the table so we can't offset to it directly.

;; ED-GRAPH
L107C:  JP      L0F81           ; jump back to ADD-CHAR

; --------------------
; Editor error routine
; --------------------
; If an error occurs while editing, or inputting, then ERR_SP
; points to the stack location holding address ED_ERROR.

;; ED-ERROR
L107F:  BIT     4,(IY+$30)      ; test FLAGS2  - is K channel in use ?
        JR      Z,L1026         ; back to ED-END if not.

; but as long as we're editing lines or inputting from the keyboard, then
; we've run out of memory so give a short rasp.

        LD      (IY+$00),$FF    ; reset ERR_NR to 'OK'.
        LD      D,$00           ; prepare for beeper.
        LD      E,(IY-$02)      ; use RASP value.
        LD      HL,$1A90        ; set a duration.
        CALL    L03B5           ; routine BEEPER emits a warning rasp.
        JP      L0F30           ; to ED-AGAIN to re-stack address of
                                ; this routine and make ERR_SP point to it.

; ---------------------
; Clear edit/work space
```

```
; ---------------------
; The editing area or workspace is cleared depending on context.
; This is called from ED-EDIT to clear workspace if edit key is
; used during input, to clear editing area if no program exists
; and to clear editing area prior to copying the edit line to it.
; It is also used by the error routine to clear the respective
; area depending on FLAGX.

;; CLEAR-SP
L1097:  PUSH    HL                  ; preserve HL
        CALL    L1190               ; routine SET-HL
                                    ; if in edit   HL = WORKSP-1, DE = E_LINE
                                    ; if in input  HL = STKBOT,   DE = WORKSP
        DEC     HL                  ; adjust
        CALL    L19E5               ; routine RECLAIM-1 reclaims space
        LD      ($5C5B),HL          ; set K_CUR to start of empty area
        LD      (IY+$07),$00        ; set MODE to 'KLC'
        POP     HL                  ; restore HL.
        RET                         ; return.

; ---------------------------
; THE 'KEYBOARD INPUT' ROUTINE
; ---------------------------
; This is the service routine for the input stream of the keyboard channel 'K'.

;; KEY-INPUT
L10A8:  BIT     3,(IY+$02)          ; test TV_FLAG  - has a key been pressed in
                                    ; editor ?

        CALL    NZ,L111D            ; routine ED-COPY, if so, to reprint the lower
                                    ; screen at every keystroke/mode change.

        AND     A                   ; clear carry flag - required exit condition.

        BIT     5,(IY+$01)          ; test FLAGS  - has a new key been pressed ?
        RET     Z                   ; return if not.                        >>

        LD      A,($5C08)           ; system variable LASTK will hold last key -
                                    ; from the interrupt routine.

        RES     5,(IY+$01)          ; update FLAGS  - reset the new key flag.
        PUSH    AF                  ; save the input character.

        BIT     5,(IY+$02)          ; test TV_FLAG  - clear lower screen ?

        CALL    NZ,L0D6E            ; routine CLS-LOWER if so.

        POP     AF                  ; restore the character code.
        CP      $20                 ; if space or higher then
        JR      NC,L111B            ; forward to KEY-DONE2 and return with carry
                                    ; set to signal key-found.

        CP      $10                 ; with 16d INK and higher skip
        JR      NC,L10FA            ; forward to KEY-CONTR.

        CP      $06                 ; for 6 - 15d
        JR      NC,L10DB            ; skip forward to KEY-M-CL to handle Modes
                                    ; and CapsLock.

; that only leaves 0-5, the flash bright inverse switches.

        LD      B,A                 ; save character in B
        AND     $01                 ; isolate the embedded parameter (0/1).
        LD      C,A                 ; and store in C
```

```
        LD      A,B             ; re-fetch copy (0-5)
        RRA                     ; halve it 0, 1 or 2.
        ADD     A,$12           ; add 18d gives 'flash', 'bright'
                                ; and 'inverse'.
        JR      L1105           ; forward to KEY-DATA with the
                                ; parameter (0/1) in C.

; ---

; Now separate capslock 06 from modes 7-15.

;; KEY-M-CL
L10DB:  JR      NZ,L10E6        ; forward to KEY-MODE if not 06 (capslock)

        LD      HL,$5C6A        ; point to FLAGS2
        LD      A,$08           ; value 00001000
        XOR     (HL)            ; toggle BIT 3 of FLAGS2 the capslock bit
        LD      (HL),A          ; and store result in FLAGS2 again.
        JR      L10F4           ; forward to KEY-FLAG to signal no-key.

; ---

;; KEY-MODE
L10E6:  CP      $0E             ; compare with chr 14d
        RET     C               ; return with carry set "key found" for
                                ; codes 7 - 13d leaving 14d and 15d
                                ; which are converted to mode codes.

        SUB     $0D             ; subtract 13d leaving 1 and 2
                                ; 1 is 'E' mode, 2 is 'G' mode.
        LD      HL,$5C41        ; address the MODE system variable.
        CP      (HL)            ; compare with existing value before
        LD      (HL),A          ; inserting the new value.
        JR      NZ,L10F4        ; forward to KEY-FLAG if it has changed.

        LD      (HL),$00        ; else make MODE zero - KLC mode
                                ; Note. while in Extended/Graphics mode,
                                ; the Extended Mode/Graphics key is pressed
                                ; again to get out.

;; KEY-FLAG
L10F4:  SET     3,(IY+$02)      ; update TV_FLAG  - show key state has changed
        CP      A               ; clear carry and reset zero flags -
                                ; no actual key returned.
        RET                     ; make the return.

; ---

; now deal with colour controls - 16-23 ink, 24-31 paper

;; KEY-CONTR
L10FA:  LD      B,A             ; make a copy of character.
        AND     $07             ; mask to leave bits 0-7
        LD      C,A             ; and store in C.
        LD      A,$10           ; initialize to 16d - INK.
        BIT     3,B             ; was it paper ?
        JR      NZ,L1105        ; forward to KEY-DATA with INK 16d and
                                ; colour in C.

        INC     A               ; else change from INK to PAPER (17d) if so.

;; KEY-DATA
L1105:  LD      (IY-$2D),C      ; put the colour (0-7)/state(0/1) in KDATA
        LD      DE,L110D        ; address: KEY-NEXT will be next input stream
```

```
        JR      L1113           ; forward to KEY-CHAN to change it ...

; ---

; ... so that INPUT_AD directs control to here at next call to WAIT-KEY

;; KEY-NEXT
L110D:  LD      A,($5C0D)       ; pick up the parameter stored in KDATA.
        LD      DE,L10A8        ; address: KEY-INPUT will be next input stream
                                ; continue to restore default channel and
                                ; make a return with the control code.

;; KEY-CHAN
L1113:  LD      HL,($5C4F)      ; address start of CHANNELS area using CHANS
                                ; system variable.
                                ; Note. One might have expected CURCHL to
                                ; have been used.
        INC     HL              ; step over the
        INC     HL              ; output address
        LD      (HL),E          ; and update the input
        INC     HL              ; routine address for
        LD      (HL),D          ; the next call to WAIT-KEY.

;; KEY-DONE2
L111B:  SCF                     ; set carry flag to show a key has been found
        RET                     ; and return.

; --------------------
; Lower screen copying
; --------------------
; This subroutine is called whenever the line in the editing area or
; input workspace is required to be printed to the lower screen.
; It is by calling this routine after any change that the cursor, for
; instance, appears to move to the left.
; Remember the edit line will contain characters and tokens
; e.g. "1000 LET a=1" is 8 characters.

;; ED-COPY
L111D:  CALL    L0D4D           ; routine TEMPS sets temporary attributes.
        RES     3,(IY+$02)      ; update TV_FLAG  - signal no change in mode
        RES     5,(IY+$02)      ; update TV_FLAG  - signal don't clear lower
                                ; screen.
        LD      HL,($5C8A)      ; fetch SPOSNL
        PUSH    HL              ; and save on stack.

        LD      HL,($5C3D)      ; fetch ERR_SP
        PUSH    HL              ; and save also
        LD      HL,L1167        ; address: ED-FULL
        PUSH    HL              ; is pushed as the error routine
        LD      ($5C3D),SP      ; and ERR_SP made to point to it.

        LD      HL,($5C82)      ; fetch ECHO_E
        PUSH    HL              ; and push also

        SCF                     ; set carry flag to control SET-DE
        CALL    L1195           ; call routine SET-DE
                                ; if in input DE = WORKSP
                                ; if in edit  DE = E_LINE
        EX      DE,HL           ; start address to HL

        CALL    L187D           ; routine OUT-LINE2 outputs entire line up to
                                ; carriage return including initial
                                ; characterized line number when present.
        EX      DE,HL           ; transfer new address to DE
```

```
        CALL    L18E1           ; routine OUT-CURS considers a
                                ; terminating cursor.

        LD      HL,($5C8A)      ; fetch updated SPOSNL
        EX      (SP),HL         ; exchange with ECHO_E on stack
        EX      DE,HL           ; transfer ECHO_E to DE
        CALL    L0D4D           ; routine TEMPS to re-set attributes
                                ; if altered.

; the lower screen was not cleared, at the outset, so if deleting then old
; text from a previous print may follow this line and requires blanking.

;; ED-BLANK
L1150:  LD      A,($5C8B)       ; fetch SPOSNL_hi is current line
        SUB     D               ; compare with old
        JR      C,L117C         ; forward to ED-C-DONE if no blanking

        JR      NZ,L115E        ; forward to ED-SPACES if line has changed

        LD      A,E             ; old column to A
        SUB     (IY+$50)        ; subtract new in SPOSNL_lo
        JR      NC,L117C        ; forward to ED-C-DONE if no backfilling.

;; ED-SPACES
L115E:  LD      A,$20           ; prepare a space.
        PUSH    DE              ; save old line/column.
        CALL    L09F4           ; routine PRINT-OUT prints a space over
                                ; any text from previous print.
                                ; Note. Since the blanking only occurs when
                                ; using $09F4 to print to the lower screen,
                                ; there is no need to vector via a RST 10
                                ; and we can use this alternate set.
        POP     DE              ; restore the old line column.
        JR      L1150           ; back to ED-BLANK until all old text blanked.

; -------------------------------
; THE 'EDITOR-FULL' ERROR ROUTINE
; -------------------------------
;   This is the error routine addressed by ERR_SP.  This is not for the out of
;   memory situation as we're just printing.  The pitch and duration are exactly
;   the same as used by ED-ERROR from which this has been augmented.  The
;   situation is that the lower screen is full and a rasp is given to suggest
;   that this is perhaps not the best idea you've had that day.

;; ED-FULL
L1167:  LD      D,$00           ; prepare to moan.
        LD      E,(IY-$02)      ; fetch RASP value.
        LD      HL,$1A90        ; set duration.

        CALL    L03B5           ; routine BEEPER.

        LD      (IY+$00),$FF    ; clear ERR_NR.
        LD      DE,($5C8A)      ; fetch SPOSNL.
        JR      L117E           ; forward to ED-C-END

; -------

; the exit point from line printing continues here.

;; ED-C-DONE
L117C:  POP     DE              ; fetch new line/column.
        POP     HL              ; fetch the error address.

; the error path rejoins here.
```

```
;; ED-C-END
L117E:  POP     HL                  ; restore the old value of ERR_SP.
        LD      ($5C3D),HL          ; update the system variable ERR_SP

        POP     BC                  ; old value of SPOSN_L
        PUSH    DE                  ; save new value

        CALL    L0DD9               ; routine CL-SET and PO-STORE
                                    ; update ECHO_E and SPOSN_L from BC

        POP     HL                  ; restore new value
        LD      ($5C82),HL          ; and overwrite ECHO_E

        LD      (IY+$26),$00        ; make error pointer X_PTR_hi out of bounds

        RET                         ; return

; -----------------------------------------------
; Point to first and last locations of work space
; -----------------------------------------------
;   These two nested routines ensure that the appropriate pointers are
;   selected for the editing area or workspace. The routines that call
;   these routines are designed to work on either area.

; this routine is called once

;; SET-HL
L1190:  LD      HL,($5C61)          ; fetch WORKSP to HL.
        DEC     HL                  ; point to last location of editing area.
        AND     A                   ; clear carry to limit exit points to first
                                    ; or last.

; this routine is called with carry set and exits at a conditional return.

;; SET-DE
L1195:  LD      DE,($5C59)          ; fetch E_LINE to DE
        BIT     5,(IY+$37)          ; test FLAGX  - Input Mode ?
        RET     Z                   ; return now if in editing mode

        LD      DE,($5C61)          ; fetch WORKSP to DE
        RET     C                   ; return if carry set ( entry = set-de)

        LD      HL,($5C63)          ; fetch STKBOT to HL as well
        RET                         ; and return  (entry = set-hl (in input))

; ---------------------------------
; THE 'REMOVE FLOATING POINT' ROUTINE
; ---------------------------------
;   When a BASIC LINE or the INPUT BUFFER is parsed any numbers will have
;   an invisible chr 14d inserted after them and the 5-byte integer or
;   floating point form inserted after that.  Similar invisible value holders
;   are also created after the numeric and string variables in a DEF FN list.
;   This routine removes these 'compiled' numbers from the edit line or
;   input workspace.

;; REMOVE-FP
L11A7:  LD      A,(HL)              ; fetch character
        CP      $0E                 ; is it the CHR$ 14 number marker ?
        LD      BC,$0006            ; prepare to strip six bytes

        CALL    Z,L19E8             ; routine RECLAIM-2 reclaims bytes if CHR$ 14.

        LD      A,(HL)              ; reload next (or same) character
```

```
        INC     HL              ; and advance address
        CP      $0D             ; end of line or input buffer ?
        JR      NZ,L11A7        ; back to REMOVE-FP until entire line done.

        RET                     ; return.


; *********************************
; ** Part 6. EXECUTIVE ROUTINES  **
; *********************************


; The memory.
;
; +---------+-----------+------------+--------------+-------------+--
; | BASIC   |  Display  | Attributes | ZX Printer   |    System   |
; |  ROM    |   File    |    File    |   Buffer     |  Variables  |
; +---------+-----------+------------+--------------+-------------+--
; ^         ^           ^            ^              ^             ^
; $0000   $4000       $5800        $5B00          $5C00         $5CB6 = CHANS
;
;
;  --+----------+---+---------+-----------+---+------------+--+---+--
;    | Channel  |$80|  BASIC  | Variables |$80| Edit Line  |NL|$80|
;    |   Info   |   | Program |   Area    |   | or Command |  |   |
;  --+----------+---+---------+-----------+---+------------+--+---+--
;    ^              ^         ^               ^                   ^
;  CHANS           PROG      VARS           E_LINE              WORKSP
;
;
;                             ---5-->         <---2---  <--3---
;  --+-------+--+------------+-------+-------+---------+-------+-+---+------+
;    | INPUT |NL| Temporary  | Calc. | Spare | Machine | GOSUB |?|$3E| UDGs |
;    | data  |  | Work Space | Stack |       |  Stack  | Stack | |   |      |
;  --+-------+--+------------+-------+-------+---------+-------+-+---+------+
;    ^                       ^       ^       ^                   ^   ^      ^
;  WORKSP                  STKBOT  STKEND   sp               RAMTOP UDG  P_RAMT
;

; -----------------
; THE 'NEW' COMMAND
; -----------------
;   The NEW command is about to set all RAM below RAMTOP to zero and then
;   re-initialize the system.  All RAM above RAMTOP should, and will be,
;   preserved.
;   There is nowhere to store values in RAM or on the stack which becomes
;   inoperable. Similarly PUSH and CALL instructions cannot be used to store
;   values or section common code. The alternate register set is the only place
;   available to store 3 persistent 16-bit system variables.

;; NEW
L11B7:  DI                      ; Disable Interrupts - machine stack will be
                                ; cleared.
        LD      A,$FF           ; Flag coming from NEW.
        LD      DE,($5CB2)      ; Fetch RAMTOP as top value.
        EXX                     ; Switch in alternate set.
        LD      BC,($5CB4)      ; Fetch P-RAMT differs on 16K/48K machines.
        LD      DE,($5C38)      ; Fetch RASP/PIP.
        LD      HL,($5C7B)      ; Fetch UDG    differs on 16K/48K machines.
        EXX                     ; Switch back to main set and continue into...

; ----------------------
; THE 'START-NEW' BRANCH
; ----------------------
```

```
;   This branch is taken from above and from RST 00h.
;   The common code tests RAM and sets it to zero re-initializing all the
;   non-zero system variables and channel information.  The A register flags
;   if coming from START or NEW.

;; START-NEW
L11CB:  LD      B,A             ; Save the flag to control later branching.

        LD      A,$07           ; Select a white border
        OUT     ($FE),A         ; and set it now by writing to a port.

        LD      A,$3F           ; Load the accumulator with last page in ROM.
        LD      I,A             ; Set the I register - this remains constant
                                ; and can't be in the range $40 - $7F as 'snow'
                                ; appears on the screen.

        NOP                     ; These seem unnecessary.
        NOP                     ;
        NOP                     ;
        NOP                     ;
        NOP                     ;
        NOP                     ;

; -----------------------
; THE 'RAM CHECK' SECTION
; -----------------------
;   Typically, a Spectrum will have 16K or 48K of RAM and this code will test
;   it all till it finds an unpopulated location or, less likely, a faulty
;   location.  Usually it stops when it reaches the top $FFFF, or in the case
;   of NEW the supplied top value.  The entire screen turns black with
;   sometimes red stripes on black paper just visible.

;; ram-check
L11DA:  LD      H,D             ; Transfer the top value to the HL register
        LD      L,E             ; pair.

;; RAM-FILL
L11DC:  LD      (HL),$02        ; Load memory with $02 - red ink on black paper.
        DEC     HL              ; Decrement memory address.
        CP      H               ; Have we reached ROM - $3F ?
        JR      NZ,L11DC        ; Back to RAM-FILL if not.

;; RAM-READ
L11E2:  AND     A               ; Clear carry - prepare to subtract.
        SBC     HL,DE           ; subtract and add back setting
        ADD     HL,DE           ; carry when back at start.
        INC     HL              ; and increment for next iteration.
        JR      NC,L11EF        ; forward to RAM-DONE if we've got back to
                                ; starting point with no errors.

        DEC     (HL)            ; decrement to 1.
        JR      Z,L11EF         ; forward to RAM-DONE if faulty.

        DEC     (HL)            ; decrement to zero.
        JR      Z,L11E2         ; back to RAM-READ if zero flag was set.

;; RAM-DONE
L11EF:  DEC     HL              ; step back to last valid location.
        EXX                     ; regardless of state, set up possibly
                                ; stored system variables in case from NEW.
        LD      ($5CB4),BC      ; insert P-RAMT.
        LD      ($5C38),DE      ; insert RASP/PIP.
        LD      ($5C7B),HL      ; insert UDG.
        EXX                     ; switch in main set.
```

```
        INC     B               ; now test if we arrived here from NEW.
        JR      Z,L1219         ; forward to RAM-SET if we did.

;   This section applies to START only.

        LD      ($5CB4),HL      ; set P-RAMT to the highest working RAM
                                ; address.
        LD      DE,$3EAF        ; address of last byte of 'U' bitmap in ROM.
        LD      BC,$00A8        ; there are 21 user defined graphics.
        EX      DE,HL           ; switch pointers and make the UDGs a
        LDDR                    ; copy of the standard characters A - U.
        EX      DE,HL           ; switch the pointer to HL.
        INC     HL              ; update to start of 'A' in RAM.
        LD      ($5C7B),HL      ; make UDG system variable address the first
                                ; bitmap.
        DEC     HL              ; point at RAMTOP again.

        LD      BC,$0040        ; set the values of
        LD      ($5C38),BC      ; the PIP and RASP system variables.

;   The NEW command path rejoins here.

;; RAM-SET
L1219:  LD      ($5CB2),HL      ; set system variable RAMTOP to HL.

;   New
;   Note. this entry point is a disabled Warm Restart that was almost certainly
;   once pointed to by the System Variable NMIADD.  It would be essential that
;   any NMI Handler would perform the tasks from here to the EI instruction
;   below.

;; NMI_VECT
L121C:
        LD      HL,$3C00        ; a strange place to set the pointer to the
        LD      ($5C36),HL      ; character set, CHARS - as no printing yet.

        LD      HL,($5CB2)      ; fetch RAMTOP to HL again as we've lost it.

        LD      (HL),$3E        ; top of user ram holds GOSUB end marker
                                ; an impossible line number - see RETURN.
                                ; no significance in the number $3E. It has
                                ; been traditional since the ZX80.

        DEC     HL              ; followed by empty byte (not important).
        LD      SP,HL           ; set up the machine stack pointer.
        DEC     HL              ;
        DEC     HL              ;
        LD      ($5C3D),HL      ; ERR_SP is where the error pointer is
                                ; at moment empty - will take address MAIN-4
                                ; at the call preceding that address,
                                ; although interrupts and calls will make use
                                ; of this location in meantime.

        IM      1               ; select interrupt mode 1.

        LD      IY,$5C3A        ; set IY to ERR_NR. IY can reach all standard
                                ; system variables but shadow ROM system
                                ; variables will be mostly out of range.

        EI                      ; enable interrupts now that we have a stack.

;   If, as suggested above, the NMI service routine pointed to this section of
;   code then a decision would have to be made at this point to jump forward,
;   in a Warm Restart scenario, to produce a report code, leaving any program
```

```
;   intact.

        LD      HL,$5CB6        ; The address of the channels - initially
                                ; following system variables.
        LD      ($5C4F),HL      ; Set the CHANS system variable.

        LD      DE,L15AF        ; Address: init-chan in ROM.
        LD      BC,$0015        ; There are 21 bytes of initial data in ROM.
        EX      DE,HL           ; swap the pointers.
        LDIR                    ; Copy the bytes to RAM.

        EX      DE,HL           ; Swap pointers. HL points to program area.
        DEC     HL              ; Decrement address.
        LD      ($5C57),HL      ; Set DATADD to location before program area.
        INC     HL              ; Increment again.

        LD      ($5C53),HL      ; Set PROG the location where BASIC starts.
        LD      ($5C4B),HL      ; Set VARS to same location with a
        LD      (HL),$80        ; variables end-marker.
        INC     HL              ; Advance address.
        LD      ($5C59),HL      ; Set E_LINE, where the edit line
                                ; will be created.
                                ; Note. it is not strictly necessary to
                                ; execute the next fifteen bytes of code
                                ; as this will be done by the call to SET-MIN.
                                ; --
        LD      (HL),$0D        ; initially just has a carriage return
        INC     HL              ; followed by
        LD      (HL),$80        ; an end-marker.
        INC     HL              ; address the next location.
        LD      ($5C61),HL      ; set WORKSP - empty workspace.
        LD      ($5C63),HL      ; set STKBOT - bottom of the empty stack.
        LD      ($5C65),HL      ; set STKEND to the end of the empty stack.
                                ; --
        LD      A,$38           ; the colour system is set to white paper,
                                ; black ink, no flash or bright.
        LD      ($5C8D),A       ; set ATTR_P permanent colour attributes.
        LD      ($5C8F),A       ; set ATTR_T temporary colour attributes.
        LD      ($5C48),A       ; set BORDCR the border colour/lower screen
                                ; attributes.

        LD      HL,$0523        ; The keyboard repeat and delay values are
        LD      ($5C09),HL      ; loaded to REPDEL and REPPER.

        DEC     (IY-$3A)        ; set KSTATE-0 to $FF - keyboard map available.
        DEC     (IY-$36)        ; set KSTATE-4 to $FF - keyboard map available.

        LD      HL,L15C6        ; set source to ROM Address: init-strm
        LD      DE,$5C10        ; set destination to system variable STRMS-FD
        LD      BC,$000E        ; copy the 14 bytes of initial 7 streams data
        LDIR                    ; from ROM to RAM.

        SET     1,(IY+$01)      ; update FLAGS  - signal printer in use.
        CALL    L0EDF           ; call routine CLEAR-PRB to initialize system
                                ; variables associated with printer.
                                ; The buffer is clear.

        LD      (IY+$31),$02    ; set DF_SZ the lower screen display size to
                                ; two lines
        CALL    L0D6B           ; call routine CLS to set up system
                                ; variables associated with screen and clear
                                ; the screen and set attributes.
        XOR     A               ; clear accumulator so that we can address
        LD      DE,L1539 - 1    ; the message table directly.
```

```
        CALL    L0C0A           ; routine PO-MSG puts
                                ; '(c) 1982 Sinclair Research Ltd'
                                ; at bottom of display.
        SET     5,(IY+$02)      ; update TV_FLAG  - signal lower screen will
                                ; require clearing.

        JR      L12A9           ; forward to MAIN-1

; -------------------
; Main execution loop
; -------------------
;
;

;; MAIN-EXEC
L12A2:  LD      (IY+$31),$02    ; set DF_SZ lower screen display file size to
                                ; two lines.
        CALL    L1795           ; routine AUTO-LIST

;; MAIN-1
L12A9:  CALL    L16B0           ; routine SET-MIN clears work areas.

;; MAIN-2
L12AC:  LD      A,$00           ; select channel 'K' the keyboard

        CALL    L1601           ; routine CHAN-OPEN opens it
        CALL    L0F2C           ; routine EDITOR is called.
                                ; Note the above routine is where the Spectrum
                                ; waits for user-interaction. Perhaps the
                                ; most common input at this stage
                                ; is LOAD "".

        CALL    L1B17           ; routine LINE-SCAN scans the input.

        BIT     7,(IY+$00)      ; test ERR_NR - will be $FF if syntax is OK.
        JR      NZ,L12CF        ; forward, if correct, to MAIN-3.

;

        BIT     4,(IY+$30)      ; test FLAGS2 - K channel in use ?
        JR      Z,L1303         ; forward to MAIN-4 if not.

;

        LD      HL,($5C59)      ; an editing error so address E_LINE.
        CALL    L11A7           ; routine REMOVE-FP removes the hidden
                                ; floating-point forms.
        LD      (IY+$00),$FF    ; system variable ERR_NR is reset to 'OK'.
        JR      L12AC           ; back to MAIN-2 to allow user to correct.

; ---

; the branch was here if syntax has passed test.

;; MAIN-3
L12CF:  LD      HL,($5C59)      ; fetch the edit line address from E_LINE.
        LD      ($5C5D),HL      ; system variable CH_ADD is set to first
                                ; character of edit line.
                                ; Note. the above two instructions are a little
                                ; inadequate.
                                ; They are repeated with a subtle difference
                                ; at the start of the next subroutine and are
                                ; therefore not required above.
```

```
        CALL    L19FB           ; routine E-LINE-NO will fetch any line
                                ; number to BC if this is a program line.

        LD      A,B             ; test if the number of
        OR      C               ; the line is non-zero.
        JP      NZ,L155D        ; jump forward to MAIN-ADD if so to add the
                                ; line to the BASIC program.

; Has the user just pressed the ENTER key ?

        RST     18H             ; GET-CHAR gets character addressed by CH_ADD.
        CP      $0D             ; is it a carriage return ?
        JR      Z,L12A2         ; back to MAIN-EXEC if so for an automatic
                                ; listing.

; this must be a direct command.

        BIT     0,(IY+$30)      ; test FLAGS2 - clear the main screen ?

        CALL    NZ,L0DAF        ; routine CL-ALL, if so, e.g. after listing.

        CALL    L0D6E           ; routine CLS-LOWER anyway.

        LD      A,$19           ; compute scroll count as 25 minus
        SUB     (IY+$4F)        ; value of S_POSN_hi.
        LD      ($5C8C),A       ; update SCR_CT system variable.
        SET     7,(IY+$01)      ; update FLAGS - signal running program.
        LD      (IY+$00),$FF    ; set ERR_NR to 'OK'.
        LD      (IY+$0A),$01    ; set NSPPC to one for first statement.
        CALL    L1B8A           ; call routine LINE-RUN to run the line.
                                ; sysvar ERR_SP therefore addresses MAIN-4

; Examples of direct commands are RUN, CLS, LOAD "", PRINT USR 40000,
; LPRINT "A"; etc..
; If a user written machine-code program disables interrupts then it
; must enable them to pass the next step. We also jumped to here if the
; keyboard was not being used.

;; MAIN-4
L1303:  HALT                    ; wait for interrupt the only routine that can
                                ; set bit 5 of FLAGS.
        RES     5,(IY+$01)      ; update bit 5 of FLAGS - signal no new key.

        BIT     1,(IY+$30)      ; test FLAGS2 - is printer buffer clear ?
        CALL    NZ,L0ECD        ; call routine COPY-BUFF if not.
                                ; Note. the programmer has neglected
                                ; to set bit 1 of FLAGS first.

        LD      A,($5C3A)       ; fetch ERR_NR
        INC     A               ; increment to give true code.

; Now deal with a runtime error as opposed to an editing error.
; However if the error code is now zero then the OK message will be printed.

;; MAIN-G
L1313:  PUSH    AF              ; save the error number.

        LD      HL,$0000        ; prepare to clear some system variables.
        LD      (IY+$37),H      ; clear all the bits of FLAGX.
        LD      (IY+$26),H      ; blank X_PTR_hi to suppress error marker.
        LD      ($5C0B),HL      ; blank DEFADD to signal that no defined
                                ; function is currently being evaluated.

        LD      HL,$0001        ; explicit - inc hl would do.
```

```
        LD      ($5C16),HL      ; ensure STRMS-00 is keyboard.

        CALL    L16B0           ; routine SET-MIN clears workspace etc.
        RES     5,(IY+$37)      ; update FLAGX - signal in EDIT not INPUT mode.
                                ; Note. all the bits were reset earlier.

        CALL    L0D6E           ; call routine CLS-LOWER.

        SET     5,(IY+$02)      ; update TV_FLAG - signal lower screen
                                ; requires clearing.

        POP     AF              ; bring back the true error number
        LD      B,A             ; and make a copy in B.
        CP      $0A             ; is it a print-ready digit ?
        JR      C,L133C         ; forward to MAIN-5 if so.

        ADD     A,$07           ; add ASCII offset to letters.

;; MAIN-5
L133C:  CALL    L15EF           ; call routine OUT-CODE to print the code.

        LD      A,$20           ; followed by a space.
        RST     10H             ; PRINT-A

        LD      A,B             ; fetch stored report code.
        LD      DE,L1391        ; address: rpt-mesgs.

        CALL    L0C0A           ; call routine PO-MSG to print the message.

X1349:  XOR     A               ; clear accumulator to directly
        LD      DE,L1537 - 1    ; address the comma and space message.

        CALL    L0C0A           ; routine PO-MSG prints ', ' although it would
                                ; be more succinct to use RST $10.

        LD      BC,($5C45)      ; fetch PPC the current line number.
        CALL    L1A1B           ; routine OUT-NUM-1 will print that

        LD      A,$3A           ; then a ':' character.
        RST     10H             ; PRINT-A

        LD      C,(IY+$0D)      ; then SUBPPC for statement
        LD      B,$00           ; limited to 127
        CALL    L1A1B           ; routine OUT-NUM-1 prints BC.

        CALL    L1097           ; routine CLEAR-SP clears editing area which
                                ; probably contained 'RUN'.

        LD      A,($5C3A)       ; fetch ERR_NR again
        INC     A               ; test for no error originally $FF.
        JR      Z,L1386         ; forward to MAIN-9 if no error.

        CP      $09             ; is code Report 9 STOP ?
        JR      Z,L1373         ; forward to MAIN-6 if so

        CP      $15             ; is code Report L Break ?
        JR      NZ,L1376        ; forward to MAIN-7 if not

; Stop or Break was encountered so consider CONTINUE.

;; MAIN-6
L1373:  INC     (IY+$0D)        ; increment SUBPPC to next statement.

;; MAIN-7
```

```
L1376:  LD      BC,$0003        ; prepare to copy 3 system variables to
        LD      DE,$5C70        ; address OSPPC - statement for CONTINUE.
                                ; also updating OLDPPC line number below.

        LD      HL,$5C44        ; set source top to NSPPC next statement.
        BIT     7,(HL)          ; did BREAK occur before the jump ?
                                ; e.g. between GO TO and next statement.
        JR      Z,L1384         ; skip forward to MAIN-8, if not, as set-up
                                ; is correct.

        ADD     HL,BC           ; set source to SUBPPC number of current
                                ; statement/line which will be repeated.

;; MAIN-8
L1384:  LDDR                    ; copy PPC to OLDPPC and SUBPPC to OSPCC
                                ; or NSPPC to OLDPPC and NEWPPC to OSPCC

;; MAIN-9
L1386:  LD      (IY+$0A),$FF    ; update NSPPC - signal 'no jump'.
        RES     3,(IY+$01)      ; update FLAGS  - signal use 'K' mode for
                                ; the first character in the editor and

        JP      L12AC           ; jump back to MAIN-2.


; ----------------------
; Canned report messages
; ----------------------
; The Error reports with the last byte inverted. The first entry
; is a dummy entry. The last, which begins with $7F, the Spectrum
; character for copyright symbol, is placed here for convenience
; as is the preceding comma and space.
; The report line must accommodate a 4-digit line number and a 3-digit
; statement number which limits the length of the message text to twenty
; characters.
; e.g.  "B Integer out of range, 1000:127"

;; rpt-mesgs
L1391:  DEFB    $80
        DEFB    'O','K'+$80                         ; 0
        DEFM    "NEXT without FO"
        DEFB    'R'+$80                             ; 1
        DEFM    "Variable not foun"
        DEFB    'd'+$80                             ; 2
        DEFM    "Subscript wron"
        DEFB    'g'+$80                             ; 3
        DEFM    "Out of memor"
        DEFB    'y'+$80                             ; 4
        DEFM    "Out of scree"
        DEFB    'n'+$80                             ; 5
        DEFM    "Number too bi"
        DEFB    'g'+$80                             ; 6
        DEFM    "RETURN without GOSU"
        DEFB    'B'+$80                             ; 7
        DEFM    "End of fil"
        DEFB    'e'+$80                             ; 8
        DEFM    "STOP statemen"
        DEFB    't'+$80                             ; 9
        DEFM    "Invalid argumen"
        DEFB    't'+$80                             ; A
        DEFM    "Integer out of rang"
        DEFB    'e'+$80                             ; B
        DEFM    "Nonsense in BASI"
        DEFB    'C'+$80                             ; C
```

```
        DEFM    "BREAK - CONT repeat"
        DEFB    's'+$80                                 ; D
        DEFM    "Out of DAT"
        DEFB    'A'+$80                                 ; E
        DEFM    "Invalid file nam"
        DEFB    'e'+$80                                 ; F
        DEFM    "No room for lin"
        DEFB    'e'+$80                                 ; G
        DEFM    "STOP in INPU"
        DEFB    'T'+$80                                 ; H
        DEFM    "FOR without NEX"
        DEFB    'T'+$80                                 ; I
        DEFM    "Invalid I/O devic"
        DEFB    'e'+$80                                 ; J
        DEFM    "Invalid colou"
        DEFB    'r'+$80                                 ; K
        DEFM    "BREAK into progra"
        DEFB    'm'+$80                                 ; L
        DEFM    "RAMTOP no goo"
        DEFB    'd'+$80                                 ; M
        DEFM    "Statement los"
        DEFB    't'+$80                                 ; N
        DEFM    "Invalid strea"
        DEFB    'm'+$80                                 ; O
        DEFM    "FN without DE"
        DEFB    'F'+$80                                 ; P
        DEFM    "Parameter erro"
        DEFB    'r'+$80                                 ; Q
        DEFM    "Tape loading erro"
        DEFB    'r'+$80                                 ; R
;; comma-sp
L1537:  DEFB    ',',' '+$80                             ; used in report line.
;; copyright
L1539:  DEFB    $7F                                     ; copyright
        DEFM    " 1982 Sinclair Research Lt"
        DEFB    'd'+$80


; -------------
; REPORT-G
; -------------
; Note ERR_SP points here during line entry which allows the
; normal 'Out of Memory' report to be augmented to the more
; precise 'No Room for line' report.

;; REPORT-G
; No Room for line
L1555:  LD      A,$10           ; i.e. 'G' -$30 -$07
        LD      BC,$0000        ; this seems unnecessary.
        JP      L1313           ; jump back to MAIN-G

; ----------------------------
; Handle addition of BASIC line
; ----------------------------
; Note this is not a subroutine but a branch of the main execution loop.
; System variable ERR_SP still points to editing error handler.
; A new line is added to the BASIC program at the appropriate place.
; An existing line with same number is deleted first.
; Entering an existing line number deletes that line.
; Entering a non-existent line allows the subsequent line to be edited next.

;; MAIN-ADD
L155D:  LD      ($5C49),BC      ; set E_PPC to extracted line number.
        LD      HL,($5C5D)      ; fetch CH_ADD - points to location after the
```

```
                                ; initial digits (set in E_LINE_NO).
        EX      DE,HL           ; save start of BASIC in DE.

        LD      HL,L1555        ; Address: REPORT-G
        PUSH    HL              ; is pushed on stack and addressed by ERR_SP.
                                ; the only error that can occur is
                                ; 'Out of memory'.

        LD      HL,($5C61)      ; fetch WORKSP - end of line.
        SCF                     ; prepare for true subtraction.
        SBC     HL,DE           ; find length of BASIC and
        PUSH    HL              ; save it on stack.
        LD      H,B             ; transfer line number
        LD      L,C             ; to HL register.
        CALL    L196E           ; routine LINE-ADDR will see if
                                ; a line with the same number exists.
        JR      NZ,L157D        ; forward if no existing line to MAIN-ADD1.

        CALL    L19B8           ; routine NEXT-ONE finds the existing line.
        CALL    L19E8           ; routine RECLAIM-2 reclaims it.

;; MAIN-ADD1
L157D:  POP     BC              ; retrieve the length of the new line.
        LD      A,C             ; and test if carriage return only
        DEC     A               ; i.e. one byte long.
        OR      B               ; result would be zero.
        JR      Z,L15AB         ; forward to MAIN-ADD2 is so.

        PUSH    BC              ; save the length again.
        INC     BC              ; adjust for inclusion
        INC     BC              ; of line number (two bytes)
        INC     BC              ; and line length
        INC     BC              ; (two bytes).
        DEC     HL              ; HL points to location before the destination

        LD      DE,($5C53)      ; fetch the address of PROG
        PUSH    DE              ; and save it on the stack
        CALL    L1655           ; routine MAKE-ROOM creates BC spaces in
                                ; program area and updates pointers.
        POP     HL              ; restore old program pointer.
        LD      ($5C53),HL      ; and put back in PROG as it may have been
                                ; altered by the POINTERS routine.

        POP     BC              ; retrieve BASIC length
        PUSH    BC              ; and save again.

        INC     DE              ; points to end of new area.
        LD      HL,($5C61)      ; set HL to WORKSP - location after edit line.
        DEC     HL              ; decrement to address end marker.
        DEC     HL              ; decrement to address carriage return.
        LDDR                    ; copy the BASIC line back to initial command.

        LD      HL,($5C49)      ; fetch E_PPC - line number.
        EX      DE,HL           ; swap it to DE, HL points to last of
                                ; four locations.
        POP     BC              ; retrieve length of line.
        LD      (HL),B          ; high byte last.
        DEC     HL              ;
        LD      (HL),C          ; then low byte of length.
        DEC     HL              ;
        LD      (HL),E          ; then low byte of line number.
        DEC     HL              ;
        LD      (HL),D          ; then high byte range $0 - $27 (1-9999).
```

```
;; MAIN-ADD2
L15AB:  POP     AF                  ; drop the address of Report G
        JP      L12A2               ; and back to MAIN-EXEC producing a listing
                                    ; and to reset ERR_SP in EDITOR.


; ---------------------------------
; THE 'INITIAL CHANNEL' INFORMATION
; ---------------------------------
;   This initial channel information is copied from ROM to RAM, during
;   initialization.  It's new location is after the system variables and is
;   addressed by the system variable CHANS which means that it can slide up and
;   down in memory.  The table is never searched, by this ROM, and the last
;   character, which could be anything other than a comma, provides a
;   convenient resting place for DATADD.

;; init-chan
L15AF:  DEFW    L09F4               ; PRINT-OUT
        DEFW    L10A8               ; KEY-INPUT
        DEFB    $4B                 ; 'K'
        DEFW    L09F4               ; PRINT-OUT
        DEFW    L15C4               ; REPORT-J
        DEFB    $53                 ; 'S'
        DEFW    L0F81               ; ADD-CHAR
        DEFW    L15C4               ; REPORT-J
        DEFB    $52                 ; 'R'
        DEFW    L09F4               ; PRINT-OUT
        DEFW    L15C4               ; REPORT-J
        DEFB    $50                 ; 'P'

        DEFB    $80                 ; End Marker

;; REPORT-J
L15C4:  RST     08H                 ; ERROR-1
        DEFB    $12                 ; Error Report: Invalid I/O device


; -------------------------
; THE 'INITIAL STREAM' DATA
; -------------------------
;   This is the initial stream data for the seven streams $FD - $03 that is
;   copied from ROM to the STRMS system variables area during initialization.
;   There are reserved locations there for another 12 streams.  Each location
;   contains an offset to the second byte of a channel.  The first byte of a
;   channel can't be used as that would result in an offset of zero for some
;   and zero is used to denote that a stream is closed.

;; init-strm
L15C6:  DEFB    $01, $00            ; stream $FD offset to channel 'K'
        DEFB    $06, $00            ; stream $FE offset to channel 'S'
        DEFB    $0B, $00            ; stream $FF offset to channel 'R'

        DEFB    $01, $00            ; stream $00 offset to channel 'K'
        DEFB    $01, $00            ; stream $01 offset to channel 'K'
        DEFB    $06, $00            ; stream $02 offset to channel 'S'
        DEFB    $10, $00            ; stream $03 offset to channel 'P'

; ------------------------------
; THE 'INPUT CONTROL' SUBROUTINE
; ------------------------------
;

;; WAIT-KEY
L15D4:  BIT     5,(IY+$02)          ; test TV_FLAG - clear lower screen ?
```

```
        JR      NZ,L15DE        ; forward to WAIT-KEY1 if so.

        SET     3,(IY+$02)      ; update TV_FLAG - signal reprint the edit
                                ; line to the lower screen.

;; WAIT-KEY1
L15DE:  CALL    L15E6           ; routine INPUT-AD is called.

        RET     C               ; return with acceptable keys.

        JR      Z,L15DE         ; back to WAIT-KEY1 if no key is pressed
                                ; or it has been handled within INPUT-AD.

;   Note. When inputting from the keyboard all characters are returned with
;   above conditions so this path is never taken.

;; REPORT-8
L15E4:  RST     08H             ; ERROR-1
        DEFB    $07             ; Error Report: End of file

; ---------------------------
; THE 'INPUT ADDRESS' ROUTINE
; ---------------------------
;   This routine fetches the address of the input stream from the current
;   channel area using the system variable CURCHL.

;; INPUT-AD
L15E6:  EXX                     ; switch in alternate set.
        PUSH    HL              ; save HL register
        LD      HL,($5C51)      ; fetch address of CURCHL - current channel.
        INC     HL              ; step over output routine
        INC     HL              ; to point to low byte of input routine.
        JR      L15F7           ; forward to CALL-SUB.

; -------------------------
; THE 'CODE OUTPUT' ROUTINE
; -------------------------
;   This routine is called on five occasions to print the ASCII equivalent of
;   a value 0-9.

;; OUT-CODE
L15EF:  LD      E,$30           ; add 48 decimal to give the ASCII character
        ADD     A,E             ; '0' to '9' and continue into the main output
                                ; routine.

; -------------------------
; THE 'MAIN OUTPUT' ROUTINE
; -------------------------
;   PRINT-A-2 is a continuation of the RST 10 restart that prints any character.
;   The routine prints to the current channel and the printing of control codes
;   may alter that channel to divert subsequent RST 10 instructions to temporary
;   routines. The normal channel is $09F4.

;; PRINT-A-2
L15F2:  EXX                     ; switch in alternate set
        PUSH    HL              ; save HL register
        LD      HL,($5C51)      ; fetch CURCHL the current channel.

; input-ad rejoins here also.

;; CALL-SUB
L15F7:  LD      E,(HL)          ; put the low byte in E.
        INC     HL              ; advance address.
        LD      D,(HL)          ; put the high byte to D.
```

```
        EX      DE,HL           ; transfer the stream to HL.
        CALL    L162C           ; use routine CALL-JUMP.
                                ; in effect CALL (HL).

        POP     HL              ; restore saved HL register.
        EXX                     ; switch back to the main set and
        RET                     ; return.

; --------------------------
; THE 'OPEN CHANNEL' ROUTINE
; --------------------------
;   This subroutine is used by the ROM to open a channel 'K', 'S', 'R' or 'P'.
;   This is either for its own use or in response to a user's request, for
;   example, when '#' is encountered with output - PRINT, LIST etc.
;   or with input - INPUT, INKEY$ etc.
;   It is entered with a system stream $FD - $FF, or a user stream $00 - $0F
;   in the accumulator.

;; CHAN-OPEN
L1601:  ADD     A,A             ; double the stream ($FF will become $FE etc.)
        ADD     A,$16           ; add the offset to stream 0 from $5C00
        LD      L,A             ; result to L
        LD      H,$5C           ; now form the address in STRMS area.
        LD      E,(HL)          ; fetch low byte of CHANS offset
        INC     HL              ; address next
        LD      D,(HL)          ; fetch high byte of offset
        LD      A,D             ; test that the stream is open.
        OR      E               ; zero if closed.
        JR      NZ,L1610        ; forward to CHAN-OP-1 if open.

;; REPORT-Oa
L160E:  RST     08H             ; ERROR-1
        DEFB    $17             ; Error Report: Invalid stream

; continue here if stream was open. Note that the offset is from CHANS
; to the second byte of the channel.

;; CHAN-OP-1
L1610:  DEC     DE              ; reduce offset so it points to the channel.
        LD      HL,($5C4F)      ; fetch CHANS the location of the base of
                                ; the channel information area
        ADD     HL,DE           ; and add the offset to address the channel.
                                ; and continue to set flags.

; -----------------
; Set channel flags
; -----------------
; This subroutine is used from ED-EDIT, str$ and read-in to reset the
; current channel when it has been temporarily altered.

;; CHAN-FLAG
L1615:  LD      ($5C51),HL      ; set CURCHL system variable to the
                                ; address in HL
        RES     4,(IY+$30)      ; update FLAGS2  - signal K channel not in use.
                                ; Note. provide a default for channel 'R'.
        INC     HL              ; advance past
        INC     HL              ; output routine.
        INC     HL              ; advance past
        INC     HL              ; input routine.
        LD      C,(HL)          ; pick up the letter.
        LD      HL,L162D        ; address: chn-cd-lu
        CALL    L16DC           ; routine INDEXER finds offset to a
                                ; flag-setting routine.
```

```
        RET     NC              ; but if the letter wasn't found in the
                                ; table just return now. - channel 'R'.

        LD      D,$00           ; prepare to add
        LD      E,(HL)          ; offset to E
        ADD     HL,DE           ; add offset to location of offset to form
                                ; address of routine

;; CALL-JUMP
L162C:  JP      (HL)            ; jump to the routine

; Footnote. calling any location that holds JP (HL) is the equivalent to
; a pseudo Z80 instruction CALL (HL). The ROM uses the instruction above.

; -------------------------
; Channel code look-up table
; -------------------------
; This table is used by the routine above to find one of the three
; flag setting routines below it.
; A zero end-marker is required as channel 'R' is not present.

;; chn-cd-lu
L162D:  DEFB    'K', L1634-$-1  ; offset $06 to CHAN-K
        DEFB    'S', L1642-$-1  ; offset $12 to CHAN-S
        DEFB    'P', L164D-$-1  ; offset $1B to CHAN-P

        DEFB    $00             ; end marker.

; --------------
; Channel K flag
; --------------
; routine to set flags for lower screen/keyboard channel.

;; CHAN-K
L1634:  SET     0,(IY+$02)      ; update TV_FLAG  - signal lower screen in use
        RES     5,(IY+$01)      ; update FLAGS    - signal no new key
        SET     4,(IY+$30)      ; update FLAGS2   - signal K channel in use
        JR      L1646           ; forward to CHAN-S-1 for indirect exit

; --------------
; Channel S flag
; --------------
; routine to set flags for upper screen channel.

;; CHAN-S
L1642:  RES     0,(IY+$02)      ; TV_FLAG  - signal main screen in use

;; CHAN-S-1
L1646:  RES     1,(IY+$01)      ; update FLAGS  - signal printer not in use
        JP      L0D4D           ; jump back to TEMPS and exit via that
                                ; routine after setting temporary attributes.
; --------------
; Channel P flag
; --------------
; This routine sets a flag so that subsequent print related commands
; print to printer or update the relevant system variables.
; This status remains in force until reset by the routine above.

;; CHAN-P
L164D:  SET     1,(IY+$01)      ; update FLAGS  - signal printer in use
        RET                     ; return

; -------------------------
; THE 'ONE SPACE' SUBROUTINE
```

```
; -------------------------
; This routine is called once only to create a single space
; in workspace by ADD-CHAR.

;; ONE-SPACE
L1652:  LD      BC,$0001        ; create space for a single character.

; ---------
; Make Room
; ---------
; This entry point is used to create BC spaces in various areas such as
; program area, variables area, workspace etc..
; The entire free RAM is available to each BASIC statement.
; On entry, HL addresses where the first location is to be created.
; Afterwards, HL will point to the location before this.

;; MAKE-ROOM
L1655:  PUSH    HL              ; save the address pointer.
        CALL    L1F05           ; routine TEST-ROOM checks if room
                                ; exists and generates an error if not.
        POP     HL              ; restore the address pointer.
        CALL    L1664           ; routine POINTERS updates the
                                ; dynamic memory location pointers.
                                ; DE now holds the old value of STKEND.
        LD      HL,($5C65)      ; fetch new STKEND the top destination.

        EX      DE,HL           ; HL now addresses the top of the area to
                                ; be moved up - old STKEND.
        LDDR                    ; the program, variables, etc are moved up.
        RET                     ; return with new area ready to be populated.
                                ; HL points to location before new area,
                                ; and DE to last of new locations.

; -----------------------------------------------
; Adjust pointers before making or reclaiming room
; -----------------------------------------------
; This routine is called by MAKE-ROOM to adjust upwards and by RECLAIM to
; adjust downwards the pointers within dynamic memory.
; The fourteen pointers to dynamic memory, starting with VARS and ending
; with STKEND, are updated adding BC if they are higher than the position
; in HL.
; The system variables are in no particular order except that STKEND, the first
; free location after dynamic memory must be the last encountered.

;; POINTERS
L1664:  PUSH    AF              ; preserve accumulator.
        PUSH    HL              ; put pos pointer on stack.
        LD      HL,$5C4B        ; address VARS the first of the
        LD      A,$0E           ; fourteen variables to consider.

;; PTR-NEXT
L166B:  LD      E,(HL)          ; fetch the low byte of the system variable.
        INC     HL              ; advance address.
        LD      D,(HL)          ; fetch high byte of the system variable.
        EX      (SP),HL         ; swap pointer on stack with the variable
                                ; pointer.
        AND     A               ; prepare to subtract.
        SBC     HL,DE           ; subtract variable address
        ADD     HL,DE           ; and add back
        EX      (SP),HL         ; swap pos with system variable pointer
        JR      NC,L167F        ; forward to PTR-DONE if var before pos

        PUSH    DE              ; save system variable address.
        EX      DE,HL           ; transfer to HL
```

```
        ADD     HL,BC           ; add the offset
        EX      DE,HL           ; back to DE
        LD      (HL),D          ; load high byte
        DEC     HL              ; move back
        LD      (HL),E          ; load low byte
        INC     HL              ; advance to high byte
        POP     DE              ; restore old system variable address.

;; PTR-DONE
L167F:  INC     HL              ; address next system variable.
        DEC     A               ; decrease counter.
        JR      NZ,L166B        ; back to PTR-NEXT if more.
        EX      DE,HL           ; transfer old value of STKEND to HL.
                                ; Note. this has always been updated.
        POP     DE              ; pop the address of the position.

        POP     AF              ; pop preserved accumulator.
        AND     A               ; clear carry flag preparing to subtract.

        SBC     HL,DE           ; subtract position from old stkend
        LD      B,H             ; to give number of data bytes
        LD      C,L             ; to be moved.
        INC     BC              ; increment as we also copy byte at old STKEND.
        ADD     HL,DE           ; recompute old stkend.
        EX      DE,HL           ; transfer to DE.
        RET                     ; return.



; -------------------
; Collect line number
; -------------------
; This routine extracts a line number, at an address that has previously
; been found using LINE-ADDR, and it is entered at LINE-NO. If it encounters
; the program 'end-marker' then the previous line is used and if that
; should also be unacceptable then zero is used as it must be a direct
; command. The program end-marker is the variables end-marker $80, or
; if variables exist, then the first character of any variable name.

;; LINE-ZERO
L168F:  DEFB    $00, $00        ; dummy line number used for direct commands


;; LINE-NO-A
L1691:  EX      DE,HL           ; fetch the previous line to HL and set
        LD      DE,L168F        ; DE to LINE-ZERO should HL also fail.

; -> The Entry Point.

;; LINE-NO
L1695:  LD      A,(HL)          ; fetch the high byte - max $2F
        AND     $C0             ; mask off the invalid bits.
        JR      NZ,L1691        ; to LINE-NO-A if an end-marker.

        LD      D,(HL)          ; reload the high byte.
        INC     HL              ; advance address.
        LD      E,(HL)          ; pick up the low byte.
        RET                     ; return from here.

; -------------------
; Handle reserve room
; -------------------
; This is a continuation of the restart BC-SPACES
```

```
;; RESERVE
L169E:  LD      HL,($5C63)      ; STKBOT first location of calculator stack
        DEC     HL              ; make one less than new location
        CALL    L1655           ; routine MAKE-ROOM creates the room.
        INC     HL              ; address the first new location
        INC     HL              ; advance to second
        POP     BC              ; restore old WORKSP
        LD      ($5C61),BC      ; system variable WORKSP was perhaps
                                ; changed by POINTERS routine.
        POP     BC              ; restore count for return value.
        EX      DE,HL           ; switch. DE = location after first new space
        INC     HL              ; HL now location after new space
        RET                     ; return.

; --------------------------
; Clear various editing areas
; --------------------------
; This routine sets the editing area, workspace and calculator stack
; to their minimum configurations as at initialization and indeed this
; routine could have been relied on to perform that task.
; This routine uses HL only and returns with that register holding
; WORKSP/STKBOT/STKEND though no use is made of this. The routines also
; reset MEM to its usual place in the systems variable area should it
; have been relocated to a FOR-NEXT variable. The main entry point
; SET-MIN is called at the start of the MAIN-EXEC loop and prior to
; displaying an error.

;; SET-MIN
L16B0:  LD      HL,($5C59)      ; fetch E_LINE
        LD      (HL),$0D        ; insert carriage return
        LD      ($5C5B),HL      ; make K_CUR keyboard cursor point there.
        INC     HL              ; next location
        LD      (HL),$80        ; holds end-marker $80
        INC     HL              ; next location becomes
        LD      ($5C61),HL      ; start of WORKSP

; This entry point is used prior to input and prior to the execution,
; or parsing, of each statement.

;; SET-WORK
L16BF:  LD      HL,($5C61)      ; fetch WORKSP value
        LD      ($5C63),HL      ; and place in STKBOT

; This entry point is used to move the stack back to its normal place
; after temporary relocation during line entry and also from ERROR-3

;; SET-STK
L16C5:  LD      HL,($5C63)      ; fetch STKBOT value
        LD      ($5C65),HL      ; and place in STKEND.

        PUSH    HL              ; perhaps an obsolete entry point.
        LD      HL,$5C92        ; normal location of MEM-0
        LD      ($5C68),HL      ; is restored to system variable MEM.
        POP     HL              ; saved value not required.
        RET                     ; return.

; ------------------
; Reclaim edit-line?
; ------------------
; This seems to be legacy code from the ZX80/ZX81 as it is
; not used in this ROM.
; That task, in fact, is performed here by the dual-area routine CLEAR-SP.
; This routine is designed to deal with something that is known to be in the
; edit buffer and not workspace.
```

```
; On entry, HL must point to the end of the something to be deleted.

;; REC-EDIT
L16D4:  LD      DE,($5C59)      ; fetch start of edit line from E_LINE.
        JP      L19E5           ; jump forward to RECLAIM-1.

; --------------------------
; The Table INDEXING routine
; --------------------------
; This routine is used to search two-byte hash tables for a character
; held in C, returning the address of the following offset byte.
; if it is known that the character is in the table e.g. for priorities,
; then the table requires no zero end-marker. If this is not known at the
; outset then a zero end-marker is required and carry is set to signal
; success.

;; INDEXER-1
L16DB:  INC     HL              ; address the next pair of values.

; -> The Entry Point.

;; INDEXER
L16DC:  LD      A,(HL)          ; fetch the first byte of pair
        AND     A               ; is it the end-marker ?
        RET     Z               ; return with carry reset if so.

        CP      C               ; is it the required character ?
        INC     HL              ; address next location.
        JR      NZ,L16DB        ; back to INDEXER-1 if no match.

        SCF                     ; else set the carry flag.
        RET                     ; return with carry set

; --------------------------------
; The Channel and Streams Routines
; --------------------------------
; A channel is an input/output route to a hardware device
; and is identified to the system by a single letter e.g. 'K' for
; the keyboard. A channel can have an input and output route
; associated with it in which case it is bi-directional like
; the keyboard. Others like the upper screen 'S' are output
; only and the input routine usually points to a report message.
; Channels 'K' and 'S' are system channels and it would be inappropriate
; to close the associated streams so a mechanism is provided to
; re-attach them. When the re-attachment is no longer required, then
; closing these streams resets them as at initialization.
; Early adverts said that the network and RS232 were in this ROM.
; Channels 'N' and 'B' are user channels and have been removed successfully
; if, as seems possible, they existed.
; Ironically the tape streamer is not accessed through streams and
; channels.
; Early demonstrations of the Spectrum showed a single microdrive being
; controlled by the main ROM.

; ---------------------
; THE 'CLOSE #' COMMAND
; ---------------------
;   This command allows streams to be closed after use.
;   Any temporary memory areas used by the stream would be reclaimed and
;   finally flags set or reset if necessary.

;; CLOSE
L16E5:  CALL    L171E           ; routine STR-DATA fetches parameter
                                ; from calculator stack and gets the
```

```
                                ; existing STRMS data pointer address in HL
                                ; and stream offset from CHANS in BC.

                                ; Note. this offset could be zero if the
                                ; stream is already closed. A check for this
                                ; should occur now and an error should be
                                ; generated, for example,
                                ; Report S 'Stream status closed'.

        CALL    L1701           ; routine CLOSE-2 would perform any actions
                                ; peculiar to that stream without disturbing
                                ; data pointer to STRMS entry in HL.

        LD      BC,$0000        ; the stream is to be blanked.
        LD      DE,$A3E2        ; the number of bytes from stream 4, $5C1E,
                                ; to $10000
        EX      DE,HL           ; transfer offset to HL, STRMS data pointer
                                ; to DE.
        ADD     HL,DE           ; add the offset to the data pointer.
        JR      C,L16FC         ; forward to CLOSE-1 if a non-system stream.
                                ; i.e. higher than 3.

; proceed with a negative result.

        LD      BC,L15C6 + 14   ; prepare the address of the byte after
                                ; the initial stream data in ROM. ($15D4)
        ADD     HL,BC           ; index into the data table with negative value.
        LD      C,(HL)          ; low byte to C
        INC     HL              ; address next.
        LD      B,(HL)          ; high byte to B.

;   and for streams 0 - 3 just enter the initial data back into the STRMS entry
;   streams 0 - 2 can't be closed as they are shared by the operating system.
;   -> for streams 4 - 15 then blank the entry.

;; CLOSE-1
L16FC:  EX      DE,HL           ; address of stream to HL.
        LD      (HL),C          ; place zero (or low byte).
        INC     HL              ; next address.
        LD      (HL),B          ; place zero (or high byte).
        RET                     ; return.

; ------------------------
; THE 'CLOSE-2' SUBROUTINE
; ------------------------
;   There is not much point in coming here.
;   The purpose was once to find the offset to a special closing routine,
;   in this ROM and within 256 bytes of the close stream look up table that
;   would reclaim any buffers associated with a stream. At least one has been
;   removed.
;   Any attempt to CLOSE streams $00 to $04, without first opening the stream,
;   will lead to either a system restart or the production of a strange report.
;   credit: Martin Wren-Hilton 1982.

;; CLOSE-2
L1701:  PUSH    HL              ; * save address of stream data pointer
                                ; in STRMS on the machine stack.
        LD      HL,($5C4F)      ; fetch CHANS address to HL
        ADD     HL,BC           ; add the offset to address the second
                                ; byte of the output routine hopefully.
        INC     HL              ; step past
        INC     HL              ; the input routine.

;    Note. When the Sinclair Interface1 is fitted then an instruction fetch
```

```
;    on the next address pages this ROM out and the shadow ROM in.

;; ROM_TRAP
L1708:  INC     HL                ; to address channel's letter
        LD      C,(HL)            ; pick it up in C.
                                  ; Note. but if stream is already closed we
                                  ; get the value $10 (the byte preceding 'K').

        EX      DE,HL             ; save the pointer to the letter in DE.

;   Note. The string pointer is saved but not used!!

        LD      HL,L1716          ; address: cl-str-lu in ROM.
        CALL    L16DC             ; routine INDEXER uses the code to get
                                  ; the 8-bit offset from the current point to
                                  ; the address of the closing routine in ROM.
                                  ; Note. it won't find $10 there!

        LD      C,(HL)            ; transfer the offset to C.
        LD      B,$00             ; prepare to add.
        ADD     HL,BC             ; add offset to point to the address of the
                                  ; routine that closes the stream.
                                  ; (and presumably removes any buffers that
                                  ; are associated with it.)
        JP      (HL)              ; jump to that routine.

; --------------------------------
; THE 'CLOSE STREAM LOOK-UP' TABLE
; --------------------------------
;   This table contains an entry for a letter found in the CHANS area.
;   followed by an 8-bit displacement, from that byte's address in the
;   table to the routine that performs any ancillary actions associated
;   with closing the stream of that channel.
;   The table doesn't require a zero end-marker as the letter has been
;   picked up from a channel that has an open stream.

;; cl-str-lu
L1716:  DEFB    'K', L171C-$-1  ; offset 5 to CLOSE-STR
        DEFB    'S', L171C-$-1  ; offset 3 to CLOSE-STR
        DEFB    'P', L171C-$-1  ; offset 1 to CLOSE-STR


; ------------------------------
; THE 'CLOSE STREAM' SUBROUTINES
; ------------------------------
; The close stream routines in fact have no ancillary actions to perform
; which is not surprising with regard to 'K' and 'S'.

;; CLOSE-STR
L171C:  POP     HL                ; * now just restore the stream data pointer
        RET                       ; in STRMS and return.

; -----------
; Stream data
; -----------
; This routine finds the data entry in the STRMS area for the specified
; stream which is passed on the calculator stack. It returns with HL
; pointing to this system variable and BC holding a displacement from
; the CHANS area to the second byte of the stream's channel. If BC holds
; zero, then that signifies that the stream is closed.

;; STR-DATA
L171E:  CALL    L1E94             ; routine FIND-INT1 fetches parameter to A
        CP      $10               ; is it less than 16d ?
```

```
        JR      C,L1727         ; skip forward to STR-DATA1 if so.

;; REPORT-Ob
L1725:  RST     08H             ; ERROR-1
        DEFB    $17             ; Error Report: Invalid stream

;; STR-DATA1
L1727:  ADD     A,$03           ; add the offset for 3 system streams.
                                ; range 00 - 15d becomes 3 - 18d.
        RLCA                    ; double as there are two bytes per
                                ; stream - now 06 - 36d
        LD      HL,$5C10        ; address STRMS - the start of the streams
                                ; data area in system variables.
        LD      C,A             ; transfer the low byte to A.
        LD      B,$00           ; prepare to add offset.
        ADD     HL,BC           ; add to address the data entry in STRMS.

; the data entry itself contains an offset from CHANS to the address of the
; stream

        LD      C,(HL)          ; low byte of displacement to C.
        INC     HL              ; address next.
        LD      B,(HL)          ; high byte of displacement to B.
        DEC     HL              ; step back to leave HL pointing to STRMS
                                ; data entry.
        RET                     ; return with CHANS displacement in BC
                                ; and address of stream data entry in HL.

; --------------------
; Handle OPEN# command
; --------------------
; Command syntax example: OPEN #5,"s"
; On entry the channel code entry is on the calculator stack with the next
; value containing the stream identifier. They have to swapped.

;; OPEN
L1736:  RST     28H             ;; FP-CALC    ;s,c.
        DEFB    $01             ;;exchange    ;c,s.
        DEFB    $38             ;;end-calc

        CALL    L171E           ; routine STR-DATA fetches the stream off
                                ; the stack and returns with the CHANS
                                ; displacement in BC and HL addressing
                                ; the STRMS data entry.
        LD      A,B             ; test for zero which
        OR      C               ; indicates the stream is closed.
        JR      Z,L1756         ; skip forward to OPEN-1 if so.

; if it is a system channel then it can re-attached.

        EX      DE,HL           ; save STRMS address in DE.
        LD      HL,($5C4F)      ; fetch CHANS.
        ADD     HL,BC           ; add the offset to address the second
                                ; byte of the channel.
        INC     HL              ; skip over the
        INC     HL              ; input routine.
        INC     HL              ; and address the letter.
        LD      A,(HL)          ; pick up the letter.
        EX      DE,HL           ; save letter pointer and bring back
                                ; the STRMS pointer.

        CP      $4B             ; is it 'K' ?
        JR      Z,L1756         ; forward to OPEN-1 if so
```

```
        CP      $53             ; is it 'S' ?
        JR      Z,L1756         ; forward to OPEN-1 if so

        CP      $50             ; is it 'P' ?
        JR      NZ,L1725        ; back to REPORT-Ob if not.
                                ; to report 'Invalid stream'.

; continue if one of the upper-case letters was found.
; and rejoin here from above if stream was closed.

;; OPEN-1
L1756:  CALL    L175D           ; routine OPEN-2 opens the stream.

; it now remains to update the STRMS variable.

        LD      (HL),E          ; insert or overwrite the low byte.
        INC     HL              ; address high byte in STRMS.
        LD      (HL),D          ; insert or overwrite the high byte.
        RET                     ; return.

; -----------------
; OPEN-2 Subroutine
; -----------------
; There is some point in coming here as, as well as once creating buffers,
; this routine also sets flags.

;; OPEN-2
L175D:  PUSH    HL              ; * save the STRMS data entry pointer.
        CALL    L2BF1           ; routine STK-FETCH now fetches the
                                ; parameters of the channel string.
                                ; start in DE, length in BC.

        LD      A,B             ; test that it is not
        OR      C               ; the null string.
        JR      NZ,L1767        ; skip forward to OPEN-3 with 1 character
                                ; or more!

;; REPORT-Fb
L1765:  RST     08H             ; ERROR-1
        DEFB    $0E             ; Error Report: Invalid file name

;; OPEN-3
L1767:  PUSH    BC              ; save the length of the string.
        LD      A,(DE)          ; pick up the first character.
                                ; Note. if the second character is used to
                                ; distinguish between a binary or text
                                ; channel then it will be simply a matter
                                ; of setting bit 7 of FLAGX.
        AND     $DF             ; make it upper-case.
        LD      C,A             ; place it in C.
        LD      HL,L177A        ; address: op-str-lu is loaded.
        CALL    L16DC           ; routine INDEXER will search for letter.
        JR      NC,L1765        ; back to REPORT-F if not found
                                ; 'Invalid filename'

        LD      C,(HL)          ; fetch the displacement to opening routine.
        LD      B,$00           ; prepare to add.
        ADD     HL,BC           ; now form address of opening routine.
        POP     BC              ; restore the length of string.
        JP      (HL)            ; now jump forward to the relevant routine.

; -------------------------
; OPEN stream look-up table
; -------------------------
```

```
; The open stream look-up table consists of matched pairs.
; The channel letter is followed by an 8-bit displacement to the
; associated stream-opening routine in this ROM.
; The table requires a zero end-marker as the letter has been
; provided by the user and not the operating system.

;; op-str-lu
L177A:  DEFB    'K', L1781-$-1  ; $06 offset to OPEN-K
        DEFB    'S', L1785-$-1  ; $08 offset to OPEN-S
        DEFB    'P', L1789-$-1  ; $0A offset to OPEN-P

        DEFB    $00             ; end-marker.

; ----------------------------
; The Stream Opening Routines.
; ----------------------------
; These routines would have opened any buffers associated with the stream
; before jumping forward to OPEN-END with the displacement value in E
; and perhaps a modified value in BC. The strange pathing does seem to
; provide for flexibility in this respect.
;
; There is no need to open the printer buffer as it is there already
; even if you are still saving up for a ZX Printer or have moved onto
; something bigger. In any case it would have to be created after
; the system variables but apart from that it is a simple task
; and all but one of the ROM routines can handle a buffer in that position.
; (PR-ALL-6 would require an extra 3 bytes of code).
; However it wouldn't be wise to have two streams attached to the ZX Printer
; as you can now, so one assumes that if PR_CC_hi was non-zero then
; the OPEN-P routine would have refused to attach a stream if another
; stream was attached.

; Something of significance is being passed to these ghost routines in the
; second character. Strings 'RB', 'RT' perhaps or a drive/station number.
; The routine would have to deal with that and exit to OPEN_END with BC
; containing $0001 or more likely there would be an exit within the routine.
; Anyway doesn't matter, these routines are long gone.

; -----------------
; OPEN-K Subroutine
; -----------------
; Open Keyboard stream.

;; OPEN-K
L1781:  LD      E,$01           ; 01 is offset to second byte of channel 'K'.
        JR      L178B           ; forward to OPEN-END

; -----------------
; OPEN-S Subroutine
; -----------------
; Open Screen stream.

;; OPEN-S
L1785:  LD      E,$06           ; 06 is offset to 2nd byte of channel 'S'
        JR      L178B           ; to OPEN-END

; -----------------
; OPEN-P Subroutine
; -----------------
; Open Printer stream.

;; OPEN-P
L1789:  LD      E,$10           ; 16d is offset to 2nd byte of channel 'P'
```

```
;; OPEN-END
L178B:  DEC     BC                  ; the stored length of 'K','S','P' or
                                    ; whatever is now tested. ??
        LD      A,B                 ; test now if initial or residual length
        OR      C                   ; is one character.
        JR      NZ,L1765            ; to REPORT-Fb 'Invalid file name' if not.

        LD      D,A                 ; load D with zero to form the displacement
                                    ; in the DE register.
        POP     HL                  ; * restore the saved STRMS pointer.
        RET                         ; return to update STRMS entry thereby
                                    ; signaling stream is open.

; ----------------------------------------
; Handle CAT, ERASE, FORMAT, MOVE commands
; ----------------------------------------
; These just generate an error report as the ROM is 'incomplete'.
;
; Luckily this provides a mechanism for extending these in a shadow ROM
; but without the powerful mechanisms set up in this ROM.
; An instruction fetch on $0008 may page in a peripheral ROM,
; e.g. the Sinclair Interface 1 ROM, to handle these commands.
; However that wasn't the plan.
; Development of this ROM continued for another three months until the cost
; of replacing it and the manual became unfeasible.
; The ultimate power of channels and streams died at birth.

;; CAT-ETC
L1793:  JR      L1725               ; to REPORT-Ob

; -----------------
; Perform AUTO-LIST
; -----------------
; This produces an automatic listing in the upper screen.

;; AUTO-LIST
L1795:  LD      ($5C3F),SP          ; save stack pointer in LIST_SP
        LD      (IY+$02),$10        ; update TV_FLAG set bit 3
        CALL    L0DAF               ; routine CL-ALL.
        SET     0,(IY+$02)          ; update TV_FLAG  - signal lower screen in use

        LD      B,(IY+$31)          ; fetch DF_SZ to B.
        CALL    L0E44               ; routine CL-LINE clears lower display
                                    ; preserving B.
        RES     0,(IY+$02)          ; update TV_FLAG  - signal main screen in use
        SET     0,(IY+$30)          ; update FLAGS2 - signal will be necessary to
                                    ; clear main screen.
        LD      HL,($5C49)          ; fetch E_PPC current edit line to HL.
        LD      DE,($5C6C)          ; fetch S_TOP to DE, the current top line
                                    ; (initially zero)
        AND     A                   ; prepare for true subtraction.
        SBC     HL,DE               ; subtract and
        ADD     HL,DE               ; add back.
        JR      C,L17E1             ; to AUTO-L-2 if S_TOP higher than E_PPC
                                    ; to set S_TOP to E_PPC

        PUSH    DE                  ; save the top line number.
        CALL    L196E               ; routine LINE-ADDR gets address of E_PPC.
        LD      DE,$02C0            ; prepare known number of characters in
                                    ; the default upper screen.
        EX      DE,HL               ; offset to HL, program address to DE.
        SBC     HL,DE               ; subtract high value from low to obtain
                                    ; negated result used in addition.
        EX      (SP),HL             ; swap result with top line number on stack.
```

```
        CALL    L196E           ; routine LINE-ADDR  gets address of that
                                ; top line in HL and next line in DE.
        POP     BC              ; restore the result to balance stack.

;; AUTO-L-1
L17CE:  PUSH    BC              ; save the result.
        CALL    L19B8           ; routine NEXT-ONE gets address in HL of
                                ; line after auto-line (in DE).
        POP     BC              ; restore result.
        ADD     HL,BC           ; compute back.
        JR      C,L17E4         ; to AUTO-L-3 if line 'should' appear

        EX      DE,HL           ; address of next line to HL.
        LD      D,(HL)          ; get line
        INC     HL              ; number
        LD      E,(HL)          ; in DE.
        DEC     HL              ; adjust back to start.
        LD      ($5C6C),DE      ; update S_TOP.
        JR      L17CE           ; to AUTO-L-1 until estimate reached.

; ---

; the jump was to here if S_TOP was greater than E_PPC

;; AUTO-L-2
L17E1:  LD      ($5C6C),HL      ; make S_TOP the same as E_PPC.

; continue here with valid starting point from above or good estimate
; from computation

;; AUTO-L-3
L17E4:  LD      HL,($5C6C)      ; fetch S_TOP line number to HL.
        CALL    L196E           ; routine LINE-ADDR gets address in HL.
                                ; address of next in DE.
        JR      Z,L17ED         ; to AUTO-L-4 if line exists.

        EX      DE,HL           ; else use address of next line.

;; AUTO-L-4
L17ED:  CALL    L1833           ; routine LIST-ALL                >>>

; The return will be to here if no scrolling occurred

        RES     4,(IY+$02)      ; update TV_FLAG  - signal no auto listing.
        RET                     ; return.

; ------------
; Handle LLIST
; ------------
; A short form of LIST #3. The listing goes to stream 3 - default printer.

;; LLIST
L17F5:  LD      A,$03           ; the usual stream for ZX Printer
        JR      L17FB           ; forward to LIST-1

; -----------
; Handle LIST
; -----------
; List to any stream.
; Note. While a starting line can be specified it is
; not possible to specify an end line.
; Just listing a line makes it the current edit line.

;; LIST
```

```
L17F9:  LD      A,$02           ; default is stream 2 - the upper screen.

;; LIST-1
L17FB:  LD      (IY+$02),$00    ; the TV_FLAG is initialized with bit 0 reset
                                ; indicating upper screen in use.
        CALL    L2530           ; routine SYNTAX-Z - checking syntax ?
        CALL    NZ,L1601        ; routine CHAN-OPEN if in run-time.

        RST     18H             ; GET-CHAR
        CALL    L2070           ; routine STR-ALTER will alter if '#'.
        JR      C,L181F         ; forward to LIST-4 not a '#' .


        RST     18H             ; GET-CHAR
        CP      $3B             ; is it ';' ?
        JR      Z,L1814         ; skip to LIST-2 if so.

        CP      $2C             ; is it ',' ?
        JR      NZ,L181A        ; forward to LIST-3 if neither separator.

; we have, say,  LIST #15, and a number must follow the separator.

;; LIST-2
L1814:  RST     20H             ; NEXT-CHAR
        CALL    L1C82           ; routine EXPT-1NUM
        JR      L1822           ; forward to LIST-5

; ---

; the branch was here with just LIST #3 etc.

;; LIST-3
L181A:  CALL    L1CE6           ; routine USE-ZERO
        JR      L1822           ; forward to LIST-5

; ---

; the branch was here with LIST

;; LIST-4
L181F:  CALL    L1CDE           ; routine FETCH-NUM checks if a number
                                ; follows else uses zero.

;; LIST-5
L1822:  CALL    L1BEE           ; routine CHECK-END quits if syntax OK >>>

        CALL    L1E99           ; routine FIND-INT2 fetches the number
                                ; from the calculator stack in run-time.
        LD      A,B             ; fetch high byte of line number and
        AND     $3F             ; make less than $40 so that NEXT-ONE
                                ; (from LINE-ADDR) doesn't lose context.
                                ; Note. this is not satisfactory and the typo
                                ; LIST 20000 will list an entirely different
                                ; section than LIST 2000. Such typos are not
                                ; available for checking if they are direct
                                ; commands.

        LD      H,A             ; transfer the modified
        LD      L,C             ; line number to HL.
        LD      ($5C49),HL      ; update E_PPC to new line number.
        CALL    L196E           ; routine LINE-ADDR gets the address of the
                                ; line.

; This routine is called from AUTO-LIST
```

```
;; LIST-ALL
L1833:  LD      E,$01           ; signal current line not yet printed

;; LIST-ALL-2
L1835:  CALL    L1855           ; routine OUT-LINE outputs a BASIC line
                                ; using PRINT-OUT and makes an early return
                                ; when no more lines to print. >>>

        RST     10H             ; PRINT-A prints the carriage return (in A)

        BIT     4,(IY+$02)      ; test TV_FLAG  - automatic listing ?
        JR      Z,L1835         ; back to LIST-ALL-2 if not
                                ; (loop exit is via OUT-LINE)

; continue here if an automatic listing required.

        LD      A,($5C6B)       ; fetch DF_SZ lower display file size.
        SUB     (IY+$4F)        ; subtract S_POSN_hi ithe current line number.
        JR      NZ,L1835        ; back to LIST-ALL-2 if upper screen not full.

        XOR     E               ; A contains zero, E contains one if the
                                ; current edit line has not been printed
                                ; or zero if it has (from OUT-LINE).
        RET     Z               ; return if the screen is full and the line
                                ; has been printed.

; continue with automatic listings if the screen is full and the current
; edit line is missing. OUT-LINE will scroll automatically.

        PUSH    HL              ; save the pointer address.
        PUSH    DE              ; save the E flag.
        LD      HL,$5C6C        ; fetch S_TOP the rough estimate.
        CALL    L190F           ; routine LN-FETCH updates S_TOP with
                                ; the number of the next line.
        POP     DE              ; restore the E flag.
        POP     HL              ; restore the address of the next line.
        JR      L1835           ; back to LIST-ALL-2.

; ------------------------
; Print a whole BASIC line
; ------------------------
; This routine prints a whole BASIC line and it is called
; from LIST-ALL to output the line to current channel
; and from ED-EDIT to 'sprint' the line to the edit buffer.

;; OUT-LINE
L1855:  LD      BC,($5C49)      ; fetch E_PPC the current line which may be
                                ; unchecked and not exist.
        CALL    L1980           ; routine CP-LINES finds match or line after.
        LD      D,$3E           ; prepare cursor '>' in D.
        JR      Z,L1865         ; to OUT-LINE1 if matched or line after.

        LD      DE,$0000        ; put zero in D, to suppress line cursor.
        RL      E               ; pick up carry in E if line before current
                                ; leave E zero if same or after.

;; OUT-LINE1
L1865:  LD      (IY+$2D),E      ; save flag in BREG which is spare.
        LD      A,(HL)          ; get high byte of line number.
        CP      $40             ; is it too high ($2F is maximum possible) ?
        POP     BC              ; drop the return address and
        RET     NC              ; make an early return if so >>>
```

```
        PUSH    BC                ; save return address
        CALL    L1A28             ; routine OUT-NUM-2 to print addressed number
                                  ; with leading space.
        INC     HL                ; skip low number byte.
        INC     HL                ; and the two
        INC     HL                ; length bytes.
        RES     0,(IY+$01)        ; update FLAGS - signal leading space required.
        LD      A,D               ; fetch the cursor.
        AND     A                 ; test for zero.
        JR      Z,L1881           ; to OUT-LINE3 if zero.


        RST     10H               ; PRINT-A prints '>' the current line cursor.

; this entry point is called from ED-COPY

;; OUT-LINE2
L187D:  SET     0,(IY+$01)        ; update FLAGS - suppress leading space.

;; OUT-LINE3
L1881:  PUSH    DE                ; save flag E for a return value.
        EX      DE,HL             ; save HL address in DE.
        RES     2,(IY+$30)        ; update FLAGS2 - signal NOT in QUOTES.

        LD      HL,$5C3B          ; point to FLAGS.
        RES     2,(HL)            ; signal 'K' mode. (starts before keyword)
        BIT     5,(IY+$37)        ; test FLAGX - input mode ?
        JR      Z,L1894           ; forward to OUT-LINE4 if not.

        SET     2,(HL)            ; signal 'L' mode. (used for input)

;; OUT-LINE4
L1894:  LD      HL,($5C5F)        ; fetch X_PTR - possibly the error pointer
                                  ; address.
        AND     A                 ; clear the carry flag.
        SBC     HL,DE             ; test if an error address has been reached.
        JR      NZ,L18A1          ; forward to OUT-LINE5 if not.

        LD      A,$3F             ; load A with '?' the error marker.
        CALL    L18C1             ; routine OUT-FLASH to print flashing marker.

;; OUT-LINE5
L18A1:  CALL    L18E1             ; routine OUT-CURS will print the cursor if
                                  ; this is the right position.
        EX      DE,HL             ; restore address pointer to HL.
        LD      A,(HL)            ; fetch the addressed character.
        CALL    L18B6             ; routine NUMBER skips a hidden floating
                                  ; point number if present.
        INC     HL                ; now increment the pointer.
        CP      $0D               ; is character end-of-line ?
        JR      Z,L18B4           ; to OUT-LINE6, if so, as line is finished.

        EX      DE,HL             ; save the pointer in DE.
        CALL    L1937             ; routine OUT-CHAR to output character/token.

        JR      L1894             ; back to OUT-LINE4 until entire line is done.

; ---

;; OUT-LINE6
L18B4:  POP     DE                ; bring back the flag E, zero if current
                                  ; line printed else 1 if still to print.
        RET                       ; return with A holding $0D
```

```
; -------------------------
; Check for a number marker
; -------------------------
; this subroutine is called from two processes. while outputting BASIC lines
; and while searching statements within a BASIC line.
; during both, this routine will pass over an invisible number indicator
; and the five bytes floating-point number that follows it.
; Note that this causes floating point numbers to be stripped from
; the BASIC line when it is fetched to the edit buffer by OUT_LINE.
; the number marker also appears after the arguments of a DEF FN statement
; and may mask old 5-byte string parameters.

;; NUMBER
L18B6:  CP      $0E             ; character fourteen ?
        RET     NZ              ; return if not.

        INC     HL              ; skip the character
        INC     HL              ; and five bytes
        INC     HL              ; following.
        INC     HL              ;
        INC     HL              ;
        INC     HL              ;
        LD      A,(HL)          ; fetch the following character
        RET                     ; for return value.

; --------------------------
; Print a flashing character
; --------------------------
; This subroutine is called from OUT-LINE to print a flashing error
; marker '?' or from the next routine to print a flashing cursor e.g. 'L'.
; However, this only gets called from OUT-LINE when printing the edit line
; or the input buffer to the lower screen so a direct call to $09F4 can
; be used, even though out-line outputs to other streams.
; In fact the alternate set is used for the whole routine.

;; OUT-FLASH
L18C1:  EXX                     ; switch in alternate set

        LD      HL,($5C8F)      ; fetch L = ATTR_T, H = MASK-T
        PUSH    HL              ; save masks.
        RES     7,H             ; reset flash mask bit so active.
        SET     7,L             ; make attribute FLASH.
        LD      ($5C8F),HL      ; resave ATTR_T and MASK-T

        LD      HL,$5C91        ; address P_FLAG
        LD      D,(HL)          ; fetch to D
        PUSH    DE              ; and save.
        LD      (HL),$00        ; clear inverse, over, ink/paper 9

        CALL    L09F4           ; routine PRINT-OUT outputs character
                                ; without the need to vector via RST 10.

        POP     HL              ; pop P_FLAG to H.
        LD      (IY+$57),H      ; and restore system variable P_FLAG.
        POP     HL              ; restore temporary masks
        LD      ($5C8F),HL      ; and restore system variables ATTR_T/MASK_T

        EXX                     ; switch back to main set
        RET                     ; return

; ----------------
; Print the cursor
; ----------------
; This routine is called before any character is output while outputting
```

```
; a BASIC line or the input buffer. This includes listing to a printer
; or screen, copying a BASIC line to the edit buffer and printing the
; input buffer or edit buffer to the lower screen. It is only in the
; latter two cases that it has any relevance and in the last case it
; performs another very important function also.

;; OUT-CURS
L18E1:  LD      HL,($5C5B)      ; fetch K_CUR the current cursor address
        AND     A               ; prepare for true subtraction.
        SBC     HL,DE           ; test against pointer address in DE and
        RET     NZ              ; return if not at exact position.

; the value of MODE, maintained by KEY-INPUT, is tested and if non-zero
; then this value 'E' or 'G' will take precedence.

        LD      A,($5C41)       ; fetch MODE  0='KLC', 1='E', 2='G'.
        RLC     A               ; double the value and set flags.
        JR      Z,L18F3         ; to OUT-C-1 if still zero ('KLC').

        ADD     A,$43           ; add 'C' - will become 'E' if originally 1
                                ; or 'G' if originally 2.
        JR      L1909           ; forward to OUT-C-2 to print.

; ---

; If mode was zero then, while printing a BASIC line, bit 2 of flags has been
; set if 'THEN' or ':' was encountered as a main character and reset otherwise.
; This is now used to determine if the 'K' cursor is to be printed but this
; transient state is also now transferred permanently to bit 3 of FLAGS
; to let the interrupt routine know how to decode the next key.

;; OUT-C-1
L18F3:  LD      HL,$5C3B        ; Address FLAGS
        RES     3,(HL)          ; signal 'K' mode initially.
        LD      A,$4B           ; prepare letter 'K'.
        BIT     2,(HL)          ; test FLAGS - was the
                                ; previous main character ':' or 'THEN' ?
        JR      Z,L1909         ; forward to OUT-C-2 if so to print.

        SET     3,(HL)          ; signal 'L' mode to interrupt routine.
                                ; Note. transient bit has been made permanent.
        INC     A               ; augment from 'K' to 'L'.

        BIT     3,(IY+$30)      ; test FLAGS2 - consider caps lock ?
                                ; which is maintained by KEY-INPUT.
        JR      Z,L1909         ; forward to OUT-C-2 if not set to print.

        LD      A,$43           ; alter 'L' to 'C'.

;; OUT-C-2
L1909:  PUSH    DE              ; save address pointer but OK as OUT-FLASH
                                ; uses alternate set without RST 10.

        CALL    L18C1           ; routine OUT-FLASH to print.

        POP     DE              ; restore and
        RET                     ; return.

; ----------------------------
; Get line number of next line
; ----------------------------
; These two subroutines are called while editing.
; This entry point is from ED-DOWN with HL addressing E_PPC
; to fetch the next line number.
```

```
; Also from AUTO-LIST with HL addressing S_TOP just to update S_TOP
; with the value of the next line number. It gets fetched but is discarded.
; These routines never get called while the editor is being used for input.

;; LN-FETCH
L190F:  LD      E,(HL)          ; fetch low byte
        INC     HL              ; address next
        LD      D,(HL)          ; fetch high byte.
        PUSH    HL              ; save system variable hi pointer.
        EX      DE,HL           ; line number to HL,
        INC     HL              ; increment as a starting point.
        CALL    L196E           ; routine LINE-ADDR gets address in HL.
        CALL    L1695           ; routine LINE-NO gets line number in DE.
        POP     HL              ; restore system variable hi pointer.

; This entry point is from the ED-UP with HL addressing E_PPC_hi

;; LN-STORE
L191C:  BIT     5,(IY+$37)      ; test FLAGX - input mode ?
        RET     NZ              ; return if so.
                                ; Note. above already checked by ED-UP/ED-DOWN.

        LD      (HL),D          ; save high byte of line number.
        DEC     HL              ; address lower
        LD      (HL),E          ; save low byte of line number.
        RET                     ; return.

; -----------------------------------------
; Outputting numbers at start of BASIC line
; -----------------------------------------
; This routine entered at OUT-SP-NO is used to compute then output the first
; three digits of a 4-digit BASIC line printing a space if necessary.
; The line number, or residual part, is held in HL and the BC register
; holds a subtraction value -1000, -100 or -10.
; Note. for example line number 200 -
; space(out_char), 2(out_code), 0(out_char) final number always out-code.

;; OUT-SP-2
L1925:  LD      A,E             ; will be space if OUT-CODE not yet called.
                                ; or $FF if spaces are suppressed.
                                ; else $30 ('0').
                                ; (from the first instruction at OUT-CODE)
                                ; this guy is just too clever.
        AND     A               ; test bit 7 of A.
        RET     M               ; return if $FF, as leading spaces not
                                ; required. This is set when printing line
                                ; number and statement in MAIN-5.

        JR      L1937           ; forward to exit via OUT-CHAR.

; ---

; -> the single entry point.

;; OUT-SP-NO
L192A:  XOR     A               ; initialize digit to 0

;; OUT-SP-1
L192B:  ADD     HL,BC           ; add negative number to HL.
        INC     A               ; increment digit
        JR      C,L192B         ; back to OUT-SP-1 until no carry from
                                ; the addition.

        SBC     HL,BC           ; cancel the last addition
```

```
        DEC     A               ; and decrement the digit.
        JR      Z,L1925         ; back to OUT-SP-2 if it is zero.

        JP      L15EF           ; jump back to exit via OUT-CODE.    ->


; -------------------------------------
; Outputting characters in a BASIC line
; -------------------------------------
; This subroutine ...

;; OUT-CHAR
L1937:  CALL    L2D1B           ; routine NUMERIC tests if it is a digit ?
        JR      NC,L196C        ; to OUT-CH-3 to print digit without
                                ; changing mode. Will be 'K' mode if digits
                                ; are at beginning of edit line.

        CP      $21             ; less than quote character ?
        JR      C,L196C         ; to OUT-CH-3 to output controls and space.

        RES     2,(IY+$01)      ; initialize FLAGS to 'K' mode and leave
                                ; unchanged if this character would precede
                                ; a keyword.

        CP      $CB             ; is character 'THEN' token ?
        JR      Z,L196C         ; to OUT-CH-3 to output if so.

        CP      $3A             ; is it ':' ?
        JR      NZ,L195A        ; to OUT-CH-1 if not statement separator
                                ; to change mode back to 'L'.

        BIT     5,(IY+$37)      ; FLAGX  - Input Mode ??
        JR      NZ,L1968        ; to OUT-CH-2 if in input as no statements.
                                ; Note. this check should seemingly be at
                                ; the start. Commands seem inappropriate in
                                ; INPUT mode and are rejected by the syntax
                                ; checker anyway.
                                ; unless INPUT LINE is being used.

        BIT     2,(IY+$30)      ; test FLAGS2 - is the ':' within quotes ?
        JR      Z,L196C         ; to OUT-CH-3 if ':' is outside quoted text.

        JR      L1968           ; to OUT-CH-2 as ':' is within quotes

; ---

;; OUT-CH-1
L195A:  CP      $22             ; is it quote character '"'  ?
        JR      NZ,L1968        ; to OUT-CH-2 with others to set 'L' mode.

        PUSH    AF              ; save character.
        LD      A,($5C6A)       ; fetch FLAGS2.
        XOR     $04             ; toggle the quotes flag.
        LD      ($5C6A),A       ; update FLAGS2
        POP     AF              ; and restore character.

;; OUT-CH-2
L1968:  SET     2,(IY+$01)      ; update FLAGS - signal L mode if the cursor
                                ; is next.

;; OUT-CH-3
L196C:  RST     10H             ; PRINT-A vectors the character to
                                ; channel 'S', 'K', 'R' or 'P'.
        RET                     ; return.
```

```
; ---------------------------------------------
; Get starting address of line, or line after
; ---------------------------------------------
; This routine is used often to get the address, in HL, of a BASIC line
; number supplied in HL, or failing that the address of the following line
; and the address of the previous line in DE.

;; LINE-ADDR
L196E:  PUSH    HL              ; save line number in HL register
        LD      HL,($5C53)      ; fetch start of program from PROG
        LD      D,H             ; transfer address to
        LD      E,L             ; the DE register pair.

;; LINE-AD-1
L1974:  POP     BC              ; restore the line number to BC
        CALL    L1980           ; routine CP-LINES compares with that
                                ; addressed by HL
        RET     NC              ; return if line has been passed or matched.
                                ; if NZ, address of previous is in DE

        PUSH    BC              ; save the current line number
        CALL    L19B8           ; routine NEXT-ONE finds address of next
                                ; line number in DE, previous in HL.
        EX      DE,HL           ; switch so next in HL
        JR      L1974           ; back to LINE-AD-1 for another comparison

; --------------------
; Compare line numbers
; --------------------
; This routine compares a line number supplied in BC with an addressed
; line number pointed to by HL.

;; CP-LINES
L1980:  LD      A,(HL)          ; Load the high byte of line number and
        CP      B               ; compare with that of supplied line number.
        RET     NZ              ; return if yet to match (carry will be set).

        INC     HL              ; address low byte of
        LD      A,(HL)          ; number and pick up in A.
        DEC     HL              ; step back to first position.
        CP      C               ; now compare.
        RET                     ; zero set if exact match.
                                ; carry set if yet to match.
                                ; no carry indicates a match or
                                ; next available BASIC line or
                                ; program end marker.

; -------------------
; Find each statement
; -------------------
; The single entry point EACH-STMT is used to
; 1) To find the D'th statement in a line.
; 2) To find a token in held E.

;; not-used
L1988:  INC     HL              ;
        INC     HL              ;
        INC     HL              ;

; -> entry point.

;; EACH-STMT
L198B:  LD      ($5C5D),HL      ; save HL in CH_ADD
```

```
        LD      C,$00           ; initialize quotes flag

;; EACH-S-1
L1990:  DEC     D               ; decrease statement count
        RET     Z               ; return if zero


        RST     20H             ; NEXT-CHAR
        CP      E               ; is it the search token ?
        JR      NZ,L199A        ; forward to EACH-S-3 if not

        AND     A               ; clear carry
        RET                     ; return signalling success.

; ---

;; EACH-S-2
L1998:  INC     HL              ; next address
        LD      A,(HL)          ; next character

;; EACH-S-3
L199A:  CALL    L18B6           ; routine NUMBER skips if number marker
        LD      ($5C5D),HL      ; save in CH_ADD
        CP      $22             ; is it quotes '"' ?
        JR      NZ,L19A5        ; to EACH-S-4 if not

        DEC     C               ; toggle bit 0 of C

;; EACH-S-4
L19A5:  CP      $3A             ; is it ':'
        JR      Z,L19AD         ; to EACH-S-5

        CP      $CB             ; 'THEN'
        JR      NZ,L19B1        ; to EACH-S-6

;; EACH-S-5
L19AD:  BIT     0,C             ; is it in quotes
        JR      Z,L1990         ; to EACH-S-1 if not

;; EACH-S-6
L19B1:  CP      $0D             ; end of line ?
        JR      NZ,L1998        ; to EACH-S-2

        DEC     D               ; decrease the statement counter
                                ; which should be zero else
                                ; 'Statement Lost'.
        SCF                     ; set carry flag - not found
        RET                     ; return

; -----------------------------------------------------------------------
; Storage of variables. For full details - see chapter 24.
; ZX Spectrum BASIC Programming by Steven Vickers 1982.
; It is bits 7-5 of the first character of a variable that allow
; the six types to be distinguished. Bits 4-0 are the reduced letter.
; So any variable name is higher that $3F and can be distinguished
; also from the variables area end-marker $80.
;
; 76543210 meaning                               brief outline of format.
; -------- ------------------------              -----------------------
; 010      string variable.                      2 byte length + contents.
; 110      string array.                         2 byte length + contents.
; 100      array of numbers.                     2 byte length + contents.
; 011      simple numeric variable.              5 bytes.
; 101      variable length named numeric.        5 bytes.
```

```
; 111      for-next loop variable.              18 bytes.
; 10000000 the variables area end-marker.
;
; Note. any of the above seven will serve as a program end-marker.
;
; -----------------------------------------------------------------------

; ------------
; Get next one
; ------------
; This versatile routine is used to find the address of the next line
; in the program area or the next variable in the variables area.
; The reason one routine is made to handle two apparently unrelated tasks
; is that it can be called indiscriminately when merging a line or a
; variable.

;; NEXT-ONE
L19B8:  PUSH    HL              ; save the pointer address.
        LD      A,(HL)          ; get first byte.
        CP      $40             ; compare with upper limit for line numbers.
        JR      C,L19D5         ; forward to NEXT-O-3 if within BASIC area.

; the continuation here is for the next variable unless the supplied
; line number was erroneously over 16383. see RESTORE command.

        BIT     5,A             ; is it a string or an array variable ?
        JR      Z,L19D6         ; forward to NEXT-O-4 to compute length.

        ADD     A,A             ; test bit 6 for single-character variables.
        JP      M,L19C7         ; forward to NEXT-O-1 if so

        CCF                     ; clear the carry for long-named variables.
                                ; it remains set for for-next loop variables.

;; NEXT-O-1
L19C7:  LD      BC,$0005        ; set BC to 5 for floating point number
        JR      NC,L19CE        ; forward to NEXT-O-2 if not a for/next
                                ; variable.

        LD      C,$12           ; set BC to eighteen locations.
                                ; value, limit, step, line and statement.

; now deal with long-named variables

;; NEXT-O-2
L19CE:  RLA                     ; test if character inverted. carry will also
                                ; be set for single character variables
        INC     HL              ; address next location.
        LD      A,(HL)          ; and load character.
        JR      NC,L19CE        ; back to NEXT-O-2 if not inverted bit.
                                ; forward immediately with single character
                                ; variable names.

        JR      L19DB           ; forward to NEXT-O-5 to add length of
                                ; floating point number(s etc.).

; ---

; this branch is for line numbers.

;; NEXT-O-3
L19D5:  INC     HL              ; increment pointer to low byte of line no.

; strings and arrays rejoin here
```

```
;; NEXT-O-4
L19D6:  INC     HL              ; increment to address the length low byte.
        LD      C,(HL)          ; transfer to C and
        INC     HL              ; point to high byte of length.
        LD      B,(HL)          ; transfer that to B
        INC     HL              ; point to start of BASIC/variable contents.

; the three types of numeric variables rejoin here

;; NEXT-O-5
L19DB:  ADD     HL,BC           ; add the length to give address of next
                                ; line/variable in HL.
        POP     DE              ; restore previous address to DE.

; ------------------
; Difference routine
; ------------------
; This routine terminates the above routine and is also called from the
; start of the next routine to calculate the length to reclaim.

;; DIFFER
L19DD:  AND     A               ; prepare for true subtraction.
        SBC     HL,DE           ; subtract the two pointers.
        LD      B,H             ; transfer result
        LD      C,L             ; to BC register pair.
        ADD     HL,DE           ; add back
        EX      DE,HL           ; and switch pointers
        RET                     ; return values are the length of area in BC,
                                ; low pointer (previous) in HL,
                                ; high pointer (next) in DE.

; -----------------------
; Handle reclaiming space
; -----------------------
;

;; RECLAIM-1
L19E5:  CALL    L19DD           ; routine DIFFER immediately above

;; RECLAIM-2
L19E8:  PUSH    BC              ;

        LD      A,B             ;
        CPL                     ;
        LD      B,A             ;
        LD      A,C             ;
        CPL                     ;
        LD      C,A             ;
        INC     BC              ;

        CALL    L1664           ; routine POINTERS
        EX      DE,HL           ;
        POP     HL              ;

        ADD     HL,DE           ;
        PUSH    DE              ;
        LDIR                    ; copy bytes

        POP     HL              ;
        RET                     ;

; ---------------------------------------
; Read line number of line in editing area
```

```
; ----------------------------------------
; This routine reads a line number in the editing area returning the number
; in the BC register or zero if no digits exist before commands.
; It is called from LINE-SCAN to check the syntax of the digits.
; It is called from MAIN-3 to extract the line number in preparation for
; inclusion of the line in the BASIC program area.
;
; Interestingly the calculator stack is moved from its normal place at the
; end of dynamic memory to an adequate area within the system variables area.
; This ensures that in a low memory situation, that valid line numbers can
; be extracted without raising an error and that memory can be reclaimed
; by deleting lines. If the stack was in its normal place then a situation
; arises whereby the Spectrum becomes locked with no means of reclaiming space.

;; E-LINE-NO
L19FB:  LD      HL,($5C59)      ; load HL from system variable E_LINE.

        DEC     HL              ; decrease so that NEXT_CHAR can be used
                                ; without skipping the first digit.

        LD      ($5C5D),HL      ; store in the system variable CH_ADD.

        RST     20H             ; NEXT-CHAR skips any noise and white-space
                                ; to point exactly at the first digit.

        LD      HL,$5C92        ; use MEM-0 as a temporary calculator stack
                                ; an overhead of three locations are needed.
        LD      ($5C65),HL      ; set new STKEND.

        CALL    L2D3B           ; routine INT-TO-FP will read digits till
                                ; a non-digit found.
        CALL    L2DA2           ; routine FP-TO-BC will retrieve number
                                ; from stack at membot.
        JR      C,L1A15         ; forward to E-L-1 if overflow i.e. > 65535.
                                ; 'Nonsense in BASIC'

        LD      HL,$D8F0        ; load HL with value -9999
        ADD     HL,BC           ; add to line number in BC

;; E-L-1
L1A15:  JP      C,L1C8A         ; to REPORT-C 'Nonsense in BASIC' if over.
                                ; Note. As ERR_SP points to ED_ERROR
                                ; the report is never produced although
                                ; the RST $08 will update X_PTR leading to
                                ; the error marker being displayed when
                                ; the ED_LOOP is reiterated.
                                ; in fact, since it is immediately
                                ; cancelled, any report will do.

; a line in the range 0 - 9999 has been entered.

        JP      L16C5           ; jump back to SET-STK to set the calculator
                                ; stack back to its normal place and exit
                                ; from there.

; ---------------------------------
; Report and line number outputting
; ---------------------------------
; Entry point OUT-NUM-1 is used by the Error Reporting code to print
; the line number and later the statement number held in BC.
; If the statement was part of a direct command then -2 is used as a
; dummy line number so that zero will be printed in the report.
; This routine is also used to print the exponent of E-format numbers.
;
```

```
; Entry point OUT-NUM-2 is used from OUT-LINE to output the line number
; addressed by HL with leading spaces if necessary.

;; OUT-NUM-1
L1A1B:  PUSH    DE              ; save the
        PUSH    HL              ; registers.
        XOR     A               ; set A to zero.
        BIT     7,B             ; is the line number minus two ?
        JR      NZ,L1A42        ; forward to OUT-NUM-4 if so to print zero
                                ; for a direct command.

        LD      H,B             ; transfer the
        LD      L,C             ; number to HL.
        LD      E,$FF           ; signal 'no leading zeros'.
        JR      L1A30           ; forward to continue at OUT-NUM-3

; ---

; from OUT-LINE - HL addresses line number.

;; OUT-NUM-2
L1A28:  PUSH    DE              ; save flags
        LD      D,(HL)          ; high byte to D
        INC     HL              ; address next
        LD      E,(HL)          ; low byte to E
        PUSH    HL              ; save pointer
        EX      DE,HL           ; transfer number to HL
        LD      E,$20           ; signal 'output leading spaces'

;; OUT-NUM-3
L1A30:  LD      BC,$FC18        ; value -1000
        CALL    L192A           ; routine OUT-SP-NO outputs space or number
        LD      BC,$FF9C        ; value -100
        CALL    L192A           ; routine OUT-SP-NO
        LD      C,$F6           ; value -10 ( B is still $FF )
        CALL    L192A           ; routine OUT-SP-NO
        LD      A,L             ; remainder to A.

;; OUT-NUM-4
L1A42:  CALL    L15EF           ; routine OUT-CODE for final digit.
                                ; else report code zero wouldn't get
                                ; printed.
        POP     HL              ; restore the
        POP     DE              ; registers and
        RET                     ; return.


;***************************************************
;** Part 7. BASIC LINE AND COMMAND INTERPRETATION **
;***************************************************

; ----------------
; The offset table
; ----------------
; The BASIC interpreter has found a command code $CE - $FF
; which is then reduced to range $00 - $31 and added to the base address
; of this table to give the address of an offset which, when added to
; the offset therein, gives the location in the following parameter table
; where a list of class codes, separators and addresses relevant to the
; command exists.

;; offst-tbl
L1A48:  DEFB    L1AF9 - $       ; B1 offset to Address: P-DEF-FN
        DEFB    L1B14 - $       ; CB offset to Address: P-CAT
```

```
        DEFB    L1B06 - $       ; BC offset to Address: P-FORMAT
        DEFB    L1B0A - $       ; BF offset to Address: P-MOVE
        DEFB    L1B10 - $       ; C4 offset to Address: P-ERASE
        DEFB    L1AFC - $       ; AF offset to Address: P-OPEN
        DEFB    L1B02 - $       ; B4 offset to Address: P-CLOSE
        DEFB    L1AE2 - $       ; 93 offset to Address: P-MERGE
        DEFB    L1AE1 - $       ; 91 offset to Address: P-VERIFY
        DEFB    L1AE3 - $       ; 92 offset to Address: P-BEEP
        DEFB    L1AE7 - $       ; 95 offset to Address: P-CIRCLE
        DEFB    L1AEB - $       ; 98 offset to Address: P-INK
        DEFB    L1AEC - $       ; 98 offset to Address: P-PAPER
        DEFB    L1AED - $       ; 98 offset to Address: P-FLASH
        DEFB    L1AEE - $       ; 98 offset to Address: P-BRIGHT
        DEFB    L1AEF - $       ; 98 offset to Address: P-INVERSE
        DEFB    L1AF0 - $       ; 98 offset to Address: P-OVER
        DEFB    L1AF1 - $       ; 98 offset to Address: P-OUT
        DEFB    L1AD9 - $       ; 7F offset to Address: P-LPRINT
        DEFB    L1ADC - $       ; 81 offset to Address: P-LLIST
        DEFB    L1A8A - $       ; 2E offset to Address: P-STOP
        DEFB    L1AC9 - $       ; 6C offset to Address: P-READ
        DEFB    L1ACC - $       ; 6E offset to Address: P-DATA
        DEFB    L1ACF - $       ; 70 offset to Address: P-RESTORE
        DEFB    L1AA8 - $       ; 48 offset to Address: P-NEW
        DEFB    L1AF5 - $       ; 94 offset to Address: P-BORDER
        DEFB    L1AB8 - $       ; 56 offset to Address: P-CONT
        DEFB    L1AA2 - $       ; 3F offset to Address: P-DIM
        DEFB    L1AA5 - $       ; 41 offset to Address: P-REM
        DEFB    L1A90 - $       ; 2B offset to Address: P-FOR
        DEFB    L1A7D - $       ; 17 offset to Address: P-GO-TO
        DEFB    L1A86 - $       ; 1F offset to Address: P-GO-SUB
        DEFB    L1A9F - $       ; 37 offset to Address: P-INPUT
        DEFB    L1AE0 - $       ; 77 offset to Address: P-LOAD
        DEFB    L1AAE - $       ; 44 offset to Address: P-LIST
        DEFB    L1A7A - $       ; 0F offset to Address: P-LET
        DEFB    L1AC5 - $       ; 59 offset to Address: P-PAUSE
        DEFB    L1A98 - $       ; 2B offset to Address: P-NEXT
        DEFB    L1AB1 - $       ; 43 offset to Address: P-POKE
        DEFB    L1A9C - $       ; 2D offset to Address: P-PRINT
        DEFB    L1AC1 - $       ; 51 offset to Address: P-PLOT
        DEFB    L1AAB - $       ; 3A offset to Address: P-RUN
        DEFB    L1ADF - $       ; 6D offset to Address: P-SAVE
        DEFB    L1AB5 - $       ; 42 offset to Address: P-RANDOM
        DEFB    L1A81 - $       ; 0D offset to Address: P-IF
        DEFB    L1ABE - $       ; 49 offset to Address: P-CLS
        DEFB    L1AD2 - $       ; 5C offset to Address: P-DRAW
        DEFB    L1ABB - $       ; 44 offset to Address: P-CLEAR
        DEFB    L1A8D - $       ; 15 offset to Address: P-RETURN
        DEFB    L1AD6 - $       ; 5D offset to Address: P-COPY


; -------------------------------
; The parameter or "Syntax" table
; -------------------------------
; For each command there exists a variable list of parameters.
; If the character is greater than a space it is a required separator.
; If less, then it is a command class in the range 00 - 0B.
; Note that classes 00, 03 and 05 will fetch the addresses from this table.
; Some classes e.g. 07 and 0B have the same address in all invocations
; and the command is re-computed from the low-byte of the parameter address.
; Some e.g. 02 are only called once so a call to the command is made from
; within the class routine rather than holding the address within the table.
; Some class routines check syntax entirely and some leave this task for the
; command itself.
; Others for example CIRCLE (x,y,z) check the first part (x,y) using the
```

```
; class routine and the final part (,z) within the command.
; The last few commands appear to have been added in a rush but their syntax
; is rather simple e.g. MOVE "M1","M2"

;; P-LET
L1A7A:  DEFB    $01                 ; Class-01 - A variable is required.
        DEFB    $3D                 ; Separator:  '='
        DEFB    $02                 ; Class-02 - An expression, numeric or string,
                                    ; must follow.

;; P-GO-TO
L1A7D:  DEFB    $06                 ; Class-06 - A numeric expression must follow.
        DEFB    $00                 ; Class-00 - No further operands.
        DEFW    L1E67               ; Address: $1E67; Address: GO-TO

;; P-IF
L1A81:  DEFB    $06                 ; Class-06 - A numeric expression must follow.
        DEFB    $CB                 ; Separator:  'THEN'
        DEFB    $05                 ; Class-05 - Variable syntax checked
                                    ; by routine.
        DEFW    L1CF0               ; Address: $1CF0; Address: IF

;; P-GO-SUB
L1A86:  DEFB    $06                 ; Class-06 - A numeric expression must follow.
        DEFB    $00                 ; Class-00 - No further operands.
        DEFW    L1EED               ; Address: $1EED; Address: GO-SUB

;; P-STOP
L1A8A:  DEFB    $00                 ; Class-00 - No further operands.
        DEFW    L1CEE               ; Address: $1CEE; Address: STOP

;; P-RETURN
L1A8D:  DEFB    $00                 ; Class-00 - No further operands.
        DEFW    L1F23               ; Address: $1F23; Address: RETURN

;; P-FOR
L1A90:  DEFB    $04                 ; Class-04 - A single character variable must
                                    ; follow.
        DEFB    $3D                 ; Separator:  '='
        DEFB    $06                 ; Class-06 - A numeric expression must follow.
        DEFB    $CC                 ; Separator:  'TO'
        DEFB    $06                 ; Class-06 - A numeric expression must follow.
        DEFB    $05                 ; Class-05 - Variable syntax checked
                                    ; by routine.
        DEFW    L1D03               ; Address: $1D03; Address: FOR

;; P-NEXT
L1A98:  DEFB    $04                 ; Class-04 - A single character variable must
                                    ; follow.
        DEFB    $00                 ; Class-00 - No further operands.
        DEFW    L1DAB               ; Address: $1DAB; Address: NEXT

;; P-PRINT
L1A9C:  DEFB    $05                 ; Class-05 - Variable syntax checked entirely
                                    ; by routine.
        DEFW    L1FCD               ; Address: $1FCD; Address: PRINT

;; P-INPUT
L1A9F:  DEFB    $05                 ; Class-05 - Variable syntax checked entirely
                                    ; by routine.
        DEFW    L2089               ; Address: $2089; Address: INPUT

;; P-DIM
L1AA2:  DEFB    $05                 ; Class-05 - Variable syntax checked entirely
```

```
                                        ; by routine.
        DEFW    L2C02                   ; Address: $2C02; Address: DIM

;; P-REM
L1AA5:  DEFB    $05                     ; Class-05 - Variable syntax checked entirely
                                        ; by routine.
        DEFW    L1BB2                   ; Address: $1BB2; Address: REM

;; P-NEW
L1AA8:  DEFB    $00                     ; Class-00 - No further operands.
        DEFW    L11B7                   ; Address: $11B7; Address: NEW

;; P-RUN
L1AAB:  DEFB    $03                     ; Class-03 - A numeric expression may follow
                                        ; else default to zero.
        DEFW    L1EA1                   ; Address: $1EA1; Address: RUN

;; P-LIST
L1AAE:  DEFB    $05                     ; Class-05 - Variable syntax checked entirely
                                        ; by routine.
        DEFW    L17F9                   ; Address: $17F9; Address: LIST

;; P-POKE
L1AB1:  DEFB    $08                     ; Class-08 - Two comma-separated numeric
                                        ; expressions required.
        DEFB    $00                     ; Class-00 - No further operands.
        DEFW    L1E80                   ; Address: $1E80; Address: POKE

;; P-RANDOM
L1AB5:  DEFB    $03                     ; Class-03 - A numeric expression may follow
                                        ; else default to zero.
        DEFW    L1E4F                   ; Address: $1E4F; Address: RANDOMIZE

;; P-CONT
L1AB8:  DEFB    $00                     ; Class-00 - No further operands.
        DEFW    L1E5F                   ; Address: $1E5F; Address: CONTINUE

;; P-CLEAR
L1ABB:  DEFB    $03                     ; Class-03 - A numeric expression may follow
                                        ; else default to zero.
        DEFW    L1EAC                   ; Address: $1EAC; Address: CLEAR

;; P-CLS
L1ABE:  DEFB    $00                     ; Class-00 - No further operands.
        DEFW    L0D6B                   ; Address: $0D6B; Address: CLS

;; P-PLOT
L1AC1:  DEFB    $09                     ; Class-09 - Two comma-separated numeric
                                        ; expressions required with optional colour
                                        ; items.
        DEFB    $00                     ; Class-00 - No further operands.
        DEFW    L22DC                   ; Address: $22DC; Address: PLOT

;; P-PAUSE
L1AC5:  DEFB    $06                     ; Class-06 - A numeric expression must follow.
        DEFB    $00                     ; Class-00 - No further operands.
        DEFW    L1F3A                   ; Address: $1F3A; Address: PAUSE

;; P-READ
L1AC9:  DEFB    $05                     ; Class-05 - Variable syntax checked entirely
                                        ; by routine.
        DEFW    L1DED                   ; Address: $1DED; Address: READ

;; P-DATA
```

```
L1ACC:  DEFB    $05                 ; Class-05 - Variable syntax checked entirely
                                    ; by routine.
        DEFW    L1E27               ; Address: $1E27; Address: DATA

;; P-RESTORE
L1ACF:  DEFB    $03                 ; Class-03 - A numeric expression may follow
                                    ; else default to zero.
        DEFW    L1E42               ; Address: $1E42; Address: RESTORE

;; P-DRAW
L1AD2:  DEFB    $09                 ; Class-09 - Two comma-separated numeric
                                    ; expressions required with optional colour
                                    ; items.
        DEFB    $05                 ; Class-05 - Variable syntax checked
                                    ; by routine.
        DEFW    L2382               ; Address: $2382; Address: DRAW

;; P-COPY
L1AD6:  DEFB    $00                 ; Class-00 - No further operands.
        DEFW    L0EAC               ; Address: $0EAC; Address: COPY

;; P-LPRINT
L1AD9:  DEFB    $05                 ; Class-05 - Variable syntax checked entirely
                                    ; by routine.
        DEFW    L1FC9               ; Address: $1FC9; Address: LPRINT

;; P-LLIST
L1ADC:  DEFB    $05                 ; Class-05 - Variable syntax checked entirely
                                    ; by routine.
        DEFW    L17F5               ; Address: $17F5; Address: LLIST

;; P-SAVE
L1ADF:  DEFB    $0B                 ; Class-0B - Offset address converted to tape
                                    ; command.

;; P-LOAD
L1AE0:  DEFB    $0B                 ; Class-0B - Offset address converted to tape
                                    ; command.

;; P-VERIFY
L1AE1:  DEFB    $0B                 ; Class-0B - Offset address converted to tape
                                    ; command.

;; P-MERGE
L1AE2:  DEFB    $0B                 ; Class-0B - Offset address converted to tape
                                    ; command.

;; P-BEEP
L1AE3:  DEFB    $08                 ; Class-08 - Two comma-separated numeric
                                    ; expressions required.
        DEFB    $00                 ; Class-00 - No further operands.
        DEFW    L03F8               ; Address: $03F8; Address: BEEP

;; P-CIRCLE
L1AE7:  DEFB    $09                 ; Class-09 - Two comma-separated numeric
                                    ; expressions required with optional colour
                                    ; items.
        DEFB    $05                 ; Class-05 - Variable syntax checked
                                    ; by routine.
        DEFW    L2320               ; Address: $2320; Address: CIRCLE

;; P-INK
L1AEB:  DEFB    $07                 ; Class-07 - Offset address is converted to
                                    ; colour code.
```

```
;; P-PAPER
L1AEC:  DEFB    $07                 ; Class-07 - Offset address is converted to
                                    ; colour code.

;; P-FLASH
L1AED:  DEFB    $07                 ; Class-07 - Offset address is converted to
                                    ; colour code.

;; P-BRIGHT
L1AEE:  DEFB    $07                 ; Class-07 - Offset address is converted to
                                    ; colour code.

;; P-INVERSE
L1AEF:  DEFB    $07                 ; Class-07 - Offset address is converted to
                                    ; colour code.

;; P-OVER
L1AF0:  DEFB    $07                 ; Class-07 - Offset address is converted to
                                    ; colour code.

;; P-OUT
L1AF1:  DEFB    $08                 ; Class-08 - Two comma-separated numeric
                                    ; expressions required.
        DEFB    $00                 ; Class-00 - No further operands.
        DEFW    L1E7A               ; Address: $1E7A; Address: OUT

;; P-BORDER
L1AF5:  DEFB    $06                 ; Class-06 - A numeric expression must follow.
        DEFB    $00                 ; Class-00 - No further operands.
        DEFW    L2294               ; Address: $2294; Address: BORDER

;; P-DEF-FN
L1AF9:  DEFB    $05                 ; Class-05 - Variable syntax checked entirely
                                    ; by routine.
        DEFW    L1F60               ; Address: $1F60; Address: DEF-FN

;; P-OPEN
L1AFC:  DEFB    $06                 ; Class-06 - A numeric expression must follow.
        DEFB    $2C                 ; Separator:  ','          see Footnote *
        DEFB    $0A                 ; Class-0A - A string expression must follow.
        DEFB    $00                 ; Class-00 - No further operands.
        DEFW    L1736               ; Address: $1736; Address: OPEN

;; P-CLOSE
L1B02:  DEFB    $06                 ; Class-06 - A numeric expression must follow.
        DEFB    $00                 ; Class-00 - No further operands.
        DEFW    L16E5               ; Address: $16E5; Address: CLOSE

;; P-FORMAT
L1B06:  DEFB    $0A                 ; Class-0A - A string expression must follow.
        DEFB    $00                 ; Class-00 - No further operands.
        DEFW    L1793               ; Address: $1793; Address: CAT-ETC

;; P-MOVE
L1B0A:  DEFB    $0A                 ; Class-0A - A string expression must follow.
        DEFB    $2C                 ; Separator:  ','
        DEFB    $0A                 ; Class-0A - A string expression must follow.
        DEFB    $00                 ; Class-00 - No further operands.
        DEFW    L1793               ; Address: $1793; Address: CAT-ETC

;; P-ERASE
L1B10:  DEFB    $0A                 ; Class-0A - A string expression must follow.
        DEFB    $00                 ; Class-00 - No further operands.
```

```
        DEFW    L1793           ; Address: $1793; Address: CAT-ETC

;; P-CAT
L1B14:  DEFB    $00             ; Class-00 - No further operands.
        DEFW    L1793           ; Address: $1793; Address: CAT-ETC

; * Note that a comma is required as a separator with the OPEN command
; but the Interface 1 programmers relaxed this allowing ';' as an
; alternative for their channels creating a confusing mixture of
; allowable syntax as it is this ROM which opens or re-opens the
; normal channels.

; -------------------------------
; Main parser (BASIC interpreter)
; -------------------------------
; This routine is called once from MAIN-2 when the BASIC line is to
; be entered or re-entered into the Program area and the syntax
; requires checking.

;; LINE-SCAN
L1B17:  RES     7,(IY+$01)      ; update FLAGS - signal checking syntax
        CALL    L19FB           ; routine E-LINE-NO              >>
                                ; fetches the line number if in range.

        XOR     A               ; clear the accumulator.
        LD      ($5C47),A       ; set statement number SUBPPC to zero.
        DEC     A               ; set accumulator to $FF.
        LD      ($5C3A),A       ; set ERR_NR to 'OK' - 1.
        JR      L1B29           ; forward to continue at STMT-L-1.

; --------------
; Statement loop
; --------------
;
;

;; STMT-LOOP
L1B28:  RST     20H             ; NEXT-CHAR

; -> the entry point from above or LINE-RUN
;; STMT-L-1
L1B29:  CALL    L16BF           ; routine SET-WORK clears workspace etc.

        INC     (IY+$0D)        ; increment statement number SUBPPC
        JP      M,L1C8A         ; to REPORT-C to raise
                                ; 'Nonsense in BASIC' if over 127.

        RST     18H             ; GET-CHAR

        LD      B,$00           ; set B to zero for later indexing.
                                ; early so any other reason ???

        CP      $0D             ; is character carriage return ?
                                ; i.e. an empty statement.
        JR      Z,L1BB3         ; forward to LINE-END if so.

        CP      $3A             ; is it statement end marker ':' ?
                                ; i.e. another type of empty statement.
        JR      Z,L1B28         ; back to STMT-LOOP if so.

        LD      HL,L1B76        ; address: STMT-RET
        PUSH    HL              ; is now pushed as a return address
        LD      C,A             ; transfer the current character to C.
```

```
; advance CH_ADD to a position after command and test if it is a command.

        RST     20H             ; NEXT-CHAR to advance pointer
        LD      A,C             ; restore current character
        SUB     $CE             ; subtract 'DEF FN' - first command
        JP      C,L1C8A         ; jump to REPORT-C if less than a command
                                ; raising
                                ; 'Nonsense in BASIC'

        LD      C,A             ; put the valid command code back in C.
                                ; register B is zero.
        LD      HL,L1A48        ; address: offst-tbl
        ADD     HL,BC           ; index into table with one of 50 commands.
        LD      C,(HL)          ; pick up displacement to syntax table entry.
        ADD     HL,BC           ; add to address the relevant entry.
        JR      L1B55           ; forward to continue at GET-PARAM

; ----------------------
; The main scanning loop
; ----------------------
; not documented properly
;

;; SCAN-LOOP
L1B52:  LD      HL,($5C74)      ; fetch temporary address from T_ADDR
                                ; during subsequent loops.

; -> the initial entry point with HL addressing start of syntax table entry.

;; GET-PARAM
L1B55:  LD      A,(HL)          ; pick up the parameter.
        INC     HL              ; address next one.
        LD      ($5C74),HL      ; save pointer in system variable T_ADDR

        LD      BC,L1B52        ; address: SCAN-LOOP
        PUSH    BC              ; is now pushed on stack as looping address.
        LD      C,A             ; store parameter in C.
        CP      $20             ; is it greater than ' '  ?
        JR      NC,L1B6F        ; forward to SEPARATOR to check that correct
                                ; separator appears in statement if so.

        LD      HL,L1C01        ; address: class-tbl.
        LD      B,$00           ; prepare to index into the class table.
        ADD     HL,BC           ; index to find displacement to routine.
        LD      C,(HL)          ; displacement to BC
        ADD     HL,BC           ; add to address the CLASS routine.
        PUSH    HL              ; push the address on the stack.

        RST     18H             ; GET-CHAR - HL points to place in statement.

        DEC     B               ; reset the zero flag - the initial state
                                ; for all class routines.

        RET                     ; and make an indirect jump to routine
                                ; and then SCAN-LOOP (also on stack).

; Note. one of the class routines will eventually drop the return address
; off the stack breaking out of the above seemingly endless loop.

; ----------------
; Verify separator
; ----------------
; This routine is called once to verify that the mandatory separator
; present in the parameter table is also present in the correct
```

```
; location following the command. For example, the 'THEN' token after
; the 'IF' token and expression.

;; SEPARATOR
L1B6F:  RST     18H             ; GET-CHAR
        CP      C               ; does it match the character in C ?
        JP      NZ,L1C8A        ; jump forward to REPORT-C if not
                                ; 'Nonsense in BASIC'.

        RST     20H             ; NEXT-CHAR advance to next character
        RET                     ; return.

; ------------------------------
; Come here after interpretation
; ------------------------------
;
;

;; STMT-RET
L1B76:  CALL    L1F54           ; routine BREAK-KEY is tested after every
                                ; statement.
        JR      C,L1B7D         ; step forward to STMT-R-1 if not pressed.

;; REPORT-L
L1B7B:  RST     08H             ; ERROR-1
        DEFB    $14             ; Error Report: BREAK into program

;; STMT-R-1
L1B7D:  BIT     7,(IY+$0A)      ; test NSPPC - will be set if $FF -
                                ; no jump to be made.
        JR      NZ,L1BF4        ; forward to STMT-NEXT if a program line.

        LD      HL,($5C42)      ; fetch line number from NEWPPC
        BIT     7,H             ; will be set if minus two - direct command(s)
        JR      Z,L1B9E         ; forward to LINE-NEW if a jump is to be
                                ; made to a new program line/statement.

; --------------------
; Run a direct command
; --------------------
; A direct command is to be run or, if continuing from above,
; the next statement of a direct command is to be considered.

;; LINE-RUN
L1B8A:  LD      HL,$FFFE        ; The dummy value minus two
        LD      ($5C45),HL      ; is set/reset as line number in PPC.
        LD      HL,($5C61)      ; point to end of line + 1 - WORKSP.
        DEC     HL              ; now point to $80 end-marker.
        LD      DE,($5C59)      ; address the start of line E_LINE.
        DEC     DE              ; now location before - for GET-CHAR.
        LD      A,($5C44)       ; load statement to A from NSPPC.
        JR      L1BD1           ; forward to NEXT-LINE.

; ------------------------------
; Find start address of new line
; ------------------------------
; The branch was to here if a jump is to made to a new line number
; and statement.
; That is the previous statement was a GO TO, GO SUB, RUN, RETURN, NEXT etc..

;; LINE-NEW
L1B9E:  CALL    L196E           ; routine LINE-ADDR gets address of line
                                ; returning zero flag set if line found.
        LD      A,($5C44)       ; fetch new statement from NSPPC
```

```
        JR      Z,L1BBF         ; forward to LINE-USE if line matched.

; continue as must be a direct command.

        AND     A               ; test statement which should be zero
        JR      NZ,L1BEC        ; forward to REPORT-N if not.
                                ; 'Statement lost'

;

        LD      B,A             ; save statement in B. ?
        LD      A,(HL)          ; fetch high byte of line number.
        AND     $C0             ; test if using direct command
                                ; a program line is less than $3F
        LD      A,B             ; retrieve statement.
                                ; (we can assume it is zero).
        JR      Z,L1BBF         ; forward to LINE-USE if was a program line

; Alternatively a direct statement has finished correctly.

;; REPORT-0
L1BB0:  RST     08H             ; ERROR-1
        DEFB    $FF             ; Error Report: OK

; ------------------
; Handle REM command
; ------------------
; The REM command routine.
; The return address STMT-RET is dropped and the rest of line ignored.

;; REM
L1BB2:  POP     BC              ; drop return address STMT-RET and
                                ; continue ignoring rest of line.

; ------------
; End of line?
; ------------
;
;

;; LINE-END
L1BB3:  CALL    L2530           ; routine SYNTAX-Z  (UNSTACK-Z?)
        RET     Z               ; return if checking syntax.

        LD      HL,($5C55)      ; fetch NXTLIN to HL.
        LD      A,$C0           ; test against the
        AND     (HL)            ; system limit $3F.
        RET     NZ              ; return if more as must be
                                ; end of program.
                                ; (or direct command)

        XOR     A               ; set statement to zero.

; and continue to set up the next following line and then consider this new one.

; ---------------------
; General line checking
; ---------------------
; The branch was here from LINE-NEW if BASIC is branching.
; or a continuation from above if dealing with a new sequential line.
; First make statement zero number one leaving others unaffected.

;; LINE-USE
L1BBF:  CP      $01             ; will set carry if zero.
```

```
        ADC     A,$00           ; add in any carry.

        LD      D,(HL)          ; high byte of line number to D.
        INC     HL              ; advance pointer.
        LD      E,(HL)          ; low byte of line number to E.
        LD      ($5C45),DE      ; set system variable PPC.

        INC     HL              ; advance pointer.
        LD      E,(HL)          ; low byte of line length to E.
        INC     HL              ; advance pointer.
        LD      D,(HL)          ; high byte of line length to D.

        EX      DE,HL           ; swap pointer to DE before
        ADD     HL,DE           ; adding to address the end of line.
        INC     HL              ; advance to start of next line.

; -----------------------------
; Update NEXT LINE but consider
; previous line or edit line.
; -----------------------------
; The pointer will be the next line if continuing from above or to
; edit line end-marker ($80) if from LINE-RUN.

;; NEXT-LINE
L1BD1:  LD      ($5C55),HL      ; store pointer in system variable NXTLIN

        EX      DE,HL           ; bring back pointer to previous or edit line
        LD      ($5C5D),HL      ; and update CH_ADD with character address.

        LD      D,A             ; store statement in D.
        LD      E,$00           ; set E to zero to suppress token searching
                                ; if EACH-STMT is to be called.
        LD      (IY+$0A),$FF    ; set statement NSPPC to $FF signalling
                                ; no jump to be made.
        DEC     D               ; decrement and test statement
        LD      (IY+$0D),D      ; set SUBPPC to decremented statement number.
        JP      Z,L1B28         ; to STMT-LOOP if result zero as statement is
                                ; at start of line and address is known.

        INC     D               ; else restore statement.
        CALL    L198B           ; routine EACH-STMT finds the D'th statement
                                ; address as E does not contain a token.
        JR      Z,L1BF4         ; forward to STMT-NEXT if address found.

;; REPORT-N
L1BEC:  RST     08H             ; ERROR-1
        DEFB    $16             ; Error Report: Statement lost

; -----------------
; End of statement?
; -----------------
; This combination of routines is called from 20 places when
; the end of a statement should have been reached and all preceding
; syntax is in order.

;; CHECK-END
L1BEE:  CALL    L2530           ; routine SYNTAX-Z
        RET     NZ              ; return immediately in runtime

        POP     BC              ; drop address of calling routine.
        POP     BC              ; drop address STMT-RET.
                                ; and continue to find next statement.

; --------------------
```

```
; Go to next statement
; --------------------
; Acceptable characters at this point are carriage return and ':'.
; If so go to next statement which in the first case will be on next line.

;; STMT-NEXT
L1BF4:  RST     18H             ; GET-CHAR - ignoring white space etc.

        CP      $0D             ; is it carriage return ?
        JR      Z,L1BB3         ; back to LINE-END if so.

        CP      $3A             ; is it ':' ?
        JP      Z,L1B28         ; jump back to STMT-LOOP to consider
                                ; further statements

        JP      L1C8A           ; jump to REPORT-C with any other character
                                ; 'Nonsense in BASIC'.

; Note. the two-byte sequence 'rst 08; defb $0b' could replace the above jp.

; -------------------
; Command class table
; -------------------
;

;; class-tbl
L1C01:  DEFB    L1C10 - $       ; 0F offset to Address: CLASS-00
        DEFB    L1C1F - $       ; 1D offset to Address: CLASS-01
        DEFB    L1C4E - $       ; 4B offset to Address: CLASS-02
        DEFB    L1C0D - $       ; 09 offset to Address: CLASS-03
        DEFB    L1C6C - $       ; 67 offset to Address: CLASS-04
        DEFB    L1C11 - $       ; 0B offset to Address: CLASS-05
        DEFB    L1C82 - $       ; 7B offset to Address: CLASS-06
        DEFB    L1C96 - $       ; 8E offset to Address: CLASS-07
        DEFB    L1C7A - $       ; 71 offset to Address: CLASS-08
        DEFB    L1CBE - $       ; B4 offset to Address: CLASS-09
        DEFB    L1C8C - $       ; 81 offset to Address: CLASS-0A
        DEFB    L1CDB - $       ; CF offset to Address: CLASS-0B


; --------------------------------
; Command classes---00, 03, and 05
; --------------------------------
; class-03 e.g. RUN or RUN 200   ;  optional operand
; class-00 e.g. CONTINUE         ;  no operand
; class-05 e.g. PRINT            ;  variable syntax checked by routine

;; CLASS-03
L1C0D:  CALL    L1CDE           ; routine FETCH-NUM

;; CLASS-00

L1C10:  CP      A               ; reset zero flag.

; if entering here then all class routines are entered with zero reset.

;; CLASS-05
L1C11:  POP     BC              ; drop address SCAN-LOOP.
        CALL    Z,L1BEE         ; if zero set then call routine CHECK-END >>>
                                ; as should be no further characters.

        EX      DE,HL           ; save HL to DE.
        LD      HL,($5C74)      ; fetch T_ADDR
        LD      C,(HL)          ; fetch low byte of routine
```

```
        INC     HL                ; address next.
        LD      B,(HL)            ; fetch high byte of routine.
        EX      DE,HL             ; restore HL from DE
        PUSH    BC                ; push the address
        RET                       ; and make an indirect jump to the command.

; ----------------------------------
; Command classes---01, 02, and 04
; ----------------------------------
; class-01  e.g. LET A = 2*3     ; a variable is reqd

; This class routine is also called from INPUT and READ to find the
; destination variable for an assignment.

;; CLASS-01
L1C1F:  CALL    L28B2             ; routine LOOK-VARS returns carry set if not
                                  ; found in runtime.

; ----------------------
; Variable in assignment
; ----------------------
;
;

;; VAR-A-1
L1C22:  LD      (IY+$37),$00      ; set FLAGX to zero
        JR      NC,L1C30          ; forward to VAR-A-2 if found or checking
                                  ; syntax.

        SET     1,(IY+$37)        ; FLAGX  - Signal a new variable
        JR      NZ,L1C46          ; to VAR-A-3 if not assigning to an array
                                  ; e.g. LET a$(3,3) = "X"

;; REPORT-2
L1C2E:  RST     08H               ; ERROR-1
        DEFB    $01               ; Error Report: Variable not found

;; VAR-A-2
L1C30:  CALL    Z,L2996           ; routine STK-VAR considers a subscript/slice
        BIT     6,(IY+$01)        ; test FLAGS  - Numeric or string result ?
        JR      NZ,L1C46          ; to VAR-A-3 if numeric

        XOR     A                 ; default to array/slice - to be retained.
        CALL    L2530             ; routine SYNTAX-Z
        CALL    NZ,L2BF1          ; routine STK-FETCH is called in runtime
                                  ; may overwrite A with 1.
        LD      HL,$5C71          ; address system variable FLAGX
        OR      (HL)              ; set bit 0 if simple variable to be reclaimed
        LD      (HL),A            ; update FLAGX
        EX      DE,HL             ; start of string/subscript to DE

;; VAR-A-3
L1C46:  LD      ($5C72),BC        ; update STRLEN
        LD      ($5C4D),HL        ; and DEST of assigned string.
        RET                       ; return.

; -------------------------------------------------
; class-02 e.g. LET a = 1 + 1   ; an expression must follow

;; CLASS-02
L1C4E:  POP     BC                ; drop return address SCAN-LOOP
        CALL    L1C56             ; routine VAL-FET-1 is called to check
                                  ; expression and assign result in runtime
        CALL    L1BEE             ; routine CHECK-END checks nothing else
```

```
                                ; is present in statement.
        RET                     ; Return

; -------------
; Fetch a value
; -------------
;
;

;; VAL-FET-1
L1C56:  LD      A,($5C3B)       ; initial FLAGS to A

;; VAL-FET-2
L1C59:  PUSH    AF              ; save A briefly
        CALL    L24FB           ; routine SCANNING evaluates expression.
        POP     AF              ; restore A
        LD      D,(IY+$01)      ; post-SCANNING FLAGS to D
        XOR     D               ; xor the two sets of flags
        AND     $40             ; pick up bit 6 of xored FLAGS should be zero
        JR      NZ,L1C8A        ; forward to REPORT-C if not zero
                                ; 'Nonsense in BASIC' - results don't agree.

        BIT     7,D             ; test FLAGS - is syntax being checked ?
        JP      NZ,L2AFF        ; jump forward to LET to make the assignment
                                ; in runtime.

        RET                     ; but return from here if checking syntax.

; ------------------
; Command class---04
; ------------------
; class-04 e.g. FOR i            ; a single character variable must follow

;; CLASS-04
L1C6C:  CALL    L28B2           ; routine LOOK-VARS
        PUSH    AF              ; preserve flags.
        LD      A,C             ; fetch type - should be 011xxxxx
        OR      $9F             ; combine with 10011111.
        INC     A               ; test if now $FF by incrementing.
        JR      NZ,L1C8A        ; forward to REPORT-C if result not zero.

        POP     AF              ; else restore flags.
        JR      L1C22           ; back to VAR-A-1


; --------------------------------
; Expect numeric/string expression
; --------------------------------
; This routine is used to get the two coordinates of STRING$, ATTR and POINT.
; It is also called from PRINT-ITEM to get the two numeric expressions that
; follow the AT ( in PRINT AT, INPUT AT).

;; NEXT-2NUM
L1C79:  RST     20H             ; NEXT-CHAR advance past 'AT' or '('.

; --------
; class-08 e.g. POKE 65535,2     ; two numeric expressions separated by comma
;; CLASS-08
;; EXPT-2NUM
L1C7A:  CALL    L1C82           ; routine EXPT-1NUM is called for first
                                ; numeric expression
        CP      $2C             ; is character ',' ?
        JR      NZ,L1C8A        ; to REPORT-C if not required separator.
                                ; 'Nonsense in BASIC'.
```

```
        RST     20H             ; NEXT-CHAR

; ->
;  class-06  e.g. GOTO a*1000   ; a numeric expression must follow
;; CLASS-06
;; EXPT-1NUM
L1C82:  CALL    L24FB           ; routine SCANNING
        BIT     6,(IY+$01)      ; test FLAGS  - Numeric or string result ?
        RET     NZ              ; return if result is numeric.

;; REPORT-C
L1C8A:  RST     08H             ; ERROR-1
        DEFB    $0B             ; Error Report: Nonsense in BASIC

; -----------------------------------------------------------------
; class-0A e.g. ERASE "????"    ; a string expression must follow.
;                               ; these only occur in unimplemented commands
;                               ; although the routine expt-exp is called
;                               ; from SAVE-ETC

;; CLASS-0A
;; EXPT-EXP
L1C8C:  CALL    L24FB           ; routine SCANNING
        BIT     6,(IY+$01)      ; test FLAGS  - Numeric or string result ?
        RET     Z               ; return if string result.

        JR      L1C8A           ; back to REPORT-C if numeric.

; ---------------------
; Set permanent colours
; class 07
; ---------------------
; class-07 e.g. PAPER 6          ; a single class for a collection of
;                               ; similar commands. Clever.
;
; Note. these commands should ensure that current channel is 'S'

;; CLASS-07
L1C96:  BIT     7,(IY+$01)      ; test FLAGS - checking syntax only ?
                                ; Note. there is a subroutine to do this.
        RES     0,(IY+$02)      ; update TV_FLAG - signal main screen in use
        CALL    NZ,L0D4D        ; routine TEMPS is called in runtime.
        POP     AF              ; drop return address SCAN-LOOP
        LD      A,($5C74)       ; T_ADDR_lo to accumulator.
                                ; points to '$07' entry + 1
                                ; e.g. for INK points to $EC now

; Note if you move alter the syntax table next line may have to be altered.

; Note. For ZASM assembler replace following expression with SUB $13.

L1CA5:  SUB     L1AEB-$D8 % 256 ; convert $EB to $D8 ('INK') etc.
                                ; ( is SUB $13 in standard ROM )

        CALL    L21FC           ; routine CO-TEMP-4
        CALL    L1BEE           ; routine CHECK-END check that nothing else
                                ; in statement.

; return here in runtime.

        LD      HL,($5C8F)      ; pick up ATTR_T and MASK_T
        LD      ($5C8D),HL      ; and store in ATTR_P and MASK_P
        LD      HL,$5C91        ; point to P_FLAG.
```

```
        LD      A,(HL)          ; pick up in A
        RLCA                    ; rotate to left
        XOR     (HL)            ; combine with HL
        AND     $AA             ; 10101010
        XOR     (HL)            ; only permanent bits affected
        LD      (HL),A          ; reload into P_FLAG.
        RET                     ; return.

; ------------------
; Command class---09
; ------------------
; e.g. PLOT PAPER 0; 128,88     ; two coordinates preceded by optional
;                               ; embedded colour items.
;
; Note. this command should ensure that current channel is actually 'S'.

;; CLASS-09
L1CBE:  CALL    L2530           ; routine SYNTAX-Z
        JR      Z,L1CD6         ; forward to CL-09-1 if checking syntax.

        RES     0,(IY+$02)      ; update TV_FLAG - signal main screen in use
        CALL    L0D4D           ; routine TEMPS is called.
        LD      HL,$5C90        ; point to MASK_T
        LD      A,(HL)          ; fetch mask to accumulator.
        OR      $F8             ; or with 11111000 paper/bright/flash 8
        LD      (HL),A          ; mask back to MASK_T system variable.
        RES     6,(IY+$57)      ; reset P_FLAG  - signal NOT PAPER 9 ?

        RST     18H             ; GET-CHAR

;; CL-09-1
L1CD6:  CALL    L21E2           ; routine CO-TEMP-2 deals with any embedded
                                ; colour items.
        JR      L1C7A           ; exit via EXPT-2NUM to check for x,y.

; Note. if either of the numeric expressions contain STR$ then the flag setting
; above will be undone when the channel flags are reset during STR$.
; e.g.
; 10 BORDER 3 : PLOT VAL STR$ 128, VAL STR$ 100
; credit John Elliott.

; ------------------
; Command class---0B
; ------------------
; Again a single class for four commands.
; This command just jumps back to SAVE-ETC to handle the four tape commands.
; The routine itself works out which command has called it by examining the
; address in T_ADDR_lo. Note therefore that the syntax table has to be
; located where these and other sequential command addresses are not split
; over a page boundary.

;; CLASS-0B
L1CDB:  JP      L0605           ; jump way back to SAVE-ETC

; --------------
; Fetch a number
; --------------
; This routine is called from CLASS-03 when a command may be followed by
; an optional numeric expression e.g. RUN. If the end of statement has
; been reached then zero is used as the default.
; Also called from LIST-4.

;; FETCH-NUM
L1CDE:  CP      $0D             ; is character a carriage return ?
```

```
        JR      Z,L1CE6         ; forward to USE-ZERO if so

        CP      $3A             ; is it ':' ?
        JR      NZ,L1C82        ; forward to EXPT-1NUM if not.
                                ; else continue and use zero.

; ----------------
; Use zero routine
; ----------------
; This routine is called four times to place the value zero on the
; calculator stack as a default value in runtime.

;; USE-ZERO
L1CE6:  CALL    L2530           ; routine SYNTAX-Z  (UNSTACK-Z?)
        RET     Z               ;

        RST     28H             ;; FP-CALC
        DEFB    $A0             ;;stk-zero       ;0.
        DEFB    $38             ;;end-calc

        RET                     ; return.

; -------------------
; Handle STOP command
; -------------------
; Command Syntax: STOP
; One of the shortest and least used commands. As with 'OK' not an error.

;; REPORT-9
;; STOP
L1CEE:  RST     08H             ; ERROR-1
        DEFB    $08             ; Error Report: STOP statement

; -----------------
; Handle IF command
; -----------------
; e.g. IF score>100 THEN PRINT "You Win"
; The parser has already checked the expression the result of which is on
; the calculator stack. The presence of the 'THEN' separator has also been
; checked and CH-ADD points to the command after THEN.
;

;; IF
L1CF0:  POP     BC              ; drop return address - STMT-RET
        CALL    L2530           ; routine SYNTAX-Z
        JR      Z,L1D00         ; forward to IF-1 if checking syntax
                                ; to check syntax of PRINT "You Win"


        RST     28H             ;; FP-CALC    score>100 (1=TRUE 0=FALSE)
        DEFB    $02             ;;delete      .
        DEFB    $38             ;;end-calc

        EX      DE,HL           ; make HL point to deleted value
        CALL    L34E9           ; routine TEST-ZERO
        JP      C,L1BB3         ; jump to LINE-END if FALSE (0)

;; IF-1
L1D00:  JP      L1B29           ; to STMT-L-1, if true (1) to execute command
                                ; after 'THEN' token.

; ------------------
; Handle FOR command
; ------------------
```

```
; e.g. FOR i = 0 TO 1 STEP 0.1
; Using the syntax tables, the parser has already checked for a start and
; limit value and also for the intervening separator.
; the two values v,l are on the calculator stack.
; CLASS-04 has also checked the variable and the name is in STRLEN_lo.
; The routine begins by checking for an optional STEP.

;; FOR
L1D03:  CP      $CD             ; is there a 'STEP' ?
        JR      NZ,L1D10        ; to F-USE-1 if not to use 1 as default.

        RST     20H             ; NEXT-CHAR
        CALL    L1C82           ; routine EXPT-1NUM
        CALL    L1BEE           ; routine CHECK-END
        JR      L1D16           ; to F-REORDER

; ---

;; F-USE-1
L1D10:  CALL    L1BEE           ; routine CHECK-END

        RST     28H             ;; FP-CALC      v,l.
        DEFB    $A1             ;;stk-one      v,l,1=s.
        DEFB    $38             ;;end-calc


;; F-REORDER
L1D16:  RST     28H             ;; FP-CALC       v,l,s.
        DEFB    $C0             ;;st-mem-0       v,l,s.
        DEFB    $02             ;;delete         v,l.
        DEFB    $01             ;;exchange       l,v.
        DEFB    $E0             ;;get-mem-0      l,v,s.
        DEFB    $01             ;;exchange       l,s,v.
        DEFB    $38             ;;end-calc

        CALL    L2AFF           ; routine LET assigns the initial value v to
                                ; the variable altering type if necessary.
        LD      ($5C68),HL      ; The system variable MEM is made to point to
                                ; the variable instead of its normal
                                ; location MEMBOT
        DEC     HL              ; point to single-character name
        LD      A,(HL)          ; fetch name
        SET     7,(HL)          ; set bit 7 at location
        LD      BC,$0006        ; add six to HL
        ADD     HL,BC           ; to address where limit should be.
        RLCA                    ; test bit 7 of original name.
        JR      C,L1D34         ; forward to F-L-S if already a FOR/NEXT
                                ; variable

        LD      C,$0D           ; otherwise an additional 13 bytes are needed.
                                ; 5 for each value, two for line number and
                                ; 1 byte for looping statement.
        CALL    L1655           ; routine MAKE-ROOM creates them.
        INC     HL              ; make HL address limit.

;; F-L-S
L1D34:  PUSH    HL              ; save position.

        RST     28H             ;; FP-CALC         l,s.
        DEFB    $02             ;;delete           l.
        DEFB    $02             ;;delete           .
        DEFB    $38             ;;end-calc
                                ; DE points to STKEND, l.
```

```
        POP     HL              ; restore variable position
        EX      DE,HL           ; swap pointers
        LD      C,$0A           ; ten bytes to move
        LDIR                    ; Copy 'deleted' values to variable.
        LD      HL,($5C45)      ; Load with current line number from PPC
        EX      DE,HL           ; exchange pointers.
        LD      (HL),E          ; save the looping line
        INC     HL              ; in the next
        LD      (HL),D          ; two locations.
        LD      D,(IY+$0D)      ; fetch statement from SUBPPC system variable.
        INC     D               ; increment statement.
        INC     HL              ; and pointer
        LD      (HL),D          ; and store the looping statement.
                                ;
        CALL    L1DDA           ; routine NEXT-LOOP considers an initial
        RET     NC              ; iteration. Return to STMT-RET if a loop is
                                ; possible to execute next statement.

; no loop is possible so execution continues after the matching 'NEXT'

        LD      B,(IY+$38)      ; get single-character name from STRLEN_lo
        LD      HL,($5C45)      ; get the current line from PPC
        LD      ($5C42),HL      ; and store it in NEWPPC
        LD      A,($5C47)       ; fetch current statement from SUBPPC
        NEG                     ; Negate as counter decrements from zero
                                ; initially and we are in the middle of a
                                ; line.
        LD      D,A             ; Store result in D.
        LD      HL,($5C5D)      ; get current address from CH_ADD
        LD      E,$F3           ; search will be for token 'NEXT'

;; F-LOOP
L1D64:  PUSH    BC              ; save variable name.
        LD      BC,($5C55)      ; fetch NXTLIN
        CALL    L1D86           ; routine LOOK-PROG searches for 'NEXT' token.
        LD      ($5C55),BC      ; update NXTLIN
        POP     BC              ; and fetch the letter
        JR      C,L1D84         ; forward to REPORT-I if the end of program
                                ; was reached by LOOK-PROG.
                                ; 'FOR without NEXT'

        RST     20H             ; NEXT-CHAR fetches character after NEXT
        OR      $20             ; ensure it is upper-case.
        CP      B               ; compare with FOR variable name
        JR      Z,L1D7C         ; forward to F-FOUND if it matches.

; but if no match i.e. nested FOR/NEXT loops then continue search.

        RST     20H             ; NEXT-CHAR
        JR      L1D64           ; back to F-LOOP

; ---


;; F-FOUND
L1D7C:  RST     20H             ; NEXT-CHAR
        LD      A,$01           ; subtract the negated counter from 1
        SUB     D               ; to give the statement after the NEXT
        LD      ($5C44),A       ; set system variable NSPPC
        RET                     ; return to STMT-RET to branch to new
                                ; line and statement. ->
; ---

;; REPORT-I
```

```
L1D84:  RST     08H             ; ERROR-1
        DEFB    $11             ; Error Report: FOR without NEXT

; ---------
; LOOK-PROG
; ---------
; Find DATA, DEF FN or NEXT.
; This routine searches the program area for one of the above three keywords.
; On entry, HL points to start of search area.
; The token is in E, and D holds a statement count, decremented from zero.

;; LOOK-PROG
L1D86:  LD      A,(HL)          ; fetch current character
        CP      $3A             ; is it ':' a statement separator ?
        JR      Z,L1DA3         ; forward to LOOK-P-2 if so.

; The starting point was PROG - 1 or the end of a line.

;; LOOK-P-1
L1D8B:  INC     HL              ; increment pointer to address
        LD      A,(HL)          ; the high byte of line number
        AND     $C0             ; test for program end marker $80 or a
                                ; variable
        SCF                     ; Set Carry Flag
        RET     NZ              ; return with carry set if at end
                                ; of program.           ->

        LD      B,(HL)          ; high byte of line number to B
        INC     HL              ;
        LD      C,(HL)          ; low byte to C.
        LD      ($5C42),BC      ; set system variable NEWPPC.
        INC     HL              ;
        LD      C,(HL)          ; low byte of line length to C.
        INC     HL              ;
        LD      B,(HL)          ; high byte to B.
        PUSH    HL              ; save address
        ADD     HL,BC           ; add length to position.
        LD      B,H             ; and save result
        LD      C,L             ; in BC.
        POP     HL              ; restore address.
        LD      D,$00           ; initialize statement counter to zero.

;; LOOK-P-2
L1DA3:  PUSH    BC              ; save address of next line
        CALL    L198B           ; routine EACH-STMT searches current line.
        POP     BC              ; restore address.
        RET     NC              ; return if match was found. ->

        JR      L1D8B           ; back to LOOK-P-1 for next line.

; -------------------
; Handle NEXT command
; -------------------
; e.g. NEXT i
; The parameter tables have already evaluated the presence of a variable

;; NEXT
L1DAB:  BIT     1,(IY+$37)      ; test FLAGX - handling a new variable ?
        JP      NZ,L1C2E        ; jump back to REPORT-2 if so
                                ; 'Variable not found'

; now test if found variable is a simple variable uninitialized by a FOR.

        LD      HL,($5C4D)      ; load address of variable from DEST
```

```
        BIT     7,(HL)          ; is it correct type ?
        JR      Z,L1DD8         ; forward to REPORT-1 if not
                                ; 'NEXT without FOR'

        INC     HL              ; step past variable name
        LD      ($5C68),HL      ; and set MEM to point to three 5-byte values
                                ; value, limit, step.

        RST     28H             ;; FP-CALC     add step and re-store
        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

        CALL    L1DDA           ; routine NEXT-LOOP tests against limit.
        RET     C               ; return if no more iterations possible.

        LD      HL,($5C68)      ; find start of variable contents from MEM.
        LD      DE,$000F        ; add 3*5 to
        ADD     HL,DE           ; address the looping line number
        LD      E,(HL)          ; low byte to E
        INC     HL              ;
        LD      D,(HL)          ; high byte to D
        INC     HL              ; address looping statement
        LD      H,(HL)          ; and store in H
        EX      DE,HL           ; swap registers
        JP      L1E73           ; exit via GO-TO-2 to execute another loop.

; ---

;; REPORT-1
L1DD8:  RST     08H             ; ERROR-1
        DEFB    $00             ; Error Report: NEXT without FOR


; -----------------
; Perform NEXT loop
; -----------------
; This routine is called from the FOR command to test for an initial
; iteration and from the NEXT command to test for all subsequent iterations.
; the system variable MEM addresses the variable's contents which, in the
; latter case, have had the step, possibly negative, added to the value.

;; NEXT-LOOP
L1DDA:  RST     28H             ;; FP-CALC
        DEFB    $E1             ;;get-mem-1        l.
        DEFB    $E0             ;;get-mem-0        l,v.
        DEFB    $E2             ;;get-mem-2        l,v,s.
        DEFB    $36             ;;less-0           l,v,(1/0) negative step ?
        DEFB    $00             ;;jump-true        l,v.(1/0)

        DEFB    $02             ;;to L1DE2, NEXT-1 if step negative

        DEFB    $01             ;;exchange         v,l.

;; NEXT-1
L1DE2:  DEFB    $03             ;;subtract         l-v OR v-l.
        DEFB    $37             ;;greater-0        (1/0)
        DEFB    $00             ;;jump-true        .

        DEFB    $04             ;;to L1DE9, NEXT-2 if no more iterations.
```

```
        DEFB    $38             ;;end-calc          .

        AND     A               ; clear carry flag signalling another loop.
        RET                     ; return

; ---

;; NEXT-2
L1DE9:  DEFB    $38             ;;end-calc          .

        SCF                     ; set carry flag signalling looping exhausted.
        RET                     ; return


; -------------------
; Handle READ command
; -------------------
; e.g. READ a, b$, c$(1000 TO 3000)
; A list of comma-separated variables is assigned from a list of
; comma-separated expressions.
; As it moves along the first list, the character address CH_ADD is stored
; in X_PTR while CH_ADD is used to read the second list.

;; READ-3
L1DEC:  RST     20H             ; NEXT-CHAR

; -> Entry point.
;; READ
L1DED:  CALL    L1C1F           ; routine CLASS-01 checks variable.
        CALL    L2530           ; routine SYNTAX-Z
        JR      Z,L1E1E         ; forward to READ-2 if checking syntax


        RST     18H             ; GET-CHAR
        LD      ($5C5F),HL      ; save character position in X_PTR.
        LD      HL,($5C57)      ; load HL with Data Address DATADD, which is
                                ; the start of the program or the address
                                ; after the last expression that was read or
                                ; the address of the line number of the
                                ; last RESTORE command.
        LD      A,(HL)          ; fetch character
        CP      $2C             ; is it a comma ?
        JR      Z,L1E0A         ; forward to READ-1 if so.

; else all data in this statement has been read so look for next DATA token

        LD      E,$E4           ; token 'DATA'
        CALL    L1D86           ; routine LOOK-PROG
        JR      NC,L1E0A        ; forward to READ-1 if DATA found

; else report the error.

;; REPORT-E
L1E08:  RST     08H             ; ERROR-1
        DEFB    $0D             ; Error Report: Out of DATA

;; READ-1
L1E0A:  CALL    L0077           ; routine TEMP-PTR1 advances updating CH_ADD
                                ; with new DATADD position.
        CALL    L1C56           ; routine VAL-FET-1 assigns value to variable
                                ; checking type match and adjusting CH_ADD.

        RST     18H             ; GET-CHAR fetches adjusted character position
        LD      ($5C57),HL      ; store back in DATADD
```

```
        LD      HL,($5C5F)       ; fetch X_PTR  the original READ CH_ADD
        LD      (IY+$26),$00     ; now nullify X_PTR_hi
        CALL    L0078            ; routine TEMP-PTR2 restores READ CH_ADD

;; READ-2
L1E1E:  RST     18H              ; GET-CHAR
        CP      $2C              ; is it ',' indicating more variables to read ?
        JR      Z,L1DEC          ; back to READ-3 if so

        CALL    L1BEE            ; routine CHECK-END
        RET                      ; return from here in runtime to STMT-RET.

; -------------------
; Handle DATA command
; -------------------
; In runtime this 'command' is passed by but the syntax is checked when such
; a statement is found while parsing a line.
; e.g. DATA 1, 2, "text", score-1, a$(location, room, object), FN r(49),
;         wages - tax, TRUE, The meaning of life

;; DATA
L1E27:  CALL    L2530            ; routine SYNTAX-Z to check status
        JR      NZ,L1E37         ; forward to DATA-2 if in runtime

;; DATA-1
L1E2C:  CALL    L24FB            ; routine SCANNING to check syntax of
                                 ; expression
        CP      $2C              ; is it a comma ?
        CALL    NZ,L1BEE         ; routine CHECK-END checks that statement
                                 ; is complete. Will make an early exit if
                                 ; so. >>>
        RST     20H              ; NEXT-CHAR
        JR      L1E2C            ; back to DATA-1

; ---

;; DATA-2
L1E37:  LD      A,$E4            ; set token to 'DATA' and continue into
                                 ; the PASS-BY routine.


; ----------------------------------
; Check statement for DATA or DEF FN
; ----------------------------------
; This routine is used to backtrack to a command token and then
; forward to the next statement in runtime.

;; PASS-BY
L1E39:  LD      B,A              ; Give BC enough space to find token.
        CPDR                     ; Compare decrement and repeat. (Only use).
                                 ; Work backwards till keyword is found which
                                 ; is start of statement before any quotes.
                                 ; HL points to location before keyword.
        LD      DE,$0200         ; count 1+1 statements, dummy value in E to
                                 ; inhibit searching for a token.
        JP      L198B            ; to EACH-STMT to find next statement

; -----------------------------------------------------------------------
; A General Note on Invalid Line Numbers.
; =======================================
; One of the revolutionary concepts of Sinclair BASIC was that it supported
; virtual line numbers. That is the destination of a GO TO, RESTORE etc. need
; not exist. It could be a point before or after an actual line number.
; Zero suffices for a before but the after should logically be infinity.
```

```
; Since the maximum actual line limit is 9999 then the system limit, 16383
; when variables kick in, would serve fine as a virtual end point.
; However, ironically, only the LOAD command gets it right. It will not
; autostart a program that has been saved with a line higher than 16383.
; All the other commands deal with the limit unsatisfactorily.
; LIST, RUN, GO TO, GO SUB and RESTORE have problems and the latter may
; crash the machine when supplied with an inappropriate virtual line number.
; This is puzzling as very careful consideration must have been given to
; this point when the new variable types were allocated their masks and also
; when the routine NEXT-ONE was successfully re-written to reflect this.
; An enigma.
; -------------------------------------------------------------------------

; ----------------------
; Handle RESTORE command
; ----------------------
; The restore command sets the system variable for the data address to
; point to the location before the supplied line number or first line
; thereafter.
; This alters the position where subsequent READ commands look for data.
; Note. If supplied with inappropriate high numbers the system may crash
; in the LINE-ADDR routine as it will pass the program/variables end-marker
; and then lose control of what it is looking for - variable or line number.
; - observation, Steven Vickers, 1984, Pitman.

;; RESTORE
L1E42:  CALL    L1E99           ; routine FIND-INT2 puts integer in BC.
                                ; Note. B should be checked against limit $3F
                                ; and an error generated if higher.

; this entry point is used from RUN command with BC holding zero

;; REST-RUN
L1E45:  LD      H,B             ; transfer the line
        LD      L,C             ; number to the HL register.
        CALL    L196E           ; routine LINE-ADDR to fetch the address.
        DEC     HL              ; point to the location before the line.
        LD      ($5C57),HL      ; update system variable DATADD.
        RET                     ; return to STMT-RET (or RUN)

; ------------------------
; Handle RANDOMIZE command
; ------------------------
; This command sets the SEED for the RND function to a fixed value.
; With the parameter zero, a random start point is used depending on
; how long the computer has been switched on.

;; RANDOMIZE
L1E4F:  CALL    L1E99           ; routine FIND-INT2 puts parameter in BC.
        LD      A,B             ; test this
        OR      C               ; for zero.
        JR      NZ,L1E5A        ; forward to RAND-1 if not zero.

        LD      BC,($5C78)      ; use the lower two bytes at FRAMES1.

;; RAND-1
L1E5A:  LD      ($5C76),BC      ; place in SEED system variable.
        RET                     ; return to STMT-RET

; -----------------------
; Handle CONTINUE command
; -----------------------
; The CONTINUE command transfers the OLD (but incremented) values of
; line number and statement to the equivalent "NEW VALUE" system variables
```

```
; by using the last part of GO TO and exits indirectly to STMT-RET.

;; CONTINUE
L1E5F:  LD      HL,($5C6E)       ; fetch OLDPPC line number.
        LD      D,(IY+$36)       ; fetch OSPPC statement.
        JR      L1E73            ; forward to GO-TO-2

; --------------------
; Handle GO TO command
; --------------------
; The GO TO command routine is also called by GO SUB and RUN routines
; to evaluate the parameters of both commands.
; It updates the system variables used to fetch the next line/statement.
; It is at STMT-RET that the actual change in control takes place.
; Unlike some BASICs the line number need not exist.
; Note. the high byte of the line number is incorrectly compared with $F0
; instead of $3F. This leads to commands with operands greater than 32767
; being considered as having been run from the editing area and the
; error report 'Statement Lost' is given instead of 'OK'.
; - Steven Vickers, 1984.

;; GO-TO
L1E67:  CALL    L1E99            ; routine FIND-INT2 puts operand in BC
        LD      H,B              ; transfer line
        LD      L,C              ; number to HL.
        LD      D,$00            ; set statement to 0 - first.
        LD      A,H              ; compare high byte only
        CP      $F0              ; to $F0 i.e. 61439 in full.
        JR      NC,L1E9F         ; forward to REPORT-B if above.

; This call entry point is used to update the system variables e.g. by RETURN.

;; GO-TO-2
L1E73:  LD      ($5C42),HL       ; save line number in NEWPPC
        LD      (IY+$0A),D       ; and statement in NSPPC
        RET                      ; to STMT-RET (or GO-SUB command)

; ------------------
; Handle OUT command
; ------------------
; Syntax has been checked and the two comma-separated values are on the
; calculator stack.

;; OUT
L1E7A:  CALL    L1E85            ; routine TWO-PARAM fetches values
                                 ; to BC and A.
        OUT     (C),A            ; perform the operation.
        RET                      ; return to STMT-RET.

; -------------------
; Handle POKE command
; -------------------
; This routine alters a single byte in the 64K address space.
; Happily no check is made as to whether ROM or RAM is addressed.
; Sinclair BASIC requires no poking of system variables.

;; POKE
L1E80:  CALL    L1E85            ; routine TWO-PARAM fetches values
                                 ; to BC and A.
        LD      (BC),A           ; load memory location with A.
        RET                      ; return to STMT-RET.

; ------------------------------------
; Fetch two  parameters from calculator stack
```

```
; -----------------------------------
; This routine fetches a byte and word from the calculator stack
; producing an error if either is out of range.

;; TWO-PARAM
L1E85:  CALL    L2DD5           ; routine FP-TO-A
        JR      C,L1E9F         ; forward to REPORT-B if overflow occurred

        JR      Z,L1E8E         ; forward to TWO-P-1 if positive

        NEG                     ; negative numbers are made positive

;; TWO-P-1
L1E8E:  PUSH    AF              ; save the value
        CALL    L1E99           ; routine FIND-INT2 gets integer to BC
        POP     AF              ; restore the value
        RET                     ; return

; -------------
; Find integers
; -------------
; The first of these routines fetches a 8-bit integer (range 0-255) from the
; calculator stack to the accumulator and is used for colours, streams,
; durations and coordinates.
; The second routine fetches 16-bit integers to the BC register pair
; and is used to fetch command and function arguments involving line numbers
; or memory addresses and also array subscripts and tab arguments.
; ->

;; FIND-INT1
L1E94:  CALL    L2DD5           ; routine FP-TO-A
        JR      L1E9C           ; forward to FIND-I-1 for common exit routine.

; ---

; ->

;; FIND-INT2
L1E99:  CALL    L2DA2           ; routine FP-TO-BC

;; FIND-I-1
L1E9C:  JR      C,L1E9F         ; to REPORT-Bb with overflow.

        RET     Z               ; return if positive.


;; REPORT-Bb
L1E9F:  RST     08H             ; ERROR-1
        DEFB    $0A             ; Error Report: Integer out of range

; ------------------
; Handle RUN command
; ------------------
; This command runs a program starting at an optional line.
; It performs a 'RESTORE 0' then CLEAR

;; RUN
L1EA1:  CALL    L1E67           ; routine GO-TO puts line number in
                                ; system variables.
        LD      BC,$0000        ; prepare to set DATADD to first line.
        CALL    L1E45           ; routine REST-RUN does the 'restore'.
                                ; Note BC still holds zero.
        JR      L1EAF           ; forward to CLEAR-RUN to clear variables
                                ; without disturbing RAMTOP and
```

```
                                ; exit indirectly to STMT-RET

; --------------------
; Handle CLEAR command
; --------------------
; This command reclaims the space used by the variables.
; It also clears the screen and the GO SUB stack.
; With an integer expression, it sets the uppermost memory
; address within the BASIC system.
; "Contrary to the manual, CLEAR doesn't execute a RESTORE" -
; Steven Vickers, Pitman Pocket Guide to the Spectrum, 1984.

;; CLEAR
L1EAC:  CALL    L1E99           ; routine FIND-INT2 fetches to BC.

;; CLEAR-RUN
L1EAF:  LD      A,B             ; test for
        OR      C               ; zero.
        JR      NZ,L1EB7        ; skip to CLEAR-1 if not zero.

        LD      BC,($5CB2)      ; use the existing value of RAMTOP if zero.

;; CLEAR-1
L1EB7:  PUSH    BC              ; save ramtop value.

        LD      DE,($5C4B)      ; fetch VARS
        LD      HL,($5C59)      ; fetch E_LINE
        DEC     HL              ; adjust to point at variables end-marker.
        CALL    L19E5           ; routine RECLAIM-1 reclaims the space used by
                                ; the variables.

        CALL    L0D6B           ; routine CLS to clear screen.

        LD      HL,($5C65)      ; fetch STKEND the start of free memory.
        LD      DE,$0032        ; allow for another 50 bytes.
        ADD     HL,DE           ; add the overhead to HL.

        POP     DE              ; restore the ramtop value.
        SBC     HL,DE           ; if HL is greater than the value then jump
        JR      NC,L1EDA        ; forward to REPORT-M
                                ; 'RAMTOP no good'

        LD      HL,($5CB4)      ; now P-RAMT ($7FFF on 16K RAM machine)
        AND     A               ; exact this time.
        SBC     HL,DE           ; new ramtop must be lower or the same.
        JR      NC,L1EDC        ; skip to CLEAR-2 if in actual RAM.

;; REPORT-M
L1EDA:  RST     08H             ; ERROR-1
        DEFB    $15             ; Error Report: RAMTOP no good

;; CLEAR-2
L1EDC:  EX      DE,HL           ; transfer ramtop value to HL.
        LD      ($5CB2),HL      ; update system variable RAMTOP.
        POP     DE              ; pop the return address STMT-RET.
        POP     BC              ; pop the Error Address.
        LD      (HL),$3E        ; now put the GO SUB end-marker at RAMTOP.
        DEC     HL              ; leave a location beneath it.
        LD      SP,HL           ; initialize the machine stack pointer.
        PUSH    BC              ; push the error address.
        LD      ($5C3D),SP      ; make ERR_SP point to location.
        EX      DE,HL           ; put STMT-RET in HL.
        JP      (HL)            ; and go there directly.
```

```
; ---------------------
; Handle GO SUB command
; ---------------------
; The GO SUB command diverts BASIC control to a new line number
; in a very similar manner to GO TO but
; the current line number and current statement + 1
; are placed on the GO SUB stack as a RETURN point.

;; GO-SUB
L1EED:  POP     DE              ; drop the address STMT-RET
        LD      H,(IY+$0D)      ; fetch statement from SUBPPC and
        INC     H               ; increment it
        EX      (SP),HL         ; swap - error address to HL,
                                ; H (statement) at top of stack,
                                ; L (unimportant) beneath.
        INC     SP              ; adjust to overwrite unimportant byte
        LD      BC,($5C45)      ; fetch the current line number from PPC
        PUSH    BC              ; and PUSH onto GO SUB stack.
                                ; the empty machine-stack can be rebuilt
        PUSH    HL              ; push the error address.
        LD      ($5C3D),SP      ; make system variable ERR_SP point to it.
        PUSH    DE              ; push the address STMT-RET.
        CALL    L1E67           ; call routine GO-TO to update the system
                                ; variables NEWPPC and NSPPC.
                                ; then make an indirect exit to STMT-RET via
        LD      BC,$0014        ; a 20-byte overhead memory check.

; ----------------------
; Check available memory
; ----------------------
; This routine is used on many occasions when extending a dynamic area
; upwards or the GO SUB stack downwards.

;; TEST-ROOM
L1F05:  LD      HL,($5C65)      ; fetch STKEND
        ADD     HL,BC           ; add the supplied test value
        JR      C,L1F15         ; forward to REPORT-4 if over $FFFF

        EX      DE,HL           ; was less so transfer to DE
        LD      HL,$0050        ; test against another 80 bytes
        ADD     HL,DE           ; anyway
        JR      C,L1F15         ; forward to REPORT-4 if this passes $FFFF

        SBC     HL,SP           ; if less than the machine stack pointer
        RET     C               ; then return - OK.

;; REPORT-4
L1F15:  LD      L,$03           ; prepare 'Out of Memory'
        JP      L0055           ; jump back to ERROR-3 at $0055
                                ; Note. this error can't be trapped at $0008

; ------------------------------
; THE 'FREE MEMORY' USER ROUTINE
; ------------------------------
; This routine is not used by the ROM but allows users to evaluate
; approximate free memory with PRINT 65536 - USR 7962.

;; free-mem
L1F1A:  LD      BC,$0000        ; allow no overhead.

        CALL    L1F05           ; routine TEST-ROOM.

        LD      B,H             ; transfer the result
        LD      C,L             ; to the BC register.
```

```
        RET                     ; the USR function returns value of BC.

; --------------------
; THE 'RETURN' COMMAND
; --------------------
; As with any command, there are two values on the machine stack at the time
; it is invoked.  The machine stack is below the GOSUB stack.  Both grow
; downwards, the machine stack by two bytes, the GOSUB stack by 3 bytes.
; The highest location is a statement byte followed by a two-byte line number.

;; RETURN
L1F23:  POP     BC              ; drop the address STMT-RET.
        POP     HL              ; now the error address.
        POP     DE              ; now a possible BASIC return line.
        LD      A,D             ; the high byte $00 - $27 is
        CP      $3E             ; compared with the traditional end-marker $3E.
        JR      Z,L1F36         ; forward to REPORT-7 with a match.
                                ; 'RETURN without GOSUB'

; It was not the end-marker so a single statement byte remains at the base of
; the calculator stack. It can't be popped off.

        DEC     SP              ; adjust stack pointer to create room for two
                                ; bytes.
        EX      (SP),HL         ; statement to H, error address to base of
                                ; new machine stack.
        EX      DE,HL           ; statement to D,  BASIC line number to HL.
        LD      ($5C3D),SP      ; adjust ERR_SP to point to new stack pointer
        PUSH    BC              ; now re-stack the address STMT-RET
        JP      L1E73           ; to GO-TO-2 to update statement and line
                                ; system variables and exit indirectly to the
                                ; address just pushed on stack.

; ---

;; REPORT-7
L1F36:  PUSH    DE              ; replace the end-marker.
        PUSH    HL              ; now restore the error address
                                ; as will be required in a few clock cycles.

        RST     08H             ; ERROR-1
        DEFB    $06             ; Error Report: RETURN without GOSUB

; --------------------
; Handle PAUSE command
; --------------------
; The pause command takes as its parameter the number of interrupts
; for which to wait. PAUSE 50 pauses for about a second.
; PAUSE 0 pauses indefinitely.
; Both forms can be finished by pressing a key.

;; PAUSE
L1F3A:  CALL    L1E99           ; routine FIND-INT2 puts value in BC

;; PAUSE-1
L1F3D:  HALT                    ; wait for interrupt.
        DEC     BC              ; decrease counter.
        LD      A,B             ; test if
        OR      C               ; result is zero.
        JR      Z,L1F4F         ; forward to PAUSE-END if so.

        LD      A,B             ; test if
        AND     C               ; now $FFFF
        INC     A               ; that is, initially zero.
```

```
        JR      NZ,L1F49        ; skip forward to PAUSE-2 if not.

        INC     BC              ; restore counter to zero.

;; PAUSE-2
L1F49:  BIT     5,(IY+$01)      ; test FLAGS - has a new key been pressed ?
        JR      Z,L1F3D         ; back to PAUSE-1 if not.

;; PAUSE-END
L1F4F:  RES     5,(IY+$01)      ; update FLAGS - signal no new key
        RET                     ; and return.

; -------------------
; Check for BREAK key
; -------------------
; This routine is called from COPY-LINE, when interrupts are disabled,
; to test if BREAK (SHIFT - SPACE) is being pressed.
; It is also called at STMT-RET after every statement.

;; BREAK-KEY
L1F54:  LD      A,$7F           ; Input address: $7FFE
        IN      A,($FE)         ; read lower right keys
        RRA                     ; rotate bit 0 - SPACE
        RET     C               ; return if not reset

        LD      A,$FE           ; Input address: $FEFE
        IN      A,($FE)         ; read lower left keys
        RRA                     ; rotate bit 0 - SHIFT
        RET                     ; carry will be set if not pressed.
                                ; return with no carry if both keys
                                ; pressed.

; ---------------------
; Handle DEF FN command
; ---------------------
; e.g. DEF FN r$(a$,a) = a$(a TO )
; this 'command' is ignored in runtime but has its syntax checked
; during line-entry.

;; DEF-FN
L1F60:  CALL    L2530           ; routine SYNTAX-Z
        JR      Z,L1F6A         ; forward to DEF-FN-1 if parsing

        LD      A,$CE           ; else load A with 'DEF FN' and
        JP      L1E39           ; jump back to PASS-BY

; ---

; continue here if checking syntax.

;; DEF-FN-1
L1F6A:  SET      6,(IY+$01)     ; set FLAGS  - Assume numeric result
        CALL    L2C8D           ; call routine ALPHA
        JR      NC,L1F89        ; if not then to DEF-FN-4 to jump to
                                ; 'Nonsense in BASIC'


        RST     20H             ; NEXT-CHAR
        CP      $24             ; is it '$' ?
        JR      NZ,L1F7D        ; to DEF-FN-2 if not as numeric.

        RES     6,(IY+$01)      ; set FLAGS  - Signal string result

        RST     20H             ; get NEXT-CHAR
```

```
;; DEF-FN-2
L1F7D:  CP      $28             ; is it '(' ?
        JR      NZ,L1FBD        ; to DEF-FN-7 'Nonsense in BASIC'


        RST     20H             ; NEXT-CHAR
        CP      $29             ; is it ')' ?
        JR      Z,L1FA6         ; to DEF-FN-6 if null argument

;; DEF-FN-3
L1F86:  CALL    L2C8D           ; routine ALPHA checks that it is the expected
                                ; alphabetic character.

;; DEF-FN-4
L1F89:  JP      NC,L1C8A        ; to REPORT-C  if not
                                ; 'Nonsense in BASIC'.

        EX      DE,HL           ; save pointer in DE

        RST     20H             ; NEXT-CHAR re-initializes HL from CH_ADD
                                ; and advances.
        CP      $24             ; '$' ? is it a string argument.
        JR      NZ,L1F94        ; forward to DEF-FN-5 if not.

        EX      DE,HL           ; save pointer to '$' in DE

        RST     20H             ; NEXT-CHAR re-initializes HL and advances

;; DEF-FN-5
L1F94:  EX      DE,HL           ; bring back pointer.
        LD      BC,$0006        ; the function requires six hidden bytes for
                                ; each parameter passed.
                                ; The first byte will be $0E
                                ; then 5-byte numeric value
                                ; or 5-byte string pointer.

        CALL    L1655           ; routine MAKE-ROOM creates space in program
                                ; area.

        INC     HL              ; adjust HL (set by LDDR)
        INC     HL              ; to point to first location.
        LD      (HL),$0E        ; insert the 'hidden' marker.

; Note. these invisible storage locations hold nothing meaningful for the
; moment. They will be used every time the corresponding function is
; evaluated in runtime.
; Now consider the following character fetched earlier.

        CP      $2C             ; is it ',' ? (more than one parameter)
        JR      NZ,L1FA6        ; to DEF-FN-6 if not


        RST     20H             ; else NEXT-CHAR
        JR      L1F86           ; and back to DEF-FN-3

; ---

;; DEF-FN-6
L1FA6:  CP      $29             ; should close with a ')'
        JR      NZ,L1FBD        ; to DEF-FN-7 if not
                                ; 'Nonsense in BASIC'
```

```
        RST     20H             ; get NEXT-CHAR
        CP      $3D             ; is it '=' ?
        JR      NZ,L1FBD        ; to DEF-FN-7 if not 'Nonsense...'


        RST     20H             ; address NEXT-CHAR
        LD      A,($5C3B)       ; get FLAGS which has been set above
        PUSH    AF              ; and preserve

        CALL    L24FB           ; routine SCANNING checks syntax of expression
                                ; and also sets flags.

        POP     AF              ; restore previous flags
        XOR     (IY+$01)        ; xor with FLAGS - bit 6 should be same
                                ; therefore will be reset.
        AND     $40             ; isolate bit 6.

;; DEF-FN-7
L1FBD:  JP      NZ,L1C8A        ; jump back to REPORT-C if the expected result
                                ; is not the same type.
                                ; 'Nonsense in BASIC'

        CALL    L1BEE           ; routine CHECK-END will return early if
                                ; at end of statement and move onto next
                                ; else produce error report. >>>

                                ; There will be no return to here.

; -------------------------------
; Returning early from subroutine
; -------------------------------
; All routines are capable of being run in two modes - syntax checking mode
; and runtime mode.  This routine is called often to allow a routine to return
; early if checking syntax.

;; UNSTACK-Z
L1FC3:  CALL    L2530           ; routine SYNTAX-Z sets zero flag if syntax
                                ; is being checked.

        POP     HL              ; drop the return address.
        RET      Z              ; return to previous call in chain if checking
                                ; syntax.

        JP      (HL)            ; jump to return address as BASIC program is
                                ; actually running.

; ---------------------
; Handle LPRINT command
; ---------------------
; A simple form of 'PRINT #3' although it can output to 16 streams.
; Probably for compatibility with other BASICs particularly ZX81 BASIC.
; An extra UDG might have been better.

;; LPRINT
L1FC9:  LD      A,$03           ; the printer channel
        JR      L1FCF           ; forward to PRINT-1

; ---------------------
; Handle PRINT commands
; ---------------------
; The Spectrum's main stream output command.
; The default stream is stream 2 which is normally the upper screen
; of the computer. However the stream can be altered in range 0 - 15.
```

```
;; PRINT
L1FCD:  LD      A,$02           ; the stream for the upper screen.

; The LPRINT command joins here.

;; PRINT-1
L1FCF:  CALL    L2530           ; routine SYNTAX-Z checks if program running
        CALL    NZ,L1601        ; routine CHAN-OPEN if so
        CALL    L0D4D           ; routine TEMPS sets temporary colours.
        CALL    L1FDF           ; routine PRINT-2 - the actual item
        CALL    L1BEE           ; routine CHECK-END gives error if not at end
                                ; of statement
        RET                     ; and return >>>

; ------------------------------------
; this subroutine is called from above
; and also from INPUT.

;; PRINT-2
L1FDF:  RST     18H             ; GET-CHAR gets printable character
        CALL    L2045           ; routine PR-END-Z checks if more printing
        JR      Z,L1FF2         ; to PRINT-4 if not     e.g. just 'PRINT :'

; This tight loop deals with combinations of positional controls and
; print items. An early return can be made from within the loop
; if the end of a print sequence is reached.

;; PRINT-3
L1FE5:  CALL    L204E           ; routine PR-POSN-1 returns zero if more
                                ; but returns early at this point if
                                ; at end of statement!
                                ;
        JR      Z,L1FE5         ; to PRINT-3 if consecutive positioners

        CALL    L1FFC           ; routine PR-ITEM-1 deals with strings etc.
        CALL    L204E           ; routine PR-POSN-1 for more position codes
        JR      Z,L1FE5         ; loop back to PRINT-3 if so

;; PRINT-4
L1FF2:  CP      $29             ; return now if this is ')' from input-item.
                                ; (see INPUT.)
        RET     Z               ; or continue and print carriage return in
                                ; runtime

; ---------------------
; Print carriage return
; ---------------------
; This routine which continues from above prints a carriage return
; in run-time. It is also called once from PRINT-POSN.

;; PRINT-CR
L1FF5:  CALL    L1FC3           ; routine UNSTACK-Z

        LD      A,$0D           ; prepare a carriage return

        RST     10H             ; PRINT-A
        RET                     ; return


; -----------
; Print items
; -----------
; This routine deals with print items as in
; PRINT AT 10,0;"The value of A is ";a
```

```
; It returns once a single item has been dealt with as it is part
; of a tight loop that considers sequences of positional and print items

;; PR-ITEM-1
L1FFC:  RST     18H             ; GET-CHAR
        CP      $AC             ; is character 'AT' ?
        JR      NZ,L200E        ; forward to PR-ITEM-2 if not.

        CALL    L1C79           ; routine NEXT-2NUM  check for two comma
                                ; separated numbers placing them on the
                                ; calculator stack in runtime.
        CALL    L1FC3           ; routine UNSTACK-Z quits if checking syntax.

        CALL    L2307           ; routine STK-TO-BC get the numbers in B and C.
        LD      A,$16           ; prepare the 'at' control.
        JR      L201E           ; forward to PR-AT-TAB to print the sequence.

; ---

;; PR-ITEM-2
L200E:  CP      $AD             ; is character 'TAB' ?
        JR      NZ,L2024        ; to PR-ITEM-3 if not


        RST     20H             ; NEXT-CHAR to address next character
        CALL    L1C82           ; routine EXPT-1NUM
        CALL    L1FC3           ; routine UNSTACK-Z quits if checking syntax.

        CALL    L1E99           ; routine FIND-INT2 puts integer in BC.
        LD      A,$17           ; prepare the 'tab' control.

;; PR-AT-TAB
L201E:  RST     10H             ; PRINT-A outputs the control

        LD      A,C             ; first value to A
        RST     10H             ; PRINT-A outputs it.

        LD      A,B             ; second value
        RST     10H             ; PRINT-A

        RET                     ; return - item finished >>>

; ---

; Now consider paper 2; #2; a$

;; PR-ITEM-3
L2024:  CALL    L21F2           ; routine CO-TEMP-3 will print any colour
        RET     NC              ; items - return if success.

        CALL    L2070           ; routine STR-ALTER considers new stream
        RET     NC              ; return if altered.

        CALL    L24FB           ; routine SCANNING now to evaluate expression
        CALL    L1FC3           ; routine UNSTACK-Z if not runtime.

        BIT     6,(IY+$01)      ; test FLAGS  - Numeric or string result ?
        CALL    Z,L2BF1         ; routine STK-FETCH if string.
                                ; note no flags affected.
        JP      NZ,L2DE3        ; to PRINT-FP to print if numeric >>>

; It was a string expression - start in DE, length in BC
; Now enter a loop to print it
```

```
;; PR-STRING
L203C:  LD      A,B             ; this tests if the
        OR      C               ; length is zero and sets flag accordingly.
        DEC     BC              ; this doesn't but decrements counter.
        RET     Z               ; return if zero.

        LD      A,(DE)          ; fetch character.
        INC     DE              ; address next location.

        RST     10H             ; PRINT-A.

        JR      L203C           ; loop back to PR-STRING.

; ---------------
; End of printing
; ---------------
; This subroutine returns zero if no further printing is required
; in the current statement.
; The first terminator is found in  escaped input items only,
; the others in print_items.

;; PR-END-Z
L2045:  CP      $29             ; is character a ')' ?
        RET     Z               ; return if so -        e.g. INPUT (p$); a$

;; PR-ST-END
L2048:  CP      $0D             ; is it a carriage return ?
        RET     Z               ; return also -         e.g. PRINT a

        CP      $3A             ; is character a ':' ?
        RET                     ; return - zero flag will be set if so.
                                ;                       e.g. PRINT a :

; --------------
; Print position
; --------------
; This routine considers a single positional character ';', ',', '''

;; PR-POSN-1
L204E:  RST     18H             ; GET-CHAR
        CP      $3B             ; is it ';' ?
                                ; i.e. print from last position.
        JR      Z,L2067         ; forward to PR-POSN-3 if so.
                                ; i.e. do nothing.

        CP      $2C             ; is it ',' ?
                                ; i.e. print at next tabstop.
        JR      NZ,L2061        ; forward to PR-POSN-2 if anything else.

        CALL    L2530           ; routine SYNTAX-Z
        JR      Z,L2067         ; forward to PR-POSN-3 if checking syntax.

        LD      A,$06           ; prepare the 'comma' control character.

        RST     10H             ; PRINT-A  outputs to current channel in
                                ; run-time.

        JR      L2067           ; skip to PR-POSN-3.

; ---

; check for newline.

;; PR-POSN-2
```

```
L2061:  CP      $27             ; is character a "'" ? (newline)
        RET     NZ              ; return if no match              >>>

        CALL    L1FF5           ; routine PRINT-CR outputs a carriage return
                                ; in runtime only.

;; PR-POSN-3
L2067:  RST     20H             ; NEXT-CHAR to A.
        CALL    L2045           ; routine PR-END-Z checks if at end.
        JR      NZ,L206E        ; to PR-POSN-4 if not.

        POP     BC              ; drop return address if at end.

;; PR-POSN-4
L206E:  CP      A               ; reset the zero flag.
        RET                     ; and return to loop or quit.

; ------------
; Alter stream
; ------------
; This routine is called from PRINT ITEMS above, and also LIST as in
; LIST #15

;; STR-ALTER
L2070:  CP      $23             ; is character '#' ?
        SCF                     ; set carry flag.
        RET     NZ              ; return if no match.


        RST      20H            ; NEXT-CHAR
        CALL    L1C82           ; routine EXPT-1NUM gets stream number
        AND     A               ; prepare to exit early with carry reset
        CALL    L1FC3           ; routine UNSTACK-Z exits early if parsing
        CALL    L1E94           ; routine FIND-INT1 gets number off stack
        CP      $10             ; must be range 0 - 15 decimal.
        JP      NC,L160E        ; jump back to REPORT-Oa if not
                                ; 'Invalid stream'.

        CALL    L1601           ; routine CHAN-OPEN
        AND     A               ; clear carry - signal item dealt with.
        RET                     ; return

; --------------------
; Handle INPUT command
; --------------------
; This command
;

;; INPUT
L2089:  CALL    L2530           ; routine SYNTAX-Z to check if in runtime.
        JR      Z,L2096         ; forward to INPUT-1 if checking syntax.

        LD      A,$01           ; select channel 'K' the keyboard for input.
        CALL    L1601           ; routine CHAN-OPEN opens the channel and sets
                                ; bit 0 of TV_FLAG.
        CALL    L0D6E           ; routine CLS-LOWER clears the lower screen
                                ; and sets DF_SZ to two and TV_FLAG to $01.

;; INPUT-1
L2096:  LD      (IY+$02),$01    ; update TV_FLAG - signal lower screen in use
                                ; ensuring that the correct set of system
                                ; variables are updated and that the border
                                ; colour is used.
```

```
;   Note. The Complete Spectrum ROM Disassembly incorrectly names DF-SZ as the
;   system variable that is updated above and if, as some have done, you make
;   this unnecessary alteration then there will be two blank lines between the
;   lower screen and the upper screen areas which will also scroll wrongly.

        CALL    L20C1           ; routine IN-ITEM-1 to handle the input.

        CALL    L1BEE           ; routine CHECK-END will make an early exit
                                ; if checking syntax. >>>

; keyboard input has been made and it remains to adjust the upper
; screen in case the lower two lines have been extended upwards.

        LD      BC,($5C88)      ; fetch S_POSN current line/column of
                                ; the upper screen.
        LD      A,($5C6B)       ; fetch DF_SZ the display file size of
                                ; the lower screen.
        CP      B               ; test that lower screen does not overlap
        JR      C,L20AD         ; forward to INPUT-2 if not.

; the two screens overlap so adjust upper screen.

        LD      C,$21           ; set column of upper screen to leftmost.
        LD      B,A             ; and line to one above lower screen.
                                ; continue forward to update upper screen
                                ; print position.

;; INPUT-2
L20AD:  LD      ($5C88),BC      ; set S_POSN update upper screen line/column.
        LD      A,$19           ; subtract from twenty five
        SUB     B               ; the new line number.
        LD      ($5C8C),A       ; and place result in SCR_CT - scroll count.
        RES     0,(IY+$02)      ; update TV_FLAG - signal main screen in use.
        CALL    L0DD9           ; routine CL-SET sets the print position
                                ; system variables for the upper screen.
        JP      L0D6E           ; jump back to CLS-LOWER and make
                                ; an indirect exit >>.

; ---------------------
; INPUT ITEM subroutine
; ---------------------
; This subroutine deals with the input items and print items.
; from  the current input channel.
; It is only called from the above INPUT routine but was obviously
; once called from somewhere else in another context.

;; IN-ITEM-1
L20C1:  CALL    L204E           ; routine PR-POSN-1 deals with a single
                                ; position item at each call.
        JR      Z,L20C1         ; back to IN-ITEM-1 until no more in a
                                ; sequence.

        CP      $28             ; is character '(' ?
        JR      NZ,L20D8        ; forward to IN-ITEM-2 if not.

; any variables within braces will be treated as part, or all, of the prompt
; instead of being used as destination variables.

        RST     20H             ; NEXT-CHAR
        CALL    L1FDF           ; routine PRINT-2 to output the dynamic
                                ; prompt.

        RST     18H             ; GET-CHAR
        CP      $29             ; is character a matching ')' ?
```

```
        JP      NZ,L1C8A        ; jump back to REPORT-C if not.
                                ; 'Nonsense in BASIC'.

        RST     20H             ; NEXT-CHAR
        JP      L21B2           ; forward to IN-NEXT-2

; ---

;; IN-ITEM-2
L20D8:  CP      $CA             ; is the character the token 'LINE' ?
        JR      NZ,L20ED        ; forward to IN-ITEM-3 if not.

        RST     20H             ; NEXT-CHAR - variable must come next.
        CALL    L1C1F           ; routine CLASS-01 returns destination
                                ; address of variable to be assigned.
                                ; or generates an error if no variable
                                ; at this position.

        SET     7,(IY+$37)      ; update FLAGX  - signal handling INPUT LINE
        BIT     6,(IY+$01)      ; test FLAGS  - numeric or string result ?
        JP      NZ,L1C8A        ; jump back to REPORT-C if not string
                                ; 'Nonsense in BASIC'.

        JR      L20FA           ; forward to IN-PROMPT to set up workspace.

; ---

; the jump was here for other variables.

;; IN-ITEM-3
L20ED:  CALL     L2C8D          ; routine ALPHA checks if character is
                                ; a suitable variable name.
        JP      NC,L21AF        ; forward to IN-NEXT-1 if not

        CALL    L1C1F           ; routine CLASS-01 returns destination
                                ; address of variable to be assigned.
        RES     7,(IY+$37)      ; update FLAGX  - signal not INPUT LINE.

;; IN-PROMPT
L20FA:  CALL    L2530           ; routine SYNTAX-Z
        JP      Z,L21B2         ; forward to IN-NEXT-2 if checking syntax.

        CALL    L16BF           ; routine SET-WORK clears workspace.
        LD      HL,$5C71        ; point to system variable FLAGX
        RES     6,(HL)          ; signal string result.
        SET     5,(HL)          ; signal in Input Mode for editor.
        LD      BC,$0001        ; initialize space required to one for
                                ; the carriage return.
        BIT     7,(HL)          ; test FLAGX - INPUT LINE in use ?
        JR      NZ,L211C        ; forward to IN-PR-2 if so as that is
                                ; all the space that is required.

        LD      A,($5C3B)       ; load accumulator from FLAGS
        AND     $40             ; mask to test BIT 6 of FLAGS and clear
                                ; the other bits in A.
                                ; numeric result expected ?
        JR      NZ,L211A        ; forward to IN-PR-1 if so

        LD      C,$03           ; increase space to three bytes for the
                                ; pair of surrounding quotes.

;; IN-PR-1
L211A:  OR      (HL)            ; if numeric result, set bit 6 of FLAGX.
        LD      (HL),A          ; and update system variable
```

```
;; IN-PR-2
L211C:  RST     30H             ; BC-SPACES opens 1 or 3 bytes in workspace
        LD      (HL),$0D        ; insert carriage return at last new location.
        LD      A,C             ; fetch the length, one or three.
        RRCA                    ; lose bit 0.
        RRCA                    ; test if quotes required.
        JR      NC,L2129        ; forward to IN-PR-3 if not.

        LD      A,$22           ; load the '"' character
        LD      (DE),A          ; place quote in first new location at DE.
        DEC     HL              ; decrease HL - from carriage return.
        LD      (HL),A          ; and place a quote in second location.

;; IN-PR-3
L2129:  LD      ($5C5B),HL      ; set keyboard cursor K_CUR to HL
        BIT     7,(IY+$37)      ; test FLAGX  - is this INPUT LINE ??
        JR      NZ,L215E        ; forward to IN-VAR-3 if so as input will
                                ; be accepted without checking its syntax.

        LD      HL,($5C5D)      ; fetch CH_ADD
        PUSH    HL              ; and save on stack.
        LD      HL,($5C3D)      ; fetch ERR_SP
        PUSH    HL              ; and save on stack

;; IN-VAR-1
L213A:  LD      HL,L213A        ; address: IN-VAR-1 - this address
        PUSH    HL              ; is saved on stack to handle errors.
        BIT     4,(IY+$30)      ; test FLAGS2  - is K channel in use ?
        JR      Z,L2148         ; forward to IN-VAR-2 if not using the
                                ; keyboard for input. (??)

        LD      ($5C3D),SP      ; set ERR_SP to point to IN-VAR-1 on stack.

;; IN-VAR-2
L2148:  LD      HL,($5C61)      ; set HL to WORKSP - start of workspace.
        CALL    L11A7           ; routine REMOVE-FP removes floating point
                                ; forms when looping in error condition.
        LD      (IY+$00),$FF    ; set ERR_NR to 'OK' cancelling the error.
                                ; but X_PTR causes flashing error marker
                                ; to be displayed at each call to the editor.
        CALL    L0F2C           ; routine EDITOR allows input to be entered
                                ; or corrected if this is second time around.

; if we pass to next then there are no system errors

        RES     7,(IY+$01)      ; update FLAGS  - signal checking syntax
        CALL    L21B9           ; routine IN-ASSIGN checks syntax using
                                ; the VAL-FET-2 and powerful SCANNING routines.
                                ; any syntax error and its back to IN-VAR-1.
                                ; but with the flashing error marker showing
                                ; where the error is.
                                ; Note. the syntax of string input has to be
                                ; checked as the user may have removed the
                                ; bounding quotes or escaped them as with
                                ; "hat" + "stand" for example.
; proceed if syntax passed.

        JR      L2161           ; jump forward to IN-VAR-4

; ---

; the jump was to here when using INPUT LINE.
```

```
;; IN-VAR-3
L215E:  CALL    L0F2C           ; routine EDITOR is called for input

; when ENTER received rejoin other route but with no syntax check.

; INPUT and INPUT LINE converge here.

;; IN-VAR-4
L2161:  LD      (IY+$22),$00    ; set K_CUR_hi to a low value so that the cursor
                                ; no longer appears in the input line.

        CALL    L21D6           ; routine IN-CHAN-K tests if the keyboard
                                ; is being used for input.
        JR      NZ,L2174        ; forward to IN-VAR-5 if using another input
                                ; channel.

; continue here if using the keyboard.

        CALL    L111D           ; routine ED-COPY overprints the edit line
                                ; to the lower screen. The only visible
                                ; affect is that the cursor disappears.
                                ; if you're inputting more than one item in
                                ; a statement then that becomes apparent.

        LD      BC,($5C82)      ; fetch line and column from ECHO_E
        CALL    L0DD9           ; routine CL-SET sets S-POSNL to those
                                ; values.

; if using another input channel rejoin here.

;; IN-VAR-5
L2174:  LD      HL,$5C71        ; point HL to FLAGX
        RES     5,(HL)          ; signal not in input mode
        BIT     7,(HL)          ; is this INPUT LINE ?
        RES     7,(HL)          ; cancel the bit anyway.
        JR      NZ,L219B        ; forward to IN-VAR-6 if INPUT LINE.

        POP     HL              ; drop the looping address
        POP     HL              ; drop the address of previous
                                ; error handler.
        LD      ($5C3D),HL      ; set ERR_SP to point to it.
        POP     HL              ; drop original CH_ADD which points to
                                ; INPUT command in BASIC line.
        LD      ($5C5F),HL      ; save in X_PTR while input is assigned.
        SET     7,(IY+$01)      ; update FLAGS - Signal running program
        CALL    L21B9           ; routine IN-ASSIGN is called again
                                ; this time the variable will be assigned
                                ; the input value without error.
                                ; Note. the previous example now
                                ; becomes "hatstand"

        LD      HL,($5C5F)      ; fetch stored CH_ADD value from X_PTR.
        LD      (IY+$26),$00    ; set X_PTR_hi so that iy is no longer relevant.
        LD      ($5C5D),HL      ; put restored value back in CH_ADD
        JR      L21B2           ; forward to IN-NEXT-2 to see if anything
                                ; more in the INPUT list.

; ---

; the jump was to here with INPUT LINE only

;; IN-VAR-6
L219B:  LD      HL,($5C63)      ; STKBOT points to the end of the input.
        LD      DE,($5C61)      ; WORKSP points to the beginning.
```

```
        SCF                         ; prepare for true subtraction.
        SBC     HL,DE               ; subtract to get length
        LD      B,H                 ; transfer it to
        LD      C,L                 ; the BC register pair.
        CALL    L2AB2               ; routine STK-STO-$ stores parameters on
                                    ; the calculator stack.
        CALL    L2AFF               ; routine LET assigns it to destination.
        JR      L21B2               ; forward to IN-NEXT-2 as print items
                                    ; not allowed with INPUT LINE.
                                    ; Note. that "hat" + "stand" will, for
                                    ; example, be unchanged as also would
                                    ; 'PRINT "Iris was here"'.

; ---

; the jump was to here when ALPHA found more items while looking for
; a variable name.

;; IN-NEXT-1
L21AF:  CALL    L1FFC               ; routine PR-ITEM-1 considers further items.

;; IN-NEXT-2
L21B2:  CALL    L204E               ; routine PR-POSN-1 handles a position item.
        JP      Z,L20C1             ; jump back to IN-ITEM-1 if the zero flag
                                    ; indicates more items are present.

        RET                         ; return.

; ---------------------------
; INPUT ASSIGNMENT Subroutine
; ---------------------------
; This subroutine is called twice from the INPUT command when normal
; keyboard input is assigned. On the first occasion syntax is checked
; using SCANNING. The final call with the syntax flag reset is to make
; the assignment.

;; IN-ASSIGN
L21B9:  LD      HL,($5C61)          ; fetch WORKSP start of input
        LD      ($5C5D),HL          ; set CH_ADD to first character

        RST     18H                 ; GET-CHAR ignoring leading white-space.
        CP      $E2                 ; is it 'STOP'
        JR      Z,L21D0             ; forward to IN-STOP if so.

        LD      A,($5C71)           ; load accumulator from FLAGX
        CALL    L1C59               ; routine VAL-FET-2 makes assignment
                                    ; or goes through the motions if checking
                                    ; syntax. SCANNING is used.

        RST     18H                 ; GET-CHAR
        CP      $0D                 ; is it carriage return ?
        RET     Z                   ; return if so
                                    ; either syntax is OK
                                    ; or assignment has been made.

; if another character was found then raise an error.
; User doesn't see report but the flashing error marker
; appears in the lower screen.

;; REPORT-Cb
L21CE:  RST     08H                 ; ERROR-1
        DEFB    $0B                 ; Error Report: Nonsense in BASIC

;; IN-STOP
```

```
L21D0:  CALL    L2530           ; routine SYNTAX-Z (UNSTACK-Z?)
        RET     Z               ; return if checking syntax
                                ; as user wouldn't see error report.
                                ; but generate visible error report
                                ; on second invocation.

;; REPORT-H
L21D4:  RST     08H             ; ERROR-1
        DEFB    $10             ; Error Report: STOP in INPUT

; -----------------------------------
; THE 'TEST FOR CHANNEL K' SUBROUTINE
; -----------------------------------
;   This subroutine is called once from the keyboard INPUT command to check if
;   the input routine in use is the one for the keyboard.

;; IN-CHAN-K
L21D6:  LD      HL,($5C51)      ; fetch address of current channel CURCHL
        INC     HL              ;
        INC     HL              ; advance past
        INC     HL              ; input and
        INC     HL              ; output streams
        LD      A,(HL)          ; fetch the channel identifier.
        CP      $4B             ; test for 'K'
        RET                     ; return with zero set if keyboard is use.

; --------------------
; Colour Item Routines
; --------------------
;
; These routines have 3 entry points -
; 1) CO-TEMP-2 to handle a series of embedded Graphic colour items.
; 2) CO-TEMP-3 to handle a single embedded print colour item.
; 3) CO TEMP-4 to handle a colour command such as FLASH 1
;
; "Due to a bug, if you bring in a peripheral channel and later use a colour
;  statement, colour controls will be sent to it by mistake." - Steven Vickers
;  Pitman Pocket Guide, 1984.
;
; To be fair, this only applies if the last channel was other than 'K', 'S'
; or 'P', which are all that are supported by this ROM, but if that last
; channel was a microdrive file, network channel etc. then
; PAPER 6; CLS will not turn the screen yellow and
; CIRCLE INK 2; 128,88,50 will not draw a red circle.
;
; This bug does not apply to embedded PRINT items as it is quite permissible
; to mix stream altering commands and colour items.
; The fix therefore would be to ensure that CLASS-07 and CLASS-09 make
; channel 'S' the current channel when not checking syntax.
; -----------------------------------------------------------------

;; CO-TEMP-1
L21E1:  RST     20H             ; NEXT-CHAR

; -> Entry point from CLASS-09. Embedded Graphic colour items.
; e.g. PLOT INK 2; PAPER 8; 128,88
; Loops till all colour items output, finally addressing the coordinates.

;; CO-TEMP-2
L21E2:  CALL    L21F2           ; routine CO-TEMP-3 to output colour control.
        RET     C               ; return if nothing more to output. ->


        RST     18H             ; GET-CHAR
```

```
        CP      $2C             ; is it ',' separator ?
        JR      Z,L21E1         ; back if so to CO-TEMP-1

        CP      $3B             ; is it ';' separator ?
        JR      Z,L21E1         ; back to CO-TEMP-1 for more.

        JP      L1C8A           ; to REPORT-C (REPORT-Cb is within range)
                                ; 'Nonsense in BASIC'

; -------------------
; CO-TEMP-3
; -------------------
; -> this routine evaluates and outputs a colour control and parameter.
; It is called from above and also from PR-ITEM-3 to handle a single embedded
; print item e.g. PRINT PAPER 6; "Hi". In the latter case, the looping for
; multiple items is within the PR-ITEM routine.
; It is quite permissible to send these to any stream.

;; CO-TEMP-3
L21F2:  CP      $D9             ; is it 'INK' ?
        RET     C               ; return if less.

        CP      $DF             ; compare with 'OUT'
        CCF                     ; Complement Carry Flag
        RET     C               ; return if greater than 'OVER', $DE.

        PUSH    AF              ; save the colour token.

        RST     20H             ; address NEXT-CHAR
        POP     AF              ; restore token and continue.

; -> this entry point used by CLASS-07. e.g. the command PAPER 6.

;; CO-TEMP-4
L21FC:  SUB     $C9             ; reduce to control character $10 (INK)
                                ; thru $15 (OVER).
        PUSH    AF              ; save control.
        CALL    L1C82           ; routine EXPT-1NUM stacks addressed
                                ; parameter on calculator stack.
        POP     AF              ; restore control.
        AND     A               ; clear carry

        CALL    L1FC3           ; routine UNSTACK-Z returns if checking syntax.

        PUSH    AF              ; save again
        CALL    L1E94           ; routine FIND-INT1 fetches parameter to A.
        LD      D,A             ; transfer now to D
        POP     AF              ; restore control.

        RST     10H             ; PRINT-A outputs the control to current
                                ; channel.
        LD      A,D             ; transfer parameter to A.

        RST     10H             ; PRINT-A outputs parameter.
        RET                     ; return. ->

; -----------------------------------------------------------------------------
;
;          {fl}{br}{   paper   }{  ink    }    The temporary colour attributes
;           ___ ___ ___ ___ ___ ___ ___ ___    system variable.
; ATTR_T   |   |   |   |   |   |   |   |   |
;          |   |   |   |   |   |   |   |   |
; 23695    |___|___|___|___|___|___|___|___|
;            7   6   5   4   3   2   1   0
```

```
;
;
;          {fl}{br}{   paper   }{  ink    }    The temporary mask used for
;           ___ ___ ___ ___ ___ ___ ___ ___    transparent colours. Any bit
; MASK_T   |   |   |   |   |   |   |   |   |   that is 1 shows that the
;          |   |   |   |   |   |   |   |   |   corresponding attribute is
; 23696    |___|___|___|___|___|___|___|___|   taken not from ATTR-T but from
;            7   6   5   4   3   2   1   0     what is already on the screen.
;
;
;          {paper9 }{ ink9 }{ inv1 }{ over1}   The print flags. Even bits are
;           ___ ___ ___ ___ ___ ___ ___ ___    temporary flags. The odd bits
; P_FLAG   |   |   |   |   |   |   |   |   |   are the permanent flags.
;          | p | t | p | t | p | t | p | t |
; 23697    |___|___|___|___|___|___|___|___|
;            7   6   5   4   3   2   1   0
;
; -----------------------------------------------------------------------

; ------------------------------------
;  The colour system variable handler.
; ------------------------------------
; This is an exit branch from PO-1-OPER, PO-2-OPER
; A holds control $10 (INK) to $15 (OVER)
; D holds parameter 0-9 for ink/paper 0,1 or 8 for bright/flash,
; 0 or 1 for over/inverse.

;; CO-TEMP-5
L2211:  SUB     $11             ; reduce range $FF-$04
        ADC     A,$00           ; add in carry if INK
        JR      Z,L2234         ; forward to CO-TEMP-7 with INK and PAPER.

        SUB     $02             ; reduce range $FF-$02
        ADC     A,$00           ; add carry if FLASH
        JR      Z,L2273         ; forward to CO-TEMP-C with FLASH and BRIGHT.

        CP      $01             ; is it 'INVERSE' ?
        LD      A,D             ; fetch parameter for INVERSE/OVER
        LD      B,$01           ; prepare OVER mask setting bit 0.
        JR      NZ,L2228        ; forward to CO-TEMP-6 if OVER

        RLCA                    ; shift bit 0
        RLCA                    ; to bit 2
        LD      B,$04           ; set bit 2 of mask for inverse.

;; CO-TEMP-6
L2228:  LD      C,A             ; save the A
        LD      A,D             ; re-fetch parameter
        CP      $02             ; is it less than 2
        JR      NC,L2244        ; to REPORT-K if not 0 or 1.
                                ; 'Invalid colour'.

        LD      A,C             ; restore A
        LD      HL,$5C91        ; address system variable P_FLAG
        JR      L226C           ; forward to exit via routine CO-CHANGE

; ---

; the branch was here with INK/PAPER and carry set for INK.

;; CO-TEMP-7
L2234:  LD      A,D             ; fetch parameter
        LD      B,$07           ; set ink mask 00000111
        JR      C,L223E         ; forward to CO-TEMP-8 with INK
```

```
        RLCA                    ; shift bits 0-2
        RLCA                    ; to
        RLCA                    ; bits 3-5
        LD      B,$38           ; set paper mask 00111000

; both paper and ink rejoin here

;; CO-TEMP-8
L223E:  LD      C,A             ; value to C
        LD      A,D             ; fetch parameter
        CP      $0A             ; is it less than 10d ?
        JR      C,L2246         ; forward to CO-TEMP-9 if so.

; ink 10 etc. is not allowed.

;; REPORT-K
L2244:  RST     08H             ; ERROR-1
        DEFB    $13             ; Error Report: Invalid colour

;; CO-TEMP-9
L2246:  LD      HL,$5C8F        ; address system variable ATTR_T initially.
        CP      $08             ; compare with 8
        JR      C,L2258         ; forward to CO-TEMP-B with 0-7.

        LD      A,(HL)          ; fetch temporary attribute as no change.
        JR      Z,L2257         ; forward to CO-TEMP-A with INK/PAPER 8

; it is either ink 9 or paper 9 (contrasting)

        OR      B               ; or with mask to make white
        CPL                     ; make black and change other to dark
        AND     $24             ; 00100100
        JR      Z,L2257         ; forward to CO-TEMP-A if black and
                                ; originally light.

        LD      A,B             ; else just use the mask (white)

;; CO-TEMP-A
L2257:  LD      C,A             ; save A in C

;; CO-TEMP-B
L2258:  LD      A,C             ; load colour to A
        CALL    L226C           ; routine CO-CHANGE addressing ATTR-T

        LD      A,$07           ; put 7 in accumulator
        CP      D               ; compare with parameter
        SBC     A,A             ; $00 if 0-7, $FF if 8
        CALL    L226C           ; routine CO-CHANGE addressing MASK-T
                                ; mask returned in A.

; now consider P-FLAG.

        RLCA                    ; 01110000 or 00001110
        RLCA                    ; 11100000 or 00011100
        AND     $50             ; 01000000 or 00010000  (AND 01010000)
        LD      B,A             ; transfer to mask
        LD      A,$08           ; load A with 8
        CP      D               ; compare with parameter
        SBC     A,A             ; $FF if was 9,  $00 if 0-8
                                ; continue while addressing P-FLAG
                                ; setting bit 4 if ink 9
                                ; setting bit 6 if paper 9
```

```
; -----------------------
; Handle change of colour
; -----------------------
; This routine addresses a system variable ATTR_T, MASK_T or P-FLAG in HL.
; colour value in A, mask in B.

;; CO-CHANGE
L226C:  XOR     (HL)            ; impress bits specified
        AND     B               ; by mask
        XOR     (HL)            ; on system variable.
        LD      (HL),A          ; update system variable.
        INC     HL              ; address next location.
        LD      A,B             ; put current value of mask in A
        RET                     ; return.

; ---

; the branch was here with flash and bright

;; CO-TEMP-C
L2273:  SBC     A,A             ; set zero flag for bright.
        LD      A,D             ; fetch original parameter 0,1 or 8
        RRCA                    ; rotate bit 0 to bit 7
        LD      B,$80           ; mask for flash 10000000
        JR      NZ,L227D        ; forward to CO-TEMP-D if flash

        RRCA                    ; rotate bit 7 to bit 6
        LD      B,$40           ; mask for bright 01000000

;; CO-TEMP-D
L227D:  LD      C,A             ; store value in C
        LD      A,D             ; fetch parameter
        CP      $08             ; compare with 8
        JR      Z,L2287         ; forward to CO-TEMP-E if 8

        CP      $02             ; test if 0 or 1
        JR      NC,L2244        ; back to REPORT-K if not
                                ; 'Invalid colour'

;; CO-TEMP-E
L2287:  LD      A,C             ; value to A
        LD      HL,$5C8F        ; address ATTR_T
        CALL    L226C           ; routine CO-CHANGE addressing ATTR_T
        LD      A,C             ; fetch value
        RRCA                    ; for flash8/bright8 complete
        RRCA                    ; rotations to put set bit in
        RRCA                    ; bit 7 (flash) bit 6 (bright)
        JR      L226C           ; back to CO-CHANGE addressing MASK_T
                                ; and indirect return.

; ---------------------
; Handle BORDER command
; ---------------------
; Command syntax example: BORDER 7
; This command routine sets the border to one of the eight colours.
; The colours used for the lower screen are based on this.

;; BORDER
L2294:  CALL    L1E94           ; routine FIND-INT1
        CP      $08             ; must be in range 0 (black) to 7 (white)
        JR      NC,L2244        ; back to REPORT-K if not
                                ; 'Invalid colour'.

        OUT     ($FE),A         ; outputting to port effects an immediate
```

```
                                ; change.
        RLCA                    ; shift the colour to
        RLCA                    ; the paper bits setting the
        RLCA                    ; ink colour black.
        BIT     5,A             ; is the number light coloured ?
                                ; i.e. in the range green to white.
        JR      NZ,L22A6        ; skip to BORDER-1 if so

        XOR     $07             ; make the ink white.

;; BORDER-1
L22A6:  LD      ($5C48),A       ; update BORDCR with new paper/ink
        RET                     ; return.

; -----------------
; Get pixel address
; -----------------
;
;

;; PIXEL-ADD
L22AA:  LD      A,$AF           ; load with 175 decimal.
        SUB     B               ; subtract the y value.
        JP      C,L24F9         ; jump forward to REPORT-Bc if greater.
                                ; 'Integer out of range'

; the high byte is derived from Y only.
; the first 3 bits are always 010
; the next 2 bits denote in which third of the screen the byte is.
; the last 3 bits denote in which of the 8 scan lines within a third
; the byte is located. There are 24 discrete values.


        LD      B,A             ; the line number from top of screen to B.
        AND     A               ; clear carry (already clear)
        RRA                     ;                     0xxxxxxx
        SCF                     ; set carry flag
        RRA                     ;                     10xxxxxx
        AND     A               ; clear carry flag
        RRA                     ;                     010xxxxx

        XOR     B               ;
        AND     $F8             ; keep the top 5 bits 11111000
        XOR     B               ;                     010xxbbb
        LD      H,A             ; transfer high byte to H.

; the low byte is derived from both X and Y.

        LD      A,C             ; the x value 0-255.
        RLCA                    ;
        RLCA                    ;
        RLCA                    ;
        XOR     B               ; the y value
        AND     $C7             ; apply mask             11000111
        XOR     B               ; restore unmasked bits  xxyyyxxx
        RLCA                    ; rotate to              xyyyxxxx
        RLCA                    ; required position.     yyyxxxxx
        LD      L,A             ; low byte to L.

; finally form the pixel position in A.

        LD      A,C             ; x value to A
        AND     $07             ; mod 8
        RET                     ; return
```

```
; ----------------
; Point Subroutine
; ----------------
; The point subroutine is called from s-point via the scanning functions
; table.

;; POINT-SUB
L22CB:  CALL    L2307           ; routine STK-TO-BC
        CALL    L22AA           ; routine PIXEL-ADD finds address of pixel.
        LD      B,A             ; pixel position to B, 0-7.
        INC     B               ; increment to give rotation count 1-8.
        LD      A,(HL)          ; fetch byte from screen.

;; POINT-LP
L22D4:  RLCA                    ; rotate and loop back
        DJNZ    L22D4           ; to POINT-LP until pixel at right.

        AND      $01            ; test to give zero or one.
        JP      L2D28           ; jump forward to STACK-A to save result.

; -------------------
; Handle PLOT command
; -------------------
; Command Syntax example: PLOT 128,88
;

;; PLOT
L22DC:  CALL    L2307           ; routine STK-TO-BC
        CALL    L22E5           ; routine PLOT-SUB
        JP      L0D4D           ; to TEMPS

; -------------------
; The Plot subroutine
; -------------------
; A screen byte holds 8 pixels so it is necessary to rotate a mask
; into the correct position to leave the other 7 pixels unaffected.
; However all 64 pixels in the character cell take any embedded colour
; items.
; A pixel can be reset (inverse 1), toggled (over 1), or set ( with inverse
; and over switches off). With both switches on, the byte is simply put
; back on the screen though the colours may change.

;; PLOT-SUB
L22E5:  LD      ($5C7D),BC      ; store new x/y values in COORDS
        CALL    L22AA           ; routine PIXEL-ADD gets address in HL,
                                ; count from left 0-7 in B.
        LD      B,A             ; transfer count to B.
        INC     B               ; increase 1-8.
        LD      A,$FE           ; 11111110 in A.

;; PLOT-LOOP
L22F0:  RRCA                    ; rotate mask.
        DJNZ    L22F0           ; to PLOT-LOOP until B circular rotations.

        LD      B,A             ; load mask to B
        LD      A,(HL)          ; fetch screen byte to A

        LD      C,(IY+$57)      ; P_FLAG to C
        BIT     0,C             ; is it to be OVER 1 ?
        JR      NZ,L22FD        ; forward to PL-TST-IN if so.

; was over 0
```

```
        AND     B               ; combine with mask to blank pixel.

;; PL-TST-IN
L22FD:  BIT     2,C             ; is it inverse 1 ?
        JR      NZ,L2303        ; to PLOT-END if so.

        XOR     B               ; switch the pixel
        CPL                     ; restore other 7 bits

;; PLOT-END
L2303:  LD      (HL),A          ; load byte to the screen.
        JP      L0BDB           ; exit to PO-ATTR to set colours for cell.

; ------------------------------
; Put two numbers in BC register
; ------------------------------
;
;

;; STK-TO-BC
L2307:  CALL    L2314           ; routine STK-TO-A
        LD      B,A             ;
        PUSH    BC              ;
        CALL    L2314           ; routine STK-TO-A
        LD      E,C             ;
        POP     BC              ;
        LD      D,C             ;
        LD      C,A             ;
        RET                     ;

; -----------------------
; Put stack in A register
; -----------------------
; This routine puts the last value on the calculator stack into the accumulator
; deleting the last value.

;; STK-TO-A
L2314:  CALL    L2DD5           ; routine FP-TO-A compresses last value into
                                ; accumulator. e.g. PI would become 3.
                                ; zero flag set if positive.
        JP      C,L24F9         ; jump forward to REPORT-Bc if >= 255.5.

        LD      C,$01           ; prepare a positive sign byte.
        RET     Z               ; return if FP-TO-BC indicated positive.

        LD      C,$FF           ; prepare negative sign byte and
        RET                     ; return.


; ---------------------
; Handle CIRCLE command
; ---------------------
;
; syntax has been partly checked using the class for draw command.

;; CIRCLE
L2320:  RST     18H             ; GET-CHAR
        CP      $2C             ; is it required comma ?
        JP      NZ,L1C8A        ; jump to REPORT-C if not


        RST     20H             ; NEXT-CHAR
        CALL    L1C82           ; routine EXPT-1NUM fetches radius
        CALL    L1BEE           ; routine CHECK-END will return here if
```

```
                                ; nothing follows command.

        RST     28H             ;; FP-CALC
        DEFB    $2A             ;;abs           ; make radius positive
        DEFB    $3D             ;;re-stack      ; in full floating point form
        DEFB    $38             ;;end-calc

        LD      A,(HL)          ; fetch first floating point byte
        CP      $81             ; compare to one
        JR      NC,L233B        ; forward to C-R-GRE-1 if circle radius
                                ; is greater than one.


        RST     28H             ;; FP-CALC
        DEFB    $02             ;;delete        ; delete the radius from stack.
        DEFB    $38             ;;end-calc

        JR      L22DC           ; to PLOT to just plot x,y.

; ---


;; C-R-GRE-1
L233B:  RST     28H             ;; FP-CALC      ; x, y, r
        DEFB    $A3             ;;stk-pi/2      ; x, y, r, pi/2.
        DEFB    $38             ;;end-calc

        LD      (HL),$83        ;               ; x, y, r, 2*PI

        RST     28H             ;; FP-CALC
        DEFB    $C5             ;;st-mem-5      ; store 2*PI in mem-5
        DEFB    $02             ;;delete        ; x, y, z.
        DEFB    $38             ;;end-calc

        CALL    L247D           ; routine CD-PRMS1
        PUSH    BC              ;

        RST     28H             ;; FP-CALC
        DEFB    $31             ;;duplicate
        DEFB    $E1             ;;get-mem-1
        DEFB    $04             ;;multiply
        DEFB    $38             ;;end-calc

        LD      A,(HL)          ;
        CP      $80             ;
        JR      NC,L235A        ; to C-ARC-GE1


        RST     28H             ;; FP-CALC
        DEFB    $02             ;;delete
        DEFB    $02             ;;delete
        DEFB    $38             ;;end-calc

        POP     BC              ;
        JP      L22DC           ; JUMP to PLOT

; ---


;; C-ARC-GE1
L235A:  RST     28H             ;; FP-CALC
        DEFB    $C2             ;;st-mem-2
        DEFB    $01             ;;exchange
        DEFB    $C0             ;;st-mem-0
```

```
        DEFB    $02             ;;delete
        DEFB    $03             ;;subtract
        DEFB    $01             ;;exchange
        DEFB    $E0             ;;get-mem-0
        DEFB    $0F             ;;addition
        DEFB    $C0             ;;st-mem-0
        DEFB    $01             ;;exchange
        DEFB    $31             ;;duplicate
        DEFB    $E0             ;;get-mem-0
        DEFB    $01             ;;exchange
        DEFB    $31             ;;duplicate
        DEFB    $E0             ;;get-mem-0
        DEFB    $A0             ;;stk-zero
        DEFB    $C1             ;;st-mem-1
        DEFB    $02             ;;delete
        DEFB    $38             ;;end-calc

        INC     (IY+$62)        ; MEM-2-1st
        CALL     L1E94          ; routine FIND-INT1
        LD      L,A             ;
        PUSH    HL              ;
        CALL    L1E94           ; routine FIND-INT1
        POP     HL              ;
        LD      H,A             ;
        LD      ($5C7D),HL      ; COORDS
        POP     BC              ;
        JP      L2420           ; to DRW-STEPS


; -------------------
; Handle DRAW command
; -------------------
;
;

;; DRAW
L2382:  RST     18H             ; GET-CHAR
        CP      $2C             ;
        JR      Z,L238D         ; to DR-3-PRMS

        CALL    L1BEE           ; routine CHECK-END
        JP      L2477           ; to LINE-DRAW

; ---

;; DR-3-PRMS
L238D:  RST     20H             ; NEXT-CHAR
        CALL    L1C82           ; routine EXPT-1NUM
        CALL    L1BEE           ; routine CHECK-END

        RST     28H             ;; FP-CALC
        DEFB    $C5             ;;st-mem-5
        DEFB    $A2             ;;stk-half
        DEFB    $04             ;;multiply
        DEFB    $1F             ;;sin
        DEFB    $31             ;;duplicate
        DEFB    $30             ;;not
        DEFB    $30             ;;not
        DEFB    $00             ;;jump-true

        DEFB    $06             ;;to L23A3, DR-SIN-NZ

        DEFB    $02             ;;delete
        DEFB    $38             ;;end-calc
```

```
        JP      L2477           ; to LINE-DRAW

; ---

;; DR-SIN-NZ
L23A3:  DEFB    $C0             ;;st-mem-0
        DEFB    $02             ;;delete
        DEFB    $C1             ;;st-mem-1
        DEFB    $02             ;;delete
        DEFB    $31             ;;duplicate
        DEFB    $2A             ;;abs
        DEFB    $E1             ;;get-mem-1
        DEFB    $01             ;;exchange
        DEFB    $E1             ;;get-mem-1
        DEFB    $2A             ;;abs
        DEFB    $0F             ;;addition
        DEFB    $E0             ;;get-mem-0
        DEFB    $05             ;;division
        DEFB    $2A             ;;abs
        DEFB    $E0             ;;get-mem-0
        DEFB    $01             ;;exchange
        DEFB    $3D             ;;re-stack
        DEFB    $38             ;;end-calc

        LD      A,(HL)          ;
        CP      $81             ;
        JR      NC,L23C1        ; to DR-PRMS


        RST     28H             ;; FP-CALC
        DEFB    $02             ;;delete
        DEFB    $02             ;;delete
        DEFB    $38             ;;end-calc

        JP      L2477           ; to LINE-DRAW

; ---

;; DR-PRMS
L23C1:  CALL    L247D           ; routine CD-PRMS1
        PUSH    BC              ;

        RST     28H             ;; FP-CALC
        DEFB    $02             ;;delete
        DEFB    $E1             ;;get-mem-1
        DEFB    $01             ;;exchange
        DEFB    $05             ;;division
        DEFB    $C1             ;;st-mem-1
        DEFB    $02             ;;delete
        DEFB    $01             ;;exchange
        DEFB    $31             ;;duplicate
        DEFB    $E1             ;;get-mem-1
        DEFB    $04             ;;multiply
        DEFB    $C2             ;;st-mem-2
        DEFB    $02             ;;delete
        DEFB    $01             ;;exchange
        DEFB    $31             ;;duplicate
        DEFB    $E1             ;;get-mem-1
        DEFB    $04             ;;multiply
        DEFB    $E2             ;;get-mem-2
        DEFB    $E5             ;;get-mem-5
        DEFB    $E0             ;;get-mem-0
        DEFB    $03             ;;subtract
```

```
DEFB    $A2             ;;stk-half
DEFB    $04             ;;multiply
DEFB    $31             ;;duplicate
DEFB    $1F             ;;sin
DEFB    $C5             ;;st-mem-5
DEFB    $02             ;;delete
DEFB    $20             ;;cos
DEFB    $C0             ;;st-mem-0
DEFB    $02             ;;delete
DEFB    $C2             ;;st-mem-2
DEFB    $02             ;;delete
DEFB    $C1             ;;st-mem-1
DEFB    $E5             ;;get-mem-5
DEFB    $04             ;;multiply
DEFB    $E0             ;;get-mem-0
DEFB    $E2             ;;get-mem-2
DEFB    $04             ;;multiply
DEFB    $0F             ;;addition
DEFB    $E1             ;;get-mem-1
DEFB    $01             ;;exchange
DEFB    $C1             ;;st-mem-1
DEFB    $02             ;;delete
DEFB    $E0             ;;get-mem-0
DEFB    $04             ;;multiply
DEFB    $E2             ;;get-mem-2
DEFB    $E5             ;;get-mem-5
DEFB    $04             ;;multiply
DEFB    $03             ;;subtract
DEFB    $C2             ;;st-mem-2
DEFB    $2A             ;;abs
DEFB    $E1             ;;get-mem-1
DEFB    $2A             ;;abs
DEFB    $0F             ;;addition
DEFB    $02             ;;delete
DEFB    $38             ;;end-calc

LD      A,(DE)          ;
CP       $81            ;
POP     BC              ;
JP      C,L2477         ; to LINE-DRAW

PUSH    BC              ;

RST     28H             ;; FP-CALC
DEFB    $01             ;;exchange
DEFB    $38             ;;end-calc

LD      A,($5C7D)       ; COORDS-x
CALL    L2D28           ; routine STACK-A

RST     28H             ;; FP-CALC
DEFB    $C0             ;;st-mem-0
DEFB    $0F             ;;addition
DEFB    $01             ;;exchange
DEFB    $38             ;;end-calc

LD      A,($5C7E)       ; COORDS-y
CALL    L2D28           ; routine STACK-A

RST     28H             ;; FP-CALC
DEFB    $C5             ;;st-mem-5
DEFB    $0F             ;;addition
DEFB    $E0             ;;get-mem-0
DEFB    $E5             ;;get-mem-5
```

```
        DEFB    $38             ;;end-calc

        POP     BC              ;

;; DRW-STEPS
L2420:  DEC     B               ;
        JR      Z,L245F         ; to ARC-END

        JR      L2439           ; to ARC-START

; ---


;; ARC-LOOP
L2425:  RST     28H             ;; FP-CALC
        DEFB    $E1             ;;get-mem-1
        DEFB    $31             ;;duplicate
        DEFB    $E3             ;;get-mem-3
        DEFB    $04             ;;multiply
        DEFB    $E2             ;;get-mem-2
        DEFB    $E4             ;;get-mem-4
        DEFB    $04             ;;multiply
        DEFB    $03             ;;subtract
        DEFB    $C1             ;;st-mem-1
        DEFB    $02             ;;delete
        DEFB    $E4             ;;get-mem-4
        DEFB    $04             ;;multiply
        DEFB    $E2             ;;get-mem-2
        DEFB    $E3             ;;get-mem-3
        DEFB    $04             ;;multiply
        DEFB    $0F             ;;addition
        DEFB    $C2             ;;st-mem-2
        DEFB    $02             ;;delete
        DEFB    $38             ;;end-calc

;; ARC-START
L2439:  PUSH    BC              ;

        RST     28H             ;; FP-CALC
        DEFB    $C0             ;;st-mem-0
        DEFB    $02             ;;delete
        DEFB    $E1             ;;get-mem-1
        DEFB    $0F             ;;addition
        DEFB    $31             ;;duplicate
        DEFB    $38             ;;end-calc

        LD      A,($5C7D)       ; COORDS-x
        CALL    L2D28           ; routine STACK-A

        RST     28H             ;; FP-CALC
        DEFB    $03             ;;subtract
        DEFB    $E0             ;;get-mem-0
        DEFB    $E2             ;;get-mem-2
        DEFB    $0F             ;;addition
        DEFB    $C0             ;;st-mem-0
        DEFB    $01             ;;exchange
        DEFB    $E0             ;;get-mem-0
        DEFB    $38             ;;end-calc

        LD      A,($5C7E)       ; COORDS-y
        CALL    L2D28           ; routine STACK-A

        RST     28H             ;; FP-CALC
        DEFB    $03             ;;subtract
```

```
        DEFB    $38             ;;end-calc

        CALL    L24B7           ; routine DRAW-LINE
        POP     BC              ;
        DJNZ    L2425           ; to ARC-LOOP


;; ARC-END
L245F:  RST     28H             ;; FP-CALC
        DEFB    $02             ;;delete
        DEFB    $02             ;;delete
        DEFB    $01             ;;exchange
        DEFB    $38             ;;end-calc

        LD      A,($5C7D)       ; COORDS-x
        CALL    L2D28           ; routine STACK-A

        RST     28H             ;; FP-CALC
        DEFB    $03             ;;subtract
        DEFB    $01             ;;exchange
        DEFB    $38             ;;end-calc

        LD      A,($5C7E)       ; COORDS-y
        CALL    L2D28           ; routine STACK-A

        RST     28H             ;; FP-CALC
        DEFB    $03             ;;subtract
        DEFB    $38             ;;end-calc

;; LINE-DRAW
L2477:  CALL    L24B7           ; routine DRAW-LINE
        JP      L0D4D           ; to TEMPS


; ------------------
; Initial parameters
; ------------------
;
;

;; CD-PRMS1
L247D:  RST     28H             ;; FP-CALC
        DEFB    $31             ;;duplicate
        DEFB    $28             ;;sqr
        DEFB    $34             ;;stk-data
        DEFB    $32             ;;Exponent: $82, Bytes: 1
        DEFB    $00             ;;(+00,+00,+00)
        DEFB    $01             ;;exchange
        DEFB    $05             ;;division
        DEFB    $E5             ;;get-mem-5
        DEFB    $01             ;;exchange
        DEFB    $05             ;;division
        DEFB    $2A             ;;abs
        DEFB    $38             ;;end-calc

        CALL    L2DD5           ; routine FP-TO-A
        JR      C,L2495         ; to USE-252

        AND     $FC             ;
        ADD     A,$04           ;
        JR      NC,L2497        ; to DRAW-SAVE

;; USE-252
L2495:  LD      A,$FC           ;
```

```
;; DRAW-SAVE
L2497:  PUSH    AF              ;
        CALL    L2D28           ; routine STACK-A

        RST     28H             ;; FP-CALC
        DEFB    $E5             ;;get-mem-5
        DEFB    $01             ;;exchange
        DEFB    $05             ;;division
        DEFB    $31             ;;duplicate
        DEFB    $1F             ;;sin
        DEFB    $C4             ;;st-mem-4
        DEFB    $02             ;;delete
        DEFB    $31             ;;duplicate
        DEFB    $A2             ;;stk-half
        DEFB    $04             ;;multiply
        DEFB    $1F             ;;sin
        DEFB    $C1             ;;st-mem-1
        DEFB    $01             ;;exchange
        DEFB    $C0             ;;st-mem-0
        DEFB    $02             ;;delete
        DEFB    $31             ;;duplicate
        DEFB    $04             ;;multiply
        DEFB    $31             ;;duplicate
        DEFB    $0F             ;;addition
        DEFB    $A1             ;;stk-one
        DEFB    $03             ;;subtract
        DEFB    $1B             ;;negate
        DEFB    $C3             ;;st-mem-3
        DEFB    $02             ;;delete
        DEFB    $38             ;;end-calc

        POP     BC              ;
        RET                     ;

; ------------
; Line drawing
; ------------
;
;

;; DRAW-LINE
L24B7:  CALL    L2307           ; routine STK-TO-BC
        LD      A,C             ;
        CP      B               ;
        JR      NC,L24C4        ; to DL-X-GE-Y

        LD      L,C             ;
        PUSH    DE              ;
        XOR     A               ;
        LD      E,A             ;
        JR      L24CB           ; to DL-LARGER

; ---

;; DL-X-GE-Y
L24C4:  OR      C               ;
        RET     Z               ;

        LD      L,B             ;
        LD      B,C             ;
        PUSH    DE              ;
        LD      D,$00           ;
```

```
;; DL-LARGER
L24CB:  LD      H,B             ;
        LD      A,B             ;
        RRA                     ;

;; D-L-LOOP
L24CE:  ADD     A,L             ;
        JR      C,L24D4         ; to D-L-DIAG

        CP      H               ;
        JR      C,L24DB         ; to D-L-HR-VT

;; D-L-DIAG
L24D4:  SUB     H               ;
        LD      C,A             ;
        EXX                     ;
        POP     BC              ;
        PUSH    BC              ;
        JR      L24DF           ; to D-L-STEP

; ---

;; D-L-HR-VT
L24DB:  LD      C,A             ;
        PUSH    DE              ;
        EXX                     ;
        POP     BC              ;

;; D-L-STEP
L24DF:  LD      HL,($5C7D)      ; COORDS
        LD      A,B             ;
        ADD     A,H             ;
        LD      B,A             ;
        LD      A,C             ;
        INC     A               ;
        ADD     A,L             ;
        JR      C,L24F7         ; to D-L-RANGE

        JR      Z,L24F9         ; to REPORT-Bc

;; D-L-PLOT
L24EC:  DEC     A               ;
        LD      C,A             ;
        CALL    L22E5           ; routine PLOT-SUB
        EXX                     ;
        LD      A,C             ;
        DJNZ    L24CE           ; to D-L-LOOP

        POP     DE              ;
        RET                     ;

; ---

;; D-L-RANGE
L24F7:  JR      Z,L24EC         ; to D-L-PLOT


;; REPORT-Bc
L24F9:  RST     08H             ; ERROR-1
        DEFB    $0A             ; Error Report: Integer out of range


;***********************************
```

```
;** Part 8. EXPRESSION EVALUATION **
;***********************************
;
; It is a this stage of the ROM that the Spectrum ceases altogether to be
; just a colourful novelty. One remarkable feature is that in all previous
; commands when the Spectrum is expecting a number or a string then an
; expression of the same type can be substituted ad infinitum.
; This is the routine that evaluates that expression.
; This is what causes 2 + 2 to give the answer 4.
; That is quite easy to understand. However you don't have to make it much
; more complex to start a remarkable juggling act.
; e.g. PRINT 2 * (VAL "2+2" + TAN 3)
; In fact, provided there is enough free RAM, the Spectrum can evaluate
; an expression of unlimited complexity.
; Apart from a couple of minor glitches, which you can now correct, the
; system is remarkably robust.


; ---------------------------------
; Scan expression or sub-expression
; ---------------------------------
;
;

;; SCANNING
L24FB:  RST     18H             ; GET-CHAR
        LD      B,$00           ; priority marker zero is pushed on stack
                                ; to signify end of expression when it is
                                ; popped off again.
        PUSH    BC              ; put in on stack.
                                ; and proceed to consider the first character
                                ; of the expression.

;; S-LOOP-1
L24FF:  LD      C,A             ; store the character while a look up is done.
        LD      HL,L2596        ; Address: scan-func
        CALL    L16DC           ; routine INDEXER is called to see if it is
                                ; part of a limited range '+', '(', 'ATTR' etc.

        LD      A,C             ; fetch the character back
        JP      NC,L2684        ; jump forward to S-ALPHNUM if not in primary
                                ; operators and functions to consider in the
                                ; first instance a digit or a variable and
                                ; then anything else.                >>>

        LD      B,$00           ; but here if it was found in table so
        LD      C,(HL)          ; fetch offset from table and make B zero.
        ADD     HL,BC           ; add the offset to position found
        JP      (HL)            ; and jump to the routine e.g. S-BIN
                                ; making an indirect exit from there.

; -------------------------------------------------------------------------
; The four service subroutines for routines in the scanning function table
; -------------------------------------------------------------------------

; PRINT """Hooray!"" he cried."

;; S-QUOTE-S
L250F:  CALL    L0074           ; routine CH-ADD+1 points to next character
                                ; and fetches that character.
        INC     BC              ; increase length counter.
        CP      $0D             ; is it carriage return ?
                                ; inside a quote.
        JP      Z,L1C8A         ; jump back to REPORT-C if so.
```

```
                                ; 'Nonsense in BASIC'.

        CP      $22             ; is it a quote '"' ?
        JR      NZ,L250F        ; back to S-QUOTE-S if not for more.

        CALL    L0074           ; routine CH-ADD+1
        CP      $22             ; compare with possible adjacent quote
        RET                     ; return. with zero set if two together.

; ---

; This subroutine is used to get two coordinate expressions for the three
; functions SCREEN$, ATTR and POINT that have two fixed parameters and
; therefore require surrounding braces.

;; S-2-COORD
L2522:  RST     20H             ; NEXT-CHAR
        CP      $28             ; is it the opening '(' ?
        JR      NZ,L252D        ; forward to S-RPORT-C if not
                                ; 'Nonsense in BASIC'.

        CALL    L1C79           ; routine NEXT-2NUM gets two comma-separated
                                ; numeric expressions. Note. this could cause
                                ; many more recursive calls to SCANNING but
                                ; the parent function will be evaluated fully
                                ; before rejoining the main juggling act.

        RST     18H             ; GET-CHAR
        CP      $29             ; is it the closing ')' ?

;; S-RPORT-C
L252D:  JP      NZ,L1C8A        ; jump back to REPORT-C if not.
                                ; 'Nonsense in BASIC'.

; ------------
; Check syntax
; ------------
; This routine is called on a number of occasions to check if syntax is being
; checked or if the program is being run. To test the flag inline would use
; four bytes of code, but a call instruction only uses 3 bytes of code.

;; SYNTAX-Z
L2530:  BIT     7,(IY+$01)      ; test FLAGS  - checking syntax only ?
        RET                     ; return.

; ----------------
; Scanning SCREEN$
; ----------------
; This function returns the code of a bit-mapped character at screen
; position at line C, column B. It is unable to detect the mosaic characters
; which are not bit-mapped but detects the ASCII 32 - 127 range.
; The bit-mapped UDGs are ignored which is curious as it requires only a
; few extra bytes of code. As usual, anything to do with CHARS is weird.
; If no match is found a null string is returned.
; No actual check on ranges is performed - that's up to the BASIC programmer.
; No real harm can come from SCREEN$(255,255) although the BASIC manual
; says that invalid values will be trapped.
; Interestingly, in the Pitman pocket guide, 1984, Vickers says that the
; range checking will be performed.

;; S-SCRN$-S
L2535:  CALL    L2307           ; routine STK-TO-BC.
        LD      HL,($5C36)      ; fetch address of CHARS.
        LD      DE,$0100        ; fetch offset to chr$ 32
```

```
        ADD     HL,DE           ; and find start of bitmaps.
                                ; Note. not inc h. ??
        LD      A,C             ; transfer line to A.
        RRCA                    ; multiply
        RRCA                    ; by
        RRCA                    ; thirty-two.
        AND     $E0             ; and with 11100000
        XOR     B               ; combine with column $00 - $1F
        LD      E,A             ; to give the low byte of top line
        LD      A,C             ; column to A range 00000000 to 00011111
        AND     $18             ; and with 00011000
        XOR     $40             ; xor with 01000000 (high byte screen start)
        LD      D,A             ; register DE now holds start address of cell.
        LD      B,$60           ; there are 96 characters in ASCII set.

;; S-SCRN-LP
L254F:  PUSH    BC              ; save count
        PUSH    DE              ; save screen start address
        PUSH    HL              ; save bitmap start
        LD      A,(DE)          ; first byte of screen to A
        XOR     (HL)            ; xor with corresponding character byte
        JR      Z,L255A         ; forward to S-SC-MTCH if they match
                                ; if inverse result would be $FF
                                ; if any other then mismatch

        INC     A               ; set to $00 if inverse
        JR      NZ,L2573        ; forward to S-SCR-NXT if a mismatch

        DEC     A               ; restore $FF

; a match has been found so seven more to test.

;; S-SC-MTCH
L255A:  LD      C,A             ; load C with inverse mask $00 or $FF
        LD      B,$07           ; count seven more bytes

;; S-SC-ROWS
L255D:  INC     D               ; increment screen address.
        INC     HL              ; increment bitmap address.
        LD      A,(DE)          ; byte to A
        XOR     (HL)            ; will give $00 or $FF (inverse)
        XOR     C               ; xor with inverse mask
        JR      NZ,L2573        ; forward to S-SCR-NXT if no match.

        DJNZ    L255D           ; back to S-SC-ROWS until all eight matched.

; continue if a match of all eight bytes was found

        POP     BC              ; discard the
        POP     BC              ; saved
        POP     BC              ; pointers
        LD      A,$80           ; the endpoint of character set
        SUB     B               ; subtract the counter
                                ; to give the code 32-127
        LD      BC,$0001        ; make one space in workspace.

        RST     30H             ; BC-SPACES creates the space sliding
                                ; the calculator stack upwards.
        LD      (DE),A          ; start is addressed by DE, so insert code
        JR      L257D           ; forward to S-SCR-STO

; ---

; the jump was here if no match and more bitmaps to test.
```

```
;; S-SCR-NXT
L2573:  POP     HL              ; restore the last bitmap start
        LD      DE,$0008        ; and prepare to add 8.
        ADD     HL,DE           ; now addresses next character bitmap.
        POP     DE              ; restore screen address
        POP     BC              ; and character counter in B
        DJNZ    L254F           ; back to S-SCRN-LP if more characters.

        LD      C,B             ; B is now zero, so BC now zero.

;; S-SCR-STO
L257D:  JP      L2AB2           ; to STK-STO-$ to store the string in
                                ; workspace or a string with zero length.
                                ; (value of DE doesn't matter in last case)

; Note. this exit seems correct but the general-purpose routine S-STRING
; that calls this one will also stack any of its string results so this
; leads to a double storing of the result in this case.
; The instruction at L257D should just be a RET.
; credit Stephen Kelly and others, 1982.

; -------------
; Scanning ATTR
; -------------
; This function subroutine returns the attributes of a screen location -
; a numeric result.
; Again it's up to the BASIC programmer to supply valid values of line/column.

;; S-ATTR-S
L2580:  CALL    L2307           ; routine STK-TO-BC fetches line to C,
                                ; and column to B.
        LD      A,C             ; line to A $00 - $17   (max 00010111)
        RRCA                    ; rotate
        RRCA                    ; bits
        RRCA                    ; left.
        LD      C,A             ; store in C as an intermediate value.

        AND     $E0             ; pick up bits 11100000 ( was 00011100 )
        XOR     B               ; combine with column $00 - $1F
        LD      L,A             ; low byte now correct.

        LD      A,C             ; bring back intermediate result from C
        AND     $03             ; mask to give correct third of
                                ; screen $00 - $02
        XOR     $58             ; combine with base address.
        LD      H,A             ; high byte correct.
        LD      A,(HL)          ; pick up the colour attribute.
        JP      L2D28           ; forward to STACK-A to store result
                                ; and make an indirect exit.

; -----------------------
; Scanning function table
; -----------------------
; This table is used by INDEXER routine to find the offsets to
; four operators and eight functions. e.g. $A8 is the token 'FN'.
; This table is used in the first instance for the first character of an
; expression or by a recursive call to SCANNING for the first character of
; any sub-expression. It eliminates functions that have no argument or
; functions that can have more than one argument and therefore require
; braces. By eliminating and dealing with these now it can later take a
; simplistic approach to all other functions and assume that they have
; one argument.
; Similarly by eliminating BIN and '.' now it is later able to assume that
```

```
; all numbers begin with a digit and that the presence of a number or
; variable can be detected by a call to ALPHANUM.
; By default all expressions are positive and the spurious '+' is eliminated
; now as in print +2. This should not be confused with the operator '+'.
; Note. this does allow a degree of nonsense to be accepted as in
; PRINT +"3 is the greatest.".
; An acquired programming skill is the ability to include brackets where
; they are not necessary.
; A bracket at the start of a sub-expression may be spurious or necessary
; to denote that the contained expression is to be evaluated as an entity.
; In either case this is dealt with by recursive calls to SCANNING.
; An expression that begins with a quote requires special treatment.

;; scan-func
L2596:  DEFB    $22, L25B3-$-1  ; $1C offset to S-QUOTE
        DEFB    '(', L25E8-$-1  ; $4F offset to S-BRACKET
        DEFB    '.', L268D-$-1  ; $F2 offset to S-DECIMAL
        DEFB    '+', L25AF-$-1  ; $12 offset to S-U-PLUS

        DEFB    $A8, L25F5-$-1  ; $56 offset to S-FN
        DEFB    $A5, L25F8-$-1  ; $57 offset to S-RND
        DEFB    $A7, L2627-$-1  ; $84 offset to S-PI
        DEFB    $A6, L2634-$-1  ; $8F offset to S-INKEY$
        DEFB    $C4, L268D-$-1  ; $E6 offset to S-BIN
        DEFB    $AA, L2668-$-1  ; $BF offset to S-SCREEN$
        DEFB    $AB, L2672-$-1  ; $C7 offset to S-ATTR
        DEFB    $A9, L267B-$-1  ; $CE offset to S-POINT

        DEFB    $00             ; zero end marker

; --------------------------
; Scanning function routines
; --------------------------
; These are the 11 subroutines accessed by the above table.
; S-BIN and S-DECIMAL are the same
; The 1-byte offset limits their location to within 255 bytes of their
; entry in the table.

; ->
;; S-U-PLUS
L25AF:  RST     20H             ; NEXT-CHAR just ignore
        JP      L24FF           ; to S-LOOP-1

; ---

; ->
;; S-QUOTE
L25B3:  RST     18H             ; GET-CHAR
        INC     HL              ; address next character (first in quotes)
        PUSH    HL              ; save start of quoted text.
        LD      BC,$0000        ; initialize length of string to zero.
        CALL    L250F           ; routine S-QUOTE-S
        JR      NZ,L25D9        ; forward to S-Q-PRMS if

;; S-Q-AGAIN
L25BE:  CALL    L250F           ; routine S-QUOTE-S copies string until a
                                ; quote is encountered
        JR      Z,L25BE         ; back to S-Q-AGAIN if two quotes WERE
                                ; together.

; but if just an isolated quote then that terminates the string.

        CALL    L2530           ; routine SYNTAX-Z
        JR      Z,L25D9         ; forward to S-Q-PRMS if checking syntax.
```

```
        RST     30H             ; BC-SPACES creates the space for true
                                ; copy of string in workspace.
        POP     HL              ; re-fetch start of quoted text.
        PUSH    DE              ; save start in workspace.

;; S-Q-COPY
L25CB:  LD      A,(HL)          ; fetch a character from source.
        INC     HL              ; advance source address.
        LD      (DE),A          ; place in destination.
        INC     DE              ; advance destination address.
        CP      $22             ; was it a '"' just copied ?
        JR      NZ,L25CB        ; back to S-Q-COPY to copy more if not

        LD      A,(HL)          ; fetch adjacent character from source.
        INC     HL              ; advance source address.
        CP      $22             ; is this '"' ? - i.e. two quotes together ?
        JR      Z,L25CB         ; to S-Q-COPY if so including just one of the
                                ; pair of quotes.

; proceed when terminating quote encountered.

;; S-Q-PRMS
L25D9:  DEC     BC              ; decrease count by 1.
        POP     DE              ; restore start of string in workspace.

;; S-STRING
L25DB:  LD      HL,$5C3B        ; Address FLAGS system variable.
        RES     6,(HL)          ; signal string result.
        BIT     7,(HL)          ; is syntax being checked.
        CALL    NZ,L2AB2        ; routine STK-STO-$ is called in runtime.
        JP      L2712           ; jump forward to S-CONT-2          ===>

; ---

; ->
;; S-BRACKET
L25E8:  RST     20H             ; NEXT-CHAR
        CALL    L24FB           ; routine SCANNING is called recursively.
        CP      $29             ; is it the closing ')' ?
        JP      NZ,L1C8A        ; jump back to REPORT-C if not
                                ; 'Nonsense in BASIC'

        RST     20H             ; NEXT-CHAR
        JP      L2712           ; jump forward to S-CONT-2          ===>

; ---

; ->
;; S-FN
L25F5:  JP      L27BD           ; jump forward to S-FN-SBRN.

; ---

; ->
;; S-RND
L25F8:  CALL    L2530           ; routine SYNTAX-Z
        JR      Z,L2625         ; forward to S-RND-END if checking syntax.

        LD      BC,($5C76)      ; fetch system variable SEED
        CALL    L2D2B           ; routine STACK-BC places on calculator stack

        RST     28H             ;; FP-CALC           ;s.
```

```
        DEFB    $A1             ;;stk-one           ;s,1.
        DEFB    $0F             ;;addition          ;s+1.
        DEFB    $34             ;;stk-data          ;
        DEFB    $37             ;;Exponent: $87,
                                ;;Bytes: 1
        DEFB    $16             ;;(+00,+00,+00)     ;s+1,75.
        DEFB    $04             ;;multiply          ;(s+1)*75 = v
        DEFB    $34             ;;stk-data          ;v.
        DEFB    $80             ;;Bytes: 3
        DEFB    $41             ;;Exponent $91
        DEFB    $00,$00,$80     ;;(+00)             ;v,65537.
        DEFB    $32             ;;n-mod-m           ;remainder, result.
        DEFB    $02             ;;delete            ;remainder.
        DEFB    $A1             ;;stk-one           ;remainder, 1.
        DEFB    $03             ;;subtract          ;remainder - 1. = rnd
        DEFB    $31             ;;duplicate         ;rnd,rnd.
        DEFB    $38             ;;end-calc

        CALL    L2DA2           ; routine FP-TO-BC
        LD      ($5C76),BC      ; store in SEED for next starting point.
        LD      A,(HL)          ; fetch exponent
        AND     A               ; is it zero ?
        JR      Z,L2625         ; forward if so to S-RND-END

        SUB     $10             ; reduce exponent by 2^16
        LD      (HL),A          ; place back

;; S-RND-END
L2625:  JR      L2630           ; forward to S-PI-END

; ---

; the number PI 3.14159...

; ->
;; S-PI
L2627:  CALL    L2530           ; routine SYNTAX-Z
        JR      Z,L2630         ; to S-PI-END if checking syntax.

        RST     28H             ;; FP-CALC
        DEFB    $A3             ;;stk-pi/2                          pi/2.
        DEFB    $38             ;;end-calc

        INC     (HL)            ; increment the exponent leaving pi
                                ; on the calculator stack.

;; S-PI-END
L2630:  RST     20H             ; NEXT-CHAR
        JP      L26C3           ; jump forward to S-NUMERIC

; ---

; ->
;; S-INKEY$
L2634:  LD      BC,$105A        ; priority $10, operation code $1A ('read-in')
                                ; +$40 for string result, numeric operand.
                                ; set this up now in case we need to use the
                                ; calculator.
        RST     20H             ; NEXT-CHAR
        CP      $23             ; '#' ?
        JP      Z,L270D         ; to S-PUSH-PO if so to use the calculator
                                ; single operation
                                ; to read from network/RS232 etc. .
```

```
; else read a key from the keyboard.

        LD      HL,$5C3B        ; fetch FLAGS
        RES     6,(HL)          ; signal string result.
        BIT     7,(HL)          ; checking syntax ?
        JR      Z,L2665         ; forward to S-INK$-EN if so

        CALL    L028E           ; routine KEY-SCAN key in E, shift in D.
        LD      C,$00           ; the length of an empty string
        JR      NZ,L2660        ; to S-IK$-STK to store empty string if
                                ; no key returned.

        CALL    L031E           ; routine K-TEST get main code in A
        JR      NC,L2660        ; to S-IK$-STK to stack null string if
                                ; invalid

        DEC     D               ; D is expected to be FLAGS so set bit 3 $FF
                                ; 'L' Mode so no keywords.
        LD      E,A             ; main key to A
                                ; C is MODE 0 'KLC' from above still.
        CALL    L0333           ; routine K-DECODE
        PUSH    AF              ; save the code
        LD      BC,$0001        ; make room for one character

        RST     30H             ; BC-SPACES
        POP     AF              ; bring the code back
        LD      (DE),A          ; put the key in workspace
        LD      C,$01           ; set C length to one

;; S-IK$-STK
L2660:  LD      B,$00           ; set high byte of length to zero
        CALL    L2AB2           ; routine STK-STO-$

;; S-INK$-EN
L2665:  JP      L2712           ; to S-CONT-2            ===>

; ---

; ->
;; S-SCREEN$
L2668:  CALL    L2522           ; routine S-2-COORD
        CALL    NZ,L2535        ; routine S-SCRN$-S

        RST     20H             ; NEXT-CHAR
        JP      L25DB           ; forward to S-STRING to stack result

; ---

; ->
;; S-ATTR
L2672:  CALL    L2522           ; routine S-2-COORD
        CALL    NZ,L2580        ; routine S-ATTR-S

        RST     20H             ; NEXT-CHAR
        JR      L26C3           ; forward to S-NUMERIC

; ---

; ->
;; S-POINT
L267B:  CALL    L2522           ; routine S-2-COORD
        CALL    NZ,L22CB        ; routine POINT-SUB

        RST     20H             ; NEXT-CHAR
```

```
        JR      L26C3           ; forward to S-NUMERIC

; ----------------------------

; ==> The branch was here if not in table.

;; S-ALPHNUM
L2684:  CALL    L2C88           ; routine ALPHANUM checks if variable or
                                ; a digit.
        JR      NC,L26DF        ; forward to S-NEGATE if not to consider
                                ; a '-' character then functions.

        CP      $41             ; compare 'A'
        JR      NC,L26C9        ; forward to S-LETTER if alpha       ->
                                ; else must have been numeric so continue
                                ; into that routine.

; This important routine is called during runtime and from LINE-SCAN
; when a BASIC line is checked for syntax. It is this routine that
; inserts, during syntax checking, the invisible floating point numbers
; after the numeric expression. During runtime it just picks these
; numbers up. It also handles BIN format numbers.

; ->
;; S-BIN
;; S-DECIMAL
L268D:  CALL    L2530           ; routine SYNTAX-Z
        JR      NZ,L26B5        ; to S-STK-DEC in runtime

; this route is taken when checking syntax.

        CALL    L2C9B           ; routine DEC-TO-FP to evaluate number

        RST     18H             ; GET-CHAR to fetch HL
        LD      BC,$0006        ; six locations required
        CALL    L1655           ; routine MAKE-ROOM
        INC     HL              ; to first new location
        LD      (HL),$0E        ; insert number marker
        INC     HL              ; address next
        EX      DE,HL           ; make DE destination.
        LD      HL,($5C65)      ; STKEND points to end of stack.
        LD      C,$05           ; result is five locations lower
        AND     A               ; prepare for true subtraction
        SBC     HL,BC           ; point to start of value.
        LD      ($5C65),HL      ; update STKEND as we are taking number.
        LDIR                    ; Copy five bytes to program location
        EX      DE,HL           ; transfer pointer to HL
        DEC     HL              ; adjust
        CALL    L0077           ; routine TEMP-PTR1 sets CH-ADD
        JR      L26C3           ; to S-NUMERIC to record nature of result

; ---

; branch here in runtime.

;; S-STK-DEC
L26B5:  RST     18H             ; GET-CHAR positions HL at digit.

;; S-SD-SKIP
L26B6:  INC     HL              ; advance pointer
        LD      A,(HL)          ; until we find
        CP      $0E             ; chr 14d - the number indicator
        JR      NZ,L26B6        ; to S-SD-SKIP until a match
                                ; it has to be here.
```

```
        INC     HL                  ; point to first byte of number
        CALL    L33B4               ; routine STACK-NUM stacks it
        LD      ($5C5D),HL          ; update system variable CH_ADD

;; S-NUMERIC
L26C3:  SET     6,(IY+$01)          ; update FLAGS  - Signal numeric result
        JR      L26DD               ; forward to S-CONT-1               ===>
                                    ; actually S-CONT-2 is destination but why
                                    ; waste a byte on a jump when a JR will do.
                                    ; actually a JR L2712 can be used. Rats.

; end of functions accessed from scanning functions table.

; --------------------------
; Scanning variable routines
; --------------------------
;
;

;; S-LETTER
L26C9:  CALL    L28B2               ; routine LOOK-VARS
        JP      C,L1C2E             ; jump back to REPORT-2 if not found
                                    ; 'Variable not found'
                                    ; but a variable is always 'found' if syntax
                                    ; is being checked.

        CALL    Z,L2996             ; routine STK-VAR considers a subscript/slice
        LD      A,($5C3B)           ; fetch FLAGS value
        CP      $C0                 ; compare 11000000
        JR      C,L26DD             ; step forward to S-CONT-1 if string  ===>

        INC     HL                  ; advance pointer
        CALL    L33B4               ; routine STACK-NUM

;; S-CONT-1
L26DD:  JR      L2712               ; forward to S-CONT-2                 ===>

; ----------------------------------------
; -> the scanning branch was here if not alphanumeric.
; All the remaining functions will be evaluated by a single call to the
; calculator. The correct priority for the operation has to be placed in
; the B register and the operation code, calculator literal in the C register.
; the operation code has bit 7 set if result is numeric and bit 6 is
; set if operand is numeric. so
; $C0 = numeric result, numeric operand.            e.g. 'sin'
; $80 = numeric result, string operand.             e.g. 'code'
; $40 = string result, numeric operand.             e.g. 'str$'
; $00 = string result, string operand.              e.g. 'val$'

;; S-NEGATE
L26DF:  LD      BC,$09DB            ; prepare priority 09, operation code $C0 +
                                    ; 'negate' ($1B) - bits 6 and 7 set for numeric
                                    ; result and numeric operand.

        CP      $2D                 ; is it '-' ?
        JR      Z,L270D             ; forward if so to S-PUSH-PO

        LD      BC,$1018            ; prepare priority $10, operation code 'val$' -
                                    ; bits 6 and 7 reset for string result and
                                    ; string operand.

        CP      $AE                 ; is it 'VAL$' ?
        JR      Z,L270D             ; forward if so to S-PUSH-PO
```

```
        SUB     $AF             ; subtract token 'CODE' value to reduce
                                ; functions 'CODE' to 'NOT' although the
                                ; upper range is, as yet, unchecked.
                                ; valid range would be $00 - $14.

        JP      C,L1C8A         ; jump back to REPORT-C with anything else
                                ; 'Nonsense in BASIC'

        LD      BC,$04F0        ; prepare priority $04, operation $C0 +
                                ; 'not' ($30)

        CP      $14             ; is it 'NOT'
        JR      Z,L270D         ; forward to S-PUSH-PO if so

        JP      NC,L1C8A        ; to REPORT-C if higher
                                ; 'Nonsense in BASIC'

        LD      B,$10           ; priority $10 for all the rest
        ADD     A,$DC           ; make range $DC - $EF
                                ; $C0 + 'code'($1C) thru 'chr$' ($2F)

        LD      C,A             ; transfer 'function' to C
        CP      $DF             ; is it 'sin' ?
        JR      NC,L2707        ; forward to S-NO-TO-$  with 'sin' through
                                ; 'chr$' as operand is numeric.

; all the rest 'cos' through 'chr$' give a numeric result except 'str$'
; and 'chr$'.

        RES     6,C             ; signal string operand for 'code', 'val' and
                                ; 'len'.

;; S-NO-TO-$
L2707:  CP      $EE             ; compare 'str$'
        JR      C,L270D         ; forward to S-PUSH-PO if lower as result
                                ; is numeric.

        RES     7,C             ; reset bit 7 of op code for 'str$', 'chr$'
                                ; as result is string.

; >> This is where they were all headed for.

;; S-PUSH-PO
L270D:  PUSH    BC              ; push the priority and calculator operation
                                ; code.

        RST     20H             ; NEXT-CHAR
        JP      L24FF           ; jump back to S-LOOP-1 to go round the loop
                                ; again with the next character.

; --------------------------------

; ===>  there were many branches forward to here

;; S-CONT-2
L2712:  RST     18H             ; GET-CHAR

;; S-CONT-3
L2713:  CP      $28             ; is it '(' ?
        JR      NZ,L2723        ; forward to S-OPERTR if not    >

        BIT     6,(IY+$01)      ; test FLAGS - numeric or string result ?
        JR      NZ,L2734        ; forward to S-LOOP if numeric to evaluate  >
```

```
; if a string preceded '(' then slice it.

        CALL    L2A52           ; routine SLICING

        RST     20H             ; NEXT-CHAR
        JR      L2713           ; back to S-CONT-3

; ---------------------------

; the branch was here when possibility of an operator '(' has been excluded.

;; S-OPERTR
L2723:  LD      B,$00           ; prepare to add
        LD      C,A             ; possible operator to C
        LD      HL,L2795        ; Address: $2795 - tbl-of-ops
        CALL    L16DC           ; routine INDEXER
        JR      NC,L2734        ; forward to S-LOOP if not in table

; but if found in table the priority has to be looked up.

        LD      C,(HL)          ; operation code to C ( B is still zero )
        LD      HL,L27B0 - $C3  ; $26ED is base of table
        ADD     HL,BC           ; index into table.
        LD      B,(HL)          ; priority to B.

; ------------------
; Scanning main loop
; ------------------
; the juggling act

;; S-LOOP
L2734:  POP     DE              ; fetch last priority and operation
        LD      A,D             ; priority to A
        CP      B               ; compare with this one
        JR      C,L2773         ; forward to S-TIGHTER to execute the
                                ; last operation before this one as it has
                                ; higher priority.

; the last priority was greater or equal this one.

        AND     A               ; if it is zero then so is this
        JP      Z,L0018         ; jump to exit via get-char pointing at
                                ; next character.
                                ; This may be the character after the
                                ; expression or, if exiting a recursive call,
                                ; the next part of the expression to be
                                ; evaluated.

        PUSH    BC              ; save current priority/operation
                                ; as it has lower precedence than the one
                                ; now in DE.

; the 'USR' function is special in that it is overloaded to give two types
; of result.

        LD      HL,$5C3B        ; address FLAGS
        LD      A,E             ; new operation to A register
        CP      $ED             ; is it $C0 + 'usr-no' ($2D)  ?
        JR      NZ,L274C        ; forward to S-STK-LST if not

        BIT     6,(HL)          ; string result expected ?
                                ; (from the lower priority operand we've
                                ; just pushed on stack )
```

```
        JR      NZ,L274C        ; forward to S-STK-LST if numeric
                                ; as operand bits match.

        LD      E,$99           ; reset bit 6 and substitute $19 'usr-$'
                                ; for string operand.

;; S-STK-LST
L274C:  PUSH    DE              ; now stack this priority/operation
        CALL    L2530           ; routine SYNTAX-Z
        JR      Z,L275B         ; forward to S-SYNTEST if checking syntax.

        LD      A,E             ; fetch the operation code
        AND     $3F             ; mask off the result/operand bits to leave
                                ; a calculator literal.
        LD      B,A             ; transfer to B register

; now use the calculator to perform the single operation - operand is on
; the calculator stack.
; Note. although the calculator is performing a single operation most
; functions e.g. TAN are written using other functions and literals and
; these in turn are written using further strings of calculator literals so
; another level of magical recursion joins the juggling act for a while
; as the calculator too is calling itself.

        RST     28H             ;; FP-CALC
        DEFB    $3B             ;;fp-calc-2
L2758:  DEFB    $38             ;;end-calc

        JR      L2764           ; forward to S-RUNTEST

; ---

; the branch was here if checking syntax only.

;; S-SYNTEST
L275B:  LD      A,E             ; fetch the operation code to accumulator
        XOR     (IY+$01)        ; compare with bits of FLAGS
        AND     $40             ; bit 6 will be zero now if operand
                                ; matched expected result.

;; S-RPORT-C2
L2761:  JP      NZ,L1C8A        ; to REPORT-C if mismatch
                                ; 'Nonsense in BASIC'
                                ; else continue to set flags for next

; the branch is to here in runtime after a successful operation.

;; S-RUNTEST
L2764:  POP     DE              ; fetch the last operation from stack
        LD      HL,$5C3B        ; address FLAGS
        SET     6,(HL)          ; set default to numeric result in FLAGS
        BIT     7,E             ; test the operational result
        JR      NZ,L2770        ; forward to S-LOOPEND if numeric

        RES     6,(HL)          ; reset bit 6 of FLAGS to show string result.

;; S-LOOPEND
L2770:  POP     BC              ; fetch the previous priority/operation
        JR      L2734           ; back to S-LOOP to perform these

; ---

; the branch was here when a stacked priority/operator had higher priority
; than the current one.
```

```
;; S-TIGHTER
L2773:  PUSH    DE                ; save high priority op on stack again
        LD      A,C               ; fetch lower priority operation code
        BIT     6,(IY+$01)        ; test FLAGS - Numeric or string result ?
        JR      NZ,L2790          ; forward to S-NEXT if numeric result

; if this is lower priority yet has string then must be a comparison.
; Since these can only be evaluated in context and were defaulted to
; numeric in operator look up they must be changed to string equivalents.

        AND     $3F               ; mask to give true calculator literal
        ADD     A,$08             ; augment numeric literals to string
                                  ; equivalents.
                                  ; 'no-&-no'  => 'str-&-no'
                                  ; 'no-l-eql' => 'str-l-eql'
                                  ; 'no-gr-eq' => 'str-gr-eq'
                                  ; 'nos-neql' => 'strs-neql'
                                  ; 'no-grtr'  => 'str-grtr'
                                  ; 'no-less'  => 'str-less'
                                  ; 'nos-eql'  => 'strs-eql'
                                  ; 'addition' => 'strs-add'
        LD      C,A               ; put modified comparison operator back
        CP      $10               ; is it now 'str-&-no' ?
        JR      NZ,L2788          ; forward to S-NOT-AND  if not.

        SET     6,C               ; set numeric operand bit
        JR      L2790             ; forward to S-NEXT

; ---

;; S-NOT-AND
L2788:  JR      C,L2761           ; back to S-RPORT-C2 if less
                                  ; 'Nonsense in BASIC'.
                                  ; e.g. a$ * b$

        CP      $17               ; is it 'strs-add' ?
        JR      Z,L2790           ; forward to S-NEXT if so
                                  ; (bit 6 and 7 are reset)

        SET     7,C               ; set numeric (Boolean) result for all others

;; S-NEXT
L2790:  PUSH    BC                ; now save this priority/operation on stack

        RST     20H               ; NEXT-CHAR
        JP      L24FF             ; jump back to S-LOOP-1

; ------------------
; Table of operators
; ------------------
; This table is used to look up the calculator literals associated with
; the operator character. The thirteen calculator operations $03 - $0F
; have bits 6 and 7 set to signify a numeric result.
; Some of these codes and bits may be altered later if the context suggests
; a string comparison or operation.
; that is '+', '=', '>', '<', '<=', '>=' or '<>'.

;; tbl-of-ops
L2795:  DEFB    '+', $CF          ;        $C0 + 'addition'
        DEFB    '-', $C3          ;        $C0 + 'subtract'
        DEFB    '*', $C4          ;        $C0 + 'multiply'
        DEFB    '/', $C5          ;        $C0 + 'division'
        DEFB    '^', $C6          ;        $C0 + 'to-power'
```

```
        DEFB    '=', $CE        ;         $C0 + 'nos-eql'
        DEFB    '>', $CC        ;         $C0 + 'no-grtr'
        DEFB    '<', $CD        ;         $C0 + 'no-less'

        DEFB    $C7, $C9        ; '<='    $C0 + 'no-l-eql'
        DEFB    $C8, $CA        ; '>='    $C0 + 'no-gr-eql'
        DEFB    $C9, $CB        ; '<>'    $C0 + 'nos-neql'
        DEFB    $C5, $C7        ; 'OR'    $C0 + 'or'
        DEFB    $C6, $C8        ; 'AND'   $C0 + 'no-&-no'

        DEFB    $00             ; zero end-marker.


; -------------------
; Table of priorities
; -------------------
; This table is indexed with the operation code obtained from the above
; table $C3 - $CF to obtain the priority for the respective operation.

;; tbl-priors
L27B0:  DEFB    $06             ; '-'   opcode $C3
        DEFB    $08             ; '*'   opcode $C4
        DEFB    $08             ; '/'   opcode $C5
        DEFB    $0A             ; '^'   opcode $C6
        DEFB    $02             ; 'OR'  opcode $C7
        DEFB    $03             ; 'AND' opcode $C8
        DEFB    $05             ; '<='  opcode $C9
        DEFB    $05             ; '>='  opcode $CA
        DEFB    $05             ; '<>'  opcode $CB
        DEFB    $05             ; '>'   opcode $CC
        DEFB    $05             ; '<'   opcode $CD
        DEFB    $05             ; '='   opcode $CE
        DEFB    $06             ; '+'   opcode $CF

; ----------------------
; Scanning function (FN)
; ----------------------
; This routine deals with user-defined functions.
; The definition can be anywhere in the program area but these are best
; placed near the start of the program as we shall see.
; The evaluation process is quite complex as the Spectrum has to parse two
; statements at the same time. Syntax of both has been checked previously
; and hidden locations have been created immediately after each argument
; of the DEF FN statement. Each of the arguments of the FN function is
; evaluated by SCANNING and placed in the hidden locations. Then the
; expression to the right of the DEF FN '=' is evaluated by SCANNING and for
; any variables encountered, a search is made in the DEF FN variable list
; in the program area before searching in the normal variables area.
;
; Recursion is not allowed: i.e. the definition of a function should not use
; the same function, either directly or indirectly ( through another function).
; You'll normally get error 4, ('Out of memory'), although sometimes the system
; will crash. - Vickers, Pitman 1984.
;
; As the definition is just an expression, there would seem to be no means
; of breaking out of such recursion.
; However, by the clever use of string expressions and VAL, such recursion is
; possible.
; e.g. DEF FN a(n) = VAL "n+FN a(n-1)+0" ((n<1) * 10 + 1 TO )
; will evaluate the full 11-character expression for all values where n is
; greater than zero but just the 11th character, "0", when n drops to zero
; thereby ending the recursion producing the correct result.
; Recursive string functions are possible using VAL$ instead of VAL and the
; null string as the final addend.
```

```
; - from a turn of the century newsgroup discussion initiated by Mike Wynne.

;; S-FN-SBRN
L27BD:  CALL    L2530           ; routine SYNTAX-Z
        JR      NZ,L27F7        ; forward to SF-RUN in runtime


        RST     20H             ; NEXT-CHAR
        CALL    L2C8D           ; routine ALPHA check for letters A-Z a-z
        JP      NC,L1C8A        ; jump back to REPORT-C if not
                                ; 'Nonsense in BASIC'


        RST     20H             ; NEXT-CHAR
        CP      $24             ; is it '$' ?
        PUSH    AF              ; save character and flags
        JR      NZ,L27D0        ; forward to SF-BRKT-1 with numeric function


        RST     20H             ; NEXT-CHAR

;; SF-BRKT-1
L27D0:  CP      $28             ; is '(' ?
        JR      NZ,L27E6        ; forward to SF-RPRT-C if not
                                ; 'Nonsense in BASIC'


        RST     20H             ; NEXT-CHAR
        CP      $29             ; is it ')' ?
        JR      Z,L27E9         ; forward to SF-FLAG-6 if no arguments.

;; SF-ARGMTS
L27D9:  CALL    L24FB           ; routine SCANNING checks each argument
                                ; which may be an expression.

        RST     18H             ; GET-CHAR
        CP      $2C             ; is it a ',' ?
        JR      NZ,L27E4        ; forward if not to SF-BRKT-2 to test bracket


        RST     20H             ; NEXT-CHAR if a comma was found
        JR      L27D9           ; back to SF-ARGMTS to parse all arguments.

; ---

;; SF-BRKT-2
L27E4:  CP      $29             ; is character the closing ')' ?

;; SF-RPRT-C
L27E6:  JP      NZ,L1C8A        ; jump to REPORT-C
                                ; 'Nonsense in BASIC'

; at this point any optional arguments have had their syntax checked.

;; SF-FLAG-6
L27E9:  RST     20H             ; NEXT-CHAR
        LD      HL,$5C3B        ; address system variable FLAGS
        RES     6,(HL)          ; signal string result
        POP     AF              ; restore test against '$'.
        JR      Z,L27F4         ; forward to SF-SYN-EN if string function.

        SET     6,(HL)          ; signal numeric result

;; SF-SYN-EN
```

```
L27F4:  JP      L2712           ; jump back to S-CONT-2 to continue scanning.

; ---

; the branch was here in runtime.

;; SF-RUN
L27F7:  RST     20H             ; NEXT-CHAR fetches name
        AND     $DF             ; AND 11101111 - reset bit 5 - upper-case.
        LD      B,A             ; save in B

        RST     20H             ; NEXT-CHAR
        SUB     $24             ; subtract '$'
        LD      C,A             ; save result in C
        JR      NZ,L2802        ; forward if not '$' to SF-ARGMT1

        RST     20H             ; NEXT-CHAR advances to bracket

;; SF-ARGMT1
L2802:  RST     20H             ; NEXT-CHAR advances to start of argument
        PUSH    HL              ; save address
        LD      HL,($5C53)      ; fetch start of program area from PROG
        DEC     HL              ; the search starting point is the previous
                                ; location.

;; SF-FND-DF
L2808:  LD      DE,$00CE        ; search is for token 'DEF FN' in E,
                                ; statement count in D.
        PUSH    BC              ; save C the string test, and B the letter.
        CALL    L1D86           ; routine LOOK-PROG will search for token.
        POP     BC              ; restore BC.
        JR      NC,L2814        ; forward to SF-CP-DEF if a match was found.


;; REPORT-P
L2812:  RST     08H             ; ERROR-1
        DEFB    $18             ; Error Report: FN without DEF

;; SF-CP-DEF
L2814:  PUSH    HL              ; save address of DEF FN
        CALL    L28AB           ; routine FN-SKPOVR skips over white-space etc.
                                ; without disturbing CH-ADD.
        AND     $DF             ; make fetched character upper-case.
        CP      B               ; compare with FN name
        JR      NZ,L2825        ; forward to SF-NOT-FD if no match.

; the letters match so test the type.

        CALL    L28AB           ; routine FN-SKPOVR skips white-space
        SUB     $24             ; subtract '$' from fetched character
        CP      C               ; compare with saved result of same operation
                                ; on FN name.
        JR      Z,L2831         ; forward to SF-VALUES with a match.

; the letters matched but one was string and the other numeric.

;; SF-NOT-FD
L2825:  POP     HL              ; restore search point.
        DEC     HL              ; make location before
        LD      DE,$0200        ; the search is to be for the end of the
                                ; current definition - 2 statements forward.
        PUSH    BC              ; save the letter/type
        CALL    L198B           ; routine EACH-STMT steps past rejected
                                ; definition.
```

```
        POP     BC                  ; restore letter/type
        JR      L2808               ; back to SF-FND-DF to continue search

; ---

; Success!
; the branch was here with matching letter and numeric/string type.

;; SF-VALUES
L2831:  AND     A                   ; test A ( will be zero if string '$' - '$' )

        CALL    Z,L28AB             ; routine FN-SKPOVR advances HL past '$'.

        POP     DE                  ; discard pointer to 'DEF FN'.
        POP     DE                  ; restore pointer to first FN argument.
        LD      ($5C5D),DE          ; save in CH_ADD

        CALL    L28AB               ; routine FN-SKPOVR advances HL past '('
        PUSH    HL                  ; save start address in DEF FN  ***
        CP      $29                 ; is character a ')' ?
        JR      Z,L2885             ; forward to SF-R-BR-2 if no arguments.

;; SF-ARG-LP
L2843:  INC     HL                  ; point to next character.
        LD      A,(HL)              ; fetch it.
        CP      $0E                 ; is it the number marker
        LD      D,$40               ; signal numeric in D.
        JR      Z,L2852             ; forward to SF-ARG-VL if numeric.

        DEC     HL                  ; back to letter
        CALL    L28AB               ; routine FN-SKPOVR skips any white-space
        INC     HL                  ; advance past the expected '$' to
                                    ; the 'hidden' marker.
        LD      D,$00               ; signal string.

;; SF-ARG-VL
L2852:  INC     HL                  ; now address first of 5-byte location.
        PUSH    HL                  ; save address in DEF FN statement
        PUSH    DE                  ; save D - result type

        CALL    L24FB               ; routine SCANNING evaluates expression in
                                    ; the FN statement setting FLAGS and leaving
                                    ; result as last value on calculator stack.

        POP     AF                  ; restore saved result type to A

        XOR     (IY+$01)            ; xor with FLAGS
        AND     $40                 ; and with 01000000 to test bit 6
        JR      NZ,L288B            ; forward to REPORT-Q if type mismatch.
                                    ; 'Parameter error'

        POP     HL                  ; pop the start address in DEF FN statement
        EX      DE,HL               ; transfer to DE ?? pop straight into de ?

        LD      HL,($5C65)          ; set HL to STKEND location after value
        LD      BC,$0005            ; five bytes to move
        SBC     HL,BC               ; decrease HL by 5 to point to start.
        LD      ($5C65),HL          ; set STKEND 'removing' value from stack.

        LDIR                        ; copy value into DEF FN statement
        EX      DE,HL               ; set HL to location after value in DEF FN
        DEC     HL                  ; step back one
        CALL    L28AB               ; routine FN-SKPOVR gets next valid character
        CP      $29                 ; is it ')' end of arguments ?
```

```
        JR      Z,L2885         ; forward to SF-R-BR-2 if so.

; a comma separator has been encountered in the DEF FN argument list.

        PUSH    HL              ; save position in DEF FN statement

        RST     18H             ; GET-CHAR from FN statement
        CP      $2C             ; is it ',' ?
        JR      NZ,L288B        ; forward to REPORT-Q if not
                                ; 'Parameter error'

        RST     20H             ; NEXT-CHAR in FN statement advances to next
                                ; argument.

        POP     HL              ; restore DEF FN pointer
        CALL    L28AB           ; routine FN-SKPOVR advances to corresponding
                                ; argument.

        JR      L2843           ; back to SF-ARG-LP looping until all
                                ; arguments are passed into the DEF FN
                                ; hidden locations.

; ---

; the branch was here when all arguments passed.

;; SF-R-BR-2
L2885:  PUSH    HL              ; save location of ')' in DEF FN

        RST     18H             ; GET-CHAR gets next character in FN
        CP      $29             ; is it a ')' also ?
        JR      Z,L288D         ; forward to SF-VALUE if so.


;; REPORT-Q
L288B:  RST     08H             ; ERROR-1
        DEFB    $19             ; Error Report: Parameter error

;; SF-VALUE
L288D:  POP     DE              ; location of ')' in DEF FN to DE.
        EX      DE,HL           ; now to HL, FN ')' pointer to DE.
        LD      ($5C5D),HL      ; initialize CH_ADD to this value.

; At this point the start of the DEF FN argument list is on the machine stack.
; We also have to consider that this defined function may form part of the
; definition of another defined function (though not itself).
; As this defined function may be part of a hierarchy of defined functions
; currently being evaluated by recursive calls to SCANNING, then we have to
; preserve the original value of DEFADD and not assume that it is zero.

        LD      HL,($5C0B)      ; get original DEFADD address
        EX      (SP),HL         ; swap with DEF FN address on stack ***
        LD      ($5C0B),HL      ; set DEFADD to point to this argument list
                                ; during scanning.

        PUSH    DE              ; save FN ')' pointer.

        RST     20H             ; NEXT-CHAR advances past ')' in define

        RST     20H             ; NEXT-CHAR advances past '=' to expression

        CALL    L24FB           ; routine SCANNING evaluates but searches
                                ; initially for variables at DEFADD
```

```
        POP     HL              ; pop the FN ')' pointer
        LD      ($5C5D),HL      ; set CH_ADD to this
        POP     HL              ; pop the original DEFADD value
        LD      ($5C0B),HL      ; and re-insert into DEFADD system variable.

        RST     20H             ; NEXT-CHAR advances to character after ')'
        JP      L2712           ; to S-CONT-2 - to continue current
                                ; invocation of scanning

; --------------------
; Used to parse DEF FN
; --------------------
; e.g. DEF FN     s $ ( x )     =  b     $ (  TO  x  ) : REM exaggerated
;
; This routine is used 10 times to advance along a DEF FN statement
; skipping spaces and colour control codes. It is similar to NEXT-CHAR
; which is, at the same time, used to skip along the corresponding FN function
; except the latter has to deal with AT and TAB characters in string
; expressions. These cannot occur in a program area so this routine is
; simpler as both colour controls and their parameters are less than space.

;; FN-SKPOVR
L28AB:  INC     HL              ; increase pointer
        LD      A,(HL)          ; fetch addressed character
        CP      $21             ; compare with space + 1
        JR      C,L28AB         ; back to FN-SKPOVR if less

        RET                     ; return pointing to a valid character.

; ---------
; LOOK-VARS
; ---------
;
;

;; LOOK-VARS
L28B2:  SET     6,(IY+$01)      ; update FLAGS - presume numeric result

        RST     18H             ; GET-CHAR
        CALL    L2C8D           ; routine ALPHA tests for A-Za-z
        JP      NC,L1C8A        ; jump to REPORT-C if not.
                                ; 'Nonsense in BASIC'

        PUSH    HL              ; save pointer to first letter       ^1
        AND     $1F             ; mask lower bits, 1 - 26 decimal     000xxxxx
        LD      C,A             ; store in C.

        RST     20H             ; NEXT-CHAR
        PUSH    HL              ; save pointer to second character   ^2
        CP      $28             ; is it '(' - an array ?
        JR      Z,L28EF         ; forward to V-RUN/SYN if so.

        SET     6,C             ; set 6 signaling string if solitary  010
        CP      $24             ; is character a '$' ?
        JR      Z,L28DE         ; forward to V-STR-VAR

        SET     5,C             ; signal numeric                       011
        CALL    L2C88           ; routine ALPHANUM sets carry if second
                                ; character is alphanumeric.
        JR      NC,L28E3        ; forward to V-TEST-FN if just one character

; It is more than one character but re-test current character so that 6 reset
; This loop renders the similar loop at V-PASS redundant.
```

```
;; V-CHAR
L28D4:  CALL    L2C88           ; routine ALPHANUM
        JR      NC,L28EF        ; to V-RUN/SYN when no more

        RES     6,C             ; make long named type                 001

        RST     20H             ; NEXT-CHAR
        JR      L28D4           ; loop back to V-CHAR

; ---


;; V-STR-VAR
L28DE:  RST     20H             ; NEXT-CHAR advances past '$'
        RES     6,(IY+$01)      ; update FLAGS - signal string result.

;; V-TEST-FN
L28E3:  LD      A,($5C0C)       ; load A with DEFADD_hi
        AND     A               ; and test for zero.
        JR      Z,L28EF         ; forward to V-RUN/SYN if a defined function
                                ; is not being evaluated.

; Note.

        CALL    L2530           ; routine SYNTAX-Z
        JP      NZ,L2951        ; JUMP to STK-F-ARG in runtime and then
                                ; back to this point if no variable found.

;; V-RUN/SYN
L28EF:  LD      B,C             ; save flags in B
        CALL    L2530           ; routine SYNTAX-Z
        JR      NZ,L28FD        ; to V-RUN to look for the variable in runtime

; if checking syntax the letter is not returned

        LD      A,C             ; copy letter/flags to A
        AND     $E0             ; and with 11100000 to get rid of the letter
        SET     7,A             ; use spare bit to signal checking syntax.
        LD      C,A             ; and transfer to C.
        JR      L2934           ; forward to V-SYNTAX

; ---

; but in runtime search for the variable.

;; V-RUN
L28FD:  LD      HL,($5C4B)      ; set HL to start of variables from VARS

;; V-EACH
L2900:  LD      A,(HL)          ; get first character
        AND     $7F             ; and with 01111111
                                ; ignoring bit 7 which distinguishes
                                ; arrays or for/next variables.

        JR      Z,L2932         ; to V-80-BYTE if zero as must be 10000000
                                ; the variables end-marker.

        CP      C               ; compare with supplied value.
        JR      NZ,L292A        ; forward to V-NEXT if no match.

        RLA                     ; destructively test
        ADD     A,A             ; bits 5 and 6 of A
                                ; jumping if bit 5 reset or 6 set
```

```
        JP      P,L293F         ; to V-FOUND-2  strings and arrays

        JR      C,L293F         ; to V-FOUND-2  simple and for next

; leaving long name variables.

        POP     DE              ; pop pointer to 2nd. char
        PUSH    DE              ; save it again
        PUSH    HL              ; save variable first character pointer

;; V-MATCHES
L2912:  INC     HL              ; address next character in vars area

;; V-SPACES
L2913:  LD      A,(DE)          ; pick up letter from prog area
        INC     DE              ; and advance address
        CP      $20             ; is it a space
        JR      Z,L2913         ; back to V-SPACES until non-space

        OR      $20             ; convert to range 1 - 26.
        CP      (HL)            ; compare with addressed variables character
        JR      Z,L2912         ; loop back to V-MATCHES if a match on an
                                ; intermediate letter.

        OR      $80             ; now set bit 7 as last character of long
                                ; names are inverted.
        CP      (HL)            ; compare again
        JR      NZ,L2929        ; forward to V-GET-PTR if no match

; but if they match check that this is also last letter in prog area

        LD      A,(DE)          ; fetch next character
        CALL    L2C88           ; routine ALPHANUM sets carry if not alphanum
        JR      NC,L293E        ; forward to V-FOUND-1 with a full match.

;; V-GET-PTR
L2929:  POP     HL              ; pop saved pointer to char 1

;; V-NEXT
L292A:  PUSH    BC              ; save flags
        CALL    L19B8           ; routine NEXT-ONE gets next variable in DE
        EX      DE,HL           ; transfer to HL.
        POP     BC              ; restore the flags
        JR      L2900           ; loop back to V-EACH
                                ; to compare each variable

; ---

;; V-80-BYTE
L2932:  SET     7,B             ; will signal not found

; the branch was here when checking syntax

;; V-SYNTAX
L2934:  POP     DE              ; discard the pointer to 2nd. character  v2
                                ; in BASIC line/workspace.

        RST     18H             ; GET-CHAR gets character after variable name.
        CP      $28             ; is it '(' ?
        JR      Z,L2943         ; forward to V-PASS
                                ; Note. could go straight to V-END ?

        SET     5,B             ; signal not an array
        JR      L294B           ; forward to V-END
```

```
; --------------------------

; the jump was here when a long name matched and HL pointing to last character
; in variables area.

;; V-FOUND-1
L293E:  POP     DE              ; discard pointer to first var letter

; the jump was here with all other matches HL points to first var char.

;; V-FOUND-2
L293F:  POP     DE              ; discard pointer to 2nd prog char       v2
        POP     DE              ; drop pointer to 1st prog char          v1
        PUSH    HL              ; save pointer to last char in vars

        RST     18H             ; GET-CHAR

;; V-PASS
L2943:  CALL    L2C88           ; routine ALPHANUM
        JR      NC,L294B        ; forward to V-END if not

; but it never will be as we advanced past long-named variables earlier.

        RST     20H             ; NEXT-CHAR
        JR      L2943           ; back to V-PASS

; ---

;; V-END
L294B:  POP     HL              ; pop the pointer to first character in
                                ; BASIC line/workspace.
        RL      B               ; rotate the B register left
                                ; bit 7 to carry
        BIT     6,B             ; test the array indicator bit.
        RET                     ; return

; -----------------------
; Stack function argument
; -----------------------
; This branch is taken from LOOK-VARS when a defined function is currently
; being evaluated.
; Scanning is evaluating the expression after the '=' and the variable
; found could be in the argument list to the left of the '=' or in the
; normal place after the program. Preference will be given to the former.
; The variable name to be matched is in C.

;; STK-F-ARG
L2951:  LD      HL,($5C0B)      ; set HL to DEFADD
        LD      A,(HL)          ; load the first character
        CP      $29             ; is it ')' ?
        JP      Z,L28EF         ; JUMP back to V-RUN/SYN, if so, as there are
                                ; no arguments.

; but proceed to search argument list of defined function first if not empty.

;; SFA-LOOP
L295A:  LD      A,(HL)          ; fetch character again.
        OR      $60             ; or with 01100000 presume a simple variable.
        LD      B,A             ; save result in B.
        INC     HL              ; address next location.
        LD      A,(HL)          ; pick up byte.
        CP      $0E             ; is it the number marker ?
        JR      Z,L296B         ; forward to SFA-CP-VR if so.
```

```
; it was a string. White-space may be present but syntax has been checked.

        DEC     HL              ; point back to letter.
        CALL    L28AB           ; routine FN-SKPOVR skips to the '$'
        INC     HL              ; now address the hidden marker.
        RES     5,B             ; signal a string variable.

;; SFA-CP-VR
L296B:  LD      A,B             ; transfer found variable letter to A.
        CP      C               ; compare with expected.
        JR      Z,L2981         ; forward to SFA-MATCH with a match.

        INC     HL              ; step
        INC     HL              ; past
        INC     HL              ; the
        INC     HL              ; five
        INC     HL              ; bytes.

        CALL    L28AB           ; routine FN-SKPOVR skips to next character
        CP      $29             ; is it ')' ?
        JP      Z,L28EF         ; jump back if so to V-RUN/SYN to look in
                                ; normal variables area.

        CALL    L28AB           ; routine FN-SKPOVR skips past the ','
                                ; all syntax has been checked and these
                                ; things can be taken as read.
        JR      L295A           ; back to SFA-LOOP while there are more
                                ; arguments.

; ---

;; SFA-MATCH
L2981:  BIT     5,C             ; test if numeric
        JR      NZ,L2991        ; to SFA-END if so as will be stacked
                                ; by scanning

        INC     HL              ; point to start of string descriptor
        LD      DE,($5C65)      ; set DE to STKEND
        CALL    L33C0           ; routine MOVE-FP puts parameters on stack.
        EX      DE,HL           ; new free location to HL.
        LD      ($5C65),HL      ; use it to set STKEND system variable.

;; SFA-END
L2991:  POP     DE              ; discard
        POP     DE              ; pointers.
        XOR     A               ; clear carry flag.
        INC     A               ; and zero flag.
        RET                     ; return.

; ------------------------
; Stack variable component
; ------------------------
; This is called to evaluate a complex structure that has been found, in
; runtime, by LOOK-VARS in the variables area.
; In this case HL points to the initial letter, bits 7-5
; of which indicate the type of variable.
; 010 - simple string, 110 - string array, 100 - array of numbers.
;
; It is called from CLASS-01 when assigning to a string or array including
; a slice.
; It is called from SCANNING to isolate the required part of the structure.
;
; An important part of the runtime process is to check that the number of
```

```
; dimensions of the variable match the number of subscripts supplied in the
; BASIC line.
;
; If checking syntax,
; the B register, which counts dimensions is set to zero (256) to allow
; the loop to continue till all subscripts are checked. While doing this it
; is reading dimension sizes from some arbitrary area of memory. Although
; these are meaningless it is of no concern as the limit is never checked by
; int-exp during syntax checking.
;
; The routine is also called from the syntax path of DIM command to check the
; syntax of both string and numeric arrays definitions except that bit 6 of C
; is reset so both are checked as numeric arrays. This ruse avoids a terminal
; slice being accepted as part of the DIM command.
; All that is being checked is that there are a valid set of comma-separated
; expressions before a terminal ')', although, as above, it will still go
; through the motions of checking dummy dimension sizes.

;; STK-VAR
L2996:  XOR     A               ; clear A
        LD      B,A             ; and B, the syntax dimension counter (256)
        BIT     7,C             ; checking syntax ?
        JR      NZ,L29E7        ; forward to SV-COUNT if so.

; runtime evaluation.

        BIT     7,(HL)          ; will be reset if a simple string.
        JR      NZ,L29AE        ; forward to SV-ARRAYS otherwise

        INC     A               ; set A to 1, simple string.

;; SV-SIMPLE$
L29A1:  INC     HL              ; address length low
        LD      C,(HL)          ; place in C
        INC     HL              ; address length high
        LD      B,(HL)          ; place in B
        INC     HL              ; address start of string
        EX      DE,HL           ; DE = start now.
        CALL    L2AB2           ; routine STK-STO-$ stacks string parameters
                                ; DE start in variables area,
                                ; BC length, A=1 simple string

; the only thing now is to consider if a slice is required.

        RST     18H             ; GET-CHAR puts character at CH_ADD in A
        JP      L2A49           ; jump forward to SV-SLICE? to test for '('

; --------------------------------------------------------

; the branch was here with string and numeric arrays in runtime.

;; SV-ARRAYS
L29AE:  INC     HL              ; step past
        INC     HL              ; the total length
        INC     HL              ; to address Number of dimensions.
        LD      B,(HL)          ; transfer to B overwriting zero.
        BIT     6,C             ; a numeric array ?
        JR      Z,L29C0         ; forward to SV-PTR with numeric arrays

        DEC     B               ; ignore the final element of a string array
                                ; the fixed string size.

        JR      Z,L29A1         ; back to SV-SIMPLE$ if result is zero as has
                                ; been created with DIM a$(10) for instance
```

```
                                ; and can be treated as a simple string.

; proceed with multi-dimensioned string arrays in runtime.

        EX      DE,HL           ; save pointer to dimensions in DE

        RST     18H             ; GET-CHAR looks at the BASIC line
        CP      $28             ; is character '(' ?
        JR      NZ,L2A20        ; to REPORT-3 if not
                                ; 'Subscript wrong'

        EX      DE,HL           ; dimensions pointer to HL to synchronize
                                ; with next instruction.

; runtime numeric arrays path rejoins here.

;; SV-PTR
L29C0:  EX      DE,HL           ; save dimension pointer in DE
        JR      L29E7           ; forward to SV-COUNT with true no of dims
                                ; in B. As there is no initial comma the
                                ; loop is entered at the midpoint.

; ------------------------------------------------------------
; the dimension counting loop which is entered at mid-point.

;; SV-COMMA
L29C3:  PUSH    HL              ; save counter

        RST     18H             ; GET-CHAR

        POP     HL              ; pop counter
        CP      $2C             ; is character ',' ?
        JR      Z,L29EA         ; forward to SV-LOOP if so

; in runtime the variable definition indicates a comma should appear here

        BIT     7,C             ; checking syntax ?
        JR      Z,L2A20         ; forward to REPORT-3 if not
                                ; 'Subscript error'

; proceed if checking syntax of an array?

        BIT     6,C             ; array of strings
        JR      NZ,L29D8        ; forward to SV-CLOSE if so

; an array of numbers.

        CP      $29             ; is character ')' ?
        JR      NZ,L2A12        ; forward to SV-RPT-C if not
                                ; 'Nonsense in BASIC'

        RST     20H             ; NEXT-CHAR moves CH-ADD past the statement
        RET                     ; return ->

; ---

; the branch was here with an array of strings.

;; SV-CLOSE
L29D8:  CP      $29             ; as above ')' could follow the expression
        JR      Z,L2A48         ; forward to SV-DIM if so

        CP      $CC             ; is it 'TO' ?
        JR      NZ,L2A12        ; to SV-RPT-C with anything else
```

```
                                ; 'Nonsense in BASIC'

; now backtrack CH_ADD to set up for slicing routine.
; Note. in a BASIC line we can safely backtrack to a colour parameter.

;; SV-CH-ADD
L29E0:  RST     18H             ; GET-CHAR
        DEC     HL              ; backtrack HL
        LD      ($5C5D),HL      ; to set CH_ADD up for slicing routine
        JR      L2A45           ; forward to SV-SLICE and make a return
                                ; when all slicing complete.

; ----------------------------------------
; -> the mid-point entry point of the loop

;; SV-COUNT
L29E7:  LD      HL,$0000        ; initialize data pointer to zero.

;; SV-LOOP
L29EA:  PUSH    HL              ; save the data pointer.

        RST     20H             ; NEXT-CHAR in BASIC area points to an
                                ; expression.

        POP     HL              ; restore the data pointer.
        LD      A,C             ; transfer name/type to A.
        CP      $C0             ; is it 11000000 ?
                                ; Note. the letter component is absent if
                                ; syntax checking.
        JR      NZ,L29FB        ; forward to SV-MULT if not an array of
                                ; strings.

; proceed to check string arrays during syntax.

        RST     18H             ; GET-CHAR
        CP      $29             ; ')'  end of subscripts ?
        JR      Z,L2A48         ; forward to SV-DIM to consider further slice

        CP      $CC             ; is it 'TO' ?
        JR      Z,L29E0         ; back to SV-CH-ADD to consider a slice.
                                ; (no need to repeat get-char at L29E0)

; if neither, then an expression is required so rejoin runtime loop ??
; registers HL and DE only point to somewhere meaningful in runtime so
; comments apply to that situation.

;; SV-MULT
L29FB:  PUSH    BC              ; save dimension number.
        PUSH    HL              ; push data pointer/rubbish.
                                ; DE points to current dimension.
        CALL    L2AEE           ; routine DE,(DE+1) gets next dimension in DE
                                ; and HL points to it.
        EX      (SP),HL         ; dim pointer to stack, data pointer to HL (*)
        EX      DE,HL           ; data pointer to DE, dim size to HL.

        CALL    L2ACC           ; routine INT-EXP1 checks integer expression
                                ; and gets result in BC in runtime.
        JR      C,L2A20         ; to REPORT-3 if > HL
                                ; 'Subscript out of range'

        DEC     BC              ; adjust returned result from 1-x to 0-x
        CALL    L2AF4           ; routine GET-HL*DE multiplies data pointer by
                                ; dimension size.
        ADD     HL,BC           ; add the integer returned by expression.
```

```
        POP     DE                  ; pop the dimension pointer.
***
        POP     BC                  ; pop dimension counter.
        DJNZ    L29C3               ; back to SV-COMMA if more dimensions
                                    ; Note. during syntax checking, unless there
                                    ; are more than 256 subscripts, the branch
                                    ; back to SV-COMMA is always taken.

        BIT     7,C                 ; are we checking syntax ?
                                    ; then we've got a joker here.

;; SV-RPT-C
L2A12:  JR      NZ,L2A7A            ; forward to SL-RPT-C if so
                                    ; 'Nonsense in BASIC'
                                    ; more than 256 subscripts in BASIC line.

; but in runtime the number of subscripts are at least the same as dims

        PUSH    HL                  ; save data pointer.
        BIT     6,C                 ; is it a string array ?
        JR      NZ,L2A2C            ; forward to SV-ELEM$ if so.

; a runtime numeric array subscript.

        LD      B,D                 ; register DE has advanced past all dimensions
        LD      C,E                 ; and points to start of data in variable.
                                    ; transfer it to BC.

        RST     18H                 ; GET-CHAR checks BASIC line
        CP      $29                 ; must be a ')' ?
        JR      Z,L2A22             ; skip to SV-NUMBER if so

; else more subscripts in BASIC line than the variable definition.

;; REPORT-3
L2A20:  RST     08H                 ; ERROR-1
        DEFB    $02                 ; Error Report: Subscript wrong

; continue if subscripts matched the numeric array.

;; SV-NUMBER
L2A22:  RST     20H                 ; NEXT-CHAR moves CH_ADD to next statement
                                    ; - finished parsing.

        POP     HL                  ; pop the data pointer.
        LD      DE,$0005            ; each numeric element is 5 bytes.
        CALL    L2AF4               ; routine GET-HL*DE multiplies.
        ADD     HL,BC               ; now add to start of data in the variable.

        RET                         ; return with HL pointing at the numeric
                                    ; array subscript.                    ->

; ---------------------------------------------------------------

; the branch was here for string subscripts when the number of subscripts
; in the BASIC line was one less than in variable definition.

;; SV-ELEM$
L2A2C:  CALL    L2AEE               ; routine DE,(DE+1) gets final dimension
                                    ; the length of strings in this array.
        EX      (SP),HL             ; start pointer to stack, data pointer to HL.
        CALL    L2AF4               ; routine GET-HL*DE multiplies by element
                                    ; size.
        POP     BC                  ; the start of data pointer is added
```

```
        ADD     HL,BC           ; in - now points to location before.
        INC     HL              ; point to start of required string.
        LD      B,D             ; transfer the length (final dimension size)
        LD      C,E             ; from DE to BC.
        EX      DE,HL           ; put start in DE.
        CALL    L2AB1           ; routine STK-ST-0 stores the string parameters
                                ; with A=0 - a slice or subscript.

; now check that there were no more subscripts in the BASIC line.

        RST     18H             ; GET-CHAR
        CP      $29             ; is it ')' ?
        JR      Z,L2A48         ; forward to SV-DIM to consider a separate
                                ; subscript or/and a slice.

        CP      $2C             ; a comma is allowed if the final subscript
                                ; is to be sliced e.g. a$(2,3,4 TO 6).
        JR      NZ,L2A20        ; to REPORT-3 with anything else
                                ; 'Subscript error'

;; SV-SLICE
L2A45:  CALL    L2A52           ; routine SLICING slices the string.

; but a slice of a simple string can itself be sliced.

;; SV-DIM
L2A48:  RST     20H             ; NEXT-CHAR

;; SV-SLICE?
L2A49:  CP      $28             ; is character '(' ?
        JR      Z,L2A45         ; loop back if so to SV-SLICE

        RES     6,(IY+$01)      ; update FLAGS  - Signal string result
        RET                     ; and return.

; ---

; The above section deals with the flexible syntax allowed.
; DIM a$(3,3,10) can be considered as two dimensional array of ten-character
; strings or a 3-dimensional array of characters.
; a$(1,1) will return a 10-character string as will a$(1,1,1 TO 10)
; a$(1,1,1) will return a single character.
; a$(1,1) (1 TO 6) is the same as a$(1,1,1 TO 6)
; A slice can itself be sliced ad infinitum
; b$ () () () () () () (2 TO 10) (2 TO 9) (3) is the same as b$(5)



; -------------------------
; Handle slicing of strings
; -------------------------
; The syntax of string slicing is very natural and it is as well to reflect
; on the permutations possible.
; a$() and a$( TO ) indicate the entire string although just a$ would do
; and would avoid coming here.
; h$(16) indicates the single character at position 16.
; a$( TO 32) indicates the first 32 characters.
; a$(257 TO) indicates all except the first 256 characters.
; a$(19000 TO 19999) indicates the thousand characters at position 19000.
; Also a$(9 TO 5) returns a null string not an error.
; This enables a$(2 TO) to return a null string if the passed string is
; of length zero or 1.
; A string expression in brackets can be sliced. e.g. (STR$ PI) (3 TO )
; We arrived here from SCANNING with CH-ADD pointing to the initial '('
```

```
; or from above.

;; SLICING
L2A52:  CALL    L2530           ; routine SYNTAX-Z
        CALL    NZ,L2BF1        ; routine STK-FETCH fetches parameters of
                                ; string at runtime, start in DE, length
                                ; in BC. This could be an array subscript.

        RST     20H             ; NEXT-CHAR
        CP      $29             ; is it ')' ?     e.g. a$()
        JR      Z,L2AAD         ; forward to SL-STORE to store entire string.

        PUSH    DE              ; else save start address of string

        XOR     A               ; clear accumulator to use as a running flag.
        PUSH    AF              ; and save on stack before any branching.

        PUSH    BC              ; save length of string to be sliced.
        LD      DE,$0001        ; default the start point to position 1.

        RST     18H             ; GET-CHAR

        POP     HL              ; pop length to HL as default end point
                                ; and limit.

        CP      $CC             ; is it 'TO' ?    e.g. a$( TO 10000)
        JR      Z,L2A81         ; to SL-SECOND to evaluate second parameter.

        POP     AF              ; pop the running flag.

        CALL    L2ACD           ; routine INT-EXP2 fetches first parameter.

        PUSH    AF              ; save flag (will be $FF if parameter>limit)

        LD      D,B             ; transfer the start
        LD      E,C             ; to DE overwriting 0001.
        PUSH    HL              ; save original length.

        RST     18H             ; GET-CHAR
        POP     HL              ; pop the limit length.
        CP      $CC             ; is it 'TO' after a start ?
        JR      Z,L2A81         ; to SL-SECOND to evaluate second parameter

        CP      $29             ; is it ')' ?       e.g. a$(365)

;; SL-RPT-C
L2A7A:  JP      NZ,L1C8A        ; jump to REPORT-C with anything else
                                ; 'Nonsense in BASIC'

        LD      H,D             ; copy start
        LD      L,E             ; to end - just a one character slice.
        JR      L2A94           ; forward to SL-DEFINE.

; ---------------------

;; SL-SECOND
L2A81:  PUSH    HL              ; save limit length.

        RST     20H             ; NEXT-CHAR

        POP     HL              ; pop the length.

        CP      $29             ; is character ')' ?        e.g. a$(7 TO )
        JR      Z,L2A94         ; to SL-DEFINE using length as end point.
```

```
        POP     AF              ; else restore flag.
        CALL    L2ACD           ; routine INT-EXP2 gets second expression.

        PUSH    AF              ; save the running flag.

        RST     18H             ; GET-CHAR

        LD      H,B             ; transfer second parameter
        LD      L,C             ; to HL.              e.g. a$(42 to 99)
        CP      $29             ; is character a ')' ?
        JR      NZ,L2A7A        ; to SL-RPT-C if not
                                ; 'Nonsense in BASIC'

; we now have start in DE and an end in HL.

;; SL-DEFINE
L2A94:  POP     AF              ; pop the running flag.
        EX      (SP),HL         ; put end point on stack, start address to HL
        ADD     HL,DE           ; add address of string to the start point.
        DEC     HL              ; point to first character of slice.
        EX      (SP),HL         ; start address to stack, end point to HL (*)
        AND     A               ; prepare to subtract.
        SBC     HL,DE           ; subtract start point from end point.
        LD      BC,$0000        ; default the length result to zero.
        JR      C,L2AA8         ; forward to SL-OVER if start > end.

        INC     HL              ; increment the length for inclusive byte.

        AND     A               ; now test the running flag.
        JP      M,L2A20         ; jump back to REPORT-3 if $FF.
                                ; 'Subscript out of range'

        LD      B,H             ; transfer the length
        LD      C,L             ; to BC.

;; SL-OVER
L2AA8:  POP     DE              ; restore start address from machine stack ***
        RES     6,(IY+$01)      ; update FLAGS - signal string result for
                                ; syntax.

;; SL-STORE
L2AAD:  CALL    L2530           ; routine SYNTAX-Z  (UNSTACK-Z?)
        RET     Z               ; return if checking syntax.
                                ; but continue to store the string in runtime.

; ------------------------------------
; other than from above, this routine is called from STK-VAR to stack
; a known string array element.
; ------------------------------------

;; STK-ST-0
L2AB1:  XOR     A               ; clear to signal a sliced string or element.

; -------------------------
; this routine is called from chr$, scrn$ etc. to store a simple string result.
; --------------------------

;; STK-STO-$
L2AB2:  RES     6,(IY+$01)      ; update FLAGS - signal string result.
                                ; and continue to store parameters of string.

; ---------------------------------------
; Pass five registers to calculator stack
```

```
; ----------------------------------------
; This subroutine puts five registers on the calculator stack.

;; STK-STORE
L2AB6:  PUSH    BC              ; save two registers
        CALL    L33A9           ; routine TEST-5-SP checks room and puts 5
                                ; in BC.
        POP     BC              ; fetch the saved registers.
        LD      HL,($5C65)      ; make HL point to first empty location STKEND
        LD      (HL),A          ; place the 5 registers.
        INC     HL              ;
        LD      (HL),E          ;
        INC     HL              ;
        LD      (HL),D          ;
        INC     HL              ;
        LD      (HL),C          ;
        INC     HL              ;
        LD      (HL),B          ;
        INC     HL              ;
        LD      ($5C65),HL      ; update system variable STKEND.
        RET                     ; and return.

; -------------------------------------------
; Return result of evaluating next expression
; -------------------------------------------
; This clever routine is used to check and evaluate an integer expression
; which is returned in BC, setting A to $FF, if greater than a limit supplied
; in HL. It is used to check array subscripts, parameters of a string slice
; and the arguments of the DIM command. In the latter case, the limit check
; is not required and H is set to $FF. When checking optional string slice
; parameters, it is entered at the second entry point so as not to disturb
; the running flag A, which may be $00 or $FF from a previous invocation.

;; INT-EXP1
L2ACC:  XOR     A               ; set result flag to zero.

; -> The entry point is here if A is used as a running flag.

;; INT-EXP2
L2ACD:  PUSH    DE              ; preserve DE register throughout.
        PUSH    HL              ; save the supplied limit.
        PUSH    AF              ; save the flag.

        CALL    L1C82           ; routine EXPT-1NUM evaluates expression
                                ; at CH_ADD returning if numeric result,
                                ; with value on calculator stack.

        POP     AF              ; pop the flag.
        CALL    L2530           ; routine SYNTAX-Z
        JR      Z,L2AEB         ; forward to I-RESTORE if checking syntax so
                                ; avoiding a comparison with supplied limit.

        PUSH    AF              ; save the flag.

        CALL    L1E99           ; routine FIND-INT2 fetches value from
                                ; calculator stack to BC producing an error
                                ; if too high.

        POP     DE              ; pop the flag to D.
        LD      A,B             ; test value for zero and reject
        OR      C               ; as arrays and strings begin at 1.
        SCF                     ; set carry flag.
        JR      Z,L2AE8         ; forward to I-CARRY if zero.
```

```
        POP     HL              ; restore the limit.
        PUSH    HL              ; and save.
        AND     A               ; prepare to subtract.
        SBC     HL,BC           ; subtract value from limit.

;; I-CARRY
L2AE8:  LD      A,D             ; move flag to accumulator $00 or $FF.
        SBC     A,$00           ; will set to $FF if carry set.

;; I-RESTORE
L2AEB:  POP     HL              ; restore the limit.
        POP     DE              ; and DE register.
        RET                     ; return.


; -----------------------
; LD DE,(DE+1) Subroutine
; -----------------------
; This routine just loads the DE register with the contents of the two
; locations following the location addressed by DE.
; It is used to step along the 16-bit dimension sizes in array definitions.
; Note. Such code is made into subroutines to make programs easier to
; write and it would use less space to include the five instructions in-line.
; However, there are so many exchanges going on at the places this is invoked
; that to implement it in-line would make the code hard to follow.
; It probably had a zippier label though as the intention is to simplify the
; program.

;; DE,(DE+1)
L2AEE:  EX      DE,HL           ;
        INC     HL              ;
        LD      E,(HL)          ;
        INC     HL              ;
        LD      D,(HL)          ;
        RET                     ;

; -------------------
; HL=HL*DE Subroutine
; -------------------
; This routine calls the mathematical routine to multiply HL by DE in runtime.
; It is called from STK-VAR and from DIM. In the latter case syntax is not
; being checked so the entry point could have been at the second CALL
; instruction to save a few clock-cycles.

;; GET-HL*DE
L2AF4:  CALL    L2530           ; routine SYNTAX-Z.
        RET     Z               ; return if checking syntax.

        CALL    L30A9           ; routine HL-HL*DE.
        JP      C,L1F15         ; jump back to REPORT-4 if over 65535.

        RET                     ; else return with 16-bit result in HL.

; -----------------
; THE 'LET' COMMAND
; -----------------
; Sinclair BASIC adheres to the ANSI-78 standard and a LET is required in
; assignments e.g. LET a = 1  :   LET h$ = "hat".
;
; Long names may contain spaces but not colour controls (when assigned).
; a substring can appear to the left of the equals sign.

; An earlier mathematician Lewis Carroll may have been pleased that
; 10 LET Babies cannot manage crocodiles = Babies are illogical AND
```

```
;    Nobody is despised who can manage a crocodile AND Illogical persons
;    are despised
; does not give the 'Nonsense..' error if the three variables exist.
; I digress.

;; LET
L2AFF:  LD      HL,($5C4D)      ; fetch system variable DEST to HL.
        BIT     1,(IY+$37)      ; test FLAGX - handling a new variable ?
        JR      Z,L2B66         ; forward to L-EXISTS if not.

; continue for a new variable. DEST points to start in BASIC line.
; from the CLASS routines.

        LD      BC,$0005        ; assume numeric and assign an initial 5 bytes

;; L-EACH-CH
L2B0B:  INC     BC              ; increase byte count for each relevant
                                ; character

;; L-NO-SP
L2B0C:  INC     HL              ; increase pointer.
        LD      A,(HL)          ; fetch character.
        CP      $20             ; is it a space ?
        JR      Z,L2B0C         ; back to L-NO-SP is so.

        JR      NC,L2B1F        ; forward to L-TEST-CH if higher.

        CP      $10             ; is it $00 - $0F ?
        JR      C,L2B29         ; forward to L-SPACES if so.

        CP      $16             ; is it $16 - $1F ?
        JR      NC,L2B29        ; forward to L-SPACES if so.

; it was $10 - $15  so step over a colour code.

        INC     HL              ; increase pointer.
        JR      L2B0C           ; loop back to L-NO-SP.

; ---

; the branch was to here if higher than space.

;; L-TEST-CH
L2B1F:  CALL    L2C88           ; routine ALPHANUM sets carry if alphanumeric
        JR      C,L2B0B         ; loop back to L-EACH-CH for more if so.

        CP      $24             ; is it '$' ?
        JP      Z,L2BC0         ; jump forward if so, to L-NEW$
                                ; with a new string.

;; L-SPACES
L2B29:  LD      A,C             ; save length lo in A.
        LD      HL,($5C59)      ; fetch E_LINE to HL.
        DEC     HL              ; point to location before, the variables
                                ; end-marker.
        CALL    L1655           ; routine MAKE-ROOM creates BC spaces
                                ; for name and numeric value.
        INC     HL              ; advance to first new location.
        INC     HL              ; then to second.
        EX      DE,HL           ; set DE to second location.
        PUSH    DE              ; save this pointer.
        LD      HL,($5C4D)      ; reload HL with DEST.
        DEC     DE              ; point to first.
        SUB     $06             ; subtract six from length_lo.
```

```
        LD      B,A             ; save count in B.
        JR      Z,L2B4F         ; forward to L-SINGLE if it was just
                                ; one character.

; HL points to start of variable name after 'LET' in BASIC line.

;; L-CHAR
L2B3E:  INC     HL              ; increase pointer.
        LD      A,(HL)          ; pick up character.
        CP      $21             ; is it space or higher ?
        JR      C,L2B3E         ; back to L-CHAR with space and less.

        OR      $20             ; make variable lower-case.
        INC     DE              ; increase destination pointer.
        LD      (DE),A          ; and load to edit line.
        DJNZ    L2B3E           ; loop back to L-CHAR until B is zero.

        OR      $80             ; invert the last character.
        LD      (DE),A          ; and overwrite that in edit line.

; now consider first character which has bit 6 set

        LD      A,$C0           ; set A 11000000 is xor mask for a long name.
                                ; %101      is xor/or  result

; single character numerics rejoin here with %00000000 in mask.
;                                            %011      will be xor/or result

;; L-SINGLE
L2B4F:  LD      HL,($5C4D)      ; fetch DEST - HL addresses first character.
        XOR     (HL)            ; apply variable type indicator mask (above).
        OR      $20             ; make lowercase - set bit 5.
        POP     HL              ; restore pointer to 2nd character.
        CALL    L2BEA           ; routine L-FIRST puts A in first character.
                                ; and returns with HL holding
                                ; new E_LINE-1  the $80 vars end-marker.

;; L-NUMERIC
L2B59:  PUSH    HL              ; save the pointer.

; the value of variable is deleted but remains after calculator stack.

        RST     28H             ;; FP-CALC
        DEFB    $02             ;;delete      ; delete variable value
        DEFB    $38             ;;end-calc

; DE (STKEND) points to start of value.

        POP     HL              ; restore the pointer.
        LD      BC,$0005        ; start of number is five bytes before.
        AND     A               ; prepare for true subtraction.
        SBC     HL,BC           ; HL points to start of value.
        JR      L2BA6           ; forward to L-ENTER  ==>

; ---


; the jump was to here if the variable already existed.

;; L-EXISTS
L2B66:  BIT     6,(IY+$01)      ; test FLAGS - numeric or string result ?
        JR      Z,L2B72         ; skip forward to L-DELETE$   -*->
                                ; if string result.
```

```
; A numeric variable could be simple or an array element.
; They are treated the same and the old value is overwritten.

        LD      DE,$0006        ; six bytes forward points to loc past value.
        ADD     HL,DE           ; add to start of number.
        JR      L2B59           ; back to L-NUMERIC to overwrite value.

; ---

; -*-> the branch was here if a string existed.

;; L-DELETE$
L2B72:  LD      HL,($5C4D)      ; fetch DEST to HL.
                                ; (still set from first instruction)
        LD      BC,($5C72)      ; fetch STRLEN to BC.
        BIT     0,(IY+$37)      ; test FLAGX - handling a complete simple
                                ; string ?
        JR      NZ,L2BAF        ; forward to L-ADD$ if so.

; must be a string array or a slice in workspace.
; Note. LET a$(3 TO 6) = h$   will assign "hat " if h$ = "hat"
;                                  and    "hats" if h$ = "hatstand".
;
; This is known as Procrustian lengthening and shortening after a
; character Procrustes in Greek legend who made travellers sleep in his bed,
; cutting off their feet or stretching them so they fitted the bed perfectly.
; The bloke was hatstand and slain by Theseus.

        LD      A,B             ; test if length
        OR      C               ; is zero and
        RET     Z               ; return if so.

        PUSH    HL              ; save pointer to start.

        RST     30H             ; BC-SPACES creates room.
        PUSH    DE              ; save pointer to first new location.
        PUSH    BC              ; and length            (*)
        LD      D,H             ; set DE to point to last location.
        LD      E,L             ;
        INC     HL              ; set HL to next location.
        LD      (HL),$20        ; place a space there.
        LDDR                    ; copy bytes filling with spaces.

        PUSH    HL              ; save pointer to start.
        CALL    L2BF1           ; routine STK-FETCH start to DE,
                                ; length to BC.
        POP     HL              ; restore the pointer.
        EX      (SP),HL         ; (*) length to HL, pointer to stack.
        AND     A               ; prepare for true subtraction.
        SBC     HL,BC           ; subtract old length from new.
        ADD     HL,BC           ; and add back.
        JR      NC,L2B9B        ; forward if it fits to L-LENGTH.

        LD      B,H             ; otherwise set
        LD      C,L             ; length to old length.
                                ; "hatstand" becomes "hats"

;; L-LENGTH
L2B9B:  EX      (SP),HL         ; (*) length to stack, pointer to HL.
        EX      DE,HL           ; pointer to DE, start of string to HL.
        LD      A,B             ; is the length zero ?
        OR      C               ;
        JR      Z,L2BA3         ; forward to L-IN-W/S if so
                                ; leaving prepared spaces.
```

```
        LDIR                        ; else copy bytes overwriting some spaces.

;; L-IN-W/S
L2BA3:  POP     BC                  ; pop the new length.  (*)
        POP     DE                  ; pop pointer to new area.
        POP     HL                  ; pop pointer to variable in assignment.
                                    ; and continue copying from workspace
                                    ; to variables area.

; ==> branch here from  L-NUMERIC

;; L-ENTER
L2BA6:  EX      DE,HL               ; exchange pointers HL=STKEND DE=end of vars.
        LD      A,B                 ; test the length
        OR      C                   ; and make a
        RET     Z                   ; return if zero (strings only).

        PUSH    DE                  ; save start of destination.
        LDIR                        ; copy bytes.
        POP     HL                  ; address the start.
        RET                         ; and return.

; ---

; the branch was here from L-DELETE$ if an existing simple string.
; register HL addresses start of string in variables area.

;; L-ADD$
L2BAF:  DEC     HL                  ; point to high byte of length.
        DEC     HL                  ; to low byte.
        DEC     HL                  ; to letter.
        LD      A,(HL)              ; fetch masked letter to A.
        PUSH    HL                  ; save the pointer on stack.
        PUSH    BC                  ; save new length.
        CALL    L2BC6               ; routine L-STRING adds new string at end
                                    ; of variables area.
                                    ; if no room we still have old one.
        POP     BC                  ; restore length.
        POP     HL                  ; restore start.
        INC     BC                  ; increase
        INC     BC                  ; length by three
        INC     BC                  ; to include character and length bytes.
        JP      L19E8               ; jump to indirect exit via RECLAIM-2
                                    ; deleting old version and adjusting pointers.

; ---

; the jump was here with a new string variable.

;; L-NEW$
L2BC0:  LD      A,$DF               ; indicator mask %11011111 for
                                    ;                %010xxxxx will be result
        LD      HL,($5C4D)          ; address DEST first character.
        AND     (HL)                ; combine mask with character.

;; L-STRING
L2BC6:  PUSH    AF                  ; save first character and mask.
        CALL    L2BF1               ; routine STK-FETCH fetches parameters of
                                    ; the string.
        EX      DE,HL               ; transfer start to HL.
        ADD     HL,BC               ; add to length.
        PUSH    BC                  ; save the length.
        DEC     HL                  ; point to end of string.
```

```
        LD      ($5C4D),HL      ; save pointer in DEST.
                                ; (updated by POINTERS if in workspace)
        INC     BC              ; extra byte for letter.
        INC     BC              ; two bytes
        INC     BC              ; for the length of string.
        LD      HL,($5C59)      ; address E_LINE.
        DEC     HL              ; now end of VARS area.
        CALL    L1655           ; routine MAKE-ROOM makes room for string.
                                ; updating pointers including DEST.
        LD      HL,($5C4D)      ; pick up pointer to end of string from DEST.
        POP     BC              ; restore length from stack.
        PUSH    BC              ; and save again on stack.
        INC     BC              ; add a byte.
        LDDR                    ; copy bytes from end to start.
        EX      DE,HL           ; HL addresses length low
        INC     HL              ; increase to address high byte
        POP     BC              ; restore length to BC
        LD      (HL),B          ; insert high byte
        DEC     HL              ; address low byte location
        LD      (HL),C          ; insert that byte
        POP     AF              ; restore character and mask

;; L-FIRST
L2BEA:  DEC     HL              ; address variable name
        LD      (HL),A          ; and insert character.
        LD      HL,($5C59)      ; load HL with E_LINE.
        DEC     HL              ; now end of VARS area.
        RET                     ; return

; ------------------------------------
; Get last value from calculator stack
; ------------------------------------
;
;

;; STK-FETCH
L2BF1:  LD      HL,($5C65)      ; STKEND
        DEC     HL              ;
        LD      B,(HL)          ;
        DEC     HL              ;
        LD      C,(HL)          ;
        DEC     HL              ;
        LD      D,(HL)          ;
        DEC     HL              ;
        LD      E,(HL)          ;
        DEC     HL              ;
        LD      A,(HL)          ;
        LD      ($5C65),HL      ; STKEND
        RET                     ;

; ------------------
; Handle DIM command
; ------------------
; e.g. DIM a(2,3,4,7): DIM a$(32) : DIM b$(20,2,768) : DIM c$(20000)
; the only limit to dimensions is memory so, for example,
; DIM a(2,2,2,2,2,2,2,2,2,2,2,2,2) is possible and creates a multi-
; dimensional array of zeros. String arrays are initialized to spaces.
; It is not possible to erase an array, but it can be re-dimensioned to
; a minimal size of 1, after use, to free up memory.

;; DIM
L2C02:  CALL    L28B2           ; routine LOOK-VARS

;; D-RPORT-C
```

```
L2C05:  JP      NZ,L1C8A        ; jump to REPORT-C if a long-name variable.
                                ; DIM lottery numbers(49) doesn't work.

        CALL    L2530           ; routine SYNTAX-Z
        JR      NZ,L2C15        ; forward to D-RUN in runtime.

        RES     6,C             ; signal 'numeric' array even if string as
                                ; this simplifies the syntax checking.

        CALL    L2996           ; routine STK-VAR checks syntax.
        CALL    L1BEE           ; routine CHECK-END performs early exit ->

; the branch was here in runtime.

;; D-RUN
L2C15:  JR      C,L2C1F         ; skip to D-LETTER if variable did not exist.
                                ; else reclaim the old one.

        PUSH    BC              ; save type in C.
        CALL    L19B8           ; routine NEXT-ONE find following variable
                                ; or position of $80 end-marker.
        CALL    L19E8           ; routine RECLAIM-2 reclaims the
                                ; space between.
        POP     BC              ; pop the type.

;; D-LETTER
L2C1F:  SET     7,C             ; signal array.
        LD      B,$00           ; initialize dimensions to zero and
        PUSH    BC              ; save with the type.
        LD      HL,$0001        ; make elements one character presuming string
        BIT     6,C             ; is it a string ?
        JR      NZ,L2C2D        ; forward to D-SIZE if so.

        LD      L,$05           ; make elements 5 bytes as is numeric.

;; D-SIZE
L2C2D:  EX      DE,HL           ; save the element size in DE.

; now enter a loop to parse each of the integers in the list.

;; D-NO-LOOP
L2C2E:  RST     20H             ; NEXT-CHAR
        LD      H,$FF           ; disable limit check by setting HL high
        CALL    L2ACC           ; routine INT-EXP1
        JP      C,L2A20         ; to REPORT-3 if > 65280 and then some
                                ; 'Subscript out of range'

        POP     HL              ; pop dimension counter, array type
        PUSH    BC              ; save dimension size                     ***
        INC     H               ; increment the dimension counter
        PUSH    HL              ; save the dimension counter
        LD      H,B             ; transfer size
        LD      L,C             ; to HL
        CALL    L2AF4           ; routine GET-HL*DE multiplies dimension by
                                ; running total of size required initially
                                ; 1 or 5.
        EX      DE,HL           ; save running total in DE

        RST     18H             ; GET-CHAR
        CP      $2C             ; is it ',' ?
        JR      Z,L2C2E         ; loop back to D-NO-LOOP until all dimensions
                                ; have been considered

; when loop complete continue.
```

```
        CP      $29             ; is it ')' ?
        JR      NZ,L2C05        ; to D-RPORT-C with anything else
                                ; 'Nonsense in BASIC'


        RST     20H             ; NEXT-CHAR advances to next statement/CR

        POP     BC              ; pop dimension counter/type
        LD      A,C             ; type to A

; now calculate space required for array variable

        LD      L,B             ; dimensions to L since these require 16 bits
                                ; then this value will be doubled
        LD      H,$00           ; set high byte to zero

; another four bytes are required for letter(1), total length(2), number of
; dimensions(1) but since we have yet to double allow for two

        INC     HL              ; increment
        INC     HL              ; increment

        ADD     HL,HL           ; now double giving 4 + dimensions * 2

        ADD     HL,DE           ; add to space required for array contents

        JP      C,L1F15         ; to REPORT-4 if > 65535
                                ; 'Out of memory'

        PUSH    DE              ; save data space
        PUSH    BC              ; save dimensions/type
        PUSH    HL              ; save total space
        LD      B,H             ; total space
        LD      C,L             ; to BC
        LD      HL,($5C59)      ; address E_LINE - first location after
                                ; variables area
        DEC     HL              ; point to location before - the $80 end-marker
        CALL    L1655           ; routine MAKE-ROOM creates the space if
                                ; memory is available.

        INC     HL              ; point to first new location and
        LD      (HL),A          ; store letter/type

        POP     BC              ; pop total space
        DEC     BC              ; exclude name
        DEC     BC              ; exclude the 16-bit
        DEC     BC              ; counter itself
        INC     HL              ; point to next location the 16-bit counter
        LD      (HL),C          ; insert low byte
        INC     HL              ; address next
        LD      (HL),B          ; insert high byte

        POP     BC              ; pop the number of dimensions.
        LD      A,B             ; dimensions to A
        INC     HL              ; address next
        LD      (HL),A          ; and insert "No. of dims"

        LD      H,D             ; transfer DE space + 1 from make-room
        LD      L,E             ; to HL
        DEC     DE              ; set DE to next location down.
        LD      (HL),$00        ; presume numeric and insert a zero
        BIT     6,C             ; test bit 6 of C. numeric or string ?
        JR      Z,L2C7C         ; skip to DIM-CLEAR if numeric
```

```
        LD      (HL),$20        ; place a space character in HL

;; DIM-CLEAR
L2C7C:  POP     BC              ; pop the data length

        LDDR                    ; LDDR sets to zeros or spaces

; The number of dimensions is still in A.
; A loop is now entered to insert the size of each dimension that was pushed
; during the D-NO-LOOP working downwards from position before start of data.

;; DIM-SIZES
L2C7F:  POP     BC              ; pop a dimension size                    ***
        LD      (HL),B          ; insert high byte at position
        DEC     HL              ; next location down
        LD      (HL),C          ; insert low byte
        DEC     HL              ; next location down
        DEC     A               ; decrement dimension counter
        JR      NZ,L2C7F        ; back to DIM-SIZES until all done.

        RET                     ; return.

; -----------------------------
; Check whether digit or letter
; -----------------------------
; This routine checks that the character in A is alphanumeric
; returning with carry set if so.

;; ALPHANUM
L2C88:  CALL    L2D1B           ; routine NUMERIC will reset carry if so.
        CCF                     ; Complement Carry Flag
        RET     C               ; Return if numeric else continue into
                                ; next routine.

; This routine checks that the character in A is alphabetic

;; ALPHA
L2C8D:  CP      $41             ; less than 'A' ?
        CCF                     ; Complement Carry Flag
        RET     NC              ; return if so

        CP      $5B             ; less than 'Z'+1 ?
        RET     C               ; is within first range

        CP      $61             ; less than 'a' ?
        CCF                     ; Complement Carry Flag
        RET     NC              ; return if so.

        CP      $7B             ; less than 'z'+1 ?
        RET                     ; carry set if within a-z.

; -------------------------
; Decimal to floating point
; -------------------------
; This routine finds the floating point number represented by an expression
; beginning with BIN, '.' or a digit.
; Note that BIN need not have any '0's or '1's after it.
; BIN is really just a notational symbol and not a function.

;; DEC-TO-FP
L2C9B:  CP      $C4             ; 'BIN' token ?
        JR      NZ,L2CB8        ; to NOT-BIN if not
```

```
        LD      DE,$0000        ; initialize 16 bit buffer register.

;; BIN-DIGIT
L2CA2:  RST     20H             ; NEXT-CHAR
        SUB     $31             ; '1'
        ADC     A,$00           ; will be zero if '1' or '0'
                                ; carry will be set if was '0'
        JR      NZ,L2CB3        ; forward to BIN-END if result not zero

        EX      DE,HL           ; buffer to HL
        CCF                     ; Carry now set if originally '1'
        ADC     HL,HL           ; shift the carry into HL
        JP      C,L31AD         ; to REPORT-6 if overflow - too many digits
                                ; after first '1'. There can be an unlimited
                                ; number of leading zeros.
                                ; 'Number too big' - raise an error

        EX      DE,HL           ; save the buffer
        JR      L2CA2           ; back to BIN-DIGIT for more digits

; ---

;; BIN-END
L2CB3:  LD      B,D             ; transfer 16 bit buffer
        LD      C,E             ; to BC register pair.
        JP      L2D2B           ; JUMP to STACK-BC to put on calculator stack

; ---

; continue here with .1,  42, 3.14, 5., 2.3 E -4

;; NOT-BIN
L2CB8:  CP      $2E             ; '.' - leading decimal point ?
        JR      Z,L2CCB         ; skip to DECIMAL if so.

        CALL    L2D3B           ; routine INT-TO-FP to evaluate all digits
                                ; This number 'x' is placed on stack.
        CP      $2E             ; '.' - mid decimal point ?

        JR      NZ,L2CEB        ; to E-FORMAT if not to consider that format

        RST     20H             ; NEXT-CHAR
        CALL    L2D1B           ; routine NUMERIC returns carry reset if 0-9

        JR      C,L2CEB         ; to E-FORMAT if not a digit e.g. '1.'

        JR      L2CD5           ; to DEC-STO-1 to add the decimal part to 'x'

; ---

; a leading decimal point has been found in a number.

;; DECIMAL
L2CCB:  RST     20H             ; NEXT-CHAR
        CALL    L2D1B           ; routine NUMERIC will reset carry if digit

;; DEC-RPT-C
L2CCF:  JP      C,L1C8A         ; to REPORT-C if just a '.'
                                ; raise 'Nonsense in BASIC'

; since there is no leading zero put one on the calculator stack.

        RST     28H             ;; FP-CALC
        DEFB    $A0             ;;stk-zero  ; 0.
```

```
        DEFB    $38             ;;end-calc

; If rejoining from earlier there will be a value 'x' on stack.
; If continuing from above the value zero.
; Now store 1 in mem-0.
; Note. At each pass of the digit loop this will be divided by ten.

;; DEC-STO-1
L2CD5:  RST     28H             ;; FP-CALC
        DEFB    $A1             ;;stk-one   ;x or 0,1.
        DEFB    $C0             ;;st-mem-0  ;x or 0,1.
        DEFB    $02             ;;delete    ;x or 0.
        DEFB    $38             ;;end-calc


;; NXT-DGT-1
L2CDA:  RST     18H             ; GET-CHAR
        CALL    L2D22           ; routine STK-DIGIT stacks single digit 'd'
        JR      C,L2CEB         ; exit to E-FORMAT when digits exhausted  >


        RST     28H             ;; FP-CALC   ;x or 0,d.           first pass.
        DEFB    $E0             ;;get-mem-0  ;x or 0,d,1.
        DEFB    $A4             ;;stk-ten    ;x or 0,d,1,10.
        DEFB    $05             ;;division   ;x or 0,d,1/10.
        DEFB    $C0             ;;st-mem-0   ;x or 0,d,1/10.
        DEFB    $04             ;;multiply   ;x or 0,d/10.
        DEFB    $0F             ;;addition   ;x or 0 + d/10.
        DEFB    $38             ;;end-calc   last value.

        RST     20H             ; NEXT-CHAR  moves to next character
        JR      L2CDA           ; back to NXT-DGT-1

; ---

; although only the first pass is shown it can be seen that at each pass
; the new less significant digit is multiplied by an increasingly smaller
; factor (1/100, 1/1000, 1/10000 ... ) before being added to the previous
; last value to form a new last value.

; Finally see if an exponent has been input.

;; E-FORMAT
L2CEB:  CP      $45             ; is character 'E' ?
        JR      Z,L2CF2         ; to SIGN-FLAG if so

        CP      $65             ; 'e' is acceptable as well.
        RET     NZ              ; return as no exponent.

;; SIGN-FLAG
L2CF2:  LD      B,$FF           ; initialize temporary sign byte to $FF

        RST     20H             ; NEXT-CHAR
        CP      $2B             ; is character '+' ?
        JR      Z,L2CFE         ; to SIGN-DONE

        CP      $2D             ; is character '-' ?
        JR      NZ,L2CFF        ; to ST-E-PART as no sign

        INC     B               ; set sign to zero

; now consider digits of exponent.
; Note. incidentally this is the only occasion in Spectrum BASIC when an
; expression may not be used when a number is expected.
```

```
;; SIGN-DONE
L2CFE:  RST     20H             ; NEXT-CHAR

;; ST-E-PART
L2CFF:  CALL    L2D1B           ; routine NUMERIC
        JR      C,L2CCF         ; to DEC-RPT-C if not
                                ; raise 'Nonsense in BASIC'.

        PUSH    BC              ; save sign (in B)
        CALL    L2D3B           ; routine INT-TO-FP places exponent on stack
        CALL    L2DD5           ; routine FP-TO-A  transfers it to A
        POP     BC              ; restore sign
        JP      C,L31AD         ; to REPORT-6 if overflow (over 255)
                                ; raise 'Number too big'.

        AND     A               ; set flags
        JP      M,L31AD         ; to REPORT-6 if over '127'.
                                ; raise 'Number too big'.
                                ; 127 is still way too high and it is
                                ; impossible to enter an exponent greater
                                ; than 39 from the keyboard. The error gets
                                ; raised later in E-TO-FP so two different
                                ; error messages depending how high A is.

        INC     B               ; $FF to $00 or $00 to $01 - expendable now.
        JR      Z,L2D18         ; forward to E-FP-JUMP if exponent positive

        NEG                     ; Negate the exponent.

;; E-FP-JUMP
L2D18:  JP      L2D4F           ; JUMP forward to E-TO-FP to assign to
                                ; last value x on stack x * 10 to power A
                                ; a relative jump would have done.

; ---------------------
; Check for valid digit
; ---------------------
; This routine checks that the ASCII character in A is numeric
; returning with carry reset if so.

;; NUMERIC
L2D1B:  CP      $30             ; '0'
        RET     C               ; return if less than zero character.

        CP      $3A             ; The upper test is '9'
        CCF                     ; Complement Carry Flag
        RET                     ; Return - carry clear if character '0' - '9'

; -----------
; Stack Digit
; -----------
; This subroutine is called from INT-TO-FP and DEC-TO-FP to stack a digit
; on the calculator stack.

;; STK-DIGIT
L2D22:  CALL    L2D1B           ; routine NUMERIC
        RET     C               ; return if not numeric character

        SUB     $30             ; convert from ASCII to digit

; -----------------
; Stack accumulator
; -----------------
```

```
;
;

;; STACK-A
L2D28:  LD      C,A             ; transfer to C
        LD      B,$00           ; and make B zero

; ----------------------
; Stack BC register pair
; ----------------------
;

;; STACK-BC
L2D2B:  LD      IY,$5C3A        ; re-initialize ERR_NR

        XOR     A               ; clear to signal small integer
        LD      E,A             ; place in E for sign
        LD      D,C             ; LSB to D
        LD      C,B             ; MSB to C
        LD      B,A             ; last byte not used
        CALL    L2AB6           ; routine STK-STORE

        RST     28H             ;; FP-CALC
        DEFB    $38             ;;end-calc  make HL = STKEND-5

        AND     A               ; clear carry
        RET                     ; before returning

; -------------------------
; Integer to floating point
; -------------------------
; This routine places one or more digits found in a BASIC line
; on the calculator stack multiplying the previous value by ten each time
; before adding in the new digit to form a last value on calculator stack.

;; INT-TO-FP
L2D3B:  PUSH    AF              ; save first character

        RST     28H             ;; FP-CALC
        DEFB    $A0             ;;stk-zero    ; v=0. initial value
        DEFB    $38             ;;end-calc

        POP     AF              ; fetch first character back.

;; NXT-DGT-2
L2D40:  CALL    L2D22           ; routine STK-DIGIT puts 0-9 on stack
        RET     C               ; will return when character is not numeric >

        RST     28H             ;; FP-CALC    ; v, d.
        DEFB    $01             ;;exchange    ; d, v.
        DEFB    $A4             ;;stk-ten     ; d, v, 10.
        DEFB    $04             ;;multiply    ; d, v*10.
        DEFB    $0F             ;;addition    ; d + v*10 = newvalue
        DEFB    $38             ;;end-calc    ; v.

        CALL    L0074           ; routine CH-ADD+1 get next character
        JR      L2D40           ; back to NXT-DGT-2 to process as a digit


;*********************************
;** Part 9. ARITHMETIC ROUTINES **
;*********************************

; --------------------------
```

```
; E-format to floating point
; --------------------------
; This subroutine is used by the PRINT-FP routine and the decimal to FP
; routines to stack a number expressed in exponent format.
; Note. Though not used by the ROM as such, it has also been set up as
; a unary calculator literal but this will not work as the accumulator
; is not available from within the calculator.

; on entry there is a value x on the calculator stack and an exponent of ten
; in A.    The required value is x + 10 ^ A

;; e-to-fp
;; E-TO-FP
L2D4F:  RLCA                    ; this will set the         x.
        RRCA                    ; carry if bit 7 is set

        JR      NC,L2D55        ; to E-SAVE  if positive.

        CPL                     ; make negative positive
        INC     A               ; without altering carry.

;; E-SAVE
L2D55:  PUSH    AF              ; save positive exp and sign in carry

        LD      HL,$5C92        ; address MEM-0

        CALL    L350B           ; routine FP-0/1
                                ; places an integer zero, if no carry,
                                ; else a one in mem-0 as a sign flag

        RST     28H             ;; FP-CALC
        DEFB    $A4             ;;stk-ten                   x, 10.
        DEFB    $38             ;;end-calc

        POP     AF              ; pop the exponent.

; now enter a loop

;; E-LOOP
L2D60:  SRL     A               ; 0>76543210>C

        JR      NC,L2D71        ; forward to E-TST-END if no bit

        PUSH    AF              ; save shifted exponent.

        RST     28H             ;; FP-CALC
        DEFB    $C1             ;;st-mem-1                  x, 10.
        DEFB    $E0             ;;get-mem-0                 x, 10, (0/1).
        DEFB    $00             ;;jump-true

        DEFB    $04             ;;to L2D6D, E-DIVSN

        DEFB    $04             ;;multiply                  x*10.
        DEFB    $33             ;;jump

        DEFB    $02             ;;to L2D6E, E-FETCH

;; E-DIVSN
L2D6D:  DEFB    $05             ;;division                  x/10.

;; E-FETCH
L2D6E:  DEFB    $E1             ;;get-mem-1                 x/10 or x*10, 10.
        DEFB    $38             ;;end-calc                  new x, 10.
```

```
        POP     AF              ; restore shifted exponent

; the loop branched to here with no carry

;; E-TST-END
L2D71:  JR      Z,L2D7B         ; forward to E-END  if A emptied of bits

        PUSH    AF              ; re-save shifted exponent

        RST     28H             ;; FP-CALC
        DEFB    $31             ;;duplicate                 new x, 10, 10.
        DEFB    $04             ;;multiply                  new x, 100.
        DEFB    $38             ;;end-calc

        POP     AF              ; restore shifted exponent
        JR      L2D60           ; back to E-LOOP  until all bits done.

; ---

; although only the first pass is shown it can be seen that for each set bit
; representing a power of two, x is multiplied or divided by the
; corresponding power of ten.

;; E-END
L2D7B:  RST     28H             ;; FP-CALC                  final x, factor.
        DEFB    $02             ;;delete                    final x.
        DEFB    $38             ;;end-calc                  x.

        RET                     ; return




; -------------
; Fetch integer
; -------------
; This routine is called by the mathematical routines - FP-TO-BC, PRINT-FP,
; mult, re-stack and negate to fetch an integer from address HL.
; HL points to the stack or a location in MEM and no deletion occurs.
; If the number is negative then a similar process to that used in INT-STORE
; is used to restore the twos complement number to normal in DE and a sign
; in C.

;; INT-FETCH
L2D7F:  INC     HL              ; skip zero indicator.
        LD      C,(HL)          ; fetch sign to C
        INC     HL              ; address low byte
        LD      A,(HL)          ; fetch to A
        XOR     C               ; two's complement
        SUB     C               ;
        LD      E,A             ; place in E
        INC     HL              ; address high byte
        LD      A,(HL)          ; fetch to A
        ADC     A,C             ; two's complement
        XOR     C               ;
        LD      D,A             ; place in D
        RET                     ; return

; ------------------------
; Store a positive integer
; ------------------------
; This entry point is not used in this ROM but would
; store any integer as positive.
```

```
;; p-int-sto
L2D8C:  LD      C,$00           ; make sign byte positive and continue

; -------------
; Store integer
; -------------
; this routine stores an integer in DE at address HL.
; It is called from mult, truncate, negate and sgn.
; The sign byte $00 +ve or $FF -ve is in C.
; If negative, the number is stored in 2's complement form so that it is
; ready to be added.

;; INT-STORE
L2D8E:  PUSH    HL              ; preserve HL

        LD      (HL),$00        ; first byte zero shows integer not exponent
        INC     HL              ;
        LD      (HL),C          ; then store the sign byte
        INC     HL              ;
                                ; e.g.             +1             -1
        LD      A,E             ; fetch low byte   00000001       00000001
        XOR     C               ; xor sign         00000000   or  11111111
                                ; gives            00000001   or  11111110
        SUB     C               ; sub sign         00000000   or  11111111
                                ; gives            00000001>0 or  11111111>C
        LD      (HL),A          ; store 2's complement.
        INC     HL              ;
        LD      A,D             ; high byte        00000000       00000000
        ADC     A,C             ; sign             00000000<0     11111111<C
                                ; gives            00000000   or  00000000
        XOR     C               ; xor sign         00000000       11111111
        LD      (HL),A          ; store 2's complement.
        INC     HL              ;
        LD      (HL),$00        ; last byte always zero for integers.
                                ; is not used and need not be looked at when
                                ; testing for zero but comes into play should
                                ; an integer be converted to fp.
        POP     HL              ; restore HL
        RET                     ; return.


; -----------------------------
; Floating point to BC register
; -----------------------------
; This routine gets a floating point number e.g. 127.4 from the calculator
; stack to the BC register.

;; FP-TO-BC
L2DA2:  RST     28H             ;; FP-CALC            set HL to
        DEFB    $38             ;;end-calc            point to last value.

        LD      A,(HL)          ; get first of 5 bytes
        AND     A               ; and test
        JR      Z,L2DAD         ; forward to FP-DELETE if an integer

; The value is first rounded up and then converted to integer.

        RST     28H             ;; FP-CALC           x.
        DEFB    $A2             ;;stk-half           x. 1/2.
        DEFB    $0F             ;;addition           x + 1/2.
        DEFB    $27             ;;int                int(x + .5)
        DEFB    $38             ;;end-calc

; now delete but leave HL pointing at integer
```

```
;; FP-DELETE
L2DAD:  RST     28H             ;; FP-CALC
        DEFB    $02             ;;delete
        DEFB    $38             ;;end-calc

        PUSH    HL              ; save pointer.
        PUSH    DE              ; and STKEND.
        EX      DE,HL           ; make HL point to exponent/zero indicator
        LD      B,(HL)          ; indicator to B
        CALL    L2D7F           ; routine INT-FETCH
                                ; gets int in DE sign byte to C
                                ; but meaningless values if a large integer

        XOR     A               ; clear A
        SUB     B               ; subtract indicator byte setting carry
                                ; if not a small integer.

        BIT     7,C             ; test a bit of the sign byte setting zero
                                ; if positive.

        LD      B,D             ; transfer int
        LD      C,E             ; to BC
        LD      A,E             ; low byte to A as a useful return value.

        POP     DE              ; pop STKEND
        POP     HL              ; and pointer to last value
        RET                     ; return
                                ; if carry is set then the number was too big.

; ------------
; LOG(2^A)
; ------------
; This routine is used when printing floating point numbers to calculate
; the number of digits before the decimal point.

; first convert a one-byte signed integer to its five byte form.

;; LOG(2^A)
L2DC1:  LD      D,A             ; store a copy of A in D.
        RLA                     ; test sign bit of A.
        SBC     A,A             ; now $FF if negative or $00
        LD      E,A             ; sign byte to E.
        LD      C,A             ; and to C
        XOR     A               ; clear A
        LD      B,A             ; and B.
        CALL    L2AB6           ; routine STK-STORE stacks number AEDCB

;  so 00 00 XX 00 00 (positive) or 00 FF XX FF 00 (negative).
;  i.e. integer indicator, sign byte, low, high, unused.

; now multiply exponent by log to the base 10 of two.

        RST      28H            ;; FP-CALC

        DEFB    $34             ;;stk-data                      .30103 (log 2)
        DEFB    $EF             ;;Exponent: $7F, Bytes: 4
        DEFB    $1A,$20,$9A,$85 ;;
        DEFB    $04             ;;multiply

        DEFB    $27             ;;int

        DEFB    $38             ;;end-calc
```

```
; -------------------
; Floating point to A
; -------------------
; this routine collects a floating point number from the stack into the
; accumulator returning carry set if not in range 0 - 255.
; Not all the calling routines raise an error with overflow so no attempt
; is made to produce an error report here.

;; FP-TO-A
L2DD5:  CALL    L2DA2           ; routine FP-TO-BC returns with C in A also.
        RET     C               ; return with carry set if > 65535, overflow

        PUSH    AF              ; save the value and flags
        DEC     B               ; and test that
        INC     B               ; the high byte is zero.
        JR      Z,L2DE1         ; forward  FP-A-END if zero

; else there has been 8-bit overflow

        POP     AF              ; retrieve the value
        SCF                     ; set carry flag to show overflow
        RET                     ; and return.

; ---

;; FP-A-END
L2DE1:  POP     AF              ; restore value and success flag and
        RET                     ; return.


; -----------------------------
; Print a floating point number
; -----------------------------
; Not a trivial task.
; Begin by considering whether to print a leading sign for negative numbers.

;; PRINT-FP
L2DE3:  RST     28H             ;; FP-CALC
        DEFB    $31             ;;duplicate
        DEFB    $36             ;;less-0
        DEFB    $00             ;;jump-true

        DEFB    $0B             ;;to L2DF2, PF-NEGTVE

        DEFB    $31             ;;duplicate
        DEFB    $37             ;;greater-0
        DEFB    $00             ;;jump-true

        DEFB    $0D             ;;to L2DF8, PF-POSTVE

; must be zero itself

        DEFB    $02             ;;delete
        DEFB    $38             ;;end-calc

        LD      A,$30           ; prepare the character '0'

        RST     10H             ; PRINT-A
        RET                     ; return.                 ->
; ---

;; PF-NEGTVE
L2DF2:  DEFB    $2A             ;;abs
        DEFB    $38             ;;end-calc
```

```
        LD      A,$2D           ; the character '-'

        RST     10H             ; PRINT-A

; and continue to print the now positive number.

        RST     28H             ;; FP-CALC

;; PF-POSTVE
L2DF8:  DEFB    $A0             ;;stk-zero     x,0.     begin by
        DEFB    $C3             ;;st-mem-3     x,0.     clearing a temporary
        DEFB    $C4             ;;st-mem-4     x,0.     output buffer to
        DEFB    $C5             ;;st-mem-5     x,0.     fifteen zeros.
        DEFB    $02             ;;delete       x.
        DEFB    $38             ;;end-calc     x.

        EXX                     ; in case called from 'str$' then save the
        PUSH    HL              ; pointer to whatever comes after
        EXX                     ; str$ as H'L' will be used.

; now enter a loop?

;; PF-LOOP
L2E01:  RST     28H             ;; FP-CALC
        DEFB    $31             ;;duplicate    x,x.
        DEFB    $27             ;;int          x,int x.
        DEFB    $C2             ;;st-mem-2     x,int x.
        DEFB    $03             ;;subtract     x-int x.     fractional part.
        DEFB    $E2             ;;get-mem-2    x-int x, int x.
        DEFB    $01             ;;exchange     int x, x-int x.
        DEFB    $C2             ;;st-mem-2     int x, x-int x.
        DEFB    $02             ;;delete       int x.
        DEFB    $38             ;;end-calc     int x.
                                ;
                                ; mem-2 holds the fractional part.

; HL points to last value int x

        LD      A,(HL)          ; fetch exponent of int x.
        AND     A               ; test
        JR      NZ,L2E56        ; forward to PF-LARGE if a large integer
                                ; > 65535

; continue with small positive integer components in range 0 - 65535
; if original number was say .999 then this integer component is zero.

        CALL    L2D7F           ; routine INT-FETCH gets x in DE
                                ; (but x is not deleted)

        LD      B,$10           ; set B, bit counter, to 16d

        LD      A,D             ; test if
        AND     A               ; high byte is zero
        JR      NZ,L2E1E        ; forward to PF-SAVE if 16-bit integer.

; and continue with integer in range 0 - 255.

        OR      E               ; test the low byte for zero
                                ; i.e. originally just point something or other.
        JR      Z,L2E24         ; forward if so to PF-SMALL

;
```

```
        LD      D,E             ; transfer E to D
        LD      B,$08           ; and reduce the bit counter to 8.

;; PF-SAVE
L2E1E:  PUSH    DE              ; save the part before decimal point.
        EXX                     ;
        POP     DE              ; and pop in into D'E'
        EXX                     ;
        JR      L2E7B           ; forward to PF-BITS

; ---------------------

; the branch was here when 'int x' was found to be zero as in say 0.5.
; The zero has been fetched from the calculator stack but not deleted and
; this should occur now. This omission leaves the stack unbalanced and while
; that causes no problems with a simple PRINT statement, it will if str$ is
; being used in an expression e.g. "2" + STR$ 0.5 gives the result "0.5"
; instead of the expected result "20.5".
; credit Tony Stratton, 1982.
; A DEFB 02 delete is required immediately on using the calculator.

;; PF-SMALL
L2E24:  RST     28H             ;; FP-CALC       int x = 0.
L2E25:  DEFB    $E2             ;;get-mem-2      int x = 0, x-int x.
        DEFB    $38             ;;end-calc

        LD      A,(HL)          ; fetch exponent of positive fractional number
        SUB     $7E             ; subtract

        CALL    L2DC1           ; routine LOG(2^A) calculates leading digits.

        LD      D,A             ; transfer count to D
        LD      A,($5CAC)       ; fetch total MEM-5-1
        SUB     D               ;
        LD      ($5CAC),A       ; MEM-5-1
        LD      A,D             ;
        CALL    L2D4F           ; routine E-TO-FP

        RST     28H             ;; FP-CALC
        DEFB    $31             ;;duplicate
        DEFB    $27             ;;int
        DEFB    $C1             ;;st-mem-1
        DEFB    $03             ;;subtract
        DEFB    $E1             ;;get-mem-1
        DEFB    $38             ;;end-calc

        CALL    L2DD5           ; routine FP-TO-A

        PUSH    HL              ; save HL
        LD      ($5CA1),A       ; MEM-3-1
        DEC     A               ;
        RLA                     ;
        SBC     A,A             ;
        INC     A               ;

        LD      HL,$5CAB        ; address MEM-5-1 leading digit counter
        LD      (HL),A          ; store counter
        INC     HL              ; address MEM-5-2 total digits
        ADD     A,(HL)          ; add counter to contents
        LD      (HL),A          ; and store updated value
        POP     HL              ; restore HL

        JP      L2ECF           ; JUMP forward to PF-FRACTN
```

```
; ---

; Note. while it would be pedantic to comment on every occasion a JP
; instruction could be replaced with a JR instruction, this applies to the
; above, which is useful if you wish to correct the unbalanced stack error
; by inserting a 'DEFB 02 delete' at L2E25, and maintain main addresses.

; the branch was here with a large positive integer > 65535 e.g. 123456789
; the accumulator holds the exponent.

;; PF-LARGE
L2E56:  SUB     $80             ; make exponent positive
        CP      $1C             ; compare to 28
        JR      C,L2E6F         ; to PF-MEDIUM if integer <= 2^27

        CALL    L2DC1           ; routine LOG(2^A)
        SUB     $07             ;
        LD      B,A             ;
        LD      HL,$5CAC        ; address MEM-5-1 the leading digits counter.
        ADD     A,(HL)          ; add A to contents
        LD      (HL),A          ; store updated value.
        LD      A,B             ;
        NEG                     ; negate
        CALL    L2D4F           ; routine E-TO-FP
        JR      L2E01           ; back to PF-LOOP

; ----------------------------

;; PF-MEDIUM
L2E6F:  EX      DE,HL           ;
        CALL    L2FBA           ; routine FETCH-TWO
        EXX                     ;
        SET     7,D             ;
        LD      A,L             ;
        EXX                     ;
        SUB     $80             ;
        LD      B,A             ;

; the branch was here to handle bits in DE with 8 or 16 in B  if small int
; and integer in D'E', 6 nibbles will accommodate 065535 but routine does
; 32-bit numbers as well from above

;; PF-BITS
L2E7B:  SLA     E               ;  C<xxxxxxxx<0
        RL      D               ;  C<xxxxxxxx<C
        EXX                     ;
        RL      E               ;  C<xxxxxxxx<C
        RL      D               ;  C<xxxxxxxx<C
        EXX                     ;

        LD      HL,$5CAA        ; set HL to mem-4-5th last byte of buffer
        LD      C,$05           ; set byte count to 5 -  10 nibbles

;; PF-BYTES
L2E8A:  LD      A,(HL)          ; fetch 0 or prev value
        ADC     A,A             ; shift left add in carry    C<xxxxxxxx<C

        DAA                     ; Decimal Adjust Accumulator.
                                ; if greater than 9 then the left hand
                                ; nibble is incremented. If greater than
                                ; 99 then adjusted and carry set.
                                ; so if we'd built up 7 and a carry came in
                                ;      0000 0111 < C
                                ;      0000 1111
```

```
                                ; daa     1 0101  which is 15 in BCD

        LD      (HL),A          ; put back
        DEC     HL              ; work down thru mem 4
        DEC     C               ; decrease the 5 counter.
        JR      NZ,L2E8A        ; back to PF-BYTES until the ten nibbles rolled

        DJNZ    L2E7B           ; back to PF-BITS until 8 or 16 (or 32) done

; at most 9 digits for 32-bit number will have been loaded with digits
; each of the 9 nibbles in mem 4 is placed into ten bytes in mem-3 and mem 4
; unless the nibble is zero as the buffer is already zero.
; ( or in the case of mem-5 will become zero as a result of RLD instruction )

        XOR     A               ; clear to accept
        LD      HL,$5CA6        ; address MEM-4-0 byte destination.
        LD      DE,$5CA1        ; address MEM-3-0 nibble source.
        LD      B,$09           ; the count is 9 (not ten) as the first
                                ; nibble is known to be blank.

        RLD                     ; shift RH nibble to left in (HL)
                                ;    A           (HL)
                                ; 0000 0000 < 0000 3210
                                ; 0000 0000   3210 0000
                                ; A picks up the blank nibble


        LD      C,$FF           ; set a flag to indicate when a significant
                                ; digit has been encountered.

;; PF-DIGITS
L2EA1:  RLD                     ; pick up leftmost nibble from (HL)
                                ;    A           (HL)
                                ; 0000 0000 < 7654 3210
                                ; 0000 7654   3210 0000


        JR      NZ,L2EA9        ; to PF-INSERT if non-zero value picked up.

        DEC     C               ; test
        INC     C               ; flag
        JR      NZ,L2EB3        ; skip forward to PF-TEST-2 if flag still $FF
                                ; indicating this is a leading zero.

; but if the zero is a significant digit e.g. 10 then include in digit totals.
; the path for non-zero digits rejoins here.

;; PF-INSERT
L2EA9:  LD      (DE),A          ; insert digit at destination
        INC     DE              ; increase the destination pointer
        INC     (IY+$71)        ; increment MEM-5-1st  digit counter
        INC     (IY+$72)        ; increment MEM-5-2nd  leading digit counter
        LD      C,$00           ; set flag to zero indicating that any
                                ; subsequent zeros are significant and not
                                ; leading.

;; PF-TEST-2
L2EB3:  BIT     0,B             ; test if the nibble count is even
        JR      Z,L2EB8         ; skip to PF-ALL-9 if so to deal with the
                                ; other nibble in the same byte

        INC     HL              ; point to next source byte if not

;; PF-ALL-9
```

```
L2EB8:  DJNZ    L2EA1           ; decrement the nibble count, back to PF-DIGITS
                                ; if all nine not done.

; For 8-bit integers there will be at most 3 digits.
; For 16-bit integers there will be at most 5 digits.
; but for larger integers there could be nine leading digits.
; if nine digits complete then the last one is rounded up as the number will
; be printed using E-format notation

        LD      A,($5CAB)       ; fetch digit count from MEM-5-1st
        SUB     $09             ; subtract 9 - max possible
        JR      C,L2ECB         ; forward if less to PF-MORE

        DEC     (IY+$71)        ; decrement digit counter MEM-5-1st to 8
        LD      A,$04           ; load A with the value 4.
        CP      (IY+$6F)        ; compare with MEM-4-4th - the ninth digit
        JR      L2F0C           ; forward to PF-ROUND
                                ; to consider rounding.

; ---------------------------------------

; now delete int x from calculator stack and fetch fractional part.

;; PF-MORE
L2ECB:  RST     28H             ;; FP-CALC         int x.
        DEFB    $02             ;;delete           .
        DEFB    $E2             ;;get-mem-2        x - int x = f.
        DEFB    $38             ;;end-calc         f.

;; PF-FRACTN
L2ECF:  EX      DE,HL           ;
        CALL    L2FBA           ; routine FETCH-TWO
        EXX                     ;
        LD      A,$80           ;
        SUB     L               ;
        LD      L,$00           ;
        SET     7,D             ;
        EXX                     ;
        CALL    L2FDD           ; routine SHIFT-FP

;; PF-FRN-LP
L2EDF:  LD      A,(IY+$71)      ; MEM-5-1st
        CP      $08             ;
        JR      C,L2EEC         ; to PF-FR-DGT

        EXX                     ;
        RL      D               ;
        EXX                     ;
        JR      L2F0C           ; to PF-ROUND

; ---

;; PF-FR-DGT
L2EEC:  LD      BC,$0200        ;

;; PF-FR-EXX
L2EEF:  LD      A,E             ;
        CALL    L2F8B           ; routine CA-10*A+C
        LD      E,A             ;
        LD      A,D             ;
        CALL    L2F8B           ; routine CA-10*A+C
        LD      D,A             ;
        PUSH    BC              ;
        EXX                     ;
```

```
        POP     BC                ;
        DJNZ    L2EEF             ; to PF-FR-EXX

        LD      HL,$5CA1          ; MEM-3
        LD      A,C               ;
        LD      C,(IY+$71)        ; MEM-5-1st
        ADD     HL,BC             ;
        LD      (HL),A            ;
        INC     (IY+$71)          ; MEM-5-1st
        JR      L2EDF             ; to PF-FRN-LP

; ----------------

; 1) with 9 digits but 8 in mem-5-1 and A holding 4, carry set if rounding up.
; e.g.
;      999999999 is printed as 1E+9
;      100000001 is printed as 1E+8
;      100000009 is printed as 1.0000001E+8

;; PF-ROUND
L2F0C:  PUSH    AF                ; save A and flags
        LD      HL,$5CA1          ; address MEM-3 start of digits
        LD      C,(IY+$71)        ; MEM-5-1st No. of digits to C
        LD      B,$00             ; prepare to add
        ADD     HL,BC             ; address last digit + 1
        LD      B,C               ; No. of digits to B counter
        POP     AF                ; restore A and carry flag from comparison.

;; PF-RND-LP
L2F18:  DEC     HL                ; address digit at rounding position.
        LD      A,(HL)            ; fetch it
        ADC     A,$00             ; add carry from the comparison
        LD      (HL),A            ; put back result even if $0A.
        AND     A                 ; test A
        JR      Z,L2F25           ; skip to PF-R-BACK if ZERO?

        CP      $0A               ; compare to 'ten' - overflow
        CCF                       ; complement carry flag so that set if ten.
        JR      NC,L2F2D          ; forward to PF-COUNT with 1 - 9.

;; PF-R-BACK
L2F25:  DJNZ    L2F18             ; loop back to PF-RND-LP

; if B counts down to zero then we've rounded right back as in 999999995.
; and the first 8 locations all hold $0A.


        LD      (HL),$01          ; load first location with digit 1.
        INC     B                 ; make B hold 1 also.
                                  ; could save an instruction byte here.
        INC     (IY+$72)          ; make MEM-5-2nd hold 1.
                                  ; and proceed to initialize total digits to 1.

;; PF-COUNT
L2F2D:  LD      (IY+$71),B        ; MEM-5-1st

; now balance the calculator stack by deleting  it

        RST     28H               ;; FP-CALC
        DEFB    $02               ;;delete
        DEFB    $38               ;;end-calc

; note if used from str$ then other values may be on the calculator stack.
; we can also restore the next literal pointer from its position on the
```

```
; machine stack.

        EXX                     ;
        POP     HL              ; restore next literal pointer.
        EXX                     ;

        LD      BC,($5CAB)      ; set C to MEM-5-1st digit counter.
                                ; set B to MEM-5-2nd leading digit counter.
        LD      HL,$5CA1        ; set HL to start of digits at MEM-3-1
        LD      A,B             ;
        CP      $09             ;
        JR      C,L2F46         ; to PF-NOT-E

        CP      $FC             ;
        JR      C,L2F6C         ; to PF-E-FRMT

;; PF-NOT-E
L2F46:  AND     A               ; test for zero leading digits as in .123

        CALL    Z,L15EF         ; routine OUT-CODE prints a zero e.g. 0.123

;; PF-E-SBRN
L2F4A:  XOR     A               ;
        SUB     B               ;
        JP      M,L2F52         ; skip forward to PF-OUT-LP if originally +ve

        LD      B,A             ; else negative count now +ve
        JR      L2F5E           ; forward to PF-DC-OUT       ->

; ---

;; PF-OUT-LP
L2F52:  LD      A,C             ; fetch total digit count
        AND     A               ; test for zero
        JR      Z,L2F59         ; forward to PF-OUT-DT if so

        LD      A,(HL)          ; fetch digit
        INC     HL              ; address next digit
        DEC     C               ; decrease total digit counter

;; PF-OUT-DT
L2F59:  CALL    L15EF           ; routine OUT-CODE outputs it.
        DJNZ    L2F52           ; loop back to PF-OUT-LP until B leading
                                ; digits output.

;; PF-DC-OUT
L2F5E:  LD      A,C             ; fetch total digits and
        AND     A               ; test if also zero
        RET     Z               ; return if so              -->

;

        INC     B               ; increment B
        LD      A,$2E           ; prepare the character '.'

;; PF-DEC-0$
L2F64:  RST     10H             ; PRINT-A outputs the character '.' or '0'

        LD      A,$30           ; prepare the character '0'
                                ; (for cases like .000012345678)
        DJNZ    L2F64           ; loop back to PF-DEC-0$ for B times.

        LD      B,C             ; load B with now trailing digit counter.
        JR      L2F52           ; back to PF-OUT-LP
```

```
; --------------------------------

; the branch was here for E-format printing e.g. 123456789 => 1.2345679e+8

;; PF-E-FRMT
L2F6C:  LD      D,B             ; counter to D
        DEC     D               ; decrement
        LD      B,$01           ; load B with 1.

        CALL    L2F4A           ; routine PF-E-SBRN above

        LD      A,$45           ; prepare character 'e'
        RST     10H             ; PRINT-A

        LD      C,D             ; exponent to C
        LD      A,C             ; and to A
        AND     A               ; test exponent
        JP      P,L2F83         ; to PF-E-POS if positive

        NEG                     ; negate
        LD      C,A             ; positive exponent to C
        LD      A,$2D           ; prepare character '-'
        JR      L2F85           ; skip to PF-E-SIGN

; ---

;; PF-E-POS
L2F83:  LD      A,$2B           ; prepare character '+'

;; PF-E-SIGN
L2F85:  RST     10H             ; PRINT-A outputs the sign

        LD      B,$00           ; make the high byte zero.
        JP      L1A1B           ; exit via OUT-NUM-1 to print exponent in BC

; ------------------------------
; Handle printing floating point
; ------------------------------
; This subroutine is called twice from above when printing floating-point
; numbers. It returns 10*A +C in registers C and A

;; CA-10*A+C
L2F8B:  PUSH    DE              ; preserve DE.
        LD      L,A             ; transfer A to L
        LD      H,$00           ; zero high byte.
        LD      E,L             ; copy HL
        LD      D,H             ; to DE.
        ADD     HL,HL           ; double (*2)
        ADD     HL,HL           ; double (*4)
        ADD     HL,DE           ; add DE (*5)
        ADD     HL,HL           ; double (*10)
        LD      E,C             ; copy C to E    (D is 0)
        ADD     HL,DE           ; and add to give required result.
        LD      C,H             ; transfer to
        LD      A,L             ; destination registers.
        POP     DE              ; restore DE
        RET                     ; return with result.

; --------------
; Prepare to add
; --------------
; This routine is called twice by addition to prepare the two numbers. The
; exponent is picked up in A and the location made zero. Then the sign bit
```

```
; is tested before being set to the implied state. Negative numbers are twos
; complemented.

;; PREP-ADD
L2F9B:  LD      A,(HL)          ; pick up exponent
        LD      (HL),$00        ; make location zero
        AND     A               ; test if number is zero
        RET     Z               ; return if so

        INC     HL              ; address mantissa
        BIT     7,(HL)          ; test the sign bit
        SET     7,(HL)          ; set it to implied state
        DEC     HL              ; point to exponent
        RET     Z               ; return if positive number.

        PUSH    BC              ; preserve BC
        LD      BC,$0005        ; length of number
        ADD     HL,BC           ; point HL past end
        LD      B,C             ; set B to 5 counter
        LD      C,A             ; store exponent in C
        SCF                     ; set carry flag

;; NEG-BYTE
L2FAF:  DEC     HL              ; work from LSB to MSB
        LD      A,(HL)          ; fetch byte
        CPL                     ; complement
        ADC     A,$00           ; add in initial carry or from prev operation
        LD      (HL),A          ; put back
        DJNZ    L2FAF           ; loop to NEG-BYTE till all 5 done

        LD      A,C             ; stored exponent to A
        POP     BC              ; restore original BC
        RET                     ; return

; -----------------
; Fetch two numbers
; -----------------
; This routine is called twice when printing floating point numbers and also
; to fetch two numbers by the addition, multiply and division routines.
; HL addresses the first number, DE addresses the second number.
; For arithmetic only, A holds the sign of the result which is stored in
; the second location.

;; FETCH-TWO
L2FBA:  PUSH    HL              ; save pointer to first number, result if math.
        PUSH    AF              ; save result sign.

        LD      C,(HL)          ;
        INC     HL              ;

        LD      B,(HL)          ;
        LD      (HL),A          ; store the sign at correct location in
                                ; destination 5 bytes for arithmetic only.
        INC     HL              ;

        LD      A,C             ;
        LD      C,(HL)          ;
        PUSH    BC              ;
        INC     HL              ;
        LD      C,(HL)          ;
        INC     HL              ;
        LD      B,(HL)          ;
        EX      DE,HL           ;
        LD      D,A             ;
```

```
        LD      E,(HL)          ;
        PUSH    DE              ;
        INC     HL              ;
        LD      D,(HL)          ;
        INC     HL              ;
        LD      E,(HL)          ;
        PUSH    DE              ;
        EXX                     ;
        POP     DE              ;
        POP     HL              ;
        POP     BC              ;
        EXX                     ;
        INC     HL              ;
        LD      D,(HL)          ;
        INC     HL              ;
        LD      E,(HL)          ;

        POP     AF              ; restore possible result sign.
        POP     HL              ; and pointer to possible result.
        RET                     ; return.

; ---------------------------------
; Shift floating point number right
; ---------------------------------
;
;

;; SHIFT-FP
L2FDD:  AND     A               ;
        RET     Z               ;

        CP      $21             ;
        JR      NC,L2FF9        ; to ADDEND-0

        PUSH    BC              ;
        LD      B,A             ;

;; ONE-SHIFT
L2FE5:  EXX                     ;
        SRA     L               ;
        RR      D               ;
        RR      E               ;
        EXX                     ;
        RR      D               ;
        RR      E               ;
        DJNZ    L2FE5           ; to ONE-SHIFT

        POP     BC              ;
        RET     NC              ;

        CALL    L3004           ; routine ADD-BACK
        RET     NZ              ;

;; ADDEND-0
L2FF9:  EXX                     ;
        XOR     A               ;

;; ZEROS-4/5
L2FFB:  LD      L,$00           ;
        LD      D,A             ;
        LD      E,L             ;
        EXX                     ;
        LD      DE,$0000        ;
        RET                     ;
```

```
; ------------------
; Add back any carry
; ------------------
;
;

;; ADD-BACK
L3004:  INC     E               ;
        RET     NZ              ;

        INC      D              ;
        RET     NZ              ;

        EXX                     ;
        INC     E               ;
        JR      NZ,L300D        ; to ALL-ADDED

        INC     D               ;

;; ALL-ADDED
L300D:  EXX                     ;
        RET                     ;

; -----------------------
; Handle subtraction (03)
; -----------------------
; Subtraction is done by switching the sign byte/bit of the second number
; which may be integer of floating point and continuing into addition.

;; subtract
L300F:  EX      DE,HL           ; address second number with HL

        CALL    L346E           ; routine NEGATE switches sign

        EX      DE,HL           ; address first number again
                                ; and continue.

; --------------------
; Handle addition (0F)
; --------------------
; HL points to first number, DE to second.
; If they are both integers, then go for the easy route.

;; addition
L3014:  LD      A,(DE)          ; fetch first byte of second
        OR      (HL)            ; combine with first byte of first
        JR      NZ,L303E        ; forward to FULL-ADDN if at least one was
                                ; in floating point form.

; continue if both were small integers.

        PUSH    DE              ; save pointer to lowest number for result.

        INC     HL              ; address sign byte and
        PUSH    HL              ; push the pointer.

        INC     HL              ; address low byte
        LD      E,(HL)          ; to E
        INC     HL              ; address high byte
        LD      D,(HL)          ; to D
        INC     HL              ; address unused byte

        INC     HL              ; address known zero indicator of 1st number
```

```
        INC     HL              ; address sign byte

        LD      A,(HL)          ; sign to A, $00 or $FF

        INC     HL              ; address low byte
        LD      C,(HL)          ; to C
        INC     HL              ; address high byte
        LD      B,(HL)          ; to B

        POP     HL              ; pop result sign pointer
        EX      DE,HL           ; integer to HL

        ADD     HL,BC           ; add to the other one in BC
                                ; setting carry if overflow.

        EX      DE,HL           ; save result in DE bringing back sign pointer

        ADC     A,(HL)          ; if pos/pos A=01 with overflow else 00
                                ; if neg/neg A=FF with overflow else FE
                                ; if mixture A=00 with overflow else FF

        RRCA                    ; bit 0 to (C)

        ADC     A,$00           ; both acceptable signs now zero

        JR      NZ,L303C        ; forward to ADDN-OFLW if not

        SBC     A,A             ; restore a negative result sign

        LD      (HL),A          ;
        INC     HL              ;
        LD      (HL),E          ;
        INC     HL              ;
        LD      (HL),D          ;
        DEC     HL              ;
        DEC     HL              ;
        DEC     HL              ;

        POP     DE              ; STKEND
        RET                     ;

; ---

;; ADDN-OFLW
L303C:  DEC     HL              ;
        POP     DE              ;

;; FULL-ADDN
L303E:  CALL    L3293           ; routine RE-ST-TWO
        EXX                     ;
        PUSH    HL              ;
        EXX                     ;
        PUSH    DE              ;
        PUSH    HL              ;
        CALL    L2F9B           ; routine PREP-ADD
        LD      B,A             ;
        EX      DE,HL           ;
        CALL    L2F9B           ; routine PREP-ADD
        LD       C,A            ;
        CP      B               ;
        JR      NC,L3055        ; to SHIFT-LEN

        LD      A,B             ;
        LD      B,C             ;
```

```
        EX      DE,HL           ;

;; SHIFT-LEN
L3055:  PUSH    AF              ;
        SUB     B               ;
        CALL    L2FBA           ; routine FETCH-TWO
        CALL    L2FDD           ; routine SHIFT-FP
        POP     AF              ;
        POP     HL              ;
        LD      (HL),A          ;
        PUSH    HL              ;
        LD      L,B             ;
        LD      H,C             ;
        ADD     HL,DE           ;
        EXX                     ;
        EX      DE,HL           ;
        ADC     HL,BC           ;
        EX      DE,HL           ;
        LD      A,H             ;
        ADC     A,L             ;
        LD      L,A             ;
        RRA                     ;
        XOR     L               ;
        EXX                     ;
        EX      DE,HL           ;
        POP     HL              ;
        RRA                     ;
        JR      NC,L307C        ; to TEST-NEG

        LD      A,$01           ;
        CALL    L2FDD           ; routine SHIFT-FP
        INC     (HL)            ;
        JR      Z,L309F         ; to ADD-REP-6

;; TEST-NEG
L307C:  EXX                     ;
        LD      A,L             ;
        AND     $80             ;
        EXX                     ;
        INC     HL              ;
        LD      (HL),A          ;
        DEC     HL              ;
        JR      Z,L30A5         ; to GO-NC-MLT

        LD      A,E             ;
        NEG                     ; Negate
        CCF                     ; Complement Carry Flag
        LD      E,A             ;
        LD      A,D             ;
        CPL                     ;
        ADC     A,$00           ;
        LD      D,A             ;
        EXX                     ;
        LD      A,E             ;
        CPL                     ;
        ADC     A,$00           ;
        LD      E,A             ;
        LD      A,D             ;
        CPL                     ;
        ADC     A,$00           ;
        JR      NC,L30A3        ; to END-COMPL

        RRA                     ;
        EXX                     ;
```

```
        INC     (HL)            ;

;; ADD-REP-6
L309F:  JP      Z,L31AD         ; to REPORT-6

        EXX                     ;

;; END-COMPL
L30A3:  LD      D,A             ;
        EXX                     ;

;; GO-NC-MLT
L30A5:  XOR     A               ;
        JP      L3155           ; to TEST-NORM

; ----------------------------
; Used in 16 bit multiplication
; ----------------------------
; This routine is used, in the first instance, by the multiply calculator
; literal to perform an integer multiplication in preference to
; 32-bit multiplication to which it will resort if this overflows.
;
; It is also used by STK-VAR to calculate array subscripts and by DIM to
; calculate the space required for multi-dimensional arrays.

;; HL-HL*DE
L30A9:  PUSH    BC              ; preserve BC throughout
        LD      B,$10           ; set B to 16
        LD      A,H             ; save H in A high byte
        LD      C,L             ; save L in C low byte
        LD      HL,$0000        ; initialize result to zero

; now enter a loop.

;; HL-LOOP
L30B1:  ADD     HL,HL           ; double result
        JR      C,L30BE         ; to HL-END if overflow

        RL      C               ; shift AC left into carry
        RLA                     ;
        JR      NC,L30BC        ; to HL-AGAIN to skip addition if no carry

        ADD     HL,DE           ; add in DE
        JR      C,L30BE         ; to HL-END if overflow

;; HL-AGAIN
L30BC:  DJNZ    L30B1           ; back to HL-LOOP for all 16 bits

;; HL-END
L30BE:  POP     BC              ; restore preserved BC
        RET                     ; return with carry reset if successful
                                ; and result in HL.

; --------------------------------------------
; THE 'PREPARE TO MULTIPLY OR DIVIDE' SUBROUTINE
; --------------------------------------------
;   This routine is called in succession from multiply and divide to prepare
;   two mantissas by setting the leftmost bit that is used for the sign.
;   On the first call A holds zero and picks up the sign bit. On the second
;   call the two bits are XORed to form the result sign - minus * minus giving
;   plus etc. If either number is zero then this is flagged.
;   HL addresses the exponent.

;; PREP-M/D
```

```
L30C0:  CALL    L34E9           ; routine TEST-ZERO  preserves accumulator.
        RET     C               ; return carry set if zero

        INC     HL              ; address first byte of mantissa
        XOR     (HL)            ; pick up the first or xor with first.
        SET     7,(HL)          ; now set to give true 32-bit mantissa
        DEC     HL              ; point to exponent
        RET                     ; return with carry reset

; ---------------------------
; Handle multiplication (04)
; ---------------------------
;
;

;; multiply
L30CA:  LD      A,(DE)          ;
        OR      (HL)            ;
        JR      NZ,L30F0        ; to MULT-LONG

        PUSH    DE              ;
        PUSH    HL              ;
        PUSH    DE              ;
        CALL    L2D7F           ; routine INT-FETCH
        EX      DE,HL           ;
        EX      (SP),HL         ;
        LD      B,C             ;
        CALL    L2D7F           ; routine INT-FETCH
        LD      A,B             ;
        XOR     C               ;
        LD      C,A             ;
        POP     HL              ;
        CALL    L30A9           ; routine HL-HL*DE
        EX      DE,HL           ;
        POP     HL              ;
        JR      C,L30EF         ; to MULT-OFLW

        LD      A,D             ;
        OR      E               ;
        JR      NZ,L30EA        ; to MULT-RSLT

        LD      C,A             ;

;; MULT-RSLT
L30EA:  CALL    L2D8E           ; routine INT-STORE
        POP      DE             ;
        RET                     ;

; ---

;; MULT-OFLW
L30EF:  POP     DE              ;

;; MULT-LONG
L30F0:  CALL    L3293           ; routine RE-ST-TWO
        XOR     A               ;
        CALL    L30C0           ; routine PREP-M/D
        RET     C               ;

        EXX                     ;
        PUSH    HL              ;
        EXX                     ;
        PUSH    DE              ;
        EX      DE,HL           ;
```

```
        CALL    L30C0           ; routine PREP-M/D
        EX      DE,HL           ;
        JR      C,L315D         ; to ZERO-RSLT

        PUSH    HL              ;
        CALL    L2FBA           ; routine FETCH-TWO
        LD      A,B             ;
        AND     A               ;
        SBC     HL,HL           ;
        EXX                     ;
        PUSH    HL              ;
        SBC     HL,HL           ;
        EXX                     ;
        LD      B,$21           ;
        JR      L3125           ; to STRT-MLT

; ---

;; MLT-LOOP
L3114:  JR      NC,L311B        ; to NO-ADD

        ADD     HL,DE           ;
        EXX                     ;
        ADC     HL,DE           ;
        EXX                     ;

;; NO-ADD
L311B:  EXX                     ;
        RR      H               ;
        RR      L               ;
        EXX                     ;
        RR      H               ;
        RR      L               ;

;; STRT-MLT
L3125:  EXX                     ;
        RR      B               ;
        RR      C               ;
        EXX                     ;
        RR      C               ;
        RRA                     ;
        DJNZ    L3114           ; to MLT-LOOP

        EX      DE,HL           ;
        EXX                     ;
        EX      DE,HL           ;
        EXX                     ;
        POP     BC              ;
        POP     HL              ;
        LD      A,B             ;
        ADD     A,C             ;
        JR      NZ,L313B        ; to MAKE-EXPT

        AND     A               ;

;; MAKE-EXPT
L313B:  DEC     A               ;
        CCF                     ; Complement Carry Flag

;; DIVN-EXPT
L313D:  RLA                     ;
        CCF                     ; Complement Carry Flag
        RRA                     ;
        JP      P,L3146         ; to OFLW1-CLR
```

```
        JR      NC,L31AD        ; to REPORT-6

        AND     A               ;

;; OFLW1-CLR
L3146:  INC     A               ;
        JR      NZ,L3151        ; to OFLW2-CLR

        JR      C,L3151         ; to OFLW2-CLR

        EXX                     ;
        BIT     7,D             ;
        EXX                     ;
        JR      NZ,L31AD        ; to REPORT-6

;; OFLW2-CLR
L3151:  LD      (HL),A          ;
        EXX                     ;
        LD      A,B             ;
        EXX                     ;

;; TEST-NORM
L3155:  JR      NC,L316C        ; to NORMALISE

        LD      A,(HL)          ;
        AND     A               ;

;; NEAR-ZERO
L3159:  LD      A,$80           ;
        JR      Z,L315E         ; to SKIP-ZERO

;; ZERO-RSLT
L315D:  XOR     A               ;

;; SKIP-ZERO
L315E:  EXX                     ;
        AND     D               ;
        CALL    L2FFB           ; routine ZEROS-4/5
        RLCA                    ;
        LD      (HL),A          ;
        JR      C,L3195         ; to OFLOW-CLR

        INC     HL              ;
        LD      (HL),A          ;
        DEC     HL              ;
        JR      L3195           ; to OFLOW-CLR

; ---

;; NORMALISE
L316C:  LD      B,$20           ;

;; SHIFT-ONE
L316E:  EXX                     ;
        BIT     7,D             ;
        EXX                     ;
        JR      NZ,L3186        ; to NORML-NOW

        RLCA                    ;
        RL      E               ;
        RL      D               ;
        EXX                     ;
        RL      E               ;
```

```
        RL      D               ;
        EXX                     ;
        DEC     (HL)            ;
        JR      Z,L3159         ; to NEAR-ZERO

        DJNZ    L316E           ; to SHIFT-ONE

        JR      L315D           ; to ZERO-RSLT

; ---

;; NORML-NOW
L3186:  RLA                     ;
        JR      NC,L3195        ; to OFLOW-CLR

        CALL    L3004           ; routine ADD-BACK
        JR      NZ,L3195        ; to OFLOW-CLR

        EXX                     ;
        LD       D,$80          ;
        EXX                     ;
        INC     (HL)            ;
        JR      Z,L31AD         ; to REPORT-6

;; OFLOW-CLR
L3195:  PUSH    HL              ;
        INC     HL              ;
        EXX                     ;
        PUSH    DE              ;
        EXX                     ;
        POP     BC              ;
        LD      A,B             ;
        RLA                     ;
        RL      (HL)            ;
        RRA                     ;
        LD      (HL),A          ;
        INC     HL              ;
        LD      (HL),C          ;
        INC     HL              ;
        LD      (HL),D          ;
        INC     HL              ;
        LD      (HL),E          ;
        POP     HL              ;
        POP     DE              ;
        EXX                     ;
        POP     HL              ;
        EXX                     ;
        RET                     ;

; ---

;; REPORT-6
L31AD:  RST     08H             ; ERROR-1
        DEFB    $05             ; Error Report: Number too big

; --------------------
; Handle division (05)
; --------------------
;
;

;; division
L31AF:  CALL    L3293           ; routine RE-ST-TWO
        EX      DE,HL           ;
```

```
        XOR     A               ;
        CALL    L30C0           ; routine PREP-M/D
        JR      C,L31AD         ; to REPORT-6

        EX      DE,HL           ;
        CALL    L30C0           ; routine PREP-M/D
        RET     C               ;

        EXX                     ;
        PUSH    HL              ;
        EXX                     ;
        PUSH    DE              ;
        PUSH    HL              ;
        CALL    L2FBA           ; routine FETCH-TWO
        EXX                     ;
        PUSH    HL              ;
        LD      H,B             ;
        LD      L,C             ;
        EXX                     ;
        LD      H,C             ;
        LD      L,B             ;
        XOR     A               ;
        LD      B,$DF           ;
        JR      L31E2           ; to DIV-START

; ---

;; DIV-LOOP
L31D2:  RLA                     ;
        RL      C               ;
        EXX                     ;
        RL      C               ;
        RL      B               ;
        EXX                     ;

;; div-34th
L31DB:  ADD     HL,HL           ;
        EXX                     ;
        ADC     HL,HL           ;
        EXX                     ;
        JR      C,L31F2         ; to SUBN-ONLY

;; DIV-START
L31E2:  SBC     HL,DE           ;
        EXX                     ;
        SBC     HL,DE           ;
        EXX                     ;
        JR      NC,L31F9        ; to NO-RSTORE

        ADD     HL,DE           ;
        EXX                     ;
        ADC     HL,DE           ;
        EXX                     ;
        AND     A               ;
        JR      L31FA           ; to COUNT-ONE

; ---

;; SUBN-ONLY
L31F2:  AND     A               ;
        SBC     HL,DE           ;
        EXX                     ;
        SBC     HL,DE           ;
        EXX                     ;
```

```
;; NO-RSTORE
L31F9:  SCF                     ; Set Carry Flag

;; COUNT-ONE
L31FA:  INC     B               ;
        JP      M,L31D2         ; to DIV-LOOP

        PUSH    AF              ;
        JR      Z,L31E2         ; to DIV-START

;
;
;
;

        LD      E,A             ;
        LD      D,C             ;
        EXX                     ;
        LD      E,C             ;
        LD      D,B             ;
        POP     AF              ;
        RR      B               ;
        POP     AF              ;
        RR      B               ;
        EXX                     ;
        POP     BC              ;
        POP     HL              ;
        LD      A,B             ;
        SUB     C               ;
        JP      L313D           ; jump back to DIVN-EXPT

; ------------------------------------
; Integer truncation towards zero ($3A)
; ------------------------------------
;
;

;; truncate
L3214:  LD      A,(HL)          ;
        AND     A               ;
        RET     Z               ;

        CP      $81             ;
        JR      NC,L3221        ; to T-GR-ZERO

        LD      (HL),$00        ;
        LD      A,$20           ;
        JR      L3272           ; to NIL-BYTES

; ---

;; T-GR-ZERO
L3221:  CP      $91             ;
        JR      NZ,L323F        ; to T-SMALL

        INC     HL              ;
        INC     HL              ;
        INC     HL              ;
        LD      A,$80           ;
        AND     (HL)            ;
        DEC     HL              ;
        OR      (HL)            ;
        DEC     HL              ;
```

```
        JR      NZ,L3233        ; to T-FIRST

        LD      A,$80           ;
        XOR     (HL)            ;

;; T-FIRST
L3233:  DEC     HL              ;
        JR      NZ,L326C        ; to T-EXPNENT

        LD      (HL),A          ;
        INC     HL              ;
        LD      (HL),$FF        ;
        DEC     HL              ;
        LD      A,$18           ;
        JR      L3272           ; to NIL-BYTES

; ---

;; T-SMALL
L323F:  JR      NC,L326D        ; to X-LARGE

        PUSH    DE              ;
        CPL                     ;
        ADD     A,$91           ;
        INC     HL              ;
        LD      D,(HL)          ;
        INC     HL              ;
        LD      E,(HL)          ;
        DEC     HL              ;
        DEC     HL              ;
        LD      C,$00           ;
        BIT     7,D             ;
        JR      Z,L3252         ; to T-NUMERIC

        DEC     C               ;

;; T-NUMERIC
L3252:  SET     7,D             ;
        LD      B,$08           ;
        SUB     B               ;
        ADD     A,B             ;
        JR      C,L325E         ; to T-TEST

        LD      E,D             ;
        LD      D,$00           ;
        SUB     B               ;

;; T-TEST
L325E:  JR      Z,L3267         ; to T-STORE

        LD      B,A             ;

;; T-SHIFT
L3261:  SRL     D               ;
        RR      E               ;
        DJNZ    L3261           ; to T-SHIFT

;; T-STORE
L3267:  CALL    L2D8E           ; routine INT-STORE
        POP     DE              ;
        RET                     ;

; ---
```

```
;; T-EXPNENT
L326C:  LD      A,(HL)          ;

;; X-LARGE
L326D:  SUB     $A0             ;
        RET     P               ;

        NEG                     ; Negate

;; NIL-BYTES
L3272:  PUSH    DE              ;
        EX      DE,HL           ;
        DEC     HL              ;
        LD      B,A             ;
        SRL     B               ;
        SRL     B               ;
        SRL     B               ;
        JR      Z,L3283         ; to BITS-ZERO

;; BYTE-ZERO
L327E:  LD      (HL),$00        ;
        DEC     HL              ;
        DJNZ    L327E           ; to BYTE-ZERO

;; BITS-ZERO
L3283:  AND     $07             ;
        JR      Z,L3290         ; to IX-END

        LD      B,A             ;
        LD      A,$FF           ;

;; LESS-MASK
L328A:  SLA     A               ;
        DJNZ    L328A           ; to LESS-MASK

        AND     (HL)            ;
        LD      (HL),A          ;

;; IX-END
L3290:  EX      DE,HL           ;
        POP     DE              ;
        RET                     ;

; ----------------------------------
; Storage of numbers in 5 byte form.
; ==================================
; Both integers and floating-point numbers can be stored in five bytes.
; Zero is a special case stored as 5 zeros.
; For integers the form is
; Byte 1 - zero,
; Byte 2 - sign byte, $00 +ve, $FF -ve.
; Byte 3 - Low byte of integer.
; Byte 4 - High byte
; Byte 5 - unused but always zero.
;
; it seems unusual to store the low byte first but it is just as easy either
; way. Statistically it just increases the chances of trailing zeros which
; is an advantage elsewhere in saving ROM code.
;
;             zero     sign     low      high    unused
; So +1 is  00000000 00000000 00000001 00000000 00000000
;
; and -1 is 00000000 11111111 11111111 11111111 00000000
;
```

```
; much of the arithmetic found in BASIC lines can be done using numbers
; in this form using the Z80's 16 bit register operation ADD.
; (multiplication is done by a sequence of additions).
;
; Storing -ve integers in two's complement form, means that they are ready for
; addition and you might like to add the numbers above to prove that the
; answer is zero. If, as in this case, the carry is set then that denotes that
; the result is positive. This only applies when the signs don't match.
; With positive numbers a carry denotes the result is out of integer range.
; With negative numbers a carry denotes the result is within range.
; The exception to the last rule is when the result is -65536
;
; Floating point form is an alternative method of storing numbers which can
; be used for integers and larger (or fractional) numbers.
;
; In this form 1 is stored as
;          10000001 00000000 00000000 00000000 00000000
;
; When a small integer is converted to a floating point number the last two
; bytes are always blank so they are omitted in the following steps
;
; first make exponent +1 +16d  (bit 7 of the exponent is set if positive)

; 10010001 00000000 00000001
; 10010000 00000000 00000010 <-  now shift left and decrement exponent
; ...
; 10000010 01000000 00000000 <-  until a 1 abuts the imaginary point
; 10000001 10000000 00000000     to the left of the mantissa.
;
; however since the leftmost bit of the mantissa is always set then it can
; be used to denote the sign of the mantissa and put back when needed by the
; PREP routines which gives
;
; 10000001 00000000 00000000

; ----------------------------------------------
; THE 'RE-STACK TWO "SMALL" INTEGERS' SUBROUTINE
; ----------------------------------------------
;   This routine is called to re-stack two numbers in full floating point form
;   e.g. from mult when integer multiplication has overflowed.

;; RE-ST-TWO
L3293:  CALL    L3296           ; routine RESTK-SUB  below and continue
                                ; into the routine to do the other one.

;; RESTK-SUB
L3296:  EX      DE,HL           ; swap pointers

; ---------------------------------------------
; THE 'RE-STACK ONE "SMALL" INTEGER' SUBROUTINE
; ---------------------------------------------
; (offset: $3D 're-stack')
;   This routine re-stacks an integer, usually on the calculator stack, in full
;   floating point form.  HL points to first byte.

;; re-stack
L3297:  LD      A,(HL)          ; Fetch Exponent byte to A
        AND     A               ; test it
        RET     NZ              ; return if not zero as already in full
                                ; floating-point form.

        PUSH    DE              ; preserve DE.
        CALL    L2D7F           ; routine INT-FETCH
                                ; integer to DE, sign to C.
```

```
; HL points to 4th byte.

        XOR     A               ; clear accumulator.
        INC     HL              ; point to 5th.
        LD      (HL),A          ; and blank.
        DEC     HL              ; point to 4th.
        LD      (HL),A          ; and blank.

        LD      B,$91           ; set exponent byte +ve $81
                                ; and imaginary dec point 16 bits to right
                                ; of first bit.

;   we could skip to normalize now but it's quicker to avoid normalizing
;   through an empty D.

        LD      A,D             ; fetch the high byte D
        AND     A               ; is it zero ?
        JR      NZ,L32B1        ; skip to RS-NRMLSE if not.

        OR      E               ; low byte E to A and test for zero
        LD      B,D             ; set B exponent to 0
        JR      Z,L32BD         ; forward to RS-STORE if value is zero.

        LD      D,E             ; transfer E to D
        LD      E,B             ; set E to 0
        LD      B,$89           ; reduce the initial exponent by eight.


;; RS-NRMLSE
L32B1:  EX      DE,HL           ; integer to HL, addr of 4th byte to DE.

;; RSTK-LOOP
L32B2:  DEC     B               ; decrease exponent
        ADD     HL,HL           ; shift DE left
        JR      NC,L32B2        ; loop back to RSTK-LOOP
                                ; until a set bit pops into carry

        RRC     C               ; now rotate the sign byte $00 or $FF
                                ; into carry to give a sign bit

        RR      H               ; rotate the sign bit to left of H
        RR      L               ; rotate any carry into L

        EX      DE,HL           ; address 4th byte, normalized int to DE

;; RS-STORE
L32BD:  DEC     HL              ; address 3rd byte
        LD      (HL),E          ; place E
        DEC     HL              ; address 2nd byte
        LD      (HL),D          ; place D
        DEC     HL              ; address 1st byte
        LD      (HL),B          ; store the exponent

        POP     DE              ; restore initial DE.
        RET                     ; return.

;****************************************
;** Part 10. FLOATING-POINT CALCULATOR **
;****************************************

; As a general rule the calculator avoids using the IY register.
; exceptions are val, val$ and str$.
; So an assembly language programmer who has disabled interrupts to use
```

```
; IY for other purposes can still use the calculator for mathematical
; purposes.


; ------------------------
; THE 'TABLE OF CONSTANTS'
; ------------------------
;
;

; used 11 times
;; stk-zero                                                 00 00 00 00 00
L32C5:  DEFB    $00             ;;Bytes: 1
        DEFB    $B0             ;;Exponent $00
        DEFB    $00             ;;(+00,+00,+00)

; used 19 times
;; stk-one                                                  00 00 01 00 00
L32C8:  DEFB    $40             ;;Bytes: 2
        DEFB    $B0             ;;Exponent $00
        DEFB    $00,$01         ;;(+00,+00)

; used 9 times
;; stk-half                                                 80 00 00 00 00
L32CC:  DEFB    $30             ;;Exponent: $80, Bytes: 1
        DEFB    $00             ;;(+00,+00,+00)

; used 4 times.
;; stk-pi/2                                                 81 49 0F DA A2
L32CE:  DEFB    $F1             ;;Exponent: $81, Bytes: 4
        DEFB    $49,$0F,$DA,$A2 ;;

; used 3 times.
;; stk-ten                                                  00 00 0A 00 00
L32D3:  DEFB    $40             ;;Bytes: 2
        DEFB    $B0             ;;Exponent $00
        DEFB    $00,$0A         ;;(+00,+00)


; ------------------------
; THE 'TABLE OF ADDRESSES'
; ------------------------
;
;   Starts with binary operations which have two operands and one result.
;   Three pseudo binary operations first.

;; tbl-addrs
L32D7:  DEFW    L368F           ; $00 Address: $368F - jump-true
        DEFW    L343C           ; $01 Address: $343C - exchange
        DEFW    L33A1           ; $02 Address: $33A1 - delete

;   True binary operations.

        DEFW    L300F           ; $03 Address: $300F - subtract
        DEFW    L30CA           ; $04 Address: $30CA - multiply
        DEFW    L31AF           ; $05 Address: $31AF - division
        DEFW    L3851           ; $06 Address: $3851 - to-power
        DEFW    L351B           ; $07 Address: $351B - or

        DEFW    L3524           ; $08 Address: $3524 - no-&-no
        DEFW    L353B           ; $09 Address: $353B - no-l-eql
        DEFW    L353B           ; $0A Address: $353B - no-gr-eql
        DEFW    L353B           ; $0B Address: $353B - nos-neql
        DEFW    L353B           ; $0C Address: $353B - no-grtr
```

```
        DEFW    L353B           ; $0D Address: $353B - no-less
        DEFW    L353B           ; $0E Address: $353B - nos-eql
        DEFW    L3014           ; $0F Address: $3014 - addition

        DEFW    L352D           ; $10 Address: $352D - str-&-no
        DEFW    L353B           ; $11 Address: $353B - str-l-eql
        DEFW    L353B           ; $12 Address: $353B - str-gr-eql
        DEFW    L353B           ; $13 Address: $353B - strs-neql
        DEFW    L353B           ; $14 Address: $353B - str-grtr
        DEFW    L353B           ; $15 Address: $353B - str-less
        DEFW    L353B           ; $16 Address: $353B - strs-eql
        DEFW    L359C           ; $17 Address: $359C - strs-add

;   Unary follow.

        DEFW    L35DE           ; $18 Address: $35DE - val$
        DEFW    L34BC           ; $19 Address: $34BC - usr-$
        DEFW    L3645           ; $1A Address: $3645 - read-in
        DEFW    L346E           ; $1B Address: $346E - negate

        DEFW    L3669           ; $1C Address: $3669 - code
        DEFW    L35DE           ; $1D Address: $35DE - val
        DEFW    L3674           ; $1E Address: $3674 - len
        DEFW    L37B5           ; $1F Address: $37B5 - sin
        DEFW    L37AA           ; $20 Address: $37AA - cos
        DEFW    L37DA           ; $21 Address: $37DA - tan
        DEFW    L3833           ; $22 Address: $3833 - asn
        DEFW    L3843           ; $23 Address: $3843 - acs
        DEFW    L37E2           ; $24 Address: $37E2 - atn
        DEFW    L3713           ; $25 Address: $3713 - ln
        DEFW    L36C4           ; $26 Address: $36C4 - exp
        DEFW    L36AF           ; $27 Address: $36AF - int
        DEFW    L384A           ; $28 Address: $384A - sqr
        DEFW    L3492           ; $29 Address: $3492 - sgn
        DEFW    L346A           ; $2A Address: $346A - abs
        DEFW    L34AC           ; $2B Address: $34AC - peek
        DEFW    L34A5           ; $2C Address: $34A5 - in
        DEFW    L34B3           ; $2D Address: $34B3 - usr-no
        DEFW    L361F           ; $2E Address: $361F - str$
        DEFW    L35C9           ; $2F Address: $35C9 - chrs
        DEFW    L3501           ; $30 Address: $3501 - not

;   End of true unary.

        DEFW    L33C0           ; $31 Address: $33C0 - duplicate
        DEFW    L36A0           ; $32 Address: $36A0 - n-mod-m
        DEFW    L3686           ; $33 Address: $3686 - jump
        DEFW    L33C6           ; $34 Address: $33C6 - stk-data
        DEFW    L367A           ; $35 Address: $367A - dec-jr-nz
        DEFW    L3506           ; $36 Address: $3506 - less-0
        DEFW    L34F9           ; $37 Address: $34F9 - greater-0
        DEFW    L369B           ; $38 Address: $369B - end-calc
        DEFW    L3783           ; $39 Address: $3783 - get-argt
        DEFW    L3214           ; $3A Address: $3214 - truncate
        DEFW    L33A2           ; $3B Address: $33A2 - fp-calc-2
        DEFW    L2D4F           ; $3C Address: $2D4F - e-to-fp
        DEFW    L3297           ; $3D Address: $3297 - re-stack

;   The following are just the next available slots for the 128 compound
;   literals which are in range $80 - $FF.

        DEFW    L3449           ;     Address: $3449 - series-xx    $80 - $9F.
        DEFW    L341B           ;     Address: $341B - stk-const-xx $A0 - $BF.
        DEFW    L342D           ;     Address: $342D - st-mem-xx    $C0 - $DF.
```

```
        DEFW    L340F           ;     Address: $340F - get-mem-xx   $E0 - $FF.

;   Aside: 3E - 3F are therefore unused calculator literals.
;   If the literal has to be also usable as a function then bits 6 and 7 are
;   used to show type of arguments and result.

; --------------
; The Calculator
; --------------
;
;

;; CALCULATE
L335B:  CALL    L35BF           ; routine STK-PNTRS is called to set up the
                                ; calculator stack pointers for a default
                                ; unary operation. HL = last value on stack.
                                ; DE = STKEND first location after stack.

; the calculate routine is called at this point by the series generator...

;; GEN-ENT-1
L335E:  LD      A,B             ; fetch the Z80 B register to A
        LD      ($5C67),A       ; and store value in system variable BREG.
                                ; this will be the counter for dec-jr-nz
                                ; or if used from fp-calc2 the calculator
                                ; instruction.

; ... and again later at this point

;; GEN-ENT-2
L3362:  EXX                     ; switch sets
        EX      (SP),HL         ; and store the address of next instruction,
                                ; the return address, in H'L'.
                                ; If this is a recursive call the H'L'
                                ; of the previous invocation goes on stack.
                                ; c.f. end-calc.
        EXX                     ; switch back to main set

; this is the re-entry looping point when handling a string of literals.

;; RE-ENTRY
L3365:  LD      ($5C65),DE      ; save end of stack in system variable STKEND
        EXX                     ; switch to alt
        LD      A,(HL)          ; get next literal
        INC     HL              ; increase pointer'

; single operation jumps back to here

;; SCAN-ENT
L336C:  PUSH    HL              ; save pointer on stack
        AND     A               ; now test the literal
        JP      P,L3380         ; forward to FIRST-3D if in range $00 - $3D
                                ; anything with bit 7 set will be one of
                                ; 128 compound literals.

; compound literals have the following format.
; bit 7 set indicates compound.
; bits 6-5 the subgroup 0-3.
; bits 4-0 the embedded parameter $00 - $1F.
; The subgroup 0-3 needs to be manipulated to form the next available four
; address places after the simple literals in the address table.

        LD      D,A             ; save literal in D
        AND     $60             ; and with 01100000 to isolate subgroup
```

```
        RRCA                        ; rotate bits
        RRCA                        ; 4 places to right
        RRCA                        ; not five as we need offset * 2
        RRCA                        ; 00000xx0
        ADD     A,$7C               ; add ($3E * 2) to give correct offset.
                                    ; alter above if you add more literals.
        LD      L,A                 ; store in L for later indexing.
        LD      A,D                 ; bring back compound literal
        AND     $1F                 ; use mask to isolate parameter bits
        JR      L338E               ; forward to ENT-TABLE

; ---

; the branch was here with simple literals.

;; FIRST-3D
L3380:  CP      $18                 ; compare with first unary operations.
        JR      NC,L338C            ; to DOUBLE-A with unary operations

; it is binary so adjust pointers.

        EXX                         ;
        LD      BC,$FFFB            ; the value -5
        LD      D,H                 ; transfer HL, the last value, to DE.
        LD      E,L                 ;
        ADD     HL,BC               ; subtract 5 making HL point to second
                                    ; value.
        EXX                         ;

;; DOUBLE-A
L338C:  RLCA                        ; double the literal
        LD      L,A                 ; and store in L for indexing

;; ENT-TABLE
L338E:  LD      DE,L32D7            ; Address: tbl-addrs
        LD      H,$00               ; prepare to index
        ADD     HL,DE               ; add to get address of routine
        LD      E,(HL)              ; low byte to E
        INC     HL                  ;
        LD      D,(HL)              ; high byte to D
        LD      HL,L3365            ; Address: RE-ENTRY
        EX      (SP),HL             ; goes to stack
        PUSH    DE                  ; now address of routine
        EXX                         ; main set
                                    ; avoid using IY register.
        LD      BC,($5C66)          ; STKEND_hi
                                    ; nothing much goes to C but BREG to B
                                    ; and continue into next ret instruction
                                    ; which has a dual identity


; ------------------
; Handle delete (02)
; ------------------
; A simple return but when used as a calculator literal this
; deletes the last value from the calculator stack.
; On entry, as always with binary operations,
; HL=first number, DE=second number
; On exit, HL=result, DE=stkend.
; So nothing to do

;; delete
L33A1:  RET                         ; return - indirect jump if from above.
```

```
; ---------------------
; Single operation (3B)
; ---------------------
; this single operation is used, in the first instance, to evaluate most
; of the mathematical and string functions found in BASIC expressions.

;; fp-calc-2
L33A2:  POP     AF              ; drop return address.
        LD      A,($5C67)       ; load accumulator from system variable BREG
                                ; value will be literal e.g. 'tan'
        EXX                     ; switch to alt
        JR      L336C           ; back to SCAN-ENT
                                ; next literal will be end-calc at L2758

; --------------------------------
; THE 'TEST FIVE SPACES' SUBROUTINE
; --------------------------------
;   This routine is called from MOVE-FP, STK-CONST and STK-STORE to test that
;   there is enough space between the calculator stack and the machine stack
;   for another five-byte value.  It returns with BC holding the value 5 ready
;   for any subsequent LDIR.

;; TEST-5-SP
L33A9:  PUSH    DE              ; save
        PUSH    HL              ; registers
        LD      BC,$0005        ; an overhead of five bytes
        CALL    L1F05           ; routine TEST-ROOM tests free RAM raising
                                ; an error if not.
        POP     HL              ; else restore
        POP     DE              ; registers.
        RET                     ; return with BC set at 5.

; ----------------------------
; THE 'STACK NUMBER' SUBROUTINE
; ----------------------------
;   This routine is called to stack a hidden floating point number found in
;   a BASIC line.  It is also called to stack a numeric variable value, and
;   from BEEP, to stack an entry in the semi-tone table.  It is not part of the
;   calculator suite of routines.  On entry, HL points to the number to be
;   stacked.

;; STACK-NUM
L33B4:  LD      DE,($5C65)      ; Load destination from STKEND system variable.

        CALL    L33C0           ; Routine MOVE-FP puts on calculator stack
                                ; with a memory check.
        LD      ($5C65),DE      ; Set STKEND to next free location.

        RET                     ; Return.

; ---------------------------------
; Move a floating point number (31)
; ---------------------------------

; This simple routine is a 5-byte LDIR instruction
; that incorporates a memory check.
; When used as a calculator literal it duplicates the last value on the
; calculator stack.
; Unary so on entry HL points to last value, DE to stkend

;; duplicate
;; MOVE-FP
L33C0:  CALL    L33A9           ; routine TEST-5-SP test free memory
                                ; and sets BC to 5.
```

```
        LDIR                        ; copy the five bytes.
        RET                         ; return with DE addressing new STKEND
                                    ; and HL addressing new last value.

; -------------------
; Stack literals ($34)
; -------------------
; When a calculator subroutine needs to put a value on the calculator
; stack that is not a regular constant this routine is called with a
; variable number of following data bytes that convey to the routine
; the integer or floating point form as succinctly as is possible.

;; stk-data
L33C6:  LD      H,D                 ; transfer STKEND
        LD      L,E                 ; to HL for result.

;; STK-CONST
L33C8:  CALL    L33A9               ; routine TEST-5-SP tests that room exists
                                    ; and sets BC to $05.

        EXX                         ; switch to alternate set
        PUSH    HL                  ; save the pointer to next literal on stack
        EXX                         ; switch back to main set

        EX      (SP),HL             ; pointer to HL, destination to stack.

        PUSH    BC                  ; save BC - value 5 from test room ??.

        LD      A,(HL)              ; fetch the byte following 'stk-data'
        AND     $C0                 ; isolate bits 7 and 6
        RLCA                        ; rotate
        RLCA                        ; to bits 1 and 0  range $00 - $03.
        LD      C,A                 ; transfer to C
        INC     C                   ; and increment to give number of bytes
                                    ; to read. $01 - $04
        LD      A,(HL)              ; reload the first byte
        AND     $3F                 ; mask off to give possible exponent.
        JR      NZ,L33DE            ; forward to FORM-EXP if it was possible to
                                    ; include the exponent.

; else byte is just a byte count and exponent comes next.

        INC     HL                  ; address next byte and
        LD      A,(HL)              ; pick up the exponent ( - $50).

;; FORM-EXP
L33DE:  ADD     A,$50               ; now add $50 to form actual exponent
        LD      (DE),A              ; and load into first destination byte.
        LD      A,$05               ; load accumulator with $05 and
        SUB     C                   ; subtract C to give count of trailing
                                    ; zeros plus one.
        INC     HL                  ; increment source
        INC     DE                  ; increment destination
        LD      B,$00               ; prepare to copy
        LDIR                        ; copy C bytes

        POP     BC                  ; restore 5 counter to BC ??.

        EX      (SP),HL             ; put HL on stack as next literal pointer
                                    ; and the stack value - result pointer -
                                    ; to HL.

        EXX                         ; switch to alternate set.
        POP     HL                  ; restore next literal pointer from stack
```

```
                                ; to H'L'.
        EXX                     ; switch back to main set.

        LD      B,A             ; zero count to B
        XOR     A               ; clear accumulator

;; STK-ZEROS
L33F1:  DEC     B               ; decrement B counter
        RET     Z               ; return if zero.          >>
                                ; DE points to new STKEND
                                ; HL to new number.

        LD      (DE),A          ; else load zero to destination
        INC     DE              ; increase destination
        JR      L33F1           ; loop back to STK-ZEROS until done.

; -------------------------------
; THE 'SKIP CONSTANTS' SUBROUTINE
; -------------------------------
;   This routine traverses variable-length entries in the table of constants,
;   stacking intermediate, unwanted constants onto a dummy calculator stack,
;   in the first five bytes of ROM.  The destination DE normally points to the
;   end of the calculator stack which might be in the normal place or in the
;   system variables area during E-LINE-NO; INT-TO-FP; stk-ten.  In any case,
;   it would be simpler all round if the routine just shoved unwanted values
;   where it is going to stick the wanted value.  The instruction LD DE, $0000
;   can be removed.

;; SKIP-CONS
L33F7:  AND     A               ; test if initially zero.

;; SKIP-NEXT
L33F8:  RET     Z               ; return if zero.          >>

        PUSH    AF              ; save count.
        PUSH    DE              ; and normal STKEND

        LD      DE,$0000        ; dummy value for STKEND at start of ROM
                                ; Note. not a fault but this has to be
                                ; moved elsewhere when running in RAM.
                                ; e.g. with Expandor Systems 'Soft ROM'.
                                ; Better still, write to the normal place.
        CALL    L33C8           ; routine STK-CONST works through variable
                                ; length records.

        POP     DE              ; restore real STKEND
        POP     AF              ; restore count
        DEC     A               ; decrease
        JR      L33F8           ; loop back to SKIP-NEXT

; ------------------------------
; THE 'LOCATE MEMORY' SUBROUTINE
; ------------------------------
;   This routine, when supplied with a base address in HL and an index in A,
;   will calculate the address of the A'th entry, where each entry occupies
;   five bytes.  It is used for reading the semi-tone table and addressing
;   floating-point numbers in the calculator's memory area.
;   It is not possible to use this routine for the table of constants as these
;   six values are held in compressed format.

;; LOC-MEM
L3406:  LD      C,A             ; store the original number $00-$1F.
        RLCA                    ; X2 - double.
        RLCA                    ; X4 - quadruple.
```

```
        ADD     A,C                 ; X5 - now add original to multiply by five.

        LD      C,A                 ; place the result in the low byte.
        LD      B,$00               ; set high byte to zero.
        ADD     HL,BC               ; add to form address of start of number in HL.

        RET                         ; return.

; ------------------------------
; Get from memory area ($E0 etc.)
; ------------------------------
; Literals $E0 to $FF
; A holds $00-$1F offset.
; The calculator stack increases by 5 bytes.

;; get-mem-xx
L340F:  PUSH    DE                  ; save STKEND
        LD      HL,($5C68)          ; MEM is base address of the memory cells.
        CALL    L3406               ; routine LOC-MEM so that HL = first byte
        CALL    L33C0               ; routine MOVE-FP moves 5 bytes with memory
                                    ; check.
                                    ; DE now points to new STKEND.
        POP     HL                  ; original STKEND is now RESULT pointer.
        RET                         ; return.

; --------------------------
; Stack a constant (A0 etc.)
; --------------------------
; This routine allows a one-byte instruction to stack up to 32 constants
; held in short form in a table of constants. In fact only 5 constants are
; required. On entry the A register holds the literal ANDed with 1F.
; It isn't very efficient and it would have been better to hold the
; numbers in full, five byte form and stack them in a similar manner
; to that used for semi-tone table values.

;; stk-const-xx
L341B:  LD      H,D                 ; save STKEND - required for result
        LD      L,E                 ;
        EXX                         ; swap
        PUSH    HL                  ; save pointer to next literal
        LD      HL,L32C5            ; Address: stk-zero - start of table of
                                    ; constants
        EXX                         ;
        CALL    L33F7               ; routine SKIP-CONS
        CALL    L33C8               ; routine STK-CONST
        EXX                         ;
        POP     HL                  ; restore pointer to next literal.
        EXX                         ;
        RET                         ; return.

; --------------------------------
; Store in a memory area ($C0 etc.)
; --------------------------------
; Offsets $C0 to $DF
; Although 32 memory storage locations can be addressed, only six
; $C0 to $C5 are required by the ROM and only the thirty bytes (6*5)
; required for these are allocated. Spectrum programmers who wish to
; use the floating point routines from assembly language may wish to
; alter the system variable MEM to point to 160 bytes of RAM to have
; use the full range available.
; A holds the derived offset $00-$1F.
; This is a unary operation, so on entry HL points to the last value and DE
; points to STKEND.
```

```
;; st-mem-xx
L342D:  PUSH    HL              ; save the result pointer.
        EX      DE,HL           ; transfer to DE.
        LD      HL,($5C68)      ; fetch MEM the base of memory area.
        CALL    L3406           ; routine LOC-MEM sets HL to the destination.
        EX      DE,HL           ; swap - HL is start, DE is destination.
        CALL    L33C0           ; routine MOVE-FP.
                                ; note. a short ld bc,5; ldir
                                ; the embedded memory check is not required
                                ; so these instructions would be faster.
        EX      DE,HL           ; DE = STKEND
        POP     HL              ; restore original result pointer
        RET                     ; return.

; -------------------------
; THE 'EXCHANGE' SUBROUTINE
; -------------------------
; (offset: $01 'exchange')
;   This routine swaps the last two values on the calculator stack.
;   On entry, as always with binary operations,
;   HL=first number, DE=second number
;   On exit, HL=result, DE=stkend.

;; exchange
L343C:  LD      B,$05           ; there are five bytes to be swapped

; start of loop.

;; SWAP-BYTE
L343E:  LD      A,(DE)          ; each byte of second
        LD      C,(HL)          ; each byte of first
        EX      DE,HL           ; swap pointers
        LD      (DE),A          ; store each byte of first
        LD      (HL),C          ; store each byte of second
        INC     HL              ; advance both
        INC     DE              ; pointers.
        DJNZ    L343E           ; loop back to SWAP-BYTE until all 5 done.

        EX      DE,HL           ; even up the exchanges so that DE addresses
                                ; STKEND.

        RET                     ; return.

; ------------------------------
; THE 'SERIES GENERATOR' ROUTINE
; ------------------------------
; (offset: $86 'series-06')
; (offset: $88 'series-08')
; (offset: $8C 'series-0C')
;   The Spectrum uses Chebyshev polynomials to generate approximations for
;   SIN, ATN, LN and EXP.  These are named after the Russian mathematician
;   Pafnuty Chebyshev, born in 1821, who did much pioneering work on numerical
;   series.  As far as calculators are concerned, Chebyshev polynomials have an
;   advantage over other series, for example the Taylor series, as they can
;   reach an approximation in just six iterations for SIN, eight for EXP and
;   twelve for LN and ATN.  The mechanics of the routine are interesting but
;   for full treatment of how these are generated with demonstrations in
;   Sinclair BASIC see "The Complete Spectrum ROM Disassembly" by Dr Ian Logan
;   and Dr Frank O'Hara, published 1983 by Melbourne House.

;; series-xx
L3449:  LD      B,A             ; parameter $00 - $1F to B counter
        CALL    L335E           ; routine GEN-ENT-1 is called.
                                ; A recursive call to a special entry point
```

```
                                ; in the calculator that puts the B register
                                ; in the system variable BREG. The return
                                ; address is the next location and where
                                ; the calculator will expect its first
                                ; instruction - now pointed to by HL'.
                                ; The previous pointer to the series of
                                ; five-byte numbers goes on the machine stack.

; The initialization phase.

        DEFB    $31             ;;duplicate       x,x
        DEFB    $0F             ;;addition        x+x
        DEFB    $C0             ;;st-mem-0        x+x
        DEFB    $02             ;;delete          .
        DEFB    $A0             ;;stk-zero        0
        DEFB    $C2             ;;st-mem-2        0

; a loop is now entered to perform the algebraic calculation for each of
; the numbers in the series

;; G-LOOP
L3453:  DEFB    $31             ;;duplicate       v,v.
        DEFB    $E0             ;;get-mem-0       v,v,x+2
        DEFB    $04             ;;multiply        v,v*x+2
        DEFB    $E2             ;;get-mem-2       v,v*x+2,v
        DEFB    $C1             ;;st-mem-1
        DEFB    $03             ;;subtract
        DEFB    $38             ;;end-calc

; the previous pointer is fetched from the machine stack to H'L' where it
; addresses one of the numbers of the series following the series literal.

        CALL    L33C6           ; routine STK-DATA is called directly to
                                ; push a value and advance H'L'.
        CALL    L3362           ; routine GEN-ENT-2 recursively re-enters
                                ; the calculator without disturbing
                                ; system variable BREG
                                ; H'L' value goes on the machine stack and is
                                ; then loaded as usual with the next address.

        DEFB    $0F             ;;addition
        DEFB    $01             ;;exchange
        DEFB    $C2             ;;st-mem-2
        DEFB    $02             ;;delete

        DEFB    $35             ;;dec-jr-nz
        DEFB    $EE             ;;back to L3453, G-LOOP

; when the counted loop is complete the final subtraction yields the result
; for example SIN X.

        DEFB    $E1             ;;get-mem-1
        DEFB    $03             ;;subtract
        DEFB    $38             ;;end-calc

        RET                     ; return with H'L' pointing to location
                                ; after last number in series.

; ---------------------------------
; THE 'ABSOLUTE MAGNITUDE' FUNCTION
; ---------------------------------
; (offset: $2A 'abs')
;   This calculator literal finds the absolute value of the last value,
;   integer or floating point, on calculator stack.
```

```
;; abs
L346A:  LD      B,$FF           ; signal abs
        JR      L3474           ; forward to NEG-TEST

; ---------------------------
; THE 'UNARY MINUS' OPERATION
; ---------------------------
; (offset: $1B 'negate')
;   Unary so on entry HL points to last value, DE to STKEND.

;; NEGATE
;; negate
L346E:  CALL    L34E9           ; call routine TEST-ZERO and
        RET     C               ; return if so leaving zero unchanged.

        LD      B,$00           ; signal negate required before joining
                                ; common code.

;; NEG-TEST
L3474:  LD      A,(HL)          ; load first byte and
        AND     A               ; test for zero
        JR      Z,L3483         ; forward to INT-CASE if a small integer

; for floating point numbers a single bit denotes the sign.

        INC     HL              ; address the first byte of mantissa.
        LD      A,B             ; action flag $FF=abs, $00=neg.
        AND     $80             ; now         $80      $00
        OR      (HL)            ; sets bit 7 for abs
        RLA                     ; sets carry for abs and if number negative
        CCF                     ; complement carry flag
        RRA                     ; and rotate back in altering sign
        LD      (HL),A          ; put the altered adjusted number back
        DEC     HL              ; HL points to result
        RET                     ; return with DE unchanged

; ---

; for integer numbers an entire byte denotes the sign.

;; INT-CASE
L3483:  PUSH    DE              ; save STKEND.

        PUSH    HL              ; save pointer to the last value/result.

        CALL    L2D7F           ; routine INT-FETCH puts integer in DE
                                ; and the sign in C.

        POP     HL              ; restore the result pointer.

        LD      A,B             ; $FF=abs, $00=neg
        OR      C               ; $FF for abs, no change neg
        CPL                     ; $00 for abs, switched for neg
        LD      C,A             ; transfer result to sign byte.

        CALL    L2D8E           ; routine INT-STORE to re-write the integer.

        POP     DE              ; restore STKEND.
        RET                     ; return.

; ---------------------
; THE 'SIGNUM' FUNCTION
; ---------------------
```

```
; (offset: $29 'sgn')
;   This routine replaces the last value on the calculator stack,
;   which may be in floating point or integer form, with the integer values
;   zero if zero, with one if positive and  with -minus one if negative.

;; sgn
L3492:  CALL    L34E9           ; call routine TEST-ZERO and
        RET     C               ; exit if so as no change is required.

        PUSH    DE              ; save pointer to STKEND.

        LD      DE,$0001        ; the result will be 1.
        INC     HL              ; skip over the exponent.
        RL      (HL)            ; rotate the sign bit into the carry flag.
        DEC     HL              ; step back to point to the result.
        SBC     A,A             ; byte will be $FF if negative, $00 if positive.
        LD      C,A             ; store the sign byte in the C register.
        CALL    L2D8E           ; routine INT-STORE to overwrite the last
                                ; value with 0001 and sign.

        POP     DE              ; restore STKEND.
        RET                     ; return.

; -----------------
; THE 'IN' FUNCTION
; -----------------
; (offset: $2C 'in')
;   This function reads a byte from an input port.

;; in
L34A5:  CALL    L1E99           ; Routine FIND-INT2 puts port address in BC.
                                ; All 16 bits are put on the address line.

        IN      A,(C)           ; Read the port.

        JR      L34B0           ; exit to STACK-A (via IN-PK-STK to save a byte
                                ; of instruction code).

; -------------------
; THE 'PEEK' FUNCTION
; -------------------
; (offset: $2B 'peek')
;   This function returns the contents of a memory address.
;   The entire address space can be peeked including the ROM.

;; peek
L34AC:  CALL    L1E99           ; routine FIND-INT2 puts address in BC.
        LD      A,(BC)          ; load contents into A register.

;; IN-PK-STK
L34B0:  JP      L2D28           ; exit via STACK-A to put the value on the
                                ; calculator stack.

; ------------------
; THE 'USR' FUNCTION
; ------------------
; (offset: $2d 'usr-no')
;   The USR function followed by a number 0-65535 is the method by which
;   the Spectrum invokes machine code programs. This function returns the
;   contents of the BC register pair.
;   Note. that STACK-BC re-initializes the IY register if a user-written
;   program has altered it.

;; usr-no
```

```
L34B3:  CALL    L1E99           ; routine FIND-INT2 to fetch the
                                ; supplied address into BC.

        LD      HL,L2D2B        ; address: STACK-BC is
        PUSH    HL              ; pushed onto the machine stack.
        PUSH    BC              ; then the address of the machine code
                                ; routine.

        RET                     ; make an indirect jump to the routine
                                ; and, hopefully, to STACK-BC also.

; -------------------------
; THE 'USR STRING' FUNCTION
; -------------------------
; (offset: $19 'usr-$')
;   The user function with a one-character string argument, calculates the
;   address of the User Defined Graphic character that is in the string.
;   As an alternative, the ASCII equivalent, upper or lower case,
;   may be supplied. This provides a user-friendly method of redefining
;   the 21 User Definable Graphics e.g.
;   POKE USR "a", BIN 10000000 will put a dot in the top left corner of the
;   character 144.
;   Note. the curious double check on the range. With 26 UDGs the first check
;   only is necessary. With anything less the second check only is required.
;   It is highly likely that the first check was written by Steven Vickers.

;; usr-$
L34BC:  CALL    L2BF1           ; routine STK-FETCH fetches the string
                                ; parameters.
        DEC     BC              ; decrease BC by
        LD      A,B             ; one to test
        OR      C               ; the length.
        JR      NZ,L34E7        ; to REPORT-A if not a single character.

        LD      A,(DE)          ; fetch the character
        CALL    L2C8D           ; routine ALPHA sets carry if 'A-Z' or 'a-z'.
        JR      C,L34D3         ; forward to USR-RANGE if ASCII.

        SUB     $90             ; make UDGs range 0-20d
        JR      C,L34E7         ; to REPORT-A if too low. e.g. usr " ".

        CP      $15             ; Note. this test is not necessary.
        JR      NC,L34E7        ; to REPORT-A if higher than 20.

        INC     A               ; make range 1-21d to match LSBs of ASCII

;; USR-RANGE
L34D3:  DEC     A               ; make range of bits 0-4 start at zero
        ADD     A,A             ; multiply by eight
        ADD     A,A             ; and lose any set bits
        ADD     A,A             ; range now 0 - 25*8
        CP      $A8             ; compare to 21*8
        JR      NC,L34E7        ; to REPORT-A if originally higher
                                ; than 'U','u' or graphics U.

        LD      BC,($5C7B)      ; fetch the UDG system variable value.
        ADD     A,C             ; add the offset to character
        LD      C,A             ; and store back in register C.
        JR      NC,L34E4        ; forward to USR-STACK if no overflow.

        INC     B               ; increment high byte.

;; USR-STACK
L34E4:  JP      L2D2B           ; jump back and exit via STACK-BC to store
```

```
; ---

;; REPORT-A
L34E7:  RST     08H             ; ERROR-1
        DEFB    $09             ; Error Report: Invalid argument

; -----------------------------
; THE 'TEST FOR ZERO' SUBROUTINE
; -----------------------------
;   Test if top value on calculator stack is zero.  The carry flag is set if
;   the last value is zero but no registers are altered.
;   All five bytes will be zero but first four only need be tested.
;   On entry, HL points to the exponent the first byte of the value.

;; TEST-ZERO
L34E9:  PUSH    HL              ; preserve HL which is used to address.
        PUSH    BC              ; preserve BC which is used as a store.
        LD      B,A             ; preserve A in B.

        LD      A,(HL)          ; load first byte to accumulator
        INC     HL              ; advance.
        OR      (HL)            ; OR with second byte and clear carry.
        INC     HL              ; advance.
        OR      (HL)            ; OR with third byte.
        INC     HL              ; advance.
        OR      (HL)            ; OR with fourth byte.

        LD      A,B             ; restore A without affecting flags.
        POP     BC              ; restore the saved
        POP     HL              ; registers.

        RET     NZ              ; return if not zero and with carry reset.

        SCF                     ; set the carry flag.
        RET                     ; return with carry set if zero.

; --------------------------------
; THE 'GREATER THAN ZERO' OPERATOR
; --------------------------------
; (offset: $37 'greater-0' )
;   Test if the last value on the calculator stack is greater than zero.
;   This routine is also called directly from the end-tests of the comparison
;   routine.

;; GREATER-0
;; greater-0
L34F9:  CALL    L34E9           ; routine TEST-ZERO
        RET     C               ; return if was zero as this
                                ; is also the Boolean 'false' value.

        LD      A,$FF           ; prepare XOR mask for sign bit
        JR      L3507           ; forward to SIGN-TO-C
                                ; to put sign in carry
                                ; (carry will become set if sign is positive)
                                ; and then overwrite location with 1 or 0
                                ; as appropriate.

; ------------------
; THE 'NOT' FUNCTION
; ------------------
; (offset: $30 'not')
;   This overwrites the last value with 1 if it was zero else with zero
;   if it was any other value.
```

```
;
;   e.g. NOT 0 returns 1, NOT 1 returns 0, NOT -3 returns 0.
;
;   The subroutine is also called directly from the end-tests of the comparison
;   operator.

;; NOT
;; not
L3501:  CALL    L34E9           ; routine TEST-ZERO sets carry if zero

        JR      L350B           ; to FP-0/1 to overwrite operand with
                                ; 1 if carry is set else to overwrite with zero.

; ------------------------------
; THE 'LESS THAN ZERO' OPERATION
; ------------------------------
; (offset: $36 'less-0' )
;   Destructively test if last value on calculator stack is less than zero.
;   Bit 7 of second byte will be set if so.

;; less-0
L3506:  XOR     A               ; set XOR mask to zero
                                ; (carry will become set if sign is negative).

;   transfer sign of mantissa to Carry Flag.

;; SIGN-TO-C
L3507:  INC     HL              ; address 2nd byte.
        XOR     (HL)            ; bit 7 of HL will be set if number is negative.
        DEC     HL              ; address 1st byte again.
        RLCA                    ; rotate bit 7 of A to carry.

; ----------------------------
; THE 'ZERO OR ONE' SUBROUTINE
; ----------------------------
;   This routine places an integer value of zero or one at the addressed
;   location of the calculator stack or MEM area.  The value one is written if
;   carry is set on entry else zero.

;; FP-0/1
L350B:  PUSH    HL              ; save pointer to the first byte
        LD      A,$00           ; load accumulator with zero - without
                                ; disturbing flags.
        LD      (HL),A          ; zero to first byte
        INC     HL              ; address next
        LD      (HL),A          ; zero to 2nd byte
        INC     HL              ; address low byte of integer
        RLA                     ; carry to bit 0 of A
        LD      (HL),A          ; load one or zero to low byte.
        RRA                     ; restore zero to accumulator.
        INC     HL              ; address high byte of integer.
        LD      (HL),A          ; put a zero there.
        INC     HL              ; address fifth byte.
        LD      (HL),A          ; put a zero there.
        POP     HL              ; restore pointer to the first byte.
        RET                     ; return.

; -----------------
; THE 'OR' OPERATOR
; -----------------
; (offset: $07 'or' )
; The Boolean OR operator. e.g. X OR Y
; The result is zero if both values are zero else a non-zero value.
;
```

```
; e.g.    0 OR 0  returns 0.
;        -3 OR 0  returns -3.
;         0 OR -3 returns 1.
;        -3 OR 2  returns 1.
;
; A binary operation.
; On entry HL points to first operand (X) and DE to second operand (Y).

;; or
L351B:  EX      DE,HL           ; make HL point to second number
        CALL    L34E9           ; routine TEST-ZERO
        EX      DE,HL           ; restore pointers
        RET     C               ; return if result was zero - first operand,
                                ; now the last value, is the result.

        SCF                     ; set carry flag
        JR      L350B           ; back to FP-0/1 to overwrite the first operand
                                ; with the value 1.


; ---------------------------------
; THE 'NUMBER AND NUMBER' OPERATION
; ---------------------------------
; (offset: $08 'no-&-no')
;   The Boolean AND operator.
;
;   e.g.    -3 AND 2  returns -3.
;           -3 AND 0  returns 0.
;            0 and -2 returns 0.
;            0 and 0  returns 0.
;
;   Compare with OR routine above.

;; no-&-no
L3524:  EX      DE,HL           ; make HL address second operand.

        CALL    L34E9           ; routine TEST-ZERO sets carry if zero.

        EX      DE,HL           ; restore pointers.
        RET     NC              ; return if second non-zero, first is result.

;

        AND     A               ; else clear carry.
        JR      L350B           ; back to FP-0/1 to overwrite first operand
                                ; with zero for return value.

; ---------------------------------
; THE 'STRING AND NUMBER' OPERATION
; ---------------------------------
; (offset: $10 'str-&-no')
;   e.g. "You Win" AND score>99 will return the string if condition is true
;   or the null string if false.

;; str-&-no
L352D:  EX      DE,HL           ; make HL point to the number.
        CALL    L34E9           ; routine TEST-ZERO.
        EX      DE,HL           ; restore pointers.
        RET     NC              ; return if number was not zero - the string
                                ; is the result.

;   if the number was zero (false) then the null string must be returned by
;   altering the length of the string on the calculator stack to zero.
```

```
        PUSH    DE              ; save pointer to the now obsolete number
                                ; (which will become the new STKEND)

        DEC     DE              ; point to the 5th byte of string descriptor.
        XOR     A               ; clear the accumulator.
        LD      (DE),A          ; place zero in high byte of length.
        DEC     DE              ; address low byte of length.
        LD      (DE),A          ; place zero there - now the null string.

        POP     DE              ; restore pointer - new STKEND.
        RET                     ; return.

; ---------------------------
; THE 'COMPARISON' OPERATIONS
; ---------------------------
; (offset: $0A 'no-gr-eql')
; (offset: $0B 'nos-neql')
; (offset: $0C 'no-grtr')
; (offset: $0D 'no-less')
; (offset: $0E 'nos-eql')
; (offset: $11 'str-l-eql')
; (offset: $12 'str-gr-eql')
; (offset: $13 'strs-neql')
; (offset: $14 'str-grtr')
; (offset: $15 'str-less')
; (offset: $16 'strs-eql')

;   True binary operations.
;   A single entry point is used to evaluate six numeric and six string
;   comparisons. On entry, the calculator literal is in the B register and
;   the two numeric values, or the two string parameters, are on the
;   calculator stack.
;   The individual bits of the literal are manipulated to group similar
;   operations although the SUB 8 instruction does nothing useful and merely
;   alters the string test bit.
;   Numbers are compared by subtracting one from the other, strings are
;   compared by comparing every character until a mismatch, or the end of one
;   or both, is reached.
;
;   Numeric Comparisons.
;   --------------------
;   The 'x>y' example is the easiest as it employs straight-thru logic.
;   Number y is subtracted from x and the result tested for greater-0 yielding
;   a final value 1 (true) or 0 (false).
;   For 'x<y' the same logic is used but the two values are first swapped on the
;   calculator stack.
;   For 'x=y' NOT is applied to the subtraction result yielding true if the
;   difference was zero and false with anything else.
;   The first three numeric comparisons are just the opposite of the last three
;   so the same processing steps are used and then a final NOT is applied.
;
; literal    Test   No  sub 8       ExOrNot  1st RRCA  exch sub  ?  End-Tests
; =========  ====   == ======== === ======== ========  ==== ===  =  === === ===
; no-l-eql   x<=y   09 00000001 dec 00000000 00000000  ---- x-y  ?  --- >0? NOT
; no-gr-eql  x>=y   0A 00000010 dec 00000001 10000000c swap y-x  ?  --- >0? NOT
; nos-neql   x<>y   0B 00000011 dec 00000010 00000001  ---- x-y  ?  NOT --- NOT
; no-grtr    x>y    0C 00000100  -  00000100 00000010  ---- x-y  ?  --- >0? ---
; no-less    x<y    0D 00000101  -  00000101 10000010c swap y-x  ?  --- >0? ---
; nos-eql    x=y    0E 00000110  -  00000110 00000011  ---- x-y  ?  NOT --- ---
;
;                                                           comp -> C/F
;                                                           ====    ===
; str-l-eql  x$<=y$ 11 00001001 dec 00001000 00000100  ---- x$y$ 0  !or >0? NOT
; str-gr-eql x$>=y$ 12 00001010 dec 00001001 10000100c swap y$x$ 0  !or >0? NOT
```

```
; strs-neql  x$<>y$ 13 00001011 dec 00001010 00000101  ---- x$y$ 0  !or >0? NOT
; str-grtr   x$>y$  14 00001100  -  00001100 00000110  ---- x$y$ 0  !or >0? ---
; str-less   x$<y$  15 00001101  -  00001101 10000110c swap y$x$ 0  !or >0? ---
; strs-eql   x$=y$  16 00001110  -  00001110 00000111  ---- x$y$ 0  !or >0? ---
;
;   String comparisons are a little different in that the eql/neql carry flag
;   from the 2nd RRCA is, as before, fed into the first of the end tests but
;   along the way it gets modified by the comparison process. The result on the
;   stack always starts off as zero and the carry fed in determines if NOT is
;   applied to it. So the only time the greater-0 test is applied is if the
;   stack holds zero which is not very efficient as the test will always yield
;   zero. The most likely explanation is that there were once separate end tests
;   for numbers and strings.

;; no-l-eql,etc.
L353B:  LD      A,B                ; transfer literal to accumulator.
        SUB     $08                ; subtract eight - which is not useful.

        BIT     2,A                ; isolate '>', '<', '='.

        JR      NZ,L3543           ; skip to EX-OR-NOT with these.

        DEC     A                  ; else make $00-$02, $08-$0A to match bits 0-2.

;; EX-OR-NOT
L3543:  RRCA                       ; the first RRCA sets carry for a swap.
        JR      NC,L354E           ; forward to NU-OR-STR with other 8 cases

; for the other 4 cases the two values on the calculator stack are exchanged.

        PUSH    AF                 ; save A and carry.
        PUSH    HL                 ; save HL - pointer to first operand.
                                   ; (DE points to second operand).

        CALL    L343C              ; routine exchange swaps the two values.
                                   ; (HL = second operand, DE = STKEND)

        POP     DE                 ; DE = first operand
        EX      DE,HL              ; as we were.
        POP     AF                 ; restore A and carry.

; Note. it would be better if the 2nd RRCA preceded the string test.
; It would save two duplicate bytes and if we also got rid of that sub 8
; at the beginning we wouldn't have to alter which bit we test.

;; NU-OR-STR
L354E:  BIT     2,A                ; test if a string comparison.
        JR      NZ,L3559           ; forward to STRINGS if so.

; continue with numeric comparisons.

        RRCA                       ; 2nd RRCA causes eql/neql to set carry.
        PUSH    AF                 ; save A and carry

        CALL    L300F              ; routine subtract leaves result on stack.
        JR      L358C              ; forward to END-TESTS

; ---

;; STRINGS
L3559:  RRCA                       ; 2nd RRCA causes eql/neql to set carry.
        PUSH    AF                 ; save A and carry.

        CALL    L2BF1              ; routine STK-FETCH gets 2nd string params
```

```
        PUSH    DE                ; save start2 *.
        PUSH    BC                ; and the length.

        CALL    L2BF1             ; routine STK-FETCH gets 1st string
                                  ; parameters - start in DE, length in BC.
        POP     HL                ; restore length of second to HL.

; A loop is now entered to compare, by subtraction, each corresponding character
; of the strings. For each successful match, the pointers are incremented and
; the lengths decreased and the branch taken back to here. If both string
; remainders become null at the same time, then an exact match exists.

;; BYTE-COMP
L3564:  LD      A,H               ; test if the second string
        OR      L                 ; is the null string and hold flags.

        EX      (SP),HL           ; put length2 on stack, bring start2 to HL *.
        LD      A,B               ; hi byte of length1 to A

        JR      NZ,L3575          ; forward to SEC-PLUS if second not null.

        OR      C                 ; test length of first string.

;; SECND-LOW
L356B:  POP     BC                ; pop the second length off stack.
        JR      Z,L3572           ; forward to BOTH-NULL if first string is also
                                  ; of zero length.

; the true condition - first is longer than second (SECND-LESS)

        POP     AF                ; restore carry (set if eql/neql)
        CCF                       ; complement carry flag.
                                  ; Note. equality becomes false.
                                  ; Inequality is true. By swapping or applying
                                  ; a terminal 'not', all comparisons have been
                                  ; manipulated so that this is success path.
        JR      L3588             ; forward to leave via STR-TEST

; ---
; the branch was here with a match

;; BOTH-NULL
L3572:  POP     AF                ; restore carry - set for eql/neql
        JR      L3588             ; forward to STR-TEST

; ---
; the branch was here when 2nd string not null and low byte of first is yet
; to be tested.


;; SEC-PLUS
L3575:  OR      C                 ; test the length of first string.
        JR      Z,L3585           ; forward to FRST-LESS if length is zero.

; both strings have at least one character left.

        LD      A,(DE)            ; fetch character of first string.
        SUB     (HL)              ; subtract with that of 2nd string.
        JR      C,L3585           ; forward to FRST-LESS if carry set

        JR      NZ,L356B          ; back to SECND-LOW and then STR-TEST
                                  ; if not exact match.

        DEC     BC                ; decrease length of 1st string.
```

```
        INC     DE                  ; increment 1st string pointer.

        INC     HL                  ; increment 2nd string pointer.
        EX      (SP),HL             ; swap with length on stack
        DEC     HL                  ; decrement 2nd string length
        JR      L3564               ; back to BYTE-COMP

; ---
; the false condition.

;; FRST-LESS
L3585:  POP     BC                  ; discard length
        POP     AF                  ; pop A
        AND     A                   ; clear the carry for false result.

; ---
; exact match and x$>y$ rejoin here

;; STR-TEST
L3588:  PUSH    AF                  ; save A and carry

        RST     28H                 ;; FP-CALC
        DEFB    $A0                 ;;stk-zero      an initial false value.
        DEFB    $38                 ;;end-calc

; both numeric and string paths converge here.

;; END-TESTS
L358C:  POP     AF                  ; pop carry  - will be set if eql/neql
        PUSH    AF                  ; save it again.

        CALL    C,L3501             ; routine NOT sets true(1) if equal(0)
                                    ; or, for strings, applies true result.

        POP     AF                  ; pop carry and
        PUSH    AF                  ; save A

        CALL    NC,L34F9            ; routine GREATER-0 tests numeric subtraction
                                    ; result but also needlessly tests the string
                                    ; value for zero - it must be.

        POP     AF                  ; pop A
        RRCA                        ; the third RRCA - test for '<=', '>=' or '<>'.
        CALL    NC,L3501            ; apply a terminal NOT if so.
        RET                         ; return.

; ------------------------------------
; THE 'STRING CONCATENATION' OPERATION
; ------------------------------------
; (offset: $17 'strs-add')
;   This literal combines two strings into one e.g. LET a$ = b$ + c$
;   The two parameters of the two strings to be combined are on the stack.

;; strs-add
L359C:  CALL    L2BF1               ; routine STK-FETCH fetches string parameters
                                    ; and deletes calculator stack entry.
        PUSH    DE                  ; save start address.
        PUSH    BC                  ; and length.

        CALL    L2BF1               ; routine STK-FETCH for first string
        POP     HL                  ; re-fetch first length
        PUSH    HL                  ; and save again
        PUSH    DE                  ; save start of second string
        PUSH    BC                  ; and its length.
```

```
        ADD     HL,BC           ; add the two lengths.
        LD      B,H             ; transfer to BC
        LD      C,L             ; and create
        RST     30H             ; BC-SPACES in workspace.
                                ; DE points to start of space.

        CALL    L2AB2           ; routine STK-STO-$ stores parameters
                                ; of new string updating STKEND.

        POP     BC              ; length of first
        POP     HL              ; address of start
        LD      A,B             ; test for
        OR      C               ; zero length.
        JR      Z,L35B7         ; to OTHER-STR if null string

        LDIR                    ; copy string to workspace.

;; OTHER-STR
L35B7:  POP     BC              ; now second length
        POP     HL              ; and start of string
        LD      A,B             ; test this one
        OR      C               ; for zero length
        JR      Z,L35BF         ; skip forward to STK-PNTRS if so as complete.

        LDIR                    ; else copy the bytes.
                                ; and continue into next routine which
                                ; sets the calculator stack pointers.

; ----------------------------------
; THE 'SET STACK POINTERS' SUBROUTINE
; ----------------------------------
;   Register DE is set to STKEND and HL, the result pointer, is set to five
;   locations below this.
;   This routine is used when it is inconvenient to save these values at the
;   time the calculator stack is manipulated due to other activity on the
;   machine stack.
;   This routine is also used to terminate the VAL and READ-IN  routines for
;   the same reason and to initialize the calculator stack at the start of
;   the CALCULATE routine.

;; STK-PNTRS
L35BF:  LD      HL,($5C65)      ; fetch STKEND value from system variable.
        LD      DE,$FFFB        ; the value -5
        PUSH    HL              ; push STKEND value.

        ADD     HL,DE           ; subtract 5 from HL.

        POP     DE              ; pop STKEND to DE.
        RET                     ; return.

; -------------------
; THE 'CHR$' FUNCTION
; -------------------
; (offset: $2f 'chr$')
;   This function returns a single character string that is a result of
;   converting a number in the range 0-255 to a string e.g. CHR$ 65 = "A".

;; chrs
L35C9:  CALL    L2DD5           ; routine FP-TO-A puts the number in A.

        JR      C,L35DC         ; forward to REPORT-Bd if overflow
        JR      NZ,L35DC        ; forward to REPORT-Bd if negative
```

```
        PUSH    AF                ; save the argument.

        LD      BC,$0001          ; one space required.
        RST     30H               ; BC-SPACES makes DE point to start

        POP     AF                ; restore the number.

        LD      (DE),A            ; and store in workspace

        CALL    L2AB2             ; routine STK-STO-$ stacks descriptor.

        EX      DE,HL             ; make HL point to result and DE to STKEND.
        RET                       ; return.

; ---

;; REPORT-Bd
L35DC:  RST     08H               ; ERROR-1
        DEFB    $0A               ; Error Report: Integer out of range

; ----------------------------
; THE 'VAL and VAL$' FUNCTIONS
; ----------------------------
; (offset: $1d 'val')
; (offset: $18 'val$')
;   VAL treats the characters in a string as a numeric expression.
;   e.g. VAL "2.3" = 2.3, VAL "2+4" = 6, VAL ("2" + "4") = 24.
;   VAL$ treats the characters in a string as a string expression.
;   e.g. VAL$ (z$+"(2)") = a$(2) if z$ happens to be "a$".

;; val
;; val$
L35DE:  LD      HL,($5C5D)        ; fetch value of system variable CH_ADD
        PUSH    HL                ; and save on the machine stack.
        LD      A,B               ; fetch the literal (either $1D or $18).
        ADD     A,$E3             ; add $E3 to form $00 (setting carry) or $FB.
        SBC     A,A               ; now form $FF bit 6 = numeric result
                                  ; or $00 bit 6 = string result.
        PUSH    AF                ; save this mask on the stack

        CALL    L2BF1             ; routine STK-FETCH fetches the string operand
                                  ; from calculator stack.

        PUSH    DE                ; save the address of the start of the string.
        INC     BC                ; increment the length for a carriage return.

        RST     30H               ; BC-SPACES creates the space in workspace.
        POP     HL                ; restore start of string to HL.
        LD      ($5C5D),DE        ; load CH_ADD with start DE in workspace.

        PUSH    DE                ; save the start in workspace
        LDIR                      ; copy string from program or variables or
                                  ; workspace to the workspace area.
        EX      DE,HL             ; end of string + 1 to HL
        DEC     HL                ; decrement HL to point to end of new area.
        LD      (HL),$0D          ; insert a carriage return at end.
        RES     7,(IY+$01)        ; update FLAGS  - signal checking syntax.
        CALL    L24FB             ; routine SCANNING evaluates string
                                  ; expression and result.

        RST     18H               ; GET-CHAR fetches next character.
        CP      $0D               ; is it the expected carriage return ?
        JR      NZ,L360C          ; forward to V-RPORT-C if not
                                  ; 'Nonsense in BASIC'.
```

```
        POP     HL              ; restore start of string in workspace.
        POP     AF              ; restore expected result flag (bit 6).
        XOR     (IY+$01)        ; xor with FLAGS now updated by SCANNING.
        AND     $40             ; test bit 6 - should be zero if result types
                                ; match.

;; V-RPORT-C
L360C:  JP      NZ,L1C8A        ; jump back to REPORT-C with a result mismatch.

        LD      ($5C5D),HL      ; set CH_ADD to the start of the string again.
        SET     7,(IY+$01)      ; update FLAGS  - signal running program.
        CALL    L24FB           ; routine SCANNING evaluates the string
                                ; in full leaving result on calculator stack.

        POP     HL              ; restore saved character address in program.
        LD      ($5C5D),HL      ; and reset the system variable CH_ADD.

        JR      L35BF           ; back to exit via STK-PNTRS.
                                ; resetting the calculator stack pointers
                                ; HL and DE from STKEND as it wasn't possible
                                ; to preserve them during this routine.

; -------------------
; THE 'STR$' FUNCTION
; -------------------
; (offset: $2e 'str$')
;   This function produces a string comprising the characters that would appear
;   if the numeric argument were printed.
;   e.g. STR$ (1/10) produces "0.1".

;; str$
L361F:  LD      BC,$0001        ; create an initial byte in workspace
        RST     30H             ; using BC-SPACES restart.

        LD      ($5C5B),HL      ; set system variable K_CUR to new location.
        PUSH    HL              ; and save start on machine stack also.

        LD      HL,($5C51)      ; fetch value of system variable CURCHL
        PUSH    HL              ; and save that too.

        LD      A,$FF           ; select system channel 'R'.
        CALL    L1601           ; routine CHAN-OPEN opens it.
        CALL    L2DE3           ; routine PRINT-FP outputs the number to
                                ; workspace updating K-CUR.

        POP     HL              ; restore current channel.
        CALL    L1615           ; routine CHAN-FLAG resets flags.

        POP     DE              ; fetch saved start of string to DE.
        LD      HL,($5C5B)      ; load HL with end of string from K_CUR.

        AND     A               ; prepare for true subtraction.
        SBC     HL,DE           ; subtract start from end to give length.
        LD      B,H             ; transfer the length to
        LD      C,L             ; the BC register pair.

        CALL    L2AB2           ; routine STK-STO-$ stores string parameters
                                ; on the calculator stack.

        EX      DE,HL           ; HL = last value, DE = STKEND.
        RET                     ; return.

; ------------------------
```

```
; THE 'READ-IN' SUBROUTINE
; ------------------------
; (offset: $1a 'read-in')
;   This is the calculator literal used by the INKEY$ function when a '#'
;   is encountered after the keyword.
;   INKEY$ # does not interact correctly with the keyboard, #0 or #1, and
;   its uses are for other channels.

;; read-in
L3645:  CALL    L1E94           ; routine FIND-INT1 fetches stream to A
        CP      $10             ; compare with 16 decimal.
        JP      NC,L1E9F        ; JUMP to REPORT-Bb if not in range 0 - 15.
                                ; 'Integer out of range'
                                ; (REPORT-Bd is within range)

        LD      HL,($5C51)      ; fetch current channel CURCHL
        PUSH    HL              ; save it

        CALL    L1601           ; routine CHAN-OPEN opens channel

        CALL    L15E6           ; routine INPUT-AD - the channel must have an
                                ; input stream or else error here from stream
                                ; stub.
        LD      BC,$0000        ; initialize length of string to zero
        JR      NC,L365F        ; forward to R-I-STORE if no key detected.

        INC     C               ; increase length to one.

        RST     30H             ; BC-SPACES creates space for one character
                                ; in workspace.
        LD      (DE),A          ; the character is inserted.

;; R-I-STORE
L365F:  CALL    L2AB2           ; routine STK-STO-$ stacks the string
                                ; parameters.
        POP     HL              ; restore current channel address

        CALL    L1615           ; routine CHAN-FLAG resets current channel
                                ; system variable and flags.

        JP      L35BF           ; jump back to STK-PNTRS

; -------------------
; THE 'CODE' FUNCTION
; -------------------
; (offset: $1c 'code')
;   Returns the ASCII code of a character or first character of a string
;   e.g. CODE "Aardvark" = 65, CODE "" = 0.

;; code
L3669:  CALL    L2BF1           ; routine STK-FETCH to fetch and delete the
                                ; string parameters.
                                ; DE points to the start, BC holds the length.

        LD      A,B             ; test length
        OR      C               ; of the string.
        JR      Z,L3671         ; skip to STK-CODE with zero if the null string.

        LD      A,(DE)          ; else fetch the first character.

;; STK-CODE
L3671:  JP      L2D28           ; jump back to STACK-A (with memory check)

; ------------------
```

```
; THE 'LEN' FUNCTION
; ------------------
; (offset: $1e 'len')
;   Returns the length of a string.
;   In Sinclair BASIC strings can be more than twenty thousand characters long
;   so a sixteen-bit register is required to store the length

;; len
L3674:  CALL    L2BF1           ; Routine STK-FETCH to fetch and delete the
                                ; string parameters from the calculator stack.
                                ; Register BC now holds the length of string.

        JP      L2D2B           ; Jump back to STACK-BC to save result on the
                                ; calculator stack (with memory check).

; ------------------------------------
; THE 'DECREASE THE COUNTER' SUBROUTINE
; ------------------------------------
; (offset: $35 'dec-jr-nz')
;   The calculator has an instruction that decrements a single-byte
;   pseudo-register and makes consequential relative jumps just like
;   the Z80's DJNZ instruction.

;; dec-jr-nz
L367A:  EXX                     ; switch in set that addresses code

        PUSH    HL              ; save pointer to offset byte
        LD      HL,$5C67        ; address BREG in system variables
        DEC     (HL)            ; decrement it
        POP     HL              ; restore pointer

        JR      NZ,L3687        ; to JUMP-2 if not zero

        INC     HL              ; step past the jump length.
        EXX                     ; switch in the main set.
        RET                     ; return.

; Note. as a general rule the calculator avoids using the IY register
; otherwise the cumbersome 4 instructions in the middle could be replaced by
; dec (iy+$2d) - three bytes instead of six.


; ---------------------
; THE 'JUMP' SUBROUTINE
; ---------------------
; (offset: $33 'jump')
;   This enables the calculator to perform relative jumps just like the Z80
;   chip's JR instruction.

;; jump
;; JUMP
L3686:  EXX                     ; switch in pointer set

;; JUMP-2
L3687:  LD      E,(HL)          ; the jump byte 0-127 forward, 128-255 back.
        LD      A,E             ; transfer to accumulator.
        RLA                     ; if backward jump, carry is set.
        SBC     A,A             ; will be $FF if backward or $00 if forward.
        LD      D,A             ; transfer to high byte.
        ADD     HL,DE           ; advance calculator pointer forward or back.

        EXX                     ; switch back.
        RET                     ; return.
```

```
; -------------------------
; THE 'JUMP-TRUE' SUBROUTINE
; -------------------------
; (offset: $00 'jump-true')
;   This enables the calculator to perform conditional relative jumps dependent
;   on whether the last test gave a true result.

;; jump-true
L368F:  INC     DE              ; Collect the
        INC     DE              ; third byte
        LD      A,(DE)          ; of the test
        DEC     DE              ; result and
        DEC     DE              ; backtrack.

        AND     A               ; Is result 0 or 1 ?
        JR      NZ,L3686        ; Back to JUMP if true (1).

        EXX                     ; Else switch in the pointer set.
        INC     HL              ; Step past the jump length.
        EXX                     ; Switch in the main set.
        RET                     ; Return.

; ------------------------
; THE 'END-CALC' SUBROUTINE
; ------------------------
; (offset: $38 'end-calc')
;   The end-calc literal terminates a mini-program written in the Spectrum's
;   internal language.

;; end-calc
L369B:  POP     AF              ; Drop the calculator return address RE-ENTRY
        EXX                     ; Switch to the other set.

        EX      (SP),HL         ; Transfer H'L' to machine stack for the
                                ; return address.
                                ; When exiting recursion, then the previous
                                ; pointer is transferred to H'L'.

        EXX                     ; Switch back to main set.
        RET                     ; Return.


; -----------------------
; THE 'MODULUS' SUBROUTINE
; -----------------------
; (offset: $32 'n-mod-m')
; (n1,n2 -- r,q)
;   Similar to FORTH's 'divide mod' /MOD
;   On the Spectrum, this is only used internally by the RND function and could
;   have been implemented inline.  On the ZX81, this calculator routine was also
;   used by PRINT-FP.

;; n-mod-m
L36A0:  RST     28H             ;; FP-CALC          17, 3.
        DEFB    $C0             ;;st-mem-0          17, 3.
        DEFB    $02             ;;delete            17.
        DEFB    $31             ;;duplicate         17, 17.
        DEFB    $E0             ;;get-mem-0         17, 17, 3.
        DEFB    $05             ;;division          17, 17/3.
        DEFB    $27             ;;int               17, 5.
        DEFB    $E0             ;;get-mem-0         17, 5, 3.
        DEFB    $01             ;;exchange          17, 3, 5.
        DEFB    $C0             ;;st-mem-0          17, 3, 5.
        DEFB    $04             ;;multiply          17, 15.
```

```
        DEFB    $03             ;;subtract          2.
        DEFB    $E0             ;;get-mem-0         2, 5.
        DEFB    $38             ;;end-calc          2, 5.

        RET                     ; return.


; ------------------
; THE 'INT' FUNCTION
; ------------------
; (offset $27: 'int' )
; This function returns the integer of x, which is just the same as truncate
; for positive numbers. The truncate literal truncates negative numbers
; upwards so that -3.4 gives -3 whereas the BASIC INT function has to
; truncate negative numbers down so that INT -3.4 is -4.
; It is best to work through using, say, +-3.4 as examples.

;; int
L36AF:  RST     28H             ;; FP-CALC              x.    (= 3.4 or -3.4).
        DEFB    $31             ;;duplicate             x, x.
        DEFB    $36             ;;less-0                x, (1/0)
        DEFB    $00             ;;jump-true             x, (1/0)
        DEFB    $04             ;;to L36B7, X-NEG

        DEFB    $3A             ;;truncate              trunc 3.4 = 3.
        DEFB    $38             ;;end-calc              3.

        RET                     ; return with + int x on stack.

; ---


;; X-NEG
L36B7:  DEFB    $31             ;;duplicate             -3.4, -3.4.
        DEFB    $3A             ;;truncate              -3.4, -3.
        DEFB    $C0             ;;st-mem-0              -3.4, -3.
        DEFB    $03             ;;subtract              -.4
        DEFB    $E0             ;;get-mem-0             -.4, -3.
        DEFB    $01             ;;exchange              -3, -.4.
        DEFB    $30             ;;not                   -3, (0).
        DEFB    $00             ;;jump-true             -3.
        DEFB    $03             ;;to L36C2, EXIT        -3.

        DEFB    $A1             ;;stk-one               -3, 1.
        DEFB    $03             ;;subtract              -4.

;; EXIT
L36C2:  DEFB    $38             ;;end-calc              -4.

        RET                     ; return.


; ------------------
; THE 'EXP' FUNCTION
; ------------------
; (offset $26: 'exp')
;   The exponential function EXP x is equal to e^x, where e is the mathematical
;   name for a number approximated to 2.718281828.
;   ERROR 6 if argument is more than about 88.

;; EXP
;; exp
L36C4:  RST     28H             ;; FP-CALC
        DEFB    $3D             ;;re-stack
```

```
        DEFB    $34                 ;;stk-data
        DEFB    $F1                 ;;Exponent: $81, Bytes: 4
        DEFB    $38,$AA,$3B,$29 ;;
        DEFB    $04                 ;;multiply
        DEFB    $31                 ;;duplicate
        DEFB    $27                 ;;int
        DEFB    $C3                 ;;st-mem-3
        DEFB    $03                 ;;subtract
        DEFB    $31                 ;;duplicate
        DEFB    $0F                 ;;addition
        DEFB    $A1                 ;;stk-one
        DEFB    $03                 ;;subtract
        DEFB    $88                 ;;series-08
        DEFB    $13                 ;;Exponent: $63, Bytes: 1
        DEFB    $36                 ;;(+00,+00,+00)
        DEFB    $58                 ;;Exponent: $68, Bytes: 2
        DEFB    $65,$66             ;;(+00,+00)
        DEFB    $9D                 ;;Exponent: $6D, Bytes: 3
        DEFB    $78,$65,$40         ;;(+00)
        DEFB    $A2                 ;;Exponent: $72, Bytes: 3
        DEFB    $60,$32,$C9         ;;(+00)
        DEFB    $E7                 ;;Exponent: $77, Bytes: 4
        DEFB    $21,$F7,$AF,$24 ;;
        DEFB    $EB                 ;;Exponent: $7B, Bytes: 4
        DEFB    $2F,$B0,$B0,$14 ;;
        DEFB    $EE                 ;;Exponent: $7E, Bytes: 4
        DEFB    $7E,$BB,$94,$58 ;;
        DEFB    $F1                 ;;Exponent: $81, Bytes: 4
        DEFB    $3A,$7E,$F8,$CF ;;
        DEFB    $E3                 ;;get-mem-3
        DEFB    $38                 ;;end-calc

        CALL    L2DD5               ; routine FP-TO-A
        JR      NZ,L3705            ; to N-NEGTV

        JR      C,L3703             ; to REPORT-6b
                                    ; 'Number too big'

        ADD     A,(HL)              ;
        JR      NC,L370C            ; to RESULT-OK


;; REPORT-6b
L3703:  RST     08H                 ; ERROR-1
        DEFB    $05                 ; Error Report: Number too big

; ---

;; N-NEGTV
L3705:  JR      C,L370E             ; to RSLT-ZERO

        SUB     (HL)                ;
        JR      NC,L370E            ; to RSLT-ZERO

        NEG                         ; Negate

;; RESULT-OK
L370C:  LD      (HL),A              ;
        RET                         ; return.

; ---


;; RSLT-ZERO
```

```
L370E:  RST     28H             ;; FP-CALC
        DEFB    $02             ;;delete
        DEFB    $A0             ;;stk-zero
        DEFB    $38             ;;end-calc

        RET                     ; return.


; --------------------------------
; THE 'NATURAL LOGARITHM' FUNCTION
; --------------------------------
; (offset $25: 'ln')
;   Function to calculate the natural logarithm (to the base e ).
;   e.g.   LN EXP 5.3 = 5.3
;   Error A if the argument is 0 or negative.

;; ln
L3713:  RST     28H             ;; FP-CALC
        DEFB    $3D             ;;re-stack
        DEFB    $31             ;;duplicate
        DEFB    $37             ;;greater-0
        DEFB    $00             ;;jump-true
        DEFB    $04             ;;to L371C, VALID

        DEFB    $38             ;;end-calc


;; REPORT-Ab
L371A:  RST     08H             ; ERROR-1
        DEFB    $09             ; Error Report: Invalid argument

;; VALID
L371C:  DEFB    $A0             ;;stk-zero
        DEFB    $02             ;;delete
        DEFB    $38             ;;end-calc
        LD      A,(HL)          ;

        LD      (HL),$80        ;
        CALL    L2D28           ; routine STACK-A

        RST     28H             ;; FP-CALC
        DEFB    $34             ;;stk-data
        DEFB    $38             ;;Exponent: $88, Bytes: 1
        DEFB    $00             ;;(+00,+00,+00)
        DEFB    $03             ;;subtract
        DEFB    $01             ;;exchange
        DEFB    $31             ;;duplicate
        DEFB    $34             ;;stk-data
        DEFB    $F0             ;;Exponent: $80, Bytes: 4
        DEFB    $4C,$CC,$CC,$CD ;;
        DEFB    $03             ;;subtract
        DEFB    $37             ;;greater-0
        DEFB    $00             ;;jump-true
        DEFB    $08             ;;to L373D, GRE.8

        DEFB    $01             ;;exchange
        DEFB    $A1             ;;stk-one
        DEFB    $03             ;;subtract
        DEFB    $01             ;;exchange
        DEFB    $38             ;;end-calc

        INC     (HL)            ;

        RST     28H             ;; FP-CALC
```

```
;; GRE.8
L373D:  DEFB    $01             ;;exchange
        DEFB    $34             ;;stk-data
        DEFB    $F0             ;;Exponent: $80, Bytes: 4
        DEFB    $31,$72,$17,$F8 ;;
        DEFB    $04             ;;multiply
        DEFB    $01             ;;exchange
        DEFB    $A2             ;;stk-half
        DEFB    $03             ;;subtract
        DEFB    $A2             ;;stk-half
        DEFB    $03             ;;subtract
        DEFB    $31             ;;duplicate
        DEFB    $34             ;;stk-data
        DEFB    $32             ;;Exponent: $82, Bytes: 1
        DEFB    $20             ;;(+00,+00,+00)
        DEFB    $04             ;;multiply
        DEFB    $A2             ;;stk-half
        DEFB    $03             ;;subtract
        DEFB    $8C             ;;series-0C
        DEFB    $11             ;;Exponent: $61, Bytes: 1
        DEFB    $AC             ;;(+00,+00,+00)
        DEFB    $14             ;;Exponent: $64, Bytes: 1
        DEFB    $09             ;;(+00,+00,+00)
        DEFB    $56             ;;Exponent: $66, Bytes: 2
        DEFB    $DA,$A5         ;;(+00,+00)
        DEFB    $59             ;;Exponent: $69, Bytes: 2
        DEFB    $30,$C5         ;;(+00,+00)
        DEFB    $5C             ;;Exponent: $6C, Bytes: 2
        DEFB    $90,$AA         ;;(+00,+00)
        DEFB    $9E             ;;Exponent: $6E, Bytes: 3
        DEFB    $70,$6F,$61     ;;(+00)
        DEFB    $A1             ;;Exponent: $71, Bytes: 3
        DEFB    $CB,$DA,$96     ;;(+00)
        DEFB    $A4             ;;Exponent: $74, Bytes: 3
        DEFB    $31,$9F,$B4     ;;(+00)
        DEFB    $E7             ;;Exponent: $77, Bytes: 4
        DEFB    $A0,$FE,$5C,$FC ;;
        DEFB    $EA             ;;Exponent: $7A, Bytes: 4
        DEFB    $1B,$43,$CA,$36 ;;
        DEFB    $ED             ;;Exponent: $7D, Bytes: 4
        DEFB    $A7,$9C,$7E,$5E ;;
        DEFB    $F0             ;;Exponent: $80, Bytes: 4
        DEFB    $6E,$23,$80,$93 ;;
        DEFB    $04             ;;multiply
        DEFB    $0F             ;;addition
        DEFB    $38             ;;end-calc

        RET                     ; return.


; ------------------------------
; THE 'TRIGONOMETRIC' FUNCTIONS
; ------------------------------
; Trigonometry is rocket science. It is also used by carpenters and pyramid
; builders.
; Some uses can be quite abstract but the principles can be seen in simple
; right-angled triangles. Triangles have some special properties -
;
; 1) The sum of the three angles is always PI radians (180 degrees).
;    Very helpful if you know two angles and wish to find the third.
; 2) In any right-angled triangle the sum of the squares of the two shorter
;    sides is equal to the square of the longest side opposite the right-angle.
;    Very useful if you know the length of two sides and wish to know the
```

```
;    length of the third side.
; 3) Functions sine, cosine and tangent enable one to calculate the length
;    of an unknown side when the length of one other side and an angle is
;    known.
; 4) Functions arcsin, arccosine and arctan enable one to calculate an unknown
;    angle when the length of two of the sides is known.

; --------------------------------
; THE 'REDUCE ARGUMENT' SUBROUTINE
; --------------------------------
; (offset $39: 'get-argt')
;
; This routine performs two functions on the angle, in radians, that forms
; the argument to the sine and cosine functions.
; First it ensures that the angle 'wraps round'. That if a ship turns through
; an angle of, say, 3*PI radians (540 degrees) then the net effect is to turn
; through an angle of PI radians (180 degrees).
; Secondly it converts the angle in radians to a fraction of a right angle,
; depending within which quadrant the angle lies, with the periodicity
; resembling that of the desired sine value.
; The result lies in the range -1 to +1.
;
;                     90 deg.
;
;                     (pi/2)
;              II       +1        I
;                       |
;        sin+      |\   |   /|    sin+
;        cos-      | \  |  / |    cos+
;        tan-      |  \ | /  |    tan+
;                  |   \|/)  |
; 180 deg. (pi) 0 -|----+----|-- 0  (0)   0 degrees
;                  |   /|\   |
;        sin-      |  / | \  |    sin-
;        cos-      | /  |  \ |    cos+
;        tan+      |/   |   \|    tan-
;                       |
;              III      -1       IV
;                     (3pi/2)
;
;                     270 deg.
;

;; get-argt
L3783:  RST     28H             ;; FP-CALC      X.
        DEFB    $3D             ;;re-stack
        DEFB    $34             ;;stk-data
        DEFB    $EE             ;;Exponent: $7E,
                                ;;Bytes: 4
        DEFB    $22,$F9,$83,$6E ;;              X, 1/(2*PI)
        DEFB    $04             ;;multiply      X/(2*PI) = fraction
        DEFB    $31             ;;duplicate
        DEFB    $A2             ;;stk-half
        DEFB    $0F             ;;addition
        DEFB    $27             ;;int

        DEFB    $03             ;;subtract      now range -.5 to .5

        DEFB    $31             ;;duplicate
        DEFB    $0F             ;;addition      now range -1 to 1.
        DEFB    $31             ;;duplicate
        DEFB    $0F             ;;addition      now range -2 to +2.

; quadrant I (0 to +1) and quadrant IV (-1 to 0) are now correct.
```

```
; quadrant II ranges +1 to +2.
; quadrant III ranges -2 to -1.

        DEFB    $31             ;;duplicate     Y, Y.
        DEFB    $2A             ;;abs           Y, abs(Y).    range 1 to 2
        DEFB    $A1             ;;stk-one       Y, abs(Y), 1.
        DEFB    $03             ;;subtract      Y, abs(Y)-1.  range 0 to 1
        DEFB    $31             ;;duplicate     Y, Z, Z.
        DEFB    $37             ;;greater-0     Y, Z, (1/0).

        DEFB    $C0             ;;st-mem-0         store as possible sign
                                ;;                 for cosine function.

        DEFB    $00             ;;jump-true
        DEFB    $04             ;;to L37A1, ZPLUS  with quadrants II and III.

; else the angle lies in quadrant I or IV and value Y is already correct.

        DEFB    $02             ;;delete        Y.   delete the test value.
        DEFB    $38             ;;end-calc      Y.

        RET                     ; return.       with Q1 and Q4           >>>

; ---

; the branch was here with quadrants II (0 to 1) and III (1 to 0).
; Y will hold -2 to -1 if this is quadrant III.

;; ZPLUS
L37A1:  DEFB    $A1             ;;stk-one         Y, Z, 1.
        DEFB    $03             ;;subtract        Y, Z-1.       Q3 = 0 to -1
        DEFB    $01             ;;exchange        Z-1, Y.
        DEFB    $36             ;;less-0          Z-1, (1/0).
        DEFB    $00             ;;jump-true       Z-1.
        DEFB    $02             ;;to L37A8, YNEG
                                ;;if angle in quadrant III

; else angle is within quadrant II (-1 to 0)

        DEFB    $1B             ;;negate          range +1 to 0.

;; YNEG
L37A8:  DEFB    $38             ;;end-calc        quadrants II and III correct.

        RET                     ; return.


; ---------------------
; THE 'COSINE' FUNCTION
; ---------------------
; (offset $20: 'cos')
; Cosines are calculated as the sine of the opposite angle rectifying the
; sign depending on the quadrant rules.
;
;
;           /|
;        h /y|
;         /  |o
;        /x  |
;       /----|
;         a
;
; The cosine of angle x is the adjacent side (a) divided by the hypotenuse 1.
; However if we examine angle y then a/h is the sine of that angle.
```

```
; Since angle x plus angle y equals a right-angle, we can find angle y by
; subtracting angle x from pi/2.
; However it's just as easy to reduce the argument first and subtract the
; reduced argument from the value 1 (a reduced right-angle).
; It's even easier to subtract 1 from the angle and rectify the sign.
; In fact, after reducing the argument, the absolute value of the argument
; is used and rectified using the test result stored in mem-0 by 'get-argt'
; for that purpose.
;

;; cos
L37AA:  RST     28H             ;; FP-CALC              angle in radians.
        DEFB    $39             ;;get-argt              X     reduce -1 to +1

        DEFB    $2A             ;;abs                   ABS X.   0 to 1
        DEFB    $A1             ;;stk-one               ABS X, 1.
        DEFB    $03             ;;subtract              now opposite angle
                                ;;                      although sign is -ve.

        DEFB    $E0             ;;get-mem-0             fetch the sign indicator
        DEFB    $00             ;;jump-true
        DEFB    $06             ;;fwd to L37B7, C-ENT
                                ;;forward to common code if in QII or QIII.

        DEFB    $1B             ;;negate                else make sign +ve.
        DEFB    $33             ;;jump
        DEFB    $03             ;;fwd to L37B7, C-ENT
                                ;; with quadrants I and IV.

; -------------------
; THE 'SINE' FUNCTION
; -------------------
; (offset $1F: 'sin')
; This is a fundamental transcendental function from which others such as cos
; and tan are directly, or indirectly, derived.
; It uses the series generator to produce Chebyshev polynomials.
;
;
;           /|
;        1 / |
;         /  |x
;        /a  |
;       /----|
;         y
;
; The 'get-argt' function is designed to modify the angle and its sign
; in line with the desired sine value and afterwards it can launch straight
; into common code.

;; sin
L37B5:  RST     28H             ;; FP-CALC      angle in radians
        DEFB    $39             ;;get-argt      reduce - sign now correct.

;; C-ENT
L37B7:  DEFB    $31             ;;duplicate
        DEFB    $31             ;;duplicate
        DEFB    $04             ;;multiply
        DEFB    $31             ;;duplicate
        DEFB    $0F             ;;addition
        DEFB    $A1             ;;stk-one
        DEFB    $03             ;;subtract

        DEFB    $86             ;;series-06
        DEFB    $14             ;;Exponent: $64, Bytes: 1
```

```
        DEFB    $E6             ;;(+00,+00,+00)
        DEFB    $5C             ;;Exponent: $6C, Bytes: 2
        DEFB    $1F,$0B         ;;(+00,+00)
        DEFB    $A3             ;;Exponent: $73, Bytes: 3
        DEFB    $8F,$38,$EE     ;;(+00)
        DEFB    $E9             ;;Exponent: $79, Bytes: 4
        DEFB    $15,$63,$BB,$23 ;;
        DEFB    $EE             ;;Exponent: $7E, Bytes: 4
        DEFB    $92,$0D,$CD,$ED ;;
        DEFB    $F1             ;;Exponent: $81, Bytes: 4
        DEFB    $23,$5D,$1B,$EA ;;
        DEFB    $04             ;;multiply
        DEFB    $38             ;;end-calc

        RET                     ; return.

; ----------------------
; THE 'TANGENT' FUNCTION
; ----------------------
; (offset $21: 'tan')
;
; Evaluates tangent x as    sin(x) / cos(x).
;
;
;           /|
;        h / |
;         /  |o
;        /x  |
;       /----|
;         a
;
; the tangent of angle x is the ratio of the length of the opposite side
; divided by the length of the adjacent side. As the opposite length can
; be calculates using sin(x) and the adjacent length using cos(x) then
; the tangent can be defined in terms of the previous two functions.

; Error 6 if the argument, in radians, is too close to one like pi/2
; which has an infinite tangent. e.g. PRINT TAN (PI/2)  evaluates as 1/0.
; Similarly PRINT TAN (3*PI/2), TAN (5*PI/2) etc.

;; tan
L37DA:  RST     28H             ;; 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).
        DEFB    $38             ;;end-calc          tan x.

        RET                     ; return.

; ---------------------
; THE 'ARCTAN' FUNCTION
; ---------------------
; (Offset $24: 'atn')
; the inverse tangent function with the result in radians.
; This is a fundamental transcendental function from which others such as asn
; and acs are directly, or indirectly, derived.
; It uses the series generator to produce Chebyshev polynomials.

;; atn
L37E2:  CALL    L3297           ; routine re-stack
        LD      A,(HL)          ; fetch exponent byte.
        CP      $81             ; compare to that for 'one'
```

```
        JR      C,L37F8         ; forward, if less, to SMALL

        RST     28H             ;; FP-CALC
        DEFB    $A1             ;;stk-one
        DEFB    $1B             ;;negate
        DEFB    $01             ;;exchange
        DEFB    $05             ;;division
        DEFB    $31             ;;duplicate
        DEFB    $36             ;;less-0
        DEFB    $A3             ;;stk-pi/2
        DEFB    $01             ;;exchange
        DEFB    $00             ;;jump-true
        DEFB    $06             ;;to L37FA, CASES

        DEFB    $1B             ;;negate
        DEFB    $33             ;;jump
        DEFB    $03             ;;to L37FA, CASES

;; SMALL
L37F8:  RST     28H             ;; FP-CALC
        DEFB    $A0             ;;stk-zero

;; CASES
L37FA:  DEFB    $01             ;;exchange
        DEFB    $31             ;;duplicate
        DEFB    $31             ;;duplicate
        DEFB    $04             ;;multiply
        DEFB    $31             ;;duplicate
        DEFB    $0F             ;;addition
        DEFB    $A1             ;;stk-one
        DEFB    $03             ;;subtract
        DEFB    $8C             ;;series-0C
        DEFB    $10             ;;Exponent: $60, Bytes: 1
        DEFB    $B2             ;;(+00,+00,+00)
        DEFB    $13             ;;Exponent: $63, Bytes: 1
        DEFB    $0E             ;;(+00,+00,+00)
        DEFB    $55             ;;Exponent: $65, Bytes: 2
        DEFB    $E4,$8D         ;;(+00,+00)
        DEFB    $58             ;;Exponent: $68, Bytes: 2
        DEFB    $39,$BC         ;;(+00,+00)
        DEFB    $5B             ;;Exponent: $6B, Bytes: 2
        DEFB    $98,$FD         ;;(+00,+00)
        DEFB    $9E             ;;Exponent: $6E, Bytes: 3
        DEFB    $00,$36,$75     ;;(+00)
        DEFB    $A0             ;;Exponent: $70, Bytes: 3
        DEFB    $DB,$E8,$B4     ;;(+00)
        DEFB    $63             ;;Exponent: $73, Bytes: 2
        DEFB    $42,$C4         ;;(+00,+00)
        DEFB    $E6             ;;Exponent: $76, Bytes: 4
        DEFB    $B5,$09,$36,$BE ;;
        DEFB    $E9             ;;Exponent: $79, Bytes: 4
        DEFB    $36,$73,$1B,$5D ;;
        DEFB    $EC             ;;Exponent: $7C, Bytes: 4
        DEFB    $D8,$DE,$63,$BE ;;
        DEFB    $F0             ;;Exponent: $80, Bytes: 4
        DEFB    $61,$A1,$B3,$0C ;;
        DEFB    $04             ;;multiply
        DEFB    $0F             ;;addition
        DEFB    $38             ;;end-calc

        RET                     ; return.


; ---------------------
```

```
; THE 'ARCSIN' FUNCTION
; ---------------------
; (Offset $22: 'asn')
; the inverse sine function with result in radians.
; derived from arctan function above.
; Error A unless the argument is between -1 and +1 inclusive.
; uses an adaptation of the formula asn(x) = atn(x/sqr(1-x*x))
;
;
;           /|
;        1 / |
;         /  |x
;        /a  |
;       /----|
;         y
;
; e.g. we know the opposite side (x) and hypotenuse (1)
; and we wish to find angle a in radians.
; we can derive length y by Pythagoras and then use ATN instead.
; since y*y + x*x = 1*1 (Pythagoras Theorem) then
; y=sqr(1-x*x)                        - no need to multiply 1 by itself.
; so, asn(a) = atn(x/y)
; or more fully,
; asn(a) = atn(x/sqr(1-x*x))

; Close but no cigar.

; While PRINT ATN (x/SQR (1-x*x)) gives the same results as PRINT ASN x,
; it leads to division by zero when x is 1 or -1.
; To overcome this, 1 is added to y giving half the required angle and the
; result is then doubled.
; That is PRINT ATN (x/(SQR (1-x*x) +1)) *2
; A value higher than 1 gives the required error as attempting to find  the
; square root of a negative number generates an error in Sinclair BASIC.

;; asn
L3833:  RST     28H             ;; 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) = y
        DEFB    $A1             ;;stk-one       x, y, 1.
        DEFB    $0F             ;;addition      x, y+1.
        DEFB    $05             ;;division      x/y+1.
        DEFB    $24             ;;atn           a/2       (half the angle)
        DEFB    $31             ;;duplicate     a/2, a/2.
        DEFB    $0F             ;;addition      a.
        DEFB    $38             ;;end-calc      a.

        RET                     ; return.


; ---------------------
; THE 'ARCCOS' FUNCTION
; ---------------------
; (Offset $23: 'acs')
; the inverse cosine function with the result in radians.
; Error A unless the argument is between -1 and +1.
; Result in range 0 to pi.
; Derived from asn above which is in turn derived from the preceding atn.
; It could have been derived directly from atn using acs(x) = atn(sqr(1-x*x)/x).
```

```
; However, as sine and cosine are horizontal translations of each other,
; uses acs(x) = pi/2 - asn(x)

; e.g. the arccosine of a known x value will give the required angle b in
; radians.
; We know, from above, how to calculate the angle a using asn(x).
; Since the three angles of any triangle add up to 180 degrees, or pi radians,
; and the largest angle in this case is a right-angle (pi/2 radians), then
; we can calculate angle b as pi/2 (both angles) minus asn(x) (angle a).
;
;
;           /|
;        1 /b|
;         /  |x
;        /a  |
;       /----|
;         y
;

;; acs
L3843:  RST     28H             ;; FP-CALC      x.
        DEFB    $22             ;;asn           asn(x).
        DEFB    $A3             ;;stk-pi/2      asn(x), pi/2.
        DEFB    $03             ;;subtract      asn(x) - pi/2.
        DEFB    $1B             ;;negate        pi/2 -asn(x)  =  acs(x).
        DEFB    $38             ;;end-calc      acs(x).

        RET                     ; return.


; --------------------------
; THE 'SQUARE ROOT' FUNCTION
; --------------------------
; (Offset $28: 'sqr')
; This routine is remarkable only in its brevity - 7 bytes.
; It wasn't written here but in the ZX81 where the programmers had to squeeze
; a bulky operating system into an 8K ROM. It simply calculates
; the square root by stacking the value .5 and continuing into the 'to-power'
; routine. With more space available the much faster Newton-Raphson method
; should have been used as on the Jupiter Ace.

;; sqr
L384A:  RST     28H             ;; FP-CALC
        DEFB    $31             ;;duplicate
        DEFB    $30             ;;not
        DEFB    $00             ;;jump-true
        DEFB    $1E             ;;to L386C, LAST

        DEFB    $A2             ;;stk-half
        DEFB    $38             ;;end-calc


; ------------------------------
; THE 'EXPONENTIATION' OPERATION
; ------------------------------
; (Offset $06: 'to-power')
; This raises the first number X to the power of the second number Y.
; As with the ZX80,
; 0 ^ 0 = 1.
; 0 ^ +n = 0.
; 0 ^ -n = arithmetic overflow.
;

;; to-power
```

```
L3851:  RST     28H             ;; 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
        DEFB    $07             ;;to L385D, XISO   if X is zero.

; else X is non-zero. Function 'ln' will catch a negative value of X.

        DEFB    $25             ;;ln                    Y, LN X.
        DEFB    $04             ;;multiply              Y * LN X.
        DEFB    $38             ;;end-calc

        JP      L36C4           ; jump back to EXP routine   ->

; ---

; these routines form the three simple results when the number is zero.
; begin by deleting the known zero to leave Y the power factor.

;; XISO
L385D:  DEFB    $02             ;;delete                Y.
        DEFB    $31             ;;duplicate             Y, Y.
        DEFB    $30             ;;not                   Y, (1/0).
        DEFB    $00             ;;jump-true
        DEFB    $09             ;;to L386A, ONE         if Y is zero.

        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 L386C, LAST        if Y was any positive
                                ;;                      number.

; else force division by zero thereby raising an Arithmetic overflow error.
; There are some one and two-byte alternatives but perhaps the most formal
; might have been to use end-calc; rst 08; defb 05.

        DEFB    $A1             ;;stk-one               0, 1.
        DEFB    $01             ;;exchange              1, 0.
        DEFB    $05             ;;division              1/0        ouch!

; ---

;; ONE
L386A:  DEFB    $02             ;;delete                .
        DEFB    $A1             ;;stk-one               1.

;; LAST
L386C:  DEFB    $38             ;;end-calc              last value is 1 or 0.

        RET                     ; return.               Whew!

; ---------------------
; THE 'SPARE' LOCATIONS
; ---------------------

;; spare
L386E:  DEFB    $FF, $FF        ;


        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
```

```
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
```

```
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
```

```
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;
        DEFB    $FF, $FF, $FF, $FF, $FF, $FF, $FF, $FF;

; ORG $3D00

; -------------------------------
; THE 'ZX SPECTRUM CHARACTER SET'
; -------------------------------

;; char-set

; $20 - Character: ' '          CHR$(32)

L3D00:  DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000

; $21 - Character: '!'          CHR$(33)

        DEFB    %00000000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00000000
        DEFB    %00010000
        DEFB    %00000000

; $22 - Character: '"'          CHR$(34)

        DEFB    %00000000
        DEFB    %00100100
        DEFB    %00100100
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000

; $23 - Character: '#'          CHR$(35)

        DEFB    %00000000
        DEFB    %00100100
        DEFB    %01111110
        DEFB    %00100100
        DEFB    %00100100
```

```
        DEFB    %01111110
        DEFB    %00100100
        DEFB    %00000000

; $24 - Character: '$'          CHR$(36)

        DEFB    %00000000
        DEFB    %00001000
        DEFB    %00111110
        DEFB    %00101000
        DEFB    %00111110
        DEFB    %00001010
        DEFB    %00111110
        DEFB    %00001000

; $25 - Character: '%'          CHR$(37)

        DEFB    %00000000
        DEFB    %01100010
        DEFB    %01100100
        DEFB    %00001000
        DEFB    %00010000
        DEFB    %00100110
        DEFB    %01000110
        DEFB    %00000000

; $26 - Character: '&'          CHR$(38)

        DEFB    %00000000
        DEFB    %00010000
        DEFB    %00101000
        DEFB    %00010000
        DEFB    %00101010
        DEFB    %01000100
        DEFB    %00111010
        DEFB    %00000000

; $27 - Character: '''          CHR$(39)

        DEFB    %00000000
        DEFB    %00001000
        DEFB    %00010000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000

; $28 - Character: '('          CHR$(40)

        DEFB    %00000000
        DEFB    %00000100
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00000100
        DEFB    %00000000

; $29 - Character: ')'          CHR$(41)

        DEFB    %00000000
        DEFB    %00100000
        DEFB    %00010000
```

```
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00100000
        DEFB    %00000000

; $2A - Character: '*'          CHR$(42)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00010100
        DEFB    %00001000
        DEFB    %00111110
        DEFB    %00001000
        DEFB    %00010100
        DEFB    %00000000

; $2B - Character: '+'          CHR$(43)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00111110
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00000000

; $2C - Character: ','          CHR$(44)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00010000

; $2D - Character: '-'          CHR$(45)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00111110
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000

; $2E - Character: '.'          CHR$(46)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00011000
        DEFB    %00011000
        DEFB    %00000000

; $2F - Character: '/'          CHR$(47)

        DEFB    %00000000
```

```
        DEFB    %00000000
        DEFB    %00000010
        DEFB    %00000100
        DEFB    %00001000
        DEFB    %00010000
        DEFB    %00100000
        DEFB    %00000000

; $30 - Character: '0'          CHR$(48)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000110
        DEFB    %01001010
        DEFB    %01010010
        DEFB    %01100010
        DEFB    %00111100
        DEFB    %00000000

; $31 - Character: '1'          CHR$(49)

        DEFB    %00000000
        DEFB    %00011000
        DEFB    %00101000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00111110
        DEFB    %00000000

; $32 - Character: '2'          CHR$(50)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000010
        DEFB    %00000010
        DEFB    %00111100
        DEFB    %01000000
        DEFB    %01111110
        DEFB    %00000000

; $33 - Character: '3'          CHR$(51)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000010
        DEFB    %00001100
        DEFB    %00000010
        DEFB    %01000010
        DEFB    %00111100
        DEFB    %00000000

; $34 - Character: '4'          CHR$(52)

        DEFB    %00000000
        DEFB    %00001000
        DEFB    %00011000
        DEFB    %00101000
        DEFB    %01001000
        DEFB    %01111110
        DEFB    %00001000
        DEFB    %00000000

; $35 - Character: '5'          CHR$(53)
```

```
        DEFB    %00000000
        DEFB    %01111110
        DEFB    %01000000
        DEFB    %01111100
        DEFB    %00000010
        DEFB    %01000010
        DEFB    %00111100
        DEFB    %00000000

; $36 - Character: '6'          CHR$(54)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000000
        DEFB    %01111100
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %00111100
        DEFB    %00000000

; $37 - Character: '7'          CHR$(55)

        DEFB    %00000000
        DEFB    %01111110
        DEFB    %00000010
        DEFB    %00000100
        DEFB    %00001000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00000000

; $38 - Character: '8'          CHR$(56)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000010
        DEFB    %00111100
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %00111100
        DEFB    %00000000

; $39 - Character: '9'          CHR$(57)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %00111110
        DEFB    %00000010
        DEFB    %00111100
        DEFB    %00000000

; $3A - Character: ':'          CHR$(58)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00010000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00010000
        DEFB    %00000000
```

```
; $3B - Character: ';'          CHR$(59)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00010000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00100000

; $3C - Character: '<'          CHR$(60)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000100
        DEFB    %00001000
        DEFB    %00010000
        DEFB    %00001000
        DEFB    %00000100
        DEFB    %00000000

; $3D - Character: '='          CHR$(61)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00111110
        DEFB    %00000000
        DEFB    %00111110
        DEFB    %00000000
        DEFB    %00000000

; $3E - Character: '>'          CHR$(62)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00010000
        DEFB    %00001000
        DEFB    %00000100
        DEFB    %00001000
        DEFB    %00010000
        DEFB    %00000000

; $3F - Character: '?'          CHR$(63)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000010
        DEFB    %00000100
        DEFB    %00001000
        DEFB    %00000000
        DEFB    %00001000
        DEFB    %00000000

; $40 - Character: '@'          CHR$(64)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01001010
        DEFB    %01010110
        DEFB    %01011110
        DEFB    %01000000
```

```
        DEFB    %00111100
        DEFB    %00000000

; $41 - Character: 'A'          CHR$(65)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01111110
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %00000000

; $42 - Character: 'B'          CHR$(66)

        DEFB    %00000000
        DEFB    %01111100
        DEFB    %01000010
        DEFB    %01111100
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01111100
        DEFB    %00000000

; $43 - Character: 'C'          CHR$(67)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000010
        DEFB    %01000000
        DEFB    %01000000
        DEFB    %01000010
        DEFB    %00111100
        DEFB    %00000000

; $44 - Character: 'D'          CHR$(68)

        DEFB    %00000000
        DEFB    %01111000
        DEFB    %01000100
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01000100
        DEFB    %01111000
        DEFB    %00000000

; $45 - Character: 'E'          CHR$(69)

        DEFB    %00000000
        DEFB    %01111110
        DEFB    %01000000
        DEFB    %01111100
        DEFB    %01000000
        DEFB    %01000000
        DEFB    %01111110
        DEFB    %00000000

; $46 - Character: 'F'          CHR$(70)

        DEFB    %00000000
        DEFB    %01111110
        DEFB    %01000000
        DEFB    %01111100
```

```
        DEFB    %01000000
        DEFB    %01000000
        DEFB    %01000000
        DEFB    %00000000

; $47 - Character: 'G'          CHR$(71)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000010
        DEFB    %01000000
        DEFB    %01001110
        DEFB    %01000010
        DEFB    %00111100
        DEFB    %00000000

; $48 - Character: 'H'          CHR$(72)

        DEFB    %00000000
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01111110
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %00000000

; $49 - Character: 'I'          CHR$(73)

        DEFB    %00000000
        DEFB    %00111110
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00111110
        DEFB    %00000000

; $4A - Character: 'J'          CHR$(74)

        DEFB    %00000000
        DEFB    %00000010
        DEFB    %00000010
        DEFB    %00000010
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %00111100
        DEFB    %00000000

; $4B - Character: 'K'          CHR$(75)

        DEFB    %00000000
        DEFB    %01000100
        DEFB    %01001000
        DEFB    %01110000
        DEFB    %01001000
        DEFB    %01000100
        DEFB    %01000010
        DEFB    %00000000

; $4C - Character: 'L'          CHR$(76)

        DEFB    %00000000
        DEFB    %01000000
```

```
        DEFB    %01000000
        DEFB    %01000000
        DEFB    %01000000
        DEFB    %01000000
        DEFB    %01111110
        DEFB    %00000000

; $4D - Character: 'M'          CHR$(77)

        DEFB    %00000000
        DEFB    %01000010
        DEFB    %01100110
        DEFB    %01011010
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %00000000

; $4E - Character: 'N'          CHR$(78)

        DEFB    %00000000
        DEFB    %01000010
        DEFB    %01100010
        DEFB    %01010010
        DEFB    %01001010
        DEFB    %01000110
        DEFB    %01000010
        DEFB    %00000000

; $4F - Character: 'O'          CHR$(79)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %00111100
        DEFB    %00000000

; $50 - Character: 'P'          CHR$(80)

        DEFB    %00000000
        DEFB    %01111100
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01111100
        DEFB    %01000000
        DEFB    %01000000
        DEFB    %00000000

; $51 - Character: 'Q'          CHR$(81)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01010010
        DEFB    %01001010
        DEFB    %00111100
        DEFB    %00000000

; $52 - Character: 'R'          CHR$(82)
```

```
        DEFB    %00000000
        DEFB    %01111100
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01111100
        DEFB    %01000100
        DEFB    %01000010
        DEFB    %00000000

; $53 - Character: 'S'          CHR$(83)

        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000000
        DEFB    %00111100
        DEFB    %00000010
        DEFB    %01000010
        DEFB    %00111100
        DEFB    %00000000

; $54 - Character: 'T'          CHR$(84)

        DEFB    %00000000
        DEFB    %11111110
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00000000

; $55 - Character: 'U'          CHR$(85)

        DEFB    %00000000
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %00111100
        DEFB    %00000000

; $56 - Character: 'V'          CHR$(86)

        DEFB    %00000000
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %00100100
        DEFB    %00011000
        DEFB    %00000000

; $57 - Character: 'W'          CHR$(87)

        DEFB    %00000000
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01000010
        DEFB    %01011010
        DEFB    %00100100
        DEFB    %00000000
```

```
; $58 - Character: 'X'          CHR$(88)

        DEFB    %00000000
        DEFB    %01000010
        DEFB    %00100100
        DEFB    %00011000
        DEFB    %00011000
        DEFB    %00100100
        DEFB    %01000010
        DEFB    %00000000

; $59 - Character: 'Y'          CHR$(89)

        DEFB    %00000000
        DEFB    %10000010
        DEFB    %01000100
        DEFB    %00101000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00000000

; $5A - Character: 'Z'          CHR$(90)

        DEFB    %00000000
        DEFB    %01111110
        DEFB    %00000100
        DEFB    %00001000
        DEFB    %00010000
        DEFB    %00100000
        DEFB    %01111110
        DEFB    %00000000

; $5B - Character: '['          CHR$(91)

        DEFB    %00000000
        DEFB    %00001110
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00001110
        DEFB    %00000000

; $5C - Character: '\'          CHR$(92)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %01000000
        DEFB    %00100000
        DEFB    %00010000
        DEFB    %00001000
        DEFB    %00000100
        DEFB    %00000000

; $5D - Character: ']'          CHR$(93)

        DEFB    %00000000
        DEFB    %01110000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %01110000
```

```
        DEFB    %00000000

; $5E - Character: '^'          CHR$(94)

        DEFB    %00000000
        DEFB    %00010000
        DEFB    %00111000
        DEFB    %01010100
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00000000

; $5F - Character: '_'          CHR$(95)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %11111111

; $60 - Character: 'ukp'        CHR$(96)

        DEFB    %00000000
        DEFB    %00011100
        DEFB    %00100010
        DEFB    %01111000
        DEFB    %00100000
        DEFB    %00100000
        DEFB    %01111110
        DEFB    %00000000

; $61 - Character: 'a'          CHR$(97)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00111000
        DEFB    %00000100
        DEFB    %00111100
        DEFB    %01000100
        DEFB    %00111100
        DEFB    %00000000

; $62 - Character: 'b'          CHR$(98)

        DEFB    %00000000
        DEFB    %00100000
        DEFB    %00100000
        DEFB    %00111100
        DEFB    %00100010
        DEFB    %00100010
        DEFB    %00111100
        DEFB    %00000000

; $63 - Character: 'c'          CHR$(99)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00011100
        DEFB    %00100000
        DEFB    %00100000
```

```
        DEFB    %00100000
        DEFB    %00011100
        DEFB    %00000000

; $64 - Character: 'd'          CHR$(100)

        DEFB    %00000000
        DEFB    %00000100
        DEFB    %00000100
        DEFB    %00111100
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %00111100
        DEFB    %00000000

; $65 - Character: 'e'          CHR$(101)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00111000
        DEFB    %01000100
        DEFB    %01111000
        DEFB    %01000000
        DEFB    %00111100
        DEFB    %00000000

; $66 - Character: 'f'          CHR$(102)

        DEFB    %00000000
        DEFB    %00001100
        DEFB    %00010000
        DEFB    %00011000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00000000

; $67 - Character: 'g'          CHR$(103)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %00111100
        DEFB    %00000100
        DEFB    %00111000

; $68 - Character: 'h'          CHR$(104)

        DEFB    %00000000
        DEFB    %01000000
        DEFB    %01000000
        DEFB    %01111000
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %00000000

; $69 - Character: 'i'          CHR$(105)

        DEFB    %00000000
        DEFB    %00010000
        DEFB    %00000000
```

```
        DEFB    %00110000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00111000
        DEFB    %00000000

; $6A - Character: 'j'          CHR$(106)

        DEFB    %00000000
        DEFB    %00000100
        DEFB    %00000000
        DEFB    %00000100
        DEFB    %00000100
        DEFB    %00000100
        DEFB    %00100100
        DEFB    %00011000

; $6B - Character: 'k'          CHR$(107)

        DEFB    %00000000
        DEFB    %00100000
        DEFB    %00101000
        DEFB    %00110000
        DEFB    %00110000
        DEFB    %00101000
        DEFB    %00100100
        DEFB    %00000000

; $6C - Character: 'l'          CHR$(108)

        DEFB    %00000000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00001100
        DEFB    %00000000

; $6D - Character: 'm'          CHR$(109)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %01101000
        DEFB    %01010100
        DEFB    %01010100
        DEFB    %01010100
        DEFB    %01010100
        DEFB    %00000000

; $6E - Character: 'n'          CHR$(110)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %01111000
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %00000000

; $6F - Character: 'o'          CHR$(111)

        DEFB    %00000000
```

```
        DEFB    %00000000
        DEFB    %00111000
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %00111000
        DEFB    %00000000

; $70 - Character: 'p'          CHR$(112)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %01111000
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %01111000
        DEFB    %01000000
        DEFB    %01000000

; $71 - Character: 'q'          CHR$(113)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00111100
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %00111100
        DEFB    %00000100
        DEFB    %00000110

; $72 - Character: 'r'          CHR$(114)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00011100
        DEFB    %00100000
        DEFB    %00100000
        DEFB    %00100000
        DEFB    %00100000
        DEFB    %00000000

; $73 - Character: 's'          CHR$(115)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00111000
        DEFB    %01000000
        DEFB    %00111000
        DEFB    %00000100
        DEFB    %01111000
        DEFB    %00000000

; $74 - Character: 't'          CHR$(116)

        DEFB    %00000000
        DEFB    %00010000
        DEFB    %00111000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %00001100
        DEFB    %00000000

; $75 - Character: 'u'          CHR$(117)
```

```
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %00111000
        DEFB    %00000000

; $76 - Character: 'v'          CHR$(118)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %00101000
        DEFB    %00101000
        DEFB    %00010000
        DEFB    %00000000

; $77 - Character: 'w'          CHR$(119)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %01000100
        DEFB    %01010100
        DEFB    %01010100
        DEFB    %01010100
        DEFB    %00101000
        DEFB    %00000000

; $78 - Character: 'x'          CHR$(120)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %01000100
        DEFB    %00101000
        DEFB    %00010000
        DEFB    %00101000
        DEFB    %01000100
        DEFB    %00000000

; $79 - Character: 'y'          CHR$(121)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %01000100
        DEFB    %00111100
        DEFB    %00000100
        DEFB    %00111000

; $7A - Character: 'z'          CHR$(122)

        DEFB    %00000000
        DEFB    %00000000
        DEFB    %01111100
        DEFB    %00001000
        DEFB    %00010000
        DEFB    %00100000
        DEFB    %01111100
        DEFB    %00000000
```

```
; $7B - Character: '{'          CHR$(123)

        DEFB    %00000000
        DEFB    %00001110
        DEFB    %00001000
        DEFB    %00110000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00001110
        DEFB    %00000000

; $7C - Character: '|'          CHR$(124)

        DEFB    %00000000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00001000
        DEFB    %00000000

; $7D - Character: '}'          CHR$(125)

        DEFB    %00000000
        DEFB    %01110000
        DEFB    %00010000
        DEFB    %00001100
        DEFB    %00010000
        DEFB    %00010000
        DEFB    %01110000
        DEFB    %00000000

; $7E - Character: '~'          CHR$(126)

        DEFB    %00000000
        DEFB    %00010100
        DEFB    %00101000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000
        DEFB    %00000000

; $7F - Character: '(c)'        CHR$(127)

        DEFB    %00111100
        DEFB    %01000010
        DEFB    %10011001
        DEFB    %10100001
        DEFB    %10100001
        DEFB    %10011001
        DEFB    %01000010
        DEFB    %00111100


#end                            ; generic cross-assembler directive
```


```
; Acknowledgements
; -----------------
; Sean Irvine               for default list of section headings
; Dr. Ian Logan             for labels and functional disassembly.
; Dr. Frank O'Hara          for labels and functional disassembly.
```

```
;
; Credits
; -------
; Alex Pallero Gonzales     for corrections.
; Mike Dailly               for comments.
; Alvin Albrecht            for comments.
; Andy Styles               for full relocatability implementation and testing.
testing.
; Andrew Owen               for ZASM compatibility and format improvements.

;   For other assemblers you may have to add directives like these near the
;   beginning - see accompanying documentation.
;   ZASM (MacOs) cross-assembler directives. (uncomment by removing ';' )
;   #target rom           ; declare target file format as binary.
;   #code   0,$4000       ; declare code segment.
;   Also see notes at Address Labels 0609 and 1CA5 if your assembler has
;   trouble with expressions.
```