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

; THE 16K "SEA CHANGE" ZX MINIMAL ROM

; -------------------------
; Last updated: 23-FEB-2003
; -------------------------

;   Mission Statement
;   -----------------
;   To produce a user-friendly operating system for a colour computer to exploit
;   the hardware available in the early 1980s.  Apart from a few sensible
;   alphabetical restrictions, there should be no other limitations other than
;   available memory.  All the computer's unused memory should be placed at the
;   disposal of the user after each statement has executed.  Whenever the
;   interpreter is expecting a number or a string, then an expression of the
;   same type can be substituted ad infinitum.


;   This is a "Concept Computer" and the ROM may not recognize the format of
;   programs saved from a conventional ZX Spectrum whether they have been saved
;   as tapes or snapshots.
;   This implementation does not try to maintain common routine addresses such
;   as $09F4. Nor are the System Variables compatible with the BASIC manual.
;   With the exception of those programs written in BASIC, third-party software
;   is unlikely to run on this platform.

;   This program is a re-arrangement of other people's code, including the
;   open standard "Sinclair Network Standard" and remains the copyright of
;   Amstrad PLC and Sinclair Research Ltd.

; 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

ORG 0000

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

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

L0000

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

```
; -----------------------
; THE OLD 'ERROR' RESTART
; -----------------------
;   Note. The ERROR restart is to be moved to L0030.
;   An instruction fetch on address $0008 may page in a peripheral ROM such as
;   the Sinclair Interface 1 or Disciple Disk Interface.  This would now be
;   disastrous as none of the routines it uses in this ROM are where they used
;   to be.  Also the network and RS232 are now controlled from this ROM.
;   The shadow ROM could also paged by an instruction fetch on address $1708.
;   Since this restart is unused, just stick a return here.  Leave room for an
;   error report but for now use location nine for release number.
;   The command PRINT PEEK 9 gives release number.

L0008

RESTART8  RET                    ;+ Disabled.

          DEFB  74               ;+ unused - but for now has release number.

          DEFB  $FF, $FF, $FF    ;+ unused
          DEFB  $FF, $FF, $FF    ;+ unused


; -----------------------------
; 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.

L0010

PRINT_A   JP    PRINT_A_2        ; Jump forward to continue at PRINT_A_2.

; ---

;;;       DEFB  $FF, $FF, $FF    ; was unused.
;;;       DEFB  $FF, $FF         ; was unused.

;   This 5-byte routine is part of the new FORMAT command and has been moved
;   here to exploit spare space. (JS)

FORMAT_T  LD    A,C              ;+ Get user-supplied TAB width
          LD    ($5BB8),A        ;+ Set it
          RET                    ;+ Return.

; -------------------------------
; 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.

L0018

GET_CHAR  LD    HL,($5B5D)       ; Fetch the address from CH_ADD.
          LD    A,(HL)           ; Use it to pick up the current character.
```

```
TEST_CHAR CALL  SKIP_OVER       ; 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.

L0020

NEXT_CHAR CALL  CH_ADD__1       ; Routine CH_ADD+1 fetches the next immediate
                                ; character.

          JR    TEST_CHAR       ; Jump back to TEST_CHAR until a valid
                                ; character is found.

; ---

; ------------------
; THE 'STOP' COMMAND
; ------------------
;   Command Syntax: STOP
;   One of the shortest and least used commands. As with 'OK' not an error.
;   This has been moved here as two bytes were unused.

STOP      RST   30H             ; ERROR_1
          DEFB  $08             ; Error Report: STOP statement

;;;       DEFB  $FF,$FF         ; was unused
          DEFB  $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.

L0028

FP_CALC   JP    CALCULATE       ; 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' operator.
;;;                             ; Note. This idea may be re-visited!

; -------------------------
; 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  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 'RST 30H' ERROR RESTART
; --------------------------
;   This restart is to be used for error handling without paging in Interface1
;   while, at the same time, allowing access to its hardware.
;   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 although not all errors
;   pass through here.

L0030

ERROR_1   LD    HL,($5B5D)      ;+ Fetch the character address from CH_ADD.
          LD    ($5B5F),HL      ;+ Copy it to the error pointer X_PTR.

          JR    ERROR_2         ;+ Forward to continue at ERROR_2.

; -------------------------------
; 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.

L0038                           ; Note Interrupts are automatically disabled.

MASK_INT  PUSH  AF              ; Save the registers that will be used.
          PUSH  HL              ;

;;;          LD    HL,($5B78)   ; Fetch the first two bytes at FRAMES1.
;;;          INC   HL           ; Increment lowest two bytes of counter.
;;;          LD    ($5B78),HL   ; Place back in FRAMES1.
;;;          LD    A,H          ; Test if the result was zero.
;;;          OR    L            ;
;;;          JR    NZ,KEY_INT   ; Forward, if not, to KEY_INT
;;;
;;;          INC   (IY+$40)     ; otherwise increment FRAMES3 the third byte.

;   Note. the above code has been replaced with this neater and shorter
;   sequence which also avoids using the IY register.

          LD    HL,$5B78        ;+ Address FRAMES
          INC   (HL)            ;+ Increment low byte of counter.
          JR    NZ,KEY_INT      ;+ Forward, if not back to zero, to KEY_INT.

          INC   L               ;+ Increment address using 4 clock cycles.
          INC   (HL)            ;+ Increment middle counter.
          JR    NZ,KEY_INT      ;+ Forward, if not back to zero, to KEY_INT.

          INC   L               ;+ All the FRAMES addresses have same high byte.
          INC   (HL)            ;+ Increment last counter.

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

```
KEY_INT   PUSH  BC               ; Save the other main registers.
          PUSH  DE               ;

          CALL  KEYBOARD         ; Routine KEYBOARD executes a stage in the
                                 ; process of reading a key-press.
                                 ; Only registers HL, DE, BC and AF can be used.

          POP   DE               ; Restore all four registers.
          POP   BC               ;

          POP   HL               ;
          POP   AF               ;

          EI                     ; Enable Interrupts.
          RET                    ; Return.

; ---------------------
; THE 'ERROR_2' ROUTINE
; ---------------------
;   A continuation of the code at ERROR_1.
;   The error code is stored and, after clearing down the calculator stack, an
;   indirect jump is made to the Error Stack Pointer to handle the error.

ERROR_2   POP   HL               ; Drop the return address - the location after
                                 ; the error restart.
          LD    L,(HL)           ; Fetch the error code that follows.

;   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   LD    (IY+$00),L       ; Store it in the system variable ERR_NR.
          LD    SP,($5B3D)       ; 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    SET_STK          ; Jump to SET_STK to clear the calculator
                                 ; stack and reset MEM to usual place in the
                                 ; systems variables area and then indirectly to
                                 ; one of the addresses above.

; -----
; SPARE
; -----

          DEFB  $FF, $FF, $FF,  ;+  Spare
          DEFB  $FF, $FF, $FF,  ;+
          DEFB  $FF, $FF, $FF,  ;+


L0066

; ------------------------------------
; THE 'NON-MASKABLE INTERRUPT' ROUTINE
; ------------------------------------
;   There was no NMI switch on the standard Spectrum.
;   There was however a well-developed NMI routine, reproduced here with one
;   major difference.  On the original Spectrum the branch to the address held
;   in the NMIADD System Variables was taken if the address was zero and not,
;   as expected, if the address was non-zero.
;
;   Sinclair Research said that, since they had never advertised the NMI, they
;   had no plans to fix the error "until the opportunity arose".  In fact, the
;   location NMIADD was later used by Interface 1 for other purposes.
;   On later Amstrad Spectrums, and the Brazilian Spectrum, the logic of this
;   routine was reversed but not as at first intended.
;
;   The original functionality is resurrected in full here.  The clue is the
;   rather clumsy initialization of CHARS in the code at RAM_SET .  The
;   NMIADD System variable now holds the address NMI_PTR by default and the
;   code there provides for a Warm Reset which re-initializes the system
;   without losing the BASIC program.
;
;   In all probability the NMI button would have been on the advertized
;   RS232/Network board.
;
;   Software houses who didn't want their programs broken into could presumably
;   set NMIADD to zero to defeat hackers.

NMI       PUSH  AF                ; Save the
          PUSH  HL                ; registers.
          LD    HL,($5BB0)        ; Fetch the system variable NMIADD.
          LD    A,H               ; Test address
          OR    L                 ; for zero.

;;;       JR    NZ,NMI_2          ;- Skip to NO_NMI if both bytes default N Z!

          JR    Z,NMI_2           ;+ Skip to NO_NMI if both bytes default ZERO.

          JP    (HL)              ; else jump to routine.

NMI_2     POP   HL                ; Restore the
          POP   AF                ; registers.

NMI_END   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 LD    HL,($5B5D)        ; fetch address from CH_ADD.

TEMP_PTR1 INC   HL                ; increase the character address by one.

TEMP_PTR2
          LD    A,(HL)            ; load character to A from HL.

TEMP_PTR3 LD    ($5B5D),HL        ; update CH_ADD with character address.

          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 CP    $21             ; test if higher than space.
          RET   NC              ; return with carry clear if higher.

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

                                ; 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, if so, with carry set.

                                ; 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,SKIPS         ; forward to SKIPS with ink, paper, flash,
                                ; bright, inverse or over controls.
                                ; Note. the high byte of tab is for RS232 only.

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

SKIPS     SCF                   ; set the carry flag

          JR    TEMP_PTR3       ;+ back to similar code above.

;;;       LD    ($5B5D),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 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 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


;  The 26 unshifted extended mode keys for the alphabetic characters.
;  The green keywords on the original keyboard.
E_UNSHIFT 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


;  The 26 shifted extended mode keys for the alphabetic characters.
;  The red keywords below keys on the original keyboard.
EXT_SHIFT 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


;   The ten control codes assigned to the top line of digits when the shift
;   key is pressed.
CTL_CODES 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


;   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.
SYM_CODES 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              ; :

;  The ten keywords assigned to the digits in extended mode.
;  The remaining red keywords below the keys.
E_DIGITS  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  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  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,KEY_DONE      ; 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 INC   D               ; now test the key buffer
          RET   NZ              ; if we have collected 2 keys already
                                ; then too many so quit.

KEY_BITS  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,KEY_BITS     ; back, if not pressed, to KEY_BITS
                                ; 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,KEY_3KEYS    ; back to KEY_3KEYS if there were more
                                ; set bits - H was not yet zero.

KEY_DONE  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,KEY_LINE      ; 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  CALL  KEY_SCAN        ; routine KEY_SCAN

          RET   NZ              ; return if invalid combinations

;   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,$5B00        ; point to KSTATE_0

K_ST_LOOP BIT   7,(HL)          ; is it free ?  (i.e. $FF)
          JR    NZ,K_CH_SET     ; forward, if so, to K_CH_SET

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

          JR    NZ,K_CH_SET     ; forward, if not at end of count, to K_CH_SET

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

K_CH_SET  LD    A,L             ; make a copy of the low address byte.

;;;       LD    HL,$5B04        ;- point to KSTATE_4 (Note. ld l,$04 would do)

          LD    L,$04           ;+ point low order byte to KSTATE_4

          CP    L               ; have both sets been considered ?
          JR    NZ,K_ST_LOOP    ; back to K_ST_LOOP to consider this 2nd set

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

          CALL  K_TEST          ; routine K_TEST to get main key in A

          RET   NC              ; return if just a single shift

          LD    HL,$5B00        ; point to KSTATE_0
          CP    (HL)            ; does the main key code match ?
          JR    Z,K_REPEAT      ; forward, if so, to K_REPEAT

;   If not consider the second key map for a repeat.

;;;       EX    DE,HL           ; save KSTATE_0 in DE
;;;       LD    HL,$5B04        ; point to KSTATE_4

          LD    L,$04           ;+ point to KSTATE_4
          CP    (HL)            ; does the main key code match ?
          JR    Z,K_REPEAT      ; forward, if so, to K_REPEAT

;   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,K_NEW        ; forward, if so, to K_NEW

;;;       EX    DE,HL           ; bring back KSTATE_0

          LD    L,$00           ;+ 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     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,($5B09)       ; 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.

          LD    L,$41           ;+ Avoid IY usage
          LD    C,(HL)          ;+ Load C register with system variable MODE.
          LD    L,$3B           ;+
          LD    D,(HL)          ;+ Load D register with system variable FLAGS.

          CALL  K_DECODE        ; routine K_DECODE

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

K_END     LD    ($5B08),A       ; update LASTK system variable.

;;;       SET   5,(IY+$01)      ;- update FLAGS  - signal a new key.

          LD    L,$3B           ;+ HL now addresses FLAGS
          SET   5,(HL)          ;+ signal 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.
;   credit: Chris Thornton 1983.

K_REPEAT  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,($5B0A)       ; 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)

          CP    $A5             ;+ Is repeat a keyword ?
          RET   NC              ;+ Ignore if a keyword.

          JR    K_END           ; Back, to accept key, at K_END

; ---------------------
; THE 'KEY_TEST' ROUTINE
; ---------------------
;   This is also called from s-inkey$
;   Begin by testing for a shift with no other.

K_TEST    LD    B,D             ; Load most significant key to B - will be $FF
                                ; if not shift.
          LD    D,$00           ; 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,K_MAIN       ; forward, if not, to K_MAIN

;   but we could have just symbol shift and no other

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


K_MAIN    LD    HL,MAIN_KEYS    ; 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
; ---------------------------------
;   This is also called from s-inkey$

K_DECODE  LD    A,E             ; pick up the stored main key

K_DECODE2 CP    $3A             ; an arbitrary point between digits and letters
          JR    C,K_DIGIT       ; forward to K_DIGIT with digits, space, enter.

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

          JP    M,K_KLC_LET     ; to K_KLC_LET if was zero

          JR    Z,K_E_LET       ; to K_E_LET if was 1 for extended letters.

;   Proceed with graphic codes.
;   Note. should not augment the keycode if code > 'U' ($55).
;   (s-inkey$ never gets into graphics mode.)

          CP    'V'             ;+ compare with non graphic keys
          JR    C,ADDIT         ;+ skip forward if this key has a UDG.

          XOR   A               ;+ set key value to zero.
          RET                   ;+ return with 'no key'.

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

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

K_E_LET   LD    HL,E_UNSHIFT-$41; base address of E_UNSHIFT.

          INC   B                ; test B is it empty i.e. not a shift.

          JR    Z,K_LOOK_UP      ; forward, if neither shift, to K_LOOK_UP

          LD    HL,EXT_SHIFT-$41; Address: EXT_SHIFT base

K_LOOK_UP 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 LD    HL,SYM_CODES-$41; prepare base of sym-codes
          BIT   0,B              ; shift=$27 sym-shift=$18
          JR    Z,K_LOOK_UP      ; back to K_LOOK_UP with symbol-shift

          BIT   3,D              ; test FLAGS is it 'K' mode (from OUT_CURS)
          JR    Z,K_TOKENS       ; skip, if so, to K_TOKENS

;;;       BIT   3,(IY+$30)       ;- test FLAGS2 - consider CAPS LOCK ?

          LD    HL,$5B6A         ;+ Address sysvar FLAGS2 using HL not IY
          BIT   3,(HL)           ;+ test FLAGS2 - consider CAPS LOCK ?

          RET   NZ               ; return, if so, with main code.

          INC   B                ; is shift being pressed ?

          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  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   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,K_KLC_DGT      ; jump to K_KLC_DGT if was 0.

          JR    NZ,K_GRA_DGT     ; forward to K_GRA_DGT if mode was 2.
```

```
;   continue with extended digits 0-9.

          LD    HL,E_DIGITS-$30 ; base of E_DIGITS
          BIT   5,B             ; test - shift=$27 sym-shift=$18
          JR    Z,K_LOOK_UP     ; back to K_LOOK_UP if sym-shift

          CP    $38             ; is character '8' ?
          JR    NC,K_8_and_9    ; 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

; ---

K_8_and_9 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 to give 0 and 1
                                ; flash codes.
          RET                   ; Return.

; ---

;   graphics mode with digits

K_GRA_DGT LD    HL,CTL_CODES-$30; base address of CTL_CODES

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

          CP    $30             ; is key '0' ?
          JR    Z,K_LOOK_UP     ; 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 accumulator bits -gives range $88-$8F.
          RET                   ; return.

; ---

;   now digits in 'KLC' mode

K_KLC_DGT INC   B               ; return with digit codes if neither
          RET   Z               ; shift key pressed.

          BIT   5,B             ; test for caps shift.

          LD    HL,CTL_CODES-$30; prepare base of table CTL_CODES.

          JR    NZ,K_LOOK_UP    ; 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,K_at_CHAR     ; forward, if so, to K_@_CHAR

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

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

; ---

K_at_CHAR 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.


; -----------------------
; THE 'BEEPER' SUBROUTINE
; -----------------------
; 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    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,BE_IX_p_3      ; 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,($5B48)         ; 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_p_3 NOP                     ;(4) optionally executed NOPs for small
                                  ;       adjustments to tone period
BE_IX_p_2 NOP                     ;(4)
```

```
BE_IX_p_1 NOP                   ;(4)

BE_IX_p_0 INC   B               ;(4)
          INC   C               ;(4)


BE_H_L_LP DEC   C               ;(4)    timing loop for duration of
          JR    NZ,BE_H_L_LP    ;(12/7)  high or low pulse of waveform

          LD    C,$3F           ;(7)
          DEC   B               ;(4)
          JP    NZ,BE_H_L_LP    ;(10)   JUMP 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,BE_AGAIN     ;(2/7)  to BE_AGAIN

          LD    A,D             ;(4)    one cycle of waveform has completed
          OR    E               ;(4)      (low->low). if cycle countdown = 0
          JR    Z,BE_END        ;(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  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    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      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,$5B92        ; 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,REPORT_B     ; to REPORT_B
                                ; 'Integer out of range'

          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,REPORT_B     ; if no, error REPORT_B
                                ; 'Integer out of range'

          INC   HL              ; if -128<=p<=127, MSB will be 0/FF if B is
                                ; pos/neg
          CP    (HL)            ; verify this
          JR    NZ,REPORT_B     ; if no, error REPORT_B
                                ; 'Integer out of range'

;   Now we know -128<=p<=127

          LD    A,B             ; A = pitch + 60
          ADD   A,$3C           ; if -60<=pitch<=67,
          JP    P,BE_I_OK       ;   goto BE_I_OK

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

                                ; and A=pitch+60 -> 0<=A<=187

BE_I_OK   LD    B,$FA           ; 6 octaves below middle C

BE_OCTAVE INC   B               ; increment octave
          SUB   $0C             ; 12 semitones = one octave
          JR    NC,BE_OCTAVE    ; 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,semi_tone    ; Address: semi-tone
          CALL  LOC_MEM         ; routine LOC_MEM
                                ;   HL = 5*A + $046E
          CALL  STACK_NUM       ; 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  FIND_INT1       ; routine FIND_INT1 ; FP duration to A
          CP    $0B             ; if dur > 10 seconds,
          JR    NC,REPORT_B     ;   goto REPORT_B
                                ; 'Integer out of range'

          ;;;   following calculation finds the tone period for HL and the cycle
count
          ;;;   DE expected in the BEEPER subroutine.  From the example in the
BEEPER comments,
          ;;;
          ;;;    ((fCPU / f) - 236) / 8 = fCPU/8/f - 236/8 = 437500/f -29.5
          ;;;    duration * frequency - 1
          ;;;
          ;;;    the different constant (30.125) used in the calculation of HL
          ;;;   w.  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  FIND_INT2       ; routine FIND_INT2
          PUSH  BC              ;   BC = tone_period(HL)
          CALL  FIND_INT2       ; 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    BEEPER          ; jump back to BEEPER

; ---

REPORT_B RST   30H             ; 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 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


;****************************************
;** 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  SCANNING    ; routine SCANNING to evaluate expression.
;;;      LD    A,($5B3B)       ; fetch system variable FLAGS.
;;;      ADD   A,A             ; test bit 7 - syntax, bit 6 - result type.
;;;      JP    M,Report_C      ; 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  STK_FETCH       ; 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
;

; -----------------------------------------
; THE NEW 'STACK TO LINE COLUMN' SUBROUTINE
; -----------------------------------------
;   This new subroutine is used by S_ATTR and S_SCRNS essentially to call the
;   routine below but, in addition, it produces a runtime error if the column
;   is greater than 31 or the line is greater than 23.
;   Both parameters must be positive as specified by the BASIC manual.

STK_TO_LC CALL  BC_POSTVE       ;

          LD    A,B             ;
          CP    $17             ;
          JR    NC,REPORT_B     ;

          LD    A,C             ;
          CP    $1F             ;

          JR    NC,REPORT_B     ;

          RET                   ;

          DEFB  0,0,0,0         ; ballast 1

; ----------------------------
; THE 'SAVE BYTES' SUBROUTINE
; ----------------------------
;   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 -
;   DE holds the length of data.
;   IX points to the start.
;   The accumulator is set to $00 for a header, $FF for data.

TAG1
L04C2:

SA_BYTES  LD    HL,SA_LD_RET    ; address: SA/LD_RET
          PUSH  HL              ; is pushed as common exit route.

          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,SA_FLAG       ; 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   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.

;   Note. the next location is trapped by emulators, see Z80.doc, in order to
;   save bytes to a real tape recorder. The address should be $04D8
;   However saving on emulators is not supported.

TAG2
L04D8:

SA_LEADER DJNZ  SA_LEADER       ; 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,SA_LEADER    ; 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,SA_LEADER     ; 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 DJNZ  SA_SYNC_1       ; self loop to SA-SYNC-1

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

SA_SYNC_2 DJNZ  SA_SYNC_2       ; 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    SA_START        ; 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   LD    A,D             ; fetch high byte
          OR    E               ; test against low byte.
          JR    Z,SA_PARITY     ; forward to SA-PARITY if zero.

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

SA_LOOP_P LD    A,H             ; fetch parity byte.
          XOR   L               ; exclusive or with new byte.

; -> the mid entry point of loop.

SA_START  LD    H,A             ; put parity byte in H.
          LD    A,$01           ; prepare blue, mic=on.
          SCF                   ; set carry flag ready to rotate in.
          JP    SA_8_BITS       ; JUMP forward to SA-8-BITS            -8->

; ---

SA_PARITY LD    L,H             ; transfer the running parity byte to L and
          JR    SA_LOOP_P       ; 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  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  DJNZ  SA_BIT_1        ; self loop for delay to SA-BIT-1

          JR    NC,SA_OUT       ; 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    DJNZ  SA_SET          ; self loop to SA-SET
                                ; (roughly an extra 66*13 clock cycles)

SA_OUT    OUT   ($FE),A         ; blue and mic on OR  yellow and mic off.
```

```
          LD    B,$3E           ; set up delay
          JR    NZ,SA_BIT_2     ; 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 RL    L               ; rotate left through carry
                                ; C<76543210<C
          JP    NZ,SA_BIT_1     ; 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,SA_LOOP      ; JUMP to SA-LOOP if more bytes.

          LD    B,$3B           ; a final delay.

SA_DELAY  DJNZ  SA_DELAY        ; 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 PUSH  AF              ; preserve accumulator throughout.

;;;       LD    A,($5B48)       ; 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.

          CALL  BORD_REST       ;+ Use new routine to restore 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,SA_LD_END    ; forward, if not, to SA/LD-END


REPORT_Da RST   30H            ; ERROR-1
          DEFB  $0C            ; Error Report: BREAK - CONT repeats

; ---

SA_LD_END POP   AF             ; restore the accumulator.
          RET                  ; return.


          DEFB  0,0,0,0,0,0,0,0 ; ballast 2


; ---------------------------
; THE 'LOAD BYTES' SUBROUTINE
; ---------------------------
;   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 to be verified only.

TAG3
L0556:

LD_BYTES  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,SA_LD_RET   ; 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.

;   Note. the next locations is trapped by emulators, see Z80.doc in order to
;   load bytes from a tape recorder. No longer supported. Was L056A

TAG4
L056A:    CP    A              ; set the zero flag.

;

LD_BREAK  RET   NZ             ; return if at any time space is pressed.

LD_START  CALL  LD_EDGE_1      ; routine LD-EDGE-1
          JR    NC,LD_BREAK    ; 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   DJNZ  LD_WAIT         ; self loop to LD-WAIT (for 256 times)

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

;   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  LD_EDGE_2       ; routine LD-EDGE-2
          JR    NC,LD_BREAK     ; back, if no edges at all, to LD-BREAK

LD_LEADER LD    B,$9C           ; set timing value.
          CALL  LD_EDGE_2       ; routine LD-EDGE-2
          JR    NC,LD_BREAK     ; back, if time-out, to LD-BREAK

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

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

;   Note. H is zero again.
;   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   LD    B,$C9           ; initial timing value in B.
          CALL  LD_EDGE_1       ; routine LD-EDGE-1
          JR    NC,LD_BREAK     ; back, with time-out, to LD-BREAK

          LD    A,B             ; fetch augmented timing value from B.
          CP    $D4             ; compare
          JR    NC,LD_SYNC      ; back, if gap too big, to LD-SYNC
                                ; it 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  LD_EDGE_1       ; 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    LD_MARKER       ; 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   EX    AF,AF'          ; restore entry flags and type in A.
          JR    NZ,LD_FLAG      ; forward to LD-FLAG if awaiting initial flag
                                ; which is to be discarded.

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

          LD    (IX+$00),L      ; place loaded byte at memory location.

          JR    LD_NEXT         ; forward to LD-NEXT

; ---

LD_FLAG   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 expected data type matches first byte received.

          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    LD_DEC          ; forward to LD-DEC.
                                ; but why not to location after ?
                                ; Timing.

; ---

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

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

;   Note. the report 'Verification has failed' could be added.

LD_NEXT   INC   IX              ; Increment the byte pointer.

LD_DEC    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 LD    L,$01           ; initialize as %00000001

LD_8_BITS CALL  LD_EDGE_2       ; 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,LD_8_BITS     ; JUMP back to LD-8-BITS

;   when the carry flag is set, then the marker bit has been passed out and
;   the received 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,LD_LOOP       ; back, while there are more, to LD-LOOP


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

          LD    A,H              ; fetch the adjusted parity byte.
          CP    $01              ; set carry if zero.
          RET                    ; return
                                 ; If 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 CALL  LD_EDGE_1        ; 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 LD    A,$16            ; a delay value of twenty two.

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

          AND    A               ; clear carry.

LD_SAMPLE 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,LD_SAMPLE      ; 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.

; -------------------------------------------
; THE 'SAVE, LOAD, VERIFY AND MERGE' COMMAND
; -------------------------------------------
;   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 original arrangement) gives a value of
;   $00 - SAVE
;   $01 - LOAD
;   $02 - VERIFY
;   $03 - MERGE
;   Note. as the Syntax table is in ROM then bit 7 of T_ADDR_hi must be reset
;   This bit can be used to indicate a non-tape operation.
;   As with all commands, the address STMT-RET is on the stack.

SAVE_ETC  POP   AF               ; discard the address STMT-RET.

;   Now reduce the low byte of the Syntax table entry to give command.

          LD    HL,$5B74         ; Address T_ADDR
          LD    A,(HL)           ; fetch value.
          SUB   P_SAVE +1 % 256 ; subtract the known offset.
          LD    (HL),A           ; and put back for future reference.


;;;       LD    A,($5B74)        ; fetch the low order address byte of T_ADDR.
;;;       SUB   P_SAVE +1 % 256 ; subtract the known offset.
;;;       LD    ($5B74),A        ; and put back for future reference.
;;;       CALL  SYNTAX_Z         ; checking syntax
;;;       JR    Z,SA_STRM        ;

          LD    A,$FD            ; select system channel 'K'
          CALL  CHN_O_SYN        ; and set as a default for tape message.

;;;       CALL  CHAN_SLCT        ; routine CHAN-OPEN

SA_STRM   CALL  STR_ALTER        ;+ Allow for SAVE #8;

          JR    C,SA_EXP         ;+ forward if no stream specified.

;   If a stream has been specified then check for a separator and set bit
;   of T_ADDR_hi to show Tape is not being used as medium.
;   e.g. SAVE #7,"marsupials"   LOAD #15; "" SCREEN$

          CALL  CLASS_0C         ;+ check for a separator
          SET   7,(IY+$3B)      ;+ flag extended command by setting T_ADDR_hi

SA_EXP    CALL  EXPT_EXP        ; routine EXPT-EXP checks that a CLASS_0A
                                ; string expression follows and stacks the
                                ; parameters in run-time.

          CALL  SYNTAX_Z        ; routine SYNTAX-Z
          JR    Z,SA_DATA       ; forward, if checking syntax, to SA-DATA

;   In runtime create the workspace which is addressed by IX register.

          LD    BC,$0011        ; presume seventeen bytes for a SAVE header.

          LD    A,($5B74)       ; fetch command from T_ADDR_lo.
          AND   A               ; test for zero, the SAVE command.

          JR    Z,SA_SPACE      ; forward, if so, to SA-SPACE

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

SA_SPACE  CALL  BC_SPACES       ; BC_SPACES creates 17/34 bytes in workspace.

          PUSH  DE              ; transfer the start of the new space to the
          POP   IX              ; 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  LD    (DE),A          ; set workspace location to space.
          INC   DE              ; next location.
          DJNZ  SA_BLANK        ; 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 a null string.

;   Now have $FF $20 $20...

          CALL  STK_FETCH       ; 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 the true length.
          JR    NC,SA_NAME      ; forward, if length 1 - 10 to SA-NAME

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

          LD    A,($5B74)       ; fetch command from T_ADDR.
          AND   A               ; test for zero, the SAVE command.
;;;       JR    NZ,SA_NULL      ; forward, if not SAVE, to SA-NULL

          JP    Z,REPORT_F      ; forward, if command is SAVE, to report
                                ; 'Invalid file name'

;   This could be a null filename or one greater than ten characters in length
;   neither of which is acceptable for the SAVE command.
;   The first ten characters of any other command parameter are acceptable.

;;; REPORT_Fa RST   30H         ; ERROR-1
;;;           DEFB  $0E         ; Error Report: Invalid file name

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

SA_NULL   LD    A,B             ; test length of filename
          OR    C               ; for zero.
          JR    Z,SA_DATA       ; forward, if zero, to SA-DATA
                                ; using $FF 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   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   RST   18H             ; GET-CHAR
          CP    $E4             ; is character after filename the token 'DATA' ?
          JR    NZ,SA_SCREEN    ; forward, if not, to SA_SCREEN
                                ; to consider SCREEN$

;   continue to consider DATA.

          LD    A,($5B74)       ; fetch command from T_ADDR
          CP    $03             ; is it 'VERIFY' ?

;   VERIFY "d" DATA is not allowed.

          JR    Z,REPORT_Ca     ; forward, if so, to REPORT-Ca.
                                ; 'Nonsense in BASIC'

;   continue with SAVE, LOAD, MERGE of DATA.

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

;   This is the runtime path only.

          LD    HL,$0000        ; set destination to zero as not fixed.
          LD    A,($5B74)       ; fetch command from T_ADDR
          DEC   A               ; test for 1 - LOAD
          JR    Z,SA_V_NEW      ; forward, with LOAD DATA, to SA-V-NEW
                                ; to load a new array.

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

```
REPORT_2a RST   30H             ; ERROR-1
          DEFB  $01             ; Error Report: Variable not found

;   continue with SAVE and LOAD of DATA

SA_V_OLD  JR    NZ,REPORT_Ca    ; forward, if not an array, to REPORT_Ca
                                ; 'Nonsense in BASIC'

          CALL  SYNTAX_Z        ; routine SYNTAX-Z
          JR    Z,SA_DATA_1     ; forward, if checking syntax, to SA-DATA-1

;   In runtime exclude a simple string by examining the VARS letter.
;   Note. the standard ROM allows these to be saved but errors when they are
;   subsequently loaded.
;   credit: Dr. Ian Logan in The Complete Spectrum ROM Disassembly.
;   solution: also by Dr. Ian Logan, in the Interface 1 ROM.

          BIT   7,(HL)          ;+ test VARS letter - is it a simple string ?
          JR    Z,REPORT_Ca     ;+ back, if so, to REPORT_Ca

;   Now transfer the array's details to the tape descriptor.

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

;   The two runtime paths converge here.  There is no syntax path error.

SA_V_NEW  LD    (IX+$0E),C      ; place the character array letter, formed
                                ; earlier, in the header.

          LD    A,$01           ; default the array type to numeric.
          BIT   6,C             ; test the result from the LOOK-VARS routine.
          JR    Z,SA_V_TYPE     ; forward, if numeric, to SA-V-TYPE

          INC   A               ; set type to 2 - a string array.

SA_V_TYPE LD    (IX+$00),A      ; place type 0, 1 or 2 in descriptor.

;   The syntax path rejoins here.

SA_DATA_1 EX    DE,HL           ; save var pointer in DE

;   Note. LOOK_VARS left CH_ADD pointing at '(' in, say, SAVE "name" DATA a().

          RST   20H             ; NEXT-CHAR

;;;       CP    $29             ; is character ')' ?
;;;       JR    NZ,SA_V_OLD     ; back, if not, to SA-V-OLD
;;;       RST   20H             ; NEXT-CHAR advances character address.

          CALL  RBRKT_NXT       ;+ check for right hand bracket and advances.

          CALL  CHECK_END       ; routine CHECK-END errors if not at end of
                                ; the statement.

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

; ---
```

```
;
; ---

TST_COM_0 XOR   A                ; default comparison
TST_COM   CP    (IY+$3A)         ; compare A to T_ADDR_lo
          RET   NZ               ; return if not.

REPORT_Ca RST   30H              ; ERROR-1
          DEFB  $0B              ; 'Nonsense in BASIC'

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

SA_SCREEN CP    $AA              ; is character the token 'SCREEN$' ?
          JR    NZ,SA_CODE       ; forward, if not, to SA_CODE

;;;       LD    A,($5B74)        ; fetch command from T_ADDR_lo
;;;       CP    $03              ; is it 'MERGE' ?
;;;       JR    NZ,SA_SCR_OK     ; skip forward, if not, to SA_SCR_OK
;;;       RST   30H              ; ERROR-1
;;;       DEFB  $0B              ; 'Nonsense in BASIC'

          LD    A,$03            ;+ Produce an error
          CALL  TST_COM          ;+ if command is 'MERGE'

; ---

;   continue with SAVE/LOAD/VERIFY SCREEN$.

SA_SCR_OK RST   20H              ; NEXT-CHAR advances past command
          CALL  CHECK_END        ; routine CHECK-END errors if not at end of
                                 ; statement.

;   continue in runtime.

          LD    HL,$4000         ;+ set start to display file start.

;;;       LD    (IX+$0B),$00     ; set descriptor length
          LD    (IX+$0B),L       ;+ 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    SA_TYPE_3        ; forward to SA-TYPE-3

; ---

;   the branch was here to consider CODE.

SA_CODE   CP    $AF              ; is character the token 'CODE' ?
          JR    NZ,SA_LINE       ; forward, if not, to SA_LINE
                                 ; to consider an auto-started BASIC program.

;;;       LD    A,($5B74)        ; fetch command from T_ADDR
;;;       CP    $03              ; is it MERGE ?
;;;       JR    Z,REPORT_Ca      ; back, if so, to REPORT-Ca.

          LD    A,$03            ;+ Produce an error
          CALL  TST_COM          ;+ if command is 'MERGE'


          RST   20H              ; NEXT-CHAR advances character address.
          CALL  PR_ST_END        ; routine PR-ST-END checks if a carriage
                                 ; return or ':' follows.
          JR    NZ,SA_CODE_1    ; forward, if there are parameters, to SA-CODE-1

;;;       LD    A,($5B74)       ; else fetch the command from T_ADDR.
;;;       AND   A               ; test for zero - SAVE without a specification.
;;;       JR    Z,REPORT_Ca     ; back, if so, to REPORT-Ca.

          CALL  TST_COM_0       ;+ Test that command is not zero - SAVE

;   For LOAD and VERIFY put a zero on the stack to signal use the address that
;   the code was saved from.

          CALL  USE_ZERO        ; routine USE-ZERO stacks a zero in runtime.
          JR    SA_CODE_2       ; forward to SA-CODE-2

; ---

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

SA_CODE_1 CALL  EXPT_1NUM       ; routine EXPT-1NUM checks for numeric
                                ; expression and stacks it in run-time.

          RST   18H             ; GET-CHAR was the last instruction.
          CP    $2C             ; does a comma follow ?
          JR    Z,SA_CODE_3     ; forward, if so, to SA-CODE-3

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

;;;       LD    A,($5B74)       ; fetch command from T_ADDR.
;;;       AND   A               ; is the command SAVE which requires length ?
;;;       JR    Z,REPORT_Ca     ; back, if so, to REPORT-Ca

          CALL  TST_COM_0       ;+ Test that command is not zero - SAVE

;   the command 'LOAD CODE' may rejoin here with zero handled as start.

SA_CODE_2 CALL  USE_ZERO        ; routine USE-ZERO stacks zero for length
                                ; if not checking syntax.
          JR    SA_CODE_4       ; forward to SA_CODE_4

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

SA_CODE_3 RST   20H             ; NEXT-CHAR advances character address.
          CALL  EXPT_1NUM       ; routine EXPT_1NUM checks for an expression
                                ; and stacks in run-time.

;   paths converge here and nothing must follow.

SA_CODE_4 CALL  CHECK_END       ; routine CHECK-END errors with extraneous
                                ; characters and quits if checking syntax.

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

          CALL  FIND_INT2       ; routine FIND-INT2 gets length.
          LD    (IX+$0B),C      ; place length
          LD    (IX+$0C),B      ; in descriptor.

          CALL  FIND_INT2       ; 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 LD    (IX+$00),$03    ; place type 3 - 'CODE' in descriptor.

RJ_SA_ALL JR    SA_ALL          ; forward to SA-ALL.

; ---
;   the branch was here with BASIC to consider an optional auto-start line
;   number e.g.
;   SAVE "some name" LINE
;   SAVE "fruitbats" LINE 200

SA_LINE   CP    $CA             ; is character the token 'LINE' ?
          JR    Z,SA_LINE_1     ; forward, if so, to SA-LINE-1

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

          CALL  CHECK_END       ; routine CHECK-END

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

;;;       LD    (IX+$0E),$80    ; place a high line number in descriptor

          LD    B,$80           ; set B to $80 as a disabling value.

          JR    SA_TYPE_0       ; forward, to save program, to SA-TYPE-0

; ---

;   the branch was here to consider auto-start.
;   Note. both the BASIC manual and the Pocket Book state that the line number
;   may be omitted

SA_LINE_1 LD    A,($5B74)       ; fetch command from T_ADDR
          AND   A               ; test for SAVE.
          JR    NZ,REPORT_Ca    ; jump forward, with anything else, to REPORT-C
                                ; 'Nonsense in BASIC'

;

          RST   20H             ; NEXT-CHAR
;;;       CALL  EXPT_1NUM       ; routine EXPT_1NUM checks for numeric
;;;                             ; expression and stacks in run-time.
          CALL  FETCH_NUM       ;+ routine FETCH_NUM checks for numeric
                                ;+ expression and stacks in run-time defaulting
                                ;+ to zero.
          CALL  CHECK_END       ; routine CHECK-END quits if syntax path.

          CALL  FIND_LINE       ; New routine FIND-LINE fetches a valid line
                                ; number expression to BC.

          LD    (IX+$0D),C      ; place the valid auto-start

SA_TYPE_0 LD    (IX+$0E),B      ; line number in the descriptor.

;   continue to save program and any variables.
;   Note. label has been moved back.

sa_type_0 LD    (IX+$00),$00    ; place type zero - program in descriptor.
          LD    HL,($5B59)      ; fetch E_LINE to HL.
          LD    DE,($5B53)      ; 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,($5B4B)      ; load HL from system variable VARS
          SBC   HL,DE           ; subtract to give program length only.
          LD    (IX+$0F),L      ; place length of program
          LD    (IX+$10),H      ; in the descriptor.
          EX    DE,HL           ; Transfer start to HL, length to DE.

SA_ALL    LD    A,($5B74)       ; fetch command from system variable T_ADDR_lo
          AND   A               ; test for zero - SAVE.

          JP    Z,SA_CONTRL     ; jump forward, with SAVE, to SA-CONTRL   ->


; -----------------------------------
; THE 'LOAD, MERGE and VERIFY' BRANCH
; -----------------------------------
;   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 PUSH  IX              ; save IX
          LD    DE,$0011        ; seventeen bytes
          XOR   A               ; reset zero flag
          SCF                   ; set carry flag to signal load the bytes.

          CALL  LD_BYTES2       ; routine LD-BYTES loads a header from tape
                                ; to second descriptor.
          POP   IX              ; restore IX.
          JR    NC,LD_LOOK_H    ; loop back, until header found, to LD-LOOK-H

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

          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 0 - 3 placed in
                                ; header by this ROM.
          JR    NZ,LD_TYPE      ; forward, with mismatch, to LD-TYPE

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


LD_TYPE   CP    $04             ; check if type is in acceptable range 0 - 3.
          JR    NC,LD_LOOK_H    ; back, with 4 and above, to LD-LOOK-H

LD_TYPE_M LD    DE,type_msgs    ; address base of last 4 tape messages

;;;       PUSH  BC              ; save BC
;;;       CALL  PO_MSG          ; routine PO-MSG outputs relevant message.

          CALL  DISP_MSG        ;+ routine DISP_MSG outputs relevant message.

;;;       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 back to 1st descriptor.

        LD    B,$0A           ; the count will be ten characters for the
                              ; filename.

;   Check if user has typed something like LOAD "".

        LD    A,(HL)          ; fetch first character of filename and test
        INC   A               ; for the value $FF.
        JR    NZ,LD_NAME      ; forward, if not the $FF wildcard, to LD-NAME

;   but if it is the wildcard, then add ten to C, which holds 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 the state set here.

        LD    A,C             ; transfer $F6 or $80 to A
        ADD   A,B             ; add $0A
        LD    C,A             ; place result, $00 or $8A, 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 INC   DE              ; Address the next input character.
        LD    A,(DE)          ; Fetch character
        CP    (HL)            ; Compare to expected
        INC   HL              ; Address next expected character
        JR    NZ,LD_CH_PR     ; Forward, with mismatch, to LD-CH-PR

        INC   C               ; Increment C - the matched character count.

LD_CH_PR
        AND   A               ;+ clear carry for 1 character.
        CALL  DISP_MSG        ;+ call directly as screen is known
;;;     RST   10H             ; PRINT-A prints the character.

        DJNZ  LD_NAME         ; loop back, for ten characters, to LD-NAME

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

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

;   else, if name matched, print a terminal carriage return.

        LD    A,$0D           ; prepare carriage return. ?????
;;;     RST   10H             ; PRINT-A outputs it.
        CALL  DISP_MSG        ;+ Call print directly.

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

        POP   HL              ; (*) restore START

        LD    A,(IX+$00)      ; Fetch the validated incoming type.
        CP    $03             ; compare with type for CODE.
        JR    Z,VR_CONTRL     ; forward, if it is CODE, to VR-CONTRL
                              ; to load or verify CODE data.

;   type is a PROGRAM or an ARRAY.

        LD    A,($5B74)       ; fetch command from T_ADDR
        DEC   A               ; was it LOAD ?
```

```
          JR    Z,LD_CONTRL      ; JUMP forward, if so, to LD-CONTRL
                                 ; to load BASIC or variables.

          CP    $02              ; was command MERGE ?

          JP    Z,ME_CONTRL      ; jump forward, if so, to ME-CONTRL

;   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_CONTRL 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,VR_CONT_1      ; forward to VR-CONT-1 if length is unspecified
                                 ; e.g. LOAD "x" CODE

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

          SBC   HL,DE            ; subtract the new length from the old length.
          JR    C,REPORT_R       ; forward to REPORT-R if the length on tape is
                                 ; larger than that specified in command.
                                 ; 'Loading error'

          JR    Z,VR_CONT_1      ; forward, if lengths match, to VR-CONT-1

;   a length on tape shorter than expected is only allowed for CODE XX

          LD    A,(IX+$00)       ; Fetch type from tape.
          CP    $03              ; Is it CODE ?
          JR    NZ,REPORT_R      ; forward, if not, to REPORT-R
                                 ; 'Loading error'

VR_CONT_1 POP   HL               ; pop the pointer to the data
          LD    A,H              ; test for zero
          OR    L                ; e.g. LOAD 'x' CODE
          JR    NZ,VR_CONT_2     ; forward, if destination given, to VR-CONT-2

          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 PUSH  HL               ; push the pointer to the start of data block.
          POP   IX               ; transfer to IX.

          LD    A,($5B74)        ; fetch the reduced command from T_ADDR

          CP    $02              ; is it VERIFY ?

;;;       SCF                    ; prepare a set carry flag
;;;       JR    NZ,VR_CONT_3     ; skip, if not, to VR-CONT-3

          JR    Z,LD_BLOCK       ;+ skip, if VERIFY, to LD_BLOCK
```

```
                                ;+ with carry clear.

;;;       AND   A               ; clear carry flag for VERIFY

; -------------------------------------------
; THE NEW 'LOAD BLOCK' WITH CARRY SET ROUTINE
; -------------------------------------------
;   This saves some bytes by consolidating the most popular conditions.

LD_BLCK_C SCF                   ;+ Set carry flag so that data is loaded.

;   Continue to use, for verification, the same routine used to LOAD data.

;;; VR_CONT_3 LD    A,$FF       ; signal data block to be loaded

; ----------------------------
; THE 'LOAD DATA BLOCK' ROUTINE
; ----------------------------
;   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  CALL  LD_BYTES    ; routine LD-BYTES

LD_BLOCK  LD    A,$FF           ;+ signal data block to be loaded, not header.

          CALL  LD_BYTES2       ; routine LD-BYTES

          RET   C               ; return if successful.

REPORT_R  RST   30H             ; ERROR-1 1a
          DEFB  $1A             ; Error Report: Loading error

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

LD_CONTRL 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 which indicates
          OR    L               ; an array - types 1 or 2.

          JR    NZ,LD_CONT_1    ; forward, if not, to LD-CONT-1

          INC   DE              ; increase array length
          INC   DE              ; for letter name
          INC   DE              ; and 16-bit length.
          EX    DE,HL           ; transfer adjusted length to HL.
          JR    LD_CONT_2       ; forward to LD-CONT-2

; ---

;   The branch was here with type PROGRAM.

LD_CONT_1 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,LD_DATA       ; to LD-DATA

LD_CONT_2 LD    DE,$0005        ; allow an 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  TEST_ROOM       ; routine TEST-ROOM fails if not enough room.

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

;   the type is a numeric or string array.

          LD    A,H             ; test the destination for zero which
          OR    L               ; indicates variable does not already exist.

          JR    Z,LD_DATA_1     ; 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 array length
          LD    B,(HL)          ; transfer to B.
          DEC   HL              ; address low byte of total array 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    ($5B5F),IX      ; save header pointer in X_PTR which is
                                ; updated by the POINTERS routine.

          CALL  RECLAIM_2       ; routine RECLAIM-2 reclaims the old variable
                                ; sliding workspace including the two headers
                                ; downwards.

          LD    IX,($5B5F)      ; reload IX from X_PTR which will have been
                                ; adjusted down by the POINTERS routine.

;;; LD_DATA_1 LD    HL,($5B59)      ; address E_LINE
;;;           DEC   HL              ; now point to the $80 variables end-marker.

LD_DATA_1 CALL  L_EL_DHL        ; instead of prev 2 lines.

          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 the letter name
          INC   BC              ; and two total length bytes.

          LD    A,(IX-$03)      ; fetch letter name from old header.

;;;       PUSH  AF              ; preserve accumulator though not corrupted.

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

TX_BLK_C  PUSH  HL              ; transfer the address

LD_BLK_R  POP   IX              ; to IX register pair.

;;;       SCF                   ; set carry flag indicating load not verify.

;;;       LD    A,$FF           ; signal data not header.

          JR    LD_BLCK_C       ;+ JUMP back to LD-BLOCK

; ---

;   The branch is here when a PROGRAM, as opposed to an ARRAY, is to be loaded.

LD_PROG   EX    DE,HL           ; transfer data destination to DE.

          CALL  L_EL_DHL        ;+ instead of next 2 lines?
;;;       LD    HL,($5B59)      ; address E_LINE
;;;       DEC   HL              ; now address variables end-marker.

          LD    ($5B5F),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  RECLAIM_1       ; routine RECLAIM-1 reclaims program and vars.
                                ; adjusting X-PTR.

          POP   BC              ; restore the new length.
          PUSH  HL              ; * save start
          PUSH  BC              ; ** and length.

          CALL  MAKE_ROOM       ; routine MAKE-ROOM creates the space.

          LD    IX,($5B5F)      ; 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    ($5B4B),HL      ; set the system variable VARS

          LD    H,(IX+$0E)      ; fetch high byte of autostart line number.

;   Note. although the line number is checked at SAVE time, this check is
;   still relevant as by default auto-start is inhibited.

          LD    A,H             ; transfer to A
          AND   $C0             ; test if greater than $3F.
          JR    NZ,LD_PROG_1    ; forward, if so, to LD-PROG-1
                                ; with no autostart.

          LD    L,(IX+$0D)      ; fetch the low byte.
          LD    ($5B42),HL      ; set system variable NEWPPC to line number
;;;       LD    (IY+$0A),$00    ; set statement NSPPC to zero.
```

```
          LD    (IY+$0A),A      ; set statement NSPPC to zero.

LD_PROG_1 POP   DE              ; ** pop the length


;;;       POP   IX              ; * and start.
;;;       SCF                   ; set carry flag
;;;       LD    A,$FF           ; signal data as opposed to a header.
;;;       JP    LD_BLCK_C       ; jump back to LD-BLOCK

          JR    LD_BLK_R        ;+ NEW relative jump back to LD-BLOCK routine
                                ;+ at the instruction POP IX

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

ME_CONTRL LD    C,(IX+$0B)      ; fetch length
          LD    B,(IX+$0C)      ; of data block on tape.
          PUSH  BC              ; save it.
          INC   BC              ; add one for the end-marker.

          CALL  BC_SPACES       ; routine 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 address of first location.

;;;       PUSH  HL              ; and transfer first location
;;;       POP   IX              ; to IX register.

;;;       SCF                   ; set carry flag to load data on tape.
;;;       LD    A,$FF           ; signal data not a header.

          CALL  TX_BLK_C        ;+ routine LD-BLOCK loads to workspace.

          POP   HL              ; restore first location in workspace to HL.
          LD    DE,($5B53)      ; 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 LD    A,(HL)          ; fetch next byte from workspace.
          AND   $C0             ; compare with $3F.

          JR    NZ,ME_VAR_LP    ; forward to ME-VAR-LP if a variable or
                                ; end-marker.

;   Continue when HL, the WORKSPACE pointer, still addresses a BASIC line
number.

ME_OLD_LP LD    A,(DE)          ; fetch high byte from PROGRAM area.
          INC   DE              ; increment the PROGRAM address.

          CP    (HL)            ; compare with line number in WORKSPACE.
          INC   HL              ; increment WORKSPACE address.

          JR    NZ,ME_OLD_L1    ; 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 low byte in WORKSPACE.

ME_OLD_L1 DEC   DE              ; point to start of
          DEC   HL              ; respective lines again.

          JR    NC,ME_NEW_L2    ; 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  NEXT_ONE        ; routine NEXT-ONE finds next line in DE.

          POP   HL              ; restore workspace pointer

          JR    ME_OLD_LP       ; 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 CALL  ME_ENTER        ; routine ME-ENTER enters the line

          JR    ME_NEW_LP       ; loop back to ME-NEW-LP.

; ---

;   the branch was here when the location in workspace held a variable.
;   New variables are easier than program lines as they are merely added at
;   the end of the VARIABLES area.

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

          PUSH  HL              ; save workspace area pointer.
          LD    HL,($5B4B)      ; load HL with VARS - start of variables area.

ME_OLD_VP LD    A,(HL)          ; fetch first byte.
          CP    $80             ; is it the VARIABLES end-marker ?
          JR    Z,ME_VAR_L2     ; 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,ME_OLD_V2     ; forward, with a match, to ME-OLD-V2
                                ; to replace.

;   else entire variables area has to be searched.

ME_OLD_V1 PUSH  BC              ; save character in C.

          CALL  NEXT_ONE        ; routine NEXT-ONE gets following variable
                                ; address in DE.

          POP   BC              ; restore character in C
          EX    DE,HL           ; transfer next address to HL.

          JR    ME_OLD_VP       ; loop back to ME-OLD-VP
```

```
; ---

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

ME_OLD_V2 AND   $E0              ; keep bits 11100000
          CP    $A0              ; compare   10100000 - a long-named variable.

          JR    NZ,ME_VAR_L1     ; 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 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,ME_OLD_V4     ; forward, with a mismatch, to ME-OLD-V4

          RLA                    ; test if it is the terminal inverted character.
          JR    NC,ME_OLD_V3     ; loop back, if more to test, to ME-OLD-V3

;   otherwise the long name matches in its entirety.

          POP   HL               ; restore pointr to first character of variable

          JR    ME_VAR_L1        ; forward to ME-VAR-L1

; ---

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

ME_OLD_V4 POP   HL               ; restore the prog/vars pointer.
          JR    ME_OLD_V1        ; back to ME-OLD-V1 to resume search.

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

ME_VAR_L1 LD    A,$FF            ; indicate a replacement.

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

ME_VAR_L2 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  ME_ENTER         ; routine ME-ENTER copies variable in.

          JR    ME_VAR_LP        ; loop back to ME-VAR-LP

; -----------------------------------------
; THE 'MERGE A LINE OR VARIABLE' SUBROUTINE
; -----------------------------------------
;   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  JR    NZ,ME_ENT_1      ; forward, for insertion only, to ME-ENT-1

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

```
          EX    AF,AF'          ; save carry - prog/var flag

          LD    ($5B5F),HL      ; preserve workspace pointer in dynamic X_PTR

          EX    DE,HL           ; transfer program dest pointer to HL.

;;;       CALL  NEXT_ONE        ; routine NEXT-ONE finds the following location
;;;                             ; in program or variables area.
;;;       CALL  RECLAIM_2       ; routine RECLAIM-2 reclaims the space between.

          CALL  NXT_1_RC2       ;+ routine combines above 2 routines.

          EX    DE,HL           ; transfer program dest pointer back to DE.

          LD    HL,($5B5F)      ; fetch adjusted workspace pointer from X_PTR

          EX    AF,AF'          ; restore carry - program/variable flag.

;   now the new line or variable is entered.

ME_ENT_1  EX    AF,AF'          ; save or re-save carry - prog/var flag.

          PUSH  DE              ; save dest pointer in prog/vars area.
          CALL  NEXT_ONE        ; routine NEXT-ONE finds next in workspace.
                                ; gets next in DE, difference in BC.
                                ; prev addr in HL

          LD    ($5B5F),HL      ; store pointer in X_PTR

          LD    HL,($5B53)      ; 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 back carry - prog/var flag.

          JR    C,ME_ENT_2      ; skip, if handling a variable, to ME-ENT-2

          CALL  MK_RM_DHL       ;+ MAKE_ROOM decrementing HL first

;;;       DEC   HL              ; address location before pointer
;;;       CALL  MAKE_ROOM       ; routine MAKE-ROOM creates room for BASIC line

          INC   HL              ; address next. (keep this one)

          JR    ME_ENT_3b       ; forward to ME-ENT-3

; ---

ME_ENT_2  CALL  MAKE_ROOM       ; routine MAKE-ROOM creates room for variable.

;;; me_ent_3  INC   HL          ; address next?

ME_ENT_3b POP   BC              ; ** pop length

          EX    DE,HL           ;+ DE now holds first new location

;   Note. HL is now used instead of DE

          POP   HL              ; * pop value for PROG which may have been
                                ; altered by POINTERS if first line.

          LD    ($5B53),HL      ; set PROG back to original value.
          LD    HL,($5B5F)      ; fetch adjusted workspace pointer from X_PTR

          PUSH  BC              ; save the length.
          PUSH  HL              ; and save the workspace pointer.
;;;       EX    DE,HL           ; make workspace pointer the 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  RECLAIM_2       ; 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.
;   For tape,
;   HL points to start of data.
;   IX points to start of descriptor.
;   For RS232 and network,
;   HL points to the start of the data
;   IX points to start of descriptor.
;   If saving to tape then channel 'K' will be open for messages.

SA_CONTRL PUSH  HL              ; save start of data.

;;;       LD    A,$FD           ; select system channel 'K'
;;;       CALL  CHAN_SLCT       ; routine CHAN-OPEN

          CALL  IN_CHAN_K       ; is tape being used ?
          JR    NZ,SA_CBN       ; skip the prompt message if not.

;;;       XOR   A               ; Clear to address table directly
          LD    DE,tape_msgs    ; Address: tape-msgs
          CALL  PO_MSG_0        ; Routine PO-MSG -
                                ; 'Start tape then press any key.'

;;;       SET   5,(IY+$02)      ; Update TV_FLAG - signal lower screen requires
;;;                             ; clearing.

          SET   3,(IY+$01)      ;+ Set 'L' key mode for prompt situation.

          CALL  WAIT_KEY        ; routine WAIT_KEY

SA_CBN    PUSH  IX              ; Save pointer to descriptor.
          LD    DE,$0011        ; There are seventeen bytes to save.
          XOR   A               ; Set A to zero - to signal a header block.

          CALL  SA_BYTES2       ; routine SA-BYTES saves block

          POP   IX              ; restore descriptor pointer.
          LD    B,$32           ; wait for a second - 50 interrupts.

SA_1_SEC  HALT                  ; wait for an interrupt
          DJNZ  SA_1_SEC        ; 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.  ( dec a )

          DEC   A               ;+ signal data bytes.

          POP   IX              ; retrieve pointer to start

;;;       JP    SA_BYTES        ; jump back to SA-BYTES

SA_BYTES2 BIT   7,(IY+$3B)      ;+ are extended streams being used. T_ADDR_hi
          JP    Z,SA_BYTES      ;+ back to tape routine if not

          LD    HL,SA_LD_RET    ; address: SA/LD_RET            Duplication.
          PUSH  HL              ; is pushed as common exit route.

; -------------------------------------------------
; THE NEW 'SAVE BYTES TO NETWORK/RS232' SUBROUTINE
; -------------------------------------------------
;   This can also, for amusement, be used to save a small program to the
;   Screen e.g. SAVE #2, "ABC"
;   DE holds the length of data.
;   IX points to the start.
;   Begin by transferring the start of data from IX to HL as the extended
;   streams will use the IX register. RST 10 preserves the main registers.

SA_BYT_NB PUSH  IX              ; Transfer start to
          POP   HL              ; the HL register.

SA_BYT_LP LD    A,D             ; Test for zero length.
          OR    E               ;
          RET   Z               ; Return if so.                          >>

          LD    A,(HL)          ; Fetch a byte to the accumulator.
          INC   HL              ; Increment address.
          DEC   DE              ; decrement byte count.

          RST   10H             ; Restart outputs a byte to current channel.

          JR    SA_BYT_LP       ; loop back to save another byte to SA_BYT_LP


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

LD_BYTES2 BIT   7,(IY+$3B)      ; Test T_ADDR_hi
          JP    Z,LD_BYTES      ; jump to tape routines

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

          EX    AF,AF'          ; preserve carry


; ---------------------------------------------------
; THE NEW 'LOAD BYTES FROM NETWORK/RS232' SUBROUTINE
; ---------------------------------------------------
;    IX points to start
;    DE holds length
;    The alternate CARRY is set if data is to be loaded.

LD_BYT_NB PUSH  IX                ; transfer the destination start address
          POP   HL                ; to the HL register pair.

LD_BYT_LP CALL  INPUT_AD          ; input a byte from the current channel


          JR    NC,LD_BYT_LP      ; repeat until byte is acceptable. XXXXXXXXXX

          EX    AF,AF'            ; fetch the carry flag.
          JR    C,LD_BYT_1        ; forward, with carry, to LOAD byte.

          EX    AF,AF'            ; preserve carry.
;;;       XOR   (HL)              ; verify against byte in memory.
          CP    (HL)              ; compare

          RET   NZ                ; return if verification failed with NC also.

          JR    LD_BYT_2          ; skip forward for next byte.

LD_BYT_1  EX    AF,AF'            ; preserve carry flag bring back new byte.
          LD    (HL),A            ; insert byte from network or RS232.

LD_BYT_2  INC   HL                ; increment memory pointer.
          DEC   DE                ; decrement the byte count.
          LD    A,D               ; Test for zero.
          OR    E                 ;
          JR    NZ,LD_BYT_LP      ; back if not zero for more.

          SCF                     ; signal success.

          RET                     ; Return.


;  ---------------------------
;  THE NEW 'DISP_MSG' ROUTINE
;  ---------------------------
;   If, on entry, carry is set then this routine prints a message without
;   disturbing the current channel.  If the carry flag is reset then the
;   single character in A is output.

DISP_MSG
          PUSH  HL                ; Preserve Main registers.
          PUSH  BC                ;
          PUSH  DE                ;

          LD    HL,($5B51)        ; fetch the current channel.
          PUSH  HL                ; and save it

          PUSH  DE                ; preserve message pointer
          PUSH  AF                ; preserve type and carry flag

          CALL  CHAN_O_FE         ; select system channel for 'S'

          POP   AF                ; bring back the type
          POP   DE                ; and the message pointer

          JR    NC,DISP_1         ; forward with no carry to output a single char

          CALL  PO_MSG            ; output message to upper screen
          LD    A,':'           ; follow the type message with ': '
          RST   10H             ;
          LD    A,' '           ;
DISP_1    RST   10H             ; else print the character

          POP   HL              ; restore channel.
          CALL  CHAN_FLAG       ; routine CHAN_FLAG updates CURCHL and flags.

          POP   DE              ; Restore main registers.
          POP   BC              ;
          POP   HL              ;

          RET                   ; Return.


; ---

;   Arrangement of the two tape cassette 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 length
;   IX-$05  IX+$0C      hi     hi    hi    hi     of datablock.
;   IX-$04  IX+$0D      Auto   -     -     Start  Various
;   IX-$03  IX+$0E      Start  a-z   a-z   addr/0 ($80 if no autostart).
;   IX-$02  IX+$0F      lo     -     -     -      Length of Program
;   IX-$01  IX+$10      hi     -     -     -      only i.e. without variables.
;
;
;   Arrangement of 9-byte Interface1 Network/RS232 header when saving loading.
;   Note. This has not been adopted by this ROM.
;
;   $5BE6   HD_00       0      1     2     3      Type.
;
;   $5BE7   HD_0B       lo     lo    lo    lo     Total length
;   $5BE8   HD_0C       hi     hi    hi    hi     of datablock
;   $5BE9   HD_0D       --     --    --    lo/00  Start
;   $5BEA   HD_0E       --     --    --    hi/00  Address.
;   $5BEB   HD_0F       lo     a-z   a-z   --     Length
;   $5BEC   HD_10       hi     --    --    --     of program.
;   $5BED   HD_11       Auto   --    --    --     Auto start line
;   $5BEE   HD_12       Start  --    --    --     number. $FFFF if none.

; ------------------------------
; THE 'CANNED CASSETTE' MESSAGES
; ------------------------------
;   The last-character-inverted Cassette messages.
;   Starts with normal initial step-over byte.
```

```
tape_msgs DEFB  $80
          DEFM  "Start tape, then press a key"

type_msgs DEFB  '.'+$80
          DEFB  $0D
          DEFM  "Progra"
          DEFB  'm'+$80
          DEFB  $0D
          DEFM  "Number arra"
          DEFB  'y'+$80
          DEFB  $0D
          DEFM  "Char arra"
          DEFB  'y'+$80
          DEFB  $0D
          DEFM  "Byte"
          DEFB  's'+$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 CALL  PO_FETCH        ; routine PO-FETCH fetches print position
                                ; to HL register pair.
          CP    $20             ; is character a space or higher ?
          JR    NC,PO_Q_ABLE    ; jump forward, if so, to PO-ABLE

          CP    $06             ; is character in range 00-05 ?
          JR    C,PO_QUEST      ; forward, if so, to PO-QUEST

          CP    $18             ; is character in range 24d - 31d ?
          JR    NC,PO_QUEST     ; forward, if so, to PO-QUEST

          LD    HL,ctlchrtab-6  ; address - 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 of routine.

          JP    PO_FETCH        ; Jump forward to PO-FETCH,
                                ; as the screen/printer position has been
                                ; disturbed, and then indirectly to the
                                ; routine on the 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 DEFB  PO_COMMA    - $ ; 06d offset  to Address: PO-COMMA
          DEFB  PO_QUEST    - $ ; 07d offset  to Address: PO-QUEST
          DEFB  PO_BACK_1   - $ ; 08d offset  to Address: PO-BACK-1
          DEFB  PO_RIGHT    - $ ; 09d offset  to Address: PO-RIGHT
          DEFB  PO_QUEST    - $ ; 10d offset  to Address: PO-QUEST
          DEFB  PO_QUEST    - $ ; 11d offset  to Address: PO-QUEST
          DEFB  PO_QUEST    - $ ; 12d offset  to Address: PO-QUEST
          DEFB  PO_ENTER    - $ ; 13d offset  to Address: PO-ENTER
          DEFB  PO_QUEST    - $ ; 14d offset  to Address: PO-QUEST
          DEFB  PO_QUEST    - $ ; 15d offset  to Address: PO-QUEST
          DEFB  PO_1_OPER   - $ ; 16d offset  to Address: PO-1-OPER
          DEFB  PO_1_OPER   - $ ; 17d offset  to Address: PO-1-OPER
          DEFB  PO_1_OPER   - $ ; 18d offset  to Address: PO-1-OPER
          DEFB  PO_1_OPER   - $ ; 19d offset  to Address: PO-1-OPER
          DEFB  PO_1_OPER   - $ ; 20d offset  to Address: PO-1-OPER
          DEFB  PO_1_OPER   - $ ; 21d offset  to Address: PO-1-OPER
          DEFB  PO_2_OPER   - $ ; 22d offset  to Address: PO-2-OPER
          DEFB  PO_2_OPER   - $ ; 23d offset  to Address: PO-2-OPER


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

PO_BACK_1 INC   C                 ; Move left one column.
          LD    A,$22             ; Value $21 is leftmost column.
          CP    C                 ; Have we passed ?
          JR    NZ,PO_BACK_3      ; Forward, if not, to PO-BACK-3
                                  ; to store the new position.

          BIT   1,(IY+$01)        ; Test FLAGS - is printer in use ?
          JR    NZ,PO_BACK_2      ; Forward, if so, to PO-BACK-2
                                  ; as it is not possible to move left.

          INC   B                 ; Move up one screen line
          LD    C,$02             ; The rightmost column position.

;;;       LD    A,$18             ; Note. This should be $19 (not $18)
;;;                               ; Credit: Dr. Frank O'Hara, 1982

          LD    A,$19             ;+ Test against the top line plus one.

          CP    B                 ; Has position moved past top of screen ?
          JR    NZ,PO_BACK_3      ; Forward, if not, to PO-BACK-3
                                  ; to store the new position.

          DEC   B                 ; else back to $18.

PO_BACK_2 LD    C,$21             ; the leftmost column position.

PO_BACK_3 JR    PO_ENTEND         ;+ Forward, indirectly, to CL-SET and PO-STORE
                                  ; to store new position in system variables.

;;;       JP    CL_SET            ; a 3-byte direct jump.

; --------------------------
; 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 LD    A,($5B91)     ; fetch P_FLAG value
;;;         PUSH  AF             ; and preserve the original value on the stack.
;;;         LD    (IY+$57),$01   ; temporarily set P_FLAG 'OVER 1'.
;;;         LD    A,$20          ; prepare a space.
;;;         CALL  PO_CHAR        ; routine PO-CHAR to print it.
;;;         POP   AF             ; restore the original P_FLAG value.
;;;         LD    ($5B91),A      ; and restore system variable P_FLAG
;;;         RET                  ; return without need to update column position.

PO_RIGHT  LD    HL,$5B91        ;+ Address system variable P_FLAG
          LD    D,(HL)          ;+ Fetch the System Variable value and
          LD    (HL),1          ;+ Set to OVER 1

          CALL  PO_SV_SP        ;+ Routine prints a space

          LD    (HL),D          ;+ and place in P_FLAG
          RET                   ;+ Return

; ------------------------------------
; THE 'PRINT CARRIAGE RETURN' ROUTINE
; ------------------------------------
;   A carriage return is 'printed' to screen or printer buffer.

PO_ENTER  BIT   1,(IY+$01)      ; test FLAGS  - is printer in use ?

          JP    NZ,COPY_BUFF    ; to COPY-BUFF if so, to flush buffer and reset
                                ; the print position.

;   Continue if writing to screen.

          LD    C,$21           ; the leftmost screen column position.

          CALL  PO_SCR          ; routine PO-SCR handles any scrolling required.

          DEC   B               ; adjust to next screen line.

PO_ENTEND JP    CL_SET          ; jump forward to CL-SET to store new position.

; ----------------------------
; THE 'PRINT COMMA' SUBROUTINE
; ----------------------------
;   The comma control character. The 32 column screen has two 16 character
;   tabstops.  The routine is only reached via the control character table.
;   If it was called from elsewhere then the call to PO-FETCH would be needed.


;;;       CALL  PO_FETCH        ; routine PO-FETCH - seems unnecessary.

PO_COMMA  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    PO_FILL         ; forward to PO-FILL

; ------------------------------------
; THE 'PRINT QUESTION MARK' SUBROUTINE
; ------------------------------------
;   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  LD    A,$3F           ; prepare the character '?'.

PO_Q_ABLE JR    PO_ABLE         ; forward to PO-ABLE.

; ---------------------------------------------
; THE 'CONTROL CHARACTERS WITH OPERANDS' ROUTINES
; ---------------------------------------------
;   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   LD    DE,PO_CONT      ; address: PO-CONT will be next output routine

PO_TV_3   LD    ($5B0F),A       ; store first operand in TVDATA-hi
          JR    PO_CHANGE       ; forward to PO-CHANGE >>

; ---

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

PO_2_OPER LD    DE,PO_TV_2      ; address: PO-TV-2 will be next output routine
          JR    PO_TV_1         ; forward to PO-TV-1

; ---

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

PO_1_OPER LD    DE,PO_CONT      ; address: PO-CONT will be next output routine

PO_TV_1   LD    ($5B0E),A       ; store control code in TVDATA-lo

PO_CHANGE LD    HL,($5B51)      ; use CURCHL to find current output channel.

PO_CH_2   LD    (HL),E          ; make it
          INC   HL              ; the supplied
          LD    (HL),D          ; address from DE.

          RET                   ; return.

; ---

PO_NORM   LD    DE,PRINT_OUT    ; prepare to make PRINT_OUT normal.
          JR    PO_CHANGE       ;

; ---

PO_CONT
;;;       LD    DE,PRINT_OUT    ; Address: PRINT-OUT
;;;       CALL  PO_CHANGE       ; routine PO-CHANGE to restore normal channel.
          CALL  PO_NORM         ;+ routine embodies above two instructions.

;   Now that all the sequence of codes have been received they can be handled.
;   The accumulator holds the final parameter and any previous codes are in
;   the system variable TVDATA.

          LD    HL,($5B0E)      ; TVDATA gives control code and possible
                                ; subsequent character
          LD    D,A             ; save current code.
          LD    A,L             ; fetch the stored control code
          CP    $16             ; was it one operand - INK to OVER ?

          JP    C,CO_TEMP_5     ; jump forward, if so, to CO-TEMP-5

;   Consider the two control codes with two operands.

          JR    NZ,PO_TAB       ; forward, if not 22 decimal, to PO-TAB (23)

;   else must have been 22 decimal - 'AT'.

          LD    B,H             ; line to H   (0-23d)
          LD    C,D             ; column to C (0-31d)
          LD    A,$1F           ; prepare the value 31d
          SUB   C               ; reverse the column number.
          JR    C,PO_AT_ERR     ; forward, if greater than 31, to PO-AT-ERR
                                ; 'Integer out of range'

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

;   Now consider the line parameter.

          BIT   1,(IY+$01)      ; test FLAGS - is printer in use ?
          JR    NZ,PO_ENTEND    ; forward, if so, ignoring line to PO-AT-SET

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

PO_AT_ERR JP    C,REPORT_Bb     ; jump, if higher than 22, to REPORT-B
                                ; 'Integer out of range'

          INC   A               ; adjust for the system range $01-$17
          LD    B,A             ; place in the line register

          INC   B               ; adjust to system range  $02-$18
          BIT   0,(IY+$02)      ; test TV_FLAG  - Lower screen in use ?
          JP    NZ,PO_SCR       ; forward, if so, to PO-SCR
                                ; to test for scrolling.

          CP    (IY+$31)        ; for upper screen, compare against DF_SZ

;;;       JP    C,REPORT_5      ; to REPORT-5 if too low
;;;                             ; 'Out of screen'
;;;       JP    CL_SET          ; to CL_SET if valid.

PO_AT_SET JR    NC,PO_ENTEND    ;+ print position is valid so exit via CL-SET

REPORT_5a RST   30H             ;+ ERROR-1
          DEFB  $04             ;+ Error Report: Out of screen

; ---

;   The branch was 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    LD    A,H             ; transfer parameter to A losing the current
                                ; contents - the high byte of the TAB parameter.

PO_FILL   CALL  PO_FETCH        ; 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 mod 32 that is 0 - 31.
          RET   Z                 ; return if result is zero.

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

PO_SPACE  CALL  PO_SV_SP          ;+ call instruction before PO_SAVE - ld a,$20

;;;       LD    A,$20             ; space character.
;;;       CALL  PO_SAVE           ; routine PO-SAVE prints the character
                                  ; using alternate set (normal output routine)

          DEC   D                 ; decrement the spaces counter.
          JR    NZ,PO_SPACE       ; back to PO-SPACE until done.

          RET                     ; Return.

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

PO_ABLE   CALL  PO_ANY            ; 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  BIT   1,(IY+$01)        ; Test FLAGS - is printer in use ?
          JR    NZ,PO_ST_PR       ; Forward, if so, to PO-ST-PR

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

;   This section deals with the upper screen.

          LD    ($5B88),BC        ; Update S_POSN - line/column upper screen
          LD    ($5B84),HL        ; Update DF_CC - upper display file address

          RET                     ; Return.

; ---

;   This section deals with the lower screen.

PO_ST_E   LD    ($5B8A),BC        ; Update SPOSNL line/column lower screen
          LD    ($5B82),BC        ; Update ECHO_E line/column input buffer
          LD    ($5B86),HL        ; Update DFCCL  lower screen memory address
          RET                     ; Return.

; ---

;   This section deals with the ZX Printer.
;   Now just update the column number $00 - $21 within the channel.

PO_ST_PR  LD    IX,($5B51)        ;+ set IX to CURCHL

          LD    (IX+$07),C        ;+ Update P_POSN column position printer
          RET                   ; Return.


; ----------------------------
; 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 is no longer used.  The output address is calculated
;   by this routine every time from the new channel variable P_POSN.
;   The output address now alters whenever a channel is reclaimed.

PO_FETCH  BIT   1,(IY+$01)      ; Test FLAGS - is printer in use ?
          JR    NZ,PO_F_PR      ; Forward, if so, to PO-F-PR

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

          LD    BC,($5B88)      ; Fetch line/column from S_POSN
          LD    HL,($5B84)      ; 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,($5B8A)      ; Fetch line/column from SPOSNL
          LD    HL,($5B86)      ; Fetch display file address from DFCCL
          RET                   ; Return.

; ---

;   This section deals with the ZX Printer.
;   The column is obtained from the location within the channel.
;   The output address HL is derived from this column number.

PO_F_PR   LD    HL,($5B51)      ;+ set HL to start of Channel from CURCHL
          LD    BC,$0007        ;+ offset to column number.
          ADD   HL,BC           ;+ add to address P_POSN
          LD    C,(HL)          ;+ Fetch column from P_POSN.
          INC   HL              ;+ Start of 256 buffer.

          LD    B,A             ;+ copy character to B.

          LD    A,$21           ;+ Reverse the column number
          SUB   C               ;+ Now $00 (left) $1F (right)
          ADD   A,L             ;+ add to low byte possibly setting carry flag.
          LD    L,A             ;+ place back in low byte.

          LD    A,B             ;+ copy character back to A

          RET   NC              ;+ return if address is correct.

          INC   H               ;+ else increase by 256 bytes.

          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
;   On entry, HL contains the output address and BC the line column or just
;   the column in the case of the ZX Printer.
```

```
PO_ANY    CP    $80               ; ASCII ?
          JR    C,PO_CHAR         ; to PO-CHAR if so.

          CP    $90               ; test if a block graphic character.
          JR    NC,PO_T_UDG       ; 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  PO_GR_1           ; routine PO-GR-1 to construct top half
                                  ; then bottom half.

          CALL  PO_FETCH          ; routine PO-FETCH re-fetches print position.

          LD    DE,$5B92          ; MEM-0 is location of 8 bytes of character

          JR    PR_ALL            ; forward to PR-ALL
                                  ; to print to screen or printer.

; ---

PO_GR_1   LD    HL,$5B92          ; address MEM-0 - a temporary buffer in
                                  ; systems variables which is normally used
                                  ; by the calculator.
          CALL  PO_GR_2           ; routine PO-GR-2 to construct top half
                                  ; and continue into routine to construct
                                  ; bottom half.

PO_GR_2   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   LD    (HL),A            ; store bit patterns in temporary buffer
          INC   HL                ; next address
          DEC   C                 ; jump back to
          JR    NZ,PO_GR_3        ; to PO-GR-3 until byte is stored 4 times

          RET                     ; return

; ---

;   Tokens and User defined graphics are now separated.

PO_T_UDG  SUB   $A5               ; subtract the 'RND' character
          JR    NC,PO_T           ; forward, if a token, to PO-T

          ADD   A,$15             ; add 21d to restore to 0 - 20
          PUSH  BC                ; save current print position
          LD    BC,($5B7B)        ; fetch UDG to address bit patterns
          JR    PO_CHAR_2         ; forward to common code at PO-CHAR-2
                                  ; to lay down a bit patterned character

; ---

;   Tokens
```

```
PO_T      CALL  PO_TOKENS       ; routine PO-TOKENS prints tokens

;;;       JP    PO_FETCH        ; an absolut jump to PO_FETCH

          JR    PO_FETCH        ;+ exit via a JUMP to PO-FETCH as this routine
                                ;+ must continue into PO-STORE.
                                ;+ A JR instruction could be used. (Done)

; ---

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

PO_CHAR   PUSH  BC              ; Preserve print position
          LD    BC,($5B36)      ; Fetch font pointer from address CHARS

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

PO_CHAR_2 EX    DE,HL           ; transfer destination address to DE

          LD    HL,$5B3B        ; point to FLAGS
          RES   0,(HL)          ; update FLAGS - allow for leading space

          CP    $20             ; is output character a space ?
          JR    NZ,PO_CHAR_3    ; skip forward, if not, to PO-CHAR-3

          SET   0,(HL)          ; update FLAGS - signal no leading space.

PO_CHAR_3 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              ; retrieve the source address from CHARS or UDG.

          EX    DE,HL           ; transfer the character bitmap 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 the mosaic characters.
;   HL = screen or printer destination
;   DE = character bitmap source
;   BC = line/column

PR_ALL    LD    A,C             ; transfer the column to A
          DEC   A               ; move to the right

          LD    A,$21           ; pre-load with leftmost position
          PUSH  DE              ;+ Save character source before any branching.
          JR    NZ,PR_ALL_1     ; forward, if not zero, to PR-ALL-1

;   If zero then move down a line, but B is of no significance if printer
;   is in use

          DEC   B               ; down one line
          LD    C,A             ; load C with $21

          BIT   1,(IY+$01)      ; test FLAGS  - is printer in use
```

```
;;;       JR    Z,PR_ALL_1       ; forward, if not, to PR-ALL-1

;   This is the printer-only path but we can trickle through.

;;;       PUSH  DE               ; save source address

;;;       CALL  COPY_BUFF        ; Routine COPY-BUFF outputs line to printer
          CALL  NZ,COPY_BUFF     ;+ Routine COPY-BUFF conditionally outputs line
                                 ;+ to printer leaving A=$00 and C=$21 and
                                 ;+ the zero flag reset - NZ.

;;;       POP   DE               ; Restore the character source address
;;;       LD    A,C              ; the new column number ($21) to A from C.

;   This is the screen-only path but we can trickle through as A!=C.

PR_ALL_1
          CP    C                ; this test is really for screen - new line ?
;;;       PUSH  DE               ; save source

          CALL  Z,PO_SCR         ; routine PO-SCR considers scrolling.

          POP   DE               ; restore source address.

;   The following applies to screen and printer.

PR_ALL_1a PUSH  BC               ; save line/column
          PUSH  HL               ; and destination

          LD    A,($5B91)        ; fetch P_FLAG to accumulator
          LD    B,$FF            ; prepare an OVER mask in B.
          RRA                    ; carry is set if temporary bit is OVER 1
          JR    C,PR_ALL_2       ; forward, if OVER 1, to PR-ALL-2

          INC   B                ; set OVER mask to 0

PR_ALL_2  RRA                    ; skip bit 1 of P_FLAG
          RRA                    ; bit 2 is temporary INVERSE
          SBC   A,A              ; will be FF for INVERSE 1 else zero
          LD    C,A              ; transfer the INVERSE mask to C

          LD    A,$08            ; prepare to count 8 bytes
          AND   A                ; clear carry to signal screen in use.

          BIT   1,(IY+$01)       ; test FLAGS  - is screen in use ?
          JR    Z,PR_ALL_3       ; forward, if screen, to PR-ALL-3

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

          SCF                    ; set the carry flag to signal printer in use.

PR_ALL_3  EX    DE,HL            ; now HL=source, DE=destination

PR_ALL_4  EX    AF,AF'           ; Save the printer/screen Carry flag

          LD    A,(DE)           ; Fetch the 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 location.

          EX    AF,AF'           ; restore discriminating flag
          JR    C,PR_ALL_6      ; forward, if printer, to PR-ALL-6

;   Continue with screen printing.

          INC   D               ; increment D - gives next screen pixel line

PR_ALL_5  INC   HL              ; address next character source byte
          DEC   A               ; the byte count is decremented
          JR    NZ,PR_ALL_4     ; back to PR-ALL-4 for all 8 bytes

          EX    DE,HL           ; transfer destination to HL
          DEC   H               ; bring back to last updated screen position
                                ; from the 'ninth' line.

          BIT   1,(IY+$01)      ; test FLAGS - is printer in use ?

          CALL  Z,PO_ATTR       ; if not, call routine PO-ATTR to update the
                                ; corresponding colour attribute.
                                ; (the address of which is now retained in DE)

          POP   HL              ; restore original screen/printer position
          POP   BC              ; and the line and column

          DEC   C               ; move column to right
          INC   HL              ; increase screen/printer position

          RET                   ; return and continue into PO-STORE
                                ; within PO-ABLE

;   Note. that DE has been made to retain the attribute byte.

; ---

;   This branch is used to update the ZX printer position by 32 places
;   Note. The high byte of the address D now increments if a page boundary
;   is crossed as this ROM supports up to thirteen ZX Printer buffers.

PR_ALL_6  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
          JR    NC,PR_ALL_7     ;+ skip forward if no wrap.

          INC   D               ;+ increment the high byte of channel address.

PR_ALL_7  EX    AF,AF'          ; fetch the flag
          JR    PR_ALL_5        ; back to PR-ALL-5

; ----------------------------------
; THE 'UPDATE ATTRIBUTE CELL' 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   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
```

```
PO_ATTR   CALL  CL_ATTR2        ;+ NEW subroutine with above code.

          LD    DE,($5B8F)      ; make D hold ATTR_T, E hold MASK-T
          LD    A,(HL)          ; fetch existing attribute from attribute file
          XOR   E               ; apply masks
          AND   D               ;
          XOR   E               ;
          BIT   6,(IY+$57)      ; test P_FLAG  - is this PAPER 9 ??
          JR    Z,PO_ATTR_1     ; skip, if not, to PO-ATTR-1

          AND   $C7             ; set paper
          BIT   2,A             ; to contrast with ink
          JR    NZ,PO_ATTR_1    ; skip to PO-ATTR-1

          XOR   $38             ;

PO_ATTR_1 BIT   4,(IY+$57)      ; test P_FLAG - is this INK 9 ??
          JR    Z,PO_ATTR_2     ; skip, if not, to PO-ATTR-2

          AND   $F8             ; make the ink colour contrast with paper.
          BIT   5,A             ; Is paper light ?

          JR    NZ,PO_ATTR_2    ; forward, if so, to PO-ATTR-2

          XOR   $07             ; toggle ink colour.

PO_ATTR_2 LD    (HL),A          ; write the new attribute to the attribute file

          EX    DE,HL           ;+ Note. NEW - return the attribute byte in DE.

          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_0  XOR   A               ;+ NEW entry point to print first message.
PO_MSG_1  SET   5,(IY+$02)      ;+ update TV_FLAG  - signal lower screen will
                                ;+ require clearing.

; -> Normal Entry Point.

PO_MSG    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    PO_TABLE        ; forward to PO-TABLE.

; ---

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

PO_TOKENS LD    DE,TKN_TABLE    ; address: TKN-TABLE
          PUSH  AF              ; stack the token number to control
                                ; trailing spaces - see later *

; ->

PO_TABLE  CALL  PO_SEARCH       ; routine PO-SEARCH will set carry for
```

```
                                ; all messages and function words.

        JR    C,PO_EACH        ; 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,PO_SAVE        ; routine PO-SAVE to print the space in A.

        CALL  Z,PO_SV_SP       ; routine PO-SV_SP to print a space without
                               ; disturbing registers.

PO_EACH LD    A,(DE)           ; Fetch character from the table.
        AND   $7F              ; Cancel any inverted bit.

        CALL  PO_SAVE          ; 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 e.g. $41 -> $82)
        JR    NC,PO_EACH       ; back, if not, to PO-EACH

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

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

        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 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.

PO_SV_SP 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 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.

; -------------------------
; THE 'TABLE SEARCH' ROUTINE
; --------------------------
;   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 PUSH  AF              ; save the original message/token number

          EX    DE,HL           ; transfer table address, DE to HL
          INC   A               ; adjust for initial step-over byte

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

;   The start of a new message token.

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

;   Register HL now addresses the first character of the required message.

          EX    DE,HL           ; transfer address to DE

          POP   AF              ; restore original message/token number

          CP    $20             ; compare to thirty two
          RET   C               ; return for all messages and function tokens.

;   Note. there are thirty error messages, originally twenty eight.

          LD    A,(DE)          ; test first character of token
          SUB   $41             ; against character 'A'

          RET                   ; Return - with carry set if it is less
                                ; i.e. '<>', '<=', '>='

; -------------------------------
; THE 'TEST FOR SCROLL' SUBROUTINE
; -------------------------------
;   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.
;   The current channel could be the upper screen 'S' in which case the
;   'scroll?' prompt is printed or from the lower screen 'K' in which case
;   no prompt is given.


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

;   Continue if handling upper or lower screen.

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

          LD    A,B             ; transfer the line to A.
          BIT   0,(IY+$02)      ; test TV_FLAG - lower screen in use ?
          JP    NZ,PO_SCR_4     ; jump forward, if so, to PO-SCR-4

          CP    (IY+$31)        ; greater than DF_SZ display file size ?
REP_5     JR    C,REPORT_5b     ; forward, if less, to REPORT-5
```

```
                                ; 'Out of screen'

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

          BIT   4,(IY+$02)      ; test TV_FLAG  - Automatic listing ?
          JR    Z,PO_SCR_2      ; forward, if not, to PO-SCR-2

          LD    E,(IY+$2D)      ; fetch BREG - the count of scroll lines to E.
          DEC   E               ; decrease
          JR    Z,PO_SCR_3      ; forward, if zero to scroll, at PO-SCR-3.

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

          CALL  CHAN_ZERO       ;+ routine CHAN-OPEN opens it invoking TEMPS.

          LD    SP,($5B3F)      ; set stack pointer to LIST_SP

PO_N_AUTO RES   4,(IY+$02)      ; Update TV_FLAG - signal auto listing finished.

          RET                   ; return, ignoring pushed value CL-SET, to MAIN
                                ; or EDITOR without updating print position  >>

; ---

REPORT_5b RST   30H             ; ERROR-1
          DEFB  $04             ; Error Report: Out of screen

; ---

;   Continue here if not an automatic listing.

PO_SCR_2  DEC   (IY+$52)        ; decrease the scroll count - SCR_CT
          JR    NZ,PO_SCR_3     ; forward, if not zero, to scroll at PO-SCR-3

;   If scroll count is zero, produce prompt, so that user can see the scrolled
;   output and BREAK if desired.

          LD    A,$18           ; prepare 24 decimal.
          SUB   B               ; subtract the current line.
          LD    ($5B8C),A       ; update the scroll count - SCR_CT

;   Although printing to lower screen will

          LD    HL,($5B8F)      ; L=ATTR_T, H=MASK_T
          PUSH  HL              ; save on stack

          LD    A,($5B91)       ; P_FLAG
          PUSH  AF              ; save on stack to prevent lower screen
                                ; attributes (BORDCR etc.) being applied.

          LD    A,$FD           ; select system channel 'K'

          CALL  CHAN_SLCT       ; routine CHAN-OPEN opens it and invokes TEMPS.

;;;       XOR   A               ; clear to address message directly
          LD    DE,scrl_mssg    ; make DE address: scrl-mssg

          CALL  PO_MSG_0        ; routine PO-MSG prints 'scroll?' to the lower
                                ; screen.

;;;       SET   5,(IY+$02)      ; set TV_FLAG  - signal lower screen requires
                                ; clearing
```

```
        LD    HL,$5B3B        ; 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  WAIT_KEY        ; routine WAIT_KEY waits for new key

        EXX                   ; switch back to alternate set.

        CP    $20             ; space is considered as BREAK
        JR    Z,REPORT_D      ; forward, if so, to REPORT-D
                              ; 'BREAK - CONT repeats'

        CP    $E2             ; is character 'STOP' ?
        JR    Z,REPORT_D      ; forward, if so, to REPORT-D
                              ; 'BREAK - CONT repeats'

        OR    $20             ; convert to lower-case
        CP    $6E             ; is character 'n' ?
        JR    Z,REPORT_D      ; forward, if so, to REPORT-D
                              ; 'BREAK - CONT repeats'

;   Scrolling is required.

;;;     LD    A,$FE           ; select system channel 'S'
;;;     CALL  CHAN_SLCT       ;

        CALL  CHAN_O_FE       ;+ Routine CHAN-OPEN opens it but applies
                              ;+ ATTR_P to ATTR_T nullifying any embedded
                              ;+ colour items in the current print statement.

        POP   AF              ; Restore original P_FLAG
        LD    ($5B91),A       ; and save in P_FLAG.
        POP   HL              ; Restore original ATTR_T, MASK_T
        LD    ($5B8F),HL      ; and reset ATTR_T, MASK-T as 'scroll?' has
                              ; been printed.

PO_SCR_3 CALL  CL_SC_ALL      ; 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  CL_ADDR         ; 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             ;

        CALL  CL_ATTR2        ;+ Note. A NEW routine with the above code.

        LD    DE,$5AE0        ; start of last 'line' of attribute area

        LD    A,(DE)          ; get attribute for last line
        LD    C,(HL)          ; get attribute for base line of upper part
```

```
          LD    B,$20           ; there are thirty two attribute bytes to copy

          EX    DE,HL           ; swap the pointers.


PO_SCR_3A LD    (DE),A          ; exchange the two
          LD    (HL),C          ; attributes.
          INC   DE              ; address next source location.
          INC   HL              ; address next destination location.
          DJNZ  PO_SCR_3A       ; loop back to PO-SCR-3A
                                ; for all adjacent attribute cells.

          POP   BC              ; restore the line/column.

          RET                   ; return via CL-SET (was pushed on stack).

; --------------------
; THE 'SCROLL?' PROMPT
; --------------------
;   The message 'scroll?' appears here with last byte inverted.

scrl_mssg DEFB  $80             ; initial step-over byte.
          DEFM  "scroll"
          DEFB  '?'+$80

; ---

REPORT_D  RST   30H             ; ERROR-1
          DEFB  $0C             ; Error Report: BREAK - CONT repeats

; ---

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

PO_SCR_4  CP    $02             ; is line number less than 2 ?
          JR    C,REPORT_5b     ; back, if so, to REPORT-5
                                ; 'Out of Screen'

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

          NEG                   ; Negate to give number of scrolls required.

          PUSH  BC              ; (*) save line/column
                                ; to prevent corruption by input AT

          LD    B,A             ; transfer count to B

;;;       LD    HL,($5B8F)      ; fetch current ATTR_T, MASK_T to HL.
;;;       PUSH  HL              ; and save
;;;       LD    HL,($5B91)      ; fetch P_FLAG
;;;       PUSH  HL              ; and save.

;;;       CALL  TEMPs           ; routine TEMPs sets to BORDCR etc.

          LD    A,B             ; transfer scroll number to A.

PO_SCR_4A PUSH  AF              ; save scroll number.

;   Now increment the lower screen display file size DF_SZ.
;   Retain the old value in the B register as scroll count
```

```
          LD    HL,$5B6B        ; 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,$5B89        ; address S_POSN_hi - line

          LD    L,$89           ; address S_POSN_hi - line
          CP    (HL)            ; compare DF_SZ to the line number.
          JR    C,PO_SCR_4B     ; forward, if less, to PO-SCR-4B
                                ; to scroll the lower screen only.

          INC   (HL)            ; else increment S_POSN_hi the upper line value

;;;       LD    B,$18           ; set count to whole display ??
;;;                             ; Note. should be $17 (not $18) and the top
;;;                             ; line will be scrolled into the ROM which
;;;                             ; is harmless on the standard set up.
;;;                             ; credit: P. Giblin 1984.
          LD    B,$17           ;+

PO_SCR_4B CALL  CL_SCROLL       ; routine CL-SCROLL scrolls bottom B lines up.

          POP   AF              ; restore the scroll counter.

          DEC   A               ; decrease counter.
          JR    NZ,PO_SCR_4A    ; 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    ($5B8F),HL      ; and update system variables.

          LD    BC,($5B88)      ; fetch upper display line/column S_POSN to BC.

          RES   0,(IY+$02)      ; signal to TV_FLAG - main screen in use.

          CALL  CL_SET          ; call routine CL-SET for upper display.

          POP   BC              ; (*) restore lower line/column

SIG_L_SCR SET   0,(IY+$02)      ; signal to TV_FLAG - lower screen in use.

          RET                   ; return via CL-SET for lower display.

; ----------------------------------------------
; THE 'SET TEMPORARY COLOUR ATTRIBUTES' ROUTINE
; ----------------------------------------------
;   This subroutine is called several times to copy the permanent colour items
;   to the temporary ones.

TEMPS     XOR   A               ; clear the accumulator
          LD    HL,($5B8D)      ; fetch L = ATTR_P and H = MASK_P

          BIT   0,(IY+$02)      ; test TV_FLAG  - is lower screen in use ?
          JR    Z,TEMPS_1       ; skip, if not lower screen, to TEMPS-1

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

TEMPS_1   LD    ($5B8F),HL      ; update system variables ATTR_T and MASK_T
```

```
;   For the print flag the permanent values are odd bits, temporary even bits.
;   For the lower screen the temporary mask bits are reset.  The ink colour
;   has already been chosen to contrast with the border colour and attributes
;   like OVER 1 are never allowed as it would confuse. For the upper screen
;   then ink 9, paper 9, inverse 1, over 1 are the same as permanent values.

          LD    HL,$5B91        ; address the print flag P_FLAG.
          JR    NZ,TEMPS_2      ; skip, if lower screen using zero, to TEMPS-2

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

TEMPS_2   XOR   (HL)            ;
          AND   $55             ; mask %01010101
          XOR   (HL)            ; permanent bits now as original

          LD    (HL),A          ; apply the updated temporary bits.

          RET                   ; 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       CALL  CL_ALL          ; 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 LD    HL,$5B3C        ; 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  TEMPs           ; 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  CL_LINE         ; 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,($5B8D)       ; fetch permanent attribute from ATTR_P.

          DEC   B               ; decrement lower screen display file size.

          JR    CLS_3           ; 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     LD    C,$20           ; set counter to 32 character cells per line

CLS_2     DEC   HL              ; decrease attribute address.
          LD    (HL),A          ; and place attributes in next line up.
          DEC   C               ; decrease the 32 counter.
          JR    NZ,CLS_2        ; loop back to CLS-2 until all 32 cells done.

CLS_3     DJNZ  CLS_1           ; 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 should they
;   have been left in an unstable state while outputting or inputting colour
;   control codes.

CL_CHAN   LD    A,$FD           ; select system channel 'K'

          CALL  CHAN_SLCT       ; routine CHAN-OPEN opens it.

;;;       LD    HL,($5B51)      ; fetch CURCHL to HL to address current channel
;;;       LD    DE,PRINT_OUT    ; set address to 'PRINT-OUT' for first pass.
;;;       AND   A               ; clear carry for first pass.

          CALL  PO_NORM         ;+ routine embodies above two instructions.

;;; CL_CHAN_A 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,KEY_INPUT    ; fetch address 'KEY-INPUT' for second pass

          CALL  KEY_CH2         ;+ inserts values

;;;       CCF                   ; complement carry flag - will set on pass 1.
;;;       JR    C,CL_CHAN_A     ; back to CL-CHAN-A if first pass else done.

          LD    BC,$1721        ; line 23 for lower screen

          JR    CL_SET          ; exit via CL-SET to set column
                                ; for lower display


; -----------------------------------
; THE 'CLEAR WHOLE DISPLAY' SUBROUTINE
; -----------------------------------
;   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    LD    HL,$0000        ; Initialize plot coordinates.
          LD    ($5B7D),HL      ; Set system variable COORDS to 0,0.

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

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

;;;       LD    A,$FE           ; select system channel 'S'

          CALL  CHAN_O_FE       ;+ routine CHAN-OPEN opens it calling TEMPS.

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

          LD    B,$18           ; There are 24 text lines to clear.

          CALL  CL_LINE         ; routine CL-LINE clears 24 text lines and sets
                                ; attributes from ATTR-P.
                                ; This routine preserves B and sets C to $21.

;;;       LD    HL,($5B51)      ; fetch CURCHL make HL address output routine.

;;;       LD    DE,PRINT_OUT    ; address: PRINT-OUT
;;;       LD    (HL),E          ; is made
;;;       INC   HL              ; the normal
;;;       LD    (HL),D          ; output address.
          CALL  PO_NORM         ;+ make PRINT_OUT normal.

          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    BIT   1,(IY+$01)      ; test FLAGS  - is printer in use ?
          JR    NZ,CL_SET_2     ; forward, if so, to CL-SET-2

          LD    A,B             ; transfer line to A.
          BIT   0,(IY+$02)      ; test TV_FLAG - lower screen in use ?
          JR    Z,CL_SET_1      ; skip, if handling upper part, to CL-SET-1

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

CL_SET_1  PUSH  BC              ; save the line/column.
          LD    B,A             ; transfer line to B
                                ; (adjusted if lower screen)

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

CL_SET_2  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    PO_STORE        ; exit via PO-STORE
                                ; to update the relevant system variables.

; --------------------------
; THE 'SCROLLING' SUBROUTINE
; --------------------------
;   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 LD    B,$17           ; scroll 23 lines, after 'scroll?'.

CL_SCROLL CALL  CL_ADDR         ; routine CL-ADDR gets screen address in HL.
          LD    C,$08           ; there are 8 pixel lines to scroll.

CL_SCR_1  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,CL_SCR_3     ; forward, if partial scroll, to CL-SCR-3

;   Register HL points to top line of the third which must be copied to bottom
;   line of the previous third.
;   ( so HL = $4800 or $5000 )

CL_SCR_2  EX    DE,HL           ; transfer 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 to HL and destination to DE
          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  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,CL_SCR_2     ; back, if not, to CL-SCR-2.

          POP   HL              ; restore topmost address.
          INC   H               ; next pixel line down.
          POP   BC              ; restore counts.
          DEC   C               ; reduce pixel line count.
          JR    NZ,CL_SCR_1     ; back, if all eight not done, to CL-SCR-1

          CALL  CL_ATTR         ; routine CL-ATTR gets address in attributes
                                ; from current 'ninth line' and 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   PUSH  BC              ; save line count

          CALL  CL_ADDR         ; routine CL-ADDR gets top address

          LD    C,$08           ; there are eight pixel lines to a text line.

CL_LINE_1 PUSH  BC              ; save pixel line count
          PUSH  HL              ; and save the screen address
          LD    A,B             ; transfer the line to A (1-24).

CL_LINE_2 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),0          ; blank the first byte.

          LD    (HL),B          ;+ blank the first byte. [ was LD (HL),0 ]

          INC   DE              ; make DE point to next byte.

          LDIR                  ; block move 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,CL_LINE_2    ; 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,CL_LINE_1    ; back to CL-LINE-1 till all done.

          CALL  CL_ATTR         ; routine CL_ATTR gets attribute address
                                ; in DE and HL 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,($5B8D)       ; fetch ATTR_P - permanent attributes
          BIT   0,(IY+$02)      ; test TV_FLAG  - lower screen in use ?
          JR    Z,CL_LINE_3     ; skip, if not, to CL-LINE-3

          LD    A,($5B48)       ; else lower screen uses BORDCR as attribute.

CL_LINE_3 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 WAS made of this)
          RET                   ; return to the calling routine.

; -------------------------------
; THE 'ATTRIBUTE ADDRESS' ROUTINE
; -------------------------------
;   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   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.

CL_ATTR   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.

          EX    DE,HL           ; restore attribute address to HL
          DEC   H               ; decrease from ninth line to eighth.
```

```
CL_ATTR2  LD    A,H             ; fetch H to A - $47, $4F, or $57.

          RRCA                  ; divide by             ???
          RRCA                  ; eight.
          RRCA                  ; $08, $09 or $0A.
          AND   $03             ; $00, $01 or $02.
          OR    $58             ; $58, $59 or $5A.
          LD    H,A             ; save high byte of attributes.

          LD    D,H             ;
          LD    E,L             ;

          RET                   ; return.

; -------------------------------
; THE 'SCREEN ADDRESS' SUBROUTINE
; -------------------------------
;   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   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             ; register L a multiple of 32.

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

          AND   $18             ; mask  to form $00, $08 or $10.

          OR    $40             ; add $40 - the base address of screen.

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

          RET                   ; return.


; ----------------------------------
; THE NEW 'CHANNEL SPECIFIER' SUBROUTINE
; ----------------------------------
;   10 bytes.
;   This subroutine checks for a single character ALPHA.
;   It is also now used by the usr_$ function to exploit similarities in
;   the functional specification.

EXPT_SPEC CALL  STK_FETCH       ; routine STK-FETCH to fetch and delete the
                                ; string parameters.
                                ; DE points to the start, BC holds the length.

          LD    A,C             ;
          DEC   A               ;
          OR    B               ;

          LD    A,(DE)          ; fetch character

          RET   Z               ; return with single character.

REPORT_Ae RST   30H             ; ERROR-1
          DEFB  $09             ; 'Invalid argument'
```

```
; ----------------------------
; THE NEW 'BC POSITIVE' SUBROUTINE
; ----------------------------
;

BC_POSTVE CALL  STK_TO_BC       ;

          LD    A,D             ; fetch sign $01 or $FF (negative)
          OR    E               ; combine both signs - $FF if either negative.
          INC   A               ;

          RET   NZ              ; Return if both positive.

REPORT_By RST   30H             ; ERROR-1
          DEFB  $0A             ; Error Report: Integer out of range

;
;   Text for banner of CAT command
;

CAT1
          DEFB  $14,$01         ; Control codes for INVERSE 1
          DEFB  $CF             ; The ' CAT ' token.
          DEFB  $06             ; The 'comma control'
          DEFM  "Free "         ; Text.
CAT2

          DEFB  0,0             ; ballast


; ------------------
; THE '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      DI                    ; disable interrupts as this is time-critical.

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

;   now enter a loop to handle each pixel line.

COPY_1    PUSH  HL              ; save the screen address.
          PUSH  BC              ; and the line counter.

          CALL  COPY_LINE       ; 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,COPY_2       ; 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    DJNZ  COPY_1          ; back to COPY-1 for all lines.

COPY_END  LD    A,$04           ;+ output value 4 to port
          OUT   ($FB),A         ;+ to stop the slowed printer motor.
          EI                    ;+ enable interrupts.

          RET                   ;+ return

; ----------------------------
; THE 'COPY BUFFER' SUBROUTINE
; ----------------------------
;   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.
;   The routine is invoked in two situations.
;   1) From PO-ENTER when a carriage return is received.
;   2) From PR-ALL when the column count C is reduced to zero.

COPY_BUFF DI                    ; Disable Interrupts

;;;       LD    HL,$5B00        ; the old way.

          LD    HL,($5B51)      ;+ Address of Current Channel.
          LD    DE,$08          ;+ The offset to the 256 byte channel buffer.
          ADD   HL,DE           ;+

;;;       LD    B,$08           ; set count to 8 lines of 32 bytes.

          LD    B,E             ; set count to 8 lines of 32 bytes.

COPY_3    PUSH  BC              ; save counter.

          CALL  COPY_LINE       ; routine COPY-LINE outputs 32 bytes

          POP   BC              ; restore counter.
          DJNZ  COPY_3          ; loop back to COPY-3 for all 8 lines.
                                ; then stop motor and clear buffer.

;;; COPY_4 LD   A,$04           ; output value 4 to port
;;;       OUT   ($FB),A         ; to stop the slowed printer motor.
;;;       EI                    ; enable interrupts.

COPY_4    CALL  COPY_END        ;+

; -------------------------------------
; THE 'CLEAR PRINTER BUFFER' SUBROUTINE
; -------------------------------------
;   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.
;   As a consequence of this omission the buffer was needlessly
;   cleared when COPY was used and the screen/printer position was set to
;   the start of the buffer and the line number to 0 (B)
;   giving an 'Out of Screen' error.

CLEAR_PRB LD    HL,($5B51)      ;+ address of Current Channel.
```

```
          LD    DE,$08          ;+ the offset to buffer.
          ADD   HL,DE           ;+ now points to start of 256 byte buffer.

;;;       LD    HL,$5B00        ; The old way.
;;;       LD    (IY+$46),L      ; update PR_CC_lo - set to zero - superfluous.

;;;       XOR   A               ; clear the accumulator.
          LD    B,D             ; set count to 256 bytes.

PRB_BYTES LD    (HL),D          ; set addressed location to zero.
          INC   HL              ; address next byte - Note. not INC L.
          DJNZ  PRB_BYTES       ; 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    CL_SET          ;

          DEC   H               ;+ Set pointer to start of buffer.
          JP    PO_STORE        ;+ exit to PO-STORE to store C only.

;   Note. The correct value of HL is required for when COPY_BUFF is called at
;   the start of PR_ALL.

; --------------------------
; THE 'COPY LINE' SUBROUTINE
; --------------------------
;   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.
;
;   The slowing of the printer ensures that the two stylii, attached to the
;   motor-driven rubber belt, come to rest off the paper.

COPY_LINE 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  CALL  BREAK_KEY       ; Call BREAK-KEY to read keyboard immediately.

          JR    C,COPY_L_2      ; Forward, if 'break' not pressed, to COPY-L-2

;;;       LD    A,$04           ; Stop the
;;;       OUT   ($FB),A         ; printer motor.
;;;       EI                    ; Enable interrupts.

          CALL  COPY_END        ;+ Routine stops the motor and performs EI.

;;;       CALL  CLEAR_PRB       ; Call routine CLEAR-PRB.

;   Now see if it is part of the fixed screen that is being copied.
```

```
          LD    A,H              ;+ Fetch high byte of address being copied.
          CP    $58              ;+ Is address less than attribute file ?

          CALL  NC,CLEAR_PRB     ;+ If not call routine CLEAR-PRB.
                                 ;+ Note. should not be cleared if COPY in use.

REPORT_Dc RST   30H              ; ERROR-1
          DEFB  $0C              ; Error Report: BREAK - CONT repeats

; ---

COPY_L_2  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,COPY_L_1      ; back, if stylus not in position, to COPY-L-1

          LD    C,$20            ; set count to 32 bytes.

COPY_L_3  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  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  IN    A,($FB)          ; read the port.
          RRA                    ; bit 0 to carry.
          JR    NC,COPY_L_5      ; back, if stylus not in position, to COPY-L-5

TAG5      LD    A,D              ; transfer command bits to A.
L0F24:    OUT   ($FB),A          ; and output to port.
          DJNZ  COPY_L_4         ; loop back, for all 8 bits, to COPY-L-4

          DEC   C                ; decrease the byte count.
          JR    NZ,COPY_L_3      ; back, until 256 bits done, to COPY-L-3

          RET                    ; return to calling routine COPY/COPY-BUFF.


; --------------------
; THE 'EDITOR' ROUTINE
; --------------------
;   The editor is called to prepare or edit a BASIC line.
;   It is also called from INPUT to input a numeric or string expression or
;   to input characters sent by a network station or serial device.
;   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    LD    HL,($5B3D)       ; fetch ERR_SP
          PUSH  HL               ; save on stack

ED_AGAIN  LD    HL,ED_ERROR      ; address: ED-ERROR
          PUSH  HL               ; save address on stack and
          LD    ($5B3D),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   CALL  WAIT_KEY        ; routine WAIT-KEY gets key possibly changing
                                ; the mode.
          PUSH  AF              ; save the key.

;   Do we need to always click?

          LD    HL,$00C8        ; Give a short click.
;;;       LD    D,$00           ;
          LD    D,H             ;+
          LD    E,(IY-$01)      ; Use PIP value for duration.
          CALL  BEEPER          ; routine BEEPER gives click - effective
                                ; with rubber keyboard.

          POP   AF              ; get saved key value.

          LD    HL,ED_LOOP      ; 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,ADD_CHAR     ; forward, if so, to ADD-CHAR.

;;;       CP    $07             ; lower than 7 ?

          CP    $06             ;+ lower than 6 ?

          JR    C,ADD_CHAR      ; 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.

          LD    BC,$0002        ; Prepare early for ink/paper etc.

          CP    $10             ; less than 16 decimal ?
          JR    C,ED_KEYS       ; forward to ED-KEYS ,if editing control in the
                                ; range 6 to 15, as 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,ED_CONTR      ; forward, if so, to ED-CONTR

                                ; 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,ED_IGNORE     ; jump to ED-IGNORE if not, else

          CALL  WAIT_KEY        ; routine WAIT-KEY - input address is KEY-NEXT
                                ; but is reset to KEY-INPUT
          LD    E,A             ; save first in E

ED_CONTR  CALL  WAIT_KEY        ; routine WAIT-KEY for control.
```

```
                                ; input address will be key-next.

          PUSH  DE              ; saved code/parameters
          LD    HL,($5B5B)      ; fetch address of keyboard cursor from K_CUR

          RES   0,(IY+$07)      ; allow MODE 'L' or 'G' cancelling 'E'

          CALL  MAKE_ROOM       ; routine MAKE-ROOM makes 2/3 spaces at cursor

          POP   BC              ; restore code/parameters
;;;       INC   HL              ; Address the 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    ADD_CH_1        ; forward to ADD-CH-1

; ------------------------
; THE 'ADD CHAR' SUBROUTINE
; ------------------------
;   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  RES   0,(IY+$07)      ; allow MODE 'L' or 'G' cancelling 'E'

          LD    HL,($5B5B)      ; fetch address of keyboard cursor from K_CUR

          LD    BC,$0001        ; one space required
          CALL  MAKE_ROOM       ; create space at K_CUR.

;;;       CALL  ONE_SPACE       ; routine ONE_SPACE creates one space.

;   Either a continuation of above or from ED-CONTR with ED-LOOP on stack.

ADD_CH_1  LD    (DE),A          ; load current character to last new location.
          INC   DE              ; address next
          LD    ($5B5B),DE      ; and update K_CUR system variable.

          RET                   ; return - either a simple return
                                ; from ADD-CHAR or to ED-LOOP on stack.

; ---------------------
; THE 'ED KEYS' SECTION
; ---------------------
;   A branch of the editing loop to deal with control characters using a
;   look-up table.  On entry BC now holds $0002.

ED_KEYS   LD    E,A             ; character to E.
;;;       LD    D,$00           ; prepare to add.
          LD    D,B             ; prepare to add.

;;;       LD    HL,ED_KEYS_T -7 ;  base address of editing keys table.

          LD    HL,ED_KEYS_T -6 ;+ NEW base address of editing keys table.

          ADD   HL,DE           ; add E
          LD    E,(HL)          ; fetch one-byte offset to E.
          ADD   HL,DE           ; add offset for address of handling routine.
          PUSH  HL              ; push the routine address on the machine stack.

          LD    HL,($5B5B)      ; load address of the cursor from K_CUR.
```

```
;   New. carry results of next test into the routine to save performing
;   the tests separately within the routines.

TST_INP_M BIT   5,(IY+$37)      ;+ Test FLAGX - INPUT mode ?

          RET                   ; Make an indirect jump forward to routine.

;   Note Zero flag determines mode, BC holds $0002.

; ------------------------
; THE '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 the correct handling of comma-separated items, over the network,
;   there should be an entry for CHR$6 with offset to ED-SYMBOL. Done.

ED_KEYS_T DEFB  ED_SYMBOL-$     ;+ 06d offset  to Address: ED-SYMBOL
          DEFB  ED_EDIT  -$     ;  07d offset  to Address: ED-EDIT
          DEFB  ED_LEFT  -$     ;  08d offset  to Address: ED-LEFT
          DEFB  ED_RIGHT -$     ;  09d offset  to Address: ED-RIGHT
          DEFB  ED_DOWN  -$     ;  10d offset  to Address: ED-DOWN
          DEFB  ED_UP    -$     ;  11d offset  to Address: ED-UP
          DEFB  ED_DELETE-$     ;  12d offset  to Address: ED-DELETE
          DEFB  ED_ENTER -$     ;  13d offset  to Address: ED-ENTER
          DEFB  ED_SYMBOL-$     ;  14d offset  to Address: ED-SYMBOL
          DEFB  ED_GRAPH -$     ;  15d offset  to Address: ED-GRAPH

; -------------------------
; THE 'EDIT KEY' SUBROUTINE
; -------------------------
;   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   LD    HL,($5B49)      ; fetch E_PPC the last line number entered.
                                ; Note. may not exist and may follow program.

;;;       BIT   5,(IY+$37)      ; test FLAGX - INPUT mode ?

          JR    NZ,CLEAR_SP     ; jump forward, if INPUT mode, to CLEAR-SP

          CALL  LINE_ADDR       ; routine LINE-ADDR to find address of line
                                ; or following line if it doesn't exist.
                                ; in DE.
          CALL  LINE_NO         ; routine LINE-NO will get line number from
                                ; address or number of previous line if at the
                                ; end-marker.
          LD    A,D             ; If there is no program then DE will
          OR    E               ; contain zero so test for this.

          JR    Z,CLEAR_SP      ; jump forward, if so, to CLEAR-SP

;   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.

          LD    ($5B49),DE      ;+ make E_PPC number the true line number.

          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  TEST_ROOM       ; routine TEST-ROOM checks free memory.

          CALL  CLEAR_SP        ; routine CLEAR-SP clears editing area.

          LD    HL,($5B51)      ; 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  CHAN_SLCT       ; 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.
                                ; fixed above

          CALL  OUT_LINE        ; routine OUT-LINE outputs the BASIC line
                                ; to the editing area.
          INC   (IY+$0F)        ; restore E_PPC_lo to the previous value.

          LD    HL,($5B59)      ; address E_LINE in editing area.

          INC   HL              ; advance
          INC   HL              ; past space
          INC   HL              ; and digit characters
          INC   HL              ; of line number.

          LD    ($5B5B),HL      ; update K_CUR to address start of BASIC.

REST_CHAN POP   HL              ; restore the address of CURCHL.

          JP    CHAN_FLAG       ;+ routine CHAN-FLAG sets flags for it.

;;;       CALL  CHAN_FLAG       ; routine CHAN-FLAG sets flags for it.
;;;       RET                   ; RETURN to ED-LOOP.

; ---------------------------
; THE 'CLEAR SPACE' SUBROUTINE
; ---------------------------
;   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  PUSH  HL              ; preserve HL throughout.

          CALL  SET_HL          ; routine SET-HL
                                ; if in edit   HL = WORKSP-1, DE = E_LINE
                                ; if in input  HL = STKBOT,   DE = WORKSP
          DEC   HL              ; adjust

          CALL  RECLAIM_1       ; routine RECLAIM-1 reclaims space setting BC
```

```
                                  ; to zero.

          LD    ($5B5B),HL      ; set K_CUR to start of empty area.

;;;       LD    (IY+$07),$00    ; set MODE to 'KLC'

          LD    (IY+$07),B      ;+  set MODE to 'KLC'

          POP   HL              ; restore HL.
          RET                   ; return.

; ------------------------------------
; THE 'CURSOR DOWN EDITING' SUBROUTINE
; ------------------------------------
;   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   BIT   5,(IY+$37)  ; test FLAGX  - Input Mode ?

ED_DOWN   JR    NZ,ED_STOP      ; skip, if INPUT mode, to ED-STOP

          LD    HL,$5B49        ; address E_PPC - 'current line'
          CALL  LN_FETCH        ; routine LN-FETCH fetches number of next
                                ; line or same if at end of program.
          JR    ED_LIST         ; forward to ED-LIST to produce an
                                ; automatic listing.

; ---

ED_STOP   LD    (IY+$00),$10    ; set ERR_NR to 'STOP in INPUT' code
          JR    ED_ENTER        ; forward to ED-ENTER to produce error.

; ------------------------------------
; THE 'CURSOR LEFT EDITING' SUBROUTINE
; ------------------------------------
;   This acts on the cursor in the lower section of the screen in both
;   editing and input mode.

ED_LEFT   CALL  ED_EDGE         ; routine ED-EDGE moves left if possible
          JR    ED_CUR          ; forward to ED-CUR to update K-CUR
                                ; and return to ED-LOOP.

; -------------------------------------
; THE 'CURSOR RIGHT EDITING' SUBROUTINE
; -------------------------------------
;   This acts on the cursor in the lower screen in both editing and input
;   mode and moves it to the right.
;   Note. The new code, suggested by Andrew Owen, avoids placing the cursor
;   between a control code and its parameter.

ED_RIGHT  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

          CP    $15             ;+ OVER or higher
          JR    NC,ED_CUR       ;+

          CP    $0F             ;+
          JR    C,ED_CUR        ;+
          INC   HL              ;+ Step over a control code parameter.

ED_CUR    LD    ($5B5B),HL      ; update K_CUR system variable

          RET                   ; return to ED-LOOP

; -------------------------------
; THE 'EDITING DELETE' SUBROUTINE
; -------------------------------
;   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.
;   Note. the second method would delete both controls and their parameters.

ED_DELETE CALL  ED_EDGE         ; routine ED-EDGE moves cursor to left

          LD    BC,$0001        ; of character to be deleted.
          JP    RECLAIM_2       ; to RECLAIM-2 reclaim the one character.

;;;       EX    DE,HL           ;
;;;       JP    RECLAIM_1       ;


; -------------------------------
; THE 'EDITING IGNORE' SUBROUTINE
; -------------------------------
;   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 #7; a$.

ED_IGNORE CALL  WAIT_KEY        ; routine WAIT-KEY to ignore code.
          CALL  WAIT_KEY        ; routine WAIT-KEY to ignore next code.

; ----------------------------------
; THE 'EDITING ENTER KEY' SUBROUTINE
; ----------------------------------
;   The ENTER key has been pressed to have BASIC line or INPUT accepted.

ED_ENTER  POP   HL              ; discard address ED-LOOP
          POP   HL              ; drop address ED-ERROR

ED_END    POP   HL              ; the previous value of ERR_SP
          LD    ($5B3D),HL      ; is restored to ERR_SP system variable
          BIT   7,(IY+$00)      ; is ERR_NR $FF ?
          RET   NZ              ; return if 'OK'

          LD    SP,HL           ; else put error routine on stack
          RET                   ; and make an indirect jump to it.

; ------------------------------------
; THE 'ED-EDGE' SUBROUTINE
; ------------------------------------
;   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   SCF                     ; carry flag must be set to call the nested
          CALL  SET_DE            ; 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 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,ED_EDGE_2      ; forward, if not, to ED-EDGE-2
                                  ; 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.
;
;   However, as Vickers later revealed these can be encountered with the
;   RS232 and Network.

          LD    A,(DE)            ; reload leftmost character
          SUB   $17               ; decimal 23 ('tab')
          ADC   A,$00             ; will be 0 for 'tab' and 'at'.
          JR    NZ,ED_EDGE_2      ; forward, if not, to ED-EDGE-2
                                  ; HL has been incremented twice

          INC   HL                ; increment a third time for 'at'/'tab'

ED_EDGE_2 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,ED_EDGE_1       ; back to ED-EDGE-1 if cursor not matched.

          RET                     ; return.

; ----------------------------------
; THE 'CURSOR UP EDITING' SUBROUTINE
; ----------------------------------
;   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 BIT   5,(IY+$37)      ; test FLAGX - INPUT mode ?

ED_UP     RET   NZ              ; return if in INPUT mode - to ED-LOOP.

          LD    HL,($5B49)      ; get current line from E_PPC

          CALL  LINE_ADDR       ; routine LINE-ADDR gets address

          EX    DE,HL           ; and previous in DE

          CALL  LINE_NO         ; routine LINE-NO gets prev line number

          LD    HL,$5B4A        ; set HL to E_PPC_hi as next routine stores
                                ; top first.
          CALL  LN_STORE        ; 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   CALL  AUTO_LIST       ; routine AUTO-LIST lists to upper screen
                                ; including adjusted current line.
;;;       LD    A,$00           ;- explicit - select lower screen again

CHAN_ZERO XOR   A               ;+ select lower screen again.
          JP    CHAN_SLCT       ; exit via CHAN-OPEN to ED-LOOP

; -------------------------------
; THE '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 directional
;   channel connected to each other:-
;   then, when the other Spectrum has said PRINT #15; 24, 7
;   INPUT #15; x ; y  would then treat the comma control as a newline and the
;   control would skip to INPUT y.
;   On the standard Spectrum, it was possible to get round the missing chr$ 6
;   handler by sending multiple print items separated by a newline '.
;   Otherwise the expression "24,7" would be assigned to the first variable x
;   raising 'Nonsense in BASIC'.

;   chr$14 would have the same functionality.

;   This is chr$ 14.
ED_SYMBOL BIT   7,(IY+$37)      ; test FLAGX - is this INPUT LINE ?
          JR    Z,ED_ENTER      ; back, if not, to ED-ENTER
                                ; 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  JP    ADD_CHAR        ; jump back to ADD-CHAR

; ----------------------
; THE 'ED_ERROR' ROUTINE
; ----------------------
;   If an error occurs while editing, or inputting, then ERR_SP
;   points to the stack location holding address ED_ERROR.
;   Note. this is specifically designed to deal with a BREAK into network input.
```

```
ED_ERROR  BIT   4,(IY+$30)      ; test FLAGS2 - is K channel in use ?
          JR    Z,ED_END        ; back, if not, to ED-END

;   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  BEEPER          ; routine BEEPER emits a warning rasp.

          CALL  ED_RASP         ;+ call the above code in new subroutine.

          JP    ED_AGAIN        ; to ED-AGAIN to re-stack the address of
                                ; this routine and make ERR_SP point to it.


; ---------------------------
; THE 'KEYBOARD INPUT' ROUTINE
; ---------------------------
;   This is the service routine for the input stream of the keyboard channel
'K'.

KEY_INPUT BIT   3,(IY+$02)      ; test TV_FLAG  - has a key been pressed in
                                ; editor ?

          CALL  NZ,ED_COPY      ; 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 ?

          LD    HL,$5B3B        ;+ Address system variable FLAGS
          BIT   5,(HL)          ;+ test FLAGS - has a new key been pressed ?

          RET   Z               ; return if no key has been pressed.    >>

;   Continue if the interrupt routine has supplied a key.

          LD    A,($5B08)       ; system variable LASTK will hold last key -
                                ; from the interrupt routine.

;;;       RES   5,(IY+$01)      ; update FLAGS - reset the new key flag.
          RES   5,(HL)          ; +update FLAGS - reset the new key flag.

;;;       PUSH  AF              ; Save the input character.

;   Now test if screen is to be cleared. after scroll?, Start tape, the
;   copyright message or an error message.


          BIT   5,(IY+$02)      ; test TV_FLAG - clear lower screen ?

          JR    Z,KEY_CMP       ;+ forward if not

          BIT   3,(HL)          ;+ is FLAGS set - L mode

          PUSH  AF              ;+ Now save the input character.

;;;       CALL  NZ,CLS_LOWER    ;
          CALL  CLS_LOWER       ;+ routine CLS-LOWER.
          POP   AF               ; restore the character code and test result.

          JR    NZ,KEY_DONE2     ;+ forward as single key required.


KEY_CMP   CP    $20              ; if space or higher then
          JR    NC,KEY_DONE2     ; forward to KEY-DONE2 and return with carry
                                 ; set to signal key-found.

          CP    $10              ; with 16d INK and higher skip
          JR    NC,KEY_CONTR     ; forward to KEY-CONTR.

          CP    $06              ; for 6 - 15d
          JR    NC,KEY_M_CL      ; 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    KEY_DATA         ; forward to KEY-DATA with the
                                 ; parameter (0/1) in C.

; ---

;   Now separate capslock 06 from modes 7-15.

KEY_M_CL  JR    NZ,KEY_MODE      ; forward to KEY-MODE if not 06 (capslock)

          LD    HL,$5B6A         ; 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    KEY_FLAG         ; forward to KEY-FLAG to signal no-key.

; ---

KEY_MODE  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,$5B41         ; address the MODE system variable.
          CP    (HL)             ; compare with existing value before
          LD    (HL),A           ; inserting the new value.
          JR    NZ,KEY_FLAG      ; forward to KEY-FLAG if it has changed.

;;;       LD    (HL),$00         ; else make MODE zero - KLC mode XXXX

          LD    (HL),$00         ;+ else make MODE zero - KLC mode (D=0,fr CLS)
                                 ; Note. while in Extended/Graphics mode,
                                 ; the Extended Mode/Graphics key is pressed
                                 ; again to get out.

KEY_FLAG  CP    A                ; clear carry and reset zero flags -
                                 ; no actual key returned.
```

```
SIG_KSTAT SET   3,(IY+$02)      ; update TV_FLAG  - show key state has changed
          RET                   ; make the return.

; ---

;   now deal with colour controls - 16-23 ink, 24-31 paper

KEY_CONTR 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,KEY_DATA     ; forward to KEY-DATA with INK 16d and
                                ; colour in C.

          INC   A               ; else change from INK to PAPER (17d)

KEY_DATA  LD    (IY-$2D),C      ; put the colour (0-7)/state(0/1) in KDATA
          LD    DE,KEY_NEXT     ; address: KEY-NEXT will be next input stream
          JR    KEY_CHAN        ; forward to KEY-CHAN to change it ...

; ---

; ... so that INPUT_AD directs control to here at next call to WAIT-KEY

KEY_NEXT  LD    A,($5B0D)       ; pick up the parameter stored in KDATA.
          LD    DE,KEY_INPUT    ; address: KEY-INPUT will be next input stream
                                ; continue to restore default channel and
                                ; make a return with the control code.

KEY_CHAN  LD    HL,($5B4F)      ; address start of CHANNELS area using CHANS
                                ; system variable.
          INC   HL              ; step over the
KEY_CH2   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 SCF                   ; set carry flag to show a key has been found

          RET                   ; Return.

; ---------------------------------
; THE 'LOWER SCREEN COPYING' ROUTINE
; ---------------------------------
;   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.
;   It may also contain embedded colour control codes.

ED_COPY   CALL  TEMPS           ;.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,($5B8A)      ; fetch SPOSNL
          PUSH  HL              ; and save on stack.

          LD    HL,($5B3D)      ; fetch ERR_SP
          PUSH  HL              ; and save also
          LD    HL,ED_FULL      ; address: ED-FULL
          PUSH  HL              ; is pushed as the error routine

          LD    ($5B3D),SP      ; and ERR_SP made to point to it.

          LD    HL,($5B82)      ; fetch ECHO_E
          PUSH  HL              ; and push also

          SCF                   ; set carry flag to control SET-DE
          CALL  SET_DE          ; call routine SET-DE
                                ; if in input DE = WORKSP
                                ; if in edit  DE = E_LINE
          EX    DE,HL           ; start address to HL

          CALL  OUT_LINE2       ; 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  OUT_CURS        ; routine OUT-CURS considers a
                                ; terminating cursor.

          LD    HL,($5B8A)      ; fetch updated SPOSNL
          EX    (SP),HL         ; exchange with ECHO_E on stack
          EX    DE,HL           ; transfer ECHO_E to DE

          CALL  TEMPS           ;.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  LD    A,($5B8B)       ; fetch SPOSNL_hi is current line
          SUB   D               ; compare with old
          JR    C,ED_C_DONE     ; forward to ED-C-DONE if no blanking

          JR    NZ,ED_SPACES    ; 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,ED_C_DONE    ; forward to ED-C-DONE if no backfilling.

ED_SPACES LD    A,$20           ; prepare a space.
          PUSH  DE              ; save old line/column.

          CALL  PRINT_OUT       ; routine PRINT-OUT prints a space over
                                ; any text from previous print.
                                ; Note. Since the blanking only occurs when
                                ; using PRINT_OUT 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    ED_BLANK        ; 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 to continue would perhaps not be the best idea you've had that day.

;;; ED_FULL   LD    D,$00       ; prepare to moan.
;;;       LD    E,(IY-$02)      ; fetch RASP value.
;;;       LD    HL,$1A90        ; set duration.

;;;       CALL  BEEPER          ; routine BEEPER.

;;;       LD    (IY+$00),$FF    ; clear ERR_NR.

ED_FULL   CALL  ED_RASP         ;+ call the above code in new subroutine.

          LD    DE,($5B8A)      ; fetch SPOSNL.
          JR    ED_C_END        ; forward to ED-C-END

; ----------------------------
; THE NEW 'ED_RASP' SUBROUTINE
; ----------------------------

ED_RASP   LD    D,$00           ;+ prepare to moan.
          LD    E,(IY-$02)      ;+ fetch RASP value.


          LD    HL,$1A90        ;+ set duration.

          CALL  BEEPER          ;+ routine BEEPER.

SET_ER_FF LD    (IY+$00),$FF    ;+ clear ERR_NR.

          RET                   ;+

; -------

;   the exit point from line printing continues here.

ED_C_DONE POP   DE              ; fetch new line/column.
          POP   HL              ; fetch the error address.

;   the error path rejoins here.

ED_C_END  POP   HL              ; restore the old value of ERR_SP.
          LD    ($5B3D),HL      ; update the system variable ERR_SP

          POP   BC              ; old value of SPOSN_L
          PUSH  DE              ; save new value

          CALL  CL_SET          ; routine CL-SET and PO-STORE update ECHO_E
                                ; and SPOSN_L from BC ( and sets D to zero)

          POP   HL              ; restore new value
          LD    ($5B82),HL      ; and overwrite ECHO_E

;;;       LD    (IY+$26),$00    ; make error pointer X_PTR_hi out of bounds

          LD    (IY+$26),D      ;+ 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    LD    HL,($5B61)      ; 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    LD    DE,($5B59)      ; fetch E_LINE to DE
;;;       BIT   5,(IY+$37)      ; test FLAGX  - Input Mode ?
          CALL  TST_INP_M       ;+ bit 5,(iy+$37) as a 3-byte call.
          RET   Z               ; return now if in editing mode

          LD    DE,($5B61)      ; fetch WORKSP to DE
          RET   C               ; return if carry set ( entry = set-de)

          LD    HL,($5B63)      ; 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 starting at a point in the
;   edit line or input workspace.

REMOVE_FP LD    A,(HL)          ; fetch character
          CP    $0E             ; is it the CHR$ 14 number marker ?
          LD    BC,$0006        ; prepare to strip six bytes

          CALL  Z,RECLAIM_2     ; 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 the line or the input buffer ?
          JR    NZ,REMOVE_FP    ; back to REMOVE-FP until entire line done.

          RET                   ; return.


; *********************************
; ** Part 6. EXECUTIVE ROUTINES  **
; *********************************


; The memory.
;
; +---------+-----------+------------+-----------+-+--
; | BASIC   |  Display  | Attributes |    System   |
; |  ROM    |   File    |    File    |  Variables  |
; +---------+-----------+------------+-----------+-+--
; ^         ^           ^            ^             ^
; $0000   $4000       $5800        $5B00         $5BC8 = 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 | GO SUB |?|$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       DI                      ; Disable Interrupts - machine stack will be
                                  ; cleared.
          LD    A,$FF             ; Flag coming from NEW.
          LD    DE,($5BB2)        ; Fetch RAMTOP as top value.
          EXX                     ; Switch in alternate set.
          LD    BC,($5BB4)        ; Fetch P-RAMT differs on 16K/48K machines.
          LD    DE,($5B38)        ; Fetch RASP/PIP.
          LD    HL,($5B7B)        ; 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 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                     ; Ho Ho Hum.
;;;       NOP                     ;
;;;       NOP                     ; Reset the network probably.
;;;       NOP                     ;

; -----------------------
; THE 'RAM CHECK' SECTION
; -----------------------
;   Typically, a Spectrum will have 16K or 48K of RAM and this code will test
;   it all until 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 LD    H,D             ; Transfer the top value to the HL register
          LD    L,E             ; pair.

RAM_FILL  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,RAM_FILL     ; Back, if not, to RAM-FILL

RAM_READ  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,RAM_DONE     ; forward to RAM-DONE if we've got back to
                                ; starting point with no errors.

          DEC   (HL)            ; decrement to 1.
          JR    Z,RAM_DONE      ; forward to RAM-DONE if faulty.

          DEC   (HL)            ; decrement to zero.
          JR    Z,RAM_READ      ; back to RAM-READ if zero flag was set.

RAM_DONE  DEC   HL              ; step back to last valid location.
          EXX                   ; regardless of state, set up possibly
                                ; stored system variables in case from NEW.
          LD    ($5BB4),BC      ; insert P-RAMT.
          LD    ($5B38),DE      ; insert RASP/PIP.
          LD    ($5B7B),HL      ; insert UDG.
          EXX                   ; switch in main set.
          INC   B               ; now test if we arrived here from NEW.
          JR    Z,RAM_SET       ; forward to RAM-SET if we did.

;   this section applies to START only.

          LD    ($5BB4),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    ($5B7B),HL      ; make UDG system variable address the first
                                ; bitmap.
          DEC   HL              ; point at RAMTOP again.

;;;       LD    BC,$0040        ; without disturbing HL, set the values of
;;;       LD    ($5B38),BC      ; the PIP and RASP system variables.
;;;                             ; Note. PIP is already zero.

          INC   A               ;+ increment from $3F to $40.
          LD    ($5B38),A       ;+ set RASP only to sixty four.

;   the NEW command path rejoins here.

RAM_SET   LD    ($5BB2),HL      ; set system variable RAMTOP to HL.

;   the NMI_ADD system variable points here by default to provide a Warm Reset.

NMI_PTR   LD    HL,$3C00        ; "A seemingly strange place to set CHARS"
          LD    ($5B36),HL      ; Note. but it all makes sense now - see L0066.

          LD    HL,($5BB2)      ; fetch RAMTOP to HL.
```

```
        LD    (HL),$3E        ; top of user ram holds GO SUB end marker
                              ; an impossible line number - see RETURN.
                              ; no significance in the number $3E. On the
                              ; ZX80 and ZX81 $3F was used.

        DEC   HL              ; followed by empty byte (not important).
        LD    SP,HL           ; set up the machine stack pointer.
        DEC   HL              ;
        DEC   HL              ;
        LD    ($5B3D),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,$5B3A        ; 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.

;   At this point check to see if the NMI has been activated.

        LD    A,($5B50)       ;+ fetch high byte of CHANS_hi
        AND   A               ;+ is it unitialized?
        JR    Z,SET_CHANS     ;+ forward if so as from NEW/START

;   else the NMI was activated and we don't want to lose the program.

        LD    A,$03           ;+ prepare to reset streams 2,1 and 0.

        CALL  NMI_STRMS       ;+ reset the streams - reclaiming any dynamic
                              ;+ buffers without incurring memory leaks.

        LD    A,$1D           ;+ prepare the NMI error code.
        JP    MAIN_G          ;+ forward to report NMI.

; ---

SET_CHANS

;;;     LD    HL,$5BB6        ; the old address of the channels

        LD    HL,$5BC9        ;+ the address of the channels - now following
                              ;+ the system variables NTHCS.

        LD    ($5B4F),HL      ; set the CHANS system variable.

        LD    DE,INIT_CHAN    ; address: init-chan in ROM.

;;;     LD    BC,$0015        ; there were 21  bytes of initial data.

        LD    C,$10           ;+ there are [16] 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    ($5B57),HL      ; set DATADD to location before program area.
        INC   HL              ; increment again.

        LD    ($5B53),HL      ; set PROG the location where BASIC starts.
```

```
          LD    ($5B4B),HL      ; set VARS to same location with a
          LD    (HL),$80        ; variables end-marker.
          INC   HL              ; advance address.
          LD    ($5B59),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.
                                ; --
;;; So let's test this theory
;;;       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    ($5B61),HL      ; set WORKSP - empty workspace.
;;;       LD    ($5B63),HL      ; set STKBOT - bottom of the empty stack.
;;;       LD    ($5B65),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    ($5B8D),A       ; set ATTR_P permanent colour attributes.
;;;       LD    ($5B8F),A       ; set ATTR_T temporary colour attributes.
          LD    ($5B48),A       ; set BORDCR the border colour/lower screen
                                ; attributes.

          LD    HL,$0523        ; The keyboard repeat and delay values are
          LD    ($5B09),HL      ; loaded to REPDEL and REPPER.

;   Now initialize BAUD and NTSTAT for RS232 and Network.

          LD    HL,$5BB6        ;+ address FLAGS3 - unused but 0 could mislead
          DEC   (HL)            ;+ so set to $FF

;   Use new WIDTH to control printer width

          INC   L               ;+ skip WIDTH lo
          INC   L               ;+ address WIDTH hi
          LD    (HL),$50        ;+ set width to 80 characters.

          INC   L               ;+ skip MAXIY - the last IY addressable loc.
          INC   L               ;+ address BAUD lo
          LD    (HL),$0C        ;+ set default BAUD rate (9600).

          INC   L               ;+ skip BAUD hi
          INC   L               ;+ address NTSTAT - own station number.
          INC   (HL)            ;+ Default Global Station Number to 1.

          LD    HL,NMI_PTR      ;+ initialize the NMI vector above.
          LD    ($5BB0),HL      ;+ set the NMI_ADD

;   Back to normal.

          DEC   (IY-$3A)        ; set KSTATE-0 to $FF - keyboard map available.
          DEC   (IY-$36)        ; set KSTATE-4 to $FF - keyboard map available.

          LD    HL,INIT_STRM    ; set source to ROM Address: init-strm
          LD    DE,$5B10        ; set destination to system variable STRMS-FD

;;;       LD    BC,$000E        ; copy the  14  bytes of initial  7  streams

          LD    C,$0C           ;+ copy the [12] bytes of initial [6] streams

          LDIR                  ; data from ROM to RAM.
```

```
;;;       SET   1,(IY+$01)      ; update FLAGS  - signal printer in use.
;;;       CALL  CLEAR_PRB       ; 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  CLS             ; 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,COPYRIGHT-1  ; the message table directly.
          CALL  PO_MSG_0        ; 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    MAIN_1          ; forward to MAIN-1

; -------------------------
; THE 'MAIN EXECUTION' LOOP
; -------------------------
;   This is the Main Execution Loop within which control remains after
;   initialization.  It is entered for the first time at MAIN-1 and thereafter
;   each iteration begins with an Automatic Listing.  An 'automatic Listing' is
;   one that appears without involving the LIST command, for example, when the
;   user presses [ENTER] after an Error Report.

MAIN_EXEC LD    (IY+$31),$02    ; set DF_SZ lower screen display file size to
                                ; two lines.
          CALL  AUTO_LIST       ; routine AUTO-LIST

;   The Initial Entry Point.

MAIN_1    CALL  SET_MIN         ; routine SET-MIN clears work areas.

;;; MAIN_2  LD    A,$00         ;- explicit - select stream zero.

MAIN_2
          CALL  CHAN_ZERO       ;+ routine CHAN_ZERO opens channel zero

MAIN_2b   res   3,(iy+$02)      ;+ Gotcha! Signal no change in Mode.

          CALL  EDITOR          ; 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  LINE_SCAN       ; routine LINE-SCAN scans the User's input.

          BIT   7,(IY+$00)      ; test ERR_NR - will be $FF if syntax is OK.
          JR    NZ,MAIN_3       ; forward, if correct, to MAIN-3.

;   Note. Now test if channel 'K' is in use

          BIT   4,(IY+$30)      ; test FLAGS2 - K channel in use ?
          JR    Z,MAIN_4        ; forward, if not, to MAIN-4
```

```
;   Channel 'K' was in use so X_PTR will have been set.

          LD    HL,($5B59)      ; an editing error so address E_LINE.

          CALL  REMOVE_FP       ; routine REMOVE-FP removes the hidden
                                ; floating-point forms.

;;;       LD    (IY+$00),$FF    ; system variable ERR_NR is reset to 'OK'.

          CALL  SET_ER_FF       ;+ NEW 3-byte call

          JR    MAIN_2b         ; back to MAIN-2 to allow user to correct.

; ---

;   The branch was here if syntax has passed test.

;;; MAIN_3    LD    HL,($5B59)      ; fetch the edit line address from E_LINE.
;;;           LD    ($5B5D),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.

MAIN_3    CALL  E_LINE_NO       ; 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,MAIN_ADD     ; 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,MAIN_EXEC     ; back, if so, to MAIN-EXEC
                                ; for an automatic listing.

;   This must be a direct command.

          BIT   0,(IY+$30)      ; test FLAGS2 - clear the main screen ?

          CALL  NZ,CL_ALL       ; routine CL-ALL, if so, e.g. after listing.

          CALL  CLS_LOWER       ; routine CLS-LOWER anyway.

          LD    A,$19           ; compute scroll count as twenty five
          SUB   (IY+$4F)        ; minus the value of S_POSN_hi.
          LD    ($5B8C),A       ; update SCR_CT system variable.

          SET   7,(IY+$01)      ; update FLAGS - signal running program.

;;;       LD    (IY+$00),$FF    ; set ERR_NR to 'OK'.
          CALL  SET_ER_FF       ;+ NEW 3-byte call

          LD    (IY+$0A),$01    ; set NSPPC to one for first statement.

          CALL  LINE_RUN        ; 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.
;   Also, OPEN #0,"n";2 which allows another Spectrum to take control of this
;   one.
;   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     HALT                    ; wait for interrupt the only routine that can
                                   ; set bit 5 of FLAGS.
           RES   5,(IY+$01)        ; reset bit 5 of FLAGS - signal no new key.

;;;        BIT   1,(IY+$30)        ; test FLAGS2 - is printer buffer clear ?
;;;        CALL  NZ,COPY_BUFF      ; call routine COPY-BUFF if not empty.
;;;                                ; Note. the programmer has neglected
;;;                                ; to set bit 1 of FLAGS first.

           LD    A,($5B3A)         ; 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     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    ($5B0B),HL        ; blank DEFADD to signal that no defined
;;;                                ; function is currently being evaluated.

           XOR   A                 ; Set accumulator to zero
           LD    (IY+$37),A        ; clear all the bits of FLAGX.
           LD    (IY+$26),A        ; blank X_PTR_hi to suppress error marker.
           LD    (IY+$2E),A        ; blank DEFADD_hi to signal inactive.

;;;        LD    HL,$0001          ; prepare stream data.

;;;        LD    ($5B16),HL        ; ensure STRMS-00 is the keyboard.
;;;                                ; and not the network as would have been set
;;;                                ; by OPEN #0, "n" ; 2

           CALL  SET_MIN           ; 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  CLS_LOWER         ; 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,MAIN_5          ; forward, if so, to MAIN-5

           ADD   A,$07             ; add ASCII offset to letters.

MAIN_5     CALL  OUT_CODE          ; 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,rpt_mesgs    ; address: rpt-mesgs.

          CALL  PO_MSG_1        ; call routine PO-MSG to print the message.

;;;       XOR   A               ; clear accumulator to directly
;;;       LD    DE,comma_sp -1  ; address the comma and space message.
;;;       CALL  PO_MSG_0        ; routine PO-MSG prints ', ' although it would
;;;                             ; be more succinct to use RST $10.

          LD    A,','           ;+ comma
          RST   10H             ;+ print
          LD    A,' '           ;+ space
          RST   10H             ;+ print

          LD    BC,($5B45)      ; fetch PPC the current line number.
          CALL  OUT_NUM_1       ; 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  OUT_NUM_0       ; routine OUT-NUM-0 prints C.

          CALL  CLEAR_SP        ; routine CLEAR-SP clears editing area which
                                ; probably contained 'RUN'.   (B = 0)

          LD    A,($5B3A)       ; fetch ERR_NR again
          INC   A               ; test for no error originally $FF.
          JR    Z,MAIN_9        ; forward, if no error, to MAIN-9

          CP    $09             ; is code Report 9 STOP ?
          JR    Z,MAIN_6        ; forward, if so, to MAIN-6

          CP    $15             ; is code Report L BREAK ?
          JR    NZ,MAIN_7       ; forward, if so, to MAIN-7

;   Stop or Break was encountered so consider CONTINUE.

MAIN_6    INC   (IY+$0D)        ; increment SUBPPC to next statement.

;;; MAIN_7 LD   BC,$0003        ; prepare to copy 3 system variables to

MAIN_7    LD    C,$03           ;+  prepare to copy 3 system variables to

          LD    DE,$5B70        ; ...address OSPPC - statement for CONTINUE.
                                ; also updating OLDPPC line number below.

          LD    HL,$5B44        ; 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,MAIN_8        ; 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    LDDR                  ; copy PPC to OLDPPC and SUBPPC to OSPCC
                                ; or NSPPC to OLDPPC and NEWPPC to OSPCC
```

```
MAIN_9    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    MAIN_2          ; jump back to MAIN-2.


; ----------------------------
; THE 'CANNED REPORT MESSAGES'
; ----------------------------
;   The 30 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 RETURN without GOSUB, 1000:127" [ 32 characters ]

rpt_mesgs 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  "Loading erro"
          DEFB  'r'+$80                          ; R

          DEFM  "Stream close"                   ;+
          DEFB  'd'+$80                          ;+ S
          DEFM  "NM"                             ;+
          DEFB  'I'+$80                          ;+ T
          DEFM  "Net R/W erro"                   ;+
          DEFB  'r'+$80                          ;+ U

;;; comma_sp DEFB  ',',' '+$80                   ; used in report line.

COPYRIGHT DEFB  $7F                              ; copyright
          DEFM  " 1982 Sinclair Research Ltd"
          DEFB  '.'+$80                          ;+ just differentiate


; ----------------------
; THE 'REPORT_G' ROUTINE
; ----------------------
;   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.  Since this can only occur as a result of a direct command,
;   there is no need to record the X-PTR via the error restart.

;   No room for line
REPORT_G  LD    A,$10           ; i.e. 'G' -$30 -$07

;;;       LD    BC,$0000        ; this seems unnecessary.

          JP    MAIN_G          ; jump back to MAIN-G

; ----------------------
; THE 'MAIN_ADD' SECTION
; ----------------------
;   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  LD    ($5B49),BC      ; set E_PPC to extracted line number.
          RST   18H             ;;;;
;;;       LD    HL,($5B5D)      ; 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,REPORT_G     ; 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,($5B61)      ; 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  LINE_ADDR         ; routine LINE-ADDR will see if
                                  ; a line with the same number exists.
          JR    NZ,MAIN_ADD1      ; forward if no existing line to MAIN-ADD1.

;;;       CALL  NEXT_ONE          ; routine NEXT-ONE finds the existing line.
;;;       CALL  RECLAIM_2         ; routine RECLAIM-2 reclaims it.

          CALL  NXT_1_RC2         ;+ routine combines above 2 routines.

MAIN_ADD1 POP   BC                ; retrieve the length of the new line.
          LD    A,C               ; and test if a carriage return only
          DEC   A                 ; i.e. one byte long.
          OR    B                 ; result would be zero.
          JR    Z,MAIN_ADD2       ; forward, if so, to MAIN-ADD2

          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,($5B53)        ; fetch the address of PROG
          PUSH  DE                ; and save it on the stack

          CALL  MK_RM_DHL         ;+ MAKE_ROOM decrementing HL first

;;;       CALL  MAKE_ROOM         ; routine MAKE-ROOM creates BC spaces in
                                  ; program area and updates pointers.
          POP   HL                ; restore old program pointer.
          LD    ($5B53),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,($5B61)        ; set HL to WORKSP - location after edit line.
          DEC   HL                ; decrement to address end marker.
          DEC   HL                ; decrement to address the carriage return.

          LDDR                    ; copy the BASIC line back to initial command.

          LD    HL,($5B49)        ; 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 POP   AF                ; drop the address of Report G

          JP    MAIN_EXEC         ; 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 DEFW  PRINT_OUT       ; PRINT-OUT
          DEFW  KEY_INPUT       ; KEY-INPUT
          DEFB  $4B             ; 'K'
          DEFW  PRINT_OUT       ; PRINT-OUT
          DEFW  REPORT_J        ; REPORT-J
          DEFB  $53             ; 'S'
          DEFW  ADD_CHAR        ; ADD-CHAR
          DEFW  REPORT_J        ; RAW_INPUT
          DEFB  $52             ; 'R'

;;;       DEFW  PRINT_OUT       ; PRINT-OUT
;;;       DEFW  REPORT_J        ; REPORT-J
;;;       DEFB  $50             ; 'P'

          DEFB  $80             ; End Marker

REPORT_J  RST   30H             ; 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 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  BIT   5,(IY+$02)      ; test TV_FLAG - clear lower screen ?
          JR    NZ,WAIT_KEY1    ; forward, if so, to WAIT-KEY1

          SET   3,(IY+$02)      ; update TV_FLAG - signal reprint the edit
                                ; line to the lower screen. SIG_KSTAT.

WAIT_KEY1 CALL  INPUT_AD        ; routine INPUT-AD is called.
```

```
          RET   C               ; return with acceptable keys.

          JR    Z,WAIT_KEY1     ; 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 normally taken.
;   It is taken when 'Iris' closes her channel.

REPORT_8  RST   30H             ; 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  EXX                   ; switch in alternate set.
          PUSH  HL              ; save HL register

          LD    HL,($5B51)      ; fetch address of CURCHL - current channel.
          INC   HL              ; step over output routine
          INC   HL              ; to point to low byte of input routine.
          JR    CALL_SUB        ; forward to CALL-SUB.

; ----------------------
; THE 'OUT CODE' ROUTINE
; ----------------------
;   This routine is called on five occasions to print the ASCII equivalent of
;   a value 0-9.

OUT_CODE  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
; -------------------------
;   The PRINT-A-2 is a continuation of the RST 10 restart that outputs 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 PRINT_OUT.

PRINT_A_2 EXX                   ; switch in alternate set
          PUSH  HL              ; save HL register
          LD    HL,($5B51)      ; fetch CURCHL the current channel.

;   input-ad rejoins here also.

CALL_SUB  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  CALL_JUMP       ; use routine CALL-JUMP in effect CALL (HL).

          POP   HL              ; restore saved HL register.
          EXX                   ; switch back to the main set and
          RET                   ; return.

; ---
;   Note. the most popular channel number could be placed here e.g. LD A,$FE

; -------------------------------
; THE 'OPEN CHANNEL 0xFE' ROUTINE
; -------------------------------

CHAN_O_FE LD    A,$FE           ;
          JR    CHAN_SLCT       ;

; ---------------------------------
; THE 'OPEN CHANNEL SYNTAX' ROUTINE
; ---------------------------------

CHN_O_SYN CALL  UNSTACK_Z       ;+ Return if Checking Syntax.

; --------------------------
; THE 'CHANNEL SELECT' ROUTINE
; --------------------------
;   This subroutine is used by the ROM to select 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_SLCT ADD   A,A             ; double the stream ($FF will become $FE etc.)
          ADD   A,$16           ; add the offset to stream 0 from $5B00
          LD    L,A             ; result to L
          LD    H,$5B           ; 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.
          JP    Z,REPORT_O      ; forward if closed to report
                                ; 'Invalid stream'

;;;       JR    NZ,CHAN_OP_1    ; forward to CHAN-OP-1 if open.
;;; REPORT_Oa   RST   30H       ; 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 DEC   DE              ; reduce offset so it points to the channel.
          LD    HL,($5B4F)      ; 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.

; ------------------------------
; THE 'CHANNEL FLAGS' SUBROUTINE
; ------------------------------
;   This subroutine is used from ED-EDIT, str$ and read-in to reset the
;   current channel when it has been temporarily altered.

CHAN_FLAG RES   4,(IY+$30)      ; update FLAGS2  - signal K channel not in use.
                                ; Note. provide a default for
                                ; channel 'R','S' and 'P'.
          LD    ($5B51),HL      ; set CURCHL system variable to the
                                ; address in HL
;;;       INC   HL              ; advance past
;;;       INC   HL              ; output routine.
;;;       INC   HL              ; advance past
;;;        INC   HL               ; input routine.
;;;        LD    C,(HL)           ; pick up the letter.

           CALL  IN_CHAN_K        ;+ routine gets channel letter in A.

           LD    HL,CHN_CD_LU-1   ; address: chn-cd-lu

           CALL  INDEXER_0        ; 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 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.

; -------------------------------
; THE '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 DEFB  'K', CHAN_K-$-1 ; offset $06 to CHAN-K
          DEFB  'S', CHAN_S-$-1 ; offset $12 to CHAN-S
          DEFB  'P', CHAN_P-$-1 ; offset $1B to CHAN-P

          DEFB  $00             ; end marker.

; ---------------------------
; THE 'CHANNEL K FLAG' ROUTINE
; ---------------------------
;   routine to set flags for lower screen/keyboard channel.


CHAN_K
;;;        SET   0,(IY+$02)       ; update TV_FLAG  - signal lower screen in use

           CALL  SIG_L_SCR        ;+ set 0,(iy+$02) as a 3-byte call.

           RES   5,(IY+$01)       ; update FLAGS    - signal no new key ??

           SET   4,(IY+$30)       ; update FLAGS2   - signal K channel in use

           JR    CHAN_S_1         ; forward to CHAN-S-1 for indirect exit

; ---------------------------
; THE 'CHANNEL S FLAG' ROUTINE
; ---------------------------
;   routine to set flags for upper screen channel.

CHAN_S    RES   0,(IY+$02)       ; TV_FLAG  - signal main screen in use

CHAN_S_1  RES   1,(IY+$01)       ; update FLAGS  - signal printer not in use

          JP    TEMPS            ; jump back to TEMPS and exit via that
                                 ; routine after setting temporary attributes.
; ----------------------------
; THE 'CHANNEL P FLAG' ROUTINE
; ----------------------------
;   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    SET   1,(IY+$01)      ; update FLAGS  - signal printer in use

          RET                   ; return


; --------------------------
; THE 'ONE SPACE' SUBROUTINE
; --------------------------
;   This routine WAS called once only to create a single space
;;; ONE_SPACE LD    BC,$0001        ; create space for a single character.


MK_RM_EL  LD    HL,($5B59)      ; fetch E_LINE to HL.

MK_RM_DHL DEC   HL              ; point to location before.

; -----------------------
; THE 'MAKE ROOM' ROUTINE
; -----------------------
;   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 address this location.
;   Note. It used to point to the location before this.

MAKE_ROOM PUSH  HL              ; save the address pointer.

          CALL  TEST_ROOM       ; routine TEST-ROOM checks if room
                                ; exists and generates an error if not.
          POP   HL              ; restore the address pointer.

          CALL  POINTERS        ; routine POINTERS updates the
                                ; dynamic memory location pointers.
                                ; DE now holds the old value of STKEND.

          LD    HL,($5B65)      ; 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.

          INC   HL              ;+ New - as suggested by James Smith.

          RET                   ; return with new area ready to be populated.

;   Note. HL now points to first location of new area, and DE to last of new
;   locations.

; -------------------------
; THE 'POINTERS' SUBROUTINE
; -------------------------
;   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  PUSH  AF               ; preserve accumulator.
          PUSH  HL               ; put pos pointer on stack.

          LD    HL,$5B4B         ; address VARS the first of the
          LD    A,$0E            ; fourteen variables to consider.

PTR_NEXT  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,PTR_DONE      ; forward, if var before pos, to PTR-DONE

          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  INC   HL               ; address next system variable.
          DEC   A                ; decrease counter.
          JR    NZ,PTR_NEXT      ; back, if more, to PTR-NEXT

          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.

; ------------------------------------
; THE 'COLLECT LINE NUMBER' SUBROUTINE
; ------------------------------------
;   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.
;   Note. any two zero bytes in ROM will do for a line zero.

;;; LINE_ZERO DEFB  $00, $00     ; dummy line number used for direct commands
;;;                              ; Note. space character is now used instead.
```

```
LINE_NO_A EX    DE,HL           ; fetch the previous line to HL and set
          LD    DE,LINE_ZERO    ; set DE to word zero pointer should HL also
                                ; fail.

;   -> The Entry Point.

LINE_NO   LD    A,(HL)          ; fetch the high byte - max $2F
          AND   $C0             ; mask off the invalid bits.
          JR    NZ,LINE_NO_A    ; 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.

; ---------------------------------
; THE 'CREATE BC SPACES' SUBROUTINES
; ---------------------------------
;+  This was formerly a restart but is now called as a subroutine
;+  to free up the RST 30 for error handling.

BC_SPACE1 LD    C,1             ;+ Creates one space - the most popular option.

BC_SPACE0 LD    B,0             ;+ Only C need be specified.

BC_SPACES PUSH  BC              ; save number of spaces.
          LD    HL,($5B61)      ; fetch WORKSP.
          PUSH  HL              ; save address of workspace.

RESERVE   LD    HL,($5B63)      ; STKBOT first location of calculator stack

          CALL  MK_RM_DHL       ;+ routine MAKE_ROOM adjusting HL

;;;       DEC   HL              ; make one less than new location
;;;       CALL  MAKE_ROOM       ; routine MAKE-ROOM creates the room.
;;;       INC   HL              ; address the first new location

          INC   HL              ; advance to second
          POP   BC              ; restore old WORKSP
          LD    ($5B61),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.

; ---------------------------
; THE 'SET MINIMUM' SUBROUTINE
; ---------------------------
;   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
;   resets 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.
;   Although not intended as such, this routine used to clear up any imbalance
;   in the calculator stack.

SET_MIN   LD    HL,($5B59)      ; fetch E_LINE
```

```
          LD    (HL),$0D        ; insert carriage return
          LD    ($5B5B),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    ($5B61),HL      ; start of WORKSP

;   This entry point is used prior to input and prior to the execution,
;   or parsing, of each statement.

SET_WORK  LD    HL,($5B61)      ; fetch WORKSP value
          LD    ($5B63),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   LD    HL,($5B63)      ; fetch STKBOT value
          LD    ($5B65),HL      ; and place in STKEND.

;;;       PUSH  HL              ; perhaps an obsolete entry point.

          LD    HL,$5B92        ; normal location of MEM-0
          LD    ($5B68),HL      ; is restored to system variable MEM.

;;;       POP   HL              ; saved value not required.

          RET                   ; return.

; ----------------------
; THE 'REC-EDIT' ROUTINE
; ----------------------
;   This is legacy code from the ZX80/ZX81 and it is not used in this ROM.
;   That task, in fact, is performed here by the dual-area routine CLEAR-SP.

;;; REC-EDIT
;;; L16D4:    LD    DE,($5B59)      ; fetch start of edit line from E_LINE.
;;;           JP    RECLAIM_1       ; jump forward to RECLAIM-1.

; ------------------------------
; THE 'TABLE INDEXING' SUBROUTINE
; ------------------------------
;   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.

;   -> The Entry Point.

INDEXER_0 LD    C,A             ;+ Replaces 4 similar instructions
          LD    B,$00           ;+ A useful return value.

INDEXER_1 INC   HL              ; Address the next pair of values.


INDEXER   LD    A,(HL)          ; Fetch the first byte of pair
          AND   A               ; Is it the end-marker ?
;;;       RET   Z               ; Return, if so, with carry reset.
          JR    NZ,INDEXER_2    ;
          LD    A,C             ;
          RET                   ;

INDEXER_2 CP    C               ; Is it the required character ?
          INC   HL              ; Address next location.
          JR    NZ,INDEXER_1    ; Back, if no match, to INDEXER-1

          LD    C,(HL)          ;
          ADD   HL,BC           ;

          SCF                   ; 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.

; --------------------------
; THE 'CLOSE STREAM' 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.
;   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     CALL  STR_DATA        ; 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 is closed'.

          JR    NZ,CLOSE_OK     ;+ Continue if stream is open.

REPORT_S  RST   30H             ;+ ERROR-1
          DEFB  $1B             ;+ 'Stream is closed'

CLOSE_OK  CALL  CLOSE_2         ; routine CLOSE-2 will  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        ;

          LD    DE,$A4E4        ;+ the number of bytes from stream 4 to $10000

          EX    DE,HL           ; transfer the offset to HL and the STRMS data
                                ; pointer to the DE register.
          ADD   HL,DE           ; add the offset to the data pointer.

          JR    C,CLOSE_1       ; forward, if a non-system stream, to CLOSE_1

;   proceed with a negative result offset now 12 (was 14).
```

```
;;;       LD    BC,INIT_STRM +14; prepare the address of the byte after streams.

          LD    BC,INIT_STRM +12;+ prepare the address of the byte after the
                                ;+ initial stream data in ROM.

          ADD   HL,BC           ; index into the ROM data table with negative
                                ; value.
          LD    C,(HL)          ; Read low-order byte from ROM to C
          INC   HL              ; address next ROM location.
          LD    B,(HL)          ; Read high-order byte from ROM to B.

;   For streams 0 - 2 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 3 - 15, the BC register holds zero, and the entry is blanked.

CLOSE_1   EX    DE,HL           ; Transfer 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
; ------------------------
;   This routine finds the offset to a special closing routine,
;   in this ROM and within 256 bytes of the close stream look up table that
;   reclaims any buffers associated with a stream.
;   IN: HL=address in STRMS BC=offset from CHANS to 2nd byte of channel

CLOSE_2   PUSH  HL              ; * save address of stream data pointer
                                ; in STRMS on the machine stack.
          LD    HL,($5B4F)      ; fetch CHANS address to HL
          ADD   HL,BC           ; add the offset to address the second byte

          DEC   HL              ; point to first byte.

          LD    ($5B51),HL      ;+ Update system variable CURCHL
                                ;+ While we have the channel in the register,
                                ;+ make it 'current; as we may have to flush.

          PUSH  HL              ;+ copy to IX register.
          POP   IX              ;+

          LD    D,B             ;+ Save offset in DE.
          LD    E,C             ;+

          LD    A,(IX+$04)      ;+ pick up the channel letter in A.

          LD    HL,CL_STR_LU-1  ; address: cl-str-lu in ROM.

          CALL  INDEXER_0       ; routine INDEXER uses the code to get
                                ; the 8-bit offset from the current point to
                                ; the address of the Closing Routine in ROM.

;;;       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.

          JP    (HL)            ; jump to that routine.

; ---
```

```
          DEFB  0,0,0,0         ;+ ballast
TAG6:     DEFB  0,0,0,0         ;+ ballast



; -------------------------------
; 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 DEFB  'K', CLOSE_E-$-1; offset to CLOSE_E
          DEFB  'S', CLOSE_E-$-1; offset to CLOSE_E
          DEFB  'P', CLOSE_P-$-1;+ offset to CLOSE_P
          DEFB  'B', CLOSE_A-$-1;+ offset to CLOSE_A
          DEFB  'T', CLOSE_A-$-1;+ offset to CLOSE_A
          DEFB  'N', CLOSE_N-$-1;+ offset to CLOSE_N

; ------------------------------
; THE 'CLOSE PRINTER STREAM' SUBROUTINE
; ------------------------------
;   The last data block must be sent as an EOF record.

CLOSE_P   CALL  COPY_BUFF       ; send EOF record.
          JR    CLOSE_A         ; skip forward to generic CLOSE_A routine.

; ------------------------------
; THE 'CLOSE NETWORK' SUBROUTINE
; ------------------------------
;   The last data block must be sent as an EOF record except when T_ADDR_hi
;   indicates that 'CLEAR #' has been used.  In this case the network buffer
;   is simply closed losing its contents.

CLOSE_N   BIT   6,(IY+$3B)      ; Test T_ADDR_hi

          CALL  Z,SEND_NEOF     ; send EOF record.


; --------------------------
; THE 'CLOSE ALL' SUBROUTINE
; --------------------------
;+  Initially, removed the 264 byte "P" channel and the ZX printer buffer.
;+  In fact this routine is generic and will remove any channel.

CLOSE_A   PUSH  DE              ; Save CHANS offset.
          PUSH  IX              ;
          POP   HL              ; HL addresses the start of the channel.

          LD    C,(IX+$05)      ;
          LD    B,(IX+$06)      ; BC contains length.

          PUSH  BC              ; Preserve bytes to reclaim.

          CALL  RECLAIM_2       ; Routine RECLAIM-2

          POP   BC              ; Restore reclaimed byte count.

;   Any open streams that point to channels beyond that deleted (offset =DE)
;   will have to have offsets reduced by the amount reclaimed (length = BC)
;   This is similar to REST-STRM in Interface 1
```

```
          LD    A,$10           ; 16 user streams

          LD    HL,$5B16        ; Start of user streams in sysvars.

NEXT_STRM LD    ($5B5F),HL      ; Save current pointer in X_PTR

          LD    E,(HL)          ; Fetch displacement for current stream.
          INC   HL              ;
          LD    D,(HL)          ;

          POP   HL              ; restore chans offset
          PUSH  HL              ; push the value again.



          AND   A               ; clear carry

          SBC   HL,DE           ; compare by subtraction.

          JR    NC,UPD_POINT    ; forward if before deleted channel to do a
                                ; dummy update as provides easier pathing.

          EX    DE,HL           ; transfer current displacement to HL.
          AND   A               ; clear carry.

          SBC   HL,BC           ; reduce displacement by amount deleted.

          EX    DE,HL           ; transfer new displacement to DE.

UPD_POINT LD    HL,($5B5F)      ; Fetch STRMS pointer from X_PTR

          LD    (HL),E          ;
          INC   HL              ;
          LD    (HL),D          ;
          INC   HL              ;

          DEC   A               ; Decrement stream counter.
          JR    NZ,NEXT_STRM    ; loop back till all sixteen tested.

          POP   HL              ; balance stack

;   Note. as long as X_PTR points to somewhere harmless it need not be set to
;   a zero value.  Interface 1 mistakenly sets the low byte anyway.

; --------------------------
; THE 'CLOSE END' SUBROUTINE
; --------------------------
;   The close stream routines have no ancillary actions to perform with regard
;   to 'K' and 'S'.

CLOSE_E   POP   HL              ; * now just restore the stream data pointer

          RET                   ; in STRMS and return.

; ----------------------------
; THE 'STREAM DATA' SUBROUTINE
; ----------------------------
;   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  CALL  FIND_INT1       ; Routine FIND-INT1 fetches parameter to A
                                ; setting B to zero.

          CP    $10             ; Is it less than 16d ?
          JR    C,STR_DATA1     ; Skip forward, if so, to STR-DATA1

;   Note. the unimplemented ERASE and MOVE commands also now point here.

REPORT_O  RST   30H             ; ERROR-1
          DEFB  $17             ; Error Report: Invalid stream

; ---

STR_DATA1 ADD   A,$03           ; add the offset for the three system streams.
                                ; range 00 - 15d becomes 3 - 18d.
          RLCA                  ; double the offset as there are two bytes per
                                ; stream - now 06 - 36d
          LD    HL,$5B10        ; address STRMS - the start of the streams
                                ; data area in the system variables.
          LD    C,A             ; transfer the low byte to C.

          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)          ; Fetch low byte of displacement to C.
          INC   HL              ; Address next.
          LD    B,(HL)          ; Fetch high byte of displacement to B.
          DEC   HL              ; Step back to leave HL pointing to STRMS
                                ; data entry.

          LD    A,B             ;+ Test for zero now
          OR    C               ;+ as a common return condition.

          RET                   ; Return with CHANS displacement in BC
                                ; and address of stream data entry in HL.

; --------------------
; THE 'OPEN #' COMMAND
; --------------------
;   This command has been changed from CLASS_03 to CLASS_05
;   Command syntax example: OPEN #6,"p"    and  OPEN #7,"n";64

OPEN      LD    A,$FF           ; set all bits of A as station indicator.
          EX    AF,AF'          ; preserve as an invalid network station.

          CALL  EXPT_SEP        ; is next character a separator ?
          JR    NZ,CHK_O_END    ; forward if not to check end - no station.

;   If there was a separator then the network station comes next.

;;;       RST   20H             ; NEXT_CHAR
;;;       CALL  EXPT_1NUM       ; routine EXPT-1NUM checks for number
;;;       CALL  CHECK_END       ; as in OPEN #9,"n",64

          CALL  CHK_END_1       ;+ above three routines combined.

;   It is runtime so the network station is on the stack.

          CALL  FIND_INT1       ;+ routine FIND-INT1 fetches parameter to A.
          EX    AF,AF'          ;+ preserve in alternate register

;   It is simpler to pass through check than jump over it.

CHK_O_END CALL  CHECK_END       ; finish if checking syntax.

;   In runtime, the channel code entry is on the calculator stack with the next
;   value containing the stream identifier.  They have to be swapped.

          RST   28H             ;; FP-CALC    ;s,c.
          DEFB  $01             ;;exchange    ;c,s.
          DEFB  $38             ;;end-calc

          CALL  STR_DATA        ; routine STR-DATA fetches the stream off
                                ; the stack and returns with the CHANS
                                ; displacement in BC and HL addressing
                                ; the STRMS data entry.  The zero flag will be
                                ; set if the stream is closed.

;;;       LD    A,B             ; test for zero which
;;;       OR    C               ; indicates the stream is closed.

          JR    Z,OPEN_1        ; skip forward, if closed, to OPEN-1

;   If it is an open system channel, then it can be re-attached.

          EX    DE,HL           ; save STRMS address in DE.
          LD    HL,($5B4F)      ; fetch CHANS.
          ADD   HL,BC           ; add the offset to address the second
                                ; byte of the channel.

;;;       INC   HL              ;
;;;       INC   HL              ;
;;;       INC   HL              ;
;;;       LD    A,(HL)          ;

          CALL  NUMBER_3        ;+ add 3 to hl and fetch A comparing to 'K'.


;   A new channel can replace an existing one only if the existing channel
;   is not associated with a dynamic buffer.  Otherwise the buffer would be
;   left hanging.  The channel to be replaced is checked against a list of
;   those that are not dynamic.
;   Note.  If the channel is dynamic then it must be closed and then opened.
;   This manual closure may involve re-instating an initial channel.

          EX    DE,HL           ; bring back the STRMS pointer.

;;;       CP    $4B             ; is it 'K' ?
          JR    Z,OPEN_1        ; forward, if so, to OPEN-1

;;;       CP    $53             ; is it 'S' ?
;;;       JR    Z,OPEN_1        ; forward, if so, to OPEN-1

          CP    $53             ; is it 'S' ?   (was 'P')
          JR    NZ,REPORT_O     ; back, if not, to REPORT-O
                                ; 'Invalid stream'.


;   Continue if one of the upper-case letters was found and rejoin here from
;   above if the stream was already closed.

OPEN_1    CALL  OPEN_2          ; routine OPEN-2 opens the stream.
```

```
;   It now remains to update the STRMS variable.

          JP    PO_CH_2         ;+ jump to similar code to that below. (JS)

;;;       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.

; -----------------------
; THE 'OPEN_2' SUBROUTINE
; -----------------------
;   As well as creating buffers, this routine also sets flags.
;   Note. that on the original Spectrum the network station was passed in
;   after the "N" channel identifier.  This made syntax checking easy but if
;   the station identifier was numeric it is a departure from the rule that
;   any number can be replaced by a numeric expression.  The station identifier
;   could have been a character.

OPEN_2    PUSH  HL              ; * save the STRMS data entry pointer throughout

          CALL  EXPT_SPEC       ;+ NEW routine fetches a one character specifier
                                ;+ to A

;;;       CALL  STK_FETCH       ; routine STK-FETCH now fetches the paremeters.
;;;       LD    A,B             ; test that it is not
;;;       OR    C               ; the null string.
;;;       JR    NZ,OPEN_3       ; skip forward to OPEN-3 with 1 character or
;;;                             ; more!!!!
;;; REPORT_F RST   30H          ; ERROR-1
;;;       DEFB  $0E             ; Error Report: Invalid file name
;;; OPEN_3    PUSH  BC          ; Save the length of the string.
;;;       LD    A,(DE)          ; Pick up the first character.

          AND   $DF             ; Make it upper-case.

;;;       LD    C,A             ; Place channel specifier in C.
          LD    HL,OP_STR_LU-1  ; Address: op-str-lu is addressed.

          CALL  INDEXER_0       ; Routine INDEXER will search for the letter.

          JR    NC,REPORT_F     ; Forward, if not found, to REPORT-F
                                ; 'Invalid filename'

;;;       LD    C,(HL)          ; Fetch the displacement to opening routine.
;;;       LD    B,$00           ; prepare to add.
;;;       ADD   HL,BC           ; Now form address of the opening routine.
;;;       POP   BC              ; Restore the length of the string.

          JP    (HL)            ; Jump forward to the relevant routine.

; -------------------------------
; THE '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.
;   Note. The table has been re-arranged so that those without buffers
;   come last providing two look-up tables in one.

OP_STR_LU
          DEFB  'P', OPEN_P-$-1 ;     offset to OPEN-P
          DEFB  'N', OPEN_N-$-1 ;+    offset to OPEN_N
          DEFB  'B', OPEN_B-$-1 ;+    offset to OPEN-B
          DEFB  'T', OPEN_T-$-1 ;+    offset to OPEN-T

NOBUF_LU  DEFB  'K', OPEN_K-$-1 ;     offset to OPEN-K
          DEFB  'S', OPEN_S-$-1 ;     offset to OPEN-S

          DEFB  $00             ;     end-marker.

; --------------------------------
; THE 'STREAM OPENING' SUBROUTINES
; --------------------------------
;   Note. That was then, this is now.
;   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.

; -----------------------
; THE 'OPEN-K' SUBROUTINE
; -----------------------
;   Open Keyboard channel.
;   Note. the full 16-bit offset is now supplied in DE.

;;; OPEN_K LD   E,$01           ; offset to channel 'K'.

OPEN_K    LD    DE,$0001        ;+ 01 is offset to 2nd byte of channel 'K'.
          JR    OPEN_END        ; forward to OPEN-END

; -----------------------
; THE 'OPEN-S' SUBROUTINE
; -----------------------
;   Open Screen channel.
;   Note. the full 16-bit offset is now supplied in DE.

;;; OPEN_S LD   E,$06           ; offset to channel 'K'.

OPEN_S    LD    DE,$0006        ;+ 06 is offset to 2nd byte of channel 'S'
          JR    OPEN_END        ; forward to OPEN-END

; -----------------------
; THE 'OPEN-P' SUBROUTINE
; -----------------------
;   Open Printer channel.

OPEN_P    LD    IX,PCHAN_DAT    ;+ point to the channel data.

          JR    OPEN_ALL        ;+ forward to generic opening routine.

; -----------------------
; THE 'OPEN-B' SUBROUTINE
; -----------------------
;   Open B RS232 channel

OPEN_B    LD    IX,BCHAN_DAT    ;+ point to the channel data.

          JR    OPEN_ALL        ;+ forward to generic opening routine.

; -----------------------
; THE 'OPEN-T' SUBROUTINE
; -----------------------
;   Open T RS232 channel

OPEN_T    LD    IX,TCHAN_DAT    ;+ point to the channel data.
```

```
          JR    OPEN_ALL        ;+ forward to generic opening routine.


; ----------------------------------------
; THE 'OPEN PERMANENT "N" CHANNEL' ROUTINE
; ----------------------------------------
;   e.g. OPEN #9,"N";2

OPEN_N    LD    IX,NCHAN_DAT    ;+

; ----------------------
; THE 'OPEN_ALL' ROUTINE
; ----------------------
;+  Generic Channel Opening Routine.
;+  DE still points to string

OPEN_ALL  LD    HL,($5B53)      ; Set pointer from PROG
          LD    C,(IX+$05)      ; length lo.
          LD    B,(IX+$06)      ; length hi.

          CALL  MK_RM_DHL       ;+ routine MAKE_ROOM decrementing HL first.

;   HL points to the 1st location, DE to last new location, BC is zero

;;;       INC   HL              ; HL points to start of new channel

          PUSH  HL              ; (*) Save channel pointer.

          EX    DE,HL           ; Transfer HL to DE.

          PUSH  IX              ; Transfer ROM data pointer
          POP   HL              ; to HL.

          LD    C,(IX-$01)      ; Find number of bytes in ROM

          LDIR                  ; Block copy the channel data.

;   Note. a call to clear the ZX Printer buffer is required here.
;   but can be done directly.

          LD    A,(IX+$04)      ;

          CP    'P'             ;
          JR    Z,P_BLANK       ; Forward, if printer, to P_BLANK


          CP    'N'             ; is it network ?
          JR    NZ,OFFSET       ;


          EX    AF,AF'          ; save device letter, bring back station.

          CP    $41             ; compare to 64
          JP    NC,REPORT_B     ; forward, if over, to report
                                ; 'Integer out of range'

          LD    (DE),A          ; set channel variable NCIRIS
          INC   DE              ; address own station number.


          LD    A,($5BBC)       ; fetch global station number from sysvar NTSTAT
          LD    (DE),A          ; update channel variable NCSELF
          EX    AF,AF'          ; save again.
          INC   DE              ; point to next location


;   Note. the network buffer does not have to be cleared. As long as we set
;   the other channel variables to zero that is sufficient so use the same
;   routine as is used for the ZX Printer buffer which does all but 4.


P_BLANK   LD    H,D             ;
          LD    L,E             ;
          INC   DE              ;

          LD    (HL),B          ; Blank first location
          DEC   C               ; set count to 255 decimal or whatever.

          LDIR                  ;

;   now calculate offset from CHANS

OFFSET    LD    HL,($5B4F)      ; Address CHANS

          POP   DE              ; (*) Restore the channel pointer

          EX    DE,HL           ;
          INC   HL              ; the second byte is used.
          AND   A               ; prepare to subtract
          SBC   HL,DE           ; result is in HL
          EX    DE,HL           ; transfer offset to DE

;;;       POP   BC              ; Restore length of string.

          CP    'N'
          JR    Z,OPEN_END2     ; skip the length test


; ----------------------
; THE 'OPEN END' ROUTINE
; ----------------------

;;; OPEN_END  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.
; OPEN_IFN  JP    NZ,REPORT_F     ; back, if not, to REPORT-Fb
;;;                             ; 'Invalid file name'
;;;       LD    D,A             ; load D with zero to form the displacement.

;   It used to go like that and now it goes like this...

OPEN_END  EX    AF,AF'          ; station number - should be $FF
          INC   A               ; test for $FF

OPEN_END2 POP   HL              ; * restore the saved STRMS pointer.

          RET   Z               ; return to update STRMS entry thereby
                                ; signaling stream is open.

;   A parameter has been supplied for a channel that does not require one
;   e.g. OPEN #6,"t",78

REPORT_F  RST   30H             ; ERROR-1
          DEFB  $0E             ; Error Report: Invalid file name

; ----------------------
; THE '"P" CHANNEL DATA'
; ----------------------
;   The eight bytes "P" channel descriptor.

          DEFB  $08             ;+ length of channel data

PCHAN_DAT DEFW  PRINT_OUT       ;+ PRINT-OUT
          DEFW  REPORT_J        ;+ REPORT-J
          DEFB  'P'             ;+ Letter as in standard ROM
          DEFW  $0108           ;+ Length of channel including printer buffer.
          DEFB  $21             ;+ P_POSN (IX+$07)

; ----------------------
; THE '"B" CHANNEL DATA'
; ----------------------
;   The seven bytes "B" channel descriptor. Maybe stick the 2-byte buffer here.

          DEFB  $07             ;+ length of channel data

BCHAN_DAT DEFW  BCHAN_OUT       ;+ BCHAN_OUT
          DEFW  BCHAN_IN        ;+ BCHAN_IN
          DEFB  'B'             ;+ Letter
          DEFW  $0007           ;+ Length of channel

; ----------------------
; THE '"T" CHANNEL DATA'
; ----------------------
;   The seven bytes "T" channel descriptor.

          DEFB  $07             ;+ length of channel data

TCHAN_DAT DEFW  TCHAN_OUT       ;+ TCHAN_OUT
          DEFW  TCHAN_IN        ;+ TCHAN_IN
          DEFB  'T'             ;+ Letter
          DEFW  $0007           ;+ Length of channel

; ----------------------
; THE '"N" CHANNEL DATA'
; ----------------------
;   The seven bytes "N" channel descriptor.

          DEFB  $07             ;+ length of data

NCHAN_DAT DEFW  NCHAN_OUT       ;+ NCHAN-OUT
          DEFW  NCHAN_IN        ;+ NCHAN_IN
          DEFB  $4E             ;+ character "N"
          DEFW  $0110           ;+ length

;   The other channel variables for network are defaulted to zero. They are -
;
;   1     NCIRIS        IX+$07  ; The destination station number.
;   1     NCSELF        IX+$08  ; This SPECTRUM's station number.
;   2     NCNUMB        IX+$09  ; The block number.
;   1     NCTYPE        IX+$0B  ; The packet type code ... 0 data, 1 EOF.
;   1     NCOBL         IX+$0C  ; The number of bytes in the data block.
;   1     NCDCS         IX+$0D  ; The data checksum.
;   1     NCHCS         IX+$0E  ; The header checksum.
;   1     NCCUR         IX+$0F  ; The position of the last character taken from
;                               ; the buffer.
;   1     NCIBL         IX+$10  ; The number of bytes in the input buffer.
;   255   NCB           IX+$11  ; A 255 byte data buffer.
; *************************
; **  THE RS232  ROUTINES **
; *************************
;
; ---------------------------------------
; THE '"T" CHANNEL INPUT SERVICE' ROUTINE
; ---------------------------------------
;   The text channel input is limited to 7 bits so use the binary channel input
;   and reset the most significant bit.

TCHAN_IN  CALL  BCHAN_IN        ; routine BCHAN-IN

          RES   7,A             ; reset the MSB.

          RET                   ; Return.


; ---------------------------------------
; THE '"B" CHANNEL INPUT SERVICE' ROUTINE
; ---------------------------------------
;   For serial input a two-byte buffer SER_FL is used.
;   Sometimes 16 bits are received at a time so the second byte is stored
;   here.

BCHAN_IN  LD    HL,$5BBE        ; Point to the SER_FL system variable.
          LD    A,(HL)          ; Fetch a byte.
          AND   A               ; Test for zero which signals no stored byte.

          JR    Z,REC_BYTE      ; Forward, if so, to REC-BYTE.

          LD    (HL),$00        ; else signal taking the stored byte.
          INC   HL              ; Point to the stored byte.
          LD    A,(HL)          ; Load it to the accumulator.
          SCF                   ; Signal success by setting carry.

          RET                   ; Return.

; ---

REC_BYTE  CALL  TEST_BRK        ; Routine TEST-BRK tests the BREAK keys.

          DI                    ; Disable Interrupts

          LD    A,($5BBD)       ; Fetch I/O colour from IOBORD system variable.
          OUT   ($FE),A         ; Change the border to show activity.

;   The value for

          LD    DE,($5BBA)      ; fetch value from BAUD system variable.
          LD    HL,$0320        ; set counter to 800 decimal.

          LD    B,D             ; copy BAUD value
          LD    C,E             ; to BC register.

          SRL   B               ;  0 -> 76543210 -> C  Halve the value
          RR    C               ;  C -> 76543210 -> C

          LD    A,$FE           ; Make CTS (Clear To Send) high.
          OUT   ($EF),A         ;

;   The other device, VTX modem, BBC computer, PC, Spectrum etc. will now send
;   the data.

READ_RS   IN    A,($F7)         ; bit 7 is TXdata serial data
```

```
          RLCA                  ; rotate into carry.
          JR    NC,TST_AGAIN    ; forward to TST-AGAIN if TXdata low

          IN    A,($F7)         ; repeat the test 3 times
          RLCA                  ;
          JR    NC,TST_AGAIN    ; forward to TST-AGAIN

          IN    A,($F7)         ;
          RLCA                  ;
          JR    NC,TST_AGAIN    ; forward to TST-AGAIN

          IN    A,($F7)         ;
          RLCA                  ;
          JR    C,START_BIT     ; forward, if high for four tests, to START-BIT


TST_AGAIN DEC   HL              ; decrement the 800 counter.
          LD    A,H             ; test for
          OR    L               ; zero.
          JR    NZ,READ_RS      ; back, if not, to READ-RS

          PUSH  AF              ; (*) Save the zero failure flag

          LD    A,$EE           ; make CTS (Clear To Send) line low.
          OUT   ($EF),A         ;

          JR    WAIT_1          ; forward to WAIT-1

; ---

;   The branch was here when TXdata was high for 4 tests.

START_BIT LD    H,B             ; Load HL with halved BAUD value.
          LD    L,C             ;

          LD    B,$80           ; Load B with start bit.

          DEC   HL              ; reduce counter by the time for the 4 tests.
          DEC   HL              ;
          DEC   HL              ;

SERIAL_IN ADD   HL,DE           ; Add the BAUD value.
          NOP                   ; (4) timing value.

BD_DELAY  DEC   HL              ; ( 6) Delay for 26 * BAUD
          LD    A,H             ; ( 4)
          OR    L               ; ( 4)
          JR    NZ,BD_DELAY     ; (12) back to BD-DELAY

          ADD   A,$00           ; (7) wait
          IN    A,($F7)         ; Read a bit
          RLCA                  ; rotate bit 7 to carry.
          RR    B               ; pick up carry in B.
          JR    NC,SERIAL_IN    ; loop back, if no start bit, to SERIAL-IN

;   After looping eight times, the start bit will pass through and B will
;   contain a received byte.

          LD    A,$EE           ; Send CTS line low.
          OUT   ($EF),A         ;

          LD    A,B             ; transfer received byte to A.
          CPL                   ; complement.
          SCF                   ; signal success.
          PUSH  AF              ; (*) push success flag

;   The success and failure (time out) paths converge here with HL holding zero.

WAIT_1    ADD   HL,DE           ; transfer DE (BAUD) to HL.

WAIT_2    DEC   HL              ; ( 6) Delay for stop bit.
          LD    A,L             ; ( 4)
          OR    H               ; ( 4)
          JR    NZ,WAIT_2       ; (12/7) back to WAIT-2

;   Register HL is now zero.

          ADD   HL,DE           ; HL = 0 + BAUD
          ADD   HL,DE           ; HL = 2 * BAUD
          ADD   HL,DE           ; HL = 3 * BAUD

;   The device at the other end of the cable may send a second byte even though
;   CTS is low.

T_FURTHER DEC   HL              ; decrement counter.
          LD    A,L             ; Test for
          OR    H               ; zero.
          JR    Z,END_RS_IN     ; forward, if no 2nd byte, to END-RS-IN

          IN    A,($F7)         ; Read TXdata.
          RLCA                  ; test bit.
          JR    NC,T_FURTHER    ; back, if none, to T-FURTHER

;   As with first byte, TXdata must be high for four tests

          IN    A,($F7)         ;
          RLCA                  ;
          JR    NC,T_FURTHER    ; back to T-FURTHER

          IN    A,($F7)         ;
          RLCA                  ;
          JR    NC,T_FURTHER    ; back to T-FURTHER

          IN    A,($F7)         ;
          RLCA                  ;
          JR    NC,T_FURTHER    ; back to T-FURTHER

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

          LD    H,D             ;
          LD    L,E             ;
          SRL   H               ;
          RR    L               ;
          LD    B,$80           ;
          DEC   HL              ;
          DEC   HL              ;
          DEC   HL              ;

SER_IN_2  ADD   HL,DE           ;
          NOP                   ; timing

BD_DELAY2 DEC   HL              ;
          LD    A,H             ;
          OR    L               ;
          JR    NZ,BD_DELAY2    ; back to BD-DELAY2

          ADD   A,$00           ;
          IN    A,($F7)         ;
          RLCA                  ;
          RR    B               ;
          JR    NC,SER_IN_2     ; back to SER-IN-2

;   The start bit has been pushed out and B contains the second received byte.

          LD    HL,$5BBE        ; Address the SER_FL system variable.
          LD    (HL),$01        ; signal there is a byte in next location
          INC   HL              ; address that location
          LD    A,B             ; transfer byte to A.
          CPL                   ; complement.
          LD    (HL),A          ; and insert in second byte of serial flag.

END_RS_IN CALL  BORD_REST       ; routine BORD-REST restores the normal border.

          POP   AF              ; restore byte and flags
                                ; (either 0 and NC or received byte and carry).

;;;       EI                    ; Enable Interrupts

          RET                   ; Return.

; --------------------------------
; THE '"T" CHANNEL OUTPUT' ROUTINE
; --------------------------------
;   The text channel output routine is able to list programs and, when printing,
;   takes correct action with TAB values etc. I think.
;   Note. The "t" channel can be tested on the RealSpec emulator as follows -
;   1) Assemble this file and note down addresses of BCHAN_IN and BCHAN_OUT
;   2) Select this ROM [F3] and serial interface [ALT F3]
;   3) Select output to a file.
;   3) Arrow down to bottom and supply IO addresses e.g. I 184E 19B0
;   4) Session "t" channel output will appear in file SERIAL.BIN


TCHAN_OUT CP    $A5             ; Compare to 'RND' - first token
          JR    C,NOT_TOKEN     ; Forward, if less, to NOT-TOKEN

          SUB   $A5             ; Reduce token to range $00-$5A

          JP    PO_TOKENS       ; Routine PO_TOKENS recursively prints tokens

;;;       RET                   ; Return.

; ---

NOT_TOKEN LD    HL,$5B3B        ; Address the FLAGS system variable.
          RES   0,(HL)          ; update FLAGS - allow for leading space.
          CP    $20             ; compare character to space
          JR    NZ,NOT_LEAD     ; forward, if not, to NOT-LEAD

          SET   0,(HL)          ; update FLAGS - signal suppress leading space.

;   The mosaic graphics and UDGs are output as '?' as also is (c) copyright.

NOT_LEAD  CP    $7F             ; compare to copyright symbol. (DEL in ASCII)
          JR    C,NOT_GRAPH     ; forward, if less, to NOT-GRAPH

          LD    A,$3F           ; Output CHR$(127) and graphics as '?'

NOT_GRAPH CP    $20             ; compare against space.
          JR    C,CTRL_CODE     ; forward, if less, to CTRL_CODE
          PUSH  AF              ; preserve character.

          INC   (IY+$7D)        ; increment width   WIDTH_lo
          LD    A,($5BB8)       ; load A with limit WIDTH_hi
          CP    (IY+$7D)        ; compare to width  WIDTH_lo
          JR    NC,EMIT_CH      ; forward, if width less or equal, to EMIT-CH

          CALL  TAB_SETZ        ; routine TAB-SETZ sets iy+$7D to zero and
                                ; emits CR/LF.

;;;       LD    (IY+$7D),$01    ; set WIDTH_lo to one - for current character.
          INC   (IY+$7D)        ;+ set WIDTH_lo to one - for current character.

EMIT_CH   POP   AF              ; restore the unprinted character.
          JR    BCH_OUT         ; jump, indirectly, to BCHAN-OUT

; ---

;   The branch was here with control codes.

CTRL_CODE CP    $0D             ; is character a carriage return ?
          JR    NZ,NOT_CR       ; forward, if not, to NOT-CR


TAB_SETZ  LD    (IY+$7D),$00    ; set width WIDTH_lo to zero.

          LD    A,$0D           ; output a CR carriage return.
          CALL  BCHAN_OUT       ; routine BCHAN-OUT

          LD    A,$0A           ; output a LF line feed.
BCH_OUT   JR    BCHAN_OUT       ; jump to BCHAN-OUT

; ---

NOT_CR    CP    $06             ; is character a comma control ?
          JR    NZ,NOT_COMMA    ; forward, if not, to NOT_COMMA

          LD    BC,($5BB7)      ; load BC with width and limit from WIDTH
          LD    E,$00           ; set the space counter to zero.

SPC_COUNT INC   E               ; increment space counter.
          INC   C               ; increment width.
          LD    A,C             ; load A with width.
          CP    B               ; and compare to limit.
          JR    Z,CMM_LP2       ; forward, if at limit, to CMM-LP2


CMM_LOOP  SUB   $08             ; subtract 8 - the tab stop.
          JR    Z,CMM_LP2       ; forward, when zero, to CMM-LP2

          JR    NC,CMM_LOOP     ; back, if higher than 8, to CMM-LOOP

;   The result is less than zero so back to space count.

          JR    SPC_COUNT       ; back to SPC-COUNT

;   The count in E is the spaces to advance to next multiple of eight.

CMM_LP2   CALL  PO_SV_SP        ;+

;;;       PUSH  DE              ; save counter.
;;;       LD    A,$20           ; prepare a space.
;;;       CALL  TCHAN_OUT       ; routine TCHAN-OUT outputs recursively.
;;;       POP   DE              ; restore counter.
```

```
          DEC   E               ; decrement
          RET   Z               ; return when zero.

          JR    CMM_LP2         ; loop back, if not, to CMM-LP2

; ---

NOT_COMMA CP    $16             ; compare to twenty two ('AT')
          JR    Z,TAB_PROC      ; forward, if so, to TAB-PROC

          CP    $17             ; compare to twenty three ('TAB')
          JR    Z,TAB_PROC      ; forward, also, to TAB-PROC

          CP    $10             ; compare to sixteen (INK)
          RET   C               ; return if less.

;   Now store code in TVDATA and alter the current channel to TAB_SERV2

          LD    DE,TAB_SERV2    ; Service routine for ink, paper etc.

          JR    STORE_COD       ; forward to STORE-COD

; ---

TAB_PROC  LD    DE,TAB_SERV     ; addr: TAB-SERV

STORE_COD JP    PO_TV_1         ; jump to similar code for tv output.

;;; STORE_COD LD    ($5B0E),A   ; store control code in TVDATA_lo
;;; ALTER_OUT LD    HL,($5B51)  ; Fetch current channel from CURCHL
;;;           LD    (HL),E      ; Update the low byte of output address.
;;;           INC   HL          ;
;;;           LD    (HL),D      ; Now update the high byte.
;;;           RET               ; Return.

; -------------------------
; THE 'TAB SERVICE ROUTINE'
; -------------------------
;   This deals with TAB and AT control codes.

TAB_SERV  LD    DE,TAB_SERV2    ; addr: TAB-SERV2

          JP    PO_TV_3         ;+ use existing PO routine

;;;       LD    ($5B0F),A       ; store second byte in TVDATA_hi
;;;       JR    ALTER_OUT       ; back to ALTER-OUT

; ---------------------------
; THE 'TAB SERVICE 2' ROUTINE
; ---------------------------
;   Once all the control sequence has been received this routine sorts them.

TAB_SERV2 LD    DE,TCHAN_OUT    ; prepare normal address TCHAN-OUT

          CALL  PO_CHANGE       ;+ routine PO_CHANGE restores it.

          LD    D,A             ; save final character in D.

          LD    A,($5B0E)       ; fetch first character from TVDATA_lo

          CP    $16             ; is it the AT control code ?
          JR    Z,TST_WIDTH     ; forward, with AT, to TST-WIDTH  ????
          CP    $17             ; is it the TAB control code ?
          CCF                   ; if less, e.g. INK the carry is set so reset
                                ; the carry for return condition.
          RET   NZ              ; return if INK - INVERSE and ignore.

;   Continue with TAB.

          LD    A,($5B0F)       ; fetch low byte 0 - 255 from TVDATA_hi
          LD    D,A             ; and store in D, ignoring high byte.

;   The TAB parameter, which is 16 bit is therefore taken mod 256 as only the
;   low byte is used. For AT the column value is used and the line is ignored.

TST_WIDTH LD    A,($5BB8)       ; fetch limit (max width, default 80) to A.
          CP    D               ; compare to column/tab value.

          JR    Z,TAB_MOD       ; forward, if a match, to TAB-MOD  ???

          JR    NC,TABZERO      ; forward if column less than limit to TABZERO


;   The column/tab value is higher than the maximum width so calculate

TAB_MOD   LD    B,A             ; Transfer maximum width to B.
          LD    A,D             ; Transfer column/tab value to A.
          SUB   B               ; subtract a full line of characters.
          LD    D,A             ; and load result back to column/tab.
          JR    TST_WIDTH       ; loop back to TST-WIDTH

; ---

;   The branch was here when the column/tab value was less than the width.

TABZERO   LD    A,D             ; Transfer column/tab to A.
          OR    A               ; Test for zero.
          JR    Z,TAB_SETZ      ; Back, if so, to TAB-SETZ
                                ; to output a carriage return and linefeed.

TABLOOP   LD    A,($5BB7)       ; Fetch current print position from WIDTH_lo
          CP    D               ; Compare to column/tab value.

          RET   Z               ; Return when positions equal.
>>

          CALL  PO_SV_SP        ;+

;;;       PUSH  DE              ; Preserve the column/tab value.
;;;       LD    A,$20           ; Prepare a space.
;;;       CALL  TCHAN_OUT       ; Routine TCHAN-OUT outputs a space.
;;;       POP   DE              ; Restore the column/tab value.

          JR    TABLOOP         ; Back to TABLOOP


; --------------------------------
; THE '"B" CHANNEL OUTPUT' ROUTINE
; --------------------------------
;   The bits of a byte are sent inverted. They are fixed in length and heralded
;   by a start bit and followed by two stop bits.

BCHAN_OUT LD    B,$0B           ; Set bit count to eleven - 1 + 8 + 2.

          CPL                   ; Invert the bits in the character.
          LD    C,A             ; Copy character to C.

          LD    A,($5BBD)       ; select I/O border colour from IOBORD
          OUT   ($FE),A         ; change the border colour.

          LD    A,$EF           ;                 11101111
          OUT   ($EF),A         ; Make CTS (Clear To Send) low.

          CPL                   ; reset bit 0,    00010000

          OUT   ($F7),A         ; Make RXdata low

          LD    HL,($5BBA)      ; fetch value from BAUD system variable
          LD    D,H             ; Copy to DE.
          LD    E,L             ;

BD_DEL_1  DEC   DE              ; ( 6) Wait 26 * BAUD cycles.
          LD    A,D             ; ( 4)
          OR    E               ; ( 4)
          JR    NZ,BD_DEL_1     ; (12) back to BD-DEL-1

TEST_DTR  CALL  TEST_BRK        ; routine TEST-BRK allows user to stop.

          IN    A,($EF)         ; Read the communication port.
          AND   $08             ; isolate DTR (Data Terminal Ready) bit.
          JR    Z,TEST_DTR      ; back, until DTR found high, to TEST-DTR

          SCF                   ; Set carry flag as start bit.

          DI                    ; Disable Interrupts.

;   The bit sending loop.

SER_OUT_L ADC   A,$00           ;  76543210 <- C
          OUT   ($F7),A         ; Send rxdata, start bit

          LD    D,H             ; transfer BAUD to DE.
          LD    E,L             ;


BD_DEL_2  DEC   DE              ; ( 6) Wait for 26 * BAUD
          LD    A,D             ; ( 4)
          OR    E               ; ( 4)
          JR    NZ,BD_DEL_2     ; (12) back to BD-DEL-2

          DEC   DE              ; (6)
          XOR   A               ;     clear rxdata bit
          SRL   C               ;     shift a bit of output byte to carry.
          DJNZ  SER_OUT_L       ; back for 11 bits to SER-OUT-L

;   Note. the last two bits will have been sent reset as C is exhausted.

;;;       EI                    ; Enable Interrupts.
;;;       LD    A,$01           ; set rxdata bit   (inc a)

          INC   A               ; set rxdata bit.

          LD    C,$EF           ; prepare port address.
          LD    B,$EE           ; prepare mask %11101110

          OUT   ($F7),A         ; Send rxdata high.
          OUT   (C),B           ; Send CTS and comms data low - switch off
RS232.
```

```
BD_DEL_3  DEC   HL              ; ( 6) The final 26 * BAUD delay.
          LD    A,L             ; ( 4)
          OR    H               ; ( 4)
          JR    NZ,BD_DEL_3     ; (12) back to BD-DEL-3

; ------------------------------
; THE 'BORDER RESTORE' SUBROUTINE
; ------------------------------
;   This routine could also be used by the tape routines
;   It restores the border colour to normal after it has been altered to show
;   communication activity.  Since interrupts are usually enabled at the same
;   time, that instruction has been incorporated here to conserve ROM space.

BORD_REST PUSH  AF              ; Preserve accumulator throughout.

          LD    A,($5B48)       ; 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.

          POP   AF              ; Restore flags.

          EI                    ;+ Enable Interrupts.

          RET                   ; Return.

; --------------------------
; THE 'TEST BREAK' SUBROUTINE
; --------------------------
;   Note. this could also be called at statement return STMT_RET.

TEST_BRK  CALL  BREAK_KEY       ; Call the standard ROM routine.
          RET   C               ; return if BREAK not pressed.

          CALL  BORD_REST       ; else restore the border colour to normal.

REPORT_Lb RST   30H             ; ERROR-1
          DEFB  $14             ; Error Report: BREAK into program


; **************************
; ** THE NETWORK ROUTINES **
; **************************

;   "Spectrum" is the Latin word for a rainbow.
;   "Iris" is the Greek word for a rainbow.
;   By convention, "Spectrum" refers to this computer and "Iris" refers to the
;   other computer which could be a ZX Spectrum or Sinclair QL.

;   While Sinclair Research were secretive about the microdrive internals,
;   there were no such restrictions on the Sinclair Network which remains an
;   open standard.  It was also adopted by the Disciple Disk Interface.

;   "Indeed the linking of one microcomputer to another should be encouraged
;    and the establishment of a Sinclair Network Standard may prove an
;    important step forward"
;    - Dr. Ian Logan, Interface 1 ROM co-author, 1983.

; ----------------------------------------
; THE '"N" CHANNEL INPUT SERVICE' ROUTINE
; ----------------------------------------
;   The address of this network input service routine is contained in the
;   channel information area and accessed by the INPUT_AD routine when the
;   current channel has been made the "N" channel.
;   The routine inputs a single byte from the network and if the 255-byte
;   network buffer is empty, this may involve receiving a packet from the
;   network.


NCHAN_IN  LD    IX,($5B51)      ; Set index register from system variable CURCHL

          LD    A,(IX+$0C)      ; Fetch number of output buffer bytes from NCOBL
                                ; This should be zero when reading.

          AND   A               ; Test for zero.
          JR    Z,TEST_BUFF     ; Forward, if so, to TEST-BUFF.

          RST   30H             ; ERROR-1
          DEFB  $1D             ; 'Net R/W error'
                                ; Should be -
                                ; 'Reading a 'write' file' 22 chars

; ---

TEST_BUFF LD    A,(IX+$10)      ; Fetch number of input buffer bytes from NCIBL
          AND   A               ; test for zero.
          JR    Z,TST_N_EOF     ; forward, if so, to TST-N-EOF

          LD    E,(IX+$0F)      ; Fetch position of last character taken NCCUR
          DEC   A               ; Decrement the total count.
          SUB   E               ; Subtract the taken count - will set the carry
                                ; flag if at end.

          JR    C,TST_N_EOF     ; Forward, if so, to TST-N-EOF

          LD    D,$00           ; Prepare to index.
          INC   E               ; Increment the position
          LD    (IX+$0F),E      ; and update the channel variable NCCUR.

          ADD   IX,DE           ; Index into the buffer at that position.
          LD    A,(IX+$10)      ; Read the byte from the buffer.
          SCF                   ; Signal success.

          RET                   ; Return.

; ---

TST_N_EOF LD    A,(IX+$0B)      ; Fetch packet type from NCTYPE - 0 data, 1 EOF.

          AND   A               ; Test for data.
          JR    Z,GET_N_BUF     ; Forward, if so, to GET-N-BUF   ->

;   Note. Iris has closed her channel.

          RET                   ; Return   (NC and NZ)
                                ; Note. causes error 'End of file'

; ---

;   ->

GET_N_BUF LD    A,($5BBD)       ; fetch I/O border colour from IOBORD
          OUT   ($FE),A         ; and change the colour to show activity.

          DI                    ; Disable Interrupts.
```

```
TRY_AGAIN CALL  WT_SCOUT        ; routine WT-SCOUT waits for the scout leader.

          JR    NC,TIME_OUT     ; forward, if none, to TIME-OUT

          CALL  GET_NBLK        ; routine GET-NBLK gets the header and data.

          JR    NZ,TIME_OUT     ; forward, if error, to TIME-OUT

;;;       EI                    ; Enable Interrupts

          CALL  BORD_REST       ; routine BORD-REST restores the border.

          LD    (IX+$0F),$00    ; Set cursor position NCCUR to zero.

          LD    A,($5BC5)       ; Fetch header type code from NTTYPE - data/EOF.

          LD    (IX+$0B),A      ; update the channel variable NCTYPE

          JR    TEST_BUFF       ; back to TEST-BUFF to read the first byte.

; ---

;;; TIME_OUT LD A,(IX+$07)      ; Fetch the destination station number NCIRIS
;;;       AND   A               ; test for zero - a broadcast.

TIME_OUT  CALL  TST_BR          ;+ New routine to test for a broadcast.

          JR    Z,TRY_AGAIN     ; back, if a broadcast, to TRY-AGAIN
                                ; Note. a broadcast will not time out.

          JR    BORD_REST       ; back to exit via BORD_REST restoring border.

;;;       EI                    ; enable interrupts
;;;       CALL  BORD_REST       ; routine BORD-REST restores border preserving
;;;                             ; the AF registers and enabling interrupts.
;;;       AND   $00             ; signal failure. (NZ NC already set)
;;;       RET                   ; Return.


; -------------------------------
; THE '"N" CHANNEL OUTPUT' ROUTINE
; -------------------------------
;   The address of this network output service routine is contained in the
;   channel information area and accessed by the RST 10H output restart
;   routine when the current channel has been made the "N" channel.
;   The routine outputs a single byte to the network and if the 255-byte
;   network buffer is full, this may involve sending a packet to the
;   network.

NCHAN_OUT LD    IX,($5B51)      ; Set index register from system variable CURCHL

          LD    B,A             ; Copy the character to B.

          LD    A,(IX+$10)      ; Fetch number of input buffer bytes from NCIBL
                                ; should be zero if channel is used for writing.
          AND   A               ; test for zero bytes.

          LD    A,B             ; bring the character back.

          JR    Z,TEST_OUT      ; forward, if zero bytes, to TEST-OUT

          RST   30H             ; ERROR-1
          DEFB  $1D             ; 'Net R/W error'
```

```
                             ; Should be -
                             ; 'Writing to a 'read' file' 24 chars

; ---

TEST_OUT  LD    E,(IX+$0C)       ; fetch number of output buffer bytes from NCOBL
          INC   E                ; increment the count. 1-255
          JR    NZ,ST_BF_LEN     ; forward, if not full, to ST-BF-LEN

;   The buffer is full and must be sent to the network.

          PUSH  AF               ; preserve character yet to be output

          XOR   A                ; Set A to 0 to signal data and not EOF.

          CALL  S_PACK_1         ; routine S-PACK-1 sends the 255-byte buffer.

;   The character can now be placed in the empty buffer at position 1.

          POP   AF               ; restore the output character.

          LD    E,$01            ; set buffer position to 1.

ST_BF_LEN LD    (IX+$0C),E       ; Update the byte count channel variable NCOBL
          LD    D,$00            ; Prepare to index.
          ADD   IX,DE            ; Index into the network buffer.
          LD    (IX+$10),A       ; and store the byte at the offset.

          RET                    ; Return.

; ----------------------------
; THE NEW 'NETWORK CR' ROUTINE
; ----------------------------
;   This routine is an extra check after outputting a carriage return.

CR_END    CALL  IN_CHAN_K        ;+ routine fetches the current channel letter.

          CP    'N'              ;+ Is it the network.

          RET   NZ               ;+ Return if not.


; --------------------------------------
; THE 'SEND CR BLOCK TO NETWORK' ROUTINE
; --------------------------------------
;   This should be sent as part of PRINT_CR to flush.

SEND_NCR  LD    C,$00            ; a data block signal.

          JR    SEND_END         ; forward to send the packet.

; ---------------------------------------
; THE 'SEND EOF BLOCK TO NETWORK' ROUTINE
; ---------------------------------------
;   This should be sent as part of CLOSE to flush.

SEND_NEOF LD    C,$01            ;+ An EOF signal

SEND_END  LD    IX,($5B51)       ; Set IX to current channel from CURCHL
          LD    A,(IX+$0C)       ; Load A from NCOBL the number of characters in
                                 ; the  output buffer.
          AND   A                ; Test for zero.
          RET   Z                ; Return with zero.
          LD    A,C             ; Signal an EOF packet if 1.

; ----------------------
; THE 'S-PACK-1' ROUTINE
; ----------------------
;

S_PACK_1  CALL  SEND_PACK       ; routine SEND-PACK sends a packet

          RET   NZ              ; Return if not a broadcast.

; -----------------------------
; THE 'BROADCAST DELAY' ROUTINE
; -----------------------------
;   This routine ensures that there is a gap between packets when broadcasting.

BR_DELAY  LD    DE,$1500        ; Set a delay counter.

DL_LOOP   DEC   DE              ; decrement.
          LD    A,E             ; Test for zero.
          OR    D               ;
          JR    NZ,DL_LOOP      ; back, if not, to DL-LOOP

          RET                   ; Return.

; -----------------------
; THE 'SEND-PACK' ROUTINE
; -----------------------
;   This routine checksums, and then outputs to the network, an 8-byte
;   header and a corresponding data block.

SEND_PACK LD    (IX+$0B),A      ; Update the channel variable NCTYPE with the
                                ; packet type - 0 data, 1 EOF.

          LD    B,(IX+$0C)      ; Load counter with number of output characters
                                ; from NCOBL channel variable.

          LD    A,($5BBD)       ; Fetch I/O border colour from IOBORD
          OUT   ($FE),A         ; Change border to show communication activity.

          PUSH  IX              ; Transfer the channel base address
          POP   DE              ; to the DE register pair.

          LD    HL,$0011        ; prepare the offset seventeen.
          ADD   HL,DE           ; and add to point to the first data byte.

;;;       XOR   A               ; Initialize the data checksum to zero.
;;; CHKS1 ADD   A,(HL)          ; add a data byte.
;;;       INC   HL              ; increment buffer pointer.
;;;       DJNZ  CHKS1           ; back, for count of characters, to CHKS1

          CALL  CHKS0           ;+ New general purpose checksum routine.

          LD    (IX+$0D),A      ; insert the checksum in NCDCS channel variable.

          LD    HL,$0007        ; prepare the offset seven

          ADD   HL,DE           ; and add to address NCIRIS first header byte.

          PUSH  HL              ; (*)save the header pointer.

;;;       LD    B,$07           ; Set byte counter to seven.
;;;       XOR   A               ; Initialize the header checksum to zero.
;;; CHKS2 ADD   A,(HL)          ; add the addressed byte.
;;;        INC   HL              ; increment the pointer.
;;;        DJNZ  CHKS2           ; loop back to CHKS2

          CALL  CHKS7           ; routine checksums seven bytes

          LD    (HL),A          ; insert the checksum into NCHCS channel var.

          DI                    ; Disable Interrupts

SENDSCOUT CALL  SEND_SC         ; routine SEND-SC

          POP   HL              ; (*) restore the header pointer - NCIRIS.
          PUSH  HL              ; (*) and preserve again.

          LD    E,$08           ; There are eight bytes in a network header.

          CALL  OUT_BLK_N       ; routine OUT-BLK-N sends HEADER and receives
                                ; the response code.

          JR    NZ,SENDSCOUT    ; back, with no response, to SENDSCOUT

          PUSH  IX              ; transfer base address of channel
          POP   HL              ; to the HL register.

          LD    DE,$0011        ; prepare an offset of seventeen.
          ADD   HL,DE           ; add to address the first byte of buffer data.

          LD    E,(IX+$0C)      ; Fetch count of output characters from NCOBL
          LD    A,E             ; copy value to A.
          AND   A               ; test for zero.
          JR    Z,INC_BLKN      ; forward, if zero, to INC-BLKN

          LD    B,$20           ; Set a delay value for a gap between the
                                ; physical 8-byte header and data.

SP_DL_1   DJNZ  SP_DL_1         ; self loop to SP-DL-1

          CALL  OUT_BLK_N       ; routine OUT-BLK-N sends the DATA and receives
                                ; the response byte.

          JR    NZ,SENDSCOUT    ; back, with no response, to SENDSCOUT


;;; INC_BLKN  INC   (IX+$09)     ; increment the channel variable NCNUMB_lo
;;;        JR    NZ,SP_N_END     ; forward, if not '256' to SP-N-END
;;;        INC   (IX+$0A)        ; increment the channel variable NCNUMB_hi

INC_BLKN  CALL  SUBINC          ;+ an existing subroutine that does above.

SP_N_END  POP   HL              ; (*) restore the pointer to NCIRIS

          CALL  BORD_REST       ; routine BORD-REST restores border.

;;;        EI                    ; enable interrupts.

          LD    (IX+$0C),0      ;+++ SET the number of output bytes to zero.

TST_BR    LD    A,(IX+$07)      ; fetch station of other machine from NCIRIS
          AND   A               ; test for zero - a broadcast.

          RET                   ; Return - with zero flag set for broadcast.

; -----------------------
; THE 'OUT-BLK-N' ROUTINE
; ----------------------
;   This routine sends a single block of data, which could be a header or
;   buffer block, and validates the response from IRIS.

OUT_BLK_N CALL  OUTPAK          ; routine OUTPAK sends the data block returning
                                ; with DE holding zero.

;;;       LD    A,(IX+$07)      ; fetch the other station number from NCIRIS
;;;       AND   A               ; test for zero.

          CALL  TST_BR          ;+ New routine to test for a broadcast.

          RET   Z               ; Return if a broadcast, no response required.

          LD    HL,$5BC0        ; Make HL address system variable NTRESP

;;;       LD    (HL),$00        ; and insert a zero.
;;;       LD    E,$01           ; set byte count to 1.

          LD    (HL),E          ;+ and insert a zero.
          INC   E               ;+ set byte count to 1.

          CALL  INPAK           ; routine INPAK reads one byte from network.

          RET   NZ              ; return, signaling failure, if no activity.

          LD    A,($5BC0)       ; Fetch updated value of NTRESP.
          DEC   A               ; test for $01.
          RET                   ; Return, with zero flag set for success.

; --------------------------------------------
; THE 'HEADER AND DATA BLOCK RECEIVING' ROUTINE
; --------------------------------------------
;   This subroutine is called once from NCHAN_IN to read in the next header
;   and data block from the network, when the buffer is empty.
;   An eight byte header is sent from the network channel of the sending
;   computer but received into eight system variables.

GET_NBLK  LD    HL,$5BC1        ; Address system variable NTDEST
          LD    E,$08           ; Set byte count to eight.

          CALL  INPAK           ; Routine INPAK reads in a header.

          RET   NZ              ; Return, signaling failure, if inactive.

;;;       LD    HL,$5BC1        ; Address system variable NTDEST again.

          LD    L,$C1           ;+ Address system variable NTDEST again

;;;       XOR   A               ; Initialize checksum to zero.
;;;       LD    B,$07           ; Set byte count to seven.
;;; CHKS3 ADD   A,(HL)          ; Add the addressed byte.
;;;       INC   HL              ; Point to next header byte.
;;;       DJNZ  CHKS3           ; Back, for all seven bytes, to CHKS3
;;;       CP    (HL)            ; Compare with eighth byte.

          CALL  CHKS7           ; routine to checksum seven bytes.

          RET   NZ              ; Return if checksum disagrees.

;   The header has been successfully received and it can be examined to see
;   what type of data this is.
```

```
;;;       LD    A,($5BC1)        ; Fetch the value of NTDEST the received sysvar

          LD    L,$C1            ;+ Set HL to NTDEST the received system variable
          LD    A,(HL)           ;+ Fetch value to A.

          INC   L                ;+ Set HL to $5BC2 NTSRCE before branching.

          AND   A                ; test for zero.
          JR    Z,BRCAST         ; forward, if so, to BRCAST

          CP    (IX+$08)         ; Compare the destination with NCSELF - this
                                 ; station's number.
          RET   NZ               ; Return if data not intended for this Spectrum.

;   The header indicates the data is for SELF.

;;;       LD    A,($5BC2)        ; Fetch value of received system variable NTSRCE
;;;                              ; the sending station.

          LD    A,(HL)           ;+ Fetch the sender from $5BCF NTSRC.

          CP    (IX+$07)         ; Compare to channel variable NCIRIS.
          RET   NZ               ; Return if not the expected sender.

;   The header indicates that both the sender and recipient (SELF) are correct.

          JR    TEST_BLKN        ; Forward to TEST-BLKN

; ---

;   The branch was here when the header indicated a broadcast.


BRCAST
;;;       LD   A,(IX+$07)        ; Check the station number of NCIRIS.
;;;       OR    A                ; Test for zero - a broadcast.

          CALL  TST_BR           ;+ New routine to test for a broadcast.

          RET   NZ               ; Return, with failure, if not.

;   The two paths converge here.

TEST_BLKN LD    HL,($5BC3)       ; Fetch, from received system variable NTNUMB,
                                 ; the number of the block.

          LD    E,(IX+$09)       ; Fetch bytes of expected block NCNUMB_lo and
          LD    D,(IX+$0A)       ; NCNUMB_hi, to DE, from channel variables.

          AND   A                ; prepare for true subtraction.
          SBC   HL,DE            ; subtract the two block numbers.
          JR    Z,GET_NBUFF      ; forward, if they match, to GET-NBUFF

          DEC   HL               ; Decrement HL in case the previously received
          LD    A,H              ; block is being resent because the sender
          OR    L                ; missed our response byte.
          RET   NZ               ; Return if this is not the previous block.

          CALL  GET_NBUFF        ; Routine GET-NBUFF gets the buffer data and
                                 ; resends the response byte also incrementing
                                 ; the block count.

;   Note. the next check is new.
```

```
          RET   NZ              ;+ Return if, say, the checksum disagrees the
                                ;+ second time around.

;   Cancel the increment and return failure so that we try for the expected
;   data again.
;   Note. The DEC instruction does not affect the carry flag!

          LD    A,(IX+$09)      ;+ Fetch copy of NCNUMB_lo to accumulator.

          DEC   (IX+$09)        ; decrement actual NCNUMB_lo

          AND   A               ;+ Was it originally zero?

;;;       JR    NC,GETNB_END    ; Forward, if not 255, to GETNB-END

          JR    NZ,GETNB_END    ;+ Forward, if not now 255, to GETNB-END

          DEC   (IX+$0A)        ; Decrement NCNUMB_hi

GETNB_END OR    $01             ; Reset the zero flag signaling failure to
                                ; the calling routine.

          RET                   ; Return.

; ---

;   The branch was here to read the data into the network buffer.
;   First send a response byte for the header if this is not a broadcast.

GET_NBUFF CALL  SEND_RESP       ; routine sends response byte if not a broadcast

          LD    A,($5BC6)       ; Fetch received system variable NTLEN
          AND   A               ; test for zero.
          JR    Z,STORE_LEN     ; forward, if empty, to STORE-LEN

          PUSH  IX              ; transfer channel base address to
          POP   HL              ; the HL register.

          LD    DE,$0011        ; prepare an offset of seventeen.
          ADD   HL,DE           ; and add to address start of the data buffer.
          PUSH  HL              ; (*) Preserve the start of network buffer.
          LD    E,A             ; Transfer block length to E, value 1-255.

          CALL  INPAK           ; routine INPAK reads in the data.

          LD    HL,($5BC6)      ;+ load two system variables NTLEN/NTDCS at
once.
          LD    B,L             ;+ count of bytes NTLEN to B.
          LD    E,H             ;+ checksum NTDCS to E.

          POP   HL              ; (*) Restore start of buffer.
          RET   NZ              ; Return failure if network was inactive.

;;;       LD    A,($5BC6)       ; Fetch count of data bytes from sysvar NTLEN
;;;       LD    B,A             ; transfer to B.      (Beyond reach of IY)
;;;       LD    A,($5BC7)       ; Fetch network data checksum from NTDCS
;;; The checksum is verified in the opposite way in which it was derived.
;;; CHKS4 SUB   (HL)            ; Subtract the addressed value.
;;;       INC   HL              ; Increment data pointer.
;;;       DJNZ  CHKS4           ; Back, for all bytes, to CHKS4

          CALL  CHKS0           ;+ general purpose checksum adding routine.
          CP    E               ;+ compare with expected.
          RET   NZ              ; Return failure if result does not agree.

;;;       LD    A,($5BC1)       ; Fetch station from system variable NTDEST
;;;       AND   A               ; Check for zero - a broadcast

          CALL  SEND_RESP       ; routine SEND-RESP sends response for data
                                ; if not a broadcast.

STORE_LEN LD    A,($5BC6)       ; Fetch the verified length from sysvar NTLEN
          LD    (IX+$10),A      ; and insert value in channel variable NCIBL.

;   Note the next could be a once-called subroutine to increment NCNUMB.

SUBINC    INC   (IX+$09)        ; Increment NCNUMB_lo

          JR    NZ,GETBF_END    ; Forward, if no wraparound, to GETBF-END

          INC   (IX+$0A)        ; Increment the high byte NCNUMB_hi

GETBF_END CP    A               ; Set the zero flag.

          RET                   ; Return with zero flag set indicating success.

; -----------------------
; THE NEW 'CHECKSUM' ROUTINES
; -----------------------
;   This routine saves just a few bytes by making a subroutine of repetitive
;   code.  It checksums the header or buffer data. The header data is always
;   seven bytes in length.  The second entry point is used for the variable
;   length buffer data.

CHKS7     LD    B,$07           ; 7 header bytes to check.

CHKS0     XOR   A               ; Initialize checksum.

CHKSL     ADD   A,(HL)          ; Add a byte.
          INC   HL              ; Address next.
          DJNZ  CHKSL           ; loop back for B bytes.

          CP    (HL)            ; compare with final value which could be sum

          RET                   ; return.


; ---------------------------
; THE 'NETWORK STATE' ROUTINE
; ---------------------------
;   This routine waits for the network to become inactive so that it may be
;   claimed by this SPECTRUM. So that two waiting Spectrums do not claim the
;   network at the same time a random count is used.

NET_STATE LD    A,R             ; Fetch a random 7-bit value from the Refresh
                                ; register

          OR    $C0             ; OR with %11000000 giving range 192 - 255.
          LD    B,A             ; transfer to B for count.

          CALL  CHK_REST        ; routine CHK-REST

          JR    C,NET_STATE     ; back to NET-STATE if network is busy.

          RET                   ; return.
```

```
; --------------------------
; THE 'CHECK-RESTING' ROUTINE
; --------------------------
;   This subroutine checks that the network is inactive for a period
;   determined by the B register which will $80 if called from below or a
;   random value if called from the above routine.

CHK_REST  CALL  TEST_BRK        ; routine TEST-BRK will return here if BREAK is
                                ; not pressed.

MAKESURE  PUSH  BC              ; timing
          POP   BC              ; timing

          IN    A,($F7)         ; Read the network port
          RRCA                  ; Test bit 0.
          RET   C               ; Return if network is claimed by other
                                ; machines.

          DJNZ  MAKESURE        ; back to MAKESURE for a random count.

          RET                   ; Return.


; ------------------------
; THE 'WAIT SCOUT' ROUTINE
; ------------------------
;   This routine is called once from NCHAN_IN to identify a SCOUT when this
;   station is expecting to receive a network packet.

WT_SCOUT  CALL  TEST_BRK        ; routine TEST-BRK allows user to abort.

          LD    HL,$01C2        ; An even timing value.

CLAIMED   LD    B,$80           ; The non-random count for CHK REST

          CALL  CHK_REST        ; routine CHK-REST

          JR    NC,WT_SYNC      ; forward, if network resting, to WT-SYNC

          DEC   HL              ; decrement counter
          DEC   HL              ; decrement counter
          LD    A,H             ; Test for
          OR    L               ; zero.
          JR    NZ,CLAIMED      ; back, if not, to CLAIMED

;   If operation has timed out, then return failure unless expecting a
;   broadcast in which case the subroutine waits indefinitely.

;;;       LD    A,(IX+$07)      ; fetch value of channel variable NCIRIS
;;;       AND   A               ; test for broadcast.

          CALL  TST_BR          ;+ test for a broadcast

          JR    Z,CLAIMED       ; back, if so, to CLAIMED

          RET                   ; Return signaling failure.

; ---

;   The branch was here when the network was found to be at rest.

WT_SYNC   IN    A,($F7)         ; read the port.
          RRCA                  ; rotate 'net input' to carry.
          JR    C,SCOUT_END     ; forward, if scout found, to SCOUT-END  ->

;   This is a clever twist.

          LD    A,$7F           ; Read port $7FFE the row with SPACE
          IN    A,($FE)         ;
          OR    $FE             ; Now read port $FEFE the row with SHIFT
          IN    A,($FE)         ;
          RRA                   ; If both SHIFT and SPACE then carry is reset.

          CALL  NC,TEST_BRK     ; routine TEST-BRK errors if BREAK was pressed.

          DEC   HL              ; decrement the counter.
          LD    A,H             ; Test for
          OR    L               ; zero.
          JR    NZ,WT_SYNC      ; back, if not, to WT-SYNC

;;;       LD    A,(IX+$07)      ; load station number from NCIRIS
;;;       AND   A               ; test for a broadcast.

          CALL  TST_BR          ;+ New routine to test for a broadcast.

          JR    Z,WT_SYNC       ; back, if zero, to WT-SYNC

          RET                   ; Return failure - NZ

; ----------------------
; THE 'SCOUT END' BRANCH
; ----------------------
;   The scout is only read by the machine that sent it and as long as it can
;   be read back then the sending machine is happy.  This receiving machine
;   uses the scout leader to synchronize its timing.

SCOUT_END LD    L,$09           ; set outer counter for 9 bits.

LP_SCOUT  DEC   L               ; ( 4) decrement counter.
          SCF                   ; ( 4) set success condition

          RET   Z               ; (  ) RETURN after nine bits have been timed.

          LD    B,$0E           ; An inner delay counter.

DELAY_SC  DJNZ  DELAY_SC        ; self loop to DELAY-SC

          JR    LP_SCOUT        ; loop back to LP-SCOUT

; --------------------------------------------------------------------------
;
; Illustration:  Network Station '10' sending its SCOUT.
;
;                  <-- values of the eight bits -->
;
;  active          '0' '0' '0' '0' '1' '0' '1' '0'             active
; - - - - - - +---+---+---+---+---+   +---+   +---+ - - - - - - - - - - - -
;             |   |   |   |   |   |   |   |   |   |
;             |                   |   |   |   |   |
;             |     7   6   5   4 | 3 | 2 | 1 | 0 |
;             |                   |   |   |   |   |
;             |                   |   |   |   |   |
;  inactive   |   |   |   |   |   |   |   |   |   |            inactive
; ------------+   +   +   +   +   +---+   +---+   +-------------------------
;
;   ^ ^ ^     <---><--------------  ------------->^
;  resting ?    \/                \/              |
;               send        send 8 bits of the     |
;               SCOUT      global station number  Network claimed.
;              leader
;

; ------------------------
; THE 'SEND SCOUT' ROUTINE
; ------------------------
;   In order to claim the network for writing, the SPECTRUM sends out a leader
;   followed by the eight inverted bits of the global station number which
;   should be unique to each machine.

SEND_SC   CALL  NET_STATE       ; routine NET-STATE repeatedly examines the
                                ; network until it is satisfied that the
                                ; network is at rest.

          LD    C,$F7           ; The comms port number.
          LD    HL,$0009        ; H is the leader value, L is the count of
                                ; the SCOUT bits.

          LD    A,($5BBC)       ; Fetch the global station number from NTSTAT
          LD    E,A             ; Transfer to E.

          IN    A,($F7)         ; Test that the network is still inactive.
          RRCA                  ; rotate 'net input' to carry.
          JR    C,SEND_SC       ; back, if network has become active, to SEND-SC

;   Now output the nine values starting with the zero leader in H.


ALL_BITS  OUT   (C),H           ; output bit 0 'net output'

          LD    D,H             ; ( 4) copy state to D for later.
          LD    H,$00           ; set output byte to zero.
          RLC   E               ; rotate a bit from 'global station' to carry.
          RL    H               ; pick it up in bit 0.


          LD    B,$08           ; set timing counter

S_SC_DEL  DJNZ  S_SC_DEL        ; self loop back to S-SC-DEL

          IN    A,($F7)         ; read the network.
          AND   $01             ; isolate the 'net input' bit.
                                ; Note. network is activated with a zero bit.
                                ; Received bit is therefore opposite state.
          CP    D               ; compare with expected state.
          JR    Z,SEND_SC       ; back, if not inverted, to SEND-SC
                                ; to start process again.

          DEC   L               ; decrement the bit counter.
          JR    NZ,ALL_BITS     ; back, if SCOUT not complete, to ALL-BITS

          LD    A,$01           ;; set the output bit.
          OUT   ($F7),A         ;; Make the network inactive.

          LD    B,$0E           ; Wait for a delay

END_S_DEL DJNZ  END_S_DEL       ; self loop to END-S-DEL

          RET                   ; Return.


; -------------------
; THE 'INPAK' ROUTINE
; -------------------
;   This routine reads into the network buffer at address HL a pack of bytes
;   the count of which is in E.
;   The value of E will be -
;   a) 1 when reading the Network Response byte.
;   b) 8 when reading an eight-byte header into the system variables.
;   c) The value of NTLEN, from within the above header, when reading data.

INPAK     LD    B,$FF           ; Set a time-out counter.

N_ACTIVE  IN    A,($F7)         ; Read the network port.
          RRA                   ; rotate 'net input' bit to carry.

          JR    C,INPAK_2       ; forward, if set, to INPAK-2

          DJNZ  N_ACTIVE        ; loop back, 255 times, to N-ACTIVE

          INC   B               ; Indicate network inactive by resetting zero.

          RET                   ; Return.  (NZ)

; ---

INPAK_2   LD    B,E             ; Set B to count the number of bytes.

;   The byte reading loop.

INPAK_L   LD    E,$80           ; prepare a receiving byte with a marker bit.

          LD    A,$CE           ; Make A %11001110 (wait,cts and comms data low)
          OUT   ($EF),A         ; Enable the network.

          NOP                   ; ( 4) Wait 48 clock cycles.
          NOP                   ; ( 4)
          INC   IX              ; (10)
          DEC   IX              ; (10)
          INC   IX              ; (10)
          DEC   IX              ; (10)

UNTIL_MK  LD    A,$00           ; ( 7) Timing.
          IN    A,($F7)         ; (10) Read net input to bit 0.
          RRA                   ; ( 4) rotate to carry.
          RR    E               ; ( 8) and pick up in E.
          JP    NC,UNTIL_MK     ; (10) JUMP, back if no marker bit, to UNTIL-MK

          LD    (HL),E          ; store the received byte.
          INC   HL              ; Address next location.
          DJNZ  INPAK_L         ; back to INPAK-L

          CP    A               ; Set zero flag to signal success.
          RET                   ; Return.


; --------------------------------
; THE 'SEND RESPONSE BYTE' ROUTINE
; --------------------------------
;   When a header or a data block is successfully received then this routine is
;   used to send a response byte to acknowledge the successful receipt of the
;   data over the network.

SEND_RESP LD    HL,$5BC1        ; Address station number NTDEST
          XOR   A               ; set accumulator to zero
          CP    (HL)            ; compare with NTDEST
          RET   Z               ; return if a broadcast.

          DEC   HL              ; Address $5BC0 NTRESP
          LD    E,$01           ; Load 1 to the byte count.
          LD    (HL),E          ; Insert the value 1

; --------------------
; THE 'OUTPAK' ROUTINE
; --------------------
;   This routine sends a packet of bytes, up to 255 in length, over the network.
;   The start of the data is in HL and the number of bytes is held in E.


OUTPAK    XOR   A               ; clear bit 0
          OUT   ($F7),A         ; send leader to port.
          LD    B,$04           ; ( 4) set timing value.

DEL_0_1   DJNZ  DEL_0_1         ; (12/7) back to DEL-0-1 for leader of

;   Now enter a loop to send E bytes each with a set start bit and a reset
;   stop bit.

OUTPAK_L  LD    A,(HL)          ; ( 6) Fetch a byte to be sent.
          CPL                   ; ( 4) complement.
          SCF                   ; ( 4) Set an initial start bit
          RLA                   ; ( 4)  C <- 76543210 <- C
          LD    B,$0A           ; ( 6) Set count to ten bits

UNT_MARK  OUT   ($F7),A         ; Output bit 0, to net.
          RRA                   ;       C -> 76543210 -> C ; Rotate next bit to
                                ;                          ; bit 0.
          AND   A               ; clear carry flag to feed in final stop bit.
          DEC   B               ; decrement bit counter.
          LD    D,$00           ; ( 7) timing
          JP    NZ,UNT_MARK     ; JUMP back for 10 bits to UNT-MARK

;   The last bit sent will be a reset stop bit.

          INC   HL              ; increment buffer address
          DEC   E               ; decrement the byte count.
          PUSH  HL              ; (11) timing.
          POP   HL              ; (11) timing.
          JP    NZ,OUTPAK_L     ; JUMP, if E not zero, to OUTPAK-L

          LD    A,$01           ; switch off network.
          OUT   ($F7),A         ;
          RET                   ; Return.

; --------------------
; THE 'FORMAT' COMMAND
; --------------------
;   by James Smith.
;   This sets the local network station number which defaulted to 1 at
;   switch-on. It can also be used to set the baud rate and printer width.
;   FORMAT "n",2    set this station to station 2 ( acceptable range 1 - 64 ).
;   FORMAT "b",1200 set baud rate for "b" and "t" RS232 transfers.
;   FORMAT "t",80   set printer width of the "t" channel.
;
;   This is a CLASS-00 command so it is only executed in runtime when the two
;   parameters will be on the calculator stack.

FORMAT    CALL  FIND_INT2       ; routine FIND-INT2 gets number to BC
```

```
          PUSH  BC               ; save on machine stack.

          CALL  EXPT_SPEC        ; gets channel specifier in A

          AND   $DF              ; Make it upper-case.

          POP   BC               ; retrieve numeric parameter.

          CP    'B'              ; channel "B" BAUD rate ?
          JR    Z,FORMAT_B       ; forward, if so, to FORMAT_B

;   After the 16 bit BAUD rate, only 8-bit values are allowed for width/station.

          INC   B                ; Test the high-order
          DEC   B                ; byte for zero.

          JP    NZ,REPORT_B      ; ERROR
                                 ; 'Integer out of range'

          CP    'T'              ; Text width ?
          JP    Z,FORMAT_T       ; jump back, if so, to FORMAT_T

          CP    'N'              ; Network ?
          JP    NZ,REPORT_C      ; back, if unknown letter, to report XXXXX
                                 ; 'Nonsense in BASIC'


FORMAT_N  LD    A,C              ; number should be 1-64
          DEC   A                ;

          CP    $40              ; compare to 64
          JP    NC,REPORT_Q      ;

          INC   A                ; correct for earlier DEC

          LD    ($5BBC),A        ; set NTSTAT

          RET                    ; Return.

; ---

;   Note. these 5 bytes fave been moved to space between restarts. (JS)

; FORMAT_T  LD    A,C            ; get TAB width
;         LD    ($5BB8),A        ; set it
;         RET                    ; Return.

; ---------------------------
; THE 'SET BAUD RATE' ROUTINE
; ---------------------------
;   by James Smith.
;   The BAUD rate is calculated as follows:
;   BAUD = (3500000/(26*baud rate)) - 2
;

FORMAT_B  CALL  STACK_BC         ; put value on calculator stack.

          RST   28H              ; FP-CALC
          DEFB  $34              ;;stk-data
          DEFB  $35              ;;Exponent $85, Bytes:1 constant = 26
          DEFB  $50              ;;$50 ($00,$00,$00)
          DEFB  $04              ;;multiply
          DEFB  $34              ;;stk-data
          DEFB  $80              ;;Exponent $96, Bytes: 3 constant = 3500000
```

```
         DEFB  $46,$55,$9F,$80 ;;($55,$9F,$80,$00)
         DEFB  $01             ;;exchange
         DEFB  $05             ;;divide
         DEFB  $38             ;;end-calc

         CALL  FP_TO_BC        ; get delay into BC

         DEC   BC              ; subtract value
         DEC   BC              ; two.

         LD    ($5BBA),BC      ; set BAUD system variable

         RET                   ; Return.



; -----------------
; THE 'CAT' COMMAND
; -----------------
;   This CAT command lists the streams to the screen and really grows on you.
;   It was inspired by Andrew Pennell's "Stream Lister" which appears in the
;   book "master your zx microdrive" published by Melbourne House.

CAT
         CALL  CL_ALL          ; clear 24 lines and leave upper screen open.

         LD    DE,CAT1         ; Point to start of banner text.
         LD    BC,CAT2-CAT1    ; Set the length.
         CALL  PR_STRING       ; routine PR_STRING outputs counted string.

         LD    BC,45           ; decimal adjustment to equate to command line.

         CALL  TEST_ROOM       ; routine TEST_ROOM returns free RAM in HL.

         LD    A,H             ; The value is negated and must be transferred
         CPL                   ; to BC registers.
         LD    B,A             ;

         LD    A,L             ;
         CPL                   ;
         LD    C,A             ;

         CALL  STACK_BC        ; stack the 16 bit value.
         CALL  PRINT_FP        ; print the free memory.

         LD    DE,CAT3         ; address the remaining text setting inverse 0
         LD    BC,CAT4-CAT3    ; set the length of the string.

         CALL  PR_STRING       ; print the rest of the banner.

         LD    A,$FD           ; The starting stream. (decimal 253).
         LD    HL,$5B10        ; The relevant system variables location.

LOOP     AND   A               ; test for zero
         PUSH  AF              ; save the stream
         PUSH  HL              ; save the address in STRMS area
         JR    NZ,NO_BLANK     ; skip forward if not stream zero.

         LD    A,$0D           ; print a carriage return as a separator
         RST   10H             ; if it is zero

NO_BLANK CALL  STACK_A         ;
         LD    A,$0D           ;
         RST   10H             ;
```

```
          CALL  PRINT_FP        ;
          LD    A,6             ;
          RST   10H             ;
          POP   HL              ;

          LD    C,(HL)          ;
          INC   L               ;
          LD    B,(HL)          ;
          INC   L               ;
          LD    A,B             ;
          OR    C               ;
          JR    Z,PR_CR         ;
          LD    IX,($5B4F)      ; CHANS
          ADD   IX,BC           ;
          LD    A,(IX+$03)      ;
          RST   10H             ;
PR_CR     POP   AF              ;
          INC   A               ;
          CP    $10             ;
          JR    NZ,LOOP         ;

          RET                   ; Return.

CAT3
          DEFB  $06             ; The 'comma control'
          DEFB  $14,$00         ; The control codes for INVERSE 0
          DEFB  $0D             ; The carriage return character.
CAT4

; ---------------------------
; THE 'AUTO-LIST' SUBROUTINE
; ---------------------------
;   This produces an automatic listing in the upper screen.

AUTO_LIST LD    ($5B3F),SP      ; save stack pointer in LIST_SP
          LD    (IY+$02),$10    ; update TV_FLAG set bit 3

          CALL  CL_ALL          ; routine CL-ALL clears 24 lines.

;;;       SET   0,(IY+$02)      ; update TV_FLAG - signal lower screen in use
          CALL  SIG_L_SCR       ; set 0,(iy+$02) as a call.

          LD    B,(IY+$31)      ; fetch lower screen DF_SZ to B.

          CALL  CL_LINE         ; 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 it will be necessary
                                ; to clear the main screen.
          LD    HL,($5B49)      ; fetch E_PPC current edit line to HL.
          LD    DE,($5B6C)      ; 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,AUTO_L_2      ; to AUTO-L-2 if S_TOP is higher than E_PPC
                                ; to set S_TOP to E_PPC

          PUSH  DE              ; save the top line number.

          CALL  LINE_ADDR       ; routine LINE-ADDR gets address of E_PPC.

          LD    DE,$02C0        ; prepare known number of characters in
```

```
                                ; the default upper screen.

          EX    DE,HL           ; transfer offset to HL, program address to DE.
          SBC   HL,DE           ; subtract high value from low to obtain the
                                ; negated result used in addition.
          EX    (SP),HL         ; swap result with top line number on stack.

          CALL  LINE_ADDR       ; routine LINE-ADDR gets address of that
                                ; top line in HL and next line in DE.

          POP   BC              ; restore the result to balance the stack.

AUTO_L_1  PUSH  BC              ; save the result.

          CALL  NEXT_ONE        ; routine NEXT-ONE gets address in HL of the
                                ; line after auto-line (in DE).

          POP   BC              ; restore result.
          ADD   HL,BC           ; compute back.
          JR    C,AUTO_L_3      ; forward, if line 'should' appear, to AUTO-L-3

          EX    DE,HL           ; transfer the 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    ($5B6C),DE      ; update system variable S_TOP.

          JR    AUTO_L_1        ; back, until estimate reached, to AUTO-L-1

; ---

;   the jump was to here if S_TOP was greater than E_PPC

AUTO_L_2  LD    ($5B6C),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  LD    HL,($5B6C)      ; fetch S_TOP line number to HL.

          CALL  LINE_ADDR       ; routine LINE-ADDR gets address in HL.
                                ; Address of next in DE.

          JR    Z,AUTO_L_4      ; forward, if line exists, to AUTO-L-4

          EX    DE,HL           ; else use address of next line.

AUTO_L_4  CALL  LIST_ALL        ; routine LIST-ALL quits when screen full  >>>

;   The return will be to here if no scrolling occurred

          JP    PO_N_AUTO       ;+ to code similar to below.

;;;       RES   4,(IY+$02)      ; update TV_FLAG - signal no auto listing.
;;;       RET                   ; return.

; -------------------
; THE 'LLIST' COMMAND
; -------------------
;   List Program to any stream.
;   As the manual points out, this is not standard BASIC.
;   A short form of LIST #3. The listing goes to stream 3 - default printer.
;   This always was a nonsense for compatibility with the ZX81 but now one is
;   unable to assume that stream 3 will be used for a printer.
;   This will be replaced with an extra UDG.

LLIST     LD    A,$03           ; the usual stream for a Printer

          JR    LIST_1          ; forward to LIST-1

; ------------------
; THE 'LIST' COMMAND
; ------------------
;   List Program 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      LD    A,$02           ; default is stream 2 - the upper screen.

LIST_1    LD    (IY+$02),$00    ; the TV_FLAG is initialized with bit 0 reset
                                ; indicating upper screen in use.
;;;       CALL  SYNTAX_Z        ; routine SYNTAX-Z - checking syntax ?
;;;       CALL  NZ,CHAN_SLCT    ; routine CHAN-OPEN if in run-time.

          CALL  CHN_O_SYN       ;+ Routine opens channel in runtime.

;;;       RST   18H             ; GET-CHAR

          CALL  STR_ALTER       ; routine STR-ALTER will alter if '#'.

          JR    C,LIST_4        ; forward, if not a hash, to LIST-4


;;;       RST   18H             ; GET-CHAR
;;;       CP    $3B             ; is character a ';' ?
;;;       JR    Z,LIST_2        ; skip, if so, to LIST-2
;;;       CP    $2C             ; is character a ',' ?

          CALL  EXPT_SEP        ;+ NEW routine to check for ';' or ','.

          JR    NZ,LIST_3       ; forward, if neither separator, to LIST-3

;   we have, say,  LIST #15, and a number must follow the separator.

LIST_2    RST   20H             ; NEXT-CHAR

          CALL  EXPT_1NUM       ; routine EXPT-1NUM checks for numeric
                                ; expression and stacks it in run-time.

          JR    LIST_5          ; forward to LIST-5

; ---

;   the branch was here with just LIST #3 etc.

LIST_3    CALL  USE_ZERO        ; routine USE-ZERO defaults first line.

          JR    LIST_5          ; forward to LIST-5

; ---

;   the branch was here with LIST

LIST_4    CALL  FETCH_NUM       ; routine FETCH-NUM checks if a number
                                ; follows else uses zero.

LIST_5    CALL  CHECK_END       ; routine CHECK-END quits if syntax OK      >>
```

```
;   Continue in runtime.

          CALL  FIND_LINE       ;+ routine FIND-LINE fetches the number from the
                                ;+ calculator stack and validates 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,B             ; transfer the modified
;;;       LD    L,C             ; line number to HL.

          LD    ($5B49),HL      ; update E_PPC to the new line number.

          CALL  LINE_ADDR       ; routine LINE-ADDR gets the address of the
                                ; line.

;   This routine is called from AUTO-LIST

LIST_ALL  LD    E,$01           ; signal current line not yet printed

LIST_ALL2 CALL  OUT_LINE        ; 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,LIST_ALL2     ; back, if not, to LIST-ALL-2
                                ; (loop exit is via OUT-LINE)

;   Continue here if an automatic listing required.

          LD    A,($5B6B)       ; fetch DF_SZ lower display file size.
          SUB   (IY+$4F)        ; subtract S_POSN_hi the current line number.
          JR    NZ,LIST_ALL2    ; 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,$5B6C        ; fetch S_TOP the rough estimate.

          CALL  LN_FETCH        ; 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    LIST_ALL2       ; back to LIST-ALL-2.

; -----------------------------------------
; THE 'PRINT A WHOLE BASIC LINE' SUBROUTINE
; -----------------------------------------
;   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  LD    BC,($5B49)      ; fetch E_PPC the current line which may be
                                ; unchecked and not exist.

          CALL  CP_LINES        ; routine CP-LINES finds match or line after.

          LD    D,$3E           ; prepare cursor '>' in D.
          JR    Z,OUT_LINE1     ; 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 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  OUT_NUM_2       ; 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,OUT_LINE3     ; forward, if zero, to OUT-LINE3

          RST   10H             ; PRINT-A prints '>' the current line cursor.

;   this entry point is called from ED-COPY

OUT_LINE2 SET   0,(IY+$01)      ; update FLAGS - suppress leading space.

OUT_LINE3 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,$5B3B        ; point to FLAGS.
          RES   2,(HL)          ; signal 'K' mode. (starts before keyword)

;;;       BIT   5,(IY+$37)      ; test FLAGX - input mode ?
          CALL  TST_INP_M       ;+ bit 5,(iy+$37) as a 3-byte call.
          JR    Z,OUT_LINE4     ; forward, if not, to OUT-LINE4

          SET   2,(HL)          ; signal 'L' mode. (used for input)

OUT_LINE4 LD    HL,($5B5F)      ; 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,OUT_LINE5    ; forward, if not, to OUT-LINE5

          LD    A,$3F           ; load A with '?' the error marker.
          CALL  OUT_FLASH       ; routine OUT-FLASH to print flashing marker.
```

```
OUT_LINE5 CALL  OUT_CURS        ; 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  NUMBER          ; 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,OUT_LINE6     ; forward, if so, to OUT-LINE6
                                ; as line is complete

          EX    DE,HL           ; save the pointer in DE.
          CALL  OUT_CHAR        ; routine OUT-CHAR to output character/token.

          JR    OUT_LINE4       ; back to OUT-LINE4 until entire line is done.

; ---

OUT_LINE6 POP   DE              ; bring back the flag E, zero if current line
                                ; printed else value one if still to print.

          RET                   ; return - with A holding $0D

; ------------------------------
; THE 'CHANNEL LETTER' SUBROUTINE
; ------------------------------
;

IN_CHAN_K LD    HL,($5B51)      ; fetch address of current channel CURCHL
          JR    NUMBER_4        ; forward to pick up channel letter.

; ----------------------
; THE 'NUMBER' SUBROUTINE
; ----------------------
;   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    CP    $0E             ; character fourteen ?
          RET   NZ              ; return if not.

          INC   HL              ; skip the character
NUMBER_5  INC   HL              ; and five bytes
NUMBER_4  INC   HL              ; following.
NUMBER_3  INC   HL              ;
          INC   HL              ;
          INC   HL              ;
          LD    A,(HL)          ; fetch the following character
          CP    $4B             ;+ default comparison - is it letter 'K' ?
          RET                   ; for return value.

; ------------------------------------------
; THE 'PRINT A FLASHING CHARACTER' SUBROUTINE
; ------------------------------------------
;   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 PRINT_OUT
;   can be used, even though out-line outputs to other streams.
;   In fact the alternate set is used for the whole routine.

OUT_FLASH EXX                   ; Switch in alternate set

;;;       LD    HL,($5B8F)      ; fetch L = ATTR_T, H = MASK-T
;;;       PUSH  HL              ; preserve original value and masks.

;;;       RES   7,H             ; reset flash mask bit so active.
;;;       SET   7,L             ; make attribute FLASH.
;;;       LD    ($5B8F),HL      ; update system variables ATTR_T and MASK-T

;;;       LD    HL,$5B91        ; Address P_FLAG
;;;       LD    D,(HL)          ; fetch value to D
;;;       PUSH  DE              ; and preserve original value.
;;;       LD    (HL),$00        ; clear inverse, over, ink/paper 9

          CALL  PRINT_OUT       ; Routine PRINT-OUT outputs character without
                                ; the need to vector via RST 10.

          EX    DE,HL           ;+ Note. NEW transfer attribute byte to HL.

          SET   7,(HL)          ;+ Make it flash.

;;;       POP   HL              ; pop the original P_FLAG to H.
;;;       LD    (IY+$57),H      ; and restore system variable P_FLAG.

;;;       POP   HL              ; restore original attribute and mask
;;;       LD    ($5B8F),HL      ; and restore system variables ATTR_T/MASK_T

          EXX                   ; Switch back to main set

          RET                   ; Return

; --------------------------------
; THE 'PRINT THE CURSOR' SUBROUTINE
; --------------------------------
;   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
;   the 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  LD    HL,($5B5B)      ; 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,($5B41)       ; fetch MODE  0='KLC', 1='E', 2='G'.
          RLC   A               ; double the value and set flags.
          JR    Z,OUT_C_1       ; forward, if still zero, to OUT-C-1 ('KLC').

          ADD   A,$43           ; Add 'C' - will become 'E' if originally 1
                                ; or 'G' if originally 2.

          JR    OUT_C_2         ; 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   LD    HL,$5B3B        ; 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,OUT_C_2       ; forward, if so to print, at OUT-C-2

          SET   3,(HL)          ; signal 'L' mode to the interrupt routine.
                                ; Note. transient bit has been made permanent.

          INC   A               ; Augment character from 'K' to 'L'.

          BIT   3,(IY+$30)      ; test FLAGS2 - consider caps lock ?
                                ; which is maintained by KEY-INPUT.

          JR    Z,OUT_C_2       ; forward, if not set to print, at OUT-C-2

          LD    A,$43           ; alter character 'L' to 'C'.

;;;       PUSH  DE              ; save address pointer but OK as OUT-FLASH
;;;                             ; uses alternate set without RST 10.

OUT_C_2   JR  OUT_FLASH         ;+ routine OUT-FLASH to print.

;;;       POP   DE              ; restore and

;;;       RET                   ; return. (replace CALL,RET with a JR)

; -------------------------
; THE 'LN_FETCH' SUBROUTINE
; -------------------------
;   These two subroutines are called while editing.
;   The first 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  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  LINE_ADDR       ; routine LINE-ADDR gets address in HL.

          CALL  LINE_NO         ; 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

;;; L191C:   BIT   5,(IY+$37)   ; test FLAGX - input mode ?
;;;           RET   NZ            ; return if not edit mode.
;;;                               ; Note. above already checked by ED-UP/ED-DOWN.

LN_STORE  LD    (HL),D          ; save high byte of line number.
          DEC   HL              ; address lower
          LD    (HL),E          ; save low byte of line number.

          RET                   ; return.

; ---------------------------------------------
; THE 'OUTPUT NUMBERS IN BASIC LINE' ROUTINE
; ---------------------------------------------
;   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 is output by
;   space(out_char), 2(out_code), 0(out_char) final number always out-code.

OUT_SP_2  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)
          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    OUT_CHAR        ; forward to exit via OUT-CHAR.

; ---

;   -> the single entry point.

OUT_SP_NO XOR   A               ; initialize digit to 0

OUT_SP_1  ADD   HL,BC           ; add negative number to HL.
          INC   A               ; increment digit
          JR    C,OUT_SP_1      ; 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,OUT_SP_2      ; back to OUT-SP-2 if it is zero.

          JP    OUT_CODE        ; jump back to exit via OUT-CODE.    ->


; -----------------------------------------------------
; THE 'OUTPUT CHARACTERS IN A BASIC LINE' SUBROUTINE
; -----------------------------------------------------
;   This subroutine ...

OUT_CHAR  CALL  NUMERIC         ; routine NUMERIC tests if it is a digit ?

          JR    NC,OUT_CH_3     ; 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,OUT_CH_3      ; 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,OUT_CH_3      ; forward, if so, to OUT-CH-3

          CP    $3A             ; is character ':' ?
          JR    NZ,OUT_CH_1     ; forward, if not, to OUT-CH-1
                                ; to change mode back to 'L'.

;;;       BIT   5,(IY+$37)      ; FLAGX  - Input Mode ??
          CALL  TST_INP_M       ;+ bit 5,(iy+$37) as a 3-byte call.
          JR    NZ,OUT_CH_2     ; forward, if in INPUT, to OUT-CH-2
                                ; 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,OUT_CH_3      ; forward, if not, to OUT-CH-3

          JR    OUT_CH_2        ; forward to OUT-CH-2 as ':' is within quotes

; ---

OUT_CH_1  CP    $22             ; is it the quote character '"'  ?
          JR    NZ,OUT_CH_2     ; forward, with others, to OUT-CH-2
                                ; to set 'L' mode.

          PUSH  AF              ; save character.
          LD    A,($5B6A)       ; fetch FLAGS2.
          XOR   $04             ; toggle the quotes flag - BIT 2, FLAGS2
          LD    ($5B6A),A       ; update FLAGS2
          POP   AF              ; and restore character.

OUT_CH_2  SET   2,(IY+$01)      ; update FLAGS - signal L mode if the cursor
                                ; is next.

OUT_CH_3  RST   10H             ; PRINT-A vectors the character to
                                ; channel 'S', 'K', 'R' or 'P'.
          RET                   ; return.

; ----------------------------
; THE 'LINE ADDRESS' SUBROUTINE
; ----------------------------
;   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 PUSH  HL              ; save line number in HL register
          LD    HL,($5B53)      ; fetch start of program from PROG
          LD    D,H             ; transfer address to
          LD    E,L             ; the DE register pair.

LINE_AD_1 POP   BC              ; restore the line number to BC

          CALL  CP_LINES        ; 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  NEXT_ONE        ; routine NEXT-ONE finds address of next
                                ; line number in DE, previous in HL.

          EX    DE,HL           ; switch so next in HL
          JR    LINE_AD_1       ; back, for another comparison, to LINE-AD-1

; ------------------------------------
; THE 'COMPARE LINE NUMBERS' SUBROUTINE
; ------------------------------------
;   This routine compares a line number supplied in BC with an addressed
;   line number pointed to by HL.

CP_LINES  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.

; -----------------------------------
; THE 'FIND EACH STATEMENT' SUBROUTINE
; -----------------------------------
;   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.

;;; L1988:    INC   HL              ; not used
;;;           INC   HL              ; not used
;;;           INC   HL              ; not used

;   -> entry point.

EACH_STMT LD    ($5B5D),HL      ; Save HL in CH_ADD
          LD    C,$00           ; Initialize the quotes flag

EACH_S_1  DEC   D               ; Decrease the statement count
          RET   Z               ; Return if zero


          RST   20H             ; NEXT-CHAR
          CP    E               ; Is it the search token ?
          JR    NZ,EACH_S_3     ; Forward, if not, to EACH-S-3

          AND   A               ; clear carry

          RET                   ; return signaling success.

; ---

EACH_S_2  INC   HL              ; next address
          LD    A,(HL)          ; next character

EACH_S_3  CALL  NUMBER          ; routine NUMBER skips if number marker

          LD    ($5B5D),HL      ; save character address in CH_ADD
          CP    $22             ; is it quotes character '"' ?
          JR    NZ,EACH_S_4     ; forward, if not, to EACH-S-4

          DEC   C               ; toggle bit 0 of C

EACH_S_4  CP    $3A             ; is character ':'
          JR    Z,EACH_S_5      ; forward, if so, to EACH-S-5

          CP    $CB             ; is character 'THEN'
          JR    NZ,EACH_S_6     ; forward, if not, to EACH-S-6

EACH_S_5  BIT   0,C             ; is it within quotes ?
          JR    Z,EACH_S_1      ; back, if not, to EACH-S-1

EACH_S_6  CP    $0D             ; end of line ?
          JR    NZ,EACH_S_2     ; back, if not, to EACH-S-2

          DEC   D               ; decrease the statement counter
                                ; which should be zero else
                                ; 'Statement Lost'.
          SCF                   ; set carry flag - signal 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.
;
; -----------------------------------------------------------------------

; -------------------------
; THE 'NEXT ONE' SUBROUTINE
; -------------------------
;   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  PUSH  HL              ; save the pointer address.
          LD    A,(HL)          ; get first byte.
          CP    $40             ; compare with upper limit for line numbers.
          JR    C,NEXT_O_3      ; forward to NEXT-O-3 if within BASIC area.

;   The continuation here is for the next variable.

          BIT   5,A             ; is it a string or an array variable ?
          JR    Z,NEXT_O_4      ; forward to NEXT-O-4 to compute length.
```

```
         ADD   A,A              ; test bit 6 for single-character variables.
         JP    M,NEXT_O_1       ; forward, if so, to NEXT-O-1

         CCF                    ; clear the carry for long-named variables.
                                ; it remains set for for-next loop variables.

NEXT_O_1  LD    BC,$0005        ; set BC to 5 for floating point number
          JR    NC,NEXT_O_2     ; 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  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,NEXT_O_2     ; back to NEXT-O-2 if not inverted bit.
                                ; forward immediately with single character
                                ; variable names.

          JR    NEXT_O_5        ; forward to NEXT-O-5 to add length of
                                ; floating point number(s etc.).

; ---

;   this branch is for line numbers.

NEXT_O_3  INC   HL              ; increment pointer to low byte of line no.

;   strings and arrays rejoin here

NEXT_O_4  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  ADD   HL,BC           ; add the length to give address of next
                                ; line/variable in HL.
          POP   DE              ; restore previous address to DE.

; ---------------------------
; THE 'DIFFERENCE' SUBROUTINE
; ---------------------------
;   This routine terminates the above routine and is also called from the
;   start of the next routine to calculate the length to reclaim.

DIFFER    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.
; --------------------------------
; THE 'NEXT_ONE/RECLAIM_2' ROUTINE
; --------------------------------
;   On three occasions the two subroutines are called in succession so this
;   5-byte routine by James Smith combines the two calls.

NXT_1_RC2 CALL  NEXT_ONE        ;+
          JR    RECLAIM_2       ;+ forward to reclaim space


; -----------------------------
; THE 'RECLAIM ROOM' SUBROUTINE
; -----------------------------
;

RECLAIM_1 CALL  DIFFER          ; routine DIFFER immediately above

RECLAIM_2 PUSH  BC              ;

          LD    A,B             ;
          CPL                   ;
          LD    B,A             ;
          LD    A,C             ;
          CPL                   ;
          LD    C,A             ;
          INC   BC              ;

          CALL  POINTERS        ; routine POINTERS

          EX    DE,HL           ;
          POP   HL              ;

          ADD   HL,DE           ;

          PUSH  DE              ;
          LDIR                  ; copy bytes

          POP   HL              ;

          RET                   ; Return.

; --------------------------------------
; THE 'READ EDIT LINE NUMBER' SUBROUTINE
; --------------------------------------
;   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
;   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 CALL  L_EL_DHL        ;+ NEW routine with below code.

;;;       LD    HL,($5B59)      ; load HL from system variable E_LINE.
;;;       DEC   HL              ; decrease so that NEXT_CHAR can be used
;;;                             ; without skipping the first digit.
```

```
          LD    ($5B5D),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,$5B92        ; use MEM-0 as a temporary calculator stack
                                ; an overhead of three locations are needed.
          LD    ($5B65),HL      ; set new STKEND.

          CALL  INT_TO_FP       ; routine INT-TO-FP will read digits till
                                ; a non-digit found.
          CALL  FP_TO_BC        ; routine FP-TO-BC will retrieve number
                                ; from stack at MEMBOT.
          JR    C,REPORT_Ce     ; forward to E-L-1 if overflow i.e. > 65535.
                                ; 'Nonsense in BASIC'

          LD    HL,$D8F0        ; load HL with the value -9999
          ADD   HL,BC           ; add to line number in BC

;   a line in the range 0 - 9999 has been entered.

          JP    NC,SET_STK      ; jump back to SET-STK to set the calculator
                                ; stack back to its normal place and exit
                                ; from there.

;;; E_L_1 JP    C,REPORT_C      ; to REPORT-C 'Nonsense in BASIC' if over.

REPORT_Ce RST   30H             ; ERROR-1
          DEFB  $0B             ; 'Nonsense in BASIC'


;;;       JP    SET_STK         ; jump back to SET-STK

; ------------------------------------------------
; THE 'REPORT AND LINE NUMBER PRINTING' SUBROUTINE
; ------------------------------------------------
;   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_0 LD    B,$00           ;+ New entry point to print C

OUT_NUM_1 PUSH  DE              ; save the
          PUSH  HL              ; registers.

          XOR   A               ; set A to zero.
          BIT   7,B             ; is the line number minus two ?
          JR    NZ,OUT_NUM_4    ; forward, if so, to OUT-NUM-4
                                ; 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    OUT_NUM_3       ; forward to continue at OUT-NUM-3

; ---

;   Entry point from OUT-LINE - HL addresses line number.
```

```
OUT_NUM_2 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 LD    BC,$FC18        ; value -1000
          CALL  OUT_SP_NO       ; routine OUT-SP-NO outputs space or number

          LD    BC,$FF9C        ; value -100
          CALL  OUT_SP_NO       ; routine OUT-SP-NO

          LD    C,$F6           ; value -10 ( B is still $FF )
          CALL  OUT_SP_NO       ; routine OUT-SP-NO

          LD    A,L             ; remainder to A.

OUT_NUM_4 CALL  OUT_CODE        ; routine OUT-CODE for final digit.
                                ; else report code zero wouldn't get printed.

          POP   HL              ; Restore the
          POP   DE              ; registers.

          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 DEFB  P_DEF_FN  - $   ; B1 offset to Address: P-DEF-FN
          DEFB  P_CAT     - $   ; CB offset to Address: P-CAT
          DEFB  P_FORMAT  - $   ; BC offset to Address: P-FORMAT
          DEFB  P_MOVE    - $   ; BF offset to Address: P-MOVE
          DEFB  P_ERASE   - $   ; C4 offset to Address: P-ERASE
          DEFB  P_OPEN    - $   ; AF offset to Address: P-OPEN
          DEFB  P_CLOSE   - $   ; B4 offset to Address: P-CLOSE
          DEFB  P_MERGE   - $   ; 93 offset to Address: P-MERGE
          DEFB  P_VERIFY  - $   ; 91 offset to Address: P-VERIFY
          DEFB  P_BEEP    - $   ; 92 offset to Address: P-BEEP
          DEFB  P_CIRCLE  - $   ; 95 offset to Address: P-CIRCLE
          DEFB  P_INK     - $   ; 98 offset to Address: P-INK
          DEFB  P_PAPER   - $   ; 98 offset to Address: P-PAPER
          DEFB  P_FLASH   - $   ; 98 offset to Address: P-FLASH
          DEFB  P_BRIGHT  - $   ; 98 offset to Address: P-BRIGHT
          DEFB  P_INVERSE - $   ; 98 offset to Address: P-INVERSE
          DEFB  P_OVER    - $   ; 98 offset to Address: P-OVER
          DEFB  P_OUT     - $   ; 98 offset to Address: P-OUT
          DEFB  P_LPRINT  - $   ; 7F offset to Address: P-LPRINT
          DEFB  P_LLIST   - $   ; 81 offset to Address: P-LLIST
          DEFB  P_STOP    - $   ; 2E offset to Address: P-STOP
```

```
        DEFB  P_READ    - $   ; 6C offset to Address: P-READ
        DEFB  P_DATA    - $   ; 6E offset to Address: P-DATA
        DEFB  P_RESTORE - $   ; 70 offset to Address: P-RESTORE
        DEFB  P_NEW     - $   ; 48 offset to Address: P-NEW
        DEFB  P_BORDER  - $   ; 94 offset to Address: P-BORDER
        DEFB  P_CONT    - $   ; 56 offset to Address: P-CONT
        DEFB  P_DIM     - $   ; 3F offset to Address: P-DIM
        DEFB  P_REM     - $   ; 41 offset to Address: P-REM
        DEFB  P_FOR     - $   ; 2B offset to Address: P-FOR
        DEFB  P_GO_TO   - $   ; 17 offset to Address: P-GO-TO
        DEFB  P_GO_SUB  - $   ; 1F offset to Address: P-GO-SUB
        DEFB  P_INPUT   - $   ; 37 offset to Address: P-INPUT
        DEFB  P_LOAD    - $   ; 77 offset to Address: P-LOAD
        DEFB  P_LIST    - $   ; 44 offset to Address: P-LIST
        DEFB  P_LET     - $   ; 0F offset to Address: P-LET
        DEFB  P_PAUSE   - $   ; 59 offset to Address: P-PAUSE
        DEFB  P_NEXT    - $   ; 2B offset to Address: P-NEXT
        DEFB  P_POKE    - $   ; 43 offset to Address: P-POKE
        DEFB  P_PRINT   - $   ; 2D offset to Address: P-PRINT
        DEFB  P_PLOT    - $   ; 51 offset to Address: P-PLOT
        DEFB  P_RUN     - $   ; 3A offset to Address: P-RUN
        DEFB  P_SAVE    - $   ; 6D offset to Address: P-SAVE
        DEFB  P_RANDOM  - $   ; 42 offset to Address: P-RANDOM
        DEFB  P_IF      - $   ; 0D offset to Address: P-IF
        DEFB  P_CLS     - $   ; 49 offset to Address: P-CLS
        DEFB  P_DRAW    - $   ; 5C offset to Address: P-DRAW
        DEFB  P_CLEAR   - $   ; 44 offset to Address: P-CLEAR
        DEFB  P_RETURN  - $   ; 15 offset to Address: P-RETURN
        DEFB  P_COPY    - $   ; 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   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 DEFB  $06             ; Class-06 - A numeric expression must follow.
        DEFB  $00             ; Class-00 - No further operands.
        DEFW  GO_TO           ; Address: GO-TO

P_IF    DEFB  $06             ; Class-06 - A numeric expression must follow.
        DEFB  $CB             ; Separator:  'THEN'
        DEFB  $05             ; Class-05 - Variable syntax checked
                              ; by routine.
        DEFW  IF              ; Address: IF
```

```
P_GO_SUB  DEFB  $06             ; Class-06 - A numeric expression must follow.
          DEFB  $00             ; Class-00 - No further operands.
          DEFW  GO_SUB          ; Address: GO-SUB

P_STOP    DEFB  $00             ; Class-00 - No further operands.
          DEFW  STOP            ; Address: STOP

P_RETURN  DEFB  $00             ; Class-00 - No further operands.
          DEFW  RETURN          ; Address: RETURN

P_FOR     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  FOR             ; Address: FOR

P_NEXT    DEFB  $04             ; Class-04 - A single character variable must
                                ; follow.
          DEFB  $00             ; Class-00 - No further operands.
          DEFW  NEXT            ; Address: NEXT

P_PRINT   DEFB  $05             ; Class-05 - Variable syntax checked entirely
                                ; by routine.
          DEFW  PRINT           ; Address: PRINT

P_INPUT   DEFB  $05             ; Class-05 - Variable syntax checked entirely
                                ; by routine.
          DEFW  INPUT           ; Address: INPUT

P_DIM     DEFB  $05             ; Class-05 - Variable syntax checked entirely
                                ; by routine.
          DEFW  DIM             ; Address: DIM

P_REM     DEFB  $05             ; Class-05 - Variable syntax checked entirely
                                ; by routine.
          DEFW  REM             ; Address: REM

P_NEW     DEFB  $00             ; Class-00 - No further operands.
          DEFW  NEW             ; Address: NEW

P_RUN     DEFB  $03             ; Class-03 - A numeric expression may follow
                                ; else default to zero.
          DEFW  RUN             ; Address: RUN

P_LIST    DEFB  $05             ; Class-05 - Variable syntax checked entirely
                                ; by routine.
          DEFW  LIST            ; Address: LIST

P_POKE    DEFB  $08             ; Class-08 - Two comma-separated numeric
                                ; expressions required.
          DEFB  $00             ; Class-00 - No further operands.
          DEFW  POKE            ; Address: POKE

P_RANDOM  DEFB  $03             ; Class-03 - A numeric expression may follow
                                ; else default to zero.
          DEFW  RANDOMIZE       ; Address: RANDOMIZE

P_CONT    DEFB  $00             ; Class-00 - No further operands.
          DEFW  CONTINUE        ; Address: CONTINUE
```

```
P_CLEAR
;;;       DEFB  $03                ; Class-03 - A numeric expression may follow
;;;                                ; else default to zero.
          DEFB  $05                ;+ Variable syntax checked by routine.
          DEFW  CLEAR              ; Address: CLEAR

P_CLS     DEFB  $00                ; Class-00 - No further operands.
          DEFW  CLS                ; Address: CLS

P_PLOT    DEFB  $09                ; Class-09 - Two comma-separated numeric
                                   ; expressions required with optional colour
                                   ; items.
          DEFB  $00                ; Class-00 - No further operands.
          DEFW  PLOT               ; Address: PLOT

P_PAUSE   DEFB  $06                ; Class-06 - A numeric expression must follow.
          DEFB  $00                ; Class-00 - No further operands.
          DEFW  PAUSE              ; Address: PAUSE

P_READ    DEFB  $05                ; Class-05 - Variable syntax checked entirely
                                   ; by routine.
          DEFW  READ               ; Address: READ

P_DATA    DEFB  $05                ; Class-05 - Variable syntax checked entirely
                                   ; by routine.
          DEFW  DATA               ; Address: DATA

P_RESTORE DEFB  $03                ; Class-03 - A numeric expression may follow
                                   ; else default to zero.
          DEFW  RESTORE            ; Address: RESTORE

P_DRAW    DEFB  $09                ; Class-09 - Two comma-separated numeric
                                   ; expressions required with optional colour
                                   ; items.
          DEFB  $05                ; Class-05 - Variable syntax checked
                                   ; by routine.
          DEFW  DRAW               ; Address: DRAW

P_COPY    DEFB  $00                ; Class-00 - No further operands.
          DEFW  COPY               ; Address: COPY

P_LPRINT  DEFB  $05                ; Class-05 - Variable syntax checked entirely
                                   ; by routine.
          DEFW  LPRINT             ; Address: LPRINT

P_LLIST   DEFB  $05                ; Class-05 - Variable syntax checked entirely
                                   ; by routine.
          DEFW  LLIST              ; Address: LLIST

P_SAVE    DEFB  $0B                ; Class-0B - Offset address converted to tape
                                   ; command.

P_LOAD    DEFB  $0B                ; Class-0B - Offset address converted to tape
                                   ; command.

P_VERIFY  DEFB  $0B                ; Class-0B - Offset address converted to tape
                                   ; command.

P_MERGE   DEFB  $0B                ; Class-0B - Offset address converted to tape
                                   ; command.

P_BEEP    DEFB  $08                ; Class-08 - Two comma-separated numeric
                                   ; expressions required.
          DEFB  $00                ; Class-00 - No further operands.
          DEFW  BEEP             ; Address: BEEP

P_CIRCLE  DEFB  $09              ; Class-09 - Two comma-separated numeric
                                 ; expressions required with optional colour
                                 ; items.
          DEFB  $05              ; Class-05 - Variable syntax checked
                                 ; by routine.
          DEFW  CIRCLE           ; Address: CIRCLE

P_INK     DEFB  $07              ; Class-07 - Offset address is converted to
                                 ; colour code.

P_PAPER   DEFB  $07              ; Class-07 - Offset address is converted to
                                 ; colour code.

P_FLASH   DEFB  $07              ; Class-07 - Offset address is converted to
                                 ; colour code.

P_BRIGHT  DEFB  $07              ; Class-07 - Offset address is converted to
                                 ; colour code.

P_INVERSE DEFB  $07              ; Class-07 - Offset address is converted to
                                 ; colour code.

P_OVER    DEFB  $07              ; Class-07 - Offset address is converted to
                                 ; colour code.

P_OUT     DEFB  $08              ; Class-08 - Two comma-separated numeric
                                 ; expressions required.
          DEFB  $00              ; Class-00 - No further operands.
          DEFW  OUT              ; Address: OUT

P_BORDER  DEFB  $06              ; Class-06 - A numeric expression must follow.
          DEFB  $00              ; Class-00 - No further operands.
          DEFW  BORDER           ; Address: BORDER

P_DEF_FN  DEFB  $05              ; Class-05 - Variable syntax checked entirely
                                 ; by routine.
          DEFW  DEF_FN           ; Address: DEF-FN

P_OPEN    DEFB  $06              ; Class-06 - A numeric expression must follow.
;;;       DEFB  $2C              ; Separator:  ','
          DEFB  $0C              ;+ Class-0C - NEW either ';' or ','
          DEFB  $0A              ; Class-0A - A string expression must follow.
;;;       DEFB  $00              ; Class-00 - Was No further operands.
          DEFB  $05              ;+ Class-05 - New Variable syntax.
          DEFW  OPEN             ; Address: OPEN

P_CLOSE   DEFB  $06              ; Class-06 - A numeric expression must follow.
          DEFB  $00              ; Class-00 - No further operands.
          DEFW  CLOSE            ; Address: CLOSE

P_FORMAT  DEFB  $0A              ; Class-0A - A string expression must follow.
          DEFB  $0C              ; Class-0C - NEW either ';' or ','
          DEFB  $06              ; Class-06 - A numeric expression must follow.
          DEFB  $00              ; Class-00 - No further operands.
          DEFW  FORMAT           ; Address: FORMAT

;   Since the commands MOVE ERASE and CAT will not be used then the syntax
;   can be removed and they can all use the CAT error-generating routine.

P_MOVE
;;;       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  CAT_ETC           ; Address: CAT-ETC

P_ERASE
;;;       DEFB  $0A               ; Class-0A - A string expression must follow.

          DEFB  $00               ; Class-00 - No further operands.
          DEFW  REPORT_O          ; Address: REPORT_O - Invalid stream

P_CAT     DEFB  $00               ; Class-00 - No further operands.
          DEFW  CAT               ; Address: CAT -


; -----------------------
; THE 'LINE SCAN' ROUTINE
; -----------------------
;   The 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 RES   7,(IY+$01)        ; update FLAGS - signal checking syntax

          CALL  E_LINE_NO         ; routine E-LINE-NO                        >>
                                  ; fetches the line number if in range.

          XOR   A                 ; clear the accumulator.
          LD    ($5B47),A         ; set statement number SUBPPC to zero.
          DEC   A                 ; set accumulator to $FF.
          LD    ($5B3A),A         ; set ERR_NR to 'OK' - 1.

          JR    STMT_L_1          ; forward to continue at STMT-L-1.

; --------------------
; THE 'STATEMENT' LOOP
; --------------------
;
;

STMT_LOOP RST   20H               ; NEXT-CHAR

;   -> the entry point from above or LINE-RUN

STMT_L_1  CALL  SET_WORK          ; routine SET-WORK clears workspace etc.

          INC   (IY+$0D)          ; increment statement number SUBPPC
          JP    M,REPORT_C        ; back, if over 127, to REPORT-C
                                  ; 'Nonsense in BASIC'

          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,LINE_END        ; forward, if so, to LINE-END

          CP    $3A               ; is it statement end marker ':' ?
                                  ; i.e. another type of empty statement.
          JR    Z,STMT_LOOP       ; back, if so, to STMT-LOOP

          LD    HL,STMT_RET       ; 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
          JR    C,SEP_RPT_C     ; jump, if less than a command, to REPORT-C
                                ; 'Nonsense in BASIC'

          LD    C,A             ; put the valid command code back in C.
                                ; register B is zero.
          LD    HL,offst_tbl    ; 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    GET_PARAM       ; forward to continue at GET-PARAM

; --------------------
; THE 'MAIN SCAN' LOOP
; --------------------
;

SCAN_LOOP LD    HL,($5B74)      ; Fetch Table Address from T_ADDR during
                                ; subsequent loops.

;   -> the initial entry point with HL addressing start of syntax table entry.

GET_PARAM LD    A,(HL)          ; Pick up the parameter.
          INC   HL              ; Address next one.
          LD    ($5B74),HL      ; Save pointer in system variable T_ADDR

;;;       LD    BC,SCAN_LOOP    ; Address: SCAN-LOOP
;;;       PUSH  BC              ; is now pushed on stack as looping address.

          LD    HL,SCAN_LOOP    ;+ address: SCAN-LOOP
          PUSH  HL              ;+ is now pushed on stack as looping address.

          LD    C,A             ; store parameter in C.
          CP    $20             ; is it greater than ' '  ?
          JR    NC,SEPARATOR    ; forward, if so, to SEPARATOR

          LD    HL,CLASS_TBL    ; 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                   ; Make an indirect jump to routine
                                ; and then to 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.
; -----------------------
; THE 'SEPARATOR' ROUTINE
; -----------------------
;   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 RST   18H             ; GET-CHAR
          CP    C               ; does it match the character in C ?

SEP_RPT_C JP    NZ,REPORT_C     ; jump forward, if not, to REPORT-C
                                ; 'Nonsense in BASIC'.

          RST   20H             ; NEXT-CHAR advance to next character
          RET                   ; return.

; ----------------------------
; THE 'STATEMENT RETURN' POINT
; ----------------------------
;   Control returns to this point after every statement by virtue of the
;   address pushed on the machine stack.

STMT_RET  CALL  TEST_BRK        ;+ the BREAK KEY is tested after every
statement.

;;;       JR    C,STMT_R_1      ; step forward to STMT-R-1 if not pressed.
;;; REPORT_L RST   30H          ; ERROR-1
;;;       DEFB  $14             ; Error Report: BREAK into program

; ---

STMT_R_1  BIT   7,(IY+$0A)      ; test a bit of NSPPC - will be set if $FF -
                                ; no jump to be made.
          JR    NZ,STMT_NEXT    ; forward, if no jump, to STMT-NEXT

          LD    HL,($5B42)      ; fetch BASIC line number from NEWPPC

          BIT   7,H             ; test the high order byte.
                                ; bit 7 is set if minus two - direct command(s)

          JR    Z,LINE_NEW      ; forward, if a jump is to be made, to LINE-NEW

; ---------------------------------
; THE 'RUN A DIRECT COMMAND' ROUTINE
; ---------------------------------
;   A direct command is to be run or, if continuing from above, the next
;   statement in a sequence of direct commands is to be considered.

LINE_RUN  LD    HL,$FFFE        ; The dummy value minus two
          LD    ($5B45),HL      ; is set/reset as line number in PPC.

          LD    HL,($5B61)      ; point to the start of workspace WORKSP.
          DEC   HL              ; now point to $80 Edit Line end-marker.
          LD    DE,($5B59)      ; address the start of line using E_LINE.

          DEC   DE              ; now location before - for GET-CHAR.

          LD    A,($5B44)       ; load statement to A from NSPPC.

          JR    NEXT_LINE       ; forward to NEXT-LINE.

; ----------------------
; THE 'LINE NEW' ROUTINE
; ---------------------
;   This routine finds the 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  CALL  LINE_ADDR       ; routine LINE-ADDR gets address of line
                                ; returning zero flag set if line found.
          LD    A,($5B44)       ; fetch new statement from NSPPC
          JR    Z,LINE_USE      ; forward to LINE-USE if line matched.

;   continue as must be a direct command.

          AND   A               ; test statement which should be zero
          JR    NZ,REPORT_N     ; forward, if not, to REPORT-N
                                ; '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,LIN_USE_0     ; forward to LINE-USE if was a program line

;   Alternatively, a direct statement has finished correctly.

REPORT_0  RST   30H             ; ERROR-1
          DEFB  $FF             ; Error Report: OK

; -----------------
; THE 'REM' COMMAND
; -----------------
;   The REM command routine.
;   The return address STMT-RET is dropped and the rest of line ignored.

REM       POP   BC              ; drop return address STMT-RET and
                                ; continue ignoring rest of line.

; ------------
; End of line?
; ------------
;
;

;;; LINE_END  CALL  SYNTAX_Z    ; routine SYNTAX_Z  (UNSTACK_Z?)
;;;           RET   Z           ; return if checking syntax.

LINE_END  CALL  UNSTACK_Z       ;+ return early if checking syntax.

          LD    HL,($5B55)      ; fetch NXTLIN to HL.
          LD    A,$C0           ; test against the
          AND   (HL)            ; system limit $3F.
          RET   NZ              ; return if higher as must be end of program.
                                ; (or direct command)

LIN_USE_0 XOR   A               ; set statement to zero.

;   and continue to set up the next following line and then consider this new
one.
```

```
; ---------------------
; THE 'LINE USE' BRANCH
; ---------------------
;   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  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    ($5B45),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 adding
          ADD   HL,DE            ; to address the end of the line.
          INC   HL               ; advance to start of next line.

; -------------------------
; THE 'NEXT LINE' SUBROUTINE
; -------------------------
;   The pointer will be the next line if continuing from above or to edit line
;   end-marker ($80) if from LINE-RUN.

NEXT_LINE LD    ($5B55),HL       ; store pointer in system variable NXTLIN

          EX    DE,HL            ; bring back pointer to previous or edit line
          LD    ($5B5D),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 signaling
                                 ; no jump to be made.
          DEC   D                ; decrement and test statement
          LD    (IY+$0D),D       ; set SUBPPC to decremented statement number.
          JP    Z,STMT_LOOP      ; to STMT-LOOP if result zero as statement is
                                 ; at start of line and address is known.

          INC   D                ; else restore statement.
          CALL  EACH_STMT        ; routine EACH-STMT finds the D'th statement
                                 ; address as E does not contain a token.
          JR    Z,STMT_NEXT      ; forward to STMT-NEXT if address found.

REPORT_N  RST   30H              ; ERROR-1
          DEFB  $16              ; 'Statement lost'

; ---------------------------------------------
; THE NEW 'CHECK FOR NUMBER AND SYNTAX' ROUTINE
; ---------------------------------------------
;   Combines two or three routines into one call.

CHK_END_1 RST   20H              ;+ NEXT_CHAR

CHK_END_2 CALL  EXPT_1NUM        ;+ Check for 1 number and stack in runtime

; -------------------------
; THE 'CHECK END' SUBROUTINE
; -------------------------
;   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 CALL  SYNTAX_Z        ; 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.

; ---------------------------
; THE 'STATEMENT NEXT' ROUTINE
; ---------------------------
;   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 RST   18H             ; GET-CHAR - ignoring white space etc.

          CP    $0D             ; is character carriage return ?
          JR    Z,LINE_END      ; back, if so, to LINE-END

          CP    $3A             ; is character a ':' ?
          JP    Z,STMT_LOOP     ; jump back, if so, to STMT-LOOP

          JR    VAL_RPT_C       ; forward, with any other, to VAL_RPT_C
                                ; 'Nonsense in BASIC'

; ------------------------
; THE 'COMMAND CLASS' TABLE
; ------------------------
;

CLASS_TBL DEFB  CLASS_00   - $  ; offset to Address: CLASS-00
          DEFB  CLASS_01   - $  ; offset to Address: CLASS-01
          DEFB  CLASS_02   - $  ; offset to Address: CLASS-02
          DEFB  CLASS_03   - $  ; offset to Address: CLASS-03
          DEFB  CLASS_04   - $  ; offset to Address: CLASS-04
          DEFB  CLASS_05   - $  ; offset to Address: CLASS-05
          DEFB  CLASS_06   - $  ; offset to Address: CLASS-06
          DEFB  CLASS_07   - $  ; offset to Address: CLASS-07
          DEFB  CLASS_08   - $  ; offset to Address: CLASS-08
          DEFB  CLASS_09   - $  ; offset to Address: CLASS-09
          DEFB  CLASS_0A   - $  ; offset to Address: CLASS-0A
          DEFB  CLASS_0B   - $  ; offset to Address: CLASS-0B

          DEFB  CLASS_0C   - $  ; offset to Address: CLASS_0C


; ----------------------------------------------
; THE 'COMMAND CLASSES 00, 03 and 05' ROUTINES
; ----------------------------------------------
;   class-03 e.g. RUN or RUN 20 ;  optional operand.
;   class-00 e.g. CONTINUE      ;  no operand.
;   class-05 e.g. PRINT         ;  variable syntax checked by routine.

CLASS_03  CALL  FETCH_NUM       ; routine FETCH-NUM


CLASS_00  CP    A               ; set zero flag.

;   if entering here then all class routines are entered with zero reset.
```

```
CLASS_05  POP   BC               ; drop address SCAN-LOOP.
          CALL  Z,CHECK_END      ; if zero set then call routine CHECK-END >>>
                                 ; as should be no further characters.

;   If checking syntax then classes 00 and 03 terminate at the above step.

          EX    DE,HL            ; save HL to DE.
          LD    HL,($5B74)       ; 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.

; ------------------------------
; THE 'COMMAND CLASS 01' ROUTINE
; ------------------------------
;   e.g. LET A = 2*3             ; A variable is required.

;   This class routine is also called from INPUT and READ to find the
;   destination variable for an assignment.

CLASS_01  CALL  LOOK_VARS        ; routine LOOK-VARS returns carry set if the
                                 ; variable is not found in runtime.


VAR_A_1   LD    (IY+$37),$00     ; Set FLAGX to zero
          JR    NC,VAR_A_2       ; Forward, if found or syntax path, to VAR-A-2

;   The variable was not found in runtime.

          SET   1,(IY+$37)       ; Update FLAGX - signal a new variable.

          JR    NZ,VAR_A_3       ; Forward, if not array subscript, to VAR-A-3
                                 ; e.g. LET a$(3,3) = "X"

REPORT_2  RST   30H              ; ERROR-1
          DEFB  $01              ; Error Report: Variable not found.

; ---

;    The branch was here when the variable was found or if checking syntax.

VAR_A_2   CALL  Z,STK_VAR        ; routine STK-VAR considers a subscript/slice.
          BIT   6,(IY+$01)       ; test FLAGS - numeric or string result ?
          JR    NZ,VAR_A_3       ; forward, if numeric, to VAR-A-3.

          XOR   A                ; Default A to array/slice - to be retained.

          CALL  SYNTAX_Z         ; Routine SYNTAX-Z
          CALL  NZ,STK_FETCH     ; Routine STK-FETCH is called in runtime
                                 ; may overwrite A with 1.

          LD    HL,$5B71         ; Address the FLAGX system variable.
          OR    (HL)             ; sets bit 0 if simple variable to be reclaimed.
          LD    (HL),A           ; update bit 0 of FLAGX
          EX    DE,HL            ; bring start of string/subscript to HL

VAR_A_3   LD    ($5B72),BC       ; update STRLEN system variable.
          LD    ($5B4D),HL       ; update DEST of assigned string.
          RET                    ; Return.
```

```
; ------------------------------
; THE 'COMMAND CLASS 02' ROUTINE
; ------------------------------
;   This is only used in the LET command.
;
;   e.g. LET A = 2*3            ; an expression must follow the separator.


CLASS_02  POP   BC              ; drop the return address SCAN-LOOP

          CALL  VAL_FET_1       ; routine VAL-FET-1 is called to check
                                ; expression and assign result in runtime.

          CALL  CHECK_END       ; routine CHECK-END checks nothing else
                                ; is present in statement.

          RET                   ; Return in runtime also.

; ------------------------------
; THE 'FETCH A VALUE' SUBROUTINE
; ------------------------------
;
;

VAL_FET_1 LD    A,($5B3B)       ; fetch initial FLAGS system variable to A.

VAL_FET_2 PUSH  AF              ; Save initial flags A briefly

          CALL  SCANNING        ; routine SCANNING evaluates expression.

          POP   AF              ; Restore the initial flags - A.

          LD    D,(IY+$01)      ; Fetch post-scanning FLAGS value to D
          XOR   D               ; XOR the before and after flags.
          AND   $40             ; isolate bit 6 of result.

VAL_RPT_C JR    NZ,REPORT_C     ; Forward, if not zero, to REPORT-C
                                ; 'Nonsense in BASIC'

          BIT   7,D             ; Test FLAGS - is syntax being checked ?

          JP    NZ,LET          ; Jump forward, in runtime, to LET
                                ; to make the assignment.

          RET                   ; Return from here when checking syntax.

; ------------------------------
; THE 'COMMAND CLASS 04' ROUTINE
; ------------------------------
;   e.g. FOR i                  ; a single character variable must follow


CLASS_04  CALL  LOOK_VARS       ; routine LOOK-VARS

          PUSH  AF              ; preserve flags.

          LD    A,C             ; fetch type - should be 011xxxxx
          OR    $9F             ; combine with 10011111.
          INC   A               ; test if result is now $FF by incrementing.

          JR    NZ,REPORT_C     ; forward, if result not zero, to REPORT-C
                                ; 'Nonsense in BASIC'

          POP   AF              ; else restore flags.
          JR    VAR_A_1         ; 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 RST   20H             ; NEXT-CHAR advance past 'AT' or '('.

CLASS_08                        ; e.g. POKE 65535,2
                                ; two numeric expressions separated by comma

EXPT_2NUM CALL  EXPT_1NUM       ; routine EXPT-1NUM is called for first
                                ; numeric expression
          CP    $2C             ; is character ',' ?
          JR    NZ,REPORT_C     ; to REPORT-C if not the required separator.
                                ; 'Nonsense in BASIC'.

          RST   20H             ; NEXT-CHAR

;   ->

CLASS_06                        ; e.g. GO TO a*1000
                                ; a numeric expression must follow

EXPT_1NUM CALL  SCANNING        ; routine SCANNING

          BIT   6,(IY+$01)      ; test FLAGS  - Numeric or string result ?

          RET   NZ              ; return if result is numeric.

REPORT_C  RST   30H             ; ERROR-1
          DEFB  $0B             ; Error Report: Nonsense in BASIC

; ------------------------------
; THE 'COMMAND CLASS 0A' ROUTINE
; ------------------------------
;
;   A string expression must follow.  These classes only occur in unimplemented
;   commands although the routine EXPT_EXP is called from SAVE_ETC.
;   It is used in the FORMAT and OPEN syntax tables.

CLASS_0A

EXPT_EXP  CALL  SCANNING        ; routine SCANNING

          BIT   6,(IY+$01)      ; test FLAGS  - Numeric or string result ?

          RET   Z               ; return if string result.

          JR    REPORT_C        ; back, if numeric, to REPORT-C.


; ------------------------------
; THE 'COMMAND CLASS 07' ROUTINE
; ------------------------------
;   Set permanent colours
;   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  BIT   7,(IY+$01)  ; test FLAGS - checking syntax only ?
;;;         RES   0,(IY+$02)      ; update TV_FLAG - signal main screen in use
;;;         CALL  NZ,TEMPs        ; routine TEMPs is called in runtime.

CLASS_07  LD    A,$FE           ;
          CALL  CHN_O_SYN       ;+ ensure control codes go to screen and not a
                                ;+ microdrive file in runtime.
                                ;+ Returns if checking syntax.

          POP   AF              ; drop return address SCAN-LOOP

          LD    A,($5B74)       ; Fetch 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.

          SUB   P_INK-$D8 % 256 ; convert $EB to $D8 ('INK') etc.
                                ; ( was SUB $13 in standard ROM )

          CALL  CO_TEMP_4       ; routine CO-TEMP-4

          CALL  CHECK_END       ; routine CHECK-END check that nothing else
                                ; appears in the statement and quits if
                                ; checking syntax. >>

;   Return to here in runtime.  The temporary attributes set up by CO_TEMP_4
;   are now copied to the permanent attributes to make the change premanent.

          LD    HL,($5B8F)      ; pick up ATTR_T and MASK_T

          LD    ($5B8D),HL      ; and transfer to ATTR_P and MASK_P

          LD    HL,$5B91        ; point to P_FLAG.
          LD    A,(HL)          ; pick up in A
          RLCA                  ; rotate to left
          XOR   (HL)            ; combine with HL
          AND   $AA             ; AND with %10101010
          XOR   (HL)            ; only only the permanent bits affected

          LD    (HL),A          ; reload into system variable P_FLAG.

          RET                   ; Return.

; ------------------------------
; THE 'COMMAND CLASS 09' ROUTINE
; ------------------------------
;   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  CALL  SYNTAX_Z        ; routine SYNTAX_Z
          JR    Z,CL_09_1       ; forward to CL_09_1 if checking syntax.

;;;       RES   0,(IY+$02)      ; update TV_FLAG - signal main screen in use
;;;       CALL  TEMPs           ; routine TEMPs is called in runtime.

          CALL  CHAN_O_FE       ;+ ensure control codes go to screen and not
                                ;+ to the network in runtime.
          LD    HL,$5B90        ; point to MASK_T
          LD    A,(HL)          ; fetch mask to accumulator.
          OR    $F8             ; or with 11111000 paper/bright/flash 8
          LD    (HL),A          ; put 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   CALL  CO_TEMP_2       ; routine CO-TEMP-2 deals with any embedded
                                ; colour items.

          JR    EXPT_2NUM       ; 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.

; ------------------------------
; THE 'COMMAND CLASS 0B' ROUTINE
; ------------------------------
;   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  JP    SAVE_ETC        ; jump way back to SAVE-ETC

; ----------------------------------
; THE NEW 'EXPECT SEPARATOR' ROUTINE
; ----------------------------------
;   Seven bytes
;   Returns with zero flag set if character is a separator.

EXPT_SEP  RST   18H             ; GET_CHAR

          CP    $2C             ; is it a comma
          RET   Z               ;
          CP    $3B             ; is it a semicolon
          RET                   ;

; -----------------------------
; THE NEW 'CLASS 0C' SUBROUTINE
; -----------------------------

CLASS_0C  CALL  EXPT_SEP        ; check for a valid separator ';' or ','.

          JR    NZ,REPORT_C     ; jump forward, if not, to REPORT-C
                                ; 'Nonsense in BASIC'.

NXT_CH    RST   20H             ; NEXT-CHAR advance to next character
          RET                   ; return.


; -------------------------------
; THE 'FETCH A NUMBER' SUBROUTINE
; -------------------------------
;   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.
;   Note. called from SAVE "program" LINE

FETCH_NUM CP    $0D              ; is character a carriage return ?
          JR    Z,USE_ZERO       ; forward, if so, to USE-ZERO

          CP    $3A              ; is it ':' ?
          JR    NZ,EXPT_1NUM     ; back, if not, to EXPT-1NUM
                                 ; else continue and use zero.

; ----------------------
; THE '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  CALL  SYNTAX_Z     ; routine SYNTAX_Z  (UNSTACK_Z?)
;;;           RET   Z            ;

USE_ZERO  CALL  UNSTACK_Z        ;+  return early if checking syntax.

          RST   28H              ;; FP-CALC        .
          DEFB  $A0              ;;stk-zero        0.
          DEFB  $38              ;;end-calc        0.

          RET                    ; Return.


; ------------------
; THE 'STOP' COMMAND
; ------------------
;   Command Syntax: STOP
;   One of the shortest and least used commands.  As with 'OK' not an error.
;   Note. moved to fill a couple of bytes at $0064.


; ----------------
; THE 'IF' COMMAND
; ----------------
;   e.g. IF Warp Factor > 8 THEN PRINT "Och! she'll blow Captain."
;   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        POP   BC               ; drop return address - STMT-RET
          CALL  SYNTAX_Z         ; routine SYNTAX-Z
          JR    Z,IF_1           ; forward, if checking syntax, to IF-1
                                 ; to check syntax of PRINT "Och! She'll blow..."


          RST   28H              ;; FP-CALC    Warp Factor > 8 (1=TRUE 0=FALSE)
          DEFB  $02              ;;delete      .
          DEFB  $38              ;;end-calc

          EX    DE,HL            ; make HL point to deleted value

          CALL  TEST_ZERO        ; routine TEST-ZERO

          JP    C,LINE_END       ; jump to LINE-END if FALSE (0)

IF_1      JP    STMT_L_1         ; to STMT-L-1, if true (1) to execute command
                                 ; after 'THEN' token.

; -----------------
; THE '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 separators.  The two values v,l
;   are on the calculator stack.  The CLASS-04 routine has also checked the
;   variable and the name is in STRLEN_lo.
;   The routine begins by checking for an optional STEP.

FOR       CP    $CD             ; is there a 'STEP' ?
          JR    NZ,F_USE_1      ; Forward, if not, to F-USE-1

;;;       RST   20H             ; NEXT-CHAR
;;;       CALL  EXPT_1NUM       ; routine EXPT-1NUM checks for number
;;;       CALL  CHECK_END       ; routine CHECK-END

          CALL  CHK_END_1       ;+ above three routines

          JR    F_REORDER       ; forward to F-REORDER

; ---

F_USE_1   CALL  CHECK_END       ; routine CHECK-END

          RST   28H             ;; FP-CALC      v,l.
          DEFB  $A1             ;;stk-one       v,l,1=s.
          DEFB  $38             ;;end-calc


F_REORDER 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  LET             ; routine LET assigns the initial value v to
                                ; the variable.

          LD    ($5B68),HL      ; The system variable MEM is made to point to
                                ; the variable instead of its normal location
                                ; at MEMBOT.
          DEC   HL              ; point to the single-character name.
          LD    A,(HL)          ; fetch character.
          SET   7,(HL)          ; set bit 7 at variable location.

          LD    BC,$0006        ; add six to HL to skip the value and
          ADD   HL,BC           ; address where limit should be.

          RLCA                  ; test bit 7 of original variable name.

          JR    C,F_L_S         ; forward, if already correct type, to F-L-S

          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  MAKE_ROOM       ; routine MAKE-ROOM creates them.

;;;       INC   HL              ; make HL address the limit.

F_L_S     PUSH  HL              ; save the limit position.

          RST   28H             ;; FP-CALC         l,s.
          DEFB  $02               ;;delete           l.
          DEFB  $02               ;;delete           .
          DEFB  $38               ;;end-calc         .

;   At this point, DE points to STKEND the start of the two deleted numbers.

          POP   HL                ; restore variable limit position
          EX    DE,HL             ; swap pointers
          LD    C,$0A             ; ten bytes to move

          LDIR                    ; Copy 'deleted' values to limit and step.

          LD    HL,($5B45)        ; Load with current line number from PPC
          EX    DE,HL             ; exchange pointers.

          LD    (HL),E            ; save the looping line in
          INC   HL                ; in the next
          LD    (HL),D            ; two variable locations.

          LD    D,(IY+$0D)        ; fetch statement from SUBPPC system variable.
          INC   D                 ; increment the statement.
          INC   HL                ; increment the variable pointer
          LD    (HL),D            ; and store the looping statement.

          CALL  NEXT_LOOP         ; routine NEXT-LOOP considers an initial
                                  ; iteration.

          RET   NC                ; Return to STMT-RET, if a loop is possible, to
                                  ; execute the next statement.

;   No loop is possible, so execution continues after the matching 'NEXT'

          LD    B,(IY+$38)        ; get the single-character name from STRLEN_lo
          LD    HL,($5B45)        ; get the current line from PPC
          LD    ($5B42),HL        ; and store it in NEWPPC
          LD    A,($5B47)         ; 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.

          RST   18H               ;;;;
;;;       LD    HL,($5B5D)        ; get current character address from CH_ADD
          LD    E,$F3             ; The search will be for the token 'NEXT'

F_LOOP    PUSH  BC                ; save the variable name in B.

          LD    BC,($5B55)        ; fetch NXTLIN

          CALL  LOOK_PROG         ; routine LOOK-PROG searches for 'NEXT' token
                                  ; setting carry flag if end of program reached
                                  ; and updating NEWPPC with line number, BC.

          LD    ($5B55),BC        ; update NXTLIN

          POP   BC                ; retrieve the variable name in B.

          JR    C,REPORT_I        ; forward, if at program end, to REPORT-I
                                  ; '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,F_FOUND         ; forward, if it matches, to F-FOUND
```

```
;   but if no match i.e. nested FOR/NEXT loops then continue search.

          RST   20H             ; NEXT-CHAR
          JR    F_LOOP          ; back to F-LOOP

; ---


F_FOUND   RST   20H             ; NEXT-CHAR
          LD    A,$01           ; subtract the negated counter from 1
          SUB   D               ; to give the statement after the NEXT
          LD    ($5B44),A       ; set system variable NSPPC
          RET                   ; return to STMT-RET to branch to new
                                ; line and statement. ->
; ---

REPORT_I  RST   30H             ; ERROR-1
          DEFB  $11             ; Error Report: FOR without NEXT

; ----------------------------
; THE 'LOOK PROGRAM' SUBROUTINE
; ----------------------------
;   Used to find tokens 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 LD    A,(HL)          ; fetch current character
          CP    $3A             ; is it ':' a statement separator ?
          JR    Z,LOOK_P_2      ; forward, if so, to LOOK-P-2

;   The starting point was PROG-1 or is now the end of a line.

LOOK_P_1  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    ($5B42),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 current address - pointing to BASIC.

          ADD   HL,BC           ; add length to current address.
          LD    B,H             ; and transfer the result - the next line -
          LD    C,L             ; to the BC register.

          POP   HL              ; retrieve the current address.

          LD    D,$00           ; initialize statement counter to zero.

LOOK_P_2  PUSH  BC              ; preserve address of next line

          CALL  EACH_STMT       ; routine EACH-STMT searches current line.
          POP   BC              ; retrieve address of next line.

          RET   NC              ; return if match was found. ->

          JR    LOOK_P_1        ; back, for next line, to LOOK-P-1

; ------------------
; THE 'NEXT' COMMAND
; ------------------
;   e.g. NEXT i
;   The parameter tables have already evaluated the presence of a variable

NEXT      BIT   1,(IY+$37)      ; test FLAGX - handling a new variable ?

          JP    NZ,REPORT_2     ;.jump back, if so, to REPORT-2
                                ; 'Variable not found'

;   now test if the found variable is a simple variable uninitialized by a FOR.

          LD    HL,($5B4D)      ; load address of variable from DEST
          BIT   7,(HL)          ; is it correct type ?
          JR    Z,REPORT_1      ; forward, if not, to REPORT-1
                                ; 'NEXT without FOR'

          INC   HL              ; step past variable name
          LD    ($5B68),HL      ; and set system variable MEM to point to the
                                ; three 5-byte numbers - value, limit, step.

;   Now add the step and put result in the value (mem-0).

          RST   28H             ;; FP-CALC     .
          DEFB  $E0             ;;get-mem-0    v.
          DEFB  $E2             ;;get-mem-2    v,s.
          DEFB  $0F             ;;addition     v+s.
          DEFB  $C0             ;;st-mem-0     v+s.
          DEFB  $02             ;;delete       .
          DEFB  $38             ;;end-calc     .

          CALL  NEXT_LOOP       ; routine NEXT-LOOP tests against limit.

          RET   C               ; return if no more iterations possible.

          LD    HL,($5B68)      ; 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           ; exchange -  HL = line number, D = statement.

          JP    GO_TO_2         ; exit via GO-TO-2 to execute another loop.

; ---

REPORT_1  RST   30H             ; ERROR-1
          DEFB  $00             ; Error Report: NEXT without FOR
```

```
; --------------------------
; THE 'NEXT LOOP' SUBROUTINE
; --------------------------
;   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 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  NEXT_1 - $      ;;to NEXT-1 if step negative

          DEFB  $01             ;;exchange         v,l.

NEXT_1    DEFB  $03             ;;subtract         l-v OR v-l.
          DEFB  $37             ;;greater-0        (1/0)
          DEFB  $00             ;;jump-true        .

          DEFB  NEXT_2 - $      ;;to NEXT-2 if no more iterations.

          DEFB  $38             ;;end-calc         .

          AND   A               ; clear carry flag signaling another loop.

          RET                   ; return

; ---

NEXT_2    DEFB  $38             ;;end-calc         .

          SCF                   ; set carry flag signaling looping exhausted.

          RET                   ; return


; ------------------
; THE '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 then used to read the second list.

READ_3    RST   20H             ; NEXT-CHAR

;   -> Entry point.

READ      CALL  CLASS_01        ; routine CLASS-01 checks variable.

          CALL  SYNTAX_Z        ; routine SYNTAX-Z

          JR    Z,READ_2        ; forward, if checking syntax, to READ-2

;   The runtime path continues.

          RST   18H             ; GET-CHAR fetches character address of variable
                                ; within BASIC to HL.
          LD    ($5B5F),HL      ; save character position in X_PTR.

          LD    HL,($5B57)      ; 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 preceding the line number of the
                                ; last RESTORE command.

          LD    A,(HL)          ; fetch character
          CP    $2C             ; is it a comma ?
          JR    Z,READ_1        ; forward, if so, to READ-1

;   else all data in this statement has been read so look for next DATA token.

          LD    E,$E4           ; prepare token 'DATA'

          CALL  LOOK_PROG       ; routine LOOK-PROG finds the token

          JR    NC,READ_1       ; forward, if 'DATA' found, to READ-1

;   else report the error.

REPORT_E  RST   30H             ; ERROR-1
          DEFB  $0D             ; Error Report: Out of DATA

; ---

READ_1    CALL  TEMP_PTR1       ; routine TEMP-PTR1 advances updating CH_ADD
                                ; with new DATADD position.

          CALL  VAL_FET_1       ; routine VAL-FET-1 assigns value to variable
                                ; checking types match and advancing CH_ADD.

          RST   18H             ; GET-CHAR fetches adjusted character position

          LD    ($5B57),HL      ; store back in DATADD

          LD    HL,($5B5F)      ; fetch original READ statement pointer from
X_PTR

          LD    (IY+$26),$00    ; nullify X_PTR_hi as redundant.

          CALL  TEMP_PTR2       ; routine TEMP-PTR2 restores the READ character
                                ; address to CH_ADD.

READ_2    RST   18H             ; GET-CHAR
          CP    $2C             ; is it ',' indicating more variables to read ?
          JR    Z,READ_3        ; back, if so, to READ-3

          CALL  CHECK_END       ; routine CHECK-END checks that nothing
                                ; follows and returns if checking syntax  >>

          RET                   ; return from here in runtime to STMT-RET.

; ------------------
; THE 'DATA' COMMAND
; ------------------
;   e.g. DATA 1, 2, "text", score-1, a$(location, room, object), FN r(49),
;        wages - tax, TRUE, The meaning of life
;   In runtime this 'command' is passed by but the syntax is checked when such
;   a statement is found while parsing a line.

DATA      CALL  SYNTAX_Z        ; routine SYNTAX-Z to check status
          JR    NZ,DATA_2       ; forward, if in runtime, to DATA-2
```

```
;   The syntax path continues.

DATA_1     CALL  SCANNING        ; routine SCANNING to check syntax of expression

           CP    $2C             ; is following character a comma ?

           CALL  NZ,CHECK_END    ; if not, routine CHECK-END checks that
                                 ; statement is complete. Will make an early
                                 ; exit if it is. >>>

           RST   20H             ; NEXT-CHAR advances past comma.

           JR    DATA_1          ; loop back to DATA-1

; ---

DATA_2     LD    A,$E4           ; in runtime, set token to 'DATA' and continue
                                 ; into the PASS-BY routine.


; ------------------------
; THE 'PASS BY' SUBROUTINE
; ------------------------
;   This routine is used to backtrack to a command token and then forward to
;   the next statement in runtime.
;   The A register contains the required token - either $E4 (DATA) from above,
;   or $CE (DEF FN) when called.

PASS_BY    LD    B,A             ; Give BC enough space to find the token.

           CPDR                  ; Compare decrement and repeat. (Only use).
                                 ; Work backwards until keyword is found which
                                 ; is the 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    EACH_STMT       ; to EACH-STMT to find next statement

; ---------------------
; THE '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    CALL  FIND_LINE       ;+ routine FIND-INT2 puts integer in BC.
                                 ;+ Note. B is now checked against limit $3F
                                 ;+ and an error generated if higher.

;   this entry point is used from RUN command with BC holding zero

REST_RUN   LD    H,B             ; transfer the line
           LD    L,C             ; number to the HL register.

           CALL  LINE_ADDR       ; routine LINE-ADDR to fetch the address.
```

```
          DEC   HL              ; point to the location before the line.
          LD    ($5B57),HL      ; update the dynamic system variable DATADD.

          RET                   ; return to STMT-RET (or RUN)

; ----------------------
; THE '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 CALL  FIND_INT2       ; routine FIND-INT2 puts parameter in BC.

          LD    A,B             ; test this
          OR    C               ; for zero.
          JR    NZ,RAND_1       ; forward to RAND-1 if not zero.

          LD    BC,($5B78)      ; use the lower two bytes at FRAMES1.

RAND_1    LD    ($5B76),BC      ; place in SEED system variable.

          RET                   ; return to STMT-RET

; ---------------------
; THE '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  LD    HL,($5B6E)      ; fetch OLDPPC line number.
          LD    D,(IY+$36)      ; fetch OSPPC statement.

          JR    GO_TO_2         ; forward to GO-TO-2

; ------------------
; THE '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     CALL  FIND_LINE       ;+ 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,REPORT_Bb    ; forward, if higher, to REPORT-B

;   This call entry point is used to update the system variables e.g. by RETURN.

GO_TO_2   LD    ($5B42),HL      ; save line number in NEWPPC
          LD    (IY+$0A),D      ; and statement in NSPPC
          RET                   ; to STMT-RET (or GO-SUB command)

; -----------------
; THE 'OUT' COMMAND
; -----------------
;   Syntax has been entirely checked and the two comma-separated values are on
;   the calculator stack.

OUT       CALL  TWO_PARAM       ; routine TWO-PARAM fetches values to BC and A.

          OUT   (C),A           ; perform the operation.

          RET                   ; return to STMT-RET.

; ------------------
; THE '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 the system variables.

POKE      CALL  TWO_PARAM       ; routine TWO-PARAM fetches values to BC and A.

          LD    (BC),A          ; load memory location with A.

          RET                   ; return to STMT-RET.

; ------------------------------------
; THE 'FETCH TWO PARAMETERS' SUBROUTINE
; ------------------------------------
;   This routine fetches a byte and word from the calculator stack producing an
;   error if either is out of range.

TWO_PARAM CALL  FP_TO_A         ; routine FP-TO-A
          JR    C,REPORT_Bb     ; forward, with 8-bit overflow, to REPORT-B
                                ; 'Integer out of range'

          JR    Z,TWO_P_1       ; skip forward, if positive, to TWO-P-1

          NEG                   ; negative numbers are made positive.

TWO_P_1   PUSH  AF              ; save the byte value

          CALL  FIND_INT2       ; routine FIND-INT2 gets 16-bit integer to BC

          POP   AF              ; restore the byte value

          RET                   ; return

; ---------------------------
; THE 'FIND INTEGERS' ROUTINES
; ---------------------------
;   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 CALL  FP_TO_A         ; routine FP-TO-A
```

```
          JR    FIND_I_1        ; forward to common exit routine at FIND-I-1

; ---

;   ->

FIND_INT2 CALL  FP_TO_BC        ; routine FP-TO-BC

;   The common exit routine checks that numbers are positive and do not overflow

FIND_I_1  JR    C,REPORT_Bb     ; skip forward, with overflow, to REPORT-Bb

          RET   Z               ; return if BC (or A) is positive.


REPORT_Bb RST   30H             ; ERROR-1
          DEFB  $0A             ; Error Report: Integer out of range

; ------------------------------
; THE NEW 'FIND LINE' SUBROUTINE
; ------------------------------
;+  This new routine is used in place of FIND_INT2 to validate the line numbers
;+  that it fetches.

FIND_LINE CALL  FIND_INT2       ;+ Routine gets 16 bit integer in BC.

          LD    H,B             ;
          LD    L,C             ;

          LD    A,B             ;+ Fetch high byte.

          CP    $40             ;+ Compare with the system limit.

          JR    NC,REPORT_Bb    ;+ Back, if higher, than 16383 to ERROR_Bb
                                ;+ 'Integer out of range'

          LD    D,$00           ;+ Useful return value.

          RET                   ;+ Return.

; ----------------------------
; THE NEW 'CLEAR HASH' ROUTINE
; ----------------------------
;   This routine responds to the command 'CLEAR #' by closing the sixteen
;   streams in turn.  Any pending printer output is flushed but network output
;   is discarded.  A hash has been found and it remains to check that nothing
;   follows.

CLR_HASH  RST   20H             ;+ NEXT_CHAR
          CALL  CHECK_END       ;+ CHECK_END quits if checking syntax >>

;   The runtime path.

          LD    A,16            ;+ Set stream to sixteen

NMI_STRMS SET   6,(IY+$3B)      ;+ Set T_ADDR_hi to indicate no Network EOF.


ALL_STRMS DEC   A               ;+ pre-decrement
          PUSH  AF              ;+ save stream and result flag.

          CALL  STR_DATA1       ;+ get the offset

          CALL  NZ,CLOSE_OK     ;+ CLOSE the stream if it's open.
          POP   AF              ;+

          JR    NZ,ALL_STRMS    ;+ do all sixteen

          RET                   ;+ Return.

; -----------------
; THE 'RUN' COMMAND
; -----------------
;   This command runs a program starting at an optional line.
;   It performs a 'RESTORE 0' then CLEAR

RUN       CALL  GO_TO           ; routine GO-TO puts line number in
                                ; system variables.

;;;       LD    BC,$0000        ; prepare to set DATADD to first line.
          LD    B,D             ;+
          LD    C,D             ;+

          CALL  REST_RUN        ; routine REST-RUN does the 'restore'.
                                ; Note. BC still holds zero.

          JR    CLEAR_RUN       ; forward to CLEAR-RUN to clear variables
                                ; without disturbing RAMTOP and
                                ; exit indirectly to STMT-RET

; -------------------
; THE '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.
;   Notice also that if an error occurs then the GOSUB stack is not cleared.

CLEAR     RST   18H             ; GET_CHAR
          CP    $23             ; is character a '#' ?
          JR    Z,CLR_HASH      ; back if so.

          CALL  FETCH_NUM       ;+ routine FETCH_NUM checks for numeric
                                ;+ expression and stacks in run-time defaulting
                                ;+ to zero.
          CALL  CHECK_END       ; routine CHECK-END quits if syntax path.

          CALL  FIND_INT2       ; routine FIND-INT2 fetches address to BC.

CLEAR_RUN LD    A,B             ; test for
          OR    C               ; zero.
          JR    NZ,CLEAR_1      ; skip, if not zero, to CLEAR-1

          LD    BC,($5BB2)      ; use the existing value of RAMTOP if zero.

CLEAR_1   PUSH  BC              ; save RAMTOP value.

          LD    DE,($5B4B)      ; fetch VARS

;;;       LD    HL,($5B59)      ; fetch E_LINE
;;;       DEC   HL              ; adjust to point at variables end-marker.

          CALL  L_EL_DHL        ;+ NEW routine with above code.
          CALL  RECLAIM_1       ; routine RECLAIM-1 reclaims the space used by
                                ; the variables, setting BC to zero.

;   Note. A call to REST_RUN here would execute a RESTORE as per BASIC manual
;   but it is difficult to decide if CLEAR should execute a RESTORE. Vickers
;   merely points out that the ROM doesn't.

          CALL  CLS             ; routine CLS to clear screen.

          LD    HL,($5B65)      ; fetch STKEND the start of free memory.
;;;       LD    DE,$0032        ; allow for another 50 bytes.
          LD    E,$32           ; 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,REPORT_M     ; forward to REPORT-M
                                ; 'RAMTOP no good'

          LD    HL,($5BB4)      ; 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,CLEAR_2      ; skipa, if in actual RAM, to CLEAR-2

REPORT_M  RST   30H             ; ERROR-1
          DEFB  $15             ; Error Report: RAMTOP no good

;   Now, even if RAMTOP has not moved, the GOSUB stack is cleared and
;   initialized.

CLEAR_2   EX    DE,HL           ; transfer RAMTOP value to HL.
          LD    ($5BB2),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    ($5B3D),SP      ; make ERR_SP point to location.
          EX    DE,HL           ; put STMT-RET in HL.

          JP    (HL)            ; and go there directly.


; --------------------
; THE '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
;   plus 1 are placed on the GO SUB stack as a RETURN point.

GO_SUB    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,($5B45)      ; 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    ($5B3D),SP      ; make system variable ERR_SP point to it.
          PUSH  DE              ; push the address STMT-RET.

          CALL  GO_TO           ; 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.

; -------------------------
; THE 'TEST ROOM' SUBROUTINE
; -------------------------
;   This routine is used on many occasions when extending a dynamic area
;   upwards or the GO SUB stack downwards.

TEST_ROOM LD    HL,($5B65)      ; fetch STKEND
          ADD   HL,BC           ; add the supplied test value
          JR    C,REPORT_4      ; forward, if over $FFFF, to REPORT-4
                                ; 'Out of memory'

          EX    DE,HL           ; The result was less so transfer to DE
          LD    HL,$0050        ; test against another 80 bytes
          ADD   HL,DE           ; anyway
          JR    C,REPORT_4      ; forward, if this passes $FFFF, to REPORT-4
                                ; 'Out of memory'

          SBC   HL,SP           ; if less than the machine stack pointer
          RET   C               ; then return - OK.
                                ; Register HL contains the negated number of
                                ; free bytes.

REPORT_4  LD    L,$03           ; prepare 'Out of memory'

          JP    ERROR_3         ; jump back to ERROR-3
                                ; 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 address.
;   Note. It has been moved, for stability, to location ninety three decimal.


; --------------------
; 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 GO SUB stack.  Both grow
;   downwards, the machine stack by two bytes, the GO SUB stack by 3 bytes.
;   The highest location is a statement byte followed by a two-byte line number.

RETURN    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,REPORT_7      ; forward, with a match, to REPORT-7
                                ; 'RETURN without GO SUB'

;   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    ($5B3D),SP      ; adjust ERR_SP to point to new stack pointer

          PUSH  BC              ; now re-stack the address STMT-RET

          JP    GO_TO_2         ; back to GO-TO-2
                                ; to update statement and line system variables
                                ; and exit indirectly to the address just pushed
                                ; on the stack.

; ---

REPORT_7  PUSH  DE              ; replace the end-marker.
          PUSH  HL              ; now restore the error address
                                ; as will be required in a few clock cycles.

          RST   30H             ; ERROR-1
          DEFB  $06             ; Error Report: RETURN without GOSUB

;   Note. 'GO SUB' won't fit in message.

; -------------------
; THE 'PAUSE' COMMAND
; -------------------
;   The PAUSE command takes as its parameter the number of interrupts
;   for which to wait. PAUSE 50 pauses for about a second in the UK.
;   PAUSE 60 waits for the same time in the USA.
;   PAUSE 0 pauses indefinitely.
;   Both forms can be finished early by pressing a key.

PAUSE     CALL  FIND_INT2       ; routine FIND-INT2 puts value in BC

PAUSE_1   HALT                  ; wait for an interrupt.
          DEC   BC              ; decrease the counter.
          LD    A,B             ; test if the
          OR    C               ; result is zero.
          JR    Z,PAUSE_END     ; forward, if so, to PAUSE-END

          LD    A,B             ; test if
          AND   C               ; now $FFFF
          INC   A               ; that is, initially zero.
          JR    NZ,PAUSE_2      ; skip forward, if not, to PAUSE-2

          INC   BC              ; restore counter to zero.

PAUSE_2   BIT   5,(IY+$01)      ; test FLAGS - has a new key been pressed ?
          JR    Z,PAUSE_1       ; back, if not, to PAUSE-1

PAUSE_END RES   5,(IY+$01)      ; update FLAGS - signal no new key

          RET                   ; Return.

; --------------------------------
; THE 'CHECK FOR BREAK' SUBROUTINE
; --------------------------------
;   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 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.

; --------------------
; THE '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    CALL  SYNTAX_Z        ; routine SYNTAX-Z

          JR    Z,DEF_FN_1      ; forward, if parsing, to DEF-FN-1

          LD    A,$CE           ; else in runtime load A with 'DEF FN' and
          JP    PASS_BY         ; jump back to PASS-BY

; ---

;   The syntax path continues here.

DEF_FN_1  SET    6,(IY+$01)     ; set FLAGS - assume numeric result

          CALL  ALPHA           ; call routine ALPHA

          JR    NC,REPORT_Cd    ; forward, if not, to DEF-FN-4
                                ; 'Nonsense in BASIC'


          RST   20H             ; NEXT-CHAR
          CP    $24             ; is character '$' ?
          JR    NZ,DEF_FN_2     ; forward, if not type string, to DEF-FN-2

;;;       RES   6,(IY+$01)      ; set FLAGS - signal string result.
          CALL  STR_RSLT        ;+

          RST   20H             ; get NEXT-CHAR

DEF_FN_2  CP    $28             ; is character '(' ?
          JR    NZ,REPORT_Cd    ; forward, if not, to DEF-FN-7
                                ; 'Nonsense in BASIC'


          RST   20H             ; NEXT-CHAR
          CP    $29             ; is character ')' ?
          JR    Z,DEF_FN_6      ; forward, if null arguments, to DEF-FN-6

DEF_FN_3  CALL  ALPHA           ; routine ALPHA checks that it is the expected
                                ; alphabetic character.

DEF_FN_4  JR    NC,REPORT_Cd    ; jump, if not, to REPORT-C
                                ; 'Nonsense in BASIC'.

          EX    DE,HL           ; save pointer in DE

          RST   20H             ; NEXT-CHAR re-initializes HL from CH_ADD
                                ; and advances.
          CP    $24             ; is character a '$' ?
          JR    NZ,DEF_FN_5     ; forward, if not string argument, to DEF-FN-5

          EX    DE,HL           ; save pointer to '$' in DE

          RST   20H             ; NEXT-CHAR re-initializes HL and advances

DEF_FN_5  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  MAKE_ROOM       ; 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,DEF_FN_6     ; forward, if not, to DEF-FN-6


          RST   20H             ; else NEXT-CHAR
          JR    DEF_FN_3        ; and back to DEF-FN-3

; ---

DEF_FN_6
;;;       CP    $29             ; is character the closing ')' ?
;;;       JR    NZ,REPORT_Cd    ; forward, if not, to DEF-FN-7
;;;       RST   20H             ; get NEXT-CHAR

          CALL  RBRKT_NXT       ;+ check for right-hand bracket and advances.


          CP    $3D             ; is it '=' ?
          JR    NZ,REPORT_Cd    ; to DEF-FN-7
                                ; 'Nonsense in BASIC'


          RST   20H             ; address NEXT-CHAR
          LD    A,($5B3B)       ; get FLAGS which has been set above

          PUSH  AF              ; and preserve

          CALL  SCANNING        ; 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  JP  NZ,REPORT_C   ; jump back to REPORT-C if the expected result
                                ; is not the same type.
                                ; 'Nonsense in BASIC'

;;;       CALL  CHECK_END     ; routine CHECK-END will return early

          CALL  Z,CHECK_END     ; 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.

REPORT_Cd RST   30H             ;+ ERROR-1
          DEFB  $0B             ;+ Error Report: Nonsense in BASIC


; --------------------------
; THE 'UNSTACK-Z' SUBROUTINE (6)
; --------------------------
;   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 CALL  SYNTAX_Z        ; 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.


; --------------------
; THE '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    LD    A,$03           ; the printer channel
          JR    PRINT_1         ; forward to PRINT-1

; -------------------
; THE 'PRINT' COMMAND
; -------------------
;   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     LD    A,$02           ; the stream for the upper screen.

;   The LPRINT command joins here.

PRINT_1   CALL  CHN_O_SYN       ;+ routine opens channel in runtime.

;;;       CALL  SYNTAX_Z        ; routine SYNTAX-Z checks if program running
;;;       CALL  NZ,CHAN_SLCT    ; routine CHAN-OPEN if so (calls TEMPS)
;;;       CALL  TEMPs           ; routine TEMPs sets temporary colours.

          CALL  PRINT_2         ; routine PRINT-2 - the actual item

          CALL  CHECK_END       ; routine CHECK-END gives error if not at end
                                ; of statement
          RET                   ; and return in runtime >>>

; ------------------------
; THE 'PRINT 2' SUBROUTINE
; ------------------------
;   This subroutine is called from above and also from INPUT.

PRINT_2   RST   18H             ; GET-CHAR gets printable character
          CALL  PR_END_Z        ; routine PR-END-Z checks if more printing

          JR    Z,PRINT_4       ; 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   CALL  PR_POSN_1       ; routine PR-POSN-1 returns zero if more
                                ; but returns early at this point if
                                ; at end of statement!
                                ;
          JR    Z,PRINT_3       ; to PRINT-3 if consecutive positioners

          CALL  PR_ITEM_1       ; routine PR-ITEM-1 deals with strings etc.
          CALL  PR_POSN_1       ; routine PR-POSN-1 for more position codes
          JR    Z,PRINT_3       ; loop back, if so, to PRINT-3

PRINT_4   CP    $29             ; return now if this is ')' from input-item.
                                ; (see INPUT.)
          RET   Z               ; or continue and print carriage return in
                                ; runtime

; -----------------------------------
; THE 'PRINT CARRIAGE RETURN' ROUTINE
; -----------------------------------
;   This routine which continues from above prints a carriage return
;   in run-time only. It is also called once from PRINT-POSN.

PRINT_CR  CALL  UNSTACK_Z       ; routine UNSTACK_Z quits if checking syntax.

          LD    A,$0D           ; prepare a carriage return

          RST   10H             ; PRINT-A outputs to current channel.

;;;       RET                   ; return.

          JP    CR_END          ;+ NEW test for network before returning.



; ----------------------------
; THE 'PRINT ITEMS' SUBROUTINE
; ----------------------------
;   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 RST   18H             ; GET-CHAR
          CP    $AC             ; is character 'AT' ?
          JR    NZ,PR_ITEM_2    ; forward, if not, to PR-ITEM-2

          CALL  NEXT_2NUM       ; routine NEXT-2NUM  check for two comma
                                ; separated numbers placing them on the
                                ; calculator stack in runtime.
          CALL  UNSTACK_Z       ; routine UNSTACK_Z quits if checking syntax.

          CALL  STK_TO_BC       ; routine STK-TO-BC get the numbers in B and C.
          LD    A,$16           ; prepare the 'at' control.
          JR    PR_AT_TAB       ; forward to PR-AT-TAB to print the sequence.

; ---

PR_ITEM_2 CP    $AD             ; is character 'TAB' ?
          JR    NZ,PR_ITEM_3    ; forward, if not, to PR-ITEM-3


          RST   20H             ; NEXT-CHAR to address next character
          CALL  EXPT_1NUM       ; routine EXPT-1NUM checks for numeric
                                ; expression and stacks it in run-time.

          CALL  UNSTACK_Z       ; routine UNSTACK_Z quits if checking syntax.

          CALL  FIND_INT2       ; routine FIND-INT2 puts integer in BC.
          LD    A,$17           ; prepare the 'tab' control.

PR_AT_TAB 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 CALL  CO_TEMP_3       ; routine CO-TEMP-3 will print any colour
          RET   NC              ; items - return if success.

;   Now consider a change in the output stream.
;   Note. as this is called from IN_ITEM it can also effect a change in the
;   stream used for INPUT.

          CALL  STR_ALTER       ; routine STR-ALTER considers new stream
          RET   NC              ; return if altered.

          CALL  SCANNING        ; routine SCANNING now to evaluate expression

          CALL  UNSTACK_Z       ; routine UNSTACK_Z quits if not runtime.

          BIT   6,(IY+$01)      ; test FLAGS  - Numeric or string result ?

;   Note. the next two instructions have been switched so that STK_FETCH
;   can return zero if BC is zero (used elsewhere).

          JP    NZ,PRINT_FP     ; to PRINT-FP to print if numeric >>>

          CALL  STK_FETCH       ; routine STK-FETCH if string.
                                ; note flags now affected.


;   It was a string expression - start in DE, length in BC
;   Now enter a loop to print it
```

```
PR_STRING 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    PR_STRING        ; loop back to PR-STRING.

; --------------------------------
; THE 'END OF PRINTING' SUBROUTINE
; --------------------------------
;   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  CP    $29              ; is character a ')' ?
          RET   Z                ; return if so -        e.g. INPUT (p$); a$

PR_ST_END 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 with match.
                                 ;                       e.g. PRINT a :

; ----------------------------
; THE 'PRINT POSITION' ROUTINE
; ----------------------------
;   This routine considers a single positional character ';', ',', '''

PR_POSN_1 RST   18H              ; GET-CHAR
          CP    $3B              ; is it ';' ?
                                 ; i.e. print from last position.
          JR    Z,PR_POSN_3      ; forward, if so, to PR-POSN-3
                                 ; i.e. do nothing.

          CP    $2C              ; is it ',' ?
                                 ; i.e. print at next tabstop.
          JR    NZ,PR_POSN_2     ; forward to PR-POSN-2 if anything else.

          CALL  SYNTAX_Z         ; routine SYNTAX-Z

          JR    Z,PR_POSN_3      ; 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    PR_POSN_3        ; skip to PR-POSN-3.

; ---

;   check for newline.

PR_POSN_2 CP    $27              ; is character a "'" ? (newline)
          RET   NZ               ; return if no match              >>>

          CALL  PRINT_CR         ; routine PRINT-CR outputs a carriage return
```

```
                                ; in runtime only.

PR_POSN_3 RST   20H             ; NEXT-CHAR to A.
          CALL  PR_END_Z        ; routine PR-END-Z checks if at end.
          JR    NZ,PR_POSN_4    ; skip forward, if not, to PR-POSN-4

          POP   BC              ; drop return address if at end.

PR_POSN_4 CP    A               ; reset the zero flag.
          RET                   ; and return to loop or quit.

; ------------------------------
; THE 'ALTER STREAM' SUBROUTINE
; ------------------------------
;   This routine is called from PRINT ITEMS above, and also LIST as in LIST #15

STR_ALTER RST   18H             ;+ GET_CHAR
          CP    $23             ; is character '#' ?
          SCF                   ; set carry flag.
          RET   NZ              ; return if no match.


          RST   20H             ; NEXT-CHAR
          CALL  EXPT_1NUM       ; routine EXPT-1NUM gets stream number
          AND   A               ; prepare to exit early with carry reset

          CALL  UNSTACK_Z       ; routine UNSTACK_Z exits early if parsing

CHAN_CHK  CALL  FIND_INT1       ; routine FIND-INT1 gets number off stack

          CP    $10             ; stream must be range 0 - 15 decimal.
          JP    NC,REPORT_O     ; jump back, if not, to REPORT-O
                                ; 'Invalid stream'.

          CALL  CHAN_SLCT       ; Routine CHAN-OPEN

          AND   A               ; Clear carry - signal item dealt with.

          RET                   ; Return.

; -------------------
; THE 'INPUT' COMMAND
; -------------------
;   This command inputs by default from the stream 1.  On the standard
;   Spectrum this is selected before CLS-LOWER so the channel that is
;   in force is the system 'K' channel and can only be overridden by the user
;   using INPUT #1.

INPUT     CALL  SYNTAX_Z        ; routine SYNTAX-Z to check if in runtime.
          JR    Z,INPUT_1       ; forward, if checking syntax, to INPUT-1

          LD    A,$01           ; select stream 1 which is reserved for INPUT.
          CALL  CHAN_SLCT       ; routine CHAN-OPEN opens the channel.

          CALL  IN_CHAN_K       ;+ routine IN-CHAN-K tests if keyboard in use.

;;;       CALL  CLS_LOWER       ; routine CLS-LOWER wrongly clears lower screen.

          CALL  Z,CLS_LOWER     ;+ routine CLS-LOWER clears the lower screen
                                ;+ and sets DF_SZ to two and TV_FLAG to $01
                                ;+ but only if channel 1 is the keyboard.

INPUT_1   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, 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  IN_ITEM_1       ; routine IN-ITEM-1 to handle the input.

          CALL  CHECK_END       ; 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,($5B88)      ; fetch S_POSN current line/column of
                                ; the upper screen.
          LD    A,($5B6B)       ; fetch DF_SZ the display file size of
                                ; the lower screen.
          CP    B               ; test that lower screen does not overlap.
          JR    C,INPUT_2       ; forward, if not, to INPUT-2

;   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   LD    ($5B88),BC      ; set S_POSN update upper screen line/column.
          LD    A,$19           ; subtract from twenty five
          SUB   B               ; the new line number.
          LD    ($5B8C),A       ; and place result in SCR_CT - scroll count.
          RES   0,(IY+$02)      ; update TV_FLAG - signal main screen in use.
          CALL  CL_SET          ; routine CL-SET sets the print position
                                ; system variables for the upper screen.
          JP    CLS_LOWER       ; jump back to CLS-LOWER and make
                                ; an indirect exit >>.

; ---------------------------
; THE 'INPUT ITEM' SUBROUTINE
; ---------------------------
;   This subroutine deals with the input items and print items from the current
;   input channel which was defaulted to 'K' above.

IN_ITEM_1 CALL  PR_POSN_1       ; routine PR-POSN-1 deals with a single
                                ; position item at each call.
          JR    Z,IN_ITEM_1     ; back to IN-ITEM-1 until no more in a
                                ; sequence.

          CP    $28             ; is character '(' ?
          JR    NZ,IN_ITEM_2    ; forward, if not, to IN-ITEM-2

;   any variables within brackets will be treated as part, or all, of the
;   prompt instead of being used as destination variables.

          RST   20H             ; NEXT-CHAR
          CALL  PRINT_2         ; routine PRINT-2 to output the dynamic
                                ; prompt.

          RST   18H             ; GET-CHAR

;;;       CP    $29             ; is character a matching ')' ?
;;;       JR    NZ,REPORT_Cy    ; forward, if not, to REPORT-Cy
;;;       RST   20H             ; NEXT-CHAR

          CALL  RBRKT_NXT       ;+ check for right-hand bracket and advance.

          JP    IN_NEXT_2       ; jump forward to IN-NEXT-2

; ---

;   Consider INPUT LINE

IN_ITEM_2 CP    $CA             ; is the character the token 'LINE' ?
          JR    NZ,IN_ITEM_3    ; forward, if not, to IN-ITEM-3

          RST   20H             ; NEXT-CHAR - variable must come next.
          CALL  CLASS_01        ; 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,REPORT_C     ; jump back to REPORT-C if not string
;;;                             ; 'Nonsense in BASIC'.

          JR    Z,IN_PROMPT     ; forward, if string, to IN-PROMPT
                                ; to set up workspace.

REPORT_Cy RST   30H             ;+ ERROR-1
          DEFB  $0B             ;+ Error Report: Nonsense in BASIC

; ---

;   the jump was here for other variables.
;   Note. the character '#' will cause a jump to IN_NEXT_1

IN_ITEM_3 CALL   ALPHA          ; routine ALPHA checks if character is
                                ; a suitable variable name.

          JP    NC,IN_NEXT_1    ; jump forward, if not, to IN-NEXT-1

          CALL  CLASS_01        ; routine CLASS-01 returns destination
                                ; address of variable to be assigned.
          RES   7,(IY+$37)      ; update FLAGX  - signal not INPUT LINE.

;   The two paths converge here.

IN_PROMPT CALL  SYNTAX_Z        ; routine SYNTAX-Z

          JP    Z,IN_NEXT_2     ; forward to IN-NEXT-2 if checking syntax.

;   Continue in runtime.

          CALL  SET_WORK        ; routine SET-WORK clears workspace.

          LD    HL,$5B71        ; 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 CR.

          LD    C,$01           ;+ initialize space required to one for the CR.
          BIT   7,(HL)          ; test FLAGX - INPUT LINE in use ?

          JR    NZ,IN_PR_2      ; forward, if so, to IN-PR-2
                                ; as that is all the space that is required.

;   If not INPUT LINE then the result can be numeric or string.

          LD    A,($5B3B)       ; 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,IN_PR_1      ; forward, if so, to IN-PR-1

          LD    C,$03           ; increase space to three bytes for the
                                ; pair of surrounding quotes.

IN_PR_1   OR    (HL)            ; if numeric result, set bit 6 of FLAGX.
          LD    (HL),A          ; and update system variable

IN_PR_2   CALL  BC_SPACE0       ; 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,IN_PR_3      ; forward, if not, to IN-PR-3

          BIT   1,C             ;+ test if quotes required.
          JR    Z,IN_PR_3       ;+ skip forward, if not, to IN_PR_3

          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   LD    ($5B5B),HL      ; set keyboard cursor K_CUR to HL

          BIT   7,(IY+$37)      ; test FLAGX - is this INPUT LINE ?
          JR    NZ,IN_VAR_3     ; forward, if so, to IN-VAR-3 as input will
                                ; be accepted without checking its syntax.

;   prepare to parse the numeric or string input if not INPUT LINE.

          RST   18H             ;+
;;;       LD    HL,($5B5D)      ; fetch CH_ADD
          PUSH  HL              ; and save on stack.
          LD    HL,($5B3D)      ; fetch ERR_SP
          PUSH  HL              ; and save on stack

IN_VAR_1  LD    HL,IN_VAR_1     ; 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,IN_VAR_2      ; forward, if not keyboard, to IN-VAR-2

;   Now update the error pointer so that user is able to alter until correct.

          LD    ($5B3D),SP      ; set ERR_SP to point to IN-VAR-1 on stack.

IN_VAR_2  LD    HL,($5B61)      ; set HL to WORKSP - start of workspace.

          CALL  REMOVE_FP       ; routine REMOVE-FP removes floating point
                                ; forms when looping in the 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  SET_ER_FF       ;+ NEW 3-byte call.

          CALL  EDITOR          ; 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  IN_ASSIGN       ; 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    IN_VAR_4        ; jump forward to IN-VAR-4

; ---

;   The jump was to here when using INPUT LINE.

IN_VAR_3  CALL  EDITOR          ; routine EDITOR is called for input

;   When ENTER received rejoin other route but with no syntax check.

;   Paths for INPUT and INPUT LINE converge here.


IN_VAR_4
;;;       LD    (IY+$22),$00    ; set K_CUR_hi to a low value
          LD    (IY+$22),A      ;+ set K_CUR_hi to a low value so that cursor
                                ;+ no longer appears in the input line. (A=13)

          CALL  IN_CHAN_K       ; routine IN-CHAN-K tests if keyboard in use.

          JR    NZ,IN_VAR_5     ; forward to IN-VAR-5 if using another input
                                ; channel.

;   continue here if using the keyboard.

          CALL  ED_COPY         ; 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,($5B82)      ; fetch line and column from ECHO_E

          CALL  CL_SET          ; routine CL-SET sets S-POSNL to those
                                ; values.

;   if using another input channel rejoin here.

IN_VAR_5  LD    HL,$5B71        ; 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,IN_VAR_6     ; forward to IN-VAR-6 if INPUT LINE.

          POP   HL              ; drop the looping address
          POP   HL              ; drop the address of previous
                                ; error handler.
          LD    ($5B3D),HL      ; set ERR_SP to point to it.
          POP   HL              ; drop original CH_ADD which points to
                                ; INPUT command in BASIC line.
          LD    ($5B5F),HL      ; save in X_PTR while input is assigned.
          SET   7,(IY+$01)      ; update FLAGS - Signal running program
          CALL  IN_ASSIGN       ; 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,($5B5F)      ; fetch stored CH_ADD value from X_PTR.
;;;       LD    (IY+$26),$00    ; set X_PTR_hi so that it is no longer relevant.
          LD    (IY+$26),A      ;+ set X_PTR_hi so that it is irrelevant. (A=13)
          LD    ($5B5D),HL      ; put restored value back in CH_ADD
          JR    IN_NEXT_2       ; 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  LD    HL,($5B63)      ; STKBOT points to the end of the input.
          LD    DE,($5B61)      ; WORKSP points to the beginning.
          SCF                   ; prepare for true subtraction.
          SBC   HL,DE           ; subtract to get length
          LD    B,H             ; transfer it to
          LD    C,L             ; the BC register pair.
          CALL  STK_STO_s       ; routine STK-STO-$ stores parameters on
                                ; the calculator stack.

          CALL  LET             ; routine LET assigns it to destination.

          JR    IN_NEXT_2       ; forward to IN-NEXT-2 as print items
                                ; not allowed with INPUT LINE.
                                ; Note. that "hat" + "stand" will, for
                                ; example, be unchanged.

; ---

;   The jump was to here when ALPHA found more items while looking for
;   a variable name.  The routine PR_ITEM_1 is called for the first time
;   which allows the stream to be altered if the character is '#'.

IN_NEXT_1 CALL  PR_ITEM_1       ; routine PR-ITEM-1 considers further items.

IN_NEXT_2 CALL  PR_POSN_1       ; routine PR-POSN-1 handles a position item.

          JP    Z,IN_ITEM_1     ; 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 LD    HL,($5B61)      ; fetch WORKSP start of input
          LD    ($5B5D),HL      ; set CH_ADD to first character

          RST   18H             ; GET-CHAR ignoring any leading white-space.
          CP    $E2             ; is it 'STOP'
          JR    Z,IN_STOP       ; forward, if so, to IN-STOP

          LD    A,($5B71)       ; load accumulator from FLAGX

          CALL  VAL_FET_2       ; routine VAL-FET-2 makes assignment
                                ; or goes through the motions if checking
                                ; syntax. SCANNING is used.

          RST   18H             ; GET-CHAR
          CP    $0D             ; is character a carriage return ?
          RET   Z               ; return with a match.
                                ; 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 RST   30H             ; ERROR-1
          DEFB  $0B             ; Error Report: Nonsense in BASIC
; ---

;;; IN_STOP   CALL  SYNTAX_Z    ; routine SYNTAX-Z (UNSTACK_Z?)
;;;           RET   Z           ; return if checking syntax

IN_STOP   CALL  UNSTACK_Z       ;+ return if checking syntax.
                                ;+ as user wouldn't see error report.
                                ;+ but generate visible error report
                                ;+ on second invocation.

REPORT_H  RST   30H             ; ERROR-1
          DEFB  $10             ; Error Report: STOP in INPUT

; ----------------------------------
; THE 'TEST FOR CHANNEL K' SUBROUTINE
; ----------------------------------
;   This subroutine is called once from the INPUT command to check if
;   the input routine in use is the one for the keyboard.
;   It returns with the zero flag set for the keyboard and reset for the
;   network and RS232.
;   Note. this routine, essentially the same as set out here, has been moved
;   to a position before the NUMBER routine with which it is now combined.

;;; IN_CHAN_K LD    HL,($5B51)      ; 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.

; -------------------------
; THE '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 were 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 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 CALL  CO_TEMP_3       ; routine CO-TEMP-3 to output colour control.
          RET   C               ; return if nothing more to output. ->

          CALL  CLASS_0C        ;+ New routine to check for ';' or ',' and
                                ;+ advance CH_ADD if so else produce error.

          JR    CO_TEMP_2       ;+ back, if no error to CO_TEMP_2

;;;       RST   18H             ; GET-CHAR
;;;       CP    $2C             ; is it ',' separator ?
;;;       JR    Z,CO_TEMP_1     ; back, if so, to CO-TEMP-1
;;;       CP    $3B             ; is it ';' separator ?
;;;       JR    Z,CO_TEMP_1     ; back, if so, to CO-TEMP-1
;;;       JR    REPORT_Cb       ; 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 CP    $D9             ; compare addressed character to 'INK'
          RET   C               ; return if less.

          CP    $DF             ; compare with 'OUT'
          CCF                   ; Complement Carry Flag
          RET   C               ; return if greater than 'OVER' ($DE).

;   The token expects one parameter so advance CH_ADD

          PUSH  AF              ; save the colour token e.g. 'PAPER'.
          RST   20H             ; NEXT-CHAR advances address.

          POP   AF              ; restore token and continue.

;   -> This entry point used by CLASS-07. e.g. the command PAPER 6.

CO_TEMP_4 SUB   $C9             ; reduce to control character $10 (INK)
                                ; through $15 (OVER) and clears CARRY flag.

          PUSH  AF              ; save control and carry flag.

          CALL  EXPT_1NUM       ; routine EXPT-1NUM stacks addressed parameter
                                ; on the calculator stack.

          POP   AF              ; restore control and clear carry flag.

;;;       AND   A               ; clear carry for success (already clear).

          CALL  UNSTACK_Z       ; routine UNSTACK_Z returns if checking syntax.

;   In runtime, output the two control characters.  There is no need to
;   assimilate the two codes first.

;;;       PUSH  AF              ; save again

          RST   10H             ;+ outputs the control altering output address.

          CALL  FIND_INT1       ; 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 restoring channel.

          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.

;   First consider INK and PAPER.

CO_TEMP_5 SUB   $11             ; reduce range $FF-$04
          LD    E,$00           ;+ Set E to zero.
;;;       ADC   A,$00           ; add in carry if INK
          ADC   A,E             ;+ add in carry if INK
          JR    Z,CO_TEMP_7     ; forward to CO-TEMP-7 with INK and PAPER.

;   Now consider FLASH and BRIGHT.

          SUB   $02             ; reduce range $FF-$02
;;;       ADC   A,$00           ; add carry if FLASH
          ADC   A,E             ;+ add carry if FLASH
          JR    Z,CO_TEMP_C     ; forward to CO-TEMP-C with FLASH and BRIGHT.

;   Now consider remaining INVERSE and OVER.

          INC   E               ;+ now make E=1
;;;       CP    $01             ; is it 'INVERSE' ?
          CP    E               ; is it 'INVERSE' ?
          LD    A,D             ; fetch parameter for INVERSE/OVER
;;;       LD    B,$01           ; prepare OVER mask setting bit 0.
          LD    B,E             ; prepare OVER mask setting bit 0.
          JR    NZ,CO_TEMP_6    ; forward to CO-TEMP-6 if OVER

;   Deal with INVERSE.

          RLCA                  ; shift bit 0
          RLCA                  ; to bit 2
          LD    B,$04           ; set bit 2 of mask for INVERSE.

;   The OVER path rejoins here.

CO_TEMP_6 LD    C,A             ; save the A
          LD    A,D             ; re-fetch parameter
          CP    $02             ; is it less than 2
          JR    NC,REPORT_K     ; to REPORT-K if not 0 or 1.
                                ; 'Invalid colour'.

          LD    A,C             ; restore A
          LD    HL,$5B91        ; address system variable P_FLAG
          JR    CO_CHANGE       ; forward to exit via routine CO-CHANGE

; ---

;   the branch was here with INK and PAPER and carry set for INK.

CO_TEMP_7 LD    A,D             ; fetch parameter
          LD    B,$07           ; set ink mask 00000111
          JR    C,CO_TEMP_8     ; 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

;   the INK path rejoins here.

CO_TEMP_8 LD    C,A             ; value to C
          LD    A,D             ; fetch parameter
          CP    $0A             ; is it less than 10 decimal ?
          JR    C,CO_TEMP_9     ; forward, if so, to CO-TEMP-9

;   INK 10 etc. is not allowed.

REPORT_K  RST   30H             ; ERROR-1
          DEFB  $13             ; Error Report: Invalid colour

; ---

CO_TEMP_9 LD    HL,$5B8F        ; Address system variable ATTR_T initially.
          CP    $08             ; compare with 8
          JR    C,CO_TEMP_B     ; forward to CO-TEMP-B with 0-7.

          LD    A,(HL)          ; fetch temporary attribute as no change.
          JR    Z,CO_TEMP_A     ; 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,CO_TEMP_A     ; forward to CO-TEMP-A if black and
                                ; originally light.

          LD    A,B             ; else just use the mask (white)

CO_TEMP_A LD    C,A             ; save A in C

CO_TEMP_B LD    A,C             ; load colour to A
          CALL  CO_CHANGE       ; 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  CO_CHANGE       ; 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

; ---------------------------
; THE 'COLOUR CHANGE' ROUTINES
; ---------------------------
;   This routine addresses a system variable ATTR_T, MASK_T or P-FLAG in HL.
;   colour value in A, mask in B.

CO_CHANGE 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 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,CO_TEMP_D    ; forward, if FLASH, to CO-TEMP-D

          RRCA                  ; rotate bit 7 to bit 6
          LD    B,$40           ; mask for BRIGHT - %01000000

CO_TEMP_D LD    C,A             ; store value in C
          LD    A,D             ; fetch parameter
          CP    $08             ; compare with 8
          JR    Z,CO_TEMP_E     ; forward, if eight, to CO-TEMP-E

          CP    $02             ; test if 0 or 1
          JR    NC,REPORT_K     ; back, if not, to REPORT-K
                                ; 'Invalid colour'

CO_TEMP_E LD    A,C             ; value to A
          LD    HL,$5B8F        ; address ATTR_T
          CALL  CO_CHANGE       ; routine CO-CHANGE addressing ATTR_T
          LD    A,C             ; fetch value
          RRCA                  ; for flash8/bright8 complete the
          RRCA                  ; rotations to put set bit in
          RRCA                  ; bit 7 (flash) bit 6 (bright)
          JR    CO_CHANGE       ; back to CO-CHANGE addressing MASK_T
                                ; and indirect return.

; --------------------
; THE '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.
;   This is a CLASS_0 command so syntax is checked by the tables and this
;   routine is only invoked in runtime.

BORDER    CALL  FIND_INT1       ; routine FIND-INT1
          CP    $08             ; must be in range 0 (black) to 7 (white)
          JR    NC,REPORT_K     ; back, if not, to REPORT-K
                                ; '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 paper number light coloured ?
```

```
                                ; i.e. in the range green to white.

          JR    NZ,BORDER_1     ; skip, if so, to BORDER-1

          XOR   $07             ; make the ink white.

BORDER_1  LD    ($5B48),A       ; update BORDCR with new paper/ink

          RET                   ; return.

; ---------------------------
; THE 'PIXEL ADDRESS' ROUTINE
; ---------------------------
;
;

PIXEL_ADD LD    A,$AF           ; load with 175 decimal.
          SUB   B               ; subtract the y value.
          JR    C,REPORT_Bz     ; 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 of the address 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

; ----------------------
; THE 'POINT' SUBROUTINE
; ---------------------
;   The point subroutine is called from s_point via the SCANNING functions
;   table.
;   Error B unless 0<=x<=255 and 0<=y<=175.
;   In accordance with the BASIC manual, parameters must now be positive.

POINT_SUB CALL  BC_POSTVE       ; routine BC_POSTVE but with check on signs.

          CALL  PIXEL_ADD       ; routine PIXEL-ADD finds address of pixel
                                ; producing an error if y is > 175.

          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  RLCA                  ; rotate and loop back
          DJNZ  POINT_LP        ; to POINT-LP until required pixel at right.

          AND   $01             ; test to give zero or one.
          JP    STACK_A         ; jump forward to STACK-A to save result.

; ------------------
; THE 'PLOT' COMMAND
; ------------------
;   Command Syntax example: PLOT 128,88
;

PLOT      CALL  BC_POSTVE       ; routine BC_POSTVE

;;;       CALL  PLOT_SUB        ; routine PLOT-SUB

;;;       JP    TEMPs           ;?to TEMPs to impose the permanent attributes
;;;                             ; onto the temporary ones as they may have been
;;;                             ; disturbed by embedded colour items ??

; ---------------------
; 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 although the colours may change.

PLOT_SUB  LD    ($5B7D),BC      ; store new x/y values in COORDS

          CALL  PIXEL_ADD       ; 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 RRCA                  ; rotate mask.
          DJNZ  PLOT_LOOP       ; 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,PL_TST_IN    ; forward, if so, to PL-TST-IN
```

```
;   was OVER 0.

          AND   B               ; combine with mask to blank pixel.

PL_TST_IN BIT   2,C             ; is it inverse 1 ?
          JR    NZ,PLOT_END     ; forward, if so, to PLOT-END

          XOR   B               ; switch the pixel
          CPL                   ; restore other 7 bits

PLOT_END  LD    (HL),A          ; load byte to the screen.
          JP    PO_ATTR         ; exit via PO-ATTR to set colours for cell.

; ---


REPORT_Bz RST   30H             ; ERROR-1
          DEFB  $0A             ; Error Report: Integer out of range

; ----------------------------------------------
; THE 'CALCULATOR STACK TO BC REGISTERS' ROUTINE
; ----------------------------------------------
;
;

STK_TO_BC CALL  STK_TO_A        ; routine STK-TO-A

          LD    B,A             ;
          PUSH  BC              ;

          CALL  STK_TO_A        ; routine STK-TO-A

          LD    E,C             ;
          POP   BC              ;
          LD    D,C             ;
          LD    C,A             ;

          RET                   ; Return.

; ---------------------------------------------
; THE 'CALCULATOR STACK TO ACCUMULATOR' ROUTINE
; ---------------------------------------------
;   This routine puts the last value on the calculator stack into the
;   accumulator deleting the last value.

STK_TO_A  CALL  FP_TO_A         ; routine FP-TO-A compresses last value into
                                ; accumulator. e.g. PI would become 3.
                                ; zero flag set if positive.

          JR    C,REPORT_Bz     ; forward, if >= 255, to REPORT-Bc
                                ; 'Integer out of range'

          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.


; --------------------
; THE 'CIRCLE' COMMAND
; --------------------
;   Syntax has been partly checked using the class for the DRAW command.

CIRCLE    RST   18H               ; GET-CHAR
          CP    $2C               ; Is character the required comma ?
          JP    NZ,REPORT_C       ; Jump, if not, to REPORT-C


;;;       RST   20H               ; NEXT-CHAR
;;;       CALL  EXPT_1NUM         ; Routine EXPT-1NUM fetches the radius.
;;;       CALL  CHECK_END         ; Routine CHECK-END will return here

          CALL  CHK_END_1         ; above 3 routines combined.

;   Continue in runtime.

          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 exponent byte
          CP    $81               ; Compare to exponent for one
          JR    NC,C_R_GRE_1      ; Forward to C-R-GRE-1 if circle radius is
                                  ; greater than a half.

;   If the diameter is no greater than one then delete the radius and plot
;   the single point.

          RST   28H               ;; FP-CALC
          DEFB  $02               ;;delete         ; delete the radius from stack.
          DEFB  $38               ;;end-calc

          JR    PLOT              ; Back to PLOT to just plot x,y.

; ---

;   Continue if the radius is greater than 1.

C_R_GRE_1 RST   28H               ;; FP-CALC       x, y, r
          DEFB  $A3               ;;stk-pi/2       x, y, r, pi/2.
          DEFB  $38               ;;end-calc       x, y, r, pi/2.

;   Cleverly multiply by four to form the circumference.

          LD    (HL),$83          ; bump exponent x, y, r, 2*PI

          RST   28H               ;; FP-CALC       x, y, r, 2*PI
          DEFB  $C5               ;;st-mem-5       store  2*PI in mem-5
          DEFB  $02               ;;delete         x, y, r.
          DEFB  $38               ;;end-calc       x, y, r.

          CALL  CD_PRMS1          ; routine CD_PRMS1 forms circle parameters.

          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,C_ARC_GE1      ; to C-ARC-GE1
          RST   28H             ;; FP-CALC
          DEFB  $02             ;;delete
          DEFB  $02             ;;delete
          DEFB  $38             ;;end-calc

          POP   BC              ;
;;;       JP    PLOT            ; JUMP to PLOT
          JR    PLOT            ;+ use relative jump to PLOT

; ---


C_ARC_GE1 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  FIND_INT1       ; routine FIND-INT1
          LD    L,A             ;
          PUSH  HL              ;
          CALL  FIND_INT1       ; routine FIND-INT1
          POP   HL              ;
          LD    H,A             ;
          LD    ($5B7D),HL      ; COORDS
          POP   BC              ;
          JP    DRW_STEPS       ; to DRW-STEPS


; -------------------
; THE 'DRAW' COMMAND
; -------------------
;   The DRAW command is rather more sophisticated than anything contemplated
;   for the ZX80 and ZX81 and, with a third parameter, it can draw an arc.
;   At this stage, syntax has been partly checked by the class routines and
;   the 'x, y' parameters have been verified.

DRAW      RST   18H             ; GET-CHAR
          CP    $2C             ; is character the optional ',' ?
          JR    Z,DR_3_PRMS     ; forward, if so, to DR_3_PRMS

          CALL  CHECK_END       ; routine CHECK-END checks that nothing follows.

          JP    LINE_DRAW       ; jump forward, in runtime, to LINE-DRAW

; ---
;   The branch was here when a comma indicated a third parameter was expected.

DR_3_PRMS CALL  CHK_END_1       ; following three routines combined.

;;;       RST   20H             ; NEXT-CHAR advances.
;;;       CALL  EXPT_1NUM       ; routine EXPT-1NUM checks for numeric
;;;       CALL  CHECK_END       ; routine CHECK-END

          RST   28H             ;; FP-CALC      x, y, z.
          DEFB  $C5             ;;st-mem-5      x, y, z.
          DEFB  $A2             ;;stk-half      x, y, z, 1/2.
          DEFB  $04             ;;multiply      x, y, z/2.
          DEFB  $1F             ;;sin
          DEFB  $31             ;;duplicate
          DEFB  $30             ;;not
          DEFB  $30             ;;not
          DEFB  $00             ;;jump-true

          DEFB  DR_SIN_NZ - $   ;;to DR_SIN_NZ

          DEFB  $02             ;;delete
          DEFB  $38             ;;end-calc

          JP    LINE_DRAW       ; to LINE-DRAW

; ---

DR_SIN_NZ 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,DR_PRMS      ; to DR-PRMS


          RST   28H             ;; FP-CALC
          DEFB  $02             ;;delete
          DEFB  $02             ;;delete
          DEFB  $38             ;;end-calc

          JP    LINE_DRAW       ; to LINE_DRAW

; ---

DR_PRMS   CALL  CD_PRMS1        ; routine CD_PRMS1 forms draw parameters.

          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              ;
JR    C,LINE_DRAW     ; JUMP to LINE-DRAW

PUSH  BC              ;
```

```
          RST   28H             ;; FP-CALC
          DEFB  $01             ;;exchange
          DEFB  $38             ;;end-calc

          LD    A,($5B7D)       ; COORDS-x
          CALL  STACK_A         ; routine STACK-A

          RST   28H             ;; FP-CALC
          DEFB  $C0             ;;st-mem-0
          DEFB  $0F             ;;addition
          DEFB  $01             ;;exchange
          DEFB  $38             ;;end-calc

          LD    A,($5B7E)       ; COORDS-y
          CALL  STACK_A         ; 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 DEC   B               ;
          JR    Z,ARC_END       ; to ARC-END

          JR    ARC_START       ; to ARC-START

; ---


ARC_LOOP  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 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,($5B7D)       ; COORDS-x
          CALL  STACK_A         ; 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,($5B7E)       ; COORDS-y
          CALL  STACK_A         ; routine STACK-A

          RST   28H             ;; FP-CALC
          DEFB  $03             ;;subtract
          DEFB  $38             ;;end-calc

          CALL  DRAW_LINE       ; routine DRAW-LINE

          POP   BC              ;
          DJNZ  ARC_LOOP        ; to ARC-LOOP


ARC_END   RST   28H             ;; FP-CALC
          DEFB  $02             ;;delete
          DEFB  $02             ;;delete
          DEFB  $01             ;;exchange
          DEFB  $38             ;;end-calc

          LD    A,($5B7D)       ; COORDS-x
          CALL  STACK_A         ; routine STACK-A

          RST   28H             ;; FP-CALC
          DEFB  $03             ;;subtract
          DEFB  $01             ;;exchange
          DEFB  $38             ;;end-calc

          LD    A,($5B7E)       ; COORDS-y
          CALL  STACK_A         ; routine STACK-A

          RST   28H             ;; FP-CALC
          DEFB  $03             ;;subtract
          DEFB  $38             ;;end-calc

LINE_DRAW
          JR    DRAW_LINE       ; routine DRAW-LINE

;;;       CALL  DRAW_LINE       ; routine DRAW_LINE
;;;       JP    TEMPs           ;?to TEMPs


; -----------------------------------------------------
; THE 'CIRCLE AND DRAW INITIAL PARAMETERS' SUBROUTINE
; -----------------------------------------------------
;
;

CD_PRMS1  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  FP_TO_A         ; routine FP-TO-A
          JR    C,USE_252       ; to USE-252

          AND   $FC             ;
          ADD   A,$04           ;
          JR    NC,DRAW_SAVE    ; to DRAW-SAVE

USE_252   LD    A,$FC           ;

DRAW_SAVE PUSH  AF              ;

          CALL  STACK_A         ; 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                   ;

; ---------------------------
; THE 'DRAW LINE' SUBROUTINE
; ---------------------------
;   B=Y C=X D=signY E=signX
;

DRAW_LINE CALL  STK_TO_BC       ; routine STK-TO-BC

          LD    A,C             ; load X to accumulator.
          CP    B               ; compare to Y.
          JR    NC,DL_X_GE_Y    ; forward, if X greater or equal, to DL-X-GE-Y

;    Y is greater than X

          LD    L,C             ; load L with X
                                ;      B  is  Y
          PUSH  DE              ; save signs.
          XOR   A               ;
          LD    E,A             ; set E to zero.

          JR    DL_LARGER       ; to DL-LARGER

; ---

DL_X_GE_Y OR    C               ; check if x is zero
          RET   Z               ; return if so

          LD    L,B             ; load L with Y
          LD    B,C             ; load B with X
          PUSH  DE              ; save signs.
          LD    D,$00           ; set D to zero.

DL_LARGER LD    H,B             ; load H with larger
          LD    A,B             ; to A also
          RRA                   ;                        LET S = INT (M/2)

D_L_LOOP  ADD   A,L             ;
          JR    C,D_L_DIAG      ; to D-L-DIAG

          CP    H               ;
          JR    C,D_L_HR_VT     ; to D-L-HR-VT

D_L_DIAG  SUB   H               ;
          LD    C,A             ;
          EXX                   ;
          POP   BC              ;
          PUSH  BC              ;
          JR    D_L_STEP        ; to D-L-STEP

; ---

D_L_HR_VT LD    C,A             ;
          PUSH  DE              ;
          EXX                   ;
          POP   BC              ;

D_L_STEP  LD    HL,($5B7D)      ; COORDS
          LD    A,B             ;
          ADD   A,H             ;
          LD    B,A             ;
          LD    A,C             ;
          INC   A               ;
          ADD   A,L             ;
          JR    C,D_L_RANGE     ; to D-L-RANGE

          JR    Z,REPORT_Bc     ; to REPORT-Bc
                                ; 'Integer out of range'

D_L_PLOT  DEC   A               ;
          LD    C,A             ;
          CALL  PLOT_SUB        ; routine PLOT-SUB
          EXX                   ;
          LD    A,C             ;
          DJNZ  D_L_LOOP        ; to D-L-LOOP

          POP   DE              ;
          RET                   ;

; ---

D_L_RANGE JR    Z,D_L_PLOT      ; to D-L-PLOT


REPORT_Bc RST   30H             ; 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.

; -------------------------
; THE 'SCANNING' SUBROUTINE
; -------------------------
;   Scan expression or sub-expression
;   The routine begins and ends with a RST 18H instruction.

SCANNING  RST   18H             ; GET-CHAR
          LD    B,$00           ; Priority marker zero is pushed on stack to
                                ; signify the end of expression when it is
                                ; popped off again.

          PUSH  BC              ; Stack the marker byte and proceed to consider
                                ; the first character of the expression.

S_LOOP_1
;;;       LD    C,A             ; place the search character in C.
          LD    HL,SCAN_FUNC-1  ; Address: scan-func
          CALL  INDEXER_0       ; 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,S_ALPHNUM    ; jump forward to S-ALPHNUM if not in complex
                                ; 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)            ; 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 CALL  CH_ADD__1       ; 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.
          JR    Z,REPORT_Cs     ; jump forward, if so, to REPORT-C
                                ; 'Nonsense in BASIC'.

          CP    $22             ; is it a quote '"' ?
          JR    NZ,S_QUOTE_S    ; back, if not, to S-QUOTE-S

          CALL  CH_ADD__1       ; 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 RST   20H             ; NEXT-CHAR

          CP    $28             ; is character the opening '(' ?
          JR    NZ,REPORT_Cs    ; forward, if not, to S-RPORT-C
                                ; 'Nonsense in BASIC'.

          CALL  NEXT_2NUM       ; 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 was exit route for SCANNING


TST_RBRKT CP    $29             ; is it the closing ')' ?


S_RPORT_C JR    NZ,REPORT_Cs    ; jump forward, if not, to REPORT-Cs



; -------------------------
; THE 'SYNTAX_Z' SUBROUTINE
; -------------------------
;   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  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_SCRNs_S CALL  STK_TO_LC       ; NEW routine STK-TO-LC.

          LD    HL,($5B36)      ; fetch address of CHARS.

;;;       LD    DE,$0100        ; fetch offset to chr$ 32
;;;       ADD   HL,DE           ; and find start of bitmaps.
;;;                             ; Note. not inc h. ??

          INC   H               ;+ increment high byte to address bitmaps.

          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 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,S_SC_MTCH     ; 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,S_SCR_NXT    ; 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 LD    C,A             ; load C with inverse mask $00 or $FF
          LD    B,$07           ; count seven more bytes

S_SC_ROWS 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,S_SCR_NXT    ; forward to S-SCR-NXT if no match.

          DJNZ  S_SC_ROWS       ; 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.

          CALL  BC_SPACE1       ; BC_SPACES creates the 1 space sliding
                                ; the calculator stack upwards.
          LD    (DE),A          ; start is addressed by DE, so insert code
          JR    S_SCR_STO       ; forward to S-SCR-STO

; ---

;   the jump was here if no match and more bitmaps to test.

S_SCR_NXT 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  S_SCRN_LP       ; back to S-SCRN-LP if more characters.

          LD    C,B             ; B is now zero, so BC now zero.

S_SCR_STO RET                   ; (WAS 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 S_SCR_STO should just be a RET. (Done! SEP-2002)
;   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  CALL  STK_TO_LC       ; NEW 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                  ; right.
          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 is 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    STACK_A         ; jump forward to STACK-A to store result
```

```
                                ; and make an indirect exit.

; ---

REPORT_Cs RST   30H             ; ERROR_1
          DEFB  $0B             ; Error Report: Nonsense in BASIC

; ----------------------------
; THE '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 DEFB  $22, S_QUOTE   -$-1 ; $1C offset to S-QUOTE
          DEFB  '(', S_BRACKET -$-1 ; $4F offset to S-BRACKET
          DEFB  '.', S_DECIMAL -$-1 ; $F2 offset to S-DECIMAL
          DEFB  '+', S_U_PLUS  -$-1 ; $12 offset to S-U-PLUS

          DEFB  $A8, S_FN      -$-1 ; $56 offset to S-FN
          DEFB  $A5, S_RND     -$-1 ; $57 offset to S-RND
          DEFB  $A7, S_PI      -$-1 ; $84 offset to S-PI
          DEFB  $A6, S_INKEYs  -$-1 ; $8F offset to S-INKEY$
          DEFB  $C4, S_DECIMAL -$-1 ; $E6 offset to S-BIN
          DEFB  $AA, S_SCREENs -$-1 ; $BF offset to S-SCREEN$
          DEFB  $AB, S_ATTR    -$-1 ; $C7 offset to S-ATTR
          DEFB  $A9, S_POINT   -$-1 ; $CE offset to S-POINT

          DEFB  $00                 ; zero end marker

; -------------------------------
; THE 'SCANNING FUNCTION' ROUTINES
; -------------------------------
;   These are the 11 subroutines accessed by the above table.
;   Addresses S-BIN and S-DECIMAL are the same
;   The 1-byte offset limits their location to within 255 bytes of their
;   entry in the above table.

; ->
S_U_PLUS

          JP    S_NEXT_1        ;+ forward to similar code.

;;;       RST   20H             ; NEXT-CHAR just ignore
;;;       JP    S_LOOP_1        ; back to S-LOOP-1
```

```
; ---

; ->
S_QUOTE   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  S_QUOTE_S       ; routine S-QUOTE-S

          JR    NZ,S_Q_PRMS     ; forward to S-Q-PRMS if

S_Q_AGAIN CALL  S_QUOTE_S       ; routine S-QUOTE-S copies string until a
                                ; quote is encountered

          JR    Z,S_Q_AGAIN     ; back to S-Q-AGAIN if two quotes WERE
                                ; together.

;   but if just an isolated quote then that terminates the string.

          CALL  SYNTAX_Z        ; routine SYNTAX-Z

          JR    Z,S_Q_PRMS      ; forward, if checking syntax, to S-Q-PRMS

;   In runtime, build the string expression result.


          CALL  BC_SPACES       ; routine BC_SPACES creates the space for true
                                ; copy of string in workspace.

          POP   HL              ; re-fetch start of quoted text.
          PUSH  DE              ; stack DE the start of string in workspace.

S_Q_COPY  LD    A,(HL)          ; fetch a character from source.
          INC   HL              ; advance the source address.
          LD    (DE),A          ; place in destination.
          INC   DE              ; advance the destination address.

          CP    $22             ; was it a '"' just copied ?
          JR    NZ,S_Q_COPY     ; back, if not, to S-Q-COPY

          LD    A,(HL)          ; fetch adjacent character from source.
          INC   HL              ; advance the source address.

          CP    $22             ; is this '"' ? - i.e. two quotes together ?
          JR    Z,S_Q_COPY      ; to S-Q-COPY if so including just one of the
                                ; pair of quotes.

;   If not two adjacent quotes then the terminating quote has just been copied.

S_Q_PRMS  DEC   BC              ; decrease the count by 1.
          POP   DE              ; restore start of string in workspace.

S_STRING  LD    HL,$5B3B        ; Address the FLAGS system variable.
          RES   6,(HL)          ; signal a string result.
          BIT   7,(HL)          ; is syntax being checked ?

          CALL  NZ,STK_STO_s    ; routine STK-STO-$ is called in runtime.

          JR    S_INKs_EN       ; jump forward to S-CONT-2          ===>

; ---
; ->
S_BRACKET RST   20H              ; NEXT-CHAR

          CALL  SCANNING         ; routine SCANNING is called recursively.

;;;       CP    $29              ; is it the closing ')' ?
;;;       JR    NZ,REPORT_Cs     ;.jump back, if not, to REPORT-C

          CALL  TST_RBRKT        ; test for a right bracket ')'

          RST   20H              ; NEXT-CHAR
          JR    S_INKs_EN        ; jump forward to S-CONT-2          ===>

; ---

; ->
S_FN      JP    S_FN_SBRN        ; jump forward to S-FN-SBRN.

; ---

; ->
S_RND     CALL  SYNTAX_Z         ; routine SYNTAX-Z

          JR    Z,S_RND_END      ; forward to S-RND-END if checking syntax.

          LD    BC,($5B76)       ; fetch system variable SEED

          CALL  STACK_BC         ; 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  FP_TO_BC         ; routine FP-TO-BC

          LD    ($5B76),BC       ; store in SEED for next starting point.

          LD    A,(HL)           ; fetch exponent
          AND   A                ; is it zero ?
          JR    Z,S_RND_END      ; forward if so to S-RND-END

          SUB   $10              ; reduce exponent by 2^16
          LD    (HL),A           ; place back

S_RND_END JR    S_PI_END         ; forward to S-PI-END

; ---

;   the number PI 3.14159...
; ->
S_PI      CALL  SYNTAX_Z        ; routine SYNTAX-Z
          JR    Z,S_PI_END      ; 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
          JR    S_AT_NUM        ;+ forward to similar code ending at S_NUMERIC

;;;       RST   20H             ; NEXT-CHAR
;;;       JP    S_NUMERIC       ; jump forward to S-NUMERIC

; ---

; ->
S_INKEYs  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,S_PUSH_PO     ; To S-PUSH-PO if so to use the calculator
                                ; single operation to read from network/RS232.

;   else read a key from the keyboard.

          LD    HL,$5B3B        ; fetch FLAGS
          RES   6,(HL)          ; signal string result.
          BIT   7,(HL)          ; checking syntax ?
          JR    Z,S_INKs_EN     ; forward, if so, to S-INK$-EN

          CALL  KEY_SCAN        ; routine KEY-SCAN key in E, shift in D.

          LD    C,$00           ; prepare the length of an empty string
          JR    NZ,S_IKs_STK    ; forward, if no key returned, to S-IK$-STK
                                ; to store empty string.

          CALL  K_TEST          ; routine K-TEST get main code in A

          JR    NC,S_IKs_STK    ; forward, if invalid, to S-IK$-STK
                                ; to stack null string.

          DEC   D               ; D is expected to be FLAGS so set bit 3 $FF
                                ; 'L' Mode so no keywords.

;;;       LD    E,A             ; main key to E
                                ; C is MODE 0 'KLC' from above still.

          CALL  K_DECODE2       ; routine K_DECODE but skip first.

;;;       PUSH  AF              ; save the code

;;;       LD    BC,$0001        ; make room for one character

          CALL  BC_SPACE1       ; routine BC_SPACE1 creates a single space.

;;;       POP   AF              ; bring the code back
```

```
          LD    (DE),A          ; put the key in workspace
;;;       LD    C,$01           ; set C length to one      (BC=1)

S_IKs_STK LD    B,$00           ; set high byte of length to zero
          CALL  STK_STO_s       ; routine STK-STO-$

S_INKs_EN JP    S_CONT_2        ; to S-CONT-2            ===>

; ---

; ->
S_SCREENs CALL  S_2_COORD       ; routine S-2-COORD

          CALL  NZ,S_SCRNs_S    ; routine S-SCRN$-S in runtime.

          RST   20H             ; NEXT-CHAR
          JP    S_STRING        ; Back to S-STRING to stack runtime result

; ---

; ->
S_ATTR    CALL  S_2_COORD       ; routine S-2-COORD

          CALL  NZ,S_ATTR_S     ; routine S-ATTR-S in runtime.

S_AT_NUM  RST   20H             ; NEXT-CHAR
          JR    S_NUMERIC       ; forward to S-NUMERIC

; ---

; ->
S_POINT   CALL  S_2_COORD       ; routine S-2-COORD

          CALL  NZ,POINT_SUB    ; routine POINT-SUB in runtime.

          JR    S_AT_NUM        ;+ back to similar sequence ending at S_NUMERIC

;;;       RST   20H             ; NEXT-CHAR
;;;       JR    S_NUMERIC       ; forward to S-NUMERIC

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

;   ==> The branch was here if not in table of exceptions.

S_ALPHNUM CALL  ALPHANUM        ; routine ALPHANUM checks if a variable or
                                ; a digit follows.

          JR    NC,S_NEGATE     ; forward, if not, to S-NEGATE
                                ; to consider a '-' character then functions.

          CP    $41             ; compare with 'A'
          JR    NC,S_LETTER     ; forward, if alpha, to S-LETTER        ->

;   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_DECIMAL CALL  SYNTAX_Z        ; routine SYNTAX-Z
          JR    NZ,S_STK_DEC    ; to S-STK-DEC in runtime
```

```
;   this route is taken when checking syntax.

          CALL  DEC_TO_FP       ; routine DEC-TO-FP to evaluate number

          RST   18H             ; GET-CHAR to fetch HL

          LD    BC,$0006        ; six locations are required
          CALL  MAKE_ROOM       ; routine MAKE-ROOM
;;;       INC   HL              ;
          LD    (HL),$0E        ; insert number marker at first location.
          INC   HL              ; address next location to receive the 5 bytes.

          EX    DE,HL           ; make DE destination.

          CALL  GET_5           ;+ NEW subroutine below embodies following

;;;       LD    HL,($5B65)      ; 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    ($5B65),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  TEMP_PTR1       ; routine TEMP-PTR1 sets CH-ADD.

          JR    S_NUMERIC       ; to S-NUMERIC to record nature of result

; ---

GET_5     LD    HL,($5B65)      ;+ STKEND points to end of stack.
          LD    BC,$0005        ;+ There are five bytes to copy.
          AND   A               ;+ Clear carry flag.
          SBC   HL,BC           ;+ Reduce HL by five.
          LD    ($5B65),HL      ;+ update STKEND as we are taking number.
          LDIR                  ;+ Copy five bytes to location
          EX    DE,HL           ;+ transfer pointer to HL
          DEC   HL              ;+ adjust
          RET                   ;+ Return
; ---

;   S_DECIMAL branches here in runtime to pick up prepared number.

S_STK_DEC RST   18H             ; GET-CHAR positions HL at digit.

S_SD_SKIP INC   HL              ; advance pointer
          LD    A,(HL)          ; until we find
          CP    $0E             ; chr 14d - the number indicator
          JR    NZ,S_SD_SKIP    ; loop back, until match found, to S-SD-SKIP
                                ; it has to be here.

          INC   HL              ; point to first byte of number

          CALL  STACK_NUM       ; routine STACK-NUM stacks it.

          LD    ($5B5D),HL      ; update system variable CH_ADD

S_NUMERIC SET   6,(IY+$01)      ; update FLAGS  - Signal numeric result
;;;       JR    S_CONT_1        ; forward to S-CONT-1               ===>
          JR    S_CONT_2        ;+ forward now directly to S-CONT-2

;   end of functions accessed from scanning functions table.
; --------------------------
; Scanning variable routines
; --------------------------
;
;

S_LETTER  CALL  LOOK_VARS       ; routine LOOK-VARS

          JP    C,REPORT_2      ; jump back to REPORT-2 if not found
                                ; 'Variable not found'
                                ; but a variable is always 'found' if syntax
                                ; is being checked.

          CALL  Z,STK_VAR       ; routine STK-VAR considers a subscript/slice
          LD    A,($5B3B)       ; fetch FLAGS value
          CP    $C0             ; compare 11000000
;;;       JR    C,S_CONT_1      ; step forward to S-CONT-1 if string  ===>
          JR    C,S_CONT_2      ;+ step forward to S-CONT-2 if string  ===>

;   The variable is a simple numeric variable.

          INC   HL              ; advance pointer past last letter.

          CALL  STACK_NUM       ; routine STACK-NUM

S_CONT_1  JR    S_CONT_2        ; 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  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 the character '-' ?
          JR    Z,S_PUSH_PO     ; 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,S_PUSH_PO     ; 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.

          JR    C,REPORT_Cw     ; forward, with anything else, to REPORT-C
                                ; 'Nonsense in BASIC'

          LD    BC,$04F0        ; prepare priority $04, operation $C0 +
                                ; 'not' ($30)
          CP    $14               ; is it 'NOT'
          JR    Z,S_PUSH_PO       ; forward, if so, to S-PUSH-PO

          JR    NC,REPORT_Cw      ; forward, if higher, to REPORT-C
                                  ; 'Nonsense in BASIC'

          LD    B,$10             ; priority $10 for all the rest
          ADD   A,$DC             ; make range $DC - $EF
                                  ; $C0 + 'code' ($1C) through 'chr$' ($2F)

          LD    C,A               ; transfer 'function' to C
          CP    $DF               ; compare to 'sin' ?
          JR    NC,S_NO_TO_s      ; forward to S-NO-TO-$  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_s CP    $EE               ; compare 'str$'
          JR    C,S_PUSH_PO       ; 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.
;   Push the Priority and Operand.

S_PUSH_PO PUSH  BC                ; push the priority and calculator operation
                                  ; code.


          JR    S_NEXT_1          ;+ forward to similar looping code.

;;;       RST   20H               ; NEXT-CHAR
;;;       JP    S_LOOP_1          ; jump back to S-LOOP-1 to go round the loop

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

;   ===>  there were many branches forward to here

S_CONT_2  RST   18H               ; GET-CHAR

S_CONT_3  CP    $28               ; is it '(' ?
          JR    NZ,S_OPERTR       ; forward, if not, to S-OPERTR    >

          BIT   6,(IY+$01)        ; test FLAGS - numeric or string result ?
          JR    NZ,S_LOOP         ; forward to S-LOOP if numeric to evaluate  >

;   if a string preceded '(' then slice it.

          CALL  SLICING           ; routine SLICING

S_CONT_4  RST   20H               ; NEXT-CHAR
          JR    S_CONT_3          ; back to S-CONT-3

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

;   the branch was here when possibility of an operator '(' has been excluded.
S_OPERTR
;;;       LD    B,$00           ; prepare to index.
;;;       LD    C,A             ; possible operator to C
          LD    HL,tbl_ofops-1  ; Address: tbl-of-ops

          CALL  INDEXER_0       ; routine INDEXER does look up sets B to zero.

          JR    NC,S_LOOP       ; 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,TBL_PRI-$C3  ; address theoretical base of table.

          ADD   HL,BC           ; index into table.
          LD    B,(HL)          ; priority to B.

; -----------------------
; Scanning Operation loop
; -----------------------
;   The juggling act.

S_LOOP    POP   DE              ; fetch last priority and operation code.
          LD    A,D             ; priority to A
          CP    B               ; compare with this one
          JR    C,S_TIGHTER     ; forward, with carry,  to S-TIGHTER
                                ; to execute the operation before this one as
                                ; it has a higher priority.

;   The last priority was greater or equal this one.

          AND   A               ; if it is zero then so is this

          JP    Z,GET_CHAR      ; jump, if zero, to exit via GET-CHAR
                                ; pointing CH_ADD 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 the current priority/operation as it
                                ; must have 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,$5B3B        ; Address the FLAGS system variable.
          LD    A,E             ; new operation to A register
          CP    $ED             ; is it $C0 + 'usr-no' ($2D)  ?
          JR    NZ,S_STK_LST    ; forward, if not, to S-STK-LST

          BIT   6,(HL)          ; is a string result expected ?
                                ; (from the lower priority operand we've
                                ; just pushed on stack )
          JR    NZ,S_STK_LST    ; forward, if numeric, to S-STK-LST
                                ; as the operand bits match.

          LD    E,$99           ; reset bit 6 and substitute $19 'usr-$'
                                ; for a string operand.

S_STK_LST PUSH  DE              ; now stack this priority/operation code.
```

```
          CALL  SYNTAX_Z        ; routine SYNTAX-Z

          JR    Z,S_SYNTEST     ; forward, if checking syntax, to S-SYNTEST

          LD    A,E             ; fetch the operation code.
          AND   $3F             ; mask off the result/operand bits to leave
                                ; a calculator literal.
          LD    B,A             ; transfer naked 'literal' to B register

;   Now use the calculator to perform the single operation - the 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              operand.
          DEFB  $3B             ;;fp-calc-2
          DEFB  $38             ;;end-calc              result.

          JR    S_RUNTEST       ; forward to S-RUNTEST

; ---

;   The branch was here if checking syntax only.

S_SYNTEST LD    A,E             ; fetch the operation code to the accumulator.
          XOR   (IY+$01)        ; XOR with the FLAGS system variable.
          AND   $40             ; bit 6 will be zero now if operand
                                ; matched expected result.

          JR    Z,S_RUNTEST     ; skip forward, if results match.

REPORT_Cw RST   30H             ; ERROR-1
          DEFB  $0B             ; Error Report: Nonsense in BASIC


;;; S_RPRT_C2 JP    NZ,REPORT_C     ; to REPORT-C if mismatch

;   else continue to set flags for next operation.
;   The branch is to here in runtime after a successful operation.

S_RUNTEST POP   DE              ; fetch the last operation from stack
          LD    HL,$5B3B        ; address FLAGS system variable.
          SET   6,(HL)          ; set default FLAGS result to numeric.
          BIT   7,E             ; test the operational result.
          JR    NZ,S_LOOPEND    ; forward, if numeric, to S-LOOPEND

          RES   6,(HL)          ; reset bit 6 of FLAGS to show string result.

S_LOOPEND POP   BC              ; fetch the previous priority/operation

          JR    S_LOOP          ; 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 PUSH  DE              ; save higher priority/operator on stack again.
          LD    A,C             ; fetch the lower priority/operation code.
          BIT   6,(IY+$01)       ; test FLAGS - numeric or string result ?
          JR    NZ,S_NEXT        ; forward, if numeric, to S-NEXT

;   If this is lower priority, yet has a string result, then it must be a
;   comparison.  Since these can only be evaluated in context and were
;   defaulted to numeric at the operator look-up stage, they must be changed
;   to their string equivalents.

          AND   $3F              ; mask to give the true calculator literal.
          ADD   A,$08            ; augment numeric literals to their 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,S_NOT_AND     ; skip forward, if not, to S-NOT-AND

          SET   6,C              ; set numeric operand bit.
          JR    S_NEXT           ; forward to S-NEXT

; ---

;   The short branch was to here when string operators had been compared.

S_NOT_AND JR    C,REPORT_Cw      ; back, if less than '&', to S-RPORT-C2
                                 ; 'Nonsense in BASIC'
                                 ; e.g. with a$ * b$

          CP    $17              ; is it 'strs-add' ?
          JR    Z,S_NEXT         ; forward, if so, to S-NEXT
                                 ; as already set for a string result.

          SET   7,C              ; set numeric (Boolean) result for all others.

S_NEXT    PUSH  BC               ; now save this priority/operation on stack.

S_NEXT_1  RST   20H              ; NEXT-CHAR advances the character address.

          JP    S_LOOP_1         ; jump back to S-LOOP-1

; ---------------------
; THE 'OPERATORS' TABLE
; ---------------------
;   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 was intended. That is '+', '=', '>', '<', '<=', '>=' or '<>'.

tbl_ofops 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.


; ----------------------
; THE 'PRIORITIES' TABLE
; ----------------------
;   This table is indexed with the operation code obtained from the above
;   table, $C3 - $CF, to obtain the priority for the respective operation.

TBL_PRI 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 CALL  SYNTAX_Z        ; routine SYNTAX-Z
```

```
          JR    NZ,SF_RUN       ; forward to SF-RUN in runtime


          RST   20H             ; NEXT-CHAR
          CALL  ALPHA           ; routine ALPHA check for letters [A-Za-z]
          JR    NC,REPORT_Cw    ; jump back, if not, to REPORT-C
                                ; 'Nonsense in BASIC'


          RST   20H             ; NEXT-CHAR
          CP    $24             ; is it '$' ?

          PUSH  AF              ; (*) save the flags

          JR    NZ,SF_BRKT_1    ; forward, with numeric function, to SF-BRKT-1


          RST   20H             ; NEXT-CHAR advances past the '$'.

SF_BRKT_1 CP    $28             ; is character a '(' ?

SF_RPT_C  JR    NZ,REPORT_Cw    ; forward, if not, to SF-RPRT-C
                                ; 'Nonsense in BASIC'


          RST   20H             ; NEXT-CHAR
          CP    $29             ; is it ')' ?
          JR    Z,SF_FLAG_6     ; forward, if no arguments, to SF-FLAG-6

SF_ARGMTS CALL  SCANNING        ; routine SCANNING checks each argument which
                                ; may be an expression and ends with RST 18H.

;;;       RST   18H             ; GET-CHAR

          CP    $2C             ; is it a ',' ?
          JR    NZ,SF_BRKT_2    ; forward if not to SF-BRKT-2 to test bracket


          RST   20H             ; NEXT-CHAR if a comma was found
          JR    SF_ARGMTS       ; back to SF-ARGMTS to parse all arguments.

; ---

SF_BRKT_2
;;;       CP    $29             ; is character the closing ')' ?
;;;       JP    NZ,REPORT_C     ; Report 'Nonsense in basic' if not.

          CALL  TST_RBRKT       ;+ routine to test for right hand bracket.


;   at this point any optional arguments have had their syntax checked.

SF_FLAG_6 RST   20H             ; NEXT-CHAR

;;;       LD    HL,$5B3B        ; address system variable FLAGS
;;;       RES   6,(HL)          ; signal a string result

          CALL  STR_RSLT        ;+ set default result to string as 3 byte call.

          POP   AF              ; (*) restore test result against '$'.

          JP    Z,S_CONT_2      ;+ to S_CONT_2 if string
          JP    S_NUMERIC       ;+ else to S_NUMERIC.
```

```
;;;        JR    Z,SF_SYN_EN     ; skip forward to SF-SYN-EN if string function.
;;;        SET   6,(HL)          ; signal a numeric result.
;;; SF_SYN_EN JP  S_CONT_2       ; jump back to S-CONT-2 to continue scanning.

; ---

;   The branch was here in runtime.

SF_RUN     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,SF_ARGMT1    ; forward if not '$' to SF-ARGMT1

           RST   20H             ; NEXT-CHAR advances to bracket

SF_ARGMT1 RST    20H             ; NEXT-CHAR advances to start of argument
           PUSH  HL              ; save address
           LD    HL,($5B53)      ; fetch start of program area from PROG
           DEC   HL              ; the search starting point is the previous
                                 ; location.

SF_FND_DF 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  LOOK_PROG       ; routine LOOK-PROG will search for token.

           POP   BC              ; restore BC.
           JR    NC,SF_CP_DEF    ; forward to SF-CP-DEF if a match was found.

REPORT_P  RST    30H             ; ERROR-1
           DEFB  $18             ; Error Report: FN without DEF

SF_CP_DEF PUSH   HL              ; save address of DEF FN

           CALL  FN_SKPOVR       ; 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,SF_NOT_FD    ; forward to SF-NOT-FD if no match.

;   the letters match so test the type.

           CALL  FN_SKPOVR       ; 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,SF_VALUES     ; forward to SF-VALUES with a match.

;   the letters matched but one was string and the other numeric.

SF_NOT_FD 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  EACH_STMT       ; routine EACH-STMT steps past the rejected
```

```
                                      ; definition.

          POP    BC                   ; restore letter/type
          JR     SF_FND_DF            ; back to SF-FND-DF to continue search

; ---

;   Success!
;   the branch was here with matching letter and numeric/string type.

SF_VALUES AND    A                    ; test A ( will be zero if string '$' - '$' )

          CALL   Z,FN_SKPOVR          ; routine FN-SKPOVR advances HL past '$'.

          POP    DE                   ; discard pointer to 'DEF FN'.
          POP    DE                   ; restore pointer to first FN argument.
          LD     ($5B5D),DE           ; save address in CH_ADD

          CALL   FN_SKPOVR            ; routine FN-SKPOVR advances HL past the '('

          PUSH   HL                   ; save start address in DEF FN  ***
          CP     $29                  ; is character a ')' ?
          JR     Z,SF_R_BR_2          ; forward, if no arguments, to SF-R-BR-2

SF_ARG_LP INC    HL                   ; point to next character.
          LD     A,(HL)               ; fetch it to A.
          CP     $0E                  ; is it the number marker ?
          LD     D,$40                ; signal numeric in D.
          JR     Z,SF_ARG_VL          ; forward, if numeric, to SF-ARG-VL

          DEC    HL                   ; back to letter

          CALL   FN_SKPOVR            ; routine FN-SKPOVR skips any white-space

          INC    HL                   ; advance past the expected '$' to
                                      ; the 'hidden' marker.
          LD     D,$00                ; signal a string result.

SF_ARG_VL INC    HL                   ; now address first of 5-byte location.
          PUSH   HL                   ; save address in DEF FN statement
          PUSH   DE                   ; save D - result type

          CALL   SCANNING             ; 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,REPORT_Q          ; forward, with type mismatch, to REPORT-Q
                                      ; 'Parameter error'

;;;       POP    HL                   ; pop the start address in DEF FN statement
;;;       EX     DE,HL                ; transfer to DE ?? pop straight into de ?

          POP    DE                   ;+ pop the start address in DEF FN to DE.

          CALL   GET_5                ;+ NEW subroutine above embodies following

;;;       LD     HL,($5B65)           ; 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     ($5B65),HL           ; set STKEND thus '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  FN_SKPOVR      ; routine FN-SKPOVR gets next valid character
          CP    $29            ; is it ')' end of arguments ?
          JR    Z,SF_R_BR_2    ; forward, if so, to SF-R-BR-2

;   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 character the corresponding ',' ?
          JR    NZ,REPORT_Q    ; forward, if not, to REPORT-Q
                               ; 'Parameter error'

          RST   20H            ; NEXT-CHAR in FN statement advances to next
                               ; argument.

          POP   HL             ; restore DEF FN pointer
          CALL  FN_SKPOVR      ; routine FN-SKPOVR advances to corresponding
                               ; argument.

          JR    SF_ARG_LP      ; 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 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,SF_VALUE     ; forward, if so, to SF-VALUE

REPORT_Q  RST   30H            ; ERROR-1
          DEFB  $19            ; Error Report: Parameter error

SF_VALUE  POP   DE             ; restore location of ')' in DEF FN to DE.
          EX    DE,HL          ; now to HL, FN ')' pointer to DE.
          LD    ($5B5D),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,($5B0B)     ; get original DEFADD address
          EX    (SP),HL        ; swap with DEF FN address on stack ***
          LD    ($5B0B),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  SCANNING          ; routine SCANNING evaluates but searches
                                  ; initially for variables at DEFADD

          POP   HL                ; pop the FN ')' pointer
          LD    ($5B5D),HL        ; set CH_ADD to this
          POP   HL                ; pop the original DEFADD value
          LD    ($5B0B),HL        ; and re-insert into DEFADD system variable.

          JP    S_CONT_4          ;+ back to similar code.

;;;       RST   20H               ; NEXT-CHAR advances to character after ')'
;;;       JP    S_CONT_2          ; jump back to S-CONT-2

; ---------------------------------
; THE 'DEF FN SKIPOVER' SUBROUTINE
; ---------------------------------
;   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
;   that 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 collate to less than
;   the space character.

FN_SKPOVR INC   HL                ; increase pointer.
          LD    A,(HL)            ; fetch the addressed character.

          CP    $21               ; compare with space + 1
          JR    C,FN_SKPOVR       ; back to FN-SKPOVR if space or less.

          RET                     ; return pointing to a significant character.

; ---------------------------------------
; THE 'SEARCH VARIABLES AREA' SUBROUTINE
; ---------------------------------------
;
;

LOOK_VARS SET   6,(IY+$01)        ; update FLAGS - presume numeric result

          RST   18H               ; GET-CHAR

          CALL  ALPHA             ; routine ALPHA tests for [A-Za-z]

          JP    NC,REPORT_C       ; jump back, if not, to REPORT-C
                                  ; 'Nonsense in BASIC'

;   The first character in BASIC is alphabetic

          PUSH  HL                ; save pointer to first character    ^1
          AND   $1F               ; mask lower bits, 1 - 26 decimal     000xxxxx
          LD    C,A               ; store in C as descriptor.

          RST   20H               ; NEXT-CHAR points to second character.
          PUSH  HL                ; save pointer to second character   ^2
          CP    $28               ; is it '(' - an array ?
          JR    Z,V_RUN_SYN       ; forward, if so, to V-RUN/SYN. with  000xxxxx

          SET   6,C               ; preset bit 6 signaling string       010xxxxx
          CP    $24               ; is character a '$' ?
          JR    Z,V_STR_VAR     ; forward, if so, to V-STR-VAR

          SET   5,C             ; signal simple numeric               011xxxxx

          CALL  ALPHANUM        ; routine ALPHANUM sets carry if second
                                ; character is also alphanumeric.

          JR    NC,V_TEST_FN    ; forward to V-TEST-FN if just one character

;   It is more than one character but re-test current character so that 6 reset
;   Subsequent characters have the character reduced to 1-26 or 33-58 if lower
;   case. Deceptively clever.

V_CHAR    CALL  ALPHANUM        ; routine ALPHANUM
          JR    NC,V_RUN_SYN    ; to V_RUN_SYN when no more

          RES   6,C             ; make long named type                001

          RST   20H             ; NEXT-CHAR
          JR    V_CHAR          ; loop back to V-CHAR

; ---

;   The jump was here when second character was '$'.

V_STR_VAR RST   20H             ; NEXT-CHAR advances past '$'
;;;       RES   6,(IY+$01)      ; update FLAGS - signal string result.
          CALL  STR_RSLT        ;+

V_TEST_FN LD    A,($5B0C)       ; load A with DEFADD_hi
          AND   A               ; and test for zero.
          JR    Z,V_RUN_SYN     ; forward to V_RUN_SYN if a defined function
                                ; is not being evaluated.

;   Note.

          CALL  SYNTAX_Z        ; routine SYNTAX-Z

          JR    NZ,STK_F_ARG    ; JUMP to STK-F-ARG in runtime and then
                                ; back to this point if no variable found.

;   All paths converge here with bits 5 and 6 describing variable.

V_RUN_SYN LD    B,C             ; save flags in B
          CALL  SYNTAX_Z        ; routine SYNTAX-Z
          JR    NZ,V_RUN        ; 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 back to C.

          JR    V_SYNTAX        ; forward to V-SYNTAX

; ---

;   In runtime search for the variable.

V_RUN     LD    HL,($5B4B)      ; set HL to start of variables from VARS

V_EACH    LD    A,(HL)          ; get first variable letter
```

```
          AND   $7F             ; AND with %01111111
                                ; ignoring bit 7 which distinguishes
                                ; arrays or for/next variables.

          JR    Z,V_80_BYTE     ; forward, if zero, to V-80-BYTE
                                ; as must be 10000000 the variables end-marker.

          CP    C               ; compare with supplied value.
          JR    NZ,V_NEXT       ; forward, with no match, to V-NEXT

          RLA                   ; destructively test
          ADD   A,A             ; bits 5 and 6 of A
                                ; jumping if bit 5 reset or 6 set

          JP    P,V_FOUND_2     ; to V-FOUND-2  strings and arrays

          JR    C,V_FOUND_2     ; to V-FOUND-2  simple and for next

;   This leaves long name variables.  x01xxxxx

          POP   DE              ; pop pointer to BASIC 2nd. character.
          PUSH  DE              ; save it again

          PUSH  HL              ; save variable first letter pointer

V_MATCHES INC   HL              ; address next letter in VARS area

V_SPACES  LD    A,(DE)          ; pick up character from BASIC area
          INC   DE              ; and advance character address
          CP    $20             ; is character a space ?
          JR    Z,V_SPACES      ; back to V-SPACES until non-space

          OR    $20             ; convert character to reduced lower case.33-58
          CP    (HL)            ; compare with addressed variables letter
          JR    Z,V_MATCHES     ; loop back to V-MATCHES if a match on an
                                ; intermediate letter.

;   the last letter won't match.

          OR    $80             ; now set bit 7 as last character of long names
                                ; is inverted.
          CP    (HL)            ; compare again
          JR    NZ,V_GET_PTR    ; 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 BASIC character

          CALL  ALPHANUM        ; routine ALPHANUM sets carry if not alphanum

          JR    NC,V_FOUND_1    ; forward to V-FOUND-1 with a full match.

V_GET_PTR POP   HL              ; pop saved pointer to 1st BASIC character.

V_NEXT    PUSH  BC              ; save flags

          CALL  NEXT_ONE        ; routine NEXT-ONE gets next variable in DE

          EX    DE,HL           ; transfer VARS address to HL.
          POP   BC              ; restore the flags

          JR    V_EACH          ; loop back to V-EACH
                                ; to compare each variable
; ---

V_80_BYTE SET   7,B             ; signals not found in runtime.

;   the branch was here when checking syntax

V_SYNTAX  POP   DE              ; discard the pointer to 2nd. character  v2
                                ; in BASIC line/workspace.
                                ; Note HL addresses 2nd BASIC character also

          RST   18H             ; GET-CHAR gets character after variable name.

          CP    $28             ; is it '(' ?
                                ; from a string array e.g. a$(

;;;       JR    Z,V_PASS        ; forward, with string array, to V-PASS
;;;                             ; Note. could go straight to V-END ?

          JR    Z,V_END         ;+ forward, with string array, to V-END

          SET   5,B             ; signal not an array
          JR    V_END           ; 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 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 POP   DE              ; discard pointer to 2nd BASIC char       v2
          POP   DE              ; drop pointer to 1st BASIC char          v1
          PUSH  HL              ; save pointer to last letter in VARS

          RST   18H             ; GET-CHAR

;;; V_PASS  CALL  ALPHANUM      ; Routine ALPHANUM
;;;         JR    NC,V_END      ; Forward, if not, to V-END

;   but it never will be as we advanced past long-named variables earlier.

;;;       RST   20H             ; NEXT-CHAR
;;;       JR    V_PASS          ; Back to V-PASS

; ---

V_END     POP   HL              ; Pop the pointer to last or only letter in
                                ; the VARS area.
          RL    B               ; Rotate the B register left, bit 7 to carry.

          BIT   6,B             ; Test the array indicator bit.

          RET                   ; Return.

; -------------------------------------
; THE 'STACK FUNCTION ARGUMENT' SECTION
; -------------------------------------
;   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 LD     HL,($5B0B)       ; set HL to DEFADD
          LD     A,(HL)           ; load the first character
          CP     $29              ; is it ')' ?
SFA_VRSYN JR     Z,V_RUN_SYN      ; 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  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,SFA_CP_VR      ; forward, if so, to SFA-CP-VR

;   it was a string. White-space may be present but syntax has been checked.

          DEC    HL               ; point back to letter.
          CALL   FN_SKPOVR        ; routine FN-SKPOVR skips to the '$'
          INC    HL               ; now address the hidden marker.
          RES    5,B              ; signal a string variable.

SFA_CP_VR LD     A,B              ; transfer found variable letter to A.
          CP     C                ; compare with expected.
          JR     Z,SFA_MATCH      ; forward to SFA-MATCH with a match.

;;;       INC    HL               ; step
;;;       INC    HL               ; past
;;;       INC    HL               ; the
;;;       INC    HL               ; five
;;;       INC    HL               ; bytes.

          CALL   NUMBER_5         ;+ new entry point to increment HL by 5.

          CALL   FN_SKPOVR        ; routine FN-SKPOVR skips to next character
          CP     $29              ; is it ')' ?
          JR     Z,SFA_VRSYN      ; jump back, if so, to V-RUN/SYN
                                  ; to look in the normal variables area.

          CALL   FN_SKPOVR        ; routine FN-SKPOVR skips past the ','
                                  ; all syntax has been checked and these
                                  ; things can be taken as read.

          JR     SFA_LOOP         ; back, until bracket encountered, to SFA-LOOP

; ---

SFA_MATCH BIT    5,C              ; test if numeric
          JR     NZ,SFA_END       ; forward, if so, to SFA-END
                                  ; as will be stacked by scanning.

          INC    HL               ; point to start of string descriptor

;;;       LD     DE,($5B65)       ; set DE to STKEND
;;;       CALL   MOVE_FP          ; routine MOVE-FP puts parameters on stack.
;;;       EX     DE,HL            ; new free location to HL.
;;;       LD     ($5B65),HL       ; use it to set STKEND system variable.

          CALL   STACK_NUM        ;+ subroutine embodies 3 of above instructions

          EX     DE,HL            ;+ HL must address STKEND
```

```
SFA_END   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   XOR   A               ; clear A
          LD    B,A             ; and B, the syntax dimension counter (256)
          BIT   7,C             ; checking syntax ?
          JR    NZ,SV_COUNT     ; forward, if so, to SV-COUNT

;   runtime evaluation.

          BIT   7,(HL)          ; will be reset if a simple string.
          JR    NZ,SV_ARRAYS    ; forward to SV-ARRAYS otherwise

          INC   A               ; set A to 1, simple string.

SV_SIMPLE 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  STK_STO_s       ; routine STK-STO-$ stacks string parameters
                                ; DE start in variables area,
                                ; BC length, A=1 indicates a 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    SV_SLICEq       ; jump forward to SV-SLICE? to test for '('

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

;   the branch was here with string and numeric arrays in runtime.

SV_ARRAYS 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,SV_PTR        ; forward to SV-PTR with numeric arrays

          DEC   B               ; ignore the final element of a string array
                                ; the fixed string size.

          JR    Z,SV_SIMPLE     ; 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,REPORT_3     ; forward, if not, to REPORT-3
                                ; 'Subscript wrong'

          EX    DE,HL           ; dimensions pointer to HL to synchronize
                                ; with next instruction.

;   runtime numeric arrays path rejoins here.

SV_PTR    EX    DE,HL           ; save dimension pointer in DE
          JR    SV_COUNT        ; 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  PUSH  HL              ; save counter

          RST   18H             ; GET-CHAR

          POP   HL              ; pop counter
          CP    $2C             ; is character ',' ?
          JR    Z,SV_LOOP       ; forward, if so, to SV-LOOP

;   in runtime the variable definition indicates a comma should appear here

          BIT   7,C             ; checking syntax ?
          JR    Z,REPORT_3      ; forward, if not, to REPORT-3
                                ; 'Subscript wrong'

;   proceed if checking syntax of an array?

          BIT   6,C             ; array of strings ?
          JR    NZ,SV_CLOSE     ; forward, if so, to SV-CLOSE

;   an array of numbers.
;;;       CP    $29             ; is character ')' ?     XXXXX
;;;       JR    NZ,SV_RPT_C     ; forward, if not, to SV-RPT-C

RBRKT_NXT CALL  TST_RBRKT       ;+ test for a right hand bracket.

          RST   20H             ; NEXT-CHAR moves CH-ADD past the statement
          RET                   ; return ->

; ---

;   the branch was here with an array of strings.

SV_CLOSE  CP    $29             ; as above ')' could follow the expression
          JR    Z,SV_DIM        ; forward, if so, to SV-DIM

          CP    $CC             ; is it 'TO' ?
          JR    NZ,SV_RPT_C     ; 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 RST   18H             ; GET-CHAR
          DEC   HL              ; backtrack HL
          LD    ($5B5D),HL      ; to set CH_ADD up for slicing routine
          JR    SV_SLICE        ; forward to SV-SLICE and make a return
                                ; when all slicing complete.

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

;   -> the mid-point entry point of the loop

SV_COUNT  LD    HL,$0000        ; initialize data pointer to zero.

SV_LOOP   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,SV_MULT      ; 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,SV_DIM        ; forward to SV-DIM to consider further slice

          CP    $CC             ; is it 'TO' ?
          JR    Z,SV_CH_ADD     ; 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   PUSH  BC              ; save dimension number.
          PUSH  HL              ; push data pointer/rubbish.
                                ; DE points to current dimension.
          EX    DE,HL           ;
          INC   HL              ;
          LD    E,(HL)          ;
          INC   HL              ;
          LD    D,(HL)          ;

;;;       CALL  DEDEplus1       ; 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  INT_EXP1        ; routine INT-EXP1 checks integer expression
                                ; and gets result in BC in runtime.
          JR    C,REPORT_3      ; to REPORT-3 if > HL
                                ; 'Subscript wrong'

          DEC   BC              ; adjust returned result from 1-x to 0-x
          CALL  GET_HLxDE       ; 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  SV_COMMA        ; 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  JP    NZ,REPORT_Cw    ; forward, if so, to SL-RPT-C
                                ; '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,SV_ELEMs     ; forward, if so, to SV-ELEM$

;   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,SV_NUMBER     ; skip, if so, to SV-NUMBER

;   else more subscripts in BASIC line than the variable definition.

REPORT_3  RST   30H             ; ERROR-1
          DEFB  $02             ; Error Report: Subscript wrong

;   continue if subscripts matched the numeric array.

SV_NUMBER 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  GET_HLxDE       ; 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_ELEMs
          EX    DE,HL           ;
          INC   HL              ;
          LD    E,(HL)          ;
          INC   HL              ;
          LD    D,(HL)          ;

;;;       CALL  DEDEplus1       ; routine DE,(DE+1) gets next dimension in DE
                                ; the length of strings in this array.
          EX    (SP),HL         ; start pointer to stack, data pointer to HL.
          CALL  GET_HLxDE       ; 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  STK_ST_0        ; routine STK-ST-0 stores the string parameters
                                ; with A=0 indicating 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,SV_DIM        ; 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,REPORT_3     ; to REPORT-3 with anything else
                                ; 'Subscript wrong'

SV_SLICE  CALL  SLICING         ; routine SLICING slices the string.

;   but a slice of a simple string can itself be sliced.

SV_DIM    RST   20H             ; NEXT-CHAR

SV_SLICEq CP    $28             ; is character '(' ?
          JR    Z,SV_SLICE      ; loop back if so to SV-SLICE

STR_RSLT  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)



; -------------------------------
; THE 'STRING SLICING' SUBROUTINE
; -------------------------------
;   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   CALL  SYNTAX_Z         ; routine SYNTAX-Z

          CALL  NZ,STK_FETCH     ; 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,SL_STORE       ; 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,SL_SECOND      ; to SL-SECOND to evaluate second parameter.

          POP   AF               ; pop the running flag.

          CALL  INT_EXP2         ; 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,SL_SECOND      ; to SL-SECOND to evaluate second parameter
```

```
;;;       CP    $29             ; is it ')' ?       e.g. a$(365)
;;; SL_RPT_C JP    NZ,REPORT_C  ; jump to REPORT-C with anything else

          CALL  TST_RBRKT       ;+ test for a right-hand bracket.

          LD    H,D             ; copy start
          LD    L,E             ; to end - just a one character slice.
          JR    SL_DEFINE       ; forward to SL-DEFINE.

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

SL_SECOND 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,SL_DEFINE     ; to SL-DEFINE using length as end point.

          POP   AF              ; else restore flag.

          CALL  INT_EXP2        ; 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,SL_RPT_C     ; back, if not, to SL-RPT-C

          CALL  TST_RBRKT       ;+ Test for a right-hand bracket.

;   we now have start in DE and an end in HL.

SL_DEFINE 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,SL_OVER       ; forward to SL-OVER if start > end.

          INC   HL              ; increment the length for inclusive byte.

          AND   A               ; now test the running flag.
          JP    M,REPORT_3      ; jump back to REPORT-3 if $FF.
                                ; 'Subscript wrong'

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

SL_OVER   POP   DE              ; restore start address from machine stack ***
;;;       RES   6,(IY+$01)      ; update FLAGS - signal string result for the
;;;                             ; syntax path.
          CALL  STR_RSLT        ;+

;;; SL_STORE  CALL  SYNTAX_Z    ; routine SYNTAX_Z  (UNSTACK_Z?)
;;;           RET   Z           ; return if checking syntax.

SL_STORE  CALL  UNSTACK_Z       ;+ return early if checking syntax.

;   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  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_s RES   6,(IY+$01)  ; update FLAGS - signal string result.

STK_STO_s CALL  STR_RSLT        ;+
                                ; and continue to store parameters of string.

; ---------------------------
; THE 'STACK STORE' SUBROUTINE
; ---------------------------
;   This subroutine puts five registers AEDCB on the calculator stack.

STK_STORE PUSH  BC              ; preserve two registers

          CALL  TEST_5_SP       ; routine TEST-5-SP checks room

          POP   BC              ; fetch the saved registers.

          LD    HL,($5B65)      ; 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    ($5B65),HL      ; update system variable STKEND.

          RET                   ; and return.

; -------------------------------------------
; THE 'INTEGER EXPRESSION EVALUATION' ROUTINE
; -------------------------------------------
;   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  XOR   A               ; set result flag to zero.

;   -> The entry point is here if A is used as a running flag.
```

```
INT_EXP2  PUSH  DE                ; preserve DE register throughout.
          PUSH  HL                ; save the supplied limit.

          PUSH  AF                ; save the flag.

          CALL  EXPT_1NUM         ; routine EXPT-1NUM evaluates expression
                                  ; at CH_ADD returning if numeric result,
                                  ; with value on calculator stack.

          POP   AF                ; pop the flag.

          CALL  SYNTAX_Z          ; routine SYNTAX-Z
          JR    Z,I_RESTORE       ; forward, if checking syntax to I-RESTORE
                                  ; so avoiding a comparison with supplied limit.

;   The runtime path.

          PUSH  AF                ; save the flag.

          CALL  FIND_INT2         ; 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,I_CARRY         ; forward, if zero, to I-CARRY

          POP   HL                ; restore the limit.
          PUSH  HL                ; and save.
          AND   A                 ; prepare to subtract.
          SBC   HL,BC             ; subtract value from limit.

I_CARRY   LD    A,D               ; move flag to accumulator $00 or $FF.
          SBC   A,$00             ; will set to $FF if carry set.

I_RESTORE 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. Note. this will probably have to go.

; DEDEplus1 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.
;   Note. UNSTACK_Z can't be used at start as HL would be corrupted :-)

GET_HLxDE CALL  SYNTAX_Z        ; routine SYNTAX-Z.
          RET   Z               ; return if checking syntax.

          CALL  HL_HLxDE        ; routine HL-HL*DE.

          JP    C,REPORT_4      ; jump back to REPORT-4 if over 65535.
                                ; 'Out of memory'

          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       LD    HL,($5B4D)      ; fetch system variable DEST to HL.

          BIT   1,(IY+$37)      ; test FLAGX - handling a new variable ?
          JR    Z,L_EXISTS      ; forward, if not, to L-EXISTS

;   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 INC   BC              ; increase byte count for each relevant
                                ; character

L_NO_SP   INC   HL              ; increase pointer.
          LD    A,(HL)          ; fetch character.
          CP    $20             ; is it a space ?
          JR    Z,L_NO_SP       ; back to L-NO-SP is so.

          JR    NC,L_TEST_CH    ; forward to L-TEST-CH if higher.

          CP    $10             ; is it $00 - $0F ?
          JR    C,L_SPACES      ; forward, if so, to L-SPACES

          CP    $16             ; is it $16 - $1F ?
          JR    NC,L_SPACES     ; forward, if so, to L-SPACES

;   it was $10 - $15  so step over a colour code.
```

```
          INC   HL              ; increase pointer.
          JR    L_NO_SP         ; loop back to L-NO-SP.

; ---

;   the branch was to here if higher than space.

L_TEST_CH CALL  ALPHANUM        ; routine ALPHANUM sets carry if alphanumeric
          JR    C,L_EACH_CH     ; loop back, if so, for more to L-EACH-CH

          CP    $24             ; is it '$' ?
          JP    Z,L_NEWs        ; jump forward if so, to L-NEW$
                                ; with a new string.

L_SPACES  LD    A,C             ; save length lo in A.

          CALL  MK_RM_EL        ;+ MAKE_ROOM at E_LINE -1

;;;       LD    HL,($5B59)      ; fetch E_LINE to HL.
;;;       DEC   HL              ; point to location before, the variables
;;;       CALL  MAKE_ROOM       ; routine MAKE-ROOM creates BC spaces
;;;       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,($5B4D)      ; 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,L_SINGLE      ; forward to L-SINGLE if it was just
                                ; one character.

;   Register HL points to start of variable name after 'LET' in BASIC line.

L_CHAR    INC   HL              ; increase pointer.
          LD    A,(HL)          ; pick up character.
          CP    $21             ; is it space or higher ?
          JR    C,L_CHAR        ; 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  L_CHAR          ; 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  LD    HL,($5B4D)      ; 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  L_FIRST         ; routine L-FIRST puts A in first character.
                                ; and returns with HL holding
                                ; new E_LINE-1  the $80 vars end-marker.
```

```
L_NUMERIC 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

;   Register 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    L_ENTER         ; forward to L-ENTER  ==>

; ---


;   the jump was to here if the variable already existed.

L_EXISTS  BIT   6,(IY+$01)      ; test FLAGS - numeric or string result ?
          JR    Z,L_DELETEs     ; skip forward to L-DELETE$   -*->
                                ; 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    L_NUMERIC       ; back to L-NUMERIC to overwrite value.

; ---

;   -*-> the branch was here if a string existed.

;;; L_DELETEs LD    HL,($5B4D)  ; fetch DEST to HL.
                                ; (still set from first instruction)

L_DELETEs LD    BC,($5B72)      ; fetch STRLEN to BC.
          BIT   0,(IY+$37)      ; test FLAGX - handling a complete simple
                                ; string ?
          JR    NZ,L_ADDs       ; forward, if so, to L-ADD$

;   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 travelers 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 zero.

          PUSH  HL              ; save pointer to start.

          CALL  BC_SPACES       ; 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                  ; block copy bytes filling area with spaces.

          PUSH  HL              ; save pointer to start.

          CALL  STK_FETCH       ; 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,L_LENGTH     ; forward if it fits to L-LENGTH.

          LD    B,H             ; otherwise set
          LD    C,L             ; length to old length.
                                ; "hatstand" becomes "hats"

L_LENGTH  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,L_IN_W_S      ; forward, if so, to L-IN-W/S
;;;                             ; leaving the prepared spaces.
;;;       LDIR                  ; else copy bytes overwriting some spaces.

          CALL  COND_MV         ;+ a Conditional (NZ) ldir routine

L_IN_W_S  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   EX    DE,HL           ; exchange pointers HL=STKEND DE=end of vars.

COND_MV   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                  ; block copy bytes.

          POP   HL              ; address the start.
          RET                   ; Return.

; ---

;   the branch was here from L-DELETE$ if an existing simple string.
;   register HL addresses start of string in variables area.

L_ADDs    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  L_STRING        ; 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    RECLAIM_2       ; jump to indirect exit via RECLAIM-2
                                ; deleting old version and adjusting pointers.

; ---

;   the jump was here with a new string variable.

L_NEWs    LD    A,$DF           ; indicator mask %11011111 for
                                ;                %010xxxxx will be result
          LD    HL,($5B4D)      ; address DEST first character.
          AND   (HL)            ; combine mask with character.

L_STRING  PUSH  AF              ; save first character and mask.

          CALL  STK_FETCH       ; routine STK-FETCH fetches parameters of
                                ; the string. Start in DE, length in BC.

          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    ($5B4D),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.

          CALL  MK_RM_EL        ;+ MAKE_ROOM at E_LINE -1

;;;       LD    HL,($5B59)      ; address E_LINE.
;;;       DEC   HL              ; now end of VARS area.
;;;       CALL  MAKE_ROOM       ; routine MAKE-ROOM makes room for string.
;;;                             ; updating pointers including DEST.

          LD    HL,($5B4D)      ; 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   DEC   HL              ; address variable name
```

```
          LD    (HL),A          ; and insert character.

L_EL_DHL  LD    HL,($5B59)      ; load HL with E_LINE.
          DEC   HL              ; now end of VARS area.
          RET                   ; return

; ---------------------------
; THE 'STACK FETCH' SUBROUTINE
; ---------------------------
;
;

STK_FETCH LD    HL,($5B65)      ; 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    ($5B65),HL      ; STKEND
          RET                   ;

; -----------------
; THE '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       CALL  LOOK_VARS       ; routine LOOK-VARS

D_RPORT_C JP    NZ,REPORT_C     ; jump to REPORT-C if a long-name variable.
                                ; DIM lottery numbers(49) doesn't work.

          CALL  SYNTAX_Z        ; routine SYNTAX-Z
          JR    NZ,D_RUN        ; forward, in runtime, to D-RUN

          RES   6,C             ; signal 'numeric' array even if of type string
                                ; as this simplifies the syntax checking.

          CALL  STK_VAR         ; routine STK-VAR checks syntax.
          CALL  CHECK_END       ; routine CHECK-END performs early exit ->

; ---

;   the branch was here in runtime.

D_RUN     JR    C,D_LETTER      ; skip to D-LETTER if variable did not exist.
                                ; else reclaim the old one.

          PUSH  BC              ; save type in C.

;;;       CALL  NEXT_ONE        ; routine NEXT-ONE find following variable
;;;                             ; or position of $80 end-marker.
;;;       CALL  RECLAIM_2       ; routine RECLAIM-2 reclaims the
;;;                             ; space between.
```

```
         CALL  NXT_1_RC2       ;+ routine combines above 2 routines.

         POP   BC              ; pop the type.

D_LETTER 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,D_SIZE       ; forward, if so, to D-SIZE

         LD    L,$05           ; make elements 5 bytes as is numeric.

D_SIZE   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 RST  20H             ; NEXT-CHAR
         LD    H,$FF           ; disable limit check by setting HL high

         CALL  INT_EXP1        ; routine INT-EXP1

         JP    C,REPORT_3      ; to REPORT-3 if > 65280 and then some
                               ; 'Subscript wrong'

         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  GET_HLxDE       ; 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,D_NO_LOOP     ; loop back to D-NO-LOOP until all dimensions
                               ; have been considered

;   when loop complete continue.

;;;      CP    $29             ; is it ')' ?
;;;      JR    NZ,D_RPORT_C    ; to D-RPORT-C with anything else
;;;      RST   20H             ; NEXT-CHAR advances to next statement/CR

         CALL  RBRKT_NXT       ;+ Test for a right hand bracket and advance.

         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,REPORT_4      ; 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

          CALL  MK_RM_EL        ;+ MAKE_ROOM at E_LINE -1

;;;       LD    HL,($5B59)      ; address E_LINE - first location after
;;;       DEC   HL              ; point to location before - the $80 end-marker
;;;       CALL  MAKE_ROOM       ; routine MAKE-ROOM creates the space if
;;;       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,DIM_CLEAR     ; skip to DIM-CLEAR if numeric

          LD    (HL),$20        ; place a space character in HL

DIM_CLEAR 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 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,DIM_SIZES    ; back to DIM-SIZES until all done.

          RET                   ; return.

; -------------------------
; THE 'ALPHANUM' SUBROUTINE
; -------------------------
;   This routine checks that the character in A is alphanumeric returning,
;   if so, with carry set.

ALPHANUM  CALL  NUMERIC         ; Routine NUMERIC resets carry if a number.

          CCF                   ; Complement Carry Flag.
          RET   C               ; Return if numeric else continue into next
                                ; routine.

;   This routine checks that the character in A is alphabetic setting the carry
;   flag if it is.

ALPHA     CP    $41             ; less than 'A' ?
          CCF                   ; Complement Carry Flag
          RET   NC              ; return if less.

          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 less.

          CP    $7B             ; less than 'z'+1 ?
          RET                   ; carry set if within a-z.

; ------------------------------------------
; THE 'DECIMAL TO FLOATING POINT' SUBROUTINE
; ------------------------------------------
;   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 CP    $C4             ; 'BIN' token ?
          JR    NZ,NOT_BIN      ; forward, if not, to NOT-BIN

          LD    DE,$0000        ; initialize 16 bit buffer register.

BIN_DIGIT 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,BIN_END      ; 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,REPORT_6      ; 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    BIN_DIGIT       ; back to BIN-DIGIT for more digits

; ---

BIN_END   LD    B,D             ; transfer 16 bit buffer
          LD    C,E             ; to BC register pair.
          JR    STACK_BC        ; JUMP to STACK-BC to put on calculator stack
```

```
; ---

;   continue here with .1,  42, 3.14, 5., 2.3 E -4

NOT_BIN   CP    $2E             ; '.' - leading decimal point ?
          JR    Z,DECIMAL       ; skip, if so, to DECIMAL

          CALL  INT_TO_FP       ; routine INT-TO-FP to evaluate all digits
                                ; This number 'x' is placed on stack.
          CP    $2E             ; '.' - mid decimal point ?

          JR    NZ,E_FORMAT     ; to E-FORMAT if not to consider that format

          RST   20H             ; NEXT-CHAR
          CALL  NUMERIC         ; routine NUMERIC returns carry reset if 0-9

          JR    C,E_FORMAT      ; to E-FORMAT if not a digit e.g. '1.'

          JR    DEC_STO_1       ; to DEC-STO-1 to add the decimal part to 'x'

; ---

;   a leading decimal point has been found in a number.

DECIMAL   RST   20H             ; NEXT-CHAR
          CALL  NUMERIC         ; routine NUMERIC will reset carry if digit

DEC_RPT_C JP    C,REPORT_C      ; 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 will be stacked.
;   Now store 1 in mem-0.
;   Note. At each pass of the digit loop this will be divided by ten.

DEC_STO_1 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 RST   18H             ; GET-CHAR

          CALL  STK_DIGIT       ; routine STK-DIGIT stacks single digit 'd'

          JR    C,E_FORMAT      ; exit to E-FORMAT when digits exhausted  >

;   Note. by switching division and multiply .5 will evaluate as 5/10 instead
;   of 5 * .1. The values 5/10 and 1/2 are therefore equal.

          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   ;obsolete
          DEFB  $04             ;+multiply   ;x or 0,d,10.
          DEFB  $C0             ;;st-mem-0   ;x or 0,d,10.
;;;       DEFB  $04             ;;multiply   ;obsolete
```

```
        DEFB  $05             ;+division   ;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    NXT_DGT_1       ; 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 divided by an increasingly larger
;   factor (100, 1000, 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  CP    $45             ; is character 'E' ?
          JR    Z,SIGN_FLAG     ; forward, if so, to SIGN-FLAG

          CP    $65             ; 'e' is acceptable as well.
          RET   NZ              ; return as no exponent.

SIGN_FLAG LD    B,$FF           ; initialize temporary sign byte to $FF

          RST   20H             ; NEXT-CHAR
          CP    $2B             ; is character '+' ?
          JR    Z,SIGN_DONE     ; to SIGN-DONE

          CP    $2D             ; is character '-' ?
          JR    NZ,ST_E_PART    ; 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 RST   20H             ; NEXT-CHAR

ST_E_PART CALL  NUMERIC         ; routine NUMERIC

          JR    C,DEC_RPT_C     ; back, if not, to DEC-RPT-C
                                ;  'Nonsense in BASIC'.

          PUSH  BC              ; save sign (in B)

          CALL  INT_TO_FP       ; routine INT-TO-FP places exponent on stack

          CALL  FP_TO_A         ; routine FP-TO-A  transfers it to A

          POP   BC              ; restore sign
          JP    C,REPORT_6      ; to REPORT-6 if overflow (over 255)
                                ; raise 'Number too big'.

          AND   A               ; set flags
          JP    M,REPORT_6      ; 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,E_FP_JUMP     ; forward to E-FP-JUMP if exponent positive

          NEG                   ; Negate the exponent.

E_FP_JUMP JR    E_TO_FP         ; 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.

; -----------------------
; THE 'NUMERIC' SUBROUTINE
; -----------------------
;   This routine checks that the ASCII character in A is numeric
;   returning, if so, with carry reset.

NUMERIC   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'

; ---------------------------------
; THE 'STACK BC and SET IY' ROUTINE
; ---------------------------------
;

STK_BC_IY LD    IY,$5B3A        ;+ re-initialize the IY register to access the
                                ;+ system variables. (14 clock cycles)
          JR    STACK_BC        ;+ forward to stack the result of USR function.

; ---------------------------
; THE 'STACK DIGIT' SUBROUTINE
; ---------------------------
;   This subroutine is called from INT-TO-FP and DEC-TO-FP to stack a digit
;   on the calculator stack.

STK_DIGIT CALL  NUMERIC         ; routine NUMERIC
          RET   C               ; return if not numeric character

          SUB   $30             ; convert from ASCII to digit

; -----------------------
; THE 'STACK A' SUBROUTINE
; -----------------------
;
;

STACK_A   LD    C,A             ; transfer to C
          LD    B,$00           ; and make B zero

; ------------------------
; THE 'STACK BC' SUBROUTINE
; ------------------------
;

;;; STACK_BC  LD    IY,$5B3A    ; re-initialize ERR_NR

STACK_BC  XOR   A               ; Clear accumulator to signal small integer
          LD    E,A             ; Place in E for the sign byte.
          LD    D,C             ; LSB to D
          LD    C,B             ; MSB to C
          LD    B,A             ; last byte not used

          CALL  STK_STORE       ; routine STK-STORE stacks number AEDCB
```

```
                                ; and sets carry.

;   Note. HL now points to new STKEND. The requirement is that it should point
;   to the 'result' and DE should point at STKEND as this is the terminating
;   routine for some calculator functions.  This can be done by simply entering
;   and leaving the calculator but that uses many clock cycles if only two
;   bytes.

          AND   A               ;+ Clear carry.
          JP    STK_PNTRS       ;+ set HL to result and DE to STKEND also
                                ;+ the carry flag is unaffected.

;;;       RST   28H             ;; FP-CALC
;;;       DEFB  $38             ;;end-calc  make HL = STKEND-5 and DE = STKEND
;;;       AND   A               ; clear the carry flag.
;;;       RET                   ; Return.

; ------------------------------------------
; THE 'INTEGER TO FLOATING POINT' SUBROUTINE
; ------------------------------------------
;   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 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 CALL  STK_DIGIT       ; 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  CH_ADD__1       ; routine CH-ADD+1 get next character
          JR    NXT_DGT_2       ; back to NXT-DGT-2 to process as a digit


;*********************************
;** Part 9. ARITHMETIC ROUTINES **
;*********************************

; -------------------------------------------
; THE 'E-FORMAT TO FLOATING POINT' SUBROUTINE
; -------------------------------------------
;   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   RLCA                  ; this will set the          x.
```

```
          RRCA                  ; carry if bit 7 is set

          JR    NC,E_SAVE       ; to E-SAVE  if positive.

          CPL                   ; make negative positive
          INC   A               ; without altering carry.

E_SAVE    PUSH  AF              ; save positive exp and sign in carry

          LD    HL,$5B92        ; address MEM-0

          CALL  FP_0_1          ; 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    SRL   A               ; 0>76543210>C

          JR    NC,E_TST_END    ; 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  E_DIVSN - $     ;;to E-DIVSN

          DEFB  $04             ;;multiply                    x*10.
          DEFB  $33             ;;jump

          DEFB  E_FETCH - $     ;;to E-FETCH

E_DIVSN   DEFB  $05             ;;division                    x/10.

E_FETCH   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 JR    Z,E_END         ; 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    E_LOOP          ; 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     RST   28H            ;; FP-CALC                     final x, factor.
          DEFB  $02            ;;delete                       final x.
          DEFB  $38            ;;end-calc                     x.

          RET                  ; return




; -----------------------------
; THE 'FETCH INTEGER' SUBROUTINE
; -----------------------------
;   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.  Register
;   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.  The contents of the B register are not affected.

INT_FETCH 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

; -------------------------------------
; THE 'Store a positive integer' ROUTINE
; -------------------------------------
;;; This entry point is not used in this ROM but would store any integer as
;;; a positive number.

;;; p-int-sto
;;; L2D8C:    LD    C,$00           ; make sign byte positive and continue

; -----------------------------
; THE 'STORE INTEGER' SUBROUTINE
; -----------------------------
;   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 PUSH  HL             ; Preserve HL throughout.

;;;       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        ; The last byte always zero for integers.
          XOR   A               ;+ Set A to zero.
          LD    (HL),A          ;+ Make fifth byte zero.
          POP   HL              ; Restore the original HL result pointer.
          LD    (HL),A          ;+ Make first byte zero.
          RET                   ; Return.


; -------------------------------------------
; THE 'FLOATING POINT TO BC REGISTER' ROUTINE
; -------------------------------------------
;   This routine gets a floating point number e.g. 127.4 from the calculator
;   stack to the BC register.
;   Begin by using two bytes of instruction to make HL address the last 5-byte
;   number on the calculator stack.
;   Note. at the expense of one byte a call to STK_PNTRS would be quicker.

FP_TO_BC
;;;       RST   28H             ;; FP-CALC            set HL to
;;;       DEFB  $38             ;;end-calc            point to 'last value'.

          CALL  STK_PNTRS       ;+ set HL to STKEND -5

          LD    A,(HL)          ; get first of the 5 bytes
          AND   A               ; and test for zero.
          JR    Z,FP_DELETE     ; forward, if a small integer, to FP-DELETE

;   The floating point 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 DE pointing at integer.

FP_DELETE RST   28H             ;; FP-CALC
          DEFB  $02             ;;delete
          DEFB  $38             ;;end-calc

          PUSH  HL              ; preserve pointer to 'last value'.
          PUSH  DE              ; preserve pointer to STKEND.

          EX    DE,HL           ; make HL point to old exponent/zero indicator
          LD    B,(HL)          ; indicator to B

          CALL  INT_FETCH       ; 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 the carry flag
                              ; if not a small integer.

          BIT   7,C           ; Test a bit of the sign byte setting zero flag
                              ; if integer is positive.

          LD    B,D           ; transfer integer
          LD    C,E           ; to BC
          LD    A,E           ; low byte to A also as a useful return value.

          POP   DE            ; Retrieve pointer to new STKEND
          POP   HL            ; Retrieve pointer to new 'last value'

          RET                 ; Return.

;   if carry is set, then the number was too big to fit into BC.

; ------------
; 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_2powA LD    D,A           ; store a copy of A in D the LSB.
          RLA                 ; test sign bit of A.
          SBC   A,A           ; now $FF if negative or $00 if positive.
          LD    E,A           ; sign byte to E the stack sign byte.
          LD    C,A           ; and also to C the MSB.
          XOR   A             ; clear A to indicate an integer.
          LD    B,A           ; and B the unused fifth byte.

          CALL  STK_STORE     ; 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 the 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

; ----------------------------------
; THE 'FLOATING POINT TO A' SUBROUTINE
; ----------------------------------
;   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   CALL  FP_TO_BC      ; 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,FP_A_END      ; forward  FP-A-END if zero

;   else there has been 8-bit overflow so set the carry flag.

          POP   AF              ; retrieve the value

          SCF                   ; set carry flag to show overflow
          RET                   ; and return.

; ---

FP_A_END  POP   AF              ; restore value and success flag and

          RET                   ; return.

; ----------------------------------------------
; THE 'PRINT A FLOATING POINT NUMBER' SUBROUTINE
; ----------------------------------------------
;   Not a trivial task.
;   Begin by considering whether to print a leading sign for negative numbers.

PRINT_FP  RST   28H             ;; FP-CALC
          DEFB  $31             ;;duplicate
          DEFB  $36             ;;less-0
          DEFB  $00             ;;jump-true

          DEFB  PF_NEGTVE - $   ;;to PF-NEGTVE

          DEFB  $31             ;;duplicate
          DEFB  $37             ;;greater-0
          DEFB  $00             ;;jump-true

          DEFB  PF_POSTVE - $   ;;to 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 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 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   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,PF_LARGE      ; 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  INT_FETCH        ; routine INT-FETCH gets x in DE
                                 ; (but x is not deleted)

          LD    B,$10            ; set B, bit counter, to 16d

          LD    A,D              ; Register A equals D from above call. ;;;

          AND   A                ; test if high byte is zero
          JR    NZ,PF_SAVE       ; 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,PF_SMALL       ; 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   PUSH  DE               ; save the part before decimal point.
          EXX                    ;
          POP   DE               ; and pop in into D'E'
          EXX                    ;
          JR    PF_BITS          ; 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" as the number zero is read as the
;   null string by the concatenate routine.
;   credit: Tony Stratton, 1982.
;   A DEFB $02 - 'delete' is required immediately on using the calculator.

PF_SMALL  RST   28H             ;; FP-CALC       int x = 0.
          DEFB  $02             ;+delete         .
          DEFB  $E2             ;;get-mem-2      x-int x.
          DEFB  $38             ;;end-calc

          LD    A,(HL)          ; fetch exponent of positive fractional number
          SUB   $7E             ; subtract

          CALL  LOG_2powA       ; routine LOG(2^A) calculates leading digits.

          LD    D,A             ; transfer count to D
          LD    A,($5BAC)       ; fetch total MEM-5-1
          SUB   D               ;
          LD    ($5BAC),A       ; MEM-5-1
          LD    A,D             ;

          CALL  E_TO_FP         ; 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  FP_TO_A         ; routine FP-TO-A

          PUSH  HL              ; save HL
          LD    ($5BA1),A       ; MEM-3-1
          DEC   A               ;
          RLA                   ;
          SBC   A,A             ;
          INC   A               ;

          LD    HL,$5BAB        ; 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

          JR    PF_FRACTN       ; 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  SUB   $80             ; make exponent positive
```

```
         CP    $1C              ; compare to 28
         JR    C,PF_MEDIUM      ; to PF-MEDIUM if integer <= 2^27

         CALL  LOG_2powA        ; routine LOG(2^A)
         SUB   $07              ;
         LD    B,A              ;
         LD    HL,$5BAC         ; 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  E_TO_FP          ; routine E-TO-FP

         JR    PF_LOOP          ; back to PF-LOOP

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

PF_MEDIUM EX     DE,HL          ;
         CALL  FETCH_TWO        ; 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   SLA   E               ;  C<xxxxxxxx<0
         RL    D                ;  C<xxxxxxxx<C
         EXX                    ;
         RL    E                ;  C<xxxxxxxx<C
         RL    D                ;  C<xxxxxxxx<C
         EXX                    ;

         LD    HL,$5BAA         ; set HL to mem-4-5th last byte of buffer
         LD    C,$05            ; set byte count to 5 -  10 nibbles

PF_BYTES  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,PF_BYTES      ; back to PF-BYTES until the ten nibbles rolled

         DJNZ  PF_BITS          ; 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,$5BA6         ; address MEM-4-0 byte destination.
          LD    DE,$5BA1         ; 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 RLD                    ; pick up leftmost nibble from (HL)
                                 ;    A           (HL)
                                 ; 0000 0000 < 7654 3210
                                 ; 0000 7654   3210 0000


          JR    NZ,PF_INSERT     ; to PF-INSERT if non-zero value picked up.

          DEC   C                ; test
          INC   C                ; flag
          JR    NZ,PF_TEST_2     ; 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 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 BIT   0,B              ; test if the nibble count is even
          JR    Z,PF_ALL_9       ; skip to PF-ALL-9 if so to deal with the
                                 ; other nibble in the same byte

          INC   HL               ; point, if not, to next source byte.

PF_ALL_9  DJNZ  PF_DIGITS        ; 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,($5BAB)        ; fetch digit count from MEM-5-1st
          SUB   $09              ; subtract 9 - max possible
          JR    C,PF_MORE        ; 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    PF_ROUND         ; forward to PF-ROUND
```

```
                                ; to consider rounding.

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

;   now delete int x from calculator stack and fetch fractional part.

PF_MORE   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 EX    DE,HL           ;
          CALL  FETCH_TWO       ; routine FETCH-TWO
          EXX                   ;
          LD    A,$80           ;
          SUB   L               ;
          LD    L,$00           ;
          SET   7,D             ;
          EXX                   ;
          CALL  SHIFT_FP        ; routine SHIFT-FP

PF_FRN_LP LD    A,(IY+$71)      ; MEM-5-1st
          CP    $08             ;
          JR    C,PF_FR_DGT     ; to PF-FR-DGT

          EXX                   ;
          RL    D               ;
          EXX                   ;
          JR    PF_ROUND        ; to PF-ROUND

; ---

PF_FR_DGT LD    BC,$0200        ;

PF_FR_EXX LD    A,E             ;
          CALL  CA_10xA_C       ; routine CA-10*A+C
          LD    E,A             ;
          LD    A,D             ;
          CALL  CA_10xA_C       ; routine CA-10*A+C
          LD    D,A             ;
          PUSH  BC              ;
          EXX                   ;
          POP   BC              ;
          DJNZ  PF_FR_EXX       ; to PF-FR-EXX

          LD    HL,$5BA1        ; 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    PF_FRN_LP       ; 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  PUSH  AF              ; save A and flags

          LD    HL,$5BA1        ; 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 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,PF_R_BACK     ; skip to PF-R-BACK if ZERO?

          CP    $0A             ; compare to 'ten' - overflow
          CCF                   ; complement carry flag so that set if ten.
          JR    NC,PF_COUNT     ; forward to PF-COUNT with 1 - 9.

PF_R_BACK DJNZ  PF_RND_LP       ; 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.

          INC   B               ; make B hold 1 also.

          LD    (HL),B          ; load first location with digit 1.

          INC   (IY+$72)        ; make MEM-5-2nd hold 1.
                                ; and proceed to initialize total digits to 1.

PF_COUNT  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,($5BAB)      ; set C to MEM-5-1st digit counter.
                                ; set B to MEM-5-2nd leading digit counter.
          LD    HL,$5BA1        ; set HL to start of digits at MEM-3-1
          LD    A,B             ;
          CP    $09             ;
          JR    C,PF_NOT_E      ; to PF-NOT-E

          CP    $FC             ;
          JR    C,PF_E_FRMT     ; to PF-E-FRMT

PF_NOT_E  AND   A               ; test for zero leading digits as in .123

          CALL  Z,OUT_CODE      ; routine OUT-CODE prints a zero e.g. 0.123

PF_E_SBRN XOR   A               ;
          SUB   B               ;
          JP    M,PF_OUT_LP     ; skip forward to PF-OUT-LP if originally +ve
          LD    B,A             ; else negative count now +ve
          JR    PF_DC_OUT       ; forward to PF-DC-OUT       ->

; ---

PF_OUT_LP LD    A,C             ; fetch total digit count
          AND   A               ; test for zero
          JR    Z,PF_OUT_DT     ; forward, if so, to PF-OUT-DT

          LD    A,(HL)          ; fetch digit
          INC   HL              ; address next digit
          DEC   C               ; decrease total digit counter

PF_OUT_DT CALL  OUT_CODE        ; routine OUT-CODE outputs it.
          DJNZ  PF_OUT_LP       ; loop back to PF-OUT-LP until B leading
                                ; digits output.

PF_DC_OUT 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_0S RST   10H             ; PRINT-A outputs the character '.' or '0'

          LD    A,$30           ; prepare the character '0'
                                ; (for cases like .000012345678)
          DJNZ  PF_DEC_0S       ; loop back to PF-DEC-0$ for B times.

          LD    B,C             ; load B with now trailing digit counter.
          JR    PF_OUT_LP       ; back to PF-OUT-LP

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

;   the branch was here for E-format printing e.g. 123456789 => 1.2345679e+8

PF_E_FRMT LD    D,B             ; counter to D
          DEC   D               ; decrement
          LD    B,$01           ; load B with 1.

          CALL  PF_E_SBRN       ; 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,PF_E_POS      ; to PF-E-POS if positive

          NEG                   ; negate
          LD    C,A             ; positive exponent to C
          LD    A,$2D           ; prepare character '-'
          JR    PF_E_SIGN       ; skip to PF-E-SIGN

; ---

PF_E_POS  LD    A,$2B           ; prepare character '+'

PF_E_SIGN RST   10H             ; PRINT-A outputs the sign
```

```
;;;       LD    B,$00           ; make the high byte zero.

          JP    OUT_NUM_0       ;+ exit via OUT-NUM-0 to print exponent in BC

; --------------------------------
; THE 'CA = 10 x A + C' SUBROUTINE
; --------------------------------
;   This subroutine is called twice from PRINT_FP when printing floating-point
;   numbers. It returns 10 * A + C in registers C and A (16 bytes)

CA_10xA_C 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.

; -------------------------------
; THE 'PREPARE TO ADD' SUBROUTINE
; -------------------------------
;   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  LD    A,(HL)          ; pick up exponent
          LD    (HL),$00        ; make location zero
          AND   A               ; test if number is zero
          RET   Z               ; return if zero.

          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  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  NEG_BYTE        ; loop to NEG-BYTE till all 5 done

          LD    A,C             ; stored exponent to A
          POP   BC              ; restore original BC
          RET                   ; return
```

```
; ------------------------------------
; THE 'FETCH TWO NUMBERS' SUBROUTINE
; ------------------------------------
;   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 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.

; -------------------------
; THE 'SHIFT FP' SUBROUTINE
; -------------------------
;
;

SHIFT_FP  AND   A               ;
          RET   Z               ;

          CP    $21             ;
          JR    NC,ADDEND_0     ; to ADDEND-0

          PUSH  BC              ;
          LD    B,A             ;

ONE_SHIFT EXX                   ;
          SRA   L               ;
          RR    D               ;
          RR    E               ;
          EXX                   ;
          RR    D               ;
          RR    E               ;
          DJNZ  ONE_SHIFT       ; to ONE-SHIFT

          POP   BC              ;
          RET   NC              ;

          CALL  ADD_BACK        ; routine ADD-BACK
          RET   NZ              ;

ADDEND_0  EXX                   ;
          XOR   A               ;

ZEROS_4_5 LD    L,$00           ;
          LD    D,A             ;
          LD    E,L             ;
          EXX                   ;
          LD    DE,$0000        ;
          RET                   ;

; -------------------------
; THE 'ADD BACK' SUBROUTINE
; -------------------------
;   Called twice to increment D'E'DE as a pseudo 32-bit register.

ADD_BACK  INC   E               ;
          RET   NZ              ;

          INC    D              ;
          RET   NZ              ;

          EXX                   ;
          INC   E               ;
          JR    NZ,ALL_ADDED    ; to ALL-ADDED

          INC   D               ;

ALL_ADDED EXX                   ;
          RET                   ;

; ---------------------------
; THE 'SUBTRACTION' OPERATION
; ---------------------------
; (offset: $03 'subtract')
;   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  EX    DE,HL           ; address second number with HL

          CALL  negate          ; routine NEGATE switches sign

          EX    DE,HL           ; address first number again
                                ; and continue.
; ------------------------
; THE 'ADDITION' OPERATION
; ------------------------
; (offset: $0F 'addition')
;   HL points to first number, DE to second.
;   If they are both integers, then go for the easy route.

addition  LD    A,(DE)          ; fetch first byte of second
          OR    (HL)            ; combine with first byte of first
          JR    NZ,FULL_ADDN    ; forward to FULL-ADDN if at least one was
                                ; in floating point form.

;   Continue if both were both small integers.

          PUSH  DE              ; save pointer to second number for new STKEND.

          INC   HL              ; address sign byte of first number 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

;   Now perform the actual addition.

          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,ADDN_OFLW    ; forward, if not, to ADDN-OFLW

          SBC   A,A             ; restore a negative result sign

; -----------------------------------------------
; THE 'INT -65536 FIX' credit: Dr. Ian Logan, 1983
; -----------------------------------------------
;   Note. the following is a modification of Dr. Ian Logan's suggested fix
;   for the -65536 problem.  At this point, the BC register pair is expendable
;   and this solution is optimized for speed by avoiding the machine stack.
```

```
          LD    C,A             ;+ Make a copy of the sign byte in C.

          INC   A               ;+ Make any $FF in A into $00.
          OR    E               ;+ Test all three
          OR    D               ;+ bytes now for zero.

          LD    A,C             ;+ Restore true sign byte of integer.

          JR    NZ,ADD_STORE    ;+ forward, if not -65536, to ADD_STORE

;   The number, in the registers, is -65536 i.e. 00 FF 00 00 00 and must be
;   made 91 80 00 00 00 on the calculator stack.  At this stage only the
;   fifth byte on the calculator stack is as required.

          DEC   HL              ;+ Point to the first byte.
          LD    (HL),$91        ;+ Enter exponent $91 in first byte.

          INC   HL              ;++ Point to the second byte

          AND   $80             ;++ set A to $80

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

ADD_STORE LD    (HL),A          ; insert second byte
          INC   HL              ;
          LD    (HL),E          ; insert third byte
          INC   HL              ;
          LD    (HL),D          ; insert fourth byte

          DEC   HL              ; back to third.
          DEC   HL              ; back to second.
          DEC   HL              ; point to result.

          POP   DE              ; restore value of STKEND

          RET                   ; Return.

; ---

;   The branch was here when simple register addition overflowed.

ADDN_OFLW DEC   HL              ;
          POP   DE              ;

FULL_ADDN CALL  RE_ST_TWO       ; routine RE-ST-TWO

          EXX                   ;
          PUSH  HL              ;
          EXX                   ;
          PUSH  DE              ;
          PUSH  HL              ;
          CALL  PREP_ADD        ; routine PREP-ADD
          LD    B,A             ;
          EX    DE,HL           ;
          CALL  PREP_ADD        ; routine PREP-ADD
          LD    C,A             ;
          CP    B               ;
          JR    NC,SHIFT_LEN    ; to SHIFT-LEN

          LD    A,B             ;
          LD    B,C             ;
          EX    DE,HL           ;

SHIFT_LEN PUSH  AF              ;
```

```
          SUB   B                 ;
          CALL  FETCH_TWO         ; routine FETCH-TWO
          CALL  SHIFT_FP          ; 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,TEST_NEG       ; to TEST-NEG

          LD    A,$01             ;
          CALL  SHIFT_FP          ; routine SHIFT-FP
          INC   (HL)              ;
          JR    Z,ADD_REP_6       ; to ADD-REP-6

TEST_NEG  EXX                     ;
          LD    A,L               ;
          AND   $80               ;
          EXX                     ;
          INC   HL                ;
          LD    (HL),A            ;
          DEC   HL                ;
          JR    Z,GO_NC_MLT       ; 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,END_COMPL      ; to END-COMPL

          RRA                     ;
          EXX                     ;
          INC   (HL)              ;
```

```
ADD_REP_6 JP    Z,REPORT_6      ; to REPORT-6
                                ; 'Number too big'

          EXX                   ;

END_COMPL LD    D,A             ;
          EXX                   ;

GO_NC_MLT XOR   A               ;
          JP    TEST_NORM       ; to TEST-NORM

; ------------------------------
; THE 'HL = HL * DE' SUBROUTINE
; ------------------------------
;   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_HLxDE  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   ADD   HL,HL           ; double result
          JR    C,HL_END        ; to HL-END if overflow

          RL    C               ; shift AC left into carry
          RLA                   ;
          JR    NC,HL_AGAIN     ; to HL-AGAIN to skip addition if no carry

          ADD   HL,DE           ; add in DE
          JR    C,HL_END        ; to HL-END if overflow

HL_AGAIN  DJNZ  HL_LOOP         ; back to HL-LOOP for all 16 bits

HL_END    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  CALL  TEST_ZERO       ; 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
```

```
; ------------------------------
; THE 'MULTIPLICATION' OPERATION
; ------------------------------
; (offset: $04 'multiply')
;   Begin by trying integer multiplication as used on the Sinclair ZX80.
;   If that overflows then use floating point multiplication.

multiply  LD    A,(DE)          ; fetch exponent byte of second number.
          OR    (HL)            ; combine with that of first number.
          JR    NZ,MULT_LONG    ; forward, if either not integer, to MULT-LONG

          PUSH  DE              ; save pointer to second number - new STKEND.
          PUSH  HL              ; save pointer to first number - result pointer.

          PUSH  DE              ; save pointer to second number on stack again.

          CALL  INT_FETCH       ; routine INT-FETCH integer to DE, sign to C.

          EX    DE,HL           ; transfer first integer from DE to HL
          EX    (SP),HL         ; integer to stack and second pointer to HL.
          LD    B,C             ; place first sign byte in B.

          CALL  INT_FETCH       ; routine INT-FETCH integer to DE, sign to C
                                ; and B preserved.

;   Now manipulate sign bytes so that minus times a minus gives a plus result.

          LD    A,B             ; fetch first sign byte        $00 or $FF.
          XOR   C               ; XOR with second sign byte.   $00 or $FF.
          LD    C,A             ; transfer sign of result to C.

          POP   HL              ; pop first integer off the machine stack.

          CALL  HL_HLxDE        ; routine HL-HL*DE multiplies the two integers.

          EX    DE,HL           ; transfer the result to DE.

          POP   HL              ; restore the result pointer to HL.

          JR    C,MULT_OFLW     ; forward, with overflow, to MULT-OFLW

;   Note. these next 5 bytes ensure that -zero (00 FF 00 00 00) is replaced
;   by zero (00 00 00 00 00).  They are required in the case of say,
;   0 * -1 which gives the result -0.  This would be printed as -1E-38.
;   Note. Contrary to the view expressed in The Complete Spectrum ROM
;   Disassembly, these 5 bytes should not be deleted.

          LD    A,D             ; test 3rd
          OR    E               ; and 4th bytes for zero.

          JR    NZ,MULT_RSLT    ; skip forward, if not, to MULT-RSLT

          LD    C,A             ; make 2nd byte, possibly $FF, zero also.

MULT_RSLT JP    INT_STO_3       ;+ jump to similar code.

;;;       CALL  INT_STORE       ; routine INT-STORE stores result at HL.
;;;       POP   DE              ; retrieve the new pointer to STKEND.
;;;       RET                   ; Return.

; ---

;   The branch was here when simple register-based multiplication overflowed.
```

```
MULT_OFLW POP   DE              ;

MULT_LONG CALL  RE_ST_TWO       ; routine RE-ST-TWO
          XOR   A               ;
          CALL  PREP_M_D        ; routine PREP-M/D
          RET   C               ;

          EXX                   ;
          PUSH  HL              ;
          EXX                   ;
          PUSH  DE              ;
          EX    DE,HL           ;
          CALL  PREP_M_D        ; routine PREP-M/D
          EX    DE,HL           ;
          JR    C,ZERO_RSLT     ; to ZERO-RSLT

          PUSH  HL              ;
          CALL  FETCH_TWO       ; routine FETCH-TWO
          LD    A,B             ;
          AND   A               ;
          SBC   HL,HL           ;
          EXX                   ;
          PUSH  HL              ;
          SBC   HL,HL           ;
          EXX                   ;
          LD    B,$21           ;
          JR    STRT_MLT        ; to STRT-MLT

; ---

MLT_LOOP  JR    NC,NO_ADD       ; to NO-ADD

          ADD   HL,DE           ;
          EXX                   ;
          ADC   HL,DE           ;
          EXX                   ;

NO_ADD    EXX                   ;
          RR    H               ;
          RR    L               ;
          EXX                   ;
          RR    H               ;
          RR    L               ;

STRT_MLT  EXX                   ;
          RR    B               ;
          RR    C               ;
          EXX                   ;
          RR    C               ;
          RRA                   ;
          DJNZ  MLT_LOOP        ; to MLT-LOOP

          EX    DE,HL           ;
          EXX                   ;
          EX    DE,HL           ;
          EXX                   ;
          POP   BC              ;
          POP   HL              ;
          LD    A,B             ;
          ADD   A,C             ;
          JR    NZ,MAKE_EXPT    ; to MAKE-EXPT

          AND   A               ;
```

```
MAKE_EXPT DEC    A                   ;
          CCF                        ; Complement Carry Flag

DIVN_EXPT RLA                        ;
          CCF                        ; Complement Carry Flag
          RRA                        ;
          JP     P,OFLW1_CLR         ; to OFLW1-CLR

          JR     NC,REPORT_6         ; to REPORT-6
                                     ; 'Number too big'

          AND    A                   ;

OFLW1_CLR INC    A                   ;
          JR     NZ,OFLW2_CLR        ; to OFLW2-CLR

          JR     C,OFLW2_CLR         ; to OFLW2-CLR

          EXX                        ;
          BIT    7,D                 ;
          EXX                        ;
          JR     NZ,REPORT_6         ; to REPORT-6

OFLW2_CLR LD     (HL),A              ;
          EXX                        ;
          LD     A,B                 ;
          EXX                        ;

TEST_NORM JR     NC,NORMALIZE        ; to NORMALISE

          LD     A,(HL)              ;
          AND    A                   ;

NEAR_ZERO LD     A,$80               ;
          JR     Z,SKIP_ZERO         ; to SKIP-ZERO

ZERO_RSLT XOR    A                   ;

SKIP_ZERO EXX                        ;
          AND    D                   ;
          CALL   ZEROS_4_5           ; routine ZEROS-4/5
          RLCA                       ;
          LD     (HL),A              ;
          JR     C,OFLOW_CLR         ; to OFLOW-CLR

          INC    HL                  ;
          LD     (HL),A              ;
          DEC    HL                  ;
          JR     OFLOW_CLR           ; to OFLOW-CLR

; ---

NORMALIZE LD     B,$20               ;

SHIFT_ONE EXX                        ;
          BIT    7,D                 ;
          EXX                        ;
          JR     NZ,NORML_NOW        ; to NORML-NOW

          RLCA                       ;
          RL     E                   ;
          RL     D                   ;
          EXX                        ;
          RL    E                ;
          RL    D                ;
          EXX                    ;
          DEC   (HL)             ;
          JR    Z,NEAR_ZERO      ; to NEAR-ZERO

          DJNZ  SHIFT_ONE        ; to SHIFT-ONE

          JR    ZERO_RSLT        ; to ZERO-RSLT

; ---

NORML_NOW RLA                    ;
          JR    NC,OFLOW_CLR     ; to OFLOW-CLR

          CALL  ADD_BACK         ; routine ADD-BACK
          JR    NZ,OFLOW_CLR     ; to OFLOW-CLR

          EXX                    ;
          LD     D,$80           ;
          EXX                    ;
          INC   (HL)             ;
          JR    Z,REPORT_6       ; to REPORT-6

OFLOW_CLR 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  RST   30H              ; ERROR-1
          DEFB  $05              ; Error Report: Number too big

; -----------------------
; THE 'DIVISION' OPERATION
; -----------------------
; (offset: $05 'division')
;
;

division  CALL  RE_ST_TWO        ; routine RE-ST-TWO

          EX    DE,HL            ;
          XOR   A                ;
          CALL  PREP_M_D         ; routine PREP-M/D
          JR    C,REPORT_6       ; to REPORT-6

          EX    DE,HL            ;
          CALL  PREP_M_D         ; routine PREP-M/D
          RET   C                ;

          EXX                    ;
          PUSH  HL               ;
          EXX                    ;

          PUSH  DE               ;
          PUSH  HL               ;
          CALL  FETCH_TWO        ; 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    DIV_START        ; to DIV-START

; ---

DIV_LOOP  RLA                    ;
          RL    C                ;
          EXX                    ;
          RL    C                ;
          RL    B                ;
          EXX                    ;

div_34th  ADD   HL,HL            ;
          EXX                    ;
          ADC   HL,HL            ;
          EXX                    ;
          JR    C,SUBN_ONLY      ; to SUBN-ONLY

DIV_START SBC   HL,DE            ;
          EXX                    ;
          SBC   HL,DE            ;
          EXX                    ;
          JR    NC,NO_RSTORE     ; to NO-RSTORE

          ADD   HL,DE            ;
          EXX                    ;
          ADC   HL,DE            ;
          EXX                    ;
          AND   A                ;
          JR    COUNT_ONE        ; to COUNT-ONE

; ---

SUBN_ONLY AND   A                ;
          SBC   HL,DE            ;
          EXX                    ;
          SBC   HL,DE            ;
          EXX                    ;

NO_RSTORE SCF                    ; Set Carry Flag

COUNT_ONE INC   B                ;
```

```
          JP    M,DIV_LOOP      ; to DIV-LOOP

          PUSH  AF              ;

          JR    Z,DIV_START     ; 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    DIVN_EXPT       ; jump back to DIVN-EXPT

; -------------------------------------------------
; THE 'INTEGER TRUNCATION TOWARDS ZERO' SUBROUTINE
; -------------------------------------------------
; (offset: $3A 'truncate')
;   This routine returns the integer of the 'last value' truncated towards zero
;   so that, for example, the result for PI would be 3 and the result for -PI
;   would be -3 (and not -4 as returned by the BASIC INT function).

truncate  LD    A,(HL)          ; Fetch the first byte.
          AND   A               ; Test for zero which indicates an integer.
          RET   Z               ; return if a small integer.

          CP    $81             ; compare exponent to +1
          JR    NC,T_GR_ZERO    ; forward, if 1 or more, to T-GR-ZERO

;   The number is smaller than plus or minus one and can be made zero.

          LD    (HL),$00        ; insert zero in first byte.
          LD    A,$20           ; prepare to reset all 32 bits of 'mantissa'
          JR    NIL_BYTES       ; forward to NIL-BYTES

; ---

T_GR_ZERO CP    $91             ; compare exponent to +16

;   Note. the next section is designed to convert 91 80 00 00 00 to the
;   integer 00 FF 00 00 00 . "This is a pity since the number would have
;   been perfectly all right if left alone.  The remedy would seem to be
;   simply to omit the 28 bytes [below] from the program."
;   credit: Dr. Ian Logan 1983.

;;;       JR    NZ,T_SMALL      ; to T-SMALL
```

```
;;;
;;;       INC   HL             ;
;;;       INC   HL             ;
;;;       INC   HL             ;
;;;       LD    A,$80          ;
;;;       AND   (HL)           ;
;;;       DEC    HL            ;
;;;       OR    (HL)           ;
;;;       DEC   HL             ;
;;;       JR    NZ,T_FIRST     ; to T-FIRST

;;;       LD    A,$80          ;
;;;       XOR   (HL)           ;

;;; T_FIRST DEC HL             ;
;;;       JR    NZ,T_EXPNENT   ; to T-EXPNENT

;;;       LD    (HL),A         ;
;;;       INC   HL             ;
;;;       LD    (HL),$FF       ;
;;;       DEC   HL             ;
;;;       LD    A,$18          ;
;;;       JR    NIL_BYTES      ; to NIL-BYTES


T_SMALL   JR    NC,X_LARGE     ; forward if more than 16-bit integer to X-LARGE

;   The number is a small integer +/- 1-65535 and can be held in two bytes.

          PUSH  DE             ; Preserve the STKEND pointer.

;   The exponent ($81 to $90) is converted to a shift count - one to sixteen.

          CPL                  ; Complement - range      $7F - $70
          ADD   A,$91          ; Add to give shift count $10 - $01

          INC   HL             ; Point to first mantissa byte.
          LD    D,(HL)         ; Load to high-order byte.
          INC   HL             ; Point to next byte of mantissa.
          LD    E,(HL)         ; Load to low-order byte.

          DEC   HL             ; Restore pointer
          DEC   HL             ; to position at first byte.

          LD    C,$00          ; prepare a positive sign byte.
          BIT   7,D            ; test sign bit of mantissa byte.

          JR    Z,T_NUMERIC    ; skip, if positive, to T-NUMERIC

          DEC   C              ; make $FF - negative sign byte.

T_NUMERIC SET   7,D            ; put back the 'implied' bit.

;   Now see if 8 bits can be right-shifted at once (if number < 256)

          LD    B,$08          ; prepare 8 in B
          SUB   B              ; subtract from shift counter in A.
          ADD   A,B            ; and add back.
          JR    C,T_TEST       ; forward, if number > 255, to T-TEST

          LD    E,D            ; Transfer MSB to LSB
          LD    D,$00          ; Make MSB zero.
          SUB   B              ; subtract 8 from the shift counter.
```

```
T_TEST    JR    Z,T_STORE       ; forward, if no more shifts, to T-STORE

          LD    B,A             ; Transfer count to B.

T_SHIFT   SRL   D               ; 0 -> 76543210 -> C
          RR    E               ; C -> 76543210 -> C
          DJNZ  T_SHIFT         ; back for count to T-SHIFT

T_STORE   JP    INT_STO_3       ;+ jump to similar code.

;;;       CALL  INT_STORE       ; routine INT-STORE stores integer DE at HL.
;;;       POP   DE              ; Restore the STKEND value from the stack.
;;;       RET                   ; Return.


; ---

;   The next instruction is made redundant by Dr. Logan's fix.

;;; T_EXPNENT LD    A,(HL)      ; This instruction is never reached.

; ---

;   The branch was here when the number was a large number e.g. 1000000.567
;   The accumulator holds the exponent.


X_LARGE   SUB   $A0             ; Subtract +32 decimal from the exponent.
          RET   P               ; Return if the result is positive as 32 bits
                                ; of the mantissa relate to the integer part.
                                ; The radix point is somewhere to the right of
                                ; the mantissa.

          NEG                   ; else negate to form number of rightmost bits
                                ; to be blanked.

;   For instance, disregarding the sign bit, the number 3.5 is held as
;   exponent $82 mantissa .11100000 00000000 00000000 00000000.
;   We need to reset $82 - $A0 = $E2, which negated = $1E (thirty) bits to
;   form the integer of 3.5.

NIL_BYTES PUSH  DE              ; Save pointer to STKEND.

          EX    DE,HL           ; Register HL now points to STKEND
          DEC   HL              ; Now at last byte of mantissa.

          LD    B,A             ; Transfer the bit count to register B,

;   Now look into the possibility of blanking eight bits at a time.

          SRL   B               ; Divide
          SRL   B               ; by
          SRL   B               ; eight.

          JR    Z,BITS_ZERO     ; forward, if zero, to BITS-ZERO

BYTE_ZERO LD    (HL),$00        ; set eight bits to zero.
          DEC   HL              ; point to more significant byte of mantissa.
          DJNZ  BYTE_ZERO       ; loop back for all to BYTE-ZERO

BITS_ZERO AND   $07             ; mask the remaining bits from original count.
          JR    Z,IX_END        ; forward, if none, to IX-END

          LD    B,A             ; transfer bit count to B counter.
          LD    A,$FF           ; form an initial mask %11111111

LESS_MASK SLA   A               ;  1 <- 76543210 <- 0 slide mask leftwards.
          DJNZ  LESS_MASK       ; loop back, for bit count, to LESS-MASK

          AND   (HL)            ; lose the unwanted rightmost bits.
          LD    (HL),A          ; and place in the mantissa byte.

IX_END    EX    DE,HL           ; Restore the result pointer.

          POP   DE              ; Restore STKEND value from stack.

          RET                   ; Return.

; ---------------------------------------
; THE '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 CALL  RESTK_SUB       ; routine RESTK-SUB  below and continue
                                ; into the routine to do the other one.

RESTK_SUB EX    DE,HL           ; swap pointers and continue into same routine.

; --------------------------------------------
; 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 the first byte.

re_stack  LD    A,(HL)          ; Fetch Exponent byte to A
          AND   A               ; test it
          RET   NZ              ; return if first byte is not zero as number
                                ; is already in floating-point form.

          PUSH  DE              ; preserve DE.

          CALL  INT_FETCH       ; routine INT-FETCH integer to DE, sign to C.

;   Note. the above routine returns HL pointing to 4th byte.

          XOR   A               ; clear the accumulator.

;   Note. The fifth byte of an integer is always zero for neatness so the next
;   step is, I imagine, unnecessary.

;;;       INC   HL              ; point to 5th.
;;;       LD    (HL),A          ; and blank.
;;;       DEC   HL              ; point to 4th.

          LD    (HL),A          ; blank the 4th byte.

          LD    B,$91           ; set exponent byte +ve $81.
                                ; and imaginary radix 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 order byte D
          AND   A               ; is it zero ?
          JR    NZ,RS_NRMLSE    ; skip, if not, to RS-NRMLSE

;   Check if the number is zero in which case no modification is required.
;   However, by updating first three bytes we convert minus zero to plus
;   zero although I'm not sure if it can arise. It is eliminated in mult.

          OR    E               ; Fetch low byte E to A and test for zero.
          LD    B,D             ; set B, the exponent, to 0
          JR    Z,RS_STORE      ; forward, if value is zero, to RS-STORE
```

```
;   Move the significant bits eight places to the left.

          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 EX    DE,HL           ; integer to HL, addr of 4th byte to DE.

RSTK_LOOP DEC   B               ; decrease exponent
          ADD   HL,HL           ; shift HL left
          JR    NC,RSTK_LOOP    ; 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  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

;    Register HL now points at result.

          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'
; ------------------------
;   These five constants are now held in full five byte integer or
;   floating-point form as it makes it much easier to pass them to the
;   calculator stack when required.


;;; 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)

TAB_CNST  DEFB  $00             ;+ the value zero.
          DEFB  $00             ;+
          DEFB  $00             ;+
          DEFB  $00             ;+
          DEFB  $00             ;+

          DEFB  $00             ;+ the integer value 1.
          DEFB  $00             ;+
          DEFB  $01             ;+
          DEFB  $00             ;+
          DEFB  $00             ;+

          DEFB  $80             ;+ the floating point value a half.
          DEFB  $00             ;+
          DEFB  $00             ;+
          DEFB  $00             ;+
          DEFB  $00             ;+

          DEFB  $81             ;+ the floating point value pi/2
          DEFB  $49             ;+
          DEFB  $0F             ;+
          DEFB  $DA             ;+
          DEFB  $A2             ;+

          DEFB  $00             ;+ the integer value ten.
          DEFB  $00             ;+
          DEFB  $0A             ;+
          DEFB  $00             ;+
          DEFB  $00             ;+

; ------------------------
; THE 'TABLE OF ADDRESSES'
; ------------------------
;
;   Starts with binary operations which have two operands and one result.
;   Three pseudo binary operations first.

tbl_addrs DEFW  jump_true       ; $00 Address: $368F - jump-true
          DEFW  exchange        ; $01 Address: $343C - exchange
          DEFW  delete          ; $02 Address: $33A1 - delete

;   True binary operations.

          DEFW  subtract        ; $03 Address: $300F - subtract
          DEFW  multiply        ; $04 Address: $30CA - multiply
          DEFW  division        ; $05 Address: $31AF - division
          DEFW  to_power        ; $06 Address: $3851 - to-power
```

```
        DEFW    or              ; $07 Address: $351B - or

        DEFW    no_v_no         ; $08 Address: $3524 - no-&-no
        DEFW    multcmp         ; $09 Address: $353B - no-l-eql
        DEFW    multcmp         ; $0A Address: $353B - no-gr-eql
        DEFW    multcmp         ; $0B Address: $353B - nos-neql
        DEFW    multcmp         ; $0C Address: $353B - no-grtr
        DEFW    multcmp         ; $0D Address: $353B - no-less
        DEFW    multcmp         ; $0E Address: $353B - nos-eql
        DEFW    addition        ; $0F Address: $3014 - addition

        DEFW    str_v_no        ; $10 Address: $352D - str-&-no
        DEFW    multcmp         ; $11 Address: $353B - str-l-eql
        DEFW    multcmp         ; $12 Address: $353B - str-gr-eql
        DEFW    multcmp         ; $13 Address: $353B - strs-neql
        DEFW    multcmp         ; $14 Address: $353B - str-grtr
        DEFW    multcmp         ; $15 Address: $353B - str-less
        DEFW    multcmp         ; $16 Address: $353B - strs-eql
        DEFW    strs_add        ; $17 Address: $359C - strs-add

;   Unary follow.

        DEFW    val_s           ; $18 Address: $35DE - val$
        DEFW    usr_str         ; $19 Address: $34BC - usr-$
        DEFW    read_in         ; $1A Address: $3645 - read-in
        DEFW    negate          ; $1B Address: $346E - negate

        DEFW    code            ; $1C Address: $3669 - code
        DEFW    val             ; $1D Address: $35DE - val
        DEFW    len             ; $1E Address: $3674 - len
        DEFW    sin             ; $1F Address: $37B5 - sin
        DEFW    cos             ; $20 Address: $37AA - cos
        DEFW    tan             ; $21 Address: $37DA - tan
        DEFW    asn             ; $22 Address: $3833 - asn
        DEFW    acs             ; $23 Address: $3843 - acs
        DEFW    atn             ; $24 Address: $37E2 - atn
        DEFW    ln              ; $25 Address: $3713 - ln
        DEFW    exp             ; $26 Address: $36C4 - exp
        DEFW    int             ; $27 Address: $36AF - int
        DEFW    sqr             ; $28 Address: $384A - sqr
        DEFW    sgn             ; $29 Address: $3492 - sgn
        DEFW    abs             ; $2A Address: $346A - abs
        DEFW    peek            ; $2B Address: $34AC - peek
        DEFW    in              ; $2C Address: $34A5 - in
        DEFW    usr_no          ; $2D Address: $34B3 - usr-no
        DEFW    str_s           ; $2E Address: $361F - str$
        DEFW    chrs            ; $2F Address: $35C9 - chrs
        DEFW    not             ; $30 Address: $3501 - not

;   End of true unary.

        DEFW    MOVE_FP         ; $31 Address: $33C0 - duplicate
        DEFW    n_mod_m         ; $32 Address: $36A0 - n-mod-m
        DEFW    JUMP            ; $33 Address: $3686 - jump
        DEFW    stk_data        ; $34 Address: $33C6 - stk-data
        DEFW    dec_jr_nz       ; $35 Address: $367A - dec-jr-nz
        DEFW    less_0          ; $36 Address: $3506 - less-0
        DEFW    greater_0       ; $37 Address: $34F9 - greater-0
        DEFW    end_calc        ; $38 Address: $369B - end-calc
        DEFW    get_argt        ; $39 Address: $3783 - get-argt
        DEFW    truncate        ; $3A Address: $3214 - truncate
        DEFW    fp_calc_2       ; $3B Address: $33A2 - fp-calc-2
        DEFW    E_TO_FP         ; $3C Address: $2D4F - e-to-fp
        DEFW    re_stack        ; $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  seriesg_x       ;     Address: $3449 - series-xx    $80 - $9F.
          DEFW  stk_con_x       ;     Address: $341B - stk-const-xx $A0 - $BF.
          DEFW  sto_mem_x       ;     Address: $342D - st-mem-xx    $C0 - $DF.
          DEFW  get_mem_x       ;     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 'CALCULATE' SUBROUTINE
; --------------------------
;
;

CALCULATE CALL  STK_PNTRS       ; 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 LD    A,B             ; fetch the Z80 B register to A
          LD    ($5B67),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 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  LD    ($5B65),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  PUSH  HL              ; save pointer on stack
          AND   A               ; now test the literal
          JP    P,FIRST_3D      ; 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    ENT_TABLE       ; forward to ENT-TABLE

; ---

;   the branch was here with simple literals.

FIRST_3D  CP    $18             ; compare with first unary operations.
          JR    NC,DOUBLE_A     ; 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.

          CALL  STK_PTRS2       ; Routine to perform the above.

          EXX                   ; switch to alternate set of registers.

DOUBLE_A  RLCA                  ; double the literal
          LD    L,A             ; and store in L for indexing

ENT_TABLE LD    DE,tbl_addrs    ; Address: tbl-addrs
          LD    H,$00           ; prepare to index
          ADD   HL,DE           ; add to point to address of routine

          LD    E,(HL)          ; low byte of address to E
          INC   HL              ;
          LD    D,(HL)          ; high byte of address to D

          LD    HL,RE_ENTRY     ; Address: RE-ENTRY
          EX    (SP),HL         ; goes to stack and address of 'next literal'
                                ; goes to HL'

          PUSH  DE              ; now stack the address of the routine

          EXX                   ; switch back to 'main' set

;   Avoid using the IY register.

          LD    BC,($5B66)      ; STKEND_hi
                                ; nothing much goes to C but BREG to B
                                ; and continue into next ret instruction
                                ; which has a dual identity

; ----------------------
; THE 'DELETE' OPERATION
; ----------------------
; (offset: $02 'delete')
;   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    RET                   ; return - indirect jump if from above.

; ------------------------------
; THE 'SINGLE OPERATION' ROUTINE
; ------------------------------
; (offset: $3B 'fp_calc2')
;   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 POP   AF              ; drop return address.
          LD    A,($5B67)       ; load accumulator from system variable BREG
                                ; value will be literal e.g. 'tan'
          EXX                   ; switch to alt
          JR    SCAN_ENT        ; 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.

TEST_5_SP PUSH  DE              ; preserve.
          PUSH  HL              ; registers
          LD    BC,$0005        ;  an overhead of eighty five bytes

          CALL  TEST_ROOM       ; routine TEST_ROOM checks space for 5 bytes.

          POP   HL              ; (balance)
          POP   DE              ;
          RET                   ; then return - OK.

; -----------------------------
; 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 LD    DE,($5B65)      ; Load destination from STKEND system variable.

          CALL  MOVE_FP         ; Routine MOVE-FP puts on calculator stack
                                ; with a memory check.
          LD    ($5B65),DE      ; Set STKEND to next free location.

          RET                   ; Return.

; -------------------------
; THE 'DUPLICATE' OPERATION
; -------------------------
; (offset: $31 'duplicate')

;   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   CALL  TEST_5_SP       ; routine TEST-5-SP test free memory
                                ; and sets B to zero.
BLK_MV    LDIR                  ; copy the five bytes.
          RET                   ; return with DE addressing new STKEND
                                ; and HL addressing new last value.

; -------------------------------
; THE 'STACK LITERALS' OPERATION
; -------------------------------
; (offset: $34 'stk_data')
;   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  LD    H,D             ; transfer STKEND
          LD    L,E             ; to HL for result.

STK_CONST CALL  TEST_5_SP       ; 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 the number of bytes
                                ; to read. $01 - $04
          LD    A,(HL)          ; reload the first byte
          AND   $3F             ; mask off bits 5 - 0 to give possible exponent.
          JR    NZ,FORM_EXP     ; Forward to FORM-EXP if it was possible to
                                ; include the exponent and count in one byte.

;   else byte is just a byte count and reduced exponent comes next.

          INC   HL              ; address next byte and
          LD    A,(HL)          ; pick up the exponent ( -$50 ).

FORM_EXP  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 (B=0, fr test5sp)

          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 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    STK_ZEROS       ; loop back to STK-ZEROS until done.

; -------------------------------------------
; THE REDUNDANT 'SKIP CONSTANTS' SUBROUTINE
; -------------------------------------------
;   This routine traversed 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  STK_CONST       ; routine STK-CONST works through variable
;;;                                 ; length records.

;;;           POP   DE              ; restore real STKEND
;;;           POP   AF              ; restore count
;;;           DEC   A               ; decrease
;;;           JR    SKIP_NEXT       ; 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   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.

; ------------------------------------
; THE 'GET FROM MEMORY AREA' OPERATION
; ------------------------------------
;   Literals $E0 to $FF
;   A holds $00-$1F offset.
;   The calculator stack increases by 5 bytes.

get_mem_x LD    HL,($5B68)      ; MEM is base address of the memory cells.

INDEX_5   PUSH  DE              ; save STKEND

          CALL  LOC_MEM         ; routine LOC-MEM so that HL = first byte
          CALL  MOVE_FP         ; routine MOVE-FP moves 5 bytes with necessary
                                ; memory check.
                                ; DE now points to new STKEND.

          POP   HL              ; original STKEND is now RESULT pointer.
          RET                   ; return.

; --------------------------------
; THE 'STACK A CONSTANT' OPERATION
; --------------------------------
;   Offsets $A0 to $A4
;   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_con_x  LD   HL,TAB_CNST     ; Address table of five byte expanded constants

           JR   INDEX_5         ; back to common code in routine above.

;;; stk-con-x
;;; L341B:    LD    H,D             ; save STKEND - required for result
;;;           LD    L,E             ;
;;;           EXX                   ; swap
;;;           PUSH  HL              ; save pointer to next literal
;;;           LD    HL,TAB_CNST     ; Address: stk-zero - start of table of
;;;                                 ; constants
;;;           EXX                   ;
;;;           CALL  SKIP_CONS       ; routine SKIP-CONS
;;;           CALL  STK_CONST       ; routine STK-CONST
;;;           EXX                   ;
;;;           POP   HL              ; restore pointer to next literal.
;;;           EXX                   ;
;;;           RET                   ; return.

; -------------------------------
; THE 'STORE IN MEMORY' OPERATION
; -------------------------------
; 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.

sto_mem_x PUSH  HL                ; save the result pointer.
          EX    DE,HL             ; transfer to DE.
          LD    HL,($5B68)        ; fetch MEM the base of memory area.
          CALL  LOC_MEM           ; routine LOC-MEM sets HL to the destination.
          EX    DE,HL             ; swap - HL is start, DE is destination.
;;;       CALL  MOVE_FP           ; routine MOVE-FP.
;;;                               ; Note. a short ld bc,5; ldir
;;;                               ; the embedded memory check is not required
;;;                               ; so these instructions would be faster.

          LD    C,$05             ;+ Set number of bytes to five, B is zero.
          LDIR                    ;+ Block copy the bytes avoiding RAM check.

          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  LD    B,$05             ; there are five bytes to be swapped

;   Start of loop.

SWAP_BYTE LD    A,(DE)            ; Each byte of second
;;;       LD    C,(HL)            ; Each byte of first
;;;       EX    DE,HL             ; Swap pointers
          LD    C,A               ;+
          LD    A,(HL)            ;+
          LD    (DE),A            ; Store each byte of first
          LD    (HL),C            ; Store each byte of second
          INC   HL                ; Advance both
          INC   DE                ; pointers.
          DJNZ  SWAP_BYTE         ; Loop back to SWAP-BYTE until all 5 done.

;;;       EX    DE,HL             ; Even up the exchanges so that DE addresses
;;;                               ; system variable 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.

seriesg_x LD    B,A             ; parameter $00 - $1F to B counter

          CALL  GEN_ENT_1       ; 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    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  stk_data        ; routine STK-DATA is called directly to
                                ; push a value and advance H'L'.
          CALL  GEN_ENT_2       ; 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  G_LOOP - $      ;;back to 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 the calculator stack.

abs       LD    B,$FF           ; signal abs
          JR    NEG_TEST        ; forward to NEG-TEST

; ---------------------------
; THE 'UNARY MINUS' OPERATION
; ---------------------------
; (offset: $1B 'negate')
;   e.g. LET balance = -2
;   Unary, so on entry HL points to last value, DE to STKEND.

negate    CALL  TEST_ZERO       ; 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  LD    A,(HL)          ; load first byte and
          AND   A               ; test for zero which indicates a small integer.
          JR    Z,INT_CASE      ; forward, if so, to INT-CASE

;   for the FLOATING POINT CASE 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  PUSH  DE              ; save STKEND.

          PUSH  HL              ; save pointer to the last value/result.

          CALL  INT_FETCH       ; 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
          JR    INT_STO_2       ;+ Forward to similar code.

;;;       LD    C,A             ; transfer result to sign byte.
;;;       CALL  INT_STORE       ; 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       CALL  TEST_ZERO       ; call routine TEST-ZERO and
          RET   C               ; exit if zero 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.

INT_STO_2 LD    C,A             ; store the sign byte in the C register.

INT_STO_3 CALL  INT_STORE       ; 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        CALL  FIND_INT2       ; Routine FIND-INT2 puts port address in BC.
                                ; All 16 bits are put on the address line.

          IN    A,(C)           ; Read the port.

          JR    IN_PK_STK       ; 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 examined including the ROM.

peek      CALL  FIND_INT2       ; routine FIND-INT2 puts the address in BC.
          LD    A,(BC)          ; load the contents into A register.

IN_PK_STK JP    STACK_A         ; 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    CALL  FIND_INT2       ; routine FIND-INT2 to fetch the
                                ; supplied address into BC.

          LD    HL,STK_BC_IY    ; NEW address: STK_BC_IY 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_str
;;;       CALL  STK_FETCH       ; routine STK-FETCH fetches the string
;;;                             ; parameters.
;;;       DEC   BC              ; decrease BC by
;;;       LD    A,B             ; one to test
;;;       OR    C               ; the length.
;;;       JR    NZ,REPORT_A     ; to REPORT-A if not a single character.
;;;
;;;       LD    A,(DE)          ; fetch the character

          CALL  EXPT_SPEC       ;+ fetch a single char

          CALL  ALPHA           ; routine ALPHA sets carry if 'A-Z' or 'a-z'.
          JR    C,USR_RANGE     ; forward, if ASCII, to USR-RANGE

          SUB   $90             ; make UDGs range 0-20d
          JR    C,REPORT_A      ; to REPORT-A if too low. e.g. usr " ".

;;;       CP    $15             ; Note. this test is not necessary.
;;;       JR    NC,REPORT_A     ; to REPORT-A if higher than 20.

          INC   A               ; make range 1-21d to match LSBs of ASCII

USR_RANGE 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,REPORT_A     ; to REPORT-A if originally higher
                                ; than 'U','u' or graphics U.
          LD    BC,($5B7B)      ; 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,USR_STACK    ; forward to USR-STACK if no overflow.

          INC   B               ; increment high byte.

USR_STACK JP    STACK_BC        ; jump back and exit via STACK-BC to store

; ---

REPORT_A  RST   30H             ; 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 only the first four bytes need be tested.
;   On entry, HL points to the exponent the first byte of the value.

TEST_ZERO 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 setting zero flag.

          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 CALL  TEST_ZERO       ; 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    SIGN_TO_C       ; 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       CALL  TEST_ZERO       ; routine TEST-ZERO sets carry if zero

          JR    FP_0_1          ; 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 the second byte will be set if it is.  This will either be the
;   first bit of a 32-bit mantissa or part of the sign byte if an integer.

less_0    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 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    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 for neatness.

          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        EX    DE,HL           ; make HL point to second number

          CALL  TEST_ZERO       ; 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    FP_0_1          ; 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_v_no   EX    DE,HL           ; make HL address second operand.

          CALL  TEST_ZERO       ; 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    FP_0_1          ; 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_v_no  EX    DE,HL           ; make HL point to the number.

          CALL  TEST_ZERO       ; routine TEST-ZERO.

          EX    DE,HL           ; Restore the two 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.

          INC   DE              ;+ Restore DE using two increments which
          INC   DE              ;+ is quicker than using the machine stack.

          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.
;                         NO
; literal    Test   No [sub 8]      ExOrNot  1st RRCA  exch sub  ?   End-Tests
; =========  ====   == ======== === ======== ========  ==== ===  =  === === ===
; no-l-eql   x<=y   09 00001001 dec 00001000 00000100  ---- x-y  ?  --- >0? NOT
; no-gr-eql  x>=y   0A 00001010 dec 00001001 10000100c swap y-x  ?  --- >0? NOT
; nos-neql   x<>y   0B 00001011 dec 00001010 00000101  ---- x-y  ?  NOT --- NOT
; no-grtr    x>y    0C 00001100  -  00001100 00000110  ---- x-y  ?  --- >0? ---
; no-less    x<y    0D 00001101  -  00001101 10000110c swap y-x  ?  --- >0? ---
; nos-eql    x=y    0E 00001110  -  00001110 00000111  ---- x-y  ?  NOT --- ---
;
;                                                      comp -> C/F
;                                                      ====    ===
; str-l-eql  x$<=y$ 11 00010001 dec 00010000 00001000  ---- x$y$ 0  !or >0? NOT
; str-gr-eql x$>=y$ 12 00010010 dec 00010001 10001000c swap y$x$ 0  !or >0? NOT
; strs-neql  x$<>y$ 13 00010011 dec 00010010 00001001  ---- x$y$ 0  !or >0? NOT
; str-grtr   x$>y$  14 00010100  -  00010100 00001010  ---- x$y$ 0  !or >0? ---
; str-less   x$<y$  15 00010101  -  00010101 10001010c swap y$x$ 0  !or >0? ---
; strs-eql   x$=y$  16 00010110  -  00010110 00001011  ---- 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.

multcmp   LD    A,B             ; transfer literal to accumulator.

;;;       SUB   $08             ; subtract eight - which is not useful.

          BIT   2,A             ; isolate '>', '<', '='.

          JR    NZ,EX_OR_NOT    ; skip to EX-OR-NOT with these.

          DEC   A               ; else make $00-$02, $08-$0A to match bits 0-2.

EX_OR_NOT RRCA                  ; the first RRCA sets carry for a swap.
          JR    NC,NU_OR_STR    ; 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  exchange        ; 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.

NU_OR_STR RRCA                  ;+ causes 'eql'/'neql' to set carry.
          PUSH  AF              ;+ save carry flag.

;   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.

          BIT   2,A             ; test if a string comparison.
          JR    NZ,STRINGS      ; forward, if so, to STRINGS

;   continue with numeric comparisons.

;;;       RRCA                  ; 2nd RRCA causes eql/neql to set carry.

;;;       PUSH  AF              ; save A and carry
```

```
          CALL  subtract        ; routine subtract leaves result on stack.
          JR    END_TESTS       ; forward to END-TESTS

; ---

;;;       RRCA                  ; 2nd RRCA causes eql/neql to set carry.

;;;       PUSH  AF              ; save A and carry.

STRINGS   CALL  STK_FETCH       ; routine STK-FETCH gets 2nd string params
          PUSH  DE              ; save start2 *.
          PUSH  BC              ; and the length.

          CALL  STK_FETCH       ; 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 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,SEC_PLUS     ; forward to SEC-PLUS if second not null.

          OR    C               ; test length of first string.

SECND_LOW POP   BC              ; pop the second length off stack.
          JR    Z,BOTH_NULL     ; 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    STR_TEST        ; forward to leave via STR-TEST

; ---
;   the branch was here with a match

BOTH_NULL POP   AF              ; restore carry - set for eql/neql
          JR    STR_TEST        ; 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  OR    C               ; test the length of first string.
          JR    Z,FRST_LESS     ; 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,FRST_LESS     ; forward to FRST-LESS if carry set

          JR    NZ,SECND_LOW    ; 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    BYTE_COMP       ; back to BYTE-COMP

; ---
;   the false condition.

FRST_LESS 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  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 POP   AF              ; pop carry  - will be set if eql/neql
          PUSH  AF              ; save it again.

          CALL  C,not           ; 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,greater_0    ; 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,not          ; if comparison then apply a terminal NOT

          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  CALL  STK_FETCH       ; routine STK-FETCH fetches string parameters
                                ; and deletes calculator stack entry.
          PUSH  DE              ; save start address.
          PUSH  BC              ; and length.
          CALL  STK_FETCH       ; routine STK-FETCH for the 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 result to BC
          LD    C,L             ;

          CALL  BC_SPACES       ; routine BC_SPACES creates room in workspace.
                                ; DE points to start of space.

          CALL  STK_STO_s       ; 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,OTHER_STR     ; to OTHER-STR if null string
;;;       LDIR                  ; copy the first string to workspace.

          CALL  COND_MV         ; A three-byte call to ldir saves a byte.

OTHER_STR POP   BC              ; now second length
          POP   HL              ; and start of string

;;;       LD    A,B             ; test this one
;;;       OR    C               ; for zero length
;;;       JR    Z,STK_PNTRS     ; skip forward to STK-PNTRS if so as complete.
;;;       LDIR                  ; else copy the bytes.

          CALL  COND_MV         ; A three-byte call to ldir saves a byte.

;   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 LD    HL,($5B65)      ; 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.

STK_PTRS2 LD    D,H             ; transfer to DE
          LD    E,L             ;
          DEC   HL              ; Make HL 5 locations lower.
          DEC   HL              ;
          DEC   HL              ;
          DEC   HL              ;
          DEC   HL              ;
          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      CALL  FP_TO_A         ; routine FP-TO-A puts the number in A.

          JR    C,REPORT_Bd     ; forward to REPORT-Bd if overflow
          JR    NZ,REPORT_Bd    ; forward to REPORT-Bd if negative

;;;       PUSH  AF              ; save the argument.

;;;       LD    BC,$0001        ; one space required.

          CALL  BC_SPACE1       ; BC_SPACE1 makes DE point to start

;;;       POP   AF              ; restore the number.

          LD    (DE),A          ; and store in workspace
          JR    str_STK         ;+ forward to similar code.

;;;       CALL  STK_STO_s       ; routine STK-STO-$ stacks descriptor.
;;;       EX    DE,HL           ; make DE point to STKEND.
;;;       RET                   ; return.

; ---

REPORT_Bd RST   30H             ; 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_s     RST   18H             ;;;
;;;       LD    HL,($5B5D)      ; 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  STK_FETCH       ; routine STK-FETCH fetches the string operand
                                ; from the calculator stack.

          PUSH  DE              ; save the address of the start of the string.
          INC   BC              ; increment the length for a carriage return.
          CALL  BC_SPACES       ; BC_SPACES creates the space in workspace.

          POP   HL              ; restore start of string to HL.
          LD    ($5B5D),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  SCANNING        ; routine SCANNING evaluates string
                                ; expression and result.

;;;       RST   18H             ; GET-CHAR fetches next character.   ????

          CP    $0D             ; is next char the expected carriage return ?
          JR    NZ,V_RPORT_C    ; forward, if not, to V-RPORT-C
                                ; '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 JP    NZ,REPORT_C     ;.jump back to REPORT-C with a result mismatch.

          LD    ($5B5D),HL      ; set CH_ADD to the start of the string again.
          SET   7,(IY+$01)      ; update FLAGS  - signal running program.

          CALL  SCANNING        ; routine SCANNING evaluates the string
                                ; in full leaving result on calculator stack.

          POP   HL              ; restore saved character address in program.
          LD    ($5B5D),HL      ; and reset the system variable CH_ADD.

V_ST_PTRS JR    STK_PNTRS       ; 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".

;;; strs  LD    BC,$0001        ; create an initial byte in workspace
;;;       RST   30H             ; using BC_SPACES restart.

str_s     CALL  BC_SPACE1       ;+ create an initial byte in workspace.

          LD    ($5B5B),HL      ; set system variable K_CUR to new location.
          PUSH  HL              ; and save start on machine stack also.

          LD    HL,($5B51)      ; fetch value of system variable CURCHL
          PUSH  HL              ; and save that too.
          LD    A,$FF           ; select system channel 'R'.

          CALL  CHAN_SLCT       ; routine CHAN-OPEN opens it.
          CALL  PRINT_FP        ; routine PRINT-FP outputs the number to
                                ; workspace updating K-CUR.

          POP   HL              ; restore current channel.
          CALL  CHAN_FLAG       ; routine CHAN-FLAG resets flags.

          POP   DE              ; fetch saved start of string to DE.
          LD    HL,($5B5B)      ; 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.

str_STK   CALL  STK_STO_s       ; routine STK-STO-$ stores string parameters
                                ; on the calculator stack.

          EX    DE,HL           ; Make DE point to 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. It appears to provide for the reading
;   of data through different streams from those available on the standard
;   Spectrum.
;   INKEY$ # does not interact correctly with the keyboard, #0 or #1, and
;   its uses are for other channels - Steven Vickers, Pitman Pocket Book.

read_in
          LD    HL,($5B51)      ; fetch current channel CURCHL
          PUSH  HL              ; save it

;;;       CALL  FIND_INT1       ; routine FIND-INT1 fetches stream to A
;;;       CP    $10             ; compare with 16 decimal.
;;;       JP    NC,REPORT_B     ; JUMP to REPORT-B  if not in range 0 - 15.
;;;       CALL  CHAN_SLCT       ; routine CHAN-OPEN opens channel

          CALL  CHAN_CHK        ;+ natural routine opens, if valid, else errors
                                ;+ with 'Invalid stream' instead of 'Integer
                                ;+ out of range'

          CALL  IN_CHAN_K       ;+ keyboard ?

          JR    NZ,READ_IT      ;+ Forward if not

          HALT                  ;+ Read the keyboard.

READ_IT   CALL  INPUT_AD        ; 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,R_I_STORE    ; forward to R-I-STORE if no key detected.

;;;       INC   C               ; increase length to one.

          CALL  BC_SPACE1       ; NEW routine BC_SPACE1 creates space for one
                                ; character in workspace.
          LD    (DE),A          ; the character is inserted.
```

```
R_I_STORE CALL  STK_STO_s       ; routine STK-STO-$ stacks the string
                                ; parameters.

          POP   HL              ; Restore current channel address

          CALL  CHAN_FLAG       ; Routine CHAN-FLAG resets current channel
                                ; system variable and flags.

          JR    V_ST_PTRS       ; jump back indirectly 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      CALL  STK_FETCH       ; 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,STK_CODE      ; skip to STK-CODE with zero if the null string.

          LD    A,(DE)          ; else fetch the first character.

STK_CODE  JP    STACK_A         ; jump back to STACK-A (with memory check)

; ------------------
; THE 'LEN' FUNCTION
; ------------------
; (offset: $1e 'len')
;   Returns the length of a string.
;   In Sinclair BASIC, workable strings can be more than twenty thousand
;   characters long so a sixteen-bit register is required to store the length.

len       CALL  STK_FETCH       ; Routine STK-FETCH to fetch and delete the
                                ; string parameters from the calculator stack.
                                ; Register BC now holds the length of string.

          JP    STACK_BC        ; 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 EXX                   ; switch in set that addresses code

          PUSH  HL              ; save pointer to offset byte
          LD    HL,$5B67        ; address BREG in system variables
          DEC   (HL)            ; decrement it
          POP   HL              ; restore pointer

          JR    NZ,JUMP_2       ; forward, if not zero, to JUMP_2

          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      EXX                     ; switch in pointer set

JUMP_2    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 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,JUMP           ; 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.
;   Note. this short 5-byte routine has been moved to space between the
;   restarts to exploit spare space.


; ------------------------
; 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   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       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  X_NEG - $       ;;to X-NEG

          DEFB  $3A             ;;truncate              trunc 3.4 = 3.
          DEFB  $38             ;;end-calc              3.

          RET                   ; return with + int x on stack.

; ---


X_NEG     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  EXIT - $        ;;to EXIT               -3.

          DEFB  $A1             ;;stk-one               -3, 1.
          DEFB  $03             ;;subtract              -4.

EXIT      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       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  FP_TO_A         ; routine FP-TO-A
          JR    NZ,N_NEGTV      ; to N-NEGTV

          JR    C,REPORT_6b     ; to REPORT-6b
                                ; 'Number too big'

          ADD   A,(HL)          ;
          JR    NC,RESULT_OK    ; to RESULT-OK


REPORT_6b RST   30H             ; ERROR-1
          DEFB  $05             ; Error Report: Number too big

; ---

N_NEGTV   JR    C,RSLT_ZERO     ; to RSLT-ZERO

          SUB   (HL)            ;
          JR    NC,RSLT_ZERO    ; to RSLT-ZERO

          NEG                   ; Negate
```

```
RESULT_OK LD    (HL),A          ;
          RET                   ; return.

; ---


RSLT_ZERO 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        RST   28H             ;; FP-CALC
          DEFB  $3D             ;;re-stack
          DEFB  $31             ;;duplicate
          DEFB  $37             ;;greater-0
          DEFB  $00             ;;jump-true
          DEFB  VALID - $       ;;to VALID

INV_ARG   DEFB  $38             ;;end-calc


REPORT_Ab RST   30H             ; ERROR-1
          DEFB  $09             ; Error Report: Invalid argument

VALID
;;;       DEFB  $A0             ;;stk-zero              This is unnecessary
;;;       DEFB  $02             ;;delete
          DEFB  $38             ;;end-calc
          LD    A,(HL)          ;

          LD    (HL),$80        ;
          CALL  STACK_A         ; 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  GRE_v_8 - $     ;;to 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_v_8   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, of a right-angled triangle, 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 of a right-angled triangle 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  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  ZPLUS - $       ;;to 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     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  YNEG - $        ;;to YNEG
                                ;;if angle in quadrant III

;   else angle is within quadrant II (-1 to 0)

          DEFB  $1B             ;;negate          range +1 to 0.

YNEG      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       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  C_ENT - $       ;;fwd to C-ENT
                                ;;forward to common code if in QII or QIII.

          DEFB  $1B             ;;negate               else make sign +ve.
          DEFB  $33             ;;jump
          DEFB  C_ENT - $       ;;fwd to 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       RST   28H             ;; FP-CALC      angle in radians
          DEFB  $39             ;;get-argt      reduce - sign now correct.

C_ENT     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       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       CALL  re_stack        ; routine re-stack
          LD    A,(HL)          ; fetch exponent byte.
          CP    $81             ; compare to that for 'one'
          JR    C,SMALL         ; 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  CASES - $       ;;to CASES

          DEFB  $1B             ;;negate
          DEFB  $33             ;;jump
          DEFB  CASES - $       ;;to CASES

SMALL     RST   28H             ;; FP-CALC
          DEFB  $A0             ;;stk-zero

CASES     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       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       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 NEW 'SQUARE ROOT' FUNCTION
; ------------------------------
; (Offset $28: 'sqr')
;   "If I have seen further, it is by standing on the shoulders of giants" -
;   Sir Isaac Newton, Cambridge 1676.
;   The sqr function has been re-written to use the Newton-Raphson method.
;   Although the method is centuries old, this one, appropriately, is based
;   on a FORTH word written by Steven Vickers in the Jupiter Ace manual.
;   Whereas that algorithm always used an initial guess of one, this one
;   manipulates the exponent byte to obtain a better guess.
;   First test for zero and return zero, if so, as the result.
;   If the argument is negative, then produce an error.

sqr       RST   28H             ;; FP-CALC              x
          DEFB  $3D             ;;re-stack              x.   (in f.p. form)
          DEFB  $C3             ;;st-mem-3              x.   (seed for guess)
          DEFB  $38             ;;end-calc

;    The HL register now addresses the exponent byte

          LD    A,(HL)          ; fetch exponent to A
          AND   A               ; test for zero.
          RET   Z               ; return if so - with zero on calculator stack.

          INC   HL              ; address the byte with the sign bit.
          BIT   7,(HL)          ; test the sign bit

          JP    NZ,REPORT_Ab    ; REPORT_A: 'Invalid argument'

;   This guess is based on a Usenet discussion.
;   Halve the exponent to achieve a good guess.(accurate with .25 16 64 etc.)

          LD    HL,$5BA1        ; Address system variable mem-3
          LD    A,(HL)          ; fetch exponent of mem-3
          XOR   $80             ; toggle sign of exponent of mem-3
          SRA    A                ; shift right, bit 7 unchanged.
          INC    A                ;
          JR     Z,ASIS           ; forward with say .25 -> .5
          JP     P,ASIS           ; leave increment if value > .5
          DEC    A                ; restore to shift only.

ASIS      XOR    $80              ; restore sign.
          LD     (HL),A           ; and put back 'halved' exponent.

;   Now re-enter the calculator.

          RST    28H              ;; FP-CALC              x

SLOOP     DEFB   $31              ;;duplicate             x,x.
          DEFB   $E3              ;;get-mem-3             x,x,guess
          DEFB   $C4              ;;st-mem-4              x,x,guess
          DEFB   $05              ;;div                   x,x/guess.
          DEFB   $E3              ;;get-mem-3             x,x/guess,guess
          DEFB   $0F              ;;addition              x,x/guess+guess
          DEFB   $A2              ;;stk-half              x,x/guess+guess,.5
          DEFB   $04              ;;multiply              x,(x/guess+guess)*.5
          DEFB   $C3              ;;st-mem-3              x,newguess
          DEFB   $E4              ;;get-mem-4             x,newguess,oldguess
          DEFB   $03              ;;subtract              x,newguess-oldguess
          DEFB   $2A              ;;abs                   x,difference.
          DEFB   $37              ;;greater-0             x,(0/1).
          DEFB   $00              ;;jump-true             x.

          DEFB   SLOOP - $        ;;to sloop              x.

          DEFB   $02              ;;delete                .
          DEFB   $E3              ;;get-mem-3             retrieve final guess.
          DEFB   $38              ;;end-calc              sqr x.

          RET                     ; return with square root on stack


; ------------------------------------
; THE OLD SLOW 'SQUARE ROOT' FUNCTION
; ------------------------------------
; (Offset $28: 'sqr')
;   This is the old 7-byte method of calculating square roots which has been
;   re-introduced at various stages during the development of this ROM due to
;   lack of space.

;;; sqr       RST     28H             ;; FP-CALC
;;;           DEFB    $31             ;;duplicate
;;;           DEFB    $30             ;;not
;;;           DEFB    $00             ;;jump-true
;;;           DEFB    LAST - $        ;;to 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  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  XIS0 - $        ;;to 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    exp             ; 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.

XIS0      DEFB  $02             ;;delete                  Y.
          DEFB  $31             ;;duplicate               Y, Y.
          DEFB  $30             ;;not                     Y, (1/0).
          DEFB  $00             ;;jump-true
          DEFB  ONE - $         ;;to 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  LAST - $        ;;to 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       DEFB  $02             ;;delete                  .
          DEFB  $A1             ;;stk-one                 1.

LAST      DEFB  $38             ;;end-calc                last value is 1 or 0.

          RET                   ; return.                 Whew!


; ----------------------------
; THE 'SPARE LOCATIONS' PART 3
; ----------------------------

TAG7

SPARE     DEFB  $FF, $FF, $FF                        ;


ORG $3D00
```

```
; -------------------------------
; THE 'ZX SPECTRUM CHARACTER SET'
; -------------------------------


; $20 - Character: ' '          CHR$(32)

char_set  DEFB  %00000000
LINE_ZERO 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.
; Gianluca Carri            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.
; Andrew Owen               for ZASM compatibility and format improvements.
; Philip Kendall            for help with Newton Raphson square root theory.
; James Smith               for optimizing some ROM routines to save space
;                               and the FORMAT routine.
;
; ----------------------
; THE 'SYSTEM VARIABLES'
; ----------------------

; 5B00 (IY-$3A)  23296  KSTATE_0        $FF (free) else raw key value.
; 5B01 (IY-$39)  23297  KSTATE_1        The 5-counter
; 5B02 (IY-$38)  23298  KSTATE_2        Initially REPDEL value then REPPER
; 5B03 (IY-$37)  23299  KSTATE_3        This location holds the decoded key.
; -------------------------------
; 5B04 (IY-$36)  23300  KSTATE_4        The second key map is arranged
; 5B05 (IY-$35)  23301  KSTATE_5        exactly as the first above and it is
; 5B06 (IY-$34)  23302  KSTATE_6        in fact this map that is considered
; 5B07 (IY-$33)  23303  KSTATE_7        first by the Keyboard routines.
; -------------------------------
; 5B08 (IY-$32)  23304  LASTK           Value of last key read from keyboard.
; 5B09 (IY-$31)  23305  REPDEL
; 5B0A (IY-$30)  23306  REPPER
; 5B0B (IY-$2F)  23307  DEFADD
; 5B0C (IY-$2E)  23308  DEFADD_hi
; 5B0D (IY-$2D)  23309  KDATA
; 5B0E (IY-$2C)  23310  TVDATA
; 5B0F (IY-$2B)  23311  TVDATA
; -------------------------------
; 5B10 (IY-$2A)  23312  STRMS_FD
; 5B11 (IY-$29)  23313  STRMS_FD_hi
; 5B12 (IY-$28)  23314  STRMS_FE
; 5B13 (IY-$27)  23315  STRMS_FE_hi
; 5B14 (IY-$26)  23316  STRMS_FF
; 5B15 (IY-$25)  23317  STRMS_FF_hi
; 5B16 (IY-$24)  23318  STRMS_00
; 5B17 (IY-$23)  23319  STRMS_00_hi
; 5B18 (IY-$22)  23320  STRMS_01
; 5B19 (IY-$21)  23321  STRMS_01_hi
; 5B1A (IY-$20)  23322  STRMS_02
; 5B1B (IY-$1F)  23323  STRMS_02_hi
; 5B1C (IY-$1E)  23324  STRMS_03
; 5B1D (IY-$1D)  23325  STRMS_03_hi
; 5B1E (IY-$1C)  23326  STRMS_04
; 5B1F (IY-$1B)  23327  STRMS_04_hi
; 5B20 (IY-$1A)  23328  STRMS_05
; 5B21 (IY-$19)  23329  STRMS_05_hi
; 5B22 (IY-$18)  23330  STRMS_06
; 5B23 (IY-$17)  23331  STRMS_06_hi
; 5B24 (IY-$16)  23332  STRMS_07
; 5B25 (IY-$15)  23333  STRMS_07_hi
; 5B26 (IY-$14)  23334  STRMS_08
; 5B27 (IY-$13)  23335  STRMS_08_hi
; 5B28 (IY-$12)  23336  STRMS_09
; 5B29 (IY-$11)  23337  STRMS_09_hi
; 5B2A (IY-$10)  23338  STRMS_0A
; 5B2B (IY-$0F)  23339  STRMS_0A_hi
; 5B2C (IY-$0E)  23340  STRMS_0B
; 5B2D (IY-$0D)  23341  STRMS_0B_hi
; 5B2E (IY-$0C)  23342  STRMS_0C
; 5B2F (IY-$0B)  23343  STRMS_0C_hi
; 5B30 (IY-$0A)  23344  STRMS_0D
; 5B31 (IY-$09)  23345  STRMS_0D_hi
; 5B32 (IY-$08)  23346  STRMS_0E
; 5B33 (IY-$07)  23347  STRMS_0E_hi
; 5B34 (IY-$06)  23348  STRMS_0F
; 5B35 (IY-$05)  23349  STRMS_0F_hi
; -------------------------------
; 5B36 (IY-$04)  23350  CHARS
; 5B37 (IY-$03)  23351  CHARS_hi
; 5B38 (IY-$02)  23352  RASP
; 5B39 (IY-$01)  23353  PIP
; 5B3A (IY+$00)  23354  ERR_NR
; -------------------------------
; 5B3B (IY+$01)  23355  FLAGS           0 - Set to suppress a leading space.
;                                       1 - Set if ZX Printer is in use.
;                                       2 - Set if 'L' mode, temporary value.
;                                       3 - Set if 'L' mode, reset for 'K' perm.
;                                       4 - Unused by 48K BASIC.
;                                       5 - Set in a new key has been pressed.
;                                       6 - Set if scanning result is numeric.
;                                       7 - Reset if checking syntax.
; -------------------------------
; 5B3C (IY+$02)  23356  TV_FLAG         0 - Set if lower screen in use.
;                                       1 - unused.
;                                       2 - unused.
;                                       3 - Set if edit key has been pressed.
;                                       4 - Set if an automatic listing.
;                                       5 - Set if lower screen to be cleared.
;                                       6 - unused.
;                                       7 - unused.
; -------------------------------
; 5B3D (IY+$03)  23357  ERR_SP
; 5B3E (IY+$04)  23358  ERR_SP_hi
; 5B3F (IY+$05)  23359  LIST_SP
; 5B40 (IY+$06)  23360  LIST_SP_hi
; 5B41 (IY+$07)  23361  MODE            Values 0, 1 or 2
; 5B42 (IY+$08)  23362  NEWPPC
; 5B43 (IY+$09)  23363  NEWPPC_hi
; 5B44 (IY+$0A)  23364  NSPPC
; 5B45 (IY+$0B)  23365  PPC
; 5B46 (IY+$0C)  23366  PPC_hi
; 5B47 (IY+$0D)  23367  SUBPPC
; 5B48 (IY+$0E)  23368  BORDCR
; 5B49 (IY+$0F)  23369  E_PPC
; 5B4A (IY+$10)  23370  E_PPC_hi
; ---------------------------------
; 5B4B (IY+$11)  23371  VARS
; 5B4C (IY+$12)  23372  VARS_hi
; 5B4D (IY+$13)  23373  DEST
; 5B4E (IY+$14)  23374  DEST_hi
; 5B4F (IY+$15)  23375  CHANS
; 5B50 (IY+$16)  23376  CHANS_hi
; 5B51 (IY+$17)  23377  CURCHL
; 5B52 (IY+$18)  23378  CURCHL_hi
; 5B53 (IY+$19)  23379  PROG
; 5B54 (IY+$1A)  23380  PROG_hi
; 5B55 (IY+$1B)  23381  NXTLIN
; 5B56 (IY+$1C)  23382  NXTLIN_hi
; 5B57 (IY+$1D)  23383  DATADD
; 5B58 (IY+$1E)  23384  DATADD_hi
; 5B59 (IY+$1F)  23385  E_LINE
; 5B5A (IY+$20)  23386  E_LINE_hi
; 5B5B (IY+$21)  23387  K_CUR
; 5B5C (IY+$22)  23388  K_CUR_hi
; 5B5D (IY+$23)  23389  CH_ADD
; 5B5E (IY+$24)  23390  CH_ADD_hi
; 5B5F (IY+$25)  23391  X_PTR
; 5B60 (IY+$26)  23392  X_PTR_hi
; 5B61 (IY+$27)  23393  WORKSP
; 5B62 (IY+$28)  23394  WORKSP_hi
; 5B63 (IY+$29)  23395  STKBOT
; 5B64 (IY+$2A)  23396  STKBOT_hi
; 5B65 (IY+$2B)  23397  STKEND
; 5B66 (IY+$2C)  23398  STKEND_hi
; ---------------------------------
; 5B67 (IY+$2D)  23399  BREG
; 5B68 (IY+$2E)  23400  MEM
; 5B69 (IY+$2F)  23401  MEM_hi
; -------------------------------
; 5B6A (IY+$30)  23402  FLAGS2          0 - Set if main screen to be cleared.
;                                       1 - Not used - held state of ZX buffer.
;                                       2 - Set if a ':' is within quotes.
;                                       3 - Set if Caps Lock on.
;                                       4 - Set if "K" channel is use.
;                                       5 - unused.
;                                       6 - unused.
;                                       7 - unused.
; ------------------------------
; 5B6B (IY+$31)  23403  DF_SZ
; 5B6C (IY+$32)  23404  S_TOP
; 5B6D (IY+$33)  23405  S_TOP_hi
; 5B6E (IY+$34)  23406  OLDPPC
; 5B6F (IY+$35)  23407  OLDPPC_hi
; 5B70 (IY+$36)  23408  OSPPC
; ------------------------------
; 5B71 (IY+$37)  23409  FLAGX           0 - Set if handling a simple string.
;                                       1 - Set if handling a new variable.
;                                       2 - unused.
;                                       3 - unused.
;                                       4 - unused.
;                                       5 - Set if in input mode.
;                                       6 - unused.
;                                       7 - Set if handling INPUT LINE.
; ------------------------------
; 5B72 (IY+$38)  23410  STRLEN
; 5B73 (IY+$39)  23411  STRLEN_hi
; 5B74 (IY+$3A)  23412  T_ADDR
; 5B75 (IY+$3B)  23413  T_ADDR_hi
; 5B76 (IY+$3C)  23414  SEED
; 5B77 (IY+$3D)  23415  SEED_hi
; 5B78 (IY+$3E)  23416  FRAMES1
; 5B79 (IY+$3F)  23417  FRAMES2
; 5B7A (IY+$40)  23418  FRAMES3
; 5B7B (IY+$41)  23419  UDG
; 5B7C (IY+$42)  23420  UDG_hi
; 5B7D (IY+$43)  23421  COORDS_x
; 5B7E (IY+$44)  23422  COORDS_y
; 5B7F (IY+$45)  23423  P_POSN (unused)
; 5B80 (IY+$46)  23424  PR_CC  (unused)
; 5B81 (IY+$47)  23425  PR_CC  (unused)
; 5B82 (IY+$48)  23426  ECHO_E
; 5B83 (IY+$49)  23427  ECHO_E_hi
; 5B84 (IY+$4A)  23428  DF_CC
; 5B85 (IY+$4B)  23429  DF_CC_hi
; 5B86 (IY+$4C)  23430  DFCCL
; 5B87 (IY+$4D)  23431  DFCCL_hi
; 5B88 (IY+$4E)  23432  S_POSN
; 5B89 (IY+$4F)  23433  S_POSN_hi
; 5B8A (IY+$50)  23434  SPOSNL
; 5B8B (IY+$51)  23435  SPOSNL_hi
; 5B8C (IY+$52)  23436  SCR_CT
; 5B8D (IY+$53)  23437  ATTR_P
; 5B8E (IY+$54)  23438  MASK_P
; 5B8F (IY+$55)  23439  ATTR_T
; 5B90 (IY+$56)  23440  MASK_T
; 5B91 (IY+$57)  23441  P_FLAG
; ------------------------------
; 5B92 (IY+$58)  23442  MEM_0
; 5B93 (IY+$59)  23443  MEM_0
; 5B94 (IY+$5A)  23444  MEM_0
; 5B95 (IY+$5B)  23445  MEM_0
; 5B96 (IY+$5C)  23446  MEM_0
; 5B97 (IY+$5D)  23447  MEM_1
; 5B98 (IY+$5E)  23448  MEM_1
; 5B99 (IY+$5F)  23449  MEM_1
; 5B9A (IY+$60)  23450  MEM_1
; 5B9B (IY+$61)  23451  MEM_1
; 5B9C (IY+$62)  23452  MEM_2
; 5B9D (IY+$63)  23453  MEM_2
; 5B9E (IY+$64)  23454  MEM_2
; 5B9F (IY+$65)  23455  MEM_2
; 5BA0 (IY+$66)  23456  MEM_2
; 5BA1 (IY+$67)  23457  MEM_3
; 5BA2 (IY+$68)  23458  MEM_3
; 5BA3 (IY+$69)  23459  MEM_3
; 5BA4 (IY+$6A)  23460  MEM_3
; 5BA5 (IY+$6B)  23461  MEM_3
; 5BA6 (IY+$6C)  23462  MEM_4
; 5BA7 (IY+$6D)  23463  MEM_4
; 5BA8 (IY+$6E)  23464  MEM_4
; 5BA9 (IY+$6F)  23465  MEM_4
; 5BAA (IY+$70)  23466  MEM_4
; 5BAB (IY+$71)  23467  MEM_5
; 5BAC (IY+$72)  23468  MEM_5
; 5BAD (IY+$73)  23469  MEM_5
; 5BAE (IY+$74)  23470  MEM_5
; 5BAF (IY+$75)  23471  MEM_5
; -------------------------------
; 5BB0 (IY+$76)  23472  NMI_ADD
; 5BB1 (IY+$77)  23473  NMI_ADD_hi
; 5BB2 (IY+$78)  23474  RAMTOP
; 5BB3 (IY+$79)  23475  RAMTOP_hi
; 5BB4 (IY+$7A)  23476  P_RAMT
; 5BB5 (IY+$7B)  23477  P_RAMT_hi
; ----------------------------------------------------------------------------
; 5BB6 (IY+$7C)  23478  FLAGS3          unused - holds FF to show no Interface1
; 5BB7 (IY+$7D)  23479  WIDTH           RS232 Printer column variable
; 5BB8 (IY+$7E)  23480  WIDTH           Printer width as set by FORMAT "t"
; 5BB9 (IY+$7F)  23481  MAXIY           unused
; 5BBA           23482  BAUD_lo         Two Byte number determining the BAUD
; 5BBB           23483  BAUD_hi         rate.   BAUD=(3500000/(26*baud rate))-2
; 5BBC           23484  NTSTAT          Own Network station number
; 5BBD           23485  IOBORD          Border colour used during I/O
; 5BBE           23486  SER_FL          2-byte workspace used by RS232
; 5BBF           23487  SER_FL          holds second character if first is one
; 5BC0           23488  NTRESP          Store for the network response code.
; 5BC1           23489  NTDEST          Destination station number.
; 5BC2           23490  NTSRCE          Source station number.
; 5BC3           23491  NTNUMB_lo       Network block number - two bytes
; 5BC4           23492  NTNUMB_hi       as received over the network
; 5BC5           23493  NTTYPE          Header type code as received.
; 5BC6           23494  NTLEN           Data block length 0-255.
; 5BC7           23495  NTDCS           Data block checksum.
; 5BC8           23496  NTHCS           Header block checksum.
;
;
; Revision History
; ----------------
;
; 25-AUG-2002
; All references to System Variables changed from $5C to $5B.
; Changed byte in CHAN-OPEN from $5C to $5B.
; The COPY command re-written so as not to clear the ZX Printer Buffer.
; Channel "P" removed from INITIAL CHANNEL DATA
; Reduced number of bytes copied during initialization from 21 to 16.
; Stream 3 entry removed from INITIAL STREAM DATA
; Reduced number of bytes copied during initialization from 14 to 12.
; Reduced offset in expression within CLOSE from 14 to 12.
; Modify CLOSE routine to error if stream offset is already closed (zero).
; Add new Error report 'Stream is closed' to error message table.
; Modify DE offset in CLOSE from $A3E2 to $A4E4 to reflect new STRMS location
;  and fewer system streams.
; Substantially alter CLOSE-2 so that IX used to access letter and offset saved
;  in DE. Start of channel saved in IX.
;  (Noticed for the first time pointer to letter was previously saved in DE).
; Create new INITIAL P CHANNEL DATA (8 bytes) for channel creation. Contains
;  usual 5 bytes plus + 2-byte length + P_POSN (column position). The output
;  address PR_CC can't be held as this would vary as channels are deleted.
;
; 26-AUG-2002
; Modify OPEN-1 so that a stream associated with "P" can't be re-attached.
; Modify OPEN-K so that LD E,$01 becomes LD DE,$0001.
; Modify OPEN-S so that LD E,$06 becomes LD DE,$0006.
; Completely re-write OPEN-P (which was no more sophisticated) so that it
;  creates a 264 byte "P" channel at end of CHANS area.
; Remove line in OPEN-END that set high byte of offset to zero.??
; Remove call to CLEAR-PRB during initialization.
; Remove call to COPY-BUFF at MAIN-4 [ * This will have to be re-visited ]
; Modify print routine so that FLAGS2 is not updated when a ZX buffer used.
; Modify PR-ALL-6 so that address can cross a 256-byte page boundary.
; Modify CLEAR-PRB so that FLAGS2 not cleared when buffer cleared.
; Modify CLEAR-PRB so that superfluous PR_CC reference removed.
; Modify CLEAR_PRB so that address of buffer is calculated from CURCHL.
; Modify COPY-LINE so that CLEAR-PRB only invoked when outputting to a channel
;  and BREAK is pressed.
; Realising that CLEAR-PRB can only be used as such, go back to OPEN-P and
;  clear the buffer bytes directly.
; Modify COPY_BUFF so that address of buffer is calculated from CURCHL.
; Remove "LD HL,$5B00" from start of CL-SET. Still works for ZX path.
; Modify PO-STORE so that IX from CURCHL used to update P_POSN (channel var).
; Substantially modify PO-FETCH so that the print address within the channel
;  buffer is formed from the column position P_POSN.
;
; 27-AUG-2002
; Write routine CLOSE-P so that channel reclaimed.
;  Although not intended as such, this routine turns out to be generic.
; Adapt routine REST-STRM (from Interface 1) so that all other streams that
;  have offsets beyond a closed stream have their offsets reduced by the
;  reclaimed amount.
; Boot using VBSpec and test that channels open and close OK.
; Switch to RealSpec to test printing. Crash due to A not being preserved
;  during new PO-FETCH. Trace and rectify using debugger and notice all OK.
; Switch to DOS Z80 emulator for final paper output tests.  Brilliant as always.
; Re-locate FREE-MEM routine to spare space between restarts as address was
;  moving around as code was added and removed.
; Free memory has increased from 41472 to 41733
; ROM space has reduced alarmingly. Only 1040 bytes spare.
; Embark on a spree to remove redundant code.
;  Routines ZX81 name routine, REC-EDIT and P-INT-STO commented out.

; 28-AUG-2002
; Table of constants expanded to five bytes.
; SKIP-CONS commented out - no longer writes to ROM.
; First two instructions of get-mem swapped to provide entry point from
; the stk-con-x routine which is now just 5 bytes.
; To inhibit all writes to ROM, limit scroll routine to 23 lines (not 24).
; Interrupt routine re-written to avoid IY register use. (saves one byte)
; Applied fixes to KEYBOARD so that keywords don't repeat and only valid keys
;  return a graphic key code i.e. only A-U.
; Generally, comment out unnecessary stack saves, double loads etc.
; Since labels have no relation to address, change them to use legalized
;  disassembly labels.

; 31-AUG-2002
; Square Root function rewritten to use the Newton-Raphson method.
;  Can be improved further by finding a better initial guess than 1.

; 01-SEP-2002
; Use better initial guess than '1' for Newton Raphson SQR function.
;  Works even better when integers are immediately re-stacked as floating point
;  numbers.

; 03-SEP-2002
; More fixes - allow SAVE "program" LINE without number as per BASIC manual.
; ROM space pruning - use UNSTACK_Z to full potential.

; 04-SEP-2002
; For consistency, use the words "CONTINUE" and "GO SUB" in error messages
;  instead of the ZX81 tokens used in the production ZX Spectrum ROM.
;  This was a mistake as both messages were at the maximum length. Reverted.
; The INT -65536 bug fixed as per Dr. Ian Logan's guidelines ensuring that the
;  3rd, 4th and 5th bytes were zero.

; 05-SEP-2002
; All quirks, features and bugs removed. Details as follows.

; Source: Understanding Your Spectrum by Dr. Ian Logan
; (12 bugs listed in appendix)
; i.   The 'division' error - is a misnomer. The inaccuracy mentioned occurs
;      in the DEC_TO_FP routine and by switching the multiply and division
;      operations then 0.5 is given the floating point form 80 00 00 00 00.
;      The suggested fix is ignored.
; ii.  The '-65536' error e.g. PRINT INT -65536 gives -1.
;      Dr. Ian Logan's fix applied (with mods) and other code sections removed
;      as suggested.
; iii. The 'program name' routine removed along with REC_EDIT and P_INT_STO.
; iv.  The 'CHR$ 9' error corrected by calling PO_ABLE in preference to a
;      terminal jump to CL_SET/PO_STORE.
; v.   The 'scroll?' and 'Start tape' errors corrected by new routine CONS_IN.
;      Later KEY_INPUT modified to recognize prompt (2 extra bytes).
; vi.  The current cursor error corrected by updating E_PPC with valid line
;      number while it is in the registers at an earlier stage.
; vii. The 'leading space error' resembles more a successful attempt to
;      maximise the text that will fit within a 32 character display and has
;      not been corrected. Ignored. (On second thoughts, this needs fixing)
; viii.The 'K-mode' error has been corrected by preventing keywords repeating
;      when held down.
; ix.  The 'CHR$ 8' error has been corrected as suggested by Dr. Frank O'Hara.
; x.   The 'SCREEN$' error has been corrected by substituting the suggested RET
;      instruction.
; xi.  The 'STR$' error has been corrected by removing the extra zero from
;      the calculator stack as suggested.
; xii. The 'CLOSE' error has been corrected by checking the status of the
;      stream and issuing a new error message if it is already closed.
;      The suggested fix - adding a zero end-marker to the Close Stream Look-up
;      Table - is ignored.
;
; Source: The Complete Spectrum ROM Disassembly. by Dr. Ian Logan and
;      Dr. Frank O'Hara (various additional features listed).
; 1)   The NMI bug has been corrected and the logic changed as suggested on
;      Page 2. The new default set-up is to produce a new informative message.
; 2)   Simple strings are not excluded when saving DATA - on Page 22.
;      e.g.  10 LET a$ = "dodo" : SAVE "animal" DATA a$()
;      These are now rejected as they won't load back in.
;      (credit: First fixed by Dr. Ian Logan in the Interface 1 ROM).
; 3)   There is no end-marker for the CLOSE STREAM LOOK UP table nor should
;      there be. Ignored.
;
; Source: ZX Spectrum BASIC programming by Steven Vickers. (discrepancies)
; 1)   Line number should be optional in SAVE "some name" LINE - Page 133.
;      Fixed.
; 2)   CLEAR does a RESTORE (Page 124).
;      Error in BASIC manual rather than ROM - ignored. Difficult to decide.
; 3)   "Notice that the numbers in a DRAW statement can be negative, although
;      those in a PLOT statement can't" - Page 92
;      Fixed. 0<=x<=255. 0<=y<=175 else Error B.
; 4)   Similarly the POINT (x,y) function allowed negative coordinates.
;      Fixed. Error B unless 0<=x<=255. 0<=y<=175. Page 153.
; 5)   The ATTR (y,x) function allows negative and invalid coordinates.
;      Fixed. Error B unless 0<=x<=31 and 0<=y<=23.  Page 152.
; 6)   The SCREEN$ (y,x) function allows negative and invalid parameters.
;      Fixed. Error B unless 0<=x<=31 and 0<=y<=23.  Page 154.
;
; Source: The Pitman Pocket Guide to the Sinclair Spectrum by Steven Vickers.
; (discrepancies not previously mentioned.)
; 1)   RESTORE. "Don't specify numbers > 9999, as the program may crash."
;      To be pedantic > 16383 - see below.                         Page 25.
; 2)   'Statement lost' can occur with RUN, GO TO and GO SUB when the line
;      number is between 32768 and 61439.                          Page 67.
;      Fixed by new routine which checks SAVE LINE, LIST, LLIST, RUN, GO TO,
;      GO SUB and RESTORE for invalid line numbers.
; 3)   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  Page 59.
;      Fixed by ensuring that the screen is first selected.
; 4)   EDITING KEYS TABLE                                          Page 58.
;      When inputting from the network or RS232 or microdrive file,
;      code 6 (comma separator): inserted in buffer.
;      ("This is a bug. It should work like CHR$ 14"). Fixed.
;
; Source: www.nonowt.com "Bugs in the ROM"
; ( many already covered. Some are Programming Guides rather than errors. )
; 1)   The Monopolizing of IY Error.
;      Although not strictly an error, the manual does not mention the
;      restriction as did the ZX81 manual.  Also some effort has gone into
;      ensuring that the calculator avoids IY mathematically and it is restored
;      following a USR function.
;      Fixed - the interrupt routine uses HL to access system variables.
;      Saves a few bytes too.
; 2)   The PR_CC error (credit: Dilwyn Jones 1983).
;      Fixed - No ZX printer system variables remain. The print position is
;      recalculated every time from a single new channel variable.
; 3)   The CLEAR PRINTER BUFFER Bug.
;      Fixed - COPY no longer clears the buffer at the end of the statement or
;      when BREAK pressed.
; 4)   The Main-4 COPY-BUFF Error.
;      Partly fixed as routine is no longer called but unprinted output is not
;      yet flushed. ( To revisit at end ) (Done.)
; 5)   The MAIN-4 HALT instruction not corrected as not really an error.
;      The fault was with programmers and also with the Interface 1.
;      The NMI fix provides a clean means of exit should the situation arise.
;      i.e. Should a programmer forget to enable interrupts before returning
;      to BASIC. However I have to admit I don't know why it is there. If you
;      press BREAK it ensures the message remains a while longer.
; 6)   The WRITE TO ROM at $0000 by SKIP_CONS has been avoided by improving
;      the way constants are stored and indexed.
;      The WRITE TO ROM by the SCROLL routine (credit: P.Giblin) has been
;      avoided, as suggested, by ensuring that the full 24 lines are never
;      scrolled.
; 7)   The unimplemented e-to-fp calculator instruction could be removed by
;       assigning $3C to 're-stack'.  Five calculator routines would require
;       alteration. This would gain two extra bytes of ROM space but has not
;       yet been done.
; 8)    The INKEY$#0 Error. This could apply to any stream although streams 0
;       and 1 read from the keyboard by default. If the selected stream has
;       been attached to the keyboard then the null string is almost always
;       returned. The read_in routine correctly cancels any keypress as we are
;       not interested in what was pressed, perhaps, half an hour ago. However
;       there is hardly anytime for an interrupt to occur before the channel
;       is read. Fixed by testing for channel 'K' and executing a HALT if so.
;       INKEY$#0 is not the same as INKEY$ as the latter always reads the
;       keyboard directly whereas using streams has to take REPDEL and REPPER
;       into account e.g. 10 PRINT ; CODE INKEY$#0 ; " "; : GO TO 10
;
; Miscellaneous BUGS and features.
;
; 1)    In graphics mode, keys V, W, X, Y and Z give inappropriate keywords.
;       Fixed by not storing key if higher than 'U'.
; 2)    USR-$ contains a double check on number of UDGs. First check removed.
;       It would be required if there were 26 UDGs and so it may be put back.
; 4)    A typo like LIST 40000 was silently changed to LIST 7232. As an error
;       is now given, the modifying code (AND $3F) has been removed.
; 3)    ZX81 keywords were used in Spectrum error messages. Fixed in error.
;       Not possible to add a space to GOSUB without exceeding 32 characters.
;
; ------------------------------------------------------------------------------

; 06-SEP-2002
; RST 30H vacated by making BC_SPACES a subroutine.
; The ERROR restart is moved to $0030 to avoid paging in Interface 1 while,
;  at the same time, allowing access to its hardware.
; RST 08H made a User Restart with a JP to three unused system variables
;  starting at the old P-POSN. This idea later scotched.
; The NMI handler is located in the other 5 bytes.
; Routine PO_ATTR has an EX DE,HL instruction added to return the attribute
;  address in DE. (see next)
; Routine OUT_FLASH rewritten to print the character and then set the FLASH
;  bit of the attribute address.
; Routine CL_ATTR rearranged to perform attribute calculation last - providing
;  a new subroutine CL_ATTR2 which is called twice where a similar sequence
;  of instructions used to be.

; 07-SEP-2002
; ROM usage reduced by reducing absolute jumps. etc.
; The same RASP routine was used in two places and this has been made a
;  subroutine.
;  Note. there is now one MORE spare byte than in the standard ZX Spectrum.
; There are 1172 unused bytes.
;  16 spare bytes moved to area before Cassette Interface to make addresses
;     04D8 and 056A the same as standard ROM as these are trapped by emulators
;     to SAVE and LOAD to tape.
; 240 spare bytes moved to area surrounding $1708 to prevent Interface 1 paging.

; 08-SEP-2002
; There are now 20 more bytes of free ROM space than in the standard Spectrum
;  and that is despite writing an optimized TEST_5_SP routine.
; Optimized STACK_BC so IY not initialized every time. Result pointer set by
;  a faster method.
; Optimized sto_mem_x so that memory not checked when removing a value from the
;  calculator stack.

; 09-SEP-2002
; Optimize FP_TO_BC and all routines that call it - FP_TO_A etc. by setting HL
;  to the initial value by a faster method.
; Optimize FP_TO_A too. Anything involving the calculator stack has to be
;  optimal.

; 10-SEP-2002
; Noticed EX AF,AF' is little used outside the cassette interface and in some
;  places it would be faster than PUSH/POP AF. Not very many.
; Corrected error introduced by pruning in GET_HLxDE.

; 11-SEP-2002
; remove redundant code from -65536 fix and optimize to avoid machine stack.
; Spare locations = 1207 bytes

; 12-SEP-2002
; Put back the five bytes before MULT_RSLT and document so I don't remove again.
; Concentrate on testing this stage.

; 13-SEP-2002
; superfluous instruction  removed from PIXEL_ADD
; 50 more bytes of spare ROM space than standard ROM.

; 14-SEP-2002
; THE OPEN_P routine made generic and called OPEN_ALL. It is only necessary
; to set IX to the channel data before calling it.

; 15-SEP-2002
; Add RS232 and Network channels. i.e. "B", "T" and "N".
; Less than fourteen bytes spare.
; Requires some tidying up.  Stuff like OPEN #7,"N:64" requires implementation.
; Also a few flag setting routines. Looks promising though.
; It is possible to use PRINT, LIST, INPUT and INKEY$ with new channels but
; not SAVE, LOAD, MERGE and VERIFY. Yet.
; The microdrives don't stand a chance but this was all the adverts ever
; promised.  Only one machine on the network requires a mass storage device.
; Commands MOVE, ERASE, CAT and FORMAT are not implemented.
; The FORMAT command would be useful for altering the BAUD rate and setting
; the network station number.

; 16-SEP-2002
; Some problems when breaking into INPUT. Debugging code is in lower case.
; Document to help trace what's happening.

; 17-SEP-2002
; Discover minor bug in the Interface 1 (and Discovery Disk Interface) at the
; end of GET_NBLK but my own bug eludes me. (BREAK message not being cleared)
; Find the bug in this ROM. Hurrah! I need to reset bit 3 of TV_FLAG before
; entering the editor.

; 18-SEP-2002
; Complete the documentation of text channel.
; I've worked out where the new string syntax should be enforced. e.g. OPEN "N2"
; In the CLASS-0A routine. The runtime path would populate D_STR1. A lot easier
; than what I was contemplating and the effect is global. i.e. on all CLASS_0A
; strings.

; 19-SEP-2002
; Alter 'sqr' so that IY not used. Improve comments. Start FILE_DESC.

; 20-SEP-2002
; Clarify documentation and increase ROM space.

; 21-SEP-2002
; Removed the 'SUB 08' from multi comparisons calculator routine.
; Re-arranged space between restarts.
; FREE memory is now PRINT 65536 - USR 93.
; 22-SEP-2002
; Documented 'truncate'. Found a redundant byte and managed to save another.
; Incorporate the network SEND_NEOF routine. Also flush ZX Printer buffer.
; Use 2 as default Iris instead of 0 (broadcast). Input gives a satisfying
; 'End of File' report now.

; 23-SEP-2002
; Test flags before branching in ED-KEYS saves 8 bytes.

; 24-SEP-2002
; Modify KEY-INPUT so it recognizes the prompt situation. Get rid of temporary
; routine CONS_IN.  Incorporate FILE_SPEC into CLASS_0A.
; 3 ROM bytes spare. Some syntax improvements to do. For instance
; OPEN #7,"printer"  now passes through which it shouldn't.
; OPEN #7,"n:64" etc. now works.

; 25-SEP-2002
; Pressed ahead and made syntax rock-solid. OPEN works fine. Did the FORMAT
; command while in the mood. Well there are now 53 bytes over the 16K limit.
; Always amazing. Found some nice similarities in the T_CHAN/TV_DATA code.
;
; The only way I could get under 16K was to go back to the old sqr routine :-(
; This was, however, merely an indulgence as I believe this was once in
; the ZX81 until the exact same situation arose as has occured now.
;
; Unless there are bugs, I doubt there will be any more updates for months.
;
; The only outstanding task is to SAVE/LOAD programs using the new channels.
;
; I stuck the 2-byte STOP command between the restarts eventually. I still
; have 5-byte and 3-byte unused sections in there. Mail me if you can think of
; a use for them - geoff{at}wearmouth{dot}demon{dot}co{dot}uk
;
; Somewhat belatedly PEEK 9 gives release number - currently 31.

; 26-SEP-2002
; Corrected errors in proposed syntax.

; 01-OCT-2002
; Abandoned Interface 1 compatible SAVE/LOAD syntax
; The "new" syntax will be OPEN #7,"n:2" : SAVE #7; "prognam" : CLOSE #7

; 02-OCT-2002
; Implemented SAVE/LOAD/MERGE/VERIFY from network and RS232 (both untested).
; Changed Error message for error code R: to just 'Loading error'
; Fixed Graphics mode bug caused by ROM space pruning (credit: Andrew Owen).

; 03-OCT-2002
; Saving and Listing to RS232 seems to work OK but not Loading.
;  This update contains a partial fix.

; 06-OCT-2002
; This version has 10 spare ROM bytes.

; 08-OCT-2002
; This has about 20 spare bytes and isolates $1708.

; 13-OCT-2002
; Improve documentation.
; Apply ROM space saving techniques from James Smith.

; 17-OCT-2002
; Apply more ROM space saving techniques from James Smith.
;  MAKE_ROOM now increments HL which was almost always the next instruction.
;  Trying to get enough room to bring back the fast square roots and tidy up.

; 18-OCT-2002
; Isolate location $1708 again.

; 27-OCT-2002
; Add the FORMAT routines of James Smith.
; About seven bytes overdrawn.

; 28-OCT-2002
; reduce object code to 16384 bytes using techniques provided by James.

; 09-NOV-2002
; Simplify K-DECODE to solve repeating key problem. (Reported by Andrew Owen)
; Rectify CLOSE_A to support 13 ZX printer buffers.

; 10-NOV-2002
; Change FORMAT command to use FORMAT "channel specifier"; number
; Created new class routine CLASS_0C to allow choice of separators.
; Make OPEN a CLASS-05 routine and allow OPEN #3,"p" and
; OPEN #8,"n",6 with full syntax checking
; This creates almost enough room to bring back the fast square roots.

; 11-NOV-2002
; Exploit the new CLASS_0C and EXPT_SEP routines to conserve ROM space.

; 13-NOV-2002
; Optimize Newton Raphson square roots but use old ones for now, as although
; the new ones fit, they leave little room for development. 59 bytes spare.

; 15-NOV-2002
; At the suggestion of James Smith, now that everything is in one ROM, use
; the PO_SAVE routine for recursive printing of spaces in the text channel TAB
; and comma control routines.  So create new routine PO_SV_SP as the preceding
; instruction loads A with a space.  What James can save I can squander.
; Rejoice in FORMAT "k",<pip> to set the keyboard beep.  On second thoughts...

; Output from the RealSpec emulator SERIAL.BIN file.
;  10 OPEN #4,"t"
;  20 LIST #4
;  30 PRINT #4,,"Hi"
;  40 PRINT #4;TAB 17;"There"
;  50 CLOSE #4
;               Hi
;                There
; --------------------------------------------------
;
; 17-NOV-2002
; PO_RIGHT shortened. FORMAT_K removed :-)
;
; 22-NOV-2002
; 'exchange' shortened, Some new routines to modularize common code.
;  100 bytes free

; 24-NOV-2002
; Added verification from RS232 and Network.
;  ED-RIGHT feature fixed. Reported by Andrew Owen.
;  Emulator tape support abandoned. Just snapshots (.SNA) and not those from a
;  normal Spectrum.

; 27-NOV-2002
; Fixed a ZX Printer 'feature' revealed by the new VBSpec emulator .
; 30-NOV-2002
; ZX Printer location $0F24 made standard for new SPIN emulator (fast mode).

; 01-DEC-2002
; Add 'CLEAR #' command to clear all streams.
;  Any ZX Spectrum buffers are flushed first which is interesting in the new
;  paperless ZX environment.

; 03-DEC-2002
; Woohoo! a CAT command. The Spectrum should have had this as standard.
;  This version catalogs the streams to the screen.
;  Not a lot of detail but give me the space.  Ten bytes free.

; 04-DEC-2002
; Banner above the CAT

; 07-DEC-2002
; CAT with free memory report. Looks good.
;  Once you've used this, it is difficult to give it up.
;  I wonder...

; 08-DEC-2002
; OPEN #7,"n" is now rejected, as previously,  without a station but there is
;  no room for a 'Missing station number' report.

; 10-DEC-2002
; Minor improvements.  ERASE gives error and not CAT.

; 14-DEC-2002
; More tweaks and document flags. Enter test phase.

; 15-DEC-2002
; Enough room for the fast square roots. Even 'ln' has been optimized.

; 22-DEC-2002
; Tidy and format code.

; 23-DEC-2002
; Trim COPY-BUFF by one byte and clean up.

; 25-DEC-2002
; Fix bug in standard ROM at start of INPUT. Stream 1 is designated as the
; user's input stream as per original designer's comments.

; 26-DEC-2002
; Allow OPEN #0,"n";2 so that commands can be accepted from another Spectrum
; as per Steven Vickers's remarks in the Pitman Pocket Guide.
; The network now sends the buffer on receipt of a carriage return.
; Scrapped the FREE MEMORY TEST - CAT will do just fine.

; 28-DEC-2002
; The NMI service routine now closes stream 1 in a proper fashion without
; incurring a memory leak.

; 29-DEC-2002
; Correct a pathing error in the revised scrolling routine.
; Tidy up initialization of BAUD rate.  More testing.
; Correct stack corruption in DISP_MSG.  Loads from network with messages now.
; Continue testing LOAD and SAVE.
; Commands LOAD, MERGE, VERIFY and SAVE all work ok from streams.

; 01-JAN-2003
; Free up some spare space. TAG important labels.
; 09-JAN-2003
; Remove RST 18H after scanning because it's there.  Freed bytes not used.
; Tidy up CO_TEMP some of which had been made obsolete earlier.

; 10-JAN-2003
; Found a natural subroutine to check for a right-hand bracket.

; 23-JAN-2003
; Correct letter in BCHAN_DAT.
; Correct label in BCHAN_OUT - Saves and Loads OK to RS232 "B" in WinZ80.
; Note. this only works if the Interface1 hardware is selected - the Interface
; ROM paging has been avoided for this reason.

; 02-FEB-2003
; Network header checksum created (accidentally omitted) one less byte of RAM.
; Cleaner NMI handler written performs a warm reset regaining standard memory.
; Restores all initial channels without memory leaks.
; Reclaims any dynamic buffers attached to them.
; It then performs a warm reset by joining at the initialization of CHARS.
; This code which appears clumsy in the standard Spectrum finally makes sense.
; Then invokes MAIN_G directly. Tested WinZ80.
; Also ROM tested in new Spectaculator 4.0 and ZX Printer routines work fine.
; Also NMI Warm Reset tested with SPIN emulator - F5 key - works great.

; 23-FEB-2002
; Fixed INKEY$#0 Error. Credit Toni Baker.
; Although Vickers says that this is intended for channels other than the
; keyboard, it should really give some functionality when used with any device.
; The returned key value is of little use and subject to the values of REPPER
; and REPDEL and not the same as INKEY$ which always reads the keyboard
; directly.
```