

```

;=====
;
;
; An Assembly Listing of the ROM of the Sinclair ZX80
;-----
;
;-----
; Last updated: 12-SEP-2002
;-----
;
;
; Note. This is not the original text file, which was
; written by John Grant in 1979, but a file that
; performs a similar function in that it assembles
; a 4K ROM file that may be used in ZX80 emulators.
;
;
; The resultant ROM file is identical to the original
; and a comparison between the object code and the original
; is made as part of the process of uploading this file.
;
;
; It would be desirable that the original file be published but, until such
; time, this file may serve as a poor substitute.
;
;
; Actually I learn that the complete Assembly Listing was published, with
; "Designer's Annotations", in 1980. Also in that year, appeared "The ZX80
; Monitor Listing" by Ian Logan, published by LINSAC.
;
;=====

```

```

#define DEFB .BYTE      ; TASM cross-assembler definitions
#define DEFW .WORD
#define EQU .EQU

```

```

;=====
;
;
; To do justice to the original program it is desirable
; that, while the instructions should not be over
; commented, what is appended should be of high quality.
;
;
; Send details of any improvements/corrections to
; geoff@wearmouth.demon.co.uk.spamnothanks
; All contributions will be credited.
;
;
; File incorporates contributions from
; Peter Liebert-Adelt
;
;
; and borrows from the writings of
; Wilf Rigter,
; Dr Ian Logan,
; Dr Frank O'Hara.
;
;=====

```

```

.ORG    $0000

```

```

;-----
; THE 'START'
;-----

```

```

;; START
L0000: LD    HL,$7FFF      ; top of possible RAM.

```

```

; (highest integer is 32767).
LD    A,$3F      ; page before RAM.
JP    L0261      ; forward to RAM-FILL.

;-----
; THE 'ERROR' RESTART
;-----

;; ERROR-1
L0008: POP    HL      ; drop the return address.
      LD     L,(HL)    ; fetch the error code after RST 8.
      BIT    7,(IY+$00) ; test ERR_NR for value $FF (OK)
      JR     L0013     ; forward to continue at ERROR-2.

;-----
; THE 'PRINT A CHARACTER' RESTART
;-----

;; PRINT-A
L0010: JP     L0560     ; jump forward immediately to PRINT-A-2

; ---

; A continuation of the previous Error restart.

;; ERROR-2
L0013: RET     Z        ; return if $FF - OK.

      LD     (IY+$00),L ; else set system variable ERR_NR
      RET                     ; return.

;-----
; THE 'COLLECT NEXT CHARACTER OR SPACE' RESTART
;-----
; This will collect any next character including space (zero).

;; NXT-CH-SP
L0018: JR     L0052     ; forward to CH_ADD+1

; ---

; This subroutine will collect the character at the current character address
; searching for the next non-space character should the fetched character be
; a space.

;; get-char
L001A: LD     HL,($4026) ; get pointer from CH_ADD
      LD     A,(HL)      ; fetch addressed character.

; This subroutine tests the current character in the accumulator retrieving
; the next non-space character should the accumulator contain a space

;; TEST-CHAR
L001E: AND     A          ; test for space (zero).
      RET     NZ         ; return if not a space.

;-----
; THE 'COLLECT NEXT VALID CHARACTER' RESTART
;-----

;; NEXT-CHAR

```

```

L0020: CALL  L0052      ; routine CH_ADD+1
        JR    L001E      ; loop back to TEST-CHAR until valid

; ---

; This subroutine advances the character pointer and evaluates the following
; expression.
; It is called twice with CH_ADD addressing the '(' character

;; EVAL-EXPR
L0025: CALL  L0055      ; routine CH_ADD_LP

; -----
; THE 'SCANNING-CALCULATOR' RESTART
; -----

;; SCAN-CALC
L0028: CALL  L001A      ; routine get-char.
        LD    B,$00      ; set B to zero as a starting
                        ; priority marker.
        JP    L09E1      ; jump forward to SCANNING

; -----
; THE 'MAKE BC SPACES' RESTART
; -----

;; BC-SPACES
L0030: CALL  L094F      ; routine TEST-ROOM
        RET    NC        ; return if not enough room.

        PUSH  BC        ; save number of bytes required.
        JP    L0CF3      ; jump forward to RESERVE

; -----
; THE 'MASKABLE INTERRUPT' ROUTINE
; -----
; Note. the maskable interrupt is concerned with generating the TV picture,
; one of the main tasks in the ZX80. This requires some understanding of
; how the video hardware interacts with the system and part of the process
; is to present to the Z80 chip a phantom display file in the upper
; unpopulated 32K of memory. This topsy-turvy display file
; executes characters like "HELLO WORLD" as NOP instructions but recognizes
; a newline ($76) as a true HALT instruction.

; The video hardware sniffs the databus and grabs the data as it flies by
; sending it on to the shifting circuits. The I register permanently holds
; $0E. The video circuitry uses this register and the lower six bits of the
; character to index into the character set bitmaps at the end of this ROM,
; at $0E00, and so cobble together a scan-line.

; If bit 7 of the character latch is set, then the serial video data is
; inverted so that any character in the range 127-191 appears as the inverse
; of normal characters 0 - 63.

; For a proper explanation of this system, I recommend Wilf Rigter's
; online documentation, available from several indexed sites.
; I have borrowed a few comments from that file to remind myself of what
; is happening. I have indicated where the Z80 instructions should be
; read in conjunction with Wilf's file by using a double semi-colon.

; On entry, B holds the line number and C the number of the scanline.

```

```

;; MASK-INT
L0038: DEC   C           ;; decrement the scan line counter in register C.
      JP    NZ,L0045     ;; JUMP to SCAN-LINE : repeats 8 times for each
                        ;; row of characters in DFILE.

      POP   HL           ;; point to the start of next DFILE row
      DEC   B            ;; decrement ROW counter
      RET   Z            ;; return if zero to

      SET   3,C          ;; load scan line counter with 08 was 00.

;; WAIT-INT
L0041: LD     R,A         ;; load refresh register with value $DD.
      EI             ;; enable interrupts.
      JP    (HL)         ;; jump to execute the NOPs in DFILE
                        ;; terminated by a NEWLINE/HALT instruction.

; ---

;; SCAN-LINE
L0045: POP    DE          ;; discard return address.
      RET    Z           ;; delay (Zero never set)

      JR     L0041       ;; back to WAIT-INT above

; -----
; THE 'EVALUATE BRACKETED EXPRESSION' SUBROUTINE
; -----
; This subroutine is used when an opening bracket is encountered to evaluate
; the expression within. It is called from LOOK-VARS when an integral function
; or array is encountered and recursively from within SCANNING when any
; bracketed argument or sub-expression is encountered.

;; BRACKET
L0049: CALL   L0025       ; routine EVAL-EXPR
      LD     A,(HL)       ; fetch subsequent character
      CP     $D9          ; is character a ')' ?
      JP     NZ,L08AE     ; jump to INS-ERR with other characters.

; else continue and get the character after the ')' ...

; -----
; THE 'INCREMENT CH_ADD' SUBROUTINE
; -----

;; CH_ADD+1
L0052: LD     HL,($4026)  ; fetch character address from CH_ADD

;; CH_ADD_LP
L0055: INC     HL         ; increment the pointer.
      LD     ($4026),HL   ; set system variable CH_ADD
      LD     A,(HL)       ; fetch the addressed value.
      CP     $B0          ; is character inverse 'K'
      RET    NZ           ; return if not.      >>

      LD     ($4004),HL   ; set P_PTR system variable
      BIT    7,(IY+$19)   ; test FLAGX - will be set if K-mode
      JR     Z,L0055      ; back to CH_ADD_LP if not K-mode

L0066: SET    2,(IY+$01)  ; update FLAGS set K mode.

```

JR L0055 ; back to CH_ADD_LP

; Note there is no NMI routine at L0066.

; -----
; THE 'KEY' TABLE
; -----

; The Key Table is indexed with a key value 1-78.

; -----
; THE 39 'UNSHIFTED' KEYS
; -----

:: MAIN-KEYS

L006C: DEFB \$3F ; Z

DEFB \$3D ; X
DEFB \$28 ; C
DEFB \$3B ; V

DEFB \$26 ; A
DEFB \$38 ; S
DEFB \$29 ; D
DEFB \$2B ; F
DEFB \$2C ; G

DEFB \$36 ; Q
DEFB \$3C ; W
DEFB \$2A ; E
DEFB \$37 ; R
DEFB \$39 ; T

DEFB \$1D ; 1
DEFB \$1E ; 2
DEFB \$1F ; 3
DEFB \$20 ; 4
DEFB \$21 ; 5

DEFB \$1C ; 0
DEFB \$25 ; 9
DEFB \$24 ; 8
DEFB \$23 ; 7
DEFB \$22 ; 6

DEFB \$35 ; P
DEFB \$34 ; O
DEFB \$2E ; I
DEFB \$3A ; U
DEFB \$3E ; Y

DEFB \$76 ; NEWLINE ED-ENTER
DEFB \$31 ; L
DEFB \$30 ; K
DEFB \$2F ; J
DEFB \$2D ; H

DEFB \$00 ; SPACE
DEFB \$1B ; .
DEFB \$32 ; M
DEFB \$33 ; N
DEFB \$27 ; B

; -----

; THE 39 'SHIFTED' CODES

;-----

DEFB \$0E ; ':'
DEFB \$D7 ; ';'
DEFB \$0F ; '?'
DEFB \$DF ; '/'

DEFB \$09 ; mosaic \$09
DEFB \$08 ; mosaic \$08
DEFB \$06 ; mosaic \$06
DEFB \$07 ; mosaic \$07
DEFB \$0B ; mosaic \$0B

DEFB \$02 ; mosaic \$02
DEFB \$03 ; mosaic \$03
DEFB \$04 ; mosaic \$0A
DEFB \$05 ; mosaic \$04
DEFB \$0A ; mosaic \$05

DEFB \$DB ; 'NOT'
DEFB \$E0 ; 'AND'
DEFB \$D5 ; 'THEN'
DEFB \$D6 ; 'TO'
DEFB \$72 ; cursor left

DEFB \$77 ; [RUBOUT]
DEFB \$74 ; [HOME]
DEFB \$73 ; cursor right
DEFB \$70 ; cursor up
DEFB \$71 ; cursor down

DEFB \$DE ; '*'
DEFB \$D9 ; ')'
DEFB \$DA ; '('
DEFB \$0D ; '\$'
DEFB \$01 ; ''

DEFB \$75 ; [EDIT]
DEFB \$E3 ; '='
DEFB \$DD ; '+'
DEFB \$DC ; '-'
DEFB \$E2 ; '**'

DEFB \$0C ; uk currency symbol
DEFB \$D8 ; ''
DEFB \$E4 ; '>'
DEFB \$E5 ; '<'
DEFB \$E1 ; 'OR'

;-----

; THE 'TOKEN' TABLE

;-----

;; TKN-TABLE

L00BA: DEFB \$D4 ; chr\$ 212 - the threshold character

; tokens below this are printed using

; the next character

DEFB \$8F ; '?' + \$80

DEFB \$81 ; '' + \$80

```

DEFB $39,$2D,$2A,$B3 ; THEN
DEFB $39,$B4 ; TO
DEFB $99 ; ;
DEFB $9A ; ,
DEFB $91 ; (
DEFB $90 ; )
DEFB $33,$34,$B9 ; NOT
DEFB $92 ; -
DEFB $93 ; +
DEFB $94 ; *
DEFB $95 ; /
DEFB $26,$33,$A9 ; AND
DEFB $34,$B7 ; OR
DEFB $14,$14+$80 ; **
DEFB $96 ; =
DEFB $97 ; <
DEFB $98 ; >
DEFB $31,$2E,$38,$B9 ; LIST
DEFB $37,$2A,$39,$3A,$37,$B3 ; RETURN
DEFB $28,$31,$B8 ; CLS
DEFB $29,$2E,$B2 ; DIM
DEFB $38,$26,$3B,$AA ; SAVE
DEFB $2B,$34,$B7 ; FOR
DEFB $2C,$34,$00,$39,$B4 ; GO TO
DEFB $35,$34,$30,$AA ; POKE
DEFB $2E,$33,$35,$3A,$B9 ; INPUT
DEFB $37,$26,$33,$29 ; ...
DEFB $34,$32,$2E,$38,$AA ; RANDOMISE
DEFB $31,$2A,$B9 ; LET
DEFB $8F ; '?' + $80
DEFB $8F ; '?' + $80
DEFB $33,$2A,$3D,$B9 ; NEXT
DEFB $35,$37,$2E,$33,$B9 ; PRINT
DEFB $8F ; '?' + $80
DEFB $33,$2A,$BC ; NEW
DEFB $37,$3A,$B3 ; RUN
DEFB $38,$39,$34,$B5 ; STOP
DEFB $28,$34,$33,$39,$2E ; ...
DEFB $33,$3A,$AA ; CONTINUE
DEFB $2E,$AB ; IF
DEFB $2C,$34,$00,$38,$3A,$A7 ; GO SUB
DEFB $31,$34,$26,$A9 ; LOAD
DEFB $28,$31,$2A,$26,$B7 ; CLEAR
DEFB $37,$2A,$B2 ; REM
DEFB $8F ; '?' + $80

```

```

; -----
; THE 'DISPLAY' ROUTINES
; -----

```

```

; ->

```

```

;; DISP-1
L013C: CALL L01AD ; routine DISP-2

```

```

; The initial entry point

```

```

;; KEYBOARD
L013F: LD B,$08 ; (7) set counter to 8

```

```

;; KB-1

```

L0141: DJNZ L0141 ; (13,8) and loop back 7 times. (7*13+8)

; "WASTE 99 T-STATES"

:: KB-2

L0143: LD HL,(\$401E) ; (16) fetch two-byte FRAMES value.

INC HL ; (6) increment

LD (\$401E),HL ; (16) and store in FRAMES again.

; now read the keyboard

LD HL,\$FFFF ; (10) prepare a buffer

LD B,\$FE ; (7) set B to \$FE

LD C,B ; (4) now BC is \$FEFE - slightly slower than
; the equally time-critical LD BC,\$FEFE (10)
; that is used in the ZX81 ROM.

IN A,(C) ; (12) now read port \$FEFE the half-row with
; the shift key.

; "START FRAME SYNC"

; START COUNTING

OR \$01 ; (7) set the rightmost bit so as to ignore
; shift.

:: EACH-LINE

L0154: OR \$E0 ; [7] OR 11100000.

LD D,A ; [4] transfer to D.

CPL ; [4] complement - only bits 4-0 meaningful now.

CP \$01 ; [7] sets carry if A is zero.

SBC A,A ; [4] \$FF if \$00 else zero.

OR B ; [4] \$FF or port FE,FD,FB...

AND L ; [4] unless more than one key, L will still
; be \$FF if more than one key pressed A
; is now invalid

LD L,A ; [4] transfer to L.

; now consider the column identifier.

LD A,H ; [4] will be \$FF if no previous keys.

AND D ; [4] 111xxxx

LD H,A ; [4] transfer A to H

; since only one key may be pressed, H will, if valid, be one of

; 11111110, 11111101, 11111011, 11110111, 11101111

; reading from the outer column, say Q, to the inner column, say T.

RLC B ; [8] rotate the 8-counter/port address.

; sets carry if more to do.

IN A,(C) ; [12] read another half-row.

; all five bits this time.

JR C,L0154 ; [12],(7) loop back, until done, to EACH-LINE

; (658 T-states).

; the last row read is SHIFT,Z,X,C,V for the second time.

RRA ; (4) test the shift key - carry reset if
; pressed.


```

;; KB-3
L0168: RL    H        ; (8) rotate H to the left picking up the carry.
        ; giving column values -
        ; $FD, $FB, $F7, $EF, $DF.
        ; or $FC, $FA, $F6, $EE, $DE if shifted.

; we now have H identifying the columns and L identifying the row of the
; keyboard matrix.

; This is a good time to test if this is an American or British machine.
; The US machine has an extra diode that causes bit 6 of a byte read from a
; port to be reset.

```

```

RLA          ; (4) compensate for the shift test.
RLA          ; (4) rotate bit 7 out.
RLA          ; (4) test bit 6.
SBC  A,A     ; (4) $FF or $00 (USA)
AND  $18     ; (7) and 24
ADD  A,$20   ; (7) add 32

```

```

; gives either 32 (USA) or 56 (UK) blank lines above the TV picture.
; This value will be decremented for the lower border.

```

```

LD    ($4023),A    ; (13) place margin in RESULT_hi.

```

```

; The next snippet tests that the same raw key is read twice in succession.
; The first time through, the routine uses a character address value,
; which is inappropriate to match against a key value, but the next time
; through it matches the key value it placed there on the first pass.
; Seems to be 713 T-states.
;
; "717 T-STATES SINCE START OF FRAME SYNC, 545 BEFORE END"

```

```

LD    BC,($4026)   ; (20) fetch possible previous key value from
        ; CH_ADD
LD    ($4026),HL   ; (16) put the fresh key value in CH_ADD.

LD    A,B         ; ( 4) fetch high byte.
ADD  A,$02        ; ( 7) test for $FF, no-key which will set
        ; carry.

SBC  HL,BC        ; (15) subtract the two raw keys.
EX   DE,HL        ; ( 4) result, possibly zero, to DE.

LD    HL,$4022     ; (10) now address system variable RESULT.
LD    A,(HL)       ; ( 7) load A from RESULT_lo.
OR    D           ; ( 4) check the
OR    E           ; ( 4) subtraction result.
RET   Z           ; ( 5,11) return if all three zero.    >>>

```

```

; T-states = 96 so far
; proceed to debounce. The 'no-key' value $FF must be returned five times
; before a new key is accepted above.
; Holding down a key causes the shift counter to be maintained at five.
; The initial state of RESULT is unimportant.

```

```

LD    A,B         ; ( 4) fetch hi byte of PREVIOUS key code.
CP    $FE         ; ( 7) sets carry if valid -
        ; $FD, $FB, $F7, $EF, $DF
SBC  A,A         ; ( 4) gives $FF if pressed or $00 if no-key.

LD    B,$1F       ; ( 7) prepare the shift counter

```

```

; (and also the timed delay)

OR   (HL)      ; ( 7) OR with RESULT_lo
AND  B         ; ( 4) limit the count to five set bits.
RRA      ; ( 4) 'shift' to right
LD   (HL),A    ; ( 7) place result in RESULT_lo

DEC  B         ; ( 4) adjust the delay counter B to thirty.

; t states = 48 ( Total 96+48=144)

;; KB-4
L0194: DJNZ  L0194      ;; (13,8) wait a while looping to KB-4
      ;; equals 13*29+8 = 385

      ; "FRAME SYNC ENDS AT NEXT M1"

OUT  ($FF),A     ;; (11) stops the VSYNC pulse

LD   A,$EC      ;; ( 7) the value for R register
LD   B,$19      ;; there are 25 HALTs including the initial
      ;; one.
LD   HL,($400C)  ;; point HL to D-FILE the first HALT
      ;; instruction.
SET  7,H        ;; now point to the DFILE echo in the
      ;; top 32K of address space.

CALL  L01AD      ;; routine DISP-2

LD   A,$F3      ;; prepare to set the R refresh register to $F3.
INC  B          ;; increment the line count
DEC  HL         ;; decrement screen address.
DEC  (Y+$23)    ;; decrement RESULT_hi the blank line counter.
JR   L013C      ;; back to display and read

; ---

;; DISP-2
L01AD: LD   C,(Y+$23)  ;; load C the col count from RESULT_hi.

LD   R,A        ;; R increments with each opcode until A6
      ;; goes low which generates the INT signal.

LD   A,$DD      ;; set the left margin of all other lines.
      ;; loaded later to R - the incremental refresh
      ;; register.

EI          ;; with R set up, enable interrupts.

JP   (HL)       ;; jump to execute the echo DFILE starting with
      ;; HALT and waits for the first INT to
      ;; come to the rescue.

; -----
; THE 'SAVE' COMMAND ROUTINE
; -----
; There isn't a program name involved.
; The routine saves the System Variables, Program Area and BASIC Variables.
; One of the five System commands that cannot be used from within a program.

;; SAVE

```

```

L01B6: POP  DE      ; discard return address.
        LD   DE,$12CB ; timing value of 5 seconds for leader.

;; SAVE-1
L01BA: LD   A,$7F      ; read port $7FFE.
        IN   A,($FE)    ; all 16 bits are placed on address bus.
        RRA           ; test for the space key.
        JR   NC,L0203    ; forward, if pressed, indirectly to MAIN-EXEC.

;; SAVE-2
L01C1: DJNZ  L01C1      ; delay self-looping to SAVE-2

        DEC  DE          ; decrement
        LD   A,D          ; and test
        OR   E            ; for zero.
        JR   NZ,L01BA     ; back if not zero to outer delay loop SAVE-1.

        LD   HL,$4000     ; commence saving at start of RAM.

;; SAVE-3
L01CB: LD   DE,$F808     ; register E counts the 8 bits.
        ; $F8 is first delay.

;; EACH-BIT
L01CE: RLC   (HL)        ; spin the actual program byte.
        SBC  A,A          ; $FF or $00.
        AND  $05          ; $05 or $00.
        ADD  A,$04        ; $09 or $04.
        LD   C,A          ; timer to C.
        ; a set bit has a pulse longer than
        ; an unset bit.

;; SAVE-4
L01D6: OUT   ($FF),A      ; pulses

        LD   B,$24        ; delay counter.

;; SAVE-5
L01DA: DJNZ  L01DA        ; self loop for delay to SAVE-5

        LD   A,$7F        ; read the space row and hold for later.
        IN   A,($FE)      ; also ...

        LD   B,$23        ; another delay counter.

;; SAVE-6
L01E2: DJNZ  L01E2        ; self loop for delay2 to SAVE-6

        DEC  C            ; decrement pulse counter
        JR   NZ,L01D6     ; back while more to SAVE-4.

        LD   B,D          ; a terminating delay - D is zero (256).

;; SAVE-7
L01E8: NOP           ; 4 T-states.
        DJNZ L01E8        ; execute the NOP 256 times.

        LD   D,$FE        ; subsequent timing value
        DEC  E            ; decrement the 8 counter.
        JR   NZ,L01CE     ; back if more to EACH-BIT.

        RRA              ; test for space key pressed at last test.

```

```

JR    NC,L0203      ; forward, if so, indirectly to MAIN-EXEC.

CALL  L01F8         ; routine TEST-END does not return if at
                    ; the end.                                >>

JR    L01CB         ; else back to do another byte.

; ---

; This subroutine is used by both the SAVE and LOAD command routines
; to check when the required area has been completed and to then make an exit
; from the called loop.
; Note. that for the LOAD command the value of E_LINE is not that at the outset
; of the LOAD command but at the start of the command that saved the section.
; The first bytes to be loaded are the System Variables and E_LINE will be the
; eleventh and twelfth bytes to be loaded. The low byte is read in before the
; high byte so after the low byte is read in, E_LINE is in an indeterminate
; state. Hence E_LINE_hi is incremented at the outset to avoid a premature
; end to loading.

;; TEST-END
L01F8: INC  HL      ; increase pointer.
      EX   DE,HL    ;
      LD   HL,($400A) ; load HL with E_LINE - the location following
                    ; the variables end-marker.
      SCF                ; force a carry when equal.
      SBC  HL,DE        ; trial subtraction.
      EX   DE,HL        ; restore pointer.
      RET  NC           ; return if more bytes to do.

      POP  HL           ; else drop the return address.

;; JUMP-EXEC
L0203: JP   L0283      ; JUMP forward to MAIN-EXEC.

; Note. the above jump could be replaced by a relative jump saving one
; instruction byte. A few other direct jumps to this destination could be
; replaced with a series of relative jumps as has been done elsewhere.

; -----
; THE 'LOAD' COMMAND ROUTINE
; -----
; A System Command to load a program from tape.

;; LOAD
L0206: POP  DE         ; discard the return address.

;; LOAD-1
L0207: LD   DE,$5712   ; set a timing constant.

;; LOAD-2
L020A: LD   A,$7F      ; read from port $7FFE.
      IN   A,($FE)     ; the keyboard row with space.
      RRA                ; test the outer key.
      JR   NC,L0203    ; back, if pressed, indirectly to MAIN-EXEC

      RLA                ; cancel the above RRA.
      RLA                ; now do an RLA to read tape signal - bit 7.

      JR   C,L0207     ; back without signal to outer loop LOAD-1.

      DEC  DE          ; decrement timer

```

```

LD    A,D        ; and test
OR    E          ; for zero.
JR    NZ,L020A    ; back if not to inner loop LOAD-2.

INC    (IY+$0B)    ; increment E_LINE_hi to prevent premature
                  ; end after loading E_LINE-lo.
                  ; see TEST-END.

LD    HL,$4000    ; start of RAM - system variables to be
                  ; overwritten.

;; LOAD-3
L0220: LD    E,$08    ; the bit counter for each byte.

;; LOAD-4
L0222: LD    A,$7F    ; test the keyboard
      IN    A,($FE)    ; reading the
      RRA          ; space key.
      JR    NC,L024D    ; forward, if space pressed, to LD-ABORT.

      RLA          ; restore to original state.
      RLA          ; now test the tape bit.
      JR    NC,L0222    ; back if ??? to LOAD-4

; start building up a byte.

      LD    C,$94    ; set timing value. The exit value of this
                  ; register determines if a bit was set or unset.

;; LOAD-5
L022F: LD    B,$1A    ; inner timer

;; LOAD-6
L0231: DEC    C        ; decrement counter.
      IN    A,($FE)    ; read the tape port.
      RLA          ; test the tape bit.

      BIT    7,C        ; test if counter above 127. A set bit.
      LD    A,C        ; save in A.

      JR    C,L022F    ; back while bit set to LOAD-5

      DJNZ   L0231    ; decrement B counter and loop while not
                  ; zero to LOAD-6.
                  ; Note. this instruction has no effect on any
                  ; flags.

      JR    NZ,L0242    ; forward if C was > $7F (with NC) to LOAD-7

      CP    $56        ; compare copy of counter to $56
      JR    NC,L0222    ; back if $56-$7F to LOAD-4

;; LOAD-7
L0242: CCF          ; else clear if from above but set carry if
                  ; branching to here.
      RL    (HL)        ; rotate the bit into position.

      DEC    E          ; decrement the eight counter
      JR    NZ,L0222    ; loop back for entire byte.

      CALL   L01F8    ; routine TEST-END quits early at end.
      JR    L0220    ; and back to load another byte.

```

```
;
;-----
; THE 'LOAD ABORT' EXIT ROUTE
;-----
; If the LOAD command has started to load data then a reset is performed.
; If it's still waiting for the leader then rejoin the main execution loop
; after restoring the location of the Edit Line to its correct value.
```

```
:: LD-ABORT
L024D: DEC  D      ; ??
        JP  P,L0000 ; a reset

        DEC  (IY+$0B) ; restore E_LINE_hi to a valid state.
        JR  L0203     ; indirect jump to MAIN-EXEC.
```

```
;
;-----
; THE 'LIST' COMMAND ROUTINE
;-----
; Another System command that can't be used from within a program.
```

```
:: LIST
L0256: RES  7,B      ; start by making the high byte,
                ; of an invalid, user-supplied,
        RES  6,B      ; line number within range $00-$3F.
```

```
; this invisible mending is inappropriate and it is preferable to tell the
; user of any typos. e.g. LIST 40000 is silently changed to LIST 7232
; when the user probably meant to type LIST 4000. However space is tight.
```

```
        LD  ($4006),BC ; set E-PPC from line number.
        POP BC         ; discard return address.
        JR  L0283      ; forward to MAIN-EXEC which produces an
                ; 'automatic listing'.
```

```
;
;-----
; THE 'INITIALIZATION' ROUTINE
;-----
; A holds $3F, HL holds $7FFF.
```

```
:: RAM-FILL
L0261: LD  (HL),$01 ; fill location with 1 (null).
        DEC HL      ; decrement address.
        CP  H       ; compare address high byte to $3F.
        JR  NZ,L0261 ; back, while higher, to RAM-FILL.
```

```
:: RAM-READ
L0267: INC  HL      ; address the next higher location.
        DEC (HL)    ; decrement to zero.
        JR  Z,L0267 ; back, if successful to RAM-READ.
```

```
; else we have encountered first unpopulated RAM location.
```

```
        LD  SP,HL   ; initialize stack pointer at end.
        PUSH AF      ; place gosub end-marker $3F??
```

```
        LD  A,$0E   ; set the I register to $0E to tell
        LD  I,A     ; the video hardware where to find
                ; the character set ($0E00).
```

```
        IM  1       ; select Interrupt Mode 1.
```

```

LD    IY,$4000    ; set IY to the start of the forty system
                  ; variables.

;-----
;
;-----
; THE 'ZX80 MEMORY MAP'
;-----
;
; There are forty ($28) system variables followed by Program area
; These are located at the start of RAM.
;
;-----+-----+-----+-----+-----+-----+-----+-----+
; |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
; | SYSVARS | Program | Variables | 80h | WKG Space | Disp File | Spare | Stack |
; |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
;-----+-----+-----+-----+-----+-----+-----+
;      ^      ^      ^      ^      ^      ^      ^
;      $4024  VARS      E_LINE  D_FILE  DF_END  SP
;                      DF_EA
;-----

LD    HL,$4028    ; set to location after sysvars.
LD    ($4008),HL  ; set the system variable VARS.
LD    (HL),$80    ; and insert variables end-marker.

INC   HL          ; address the next location.
LD    ($400A),HL  ; set the system variable E_LINE.
                  ; and continue...

;-----
; THE 'MAIN EXECUTION' LOOP
;-----
; This is the MAIN EXECUTION LOOP that handles the creation and interpretation
; of user input. The various 'subroutines' from this main loop including those
; launched from the Editing Keys Table are really just branches which all
; ultimately jump back to here. Although service routines make use of the
; machine stack, the stack is generally empty and only has one return address
; on it during command execution.

;; MAIN-EXEC
L0283: LD    HL,($400A)    ; fetch E-LINE
      LD    (HL),$B0      ; insert the character inverse 'K'.

      INC   HL            ; address the next location.
      LD    (HL),$76      ; insert a newline.

      INC   HL            ; address the next location.
      LD    ($400C),HL    ; set D-FILE to start of dynamic display file.
      LD    (IY+$12),$02  ; set DF-SZ to 2 lines.

; ->

;; AUTO-LIST
L0293: CALL  L0747        ; routine CLS sets a minimal display and
                  ; initializes screen values in registers.

      EX    DE,HL        ;
      LD    A,B          ; load line value, 23, to A.
      SUB   (IY+$12)      ; subtract DF-SZ of lower screen.
      JR    C,L02F7      ; forward if the lower screen is 24 lines

```

```

; to ED-COPY.

INC  A      ; allow for a blank line.
LD   B,A    ; place in B line

EXX      ; switch to preserve line/column values.

LD   HL,($4006) ; fetch E_PPC the current line number.
LD   DE,($4013) ; fetch the top line on screen from S_TOP.
SBC  HL,DE    ; subtract the two BASIC line numbers
EX   DE,HL    ; and bring S_TOP to HL.
JR   NC,L02B0 ; forward if current line >= top line to LIST-1.

ADD  HL,DE    ; else reform the E_PPC value
LD   ($4013),HL ; and make S_TOP the same.

;; LIST-1
L02B0: CALL L060A ; routine LINE-ADDR gets the address of the
               ; BASIC line in HL.
LD   E,$00    ; signal current line yet to be printed

;; LIST-ALL
L02B5: CALL L04F7 ; routine OUT-LINE

JR   C,L02B5  ; loop until upper screen is full to LIST-ALL.

DEC  E      ; test if current line has appeared.
JR   NZ,L02F0 ; forward to LIST-DONE if current line
               ; has appeared.

; else the current line has yet to appear.

PUSH HL      ; else save HL ( )
LD   HL,($4006) ; fetch E_PPC - the current line.
CALL L060A    ; routine LINE-ADDR in DE
POP  HL      ; restore HL

AND  A      ; prepare to subtract.
SBC  HL,DE   ; subtract setting carry.

LD   HL,$4013 ; address system variable S_TOP
JR   NC,L02D8 ; forward if E_PPC precedes to LN-FETCH

EX   DE,HL   ; else swap pointers.

LD   A,(HL)  ; pick up high byte.
INC  HL      ; address low byte.
LDI      ; copy low byte to S_TOP_lo.
LD   (DE),A  ; insert the high byte.

;; AUTO-L-J
L02D3: JR   L0293 ; back to AUTO-LIST.

; -----
; THE 'CURSOR DOWN EDITING' SUBROUTINE
; -----

;; ED-DOWN
L02D5: LD   HL,$4006 ; address system variable E_PPC
               ; and continue...

```



```
;
;-----
; THE 'LN-FETCH' SECTION
;-----
```

```
:: LN-FETCH
L02D8: LD    E,(HL)    ;
      INC    HL        ;
      LD     D,(HL)    ;
      PUSH   HL        ;
      EX     DE,HL     ;
      INC    HL        ; increment as starting point
      CALL   L060A     ; routine LINE-ADDR
      CALL   L03C2     ; LINE-NO
      POP    HL        ; restore hi pointer.
```

```
;
;-----
; THE 'LN-STORE' SECTION
;-----
; On entry, HL holds E_PPC_hi.
```

```
:: LN-STORE
L02E5: BIT    5,(IY+$19) ; test FLAGX.
      JR     NZ,L02F7    ; forward if INPUT to ED-COPY.

      LD     (HL),D      ; insert high byte
      DEC    HL          ; DECrement
      LD     (HL),E      ; insert low byte
```

```
;
;
      JR     L0293      ; back to AUTO-LIST
```

```
;
;-----
; THE 'LIST-DONE' SUBROUTINE
;-----
; When the listing is complete then the rest of the upper display is blanked,
; to erase what may have been printed during the interim, the display file
; cursor is updated and the current line is printed in the lower screen.
```

```
:: LIST-DONE
L02F0: CALL   L05C2      ; CL-EOD clear to end of upper display.

      LD     ($400E),DE   ; set lower screen position DF_EA
                        ; to end
                        ; and continue...
```

```
;
;-----
; THE 'LOWER SCREEN COPYING' SUBROUTINE
;-----
; This is called.
; When the line in the editing area is to be printed in the lower screen.
; It is by repeatedly printing the line when any key is pressed that the
; cursor for instance appears to move.
; It is called in a similar fashion to animate the input line.
```

```
:: ED-COPY
L02F7: LD     (IY+$01),$01 ; set FLAGS leading space allowed
      LD     HL,($400A)    ; E_LINE

      CALL   L07BE        ; routine MAIN-G checks syntax of line.

      LD     DE,($400E)    ; fetch start of lower screen from DF_EA
```

```

LD    B,(IY+$12)    ; fetch lines in lower screen from DF_SZ
LD    C,$01         ; set column to 1
                        ; to print an initial newline for gap?
EXX
                        ;

LD    HL,($400A)     ; fetch start of edit line from E_LINE

CALL  L0512         ; routine OUT-LINE-2 prints characters starting
                        ; with the individual digits of line number.

JR    C,L031D       ; forward with success to LINE-DONE

; else there wasn't enough room in lower screen for line.

LD    HL,$4012       ; address DF_SZ the Display Size for
                        ; the lower screen.
INC   (HL)          ; increment it.
LD    A,$18         ; load A with 24 decimal.
CP    (HL)          ; compare to DF-SZ

JR    NC,L02D3       ; indirect jump back to AUTO-LIST
                        ; if no greater than 24 lines.

LD    (HL),A        ; else limit to 24 lines.

;; LINE-DONE
L031D: CALL  L05C2     ; routine CL-EOD clears to the end of lower
                        ; screen

CALL  L013F         ; routine KEYBOARD gets key values in BC.

; now decode the value

SRA   B             ; sets carry if unshifted (bit 7 remains set)
SBC   A,A           ; $FF unshifted, else $00
OR    $26           ; $FF unshifted, else $26
LD    L,$05         ; there are five keys in each row.
SUB   L             ; set the starting point

;; KEY-LINE
L032B: ADD   A,L      ; add value 5 (or 1)
SCF                    ; carry will go to bit 7
RR    C             ; test C (which has 1 unset bit identifying row)
JR    C,L032B       ; back if carry to KEY-LINE

; if only one key pressed C should now be $FF.

INC   C             ; test for $FF
JR    NZ,L02F7       ; back if multiple keys to ED-COPY

; the high byte of the key value identifies the column - again only one bit is
; now reset.

LD    C,B           ; transfer to B
DEC   L             ; test if this is first time through
LD    L,$01         ; reduce increment from five to one.
JR    NZ,L032B       ; back if L was five to KEY-LINE

; The accumulator now holds a key value 1-78 decimal.

LD    HL,L006C - 1   ; location before the MAIN-KEYS table ($006B)
                        ; the index value is 1 - 78.

```

```

LD    E,A          ; code to E (D is zero from keyboard)
ADD   HL,DE        ; index into the table.
LD    A,(HL)       ; pick up the letter/number/.

BIT   2,(IY+$01)   ; test FLAGS K-MODE ?
JR    Z,L034D      ; skip forward if not

ADD   A,$C0        ; add 192 decimal
                        ; e.g. 'A' 38d + 192 = 230 (LIST)

CP    $E6          ; compare to 'LIST'
JR    NC,L034D     ; skip forward if command tokens to EDC-2.

LD    A,(HL)       ; else load A from HL again
                        ; (numbers and symbols)

;; EDC-2
L034D: CP    $C0    ; set the overflow flag for editing key $70-$77

        JP    PE,L035E    ; forward with range $40 - $7F to ED-KEYS

LD    HL,($4004)    ; else fetch keyboard cursor from P_PTR
LD    BC,$0001      ; one space required.
CALL  L05D5         ; routine MAKE-ROOM makes room at cursor.
                        ; note HL - first, DE - LAST

LD    (DE),A        ; and insert the keyboard character.

;; EDC-JR
L035C: JR    L02F7    ; loop back to ED-COPY

; -----
; THE 'EDITING KEYS' SUBROUTINE
; -----

;; ED-KEYS
L035E: LD    E,A      ; transfer code to E.
                        ; (D holds zero from 'keyboard')

LD    HL,L0372-$70-$70; theoretical base of ED-K-TAB $0292

ADD   HL,DE        ; index twice
ADD   HL,DE        ; as a two-byte address is required.

LD    C,(HL)       ; low byte of routine.
INC   HL
LD    B,(HL)       ; high byte of routine.
PUSH  BC           ; push routine address to stack.
LD    HL,($4004)    ; set HL to cursor from P_PTR
RET                                ; jump to routine.

                        ; Note the stack is empty.

; -----
; THE EDITING 'DELETE ONE CHARACTER' SUBROUTINE
; -----

;; ED-DEL-1
L036C: LD    BC,$0001    ; one character
        JP    L0666      ; routine RECLAIM-2

```

```
;
;-----
; THE 'EDITING KEYS' TABLE
;-----
```

```
:: ED-K-TAB
L0372: DEFW L03A9      ; ED-UP      $70
        DEFW L02D5      ; ED-DOWN    $71
        DEFW L0382      ; ED-LEFT    $72
        DEFW L0387      ; ED-RIGHT   $73
        DEFW L03B9      ; ED-HOME    $74
        DEFW L03CB      ; ED-EDIT    $75
        DEFW L0408      ; ED-ENTER   $76
        DEFW L0395      ; ED-DELETE  $77
```

```
;
;-----
; THE 'CURSOR LEFT EDITING' SUBROUTINE
;-----
```

```
:: ED-LEFT
L0382: CALL L039E      ; routine ED-EDGE checks that cursor
                        ; not at start without disturbing HL.
                        ; quits early if not possible. >>

        DEC HL        ; move left.
        DEC HL        ; and again for luck.
                        ; ...
```

```
;
;-----
; THE 'CURSOR RIGHT EDITING' SUBROUTINE
;-----
```

```
:: ED-RIGHT
L0387: INC HL          ; move right

        LD A,(HL)      ; pick up the character.
        CP $76         ; is it newline ?

        JR Z,L03A7      ; triple jump back to ED-COPY if so.

        LD (HL),$B0     ; else place inverse cursor there.
        LD HL,($4004)   ; fetch P_PTR
        LD (HL),A       ; and put character there
        JR L035C        ; double jump back to ED-COPY
```

```
;
;-----
; THE 'DELETE EDITING' SUBROUTINE
;-----
```

```
:: ED-DELETE
L0395: CALL L039E      ; routine ED-EDGE will loop back to
                        ; ED-COPY if no deletion possible >>

        DEC HL        ; decrement position
        CALL L036C     ; routine ED-DEL-1
        JR L035C       ; back to ED-COPY
```

```
;
;-----
; THE 'ED-EDGE' SUBROUTINE
;-----
```

```
:: ED-EDGE
L039E: LD DE,($400A)   ; fetch E_LINE - start of edit line.
```

```

        LD    A,(DE)      ; pick up first character.
        CP    $B0        ; test for inverse 'K'
        RET    NZ         ; return if cursor not at start.

        POP    DE         ; else drop the return address.

;; EDC-JR2
L03A7: JR     L035C        ; and back to ED-COPY

; -----
; THE 'CURSOR UP EDITING' SUBROUTINE
; -----

;; ED-UP
L03A9: LD     HL,($4006)   ; E_PPC
        CALL  L060A        ; routine LINE-ADDR
        EX    DE,HL
        CALL  L03C2        ; LINE-NO

;; ED-LINE
L03B3: LD     HL,$4007     ; E_PPC_hi
        JP    L02E5        ; to LN-STORE to store new line
                          ; and produce an automatic listing.

; -----
; THE 'ED-HOME' SUBROUTINE
; -----
; ED-HOME (SHIFT 9) starts the listing at the first line.
; dropped in later ZX computers.

;; ED-HOME
L03B9: LD     DE,$0000     ; start at 'line zero'
        JR    L03B3        ; back to ED-LINE above.

; -----
; THE 'COLLECT A LINE NUMBER' SUBROUTINE
; -----

;; LINE-NO-A
L03BE: EX     DE,HL        ; bring previous line to HL
                          ; and set DE in case we loop back a second time.
        LD     DE,L03B9 + 1 ; address of $00 $00 within the subroutine
                          ; above.

; -> The Entry Point.

;; LINE-NO
L03C2: LD     A,(HL)       ; fetch hi byte of line number
        AND    $C0         ; test against $3F
        JR     NZ,L03BE    ; back to LINE-NO-A if at end.

        LD     D,(HL)      ; else high byte to D
        INC    HL          ; increase pointer
        LD     E,(HL)      ; low byte in E.
        RET              ; return.
                          ; with next line number in DE

; -----
; THE 'EDIT KEY' SUBROUTINE
; -----
; Pressing the EDIT key causes the current line to be copied to the
; edit line. The two-byte line number is converted into 4 characters

```

```
; using leading spaces if the line is less than 1000. Next the 'K'
; cursor is inserted and the rest of the characters are copied verbatim
; into the edit buffer, keywords remaining as single character tokens.
```

```
:: ED-EDIT
```

```
L03CB: LD    C,$00      ; set column to zero to inhibit a line feed
          ; while 'sprinting' to the edit line.
          ; see PRINT-A-2.
```

```
LD    DE,($400A)      ; set DE (print destination) to E_LINE
EXX                      ; switch.
```

```
LD    HL,($4006)      ; E_PPC current line.
CALL  L060A           ; routine LINE-ADDR
CALL  L03C2           ; routine LINE-NO
LD    A,D
OR    E
JP    Z,L0283         ; back if zero to MAIN-EXEC
          ; no program.
```

```
DEC    HL             ; point to location before
CALL  L06BF           ; routine OUT-NUM-2 prints line number
          ; to the edit line (unseen).
```

```
DEC    HL             ; point to line number again
CALL  L0624           ; routine NEXT-ONE gets length in
          ; BC register.
```

```
INC    HL             ; point to the
INC    HL             ; first token.
DEC    BC             ; decrease the length
DEC    BC             ; by the same.
```

```
EXX
PUSH   DE             ; pick up the print position in the
EXX                      ; edit line.
```

```
POP    DE             ; and pop it to this set of registers
LD    A,$B0           ; the inverse 'K' cursor
LD    (DE),A          ; is inserted after line number.
INC    DE             ; address next 'print' location.
```

```
PUSH   HL             ; push position within program.
```

```
LD    HL,$0022        ; an overhead of 34d bytes.
ADD    HL,DE           ; add to edit line position
ADD    HL,BC           ; add in length of line.
SBC    HL,SP           ; subtract the stack pointer.
JR     NC,L03A7        ; back to ED-COPY if not enough
          ; room to fill edit line.
```

```
POP    HL             ; restore program position.
LDIR                      ; and copy it to edit line.
LD    ($400C),DE       ; update D_FILE
```

```
JP    L0293           ; jump back to AUTO-LIST
```

```
; -----
; THE 'ENTER EDITING' SUBROUTINE
; -----
```

```
; This causes the line to be parsed.
; The subroutine then loops back to MAIN-EXEC.
```

```
:: ED-ENTER
```

```

L0408: LD    HL,($4015)    ; fetch X_PTR the error pointer.
      LD    A,H           ; check that it is
      OR    L             ; zero - no error.
      JR    NZ,L03A7      ; double jump back to ED-COPY
                          ; if an error has occurred during
                          ; syntax checking.

      LD    HL,($4004)    ; P_PTR

      CALL  L036C         ; ED-DEL-1 gets rid of cursor.

      LD    HL,($400A)    ; E_LINE

      LD    ($4026),HL    ; CH_ADD
      CALL  L001A         ; get-char
      BIT   5,(IY+$19)    ; FLAGX      input 1/edit 0
      JR    NZ,L043C      ; forward to MAIN-1 if in input mode.

; else the edit line is to be run.

      CALL  L0679         ; INT-TO-HL line number to HL'
      EXX                     ; switch in set with the line number.
      LD    A,H           ; and test
      OR    L             ; for zero.
      JP    NZ,L04BA      ; jump forward with a number to MAIN-ADD
                          ; to add a new BASIC line or replacement.

; else must be a direct command.

      DEC   HL            ; make the line number
      DEC   HL            ; the value minus two.

      LD    ($4002),HL    ; and set PPC

      CALL  L0747         ; routine CLS

      EXX                     ;
      LD    A,(HL)        ; fetch first character.
      CP    $76           ; is it just a newline ?
      JP    Z,L0283       ; jump back with newline to MAIN-EXEC
                          ; to produce an automatic listing.

; else check syntax and enter

;; MAIN-1
L043C: LD    (IY+$00),$FF  ; set ERR_NR to no error
      LD    (IY+$01),$88  ; update FLAGS
                          ; set bit 7 - syntax checking off
                          ; set bit 3 - 'K' mode

;; M-2
L0444: CALL  L07BE         ; routine MAIN-G parses and executes the line.

                          ; Note. this causes the value L0447 to be placed
                          ; on the machine stack as a return address.

;; M-3
L0447: CALL  L0D0A         ; REC-EDIT reclaims the edit line

      LD    DE,($4002)    ; fetch current line number from PPC
      LD    HL,$4019      ; address FLAGX

```

```

    BIT    5,(HL)    ; test FLAGX - input???
    JR     Z,L0458   ; skip if editing to ->

    RES    5,(HL)    ; update FLAGX - signal editing.
    INC    DE        ; increase line number so cursor doesn't show.

;; M-4
L0458: BIT    7,(IY+$00) ; check ERR_NR.
    JR     Z,L0488   ; forward if an error has occurred.

    LD     HL,$4001   ; address FLAGS system variable

    BIT    3,(HL)    ; test FLAGS - K mode ?
    RES    3,(HL)    ; update FLAGS - set L mode for future anyway.

    LD     HL,($4026) ; fetch character address CH_ADD
    INC    HL        ;
    JR     Z,L0474   ; forward if not K mode.

    EX     DE,HL     ; current line to HL, next char to DE.

    LD     A,H        ; fetch high byte of line number.
    AND    $C0        ; test for -2, -1 - direct command.
    JR     NZ,L0488   ; forward to MAIN-ERR if so

    CALL   L060A      ; routine LINE-ADDR gets address of this line.

;; M-5
L0474: LD     A,(HL)   ; fetch
    AND    $C0        ;
    JR     NZ,L0488   ; at program end

; else pick up the next line number

    LD     D,(HL)     ;
    INC    HL         ;
    LD     E,(HL)     ;
    LD     ($4002),DE ; place in PPC system variable
    INC    HL         ; point to first character
                    ; (space or command)

    LD     A,$7F      ; test for
    IN     A,($FE)    ; space key pressed.
    RRA           ; the space bit.
    JR     C,L0444    ; back if BREAK
                    ; else continue...

;; MAIN-ERR
L0488: CALL   L06E0    ; UNSTACK-Z quits if checking syntax >>>

    CALL   L05C2      ; routine CL-EOD clears to the end of upper
                    ; display area.
    LD     BC,$0120   ; set line 1, column 32 for lower screen.
    EXX           ;

    LD     A,($4000)   ; fetch the error number from ERR_NR
    LD     BC,($4002)  ; fetch the current line from PPC
    INC    A          ; test if error still $FF
    JR     Z,L04A8    ; forward if so to MAIN-5.

    CP     $09        ; is the error the STOP statement ?
    JR     NZ,L04A1    ; forward if not STOP to SET-CONT to make the

```


; continuing line the same as current.

INC BC ; else increment line number for STOP.

:: SET-CONT

L04A1: LD (\$4017),BC ; store line number in OLDPPC
JR NZ,L04A8 ; forward if not STOP as line number is current

DEC BC ; else decrement line number again.

; Now print the report line e.g. 100/0 (terminated OK at line 100)

:: MAIN-5

L04A8: CALL L0556 ; routine OUT-CODE prints line number

LD A,\$15 ; prepare character '/'
RST 10H ; print the separator

CALL L06A1 ; OUT-NUM-1 to print error-code in A.

CALL L05C2 ; routine CL-EOD

CALL L013F ; routine KEYBOARD

JP L0283 ; jump back to MAIN-EXEC

; -----

; THE 'MAIN-ADD' BRANCH

; -----

; This section allows a new BASIC line to be added to the Program.

:: MAIN-ADD

L04BA: LD (\$4006),HL ; make E_PPC the new line number.

EXX ;
EX DE,HL ;
CALL L0747 ; routine CLS
SBC HL,DE ;
EXX ;

CALL L060A ; routine LINE-ADDR
PUSH HL ;
JR NZ,L04D1 ; forward if line doesn't exist to MAIN-ADD1.

CALL L0624 ; routine NEXT-ONE gets length of old line
CALL L0666 ; routine RECLAIM-2

:: MAIN-ADD1

L04D1: EXX ;
INC HL ;
LD B,H ;
LD C,L ;
LD A,L ;
SUB \$03 ;
OR H ;
CALL NZ,L094F ; routine TEST-ROOM

POP HL ;
JR NC,L04F4 ; double jump back to MAIN-EXEC
; not possible.

PUSH BC ;

```

    DEC    HL            ;
    CALL   L05D5         ; routine MAKE-ROOM
    INC    DE            ;
    LD     HL,($400C)    ; set HL from D_FILE
    DEC    HL            ; now points to end of edit line.
    POP    BC            ; restore length

    DEC    BC            ;
    LDDR                     ; copy line from edit line to prog.
    LD     HL,($4006)    ; E_PPC - line number
    EX     DE,HL         ; swap
    LD     (HL),D        ; insert high byte
    INC    HL            ;
    LD     (HL),E        ; insert low byte

;; MAIN-JR
L04F4: JP    L0283        ; jump back to MAIN-EXEC

; -----
; THE 'PRINT A WHOLE BASIC LINE' SUBROUTINE
; -----

;; OUT-LINE
L04F7: LD     BC,($4006)  ; fetch E_PPC
      CALL   L061C        ; routine CP-LINES
      LD     D,$97        ; prepare character '>'
      JR     Z,L0507      ; forward with line cursor if line is the
                          ; current edit line to OUT-LINE-1

      LD     DE,$0000     ; else replace line cursor with a
                          ; space in D, and zero to E.
      RL     E            ; pick up any carry from CP-LINES
                          ; should the line precede the
                          ; current edit line.

;; OUT-LINE-1
L0507: LD     A,(HL)      ; fetch the high byte of line number.
      CP     $40          ; compare with end marker
      CALL   C,L06BF      ; routine OUT-NUM-2 if a valid line number.
      RET    NC           ; return if out of screen      >>>

      INC    HL           ; address the first command character.
      LD     A,D          ; fetch the space/cursor
      RST    10H         ; print it.
      RET    NC           ; return if out of screen.

;; OUT-LINE-2
L0512: SET    0,(IY+$01)  ; update FLAGS - suppress a leading space

;; OUT-LINE-3
L0516: LD     BC,($4015)  ; fetch error pointer - X_PTR
      AND    A           ; prepare to subtract.
      SBC    HL,BC       ; subtract the current address.
      JR     NZ,L0523     ; forward to OUT-LINE-4 if not an
                          ; exact match.
      LD     A,$B8       ; prepare inverse 'S' to show syntax error.
      RST    10H         ; print it.
      RET    Z           ; return if at end

;; OUT-LINE-4
L0523: ADD    HL,BC       ; restore pointer.
      LD     A,(HL)      ; fetch character.

```

```

    INC    HL            ; address next character.

    CP     $B0           ; is character inverse 'K' ?
    JR     Z,L053C       ; forward if so to OUT-CURS.

; then cleverly split the characters into 4 streams.

    CP     $C0           ; compare character to 192 ?

    JP     PE,L0559      ; jump forward with 64-127 to OUT-SP-CH
                        ; thereby exiting the routine
                        ; as it must be the 118, NEWLINE character.

    JR     C,L0536       ; forward with 0-63, 128-191 to OUT-LINE-5
                        ; to print simple characters and their inverse
                        ; forms.

; that leaves tokens $C0 - $FF

    CALL   L0584         ; routine PO-TOKEN

    JR     L0539         ; forward to OUT-LINE-6

; ---

;; OUT-LINE-5
L0536: CALL   L0559      ; routine OUT-SP-CH

;; OUT-LINE-6
L0539: RET     NC        ; return if out of screen.      >>
      JR     L0516       ; else back to OUT-LINE-3 for more.

; -----
; Z80 PARITY/OVERFLOW FLAG:
; -----
; The use of this flag is two-fold depending on the type of operation.
; It indicates the parity of the result of a LOGICAL operation such as an AND,
; OR, XOR by being set PE if there are an even number of set bits and reset
; PO if there are an odd number of set bits.
; so 10101010 is parity even, 00000001 is parity odd.
; JP PE, LABEL
; JP PO, LABEL are obvious.
; For MATHEMATICAL operations, (ADD, SUB, CP etc.) the P/V bit indicates a
; carry out of bit position 6 of the accumulator if signed values are being
; used.
; This indicates an overflow of a result greater than 127, which carries
; into bit 7, the sign bit.
; So as CP is just a SUB with the result thrown away.
; $C0 SUB $C0 gives result $00 (PO - no overflow from 6 to 7)
; $80 SUB $C0 gives result $C0 (PO - no overflow from 6 to 7)
; $00 SUB $C0 gives result $40 (PO - no overflow from 6 to 7)
; $40 SUB $C0 gives result $80 (PE - overflow from 6 to 7)
; The overflow flag is similarly set following 16-bit addition and subtraction
; routines.
; -----

; -----
; THE 'PRINT THE CURSOR' BRANCH
; -----

;; OUT-CURS

```

```

L053C: BIT 2,(IY+$01) ; test FLAGS - K-mode ?
        JR NZ,L0543 ; skip to OUT-K if 'K' mode.

        INC A ; change from 'K' to 'L' cursor.

;; OUT-K
L0543: RST 10H ; print the cursor.
        JR L0539 ; back to OUT-LINE-6 above.

;-----
; THE 'PRINTING CHARACTERS IN A BASIC LINE' SUBROUTINES
;-----

;; OUT-SP-2
L0546: LD A,E ; transfer E to A
        ; register E will be
        ; $FF - no leading space.
        ; $01 - the leading space itself.
        ; $1C - '0' from a previous non-space print.
        RLCA ; test for the
        RRCA ; value $FF.
        RET C ; return if no leading space

        JR L055C ; forward to OUT-LD-SP

; ---

; --> The Entry Point.

;; OUT-SP-NO
L054C: XOR A ; set accumulator to zero.

;; OUT-SP-1
L054D: ADD HL,BC ; addition of negative number.
        INC A ; increment the digit.
        JR C,L054D ; back while overflow exists to OUT-SP-1

        SBC HL,BC ; else reverse the last addition.
        DEC A ; and decrement the digit.

        JR Z,L0546 ; back to OUT-SP-2 if digit is zero again.

; else continue to print the final digit using OUT-CODE.

;; OUT-CODE
L0556: LD E,$1C ; load E with '0'
        ; Note. that E will remain as such for all
        ; further calls. The leading space is no more.
        ADD A,E ; add the digit 1-9 to give '1' to '9'

;; OUT-SP-CH
L0559: AND A ; test value for space.
        JR Z,L0560 ; skip if zero to PRINT-A-2

;; OUT-LD-SP
L055C: RES 0,(IY+$01) ; signal allow leading space to FLAGS
        ; and continue...

;-----
; THE 'MAIN PRINTING' SUBROUTINE
;-----
; This is a continuation of the PRINT restart.

```

```
; It is used primarily to print to the dynamic screen checking free memory
; before every character is printed.
; However it can also be used as an invisible process to 'sprint' the line
; number of a BASIC line to the Edit Line by ED-EDIT setting DE from E_LINE.
;
; As lines are unexpanded, then when the column count is reduced from 32 to 0 a
; newline is inserted before the character and the column count is reset.
```

```
:: PRINT-A-2
```

```
L0560: EXX          ; switch sets.
```

```
LD    H,A          ; preserve character in H.
```

```
          ; Note. this is restored by TEST-RM-2
```

```
RLA          ; rotate character twice to
```

```
RLA          ; test bit 6 - sets carry for NEWLINE.
```

```
DEC    C          ; decrease column count - affects zero / sign.
```

```
JR     NC,L0569    ; forward if 0-63 or inverse to NO-NL
```

```
; else the incoming character is a NEWLINE $76
```

```
LD     C,$00       ; set column to zero without disturbing flags.
```

```
          ; if this is a received NEWLINE.
```

```
          ; this will be set to 32 if a subsequent
```

```
          ; character is printed
```

```
:: NO-NL
```

```
L0569: JP     M,L0574    ; jump to PR-SPR if column was originally 0
```

```
JR     C,L057C     ; forward to PRI-CHAR with a received NEWLINE.
```

```
JR     NZ,L057C     ; forward if column not yet reduced to zero
```

```
          ; to PRI-CHAR
```

```
; else an automatic newline is required before the received character as
```

```
; we are at end of line.
```

```
LD     A,$76       ; prepare the newline
```

```
LD     (DE),A      ; insert at screen position
```

```
INC     DE          ; increase the address pointer.
```

```
:: PR-SPR
```

```
L0574: JR     C,L0578    ; skip if a received newline to PRI-SKIP
```

```
LD     C,$20       ; reset column to 32 decimal.
```

```
:: PRI-SKIP
```

```
L0578: AND     A          ; clear carry now to signal failure should the
          ; next test fail.
```

```
DEC     B          ; decrease line.
```

```
JR     Z,L0582     ; forward with out of screen to PR-END.
```

```
:: PRI-CH
```

```
L057C: LD     L,B          ; transfer line number, B to L for next routine.
```

```
CALL    L0958       ; routine TEST-RM-2 tests room.
```

```
          ; (character is in H returned in A)
```

```
          ; carry set if there is room.
```

```
LD     (DE),A      ; insert chr at screen (or edit line).
```

```
INC     DE          ; increase destination address.
```

```

;; PR-END
L0582: EXX          ; switch to protect registers.
      RET          ; return

;-----
; THE 'TOKEN PRINTING' SUBROUTINE
;-----

;; PO-TOKEN
L0584: CALL  L05A8      ; routine PO-SEARCH locates token
      JR   NC,L0592     ; forward to PO-LOOP if first character is
                        ; not alphanumeric. e.g. '***'

; else consider a leading space.

      BIT  0,(IY+$01)   ; test FLAGS - leading space allowed ?
      JR   NZ,L0592     ; forward to PO-LOOP if not.

; else print a leading space.

      XOR  A           ; prepare a space
      RST  10H         ; print it
      RET  NC          ; return if out of screen.

; now enter a loop to print each character and then consider a trailing space.

;; PO-LOOP
L0592: LD   A,(BC)      ; fetch character from token table.
      AND  $3F         ; mask to give range ' ' to 'Z'

      CALL L0559        ; routine OUT-SP-CH

      RET  NC          ; return if out of screen.

      LD   A,(BC)      ; reload the character
      INC  BC          ; point to next.
      ADD  A,A         ; test for the inverted bit.
      JR   NC,L0592     ; loop back if not inverted to PO-LOOP

;

      CP   $38         ; compare with what was '0' before doubling.
      RET  C           ; return if less. i.e. not a command.  >>

      XOR  A           ; else prepare a space
      SET  0,(IY+$01)   ; update FLAGS - use no leading space
      JR   L0560        ; back to PRINT-A-2 for trailing space.  >>

;-----
; THE 'TABLE SEARCH' SUBROUTINE
;-----

;; PO-SEARCH
L05A8: PUSH  HL         ; * preserve character pointer

      LD   HL,$00BA     ; point to start of the table
      SUB  (HL)         ; test against the threshold character 212
      INC  HL          ; address next in table. ('?' + $80 )
      JR   C,L05B9      ; forward to PO-FOUND if less than 212
                        ; to print a question mark.

```

```

        INC    A            ; make range start at 1 for chr 212.
                                ; note - should the required token be 212
                                ; the printable quote character then the
                                ; pointer currently addresses "" + $80.
        LD     B,A          ; save reduced token in B as a counter.

;; PO-STEP
L05B2: BIT    7,(HL)        ; test for inverted bit
        INC    HL          ; increase address
        JR     Z,L05B2      ; back to PO-STEP for inverted bit

        DJNZ   L05B2        ; decrement counter and loop back to PO-STEP
                                ; until at required token.

;; PO-FOUND
L05B9: LD     B,H          ; transfer the address
        LD     C,L          ; to BC.

        POP    HL          ; * restore string address
        LD     A,(BC)       ; fetch first character from token.
        AND    $3F          ; mask off range 0-63d, SPACE to Z
        ADD    A,$E4        ; add value 228
        RET                     ; return with carry set if alphanumeric and a
                                ; leading space is required.

; -----
; THE 'CLEAR TO END OF DISPLAY' ROUTINE
; -----

;; CL-EOD
L05C2: EXX                ; switch in the set with screen values.

        XOR    A            ; clear accumulator.
        CP     B            ; compare with line counter - 0 to 23.
        JR     Z,L05D0      ; forward if clear to SET-EOD.

        CP     C            ; compare to column count - 0 to 32.
        LD     A,$76        ; prepare a NEWLINE.
        JR     Z,L05CE      ; forward, if zero, to CL-EOL.

;; INS-CR
L05CC: LD     (DE),A        ; insert a newline/carriage return.
        INC    DE          ; address next position.

;; CL-EOL
L05CE: DJNZ   L05CC        ; reduce line counter and loop back to INS-CR.

;; SET-EOD
L05D0: LD     ($4010),DE    ; update DF_END - display file end.
        RET                ; return.

; -----
; THE 'MAKE-ROOM' SUBROUTINE
; -----

;; MAKE-ROOM
L05D5: CALL   L05DF        ; routine POINTERS also sets BC
        LD     HL,($4010)   ; fetch new display file end DF_END
        EX     DE,HL        ; switch source/destination.
        LDDR                     ; now make the room.
        RET                ; return.
                                ; with HL pointing at first new location.

```

```

;-----
; THE 'POINTERS' SUBROUTINE
;-----

```

```

;; POINTERS
L05DF: PUSH  AF      ;
      PUSH  HL      ;
      LD   HL,$4008  ; VARS
      LD   A,$05     ;

```

```

;; PTR-NEXT
L05E6: LD    E,(HL)   ;
      INC  HL        ;
      LD   D,(HL)    ;
      EX  (SP),HL    ;
      AND  A         ;
      SBC  HL,DE     ;
      ADD  HL,DE     ;
      EX  (SP),HL    ;
      JR   NC,L05FA  ; forward to PTR-DONE

```

```

      PUSH DE        ;
      EX  DE,HL      ;
      ADD  HL,BC      ;
      EX  DE,HL      ;
      LD  (HL),D      ;
      DEC  HL         ;
      LD  (HL),E      ;
      INC  HL         ;
      POP  DE         ;

```

```

;; PTR-DONE
L05FA: INC  HL        ;
      DEC  A         ;
      JR   NZ,L05E6  ; back to PTR-NEXT for all five
                        ; dynamic variables.

```

; now find the size of the block to be moved.

```

      EX  DE,HL      ;
      POP  DE        ;
      POP  AF        ;
      AND  A         ;
      SBC  HL,DE     ;
      LD  B,H        ;
      LD  C,L        ;
      INC  BC         ;
      ADD  HL,DE     ;
      EX  DE,HL      ;
      RET            ; return ->

```

```

;-----
; THE 'LINE-ADDR' SUBROUTINE
;-----

```

```

;; LINE-ADDR
L060A: PUSH  HL      ; save the given line number.
      LD   HL,$4028  ; start of PROG
      LD   D,H       ; transfer the address
      LD   E,L       ; to the DE register pair.

```



```

;; LINE-AD-1
L0610: POP  BC      ; the given line number.
      EX   DE,HL    ;

      CALL  L061C    ; routine CP-LINES

      RET   NC       ; return if carry set      >>

      PUSH  BC       ; otherwise save given line number

      CALL  L0624    ; routine NEXT-ONE

      JR    L0610    ; back to LINE-AD-1 to consider the next
                      ; line of the program.

;-----
; THE 'COMPARE LINE NUMBERS' SUBROUTINE
;-----

;; CP-LINES
L061C: LD   A,(HL)   ; fetch the high byte of the addressed line
      CP   B        ; number and compare it.
      RET  NZ       ; return if they do not match.

      INC  HL       ; next compare the low bytes.
      LD   A,(HL)   ;
      DEC  HL       ;
      CP   C        ;
      RET                ; return with carry flag set if the addressed
                      ; line number has yet to reach the
                      ; given line number.

;-----
; Storage of variables. For full details - see Page 107
; ZX80 BASIC Programming by Hugo Davenport 1980.
; It is bits 7-5 of the first character of a variable that allow
; the five types to be distinguished. Bits 4-0 are the reduced letter.
; So any variable name is higher than $3F and can be distinguished
; also from the variables area end-marker $80.
;
;
; 76543210 meaning          brief outline of format after letter.
;-----
; 011  simple integer variable.  2 bytes. (after letter)
; 010  long-named integer variable  2 bytes. (after inverted name)
; 100  string          letter + contents + $01.
; 101  array of integers  letter + max subs byte + subs * 2.
; 111  for-next loop variable.  7 bytes - letter, value, limit, line.
; 10000000 the variables end-marker.
;
; Note. any of the above six will serve as a program end-marker.
;
;-----

;-----
; THE 'NEXT-ONE' SUBROUTINE
;-----

;; NEXT-ONE
L0624: PUSH  HL      ; save address of current line or variable.

      LD   A,(HL)   ; fetch the first byte.
      ADD  A,A      ; test bits 7 and 6

```

```
JP    M,L0635      ; jump forward if simple, long-named or for-next
                        ; control variable to NO-SLNFM
```

```
JR    C,L0643      ; forward if string or arrays to NO-STR-AR
```

; that leaves program line numbers.

```
INC    HL          ; step past high byte
LD     A,$76        ; the search is for newline
```

:: NO-SEARCH

```
L062F: INC    HL      ; skip to next address past low byte.
```

```
LD     B,A          ; save search byte in B to create
                        ; a large value in BC so that search is
                        ; not curtailed.
```

```
CPIR                    ; and locate the known character.
```

```
JR     L0652        ; forward to ??? with HL addressing
                        ; the following character.
```

; ---

; the branch was here with simple, long-named and for-next variables

:: NO-SLNFM

```
L0635: LD     BC,$0002    ; presume a for-next variable (1+2 cells)
```

```
JR     C,L063B      ; skip forward if for-next variable.
```

```
LD     C,B          ; set C to zero - just one cell for simple
                        ; and long-named.
```

:: NO-FNXT

```
L063B: RLA          ; original bit 5 is now bit 7.
```

:: NO-LNLP

```
L063C: RLA          ; test original bit 5 of letter.
```

```
INC    HL          ; advance address.
```

```
LD     A,(HL)       ; pick up next byte - possibly a letter
```

```
JR     NC,L063C      ; back if originally long-named or if
                        ; on subsequent loops character is not inverted
```

; whatever the route we are now pointing at the first cell with the number
; of cells less one in register C.

```
JR     L064F        ; forward to NO-CELLS to calculate space to the
                        ; end of variable.
```

; ---

; the branch was here with either single strings or numeric array variables

:: NO-STR_AR

```
L0643: AND    $40        ; test shifted bit 6 - will be set for arrays
```

```
LD     A,$01        ; set search for null terminator
```

```
JR     Z,L062F      ; back if not an array to NO-SEARCH to
                        ; search for the end of string.
```

; the object is a NUMERIC ARRAY

```
INC    HL          ; point to maximum subscription
```

```
LD     A,(HL)       ; and fetch
```

```
INC    HL          ; point to first cell.
```

```

        LD    B,$00      ; prepare to index
        LD    C,A        ; max subscription to C
                        ; and continue to find following byte.

;; NXT-O-6
L064F: INC    BC          ; bump the range
        ADD   HL,BC       ; add to start
        ADD   HL,BC       ; add again as each cell is two bytes.

;; NXT-O-7
L0652: POP    DE          ; restore previous address to DE and
                        ; continue into the difference routine...

; -----
; THE 'DIFFERENCE' SUBROUTINE
; -----

;; DIFFER
L0653: AND    A           ; prepare to subtract.
        SBC   HL,DE       ; calculate the length of the line/var

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

        ADD   HL,DE       ; reform the address of next one in HL.
        EX    DE,HL       ; swap pointers
        RET              ; return.

; -----
; THE 'CLEAR' COMMAND SUBROUTINE
; -----
; The CLEAR command removes all BASIC variables.

;; CLEAR
L065B: LD     HL,($400A)   ; set HL to E_LINE.
        DEC   HL          ; decrement to point to the $80 end-marker.
        LD    DE,($4008)  ; set start from VARS system variable.

; -----
; THE 'RECLAIMING' SUBROUTINES
; -----

;; RECLAIM-1
L0663: CALL   L0653       ; routine DIFFER

;; RECLAIM-2
L0666: PUSH   BC          ;

        LD    A,B         ;
        CPL                     ;
        LD    B,A         ;

        LD    A,C         ;
        CPL                     ;
        LD    C,A         ;

        INC   BC          ;

        CALL  L05DF       ; routine POINTERS

        EX    DE,HL       ;
        POP   HL          ;

```

```

    ADD    HL,DE    ;
    PUSH   DE       ;
    LDIR                   ;
    POP    HL       ;
    RET                      ; return.

;-----
; THE 'INTEGER TO ALTERNATE HL' SUBROUTINE
;-----

;; INT-TO-HL
L0679: LD    A,(HL)    ; fetch first digit
      EXX          ; switch
      LD    HL,$0000    ; initialize result register to zero.
      LD    B,H        ; make B zero also.

;; DEC-LP
L067F: SUB    $1C        ; subtract chr '0'
      JR    C,L069A    ; forward to STOR-RSLT if less.    >>

      CP    $0A        ; compare with 'ten'
      JR    NC,L069A    ; forward to STOR-RSLT if higher than '9'. >>

      LD    C,A        ; save unit in C.

; now test that the result is not about to enter the 32768 - 65535 region.

      LD    A,$0D        ; value 13 to A
      CP    H          ; compare to result_hi
      JR    NC,L068E    ; forward if less to NO-OVERFLW

      LD    H,A        ; else maintain the overflow condition.

;; NO-OVRFLW
L068E: LD    D,H        ; copy HL.
      LD    E,L        ; to DE.
      ADD   HL,HL        ; double result
      ADD   HL,HL        ; and again.
      ADD   HL,DE        ; now * 5
      ADD   HL,HL        ; now *10
      ADD   HL,BC        ; add in new digit.

      EXX          ; switch
      RST    18H        ; NXT-CH-SP
      EXX          ; switch

      JR    L067F        ; loop back to DEC-LP for more digits.

;-----
; THE 'STORE INTEGER RESULT' SUBROUTINE
;-----

;; STOR-RSLT
L069A: LD    A,H        ; transfer high byte to A.
      LD    ($4022),HL    ; set value of expression RESULT
      EXX          ; switch
      RLA          ; sets carry if higher than 32767
      RET          ; return.

;-----
; THE 'REPORT AND LINE NUMBER PRINTING' SUBROUTINE
;-----

```

; Actually the first entry point prints any number in the
; range -32768 to 32767.

; --> This entry point prints a number in BC.

;; OUT-NUM-1

L06A1: PUSH DE ; preserve registers
PUSH HL ; throughout

LD H,B ; transfer number
LD L,C ; to be printed to HL.

BIT 7,B ; test the sign bit
JR Z,L06B5 ; forward if positive to OUT-NUM-P

LD A,\$12 ; prepare character '-'
CALL L0559 ; routine OUT-SP-CH

JR NC,L06DD ; forward if out of screen to OUT-NUM-4

LD HL,\$0001 ; else make the negative number
SBC HL,BC ; positive.

; at this stage the number is positive

;; OUT-NUM-P

L06B5: LD E,\$FF ; signal no leading space.

LD BC,\$D8F0 ; prepare the value -10000

CALL L054C ; routine OUT-SP-NO will print the first digit
; of a 5-digit number but nothing if smaller.

JR L06C8 ; forward to OUT-NUM-3
; to consider other four digits in turn.
; (with carry set from a successful print)

; ---

; --> This entry point prints a BASIC line number addressed by HL.

;; OUT-NUM-2

L06BF: PUSH DE ; save DE throughout
LD D,(HL) ; fetch high byte of number to D
INC HL
LD E,(HL) ; fetch low byte of number to E

PUSH HL ; save HL now till the end.
EX DE,HL ; number to HL.

LD E,\$00 ; prepare a leading space
SCF ; set carry flag for subtractions.

; both paths converge here.

;; OUT-NUM-3

L06C8: LD BC,\$FC18 ; the value -1000
CALL C,L054C ; routine OUT-SP-NO

LD BC,\$FF9C ; the value -100
CALL C,L054C ; routine OUT-SP-NO

```

        LD    C,$F6        ; the value -10
        CALL  C,L054C      ; routine OUT-SP-NO

        LD    A,L          ; the remainder.
        CALL  C,L0556      ; routine OUT-CODE

;; OUT-NUM-4
L06DD: POP    HL           ; restore original
        POP    DE          ; registers.
        RET                     ; return.

; -----
; THE 'UNSTACK-Z' SUBROUTINE
; -----

;; UNSTACK-Z
L06E0: BIT    7,(IY+$01)    ; test FLAGS - Checking Syntax ?
        POP    HL          ; drop the return address
        RET    Z           ; return if so.

; else fetch screen coordinates alternate registers for the run-time situation.

        EXX
        LD    DE,($400E)    ; fetch display print position DF_EA
        LD    BC,($4024)    ; fetch line and column from SPOSN
        EXX                ; exchange and continue...

; and jump back to the calling routine...

; -----
; THE 'USR' FUNCTION
; -----

;; USR
L06F0: JP     (HL)          ; that appears to be it.

; -----
; THE 'PRINT ITEM' SUBROUTINE
; -----

;; PR-ITEM
L06F1: BIT    7,(IY+$00)    ; ERR_NR
        RET    Z           ; return if an error has already been
                        ; encountered.

        CALL  L06E0        ; UNSTACK-Z quits if checking syntax

        LD    HL,($4022)    ; fetch result of SCANNING from RESULT
        BIT    6,(IY+$01)    ; test FLAGS for result type.
        JR    Z,L070C      ; forward to PR-STRING if type string.

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

        CALL  L06A1        ; routine OUT-NUM-1
        JR    L0723        ; forward to PO-CHECK to check for
                        ; success and store position

; -----
; THE 'PRINT STRING' SUBROUTINE
; -----

```

```

;; PO-CHAR
L0709: RST 10H ; PRINT-A

;; PO-LOOP
L070A: JR NC,L0725 ; forward to ERROR-05 with carry
; Out of screen.
; --> Entry Point.

;; PR-STRING
L070C: LD A,(HL) ; fetch a character.
INC HL ; increment pointer.
CP $01 ; is it null-terminator.
JR Z,L073A ; forward to PO-STORE if so.

BIT 6,A ; test if simple character or inverse
JR Z,L0709 ; back to PO-CHAR if so

CALL L0584 ; routine PO-TOKEN to print
; ranges $40 - $7f, $0C - $FF
JR L070A ; loop back to PO-LOOP

; -----
; THE 'CARRIAGE RETURN' SUBROUTINE
; -----

;; PRINT-CR
L071B: CALL L06E0 ; UNSTACK-Z quits if checking syntax

LD A,$76 ; prepare a NEWLINE character
CALL L0559 ; routine OUT-SP-CH prints it
; returning with carry reset if there
; was no room on the screen.

;; PO-CHECK
L0723: JR C,L073A ; forward to PO-STORE if OK

;; ERROR-05
L0725: RST 08H ; ERROR restart
DEFB $04 ; No more room on screen.

; -----
; THE 'PO-FILL' SUBROUTINE
; -----

;; PO-FILL
L0727: CALL L06E0 ; UNSTACK-Z return if checking syntax.

SET 0,(IY+$01) ; signal no leading space.

;; PO-SPACE
L072E: XOR A ; prepare a space
RST 10H ; PRINT-A outputs the character.
JR NC,L0725 ; back to ERROR-05 if out of screen

EXX ;
LD A,C ; get updated column
EXX ;

DEC A ; decrement it.
AND $07 ; isolate values 0 - 7
JR NZ,L072E ; back to PO-SPACE for more.

```

```

;-----
; THE 'POSITION STORE' SUBROUTINE
;-----

;; PO-STORE
L073A: EXX      ; switch in the set that maintains the print
              ; positions in the registers.
        EX  DE,HL      ; switch print position to HL for easier coding.

;; PO-STOR-2
L073C: LD      ($4024),BC      ; set SPOSN to line/column
        LD      ($400E),HL      ; set DF_EA to output address
        LD      ($4010),HL      ; set DF_END output address
        RET              ; return.

;-----
; THE 'CLS' COMMAND SUBROUTINE
;-----

;; CLS
L0747: LD      HL,($400C)      ; fetch start of display from D_FILE

        LD      (HL),$76      ; insert a single newline.
        INC     HL              ; advance address.

        LD      BC,$1721      ; set line to 23 and column to 33.
        JR      L073C          ; back to PO-STOR-2 above

;-----
; THE 'SYNTAX TABLES'
;-----

;; i. The offset table

L0752: DEFB    L07A1 - $      ; $4F offset to $07A1 P-LIST
        DEFB    L077F - $      ; $2C offset to $077F P-RETURN
        DEFB    L07B8 - $      ; $64 offset to $07B8 P-CLS
        DEFB    L0794 - $      ; $3F offset to $0794 P-DIM
        DEFB    L07AF - $      ; $59 offset to $07AF P-SAVE
        DEFB    L0782 - $      ; $2B offset to $0782 P-FOR
        DEFB    L076F - $      ; $17 offset to $076F P-GO-TO
        DEFB    L07A4 - $      ; $4B offset to $07A4 P-POKE
        DEFB    L0790 - $      ; $36 offset to $0790 P-INPUT
        DEFB    L07A9 - $      ; $4E offset to $07A9 P-RANDOMISE
        DEFB    L076C - $      ; $10 offset to $076C P-LET
        DEFB    L07BB - $      ; $5E offset to $07BB P-CH-END
        DEFB    L07BB - $      ; $5D offset to $07BB P-CH-END
        DEFB    L0789 - $      ; $2A offset to $0789 P-NEXT
        DEFB    L078D - $      ; $2D offset to $078D P-PRINT
        DEFB    L07BB - $      ; $5A offset to $07BB P-CH-END
        DEFB    L07C2 + 1 - $    ; $61 offset to $07C3 P-NEW
        DEFB    L079E - $      ; $3B offset to $079E P-RUN
        DEFB    L077C - $      ; $18 offset to $077C P-STOP
        DEFB    L07B2 - $      ; $4D offset to $07B2 P-CONTINUE
        DEFB    L0773 - $      ; $0D offset to $0773 P-IF
        DEFB    L0778 - $      ; $11 offset to $0778 P-GOSUB
        DEFB    L07AC - $      ; $44 offset to $07AC P-LOAD
        DEFB    L07B5 - $      ; $4C offset to $07B5 P-CLEAR
        DEFB    L079B - $      ; $31 offset to $079B P-REM
        DEFB    L07BB - $      ; $50 offset to $07BB P-CH-END

```


:: ii. The parameter table.

:: P-LET

L076C: DEFB \$01 ; Class-01 - a variable is required.
DEFB \$E3 ; separator '='
DEFB \$02 ; Class-02 - an expression, of type integer or
; string must follow.

:: P-GO-TO

L076F: DEFB \$06 ; Class-06 - a numeric expression must follow.
DEFB \$00 ; Class-00 - no further operands.
DEFW L0934 ; address: \$0934

:: P-IF

L0773: DEFB \$06 ; Class-06 - a numeric expression must follow.
DEFB \$D5 ; separator 'THEN'
DEFB \$05 ; Class-05 - variable syntax checked entirely
; by routine.
DEFW L08B9 ; address: \$08B9

:: P-GOSUB

L0778: DEFB \$06 ; Class-06 - a numeric expression must follow.
DEFB \$00 ; Class-00 - no further operands.
DEFW L0943 ; address: \$0943

:: P-STOP

L077C: DEFB \$00 ; Class-00 - no further operands.
DEFW L092E ; address: \$092E

:: P-RETURN

L077F: DEFB \$00 ; Class-00 - no further operands.
DEFW L0965 ; address: \$0965

:: P-FOR

L0782: DEFB \$04 ; Class-04 - a single-character variable must
; follow.
DEFB \$E3 ; separator '='
DEFB \$06 ; Class-06 - a numeric expression must follow.
DEFB \$D6 ; separator 'TO'
DEFB \$05 ; Class-05 - variable syntax checked entirely
; by routine.
DEFW L08C4 ; address: \$08C4

:: P-NEXT

L0789: DEFB \$04 ; Class-04 - a single-character variable must
; follow.
DEFB \$00 ; Class-00 - no further operands.
DEFW L08F9 ; address: \$08F9

:: P-PRINT

L078D: DEFB \$05 ; Class-05 - variable syntax checked entirely
; by routine.
DEFW L0972 ; address: \$0972

:: P-INPUT

L0790: DEFB \$01 ; Class-01 - a variable is required.
DEFB \$00 ; Class-00 - no further operands.
DEFW L099A ; address: \$099A

:: P-DIM

L0794: DEFB \$04 ; Class-04 - a single-character variable must

```

; follow.
DEFB $DA ; separator '('
DEFB $06 ; Class-06 - a numeric expression must follow.
DEFB $D9 ; separator ')'
DEFB $00 ; Class-00 - no further operands.
DEFW L0CD3 ; address: $0CD3

;; P-REM
L079B: DEFB $05 ; Class-05 - variable syntax checked entirely
; by routine.
DEFW L084A ; address: $084A

;; P-RUN
L079E: DEFB $03 ; Class-03 - a numeric expression may follow
; otherwise zero will be used.
DEFW L093D ; address: $093D

;; P-LIST
L07A1: DEFB $03 ; Class-03 - a numeric expression may follow
; else default to zero.
DEFW L0256 ; Address: $0256

;; P-POKE
L07A4: DEFB $06 ; Class-06 - a numeric expression must follow.
DEFB $D8 ; separator ','
DEFB $05 ; Class-05 - variable syntax checked entirely
; by routine.
DEFW L09D1 ; address: $09D1

;; P-RANDOMISE
L07A9: DEFB $03 ; Class-03 - a numeric expression may follow
; otherwise zero will be used.
DEFW L0923 ; address: $0923

;; P-LOAD
L07AC: DEFB $00 ; Class-00 - no further operands.
DEFW L0206 ; address: $0206

;; P-SAVE
L07AF: DEFB $00 ; Class-00 - no further operands.
DEFW L01B6 ; address: $01B6

;; P-CONTINUE
L07B2: DEFB $00 ; Class-00 - no further operands.
DEFW L0930 ; address: $0930

;; P-CLEAR
L07B5: DEFB $00 ; Class-00 - no further operands.
DEFW L065B ; address: $065B

;; P-CLS
L07B8: DEFB $00 ; Class-00 - no further operands.
DEFW L0747 ; Address: $0747

;; P-CH-END
L07BB: DEFB $05 ; Class-05 - variable syntax checked entirely
; by routine.
DEFW L0844 ; address: $0844

; Note. one would expect the entry for the P-NEW parameters to be here.

```

```
; It should consist of a class 0, followed by the address word zero as,
; without any protected RAM, the NEW command is no more sophisticated than
; a reset.
; However, there just isn't room. All 4096 bytes of the ROM have been
; put to good use so the required entry, three zero bytes, is embedded
; in the next routine, adding a harmless NOP to make up the three zero bytes.
```

```
; Aye, and you try telling young people of today that. And they won't
; believe you.
```

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

```
:: MAIN-G
```

```
L07BE: DEC HL
      LD ($4026),HL ; CH_ADD
```

```
:: P-NEW-1
```

```
L07C2: LD HL,$0000 ; prepare to clear error pointer.
```

```
      NOP ; Note. See comment above.
```

```
      LD ($4015),HL ; clear X_PTR
```

```
      LD HL,$4019 ; address FLAGX
      BIT 5,(HL) ; is INPUT mode set ?
```

```
      JR Z,L07D7 ; forward if not to E-LINE-NO
```

```
; else runtime input.
```

```
      RES 7,(HL) ; signal L mode.
```

```
      LD B,(HL) ; FLAGX to B for class routine.
      RST 18H ; NXT-CH-SP advances.
```

```
      JP L0889 ; jump forward to VAL-FETCH.
```

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

```
; THE 'E-LINE-NO' SECTION
```

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

```
:: E-LINE-NO
```

```
L07D7: SET 7,(HL) ; update FLAGX - signal K mode
```

```
      RST 20H ; NEXT-CHAR
      CALL L0679 ; routine INT-TO-HL puts the BASIC Line Number
                ; into HL'
```

```
      JR C,L07E5 ; forward if a negative to insert error.
```

```
; else test against upper limit.
```

```
      EXX ;
      LD DE,$D8F0 ; value -9999
      ADD HL,DE ;
      EXX ;
```

```
:: E-L-ERR
```

```
L07E5: CALL C,L08AE ; routine INS-ERR if greater than 9999
```

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

```
; THE 'LINE-SCAN' SECTION
```

; -----

;; LINE-SCAN

L07E8: CALL L001A ; get the COMMAND CHARACTER.

RES 7,(IY+\$19) ; update FLAGX signal not K mode anymore.

LD BC,\$0000 ; this also sets B to zero for later.

LD (\$4022),BC ; default RESULT to ZERO
; for, say, RUN without an operand.

CP \$76 ; compare to just newline
RET Z ; return if so.
; for example with a space for formatting.

LD C,A ; transfer the character to C
RST 20H ; NEXT_CHAR advances pointer
LD A,C ; fetch back character to A.

SUB \$E6 ; subtract lowest command 'LIST'

JR C,L07E5 ; back if not a command to E-L-ERR
; the loop will eventually find the newline
; and the original error point will not be
; altered.

LD C,A ; place reduced character in C.

LD HL,L0752 ; set HL to offset table
ADD HL,BC ; add the one-byte offset

LD C,(HL) ; fetch the offset from table
ADD HL,BC ; add to form address of parameters.
JR L080C ; forward to GET-PARAM

; -----

; THE 'MAIN SCANNING LOOP'

; -----

; entered at GET-PARAM after first instruction.

;; SCAN-LOOP

L0809: LD HL,(\$401A) ; T_ADDR

; --> Entry Point.

;; GET-PARAM

L080C: LD A,(HL) ; get parameter from syntax table.
INC HL ; point to next one.
LD (\$401A),HL ; initialize or update T_ADDR

LD BC,\$0809 ; pre-load the machine stack with the
PUSH BC ; return address SCAN-LOOP above.

LD C,A ; copy parameter entry to C for later.
RLA ; test bit 7
JR C,L0826 ; forward to SEPARATOR if inverted.

LD HL,L0836 ; base address of command class table.
LD B,\$00 ; prepare to index.

ADD HL,BC ; add the command class 0 - 6

```

LD    C,(HL)    ; fetch the addressed byte to C

ADD    HL,BC    ; compute starting address of routine.

PUSH    HL    ; push the address on the machine stack.

CALL    L001A    ; routine GET-CHAR advances character position
                ; and resets the zero flag - see later.

RET    ; >> an indirect jump to the COMMAND CLASS
                ; routine.

                ; Note. HL addresses the next non-space
                ; character e.g. the variable in LET I = 1
                ; the non-space character is in A

; -----
; THE 'SEPARATOR' BRANCH
; -----
; branch to here if the parameter has bit seven set.

;; SEPARATOR
L0826: CALL    L001A    ; get character in A
      CP    $D5    ; compare to the token 'THEN'
      JR    NZ,L0831    ; forward if another character to SEP-1.

      SET    7,(IY+$19)    ; else update FLAGX back to K mode

;; SEP-1
L0831: CP    C    ; compare with expected token/character
      JR    NZ,L08AE    ; forward if no match to set X-PTR
                ; using INS-ERR

      RST    20H    ; else step past a correct character.
      RET    ; return >>
                ; (to SCAN-LOOP)

; -----
; THE 'COMMAND CLASS' TABLE
; -----

;; TAB-CLASS
L0836: DEFB    L0855 - $    ; $1F offset to class-0 $0855
      DEFB    L086A - $    ; $33 offset to class-1 $086A
      DEFB    L0885 - $    ; $4D offset to class-2 $0885
      DEFB    L0850 - $    ; $17 offset to class-3 $0850
      DEFB    L089E - $    ; $64 offset to class-4 $089E
      DEFB    L0856 - $    ; $1B offset to class-5 $0856
      DEFB    L08A8 - $    ; $6C offset to class-6 $08A8

; -----
; THE 'CHECK END' SUBROUTINE
; -----

;; CHECK-END
L083D: BIT    7,(IY+$01)    ; check FLAGS - checking syntax ?
      RET    NZ    ; return if running program.

      POP    BC    ; else drop the return address.

;; CH-END-2
L0843: LD    A,(HL)    ; fetch character from CH_ADD address

```

```

;; CH-END-3
L0844: CP    $76        ; compare to carriage return.

        CALL  NZ,L08AE    ; routine INS-ERR if not disturbing the
                        ; accumulator.

;; SEE-BELOW
L0849: LD     A,(HL)      ; reload character again.
                        ; and continue...

; -----
; THE 'REM' COMMAND ROUTINE
; -----
; The REM command compares each character until a newline is encountered.
; However this is a class 5 routine so the initial accumulator value will
; be zero (from the BC test) and not the character following REM.
; A line consisting of a single REM will have the newline skipped and if no
; $76 is encountered in the binary line number then the following line will
; be skipped also as in
; 10 REM
; 20 PRINT "THIS IS NOT HERE"
; The command address should be that of the previous instruction L0849 as the
; accumulator has been disturbed.

;; REM
L084A: CP    $76        ; compare with newline.
        RET   Z          ; return with newline.

        RST   20H        ; NEXT-CHAR
        JR    L084A      ; loop back to REM until newline found.

; -----
; THE 'COMMAND CLASSES - 00, 03 & 05'
; -----
; these three commands always terminate a sequence of parameters and
; are followed by the address of a routine.

;; CLASS-03
L0850: CP    $76        ; check for carriage return
        CALL  NZ,L08A8    ; else look for optional number using CLASS-06
                        ; e.g. RUN & RUN 100
                        ; return and continue through other two classes.

;; CLASS-00
L0855: CP    A          ; set the zero flag to invoke CHECK-END later.
                        ; this class has no operands e.g. CONTINUE.

;; CLASS-05
L0856: POP    BC        ; drop the looping address - last in sequence.

        CALL  Z,L083D    ; routine CHECK-END if zero flag set.
                        ; (classes 03 and 00)

        EX    DE,HL      ; save HL in DE (original CH_ADD)

        LD    HL,($401A) ; fetch table address from T_ADDR

        LD    C,(HL)     ; low byte to C
        INC   HL         ;
        LD    B,(HL)     ; high byte to B

```

```

EX    DE,HL        ; bring back the original character address

;; JUMP-BC
L0862: PUSH  BC      ; push routine address on machine stack
      LD    BC,($4022) ; load value of last expression from RESULT
      LD    A,B      ; test the value
      OR    C        ; for zero.
      RET           ; jump to the command routine.
                        ; with HL pointing at original CH_ADD
                        ; DE pointing to T_ADDR
                        ; BC holding parameter

; -----
; THE 'COMMAND CLASSES - 01, 02, 04 & 06'
; -----

; the first routine is for LET or INPUT.

;; CLASS-01
L086A: CALL  L0D14    ; routine ALPHA tests the character.
      JR    NC,L08AE  ; forward to INS-ERR if character not A-Z.

      BIT   7,(IY+$01) ; test FLAGS - the syntax bit.
      JP    Z,L0AAD    ; jump forward to LOOK-VARS if checking syntax.

; continue in runtime

      LD    ($4020),HL ; save address of destination variable
                        ; in BASIC line in DEST system variable.

      RES   7,(IY+$01) ; signal to FLAGS that syntax is being checked.

      CALL  L0AAD      ; routine LOOK-VARS.

      SET   7,(IY+$01) ; set FLAGS back to 'running program' status.
      RET           ; return (to SCAN-LOOP).

; -----

; used only for LET - an expression of the correct type must be present.

;; CLASS-02
L0885: POP   BC        ; drop the looping address as CLASS-02 is the
                        ; last in a sequence of parameters. It is
                        ; relevant only to the LET command.

      LD    B,(IY+$01) ; load B with value of FLAGS.

; (runtime input joins here with FLAGX in B instead of FLAGS)

; -----
; THE 'FETCH A VALUE' SECTION
; -----

;; VAL-FETCH
L0889: PUSH  BC        ; preserve value of FLAGS (or FLAGX if input)
      RST   28H        ; SCAN-CALC evaluates the expression
                        ; to be assigned setting the result type flag.
      POP   DE         ; restore the pre-evaluation copy of the
                        ; flag register to D.

      LD    BC,L0C3D    ; the address of the LET routine is pushed on

```

```

; the machine stack.

LD    A,($4001)    ; fetch the post-evaluation FLAGS to A
BIT    7,A          ; test the syntax bit.
JR     NZ,L0862     ; back in runtime to JUMP-BC and then LET

; if checking syntax.

XOR    D            ; exclusive or the two flags
AND    $40          ; AND 01000000 to isolate the type bit.

CALL   NZ,L08AE     ; routine INS-ERR inserts the error position
                    ; when they are not the same type.

JR     L0843        ; back to CH-END-2 to consider lesser errors
                    ; and advance to end of line.

; -----

; FOR, NEXT, DIM - HL points to variable in BASIC line, A holds the character

;; CLASS-04
L089E: LD    ($4020),HL    ; set system variable DEST from HL.

CALL   L0D14        ; routine ALPHA checks the character.

JR     NC,L08AE     ; forward to INS-ERR if not A-Z.

RST    18H          ; NXT-CH-SP advances character address.

RET                    ; return to SCAN-LOOP >>

; -----

; a mandatory INTEGER expression must follow. e.g. GO TO 100

;; CLASS-06
L08A8: RST    28H          ; SCAN-CALC evaluates expression.
      BIT    6,(IY+$01)    ; test FLAGS - numeric result ?
      RET    NZ          ; return if numeric.

; -----
; THE 'INSERT ERROR' SUBROUTINE
; -----

;; INS-ERR
L08AE: LD    A,($4015)    ; check that error pointer X_PTR
      OR     (IY+$16)      ; contains zero.
      RET    NZ          ; return if there is already an error

      LD     ($4015),HL    ; else place error address at X-PTR
      RET                    ; return.

; -----
; THE 'IF' COMMAND ROUTINE
; -----

;; IF
L08B9: JR     NZ,L08C1    ; if expression is TRUE forward to IF-1

      BIT    7,(IY+$01)    ; test FLAGS - checking syntax ?
      JR     NZ,L084A     ; back to REM to ignore rest of the line

```



```

; in runtime.

; - else continue and check the syntax of the rest of the line.

;; IF-1
L08C1: JP    L07E8      ; jump back to LINE-SCAN to execute what
                       ; follows the 'THEN'

; -----
; THE 'FOR' COMMAND ROUTINE
; -----
; for example, FOR X = 1 TO 10
; There is no step or direction.
; The body of the loop is always entered at least once - even if the initial
; value exceeds the limit.
; The ZX81 and ZX Spectrum adhered more closely to the ANS X3.60 1978 BASIC
; standard.

;; FOR
L08C4: PUSH  BC          ; save the start value.

      CALL  L08A8        ; routine CLASS-06 evaluates LIMIT
                       ; expression.

      POP   BC           ; start value back to BC

      CALL  L083D        ; routine CHECK-END quits if checking
                       ; syntax          >>
      LD    HL,($4022)   ; fetch limit from RESULT
      PUSH  HL           ; save limit

      CALL  L0C3D        ; routine LET

      POP   BC           ; restore limit to BC
      BIT   7,(IY+$00)   ; examine ERR_NR
      RET   Z            ; return if not $FF      >>

      PUSH  BC           ; push the limit value.
      DEC   HL           ; point to letter.
      BIT   7,(HL)       ; test bit 7 - is it a FOR-NEXT variable.
      SET   7,(HL)       ; set bit 7 as it is going to be.

      INC   HL           ; point to end of value
      INC   HL
      JR    NZ,L08EA     ; skip forward if it is a proper
                       ; for/next variable to FOR-2

      LD    BC,$0004     ; else an extra 4 bytes are needed.
      INC   HL           ; point to start of new space.

      CALL  L05D5        ; routine MAKE-ROOM creates it.
                       ; HL - first, DE- last

;; FOR-2
L08EA: INC    HL         ; address limit location
      POP    DE         ; retrieve limit value to DE.
      LD     (HL),E      ; insert low byte of limit.
      INC    HL
      LD     (HL),D      ; and then the high byte

      INC    HL         ; point to the looping line cell.
      LD     DE,($4002)  ; load DE with the current line from PPC

```

```

INC    DE            ; increment as iteration will start from the
                    ; next line at least.

LD     (HL),E        ; insert low byte of line number.
INC    HL
LD     (HL),D        ; insert high byte of line number.
RET                                ; return.

; -----
; THE 'NEXT' COMMAND ROUTINE
; -----

;; NEXT
L08F9: LD     HL,($4020)    ; fetch address of variable in BASIC from DEST.

CALL   L0B3B          ; routine LV-FIND finds the equivalent in the
                    ; variables area and returns the value in HL.
BIT    7,(IY+$00)      ; test ERR_NR
RET    Z              ; return with error.
                    ; will be 02 - variable not found.

; continue if LV-FIND found the variable - HL contains the value, DE points
; to the high byte of value location.

EX     DE,HL          ; value to DE, address to HL
DEC    HL              ; point to low byte
DEC    HL              ; point to the variable letter.
BIT    7,(HL)          ; - should have letter mask 111xxxxx

JR     Z,L0921          ; forward to ERROR-01 if not initialized by FOR.
                    ; - NEXT without FOR.

INC    DE              ; increment the integer value
                    ; no step or direction possible.
INC    HL              ; address first location
LD     (HL),E          ; store low byte of value.
INC    HL              ; next
LD     (HL),D          ; store high byte of value.
INC    HL              ;
LD     C,(HL)          ; pick up limit low
INC    HL              ;
LD     B,(HL)          ; and limit high.
PUSH   BC              ; save limit.
EX     (SP),HL         ; limit to HL, pointer to stack.

CALL   L0DCD          ; routine no-less compares HL DE
                    ; setting carry if HL is less.

POP    HL              ; retrieve the pointer from the stack.
RET    C              ; return if no more iterations possible >>

INC    HL              ; else address next location.
LD     C,(HL)          ; pick up low byte of line number
INC    HL              ; address next
LD     B,(HL)          ; pick up high byte of looping line.

JR     L0934          ; jump to GOTO to perform another
                    ; iteration

; ---

;; ERROR-01

```

```

L0921: RST  08H      ; ERROR restart
        DEFB  $00      ; NEXT without FOR

; -----
; THE 'RANDOMISE' COMMAND ROUTINE
; -----
; This command sets the seed to the supplied integer -32767 to 32767.
; In the absence of a parameter the FRAMES counter, related to the time
; the computer has been switched on, is used.

;; RANDOMISE
L0923: JR   NZ,L0929      ; forward to RAND-1 if parameter is
                        ; not zero.

        LD   BC,($401E)   ; else use value of system variable FRAMES.

;; RAND-1
L0929: LD   ($401C),BC     ; insert value in system variable SEED.
        RET                ; return.

; -----
; THE 'STOP' COMMAND ROUTINE
; -----

;; STOP
;; ERROR-9
L092E: RST  08H      ; ERROR restart
        DEFB  $08      ; - STOP statement executed.

; -----
; THE 'CONTINUE' COMMAND ROUTINE
; -----

;; CONTINUE
L0930: LD   BC,($4017)   ; fetch continuing line number from OLDPPC
                        ; and continue into GOTO routine.

; -----
; THE 'GO TO' COMMAND ROUTINE
; -----

;; GOTO
L0934: LD   ($4002),BC   ; set PPC to supplied line number.
        SET  3,(IY+$01)  ; update FLAGS - use K cursor.
        RET                ; return.

; -----
; THE 'RUN' COMMAND ROUTINE
; -----
; The RUN command may have an optional line number that will be passed to
; the GOTO routine before erasing any variables and executing the line
; (or first line after zero).

;; RUN
L093D: CALL L0934      ; routine GOTO sets up any supplied line number.
        JP   L065B      ; exit via CLEAR to erase variables.

; -----
; THE 'GO SUB' COMMAND ROUTINE
; -----

;; GOSUB

```

```

L0943: LD    HL,($4002)    ; fetch current line from PPC
      INC    HL            ; increment the line number
      EX     (SP),HL       ; place on machine stack
      ;
      PUSH   HL            ; push what was on the stack back up there.
      CALL   L0934         ; routine GOTO sets up a branch to the line
      ; number.
      LD     BC,$0006      ; and exit by a six-byte memory check.

```

```

;-----
; THE 'TEST ROOM' SUBROUTINE
;-----

```

```

; The ZX80 dates from the days when RAM chips cost a fortune and it came with
; only 1K of RAM, 1024 bytes.
; The screen could show 768 characters and to economize it is dynamic and
; initialized to a single newline ($76) by CLS. The TEST-ROOM routine has to
; allow for enough newlines to expand down to the bottom line and a few extra
; for the report codes "0/9999".
; The second entry point is from PRINT-A and the character is similarly
; in H and the line number in L.

```

```

;; TEST-ROOM

```

```

L094F: LD     HL,($4010)    ; fetch DF_END last location before
      ; spare memory.
      ADD     HL,BC         ; add the supplied overhead.
      EX      DE,HL        ; save the result in DE.

      LD      HL,($4025)    ; SPOSN-Y to L gives 24 - number
      ; of screen lines used so far.
      LD      H,A          ; preserve the accumulator in H

```

```

;; TEST-RM-2

```

```

L0958: LD     A,$13        ; load A with 19
      ADD     A,L          ; add to L to give the number of bytes
      ; required to fill rest of screen with
      ; newlines - plus a bit extra.
      LD      L,A          ; put result in L.
      LD      A,H          ; restore the accumulator.
      LD      H,$00        ; set H to zero.
      ADD     HL,DE        ; add this extra screen allowance
      ; to the previous result.
      SBC     HL,SP        ; subtract the stack pointer.
      RET     C            ; return if the stack pointer is
      ; above the estimate. All is well.

```

```

;

```

```

;; ERROR-4

```

```

L0963: RST    08H         ; ERROR restart
      DEFB    $03         ; No room

```

```

;-----
; THE 'RETURN' COMMAND ROUTINE
;-----

```

```

; As with all commands, there is only one value on the machine stack during
; command execution. This is the return address.
; Above the machine stack is the gosub stack that contains a line number
; (only one statement per line).

```

```

;; RETURN

```

```

L0965: POP     HL          ; drop the return address clearing the stack.
      POP     BC          ; drop a line number off the gosub stack.

```

```

    PUSH    HL            ; restore the machine stack.

    LD      A,B           ; test high byte of line number.
    CP      $3F           ; against the gosub stack end-marker.
    JR      NZ,L0934       ; back to GOTO if a valid line number.

    POP     HL            ; else collapse the machine stack.
    PUSH    BC            ; push the end-marker.
    PUSH    HL            ; restore the machine stack.

;; ERROR-07
    RST     08H           ; ERROR restart
    DEFB    $06           ; RETURN with no corresponding GO SUB.

; -----
; THE 'PRINT' COMMAND ROUTINE
; -----

;; PRINT
L0972: LD    A,(HL)       ; fetch the character
      CP     $76          ; compare to NEWLINE
      JP     Z,L071B       ; back to PRINT-CR if so.

;; PR-POSN-1
L0978: SUB    $D8          ; subtract ','
      ; (';' gives -1 and carry set)

      ADC    A,$00        ; convert the two separators to zero.
      JR     Z,L0991       ; forward to PR-POSN-2 with ';' and ','

      RST     28H         ; else SCAN-CALC evaluates expression.
      CALL    L06F1        ; routine PRINT-ITEM prints it.
      CALL    L001A        ; routine GET-CHAR gets following character.

      SUB     $D8          ; compare with ',' and test for
      ADC     A,$00        ; terminating separators.
      JR     Z,L0991       ; forward to PR-POSN-2 with ';' and ','

      CALL    L083D        ; routine CHECK-END errors with anything else.
      JP     L071B         ; jump to PRINT-CR for carriage return.

; ---

;; PR-POSN-2
L0991: CALL    NC,L0727     ; routine PO-FILL if comma control.

      RST     20H         ; NEXT-CHAR
      CP     $76          ; compare to NEWLINE
      RET     Z           ; return if so leaving print position
      ; unchanged.

      JR     L0978         ; else loop back to PR-POSN-1 to consider
      ; more sequences of positional
      ; controls and print items.

; -----
; THE 'INPUT' COMMAND ROUTINE
; -----

; INPUT must be used from a running program. It is not available as a
; direct command.

```

```

;; INPUT
L099A: BIT 7,(IY+$03) ; test PPC_hi - will be -2 if a direct command
JR NZ,L09CF ; forward if so, to ERROR-08

POP HL ; discard return address - L0447

LD HL,$4019 ; point to FLAGX

SET 5,(HL) ; signal input
RES 6,(HL) ; reset so as not to affect combine

LD A,($4001) ; fetch FLAGS to A

AND $40 ; isolate bit 6 - the result type

LD BC,$0002 ; allow two locations for numeric.

JR NZ,L09B4 ; skip forward to IN-PR-1 if numeric.

LD C,$04 ; allow two extra spaces for quotes.

;; IN-PR-1
L09B4: OR (HL) ; combine FLAG bit with FLAGX.
LD (HL),A ; and place result in FLAGS.

RST 30H ; BC-SPACES creates 2/4 locations.
RET NC ; return with problems.

LD (HL),$76 ; insert a newline at end.

LD A,C ; now test C - 2 (num) 4 (str).
RRCA ; 1 2
RRCA ; carry 1
JR C,L09C2 ; skip forward with numeric to IN-PR-3

LD (DE),A ; insert initial quote (chr$ 1) at DE
DEC HL ; decrease HL pointer
LD (HL),A ; insert closing quote.

;; IN-PR-3
L09C2: DEC HL ; decrease pointer
LD (HL),$B0 ; insert cursor inverse 'K'

LD A,($4025) ; SPOSN-Y
INC A ; allow a blank line
LD ($4012),A ; set DF-SZ

JP L02F7 ; jump back to ED-COPY

; ---

;; ERROR-08
L09CF: RST 08H ; ERROR restart
DEFB $07 ; INPUT can only be used in a program.

; -----
; THE 'POKE' COMMAND ROUTINE
; -----

;; POKE
L09D1: PUSH BC ; save result of first expression.

```

```
RST 28H      ; use SCAN-CALC to evaluate expression
           ; after the comma.
```

```
POP DE      ; restore destination address.
CALL L083D   ; routine CHECK-END
LD A,$4022   ; RESULT
BIT 7,(IY+$00) ; ERR_NR
RET Z        ; return if error
```

```
LD (DE),A    ; load memory location with A
RET          ; return
```

```
; -----
; THE 'SCANNING' ROUTINE
; -----
```

```
; The scanning routine is a continuation of RST 28.
; The B register has been set to zero as a starting priority.
; The HL register contains the character address CH_ADD.
; The addressed character is in A.
```

```
:: SCANNING
```

```
L09E1: LD C,B      ; make BC zero - the starting priority
           ; marker.
      PUSH BC      ; save on machine stack.
```

```
:: S-LOOP-1
```

```
L09E3: CALL L0D18   ; routine ALPHANUM
      JR C,L0A24    ; forward if a variable or digit. to S-VAR-NUM
```

```
; now consider negate (-) and perform '$0000 - value' if so.
```

```
LD BC,$0900      ; prepare priority $09, operation 'subtract'
LD D,C           ; set DE to $0000 for value to be stacked.
LD E,C           ;
SUB $DC          ; subtract the character '-'
JR Z,L0A17        ; forward with unary minus to S-PUSH-PO
```

```
; now consider 'not' and perform $FFFF - value if so.
```

```
DEC DE          ; set DE to $FFFF for value to be stacked.
LD B,$04        ; prepare priority 4, operation still 'subtract'
INC A           ; test for 'NOT' ?
JR Z,L0A17       ; forward with NOT to S-PUSH-PO
```

```
; now consider an opening bracket.
```

```
INC A           ; test the character.
JR Z,L0A1C       ; forward with '(' to S-BRACKET
           ; to evaluate the sub-expression recursively
           ; using SCANNING.
```

```
CP $27          ; commencing quote ?
JR NZ,L0A0E      ; forward to S-ABORT if not, as all valid
           ; possibilities have been exhausted.
```

```
; continue to evaluate a string.
```

```
RES 6,(IY+$01)   ; signal string result to FLAGS.
INC HL          ; step past the opening quote.
LD ($4022),HL    ; store the string pointer in
           ; system variable RESULT.
```

```

;; S-Q-CHAR
L0A06: RST 18H      ; NXT-CH-SP
      DEC  A        ; test for the string terminator.
      JR   Z,L0A21   ; forward to S-CONT if found.      >>

      CP   $75       ; [ EDIT ]  SHIFT-ENTER
      JR   NZ,L0A06   ; loop back to S-Q-CHAR till terminator found.

; ---

; the branch was here when something unexpected appeared in the expression
; or, if from above, in the string.

;; S-ABORT
L0A0E: CALL L08AE    ; routine INS-ERR marks the spot.
      EXX           ;
      LD   BC,$0000  ; this forces the zero priority marker down
                        ; from the stack.
                        ; Note. just setting B to zero should do.
      JR   L0A4C      ; forward to S-LOOP to balance and exit

; ---

; the ZX80 juggles with expression components using just the machine stack
; pushing first the value and then the priority/operator beneath.
; As with all ZX computers, provided there is enough memory, an expression of
; unlimited complexity can be evaluated.

;; S-PUSH-PO
L0A17: PUSH DE       ; push the value ($0000 if '-', $FFFF if 'NOT')
      PUSH BC        ; then push the priority and operator.

;; SCAN-LOOP
L0A19: RST 20H       ; NEXT-CHAR advances the character address.
      JR   L09E3      ; back to S-LOOP-1

; ---

;; S-BRACKET
L0A1C: CALL L0049     ; routine BRACKET evaluates expression
                        ; inside the brackets checking for
                        ; terminator using SCANNING
                        ; recursively.
      JR   L0A37      ; forward to S-OPERTR

; ---

; the branch was here when the end of a string had been found.

;; S-CONT
L0A21: RST 18H       ; NXT-CH-SP
      JR   L0A37      ; forward to S-OPERTR to consider comparisons

; ---

;; S-VAR-NUM
L0A24: CP  $26        ; compare to 'A'
      JR   C,L0A2D     ; forward if numeric to S-DIGIT

; present character is alpha

      CALL L0AAD       ; routine LOOK-VARS

```



```

        JR    L0A37        ; forward to S-OPERTR

; ---

;; S-DIGIT
L0A2D: CALL  L0679        ; routine INT-TO-HL
        CALL  C,L08AE      ; routine INS-ERR with overflow.
        SET   6,(IY+$01)   ; signal numeric result in FLAGS

;; S-OPERTR
L0A37: CALL  L001A        ; routine get-char

        EXX
        LD    BC,$0000    ; prepare zero priority in case not an operator
                        ; in which case at end of expression

        SUB   $DC         ; reduce by '-'
        JR    C,L0A4C      ; forward if less than an operator to S-LOOP

        CP    $0A         ; compare to ten.
        JR    NC,L0A4C     ; forward if higher than nine to S-LOOP

; leaves ten operators -, +, *, /, AND, OR, **, =, >, <.

        LD    C,A         ; transfer operation to C, register B is zero.
        LD    HL,L0AA3     ; address table of priorities.
        ADD   HL,BC        ; index into table.
        LD    B,(HL)       ; pick up the priority.

;; S-LOOP
L0A4C: POP    DE          ; pop the previous priority/operation
        LD    A,D          ; priority to A
        CP    B            ; compare with current priority B
        JR    C,L0A88      ; forward to S-TIGHTER if current priority is
                        ; higher

; else this is the correct place in the expression to perform this operation.

        AND   A            ; first test for zero priority marker

        EXX                ;
        RET    Z           ; return if so, HL is result. >>>>>
        EXX                ;

        BIT   7,(IY+$01)   ; FLAGS
        JR    Z,L0A6F      ; forward if checking syntax to S-SYNTEST

; but in runtime the operation is performed.

        LD    D,$00        ; prepare to index.
        LD    HL,L0D1F     ; address the table of operators and addresses.

        ADD   HL,DE        ; index twice using the operation code.
        ADD   HL,DE        ; as there are two bytes per entry.
        LD    E,(HL)       ; pick up low byte of address.
        INC   HL           ; next location.
        LD    D,(HL)       ; get high byte of address.

        LD    HL,L0A7F     ; the return address S-INS-VAL
        EX    (SP),HL      ; goes to the stack and argument to HL

        PUSH  DE           ; now push the address of the routine.

```

```
LD    DE,($4022)    ; pick up last value from RESULT
RET                                ; and make an indirect jump to
                                ; the routine.      >>>>>>>>
```

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

```
:: S-SYNTEST
```

```
L0A6F: LD    A,E      ; get the last operation code
      CP    $0A      ; compare to ten - sets carry if numeric
      RRA      ; carry to bit 7
      RRA      ; carry to bit 6
      XOR    (IY+$01) ; exclusive or with FLAGS
      AND    $40      ; isolate bit 6 - the result type.

      EXX      ;
      CALL   NZ,L08AE  ; routine INS-ERR if not of same type.
      EXX      ;

      POP    HL      ; fetch the last value from machine stack
```

```
; >>>>>>>>
; Note. this is also the return address from mathematical and string
; comparisons, see above, in which case HL will contain the result and BC
; the priority/operation.
```

```
:: S-INS-VAL
```

```
L0A7F: LD    ($4022),HL ; place value in system variable RESULT
      SET    6,(IY+$01) ; signal numeric result to FLAGS
      JR     L0A4C      ; back to S-LOOP
```

```
; ---
```

```
:: S-TIGHTER
```

```
L0A88: PUSH   DE      ; push lower priority

      LD     A,C      ; fetch operator
      BIT    6,(IY+$01) ; test FLAGS
      JR     NZ,L0A9A  ; forward if numeric to S-NEXT.

      ADD    A,$03     ; augment nos-eql to str-eql etc.
      LD     C,A      ; and put back in C

      CP     $0A      ; compare to ten - start of string comparisons

      EXX      ;
      CALL   C,L08AE   ; routine INS-ERR if lower
                        ; a$ * b$ is invalid but so too
                        ; is a$ + b$ (no string concatenation)
      EXX      ;
```

```
:: S-NEXT
```

```
L0A9A: LD     HL,($4022) ; fetch RESULT to HL
      PUSH   HL      ; push intermediate result
      PUSH   BC      ; and then priority/operator
      EXX      ;
      JP     L0A19    ; jump back to SCAN-LOOP
```

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

```
; THE 'TABLE OF PRIORITIES'
```

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

```
; Table of mathematical priorities that dictate, in the absence of brackets,
```

; the order in which operations are performed.
; unary minus (priority \$09) and NOT (priority \$04) are handled directly.

```
:: TAB-PRIO
L0AA3: DEFB $06      ; $00 subtract
      DEFB $06      ; $01 addition
      DEFB $08      ; $02 multiply
      DEFB $07      ; $03 division
      DEFB $03      ; $04 and
      DEFB $02      ; $05 or
      DEFB $0A      ; $06 to-power
      DEFB $05      ; $07 nos-eql
      DEFB $05      ; $08 no-grtr
      DEFB $05      ; $09 no-less
```

```
; -----
; THE 'LOOK-VARS' SUBROUTINE
; -----
```

```
:: LOOK-VARS
L0AAD: PUSH HL      ; * push pointer to first letter

      LD HL,$4001   ; address FLAGS
      RES 5,(HL)    ; update FLAGS - signal not a function yet.
                      ; but no use is made of this flag bit.
      SET 6,(HL)    ; update FLAGS - presume a numeric result.
      RST 18H       ; NXT-CH-SP
      CP $0D        ; compare to '$' ?
      JP Z,L0B30    ;; JUMP forward with match to STRING

      CP $DA        ; compare to '(' ?
      JP Z,L0B2B    ;; JUMP forward with match to ARRAY
```

; that leaves three types of integer plus functions.

```
:: V-CHAR
L0AC0: CALL L0D18    ; routine ALPHANUM
      JR NC,L0AC8    ; forward when not alphanumeric to FUNC-LOOP.

      RST 18H       ; fetch NXT-CH-SP.
      JR L0AC0      ; loop back to V-CHAR for more.
```

; ---

```
:: FUNC-LOOP
L0AC8: CP $DA        ; compare to '(' ?
      JR Z,L0AD6     ; forward with a match to FUNC-SRCH

      CP $0D        ; compare to '$' ?
      JP NZ,L0B35    ;; JUMP forward if not to V-SYN
```

; but if this is a string function such as CHR\$ then the bracket must follow.

```
      RST 18H       ; NXT-CH-SP
      CP $DA        ; compare to '(' ?
      JR NZ,L0B27    ; forward if not to FUNC-ERR.
```

; This has the correct format for a function and an exact match must now be
; made to one of the entries in the functions table.

```
:: FUNC-SRCH
L0AD6: LD DE,L0BC0 - 1 ; point to location before TAB-FUNC
```

```
;; FUNC-LOOP
LOAD9: POP   HL           ; pop pointer to first character in command
        PUSH  HL           ; and push again.
```

```
;; FUNC-CHAR
LOADB: LD    C,(HL)       ; fetch command character to C.
        CALL  L0055        ; routine CH-ADD-LP advances CH-ADD
                           ; to next non-space position.
        INC   DE           ; increment position in table
        LD    A,(DE)       ; fetch table character to A.
        CP    C           ; compare with one in command.
        JR    Z,L0ADB      ; loop back with match to FUNC-CHAR
                           ; e.g. PEEK

        AND   $3F          ; cancel any inversion.
        CP    C           ; and compare again
        JR    NZ,L0AEE     ; skip if no match to FUNC-NEXT.

        LD    A,$DA        ; load with '('
        CP    (HL)         ; compare to next valid character
        JR    Z,L0AF9      ; forward with success to FUNC-MTCH.
```

```
;; FUNC-NEXT
L0AEE: LD    A,(DE)       ; fetch next character from table.
        AND   A           ; test for zero end-marker.
        JR    Z,L0B27     ; forward if at end of table to FUNC-ERR.

        INC   DE           ; else increment pointer.
        RLA             ; test for inverted bit.
        JR    NC,L0AEE     ; loop back to FUNC-NEXT
                           ; until new token found.

        INC   DE           ; increment pointer.
                           ; to skip address in table.

        JR    LOAD9        ; loop back to FUNC-LOOP
                           ; which begins by skipping the
                           ; remaining address byte.
```

```
; ---
```

; A function such as PEEK has been found with the necessary opening bracket.

```
;; FUNC-MTCH
L0AF9: PUSH  DE           ; save pointer to address within
                           ; table.

        CALL  L0049        ; routine BRACKET evaluates an
                           ; expression within brackets in command.
                           ; result in HL

        POP   DE           ; retrieve table address pointer.

        EX    (SP),HL      ; result to stack, discarding command
                           ; character pointer.
        LD    HL,$4001     ; load with address FLAGS

        LD    A,(DE)       ; fetch the last inverted character.
        XOR   (HL)         ; XOR with FLAGS
        AND   $40          ; isolate bit 6 - result type.
```

```

JR    NZ,L0B27    ; to FUNC-ERR to insert an error with
                ; an argument type mismatch.

SET    5,(HL)     ; update FLAGS signal a function has been found
                ; but no use is made of this ?????
SET    6,(HL)     ; default the result type to be numeric.

LD     A,(DE)     ; fetch last character
AND    $3F        ; lose the indicator bits.

CP     $0D        ; is character '$' ?
                ; i.e. CHR$, STR$ or TL$.
JR     NZ,L0B15    ; forward with numeric function results
                ; to FUNC-SYN.

RES    6,(HL)     ; else set FLAGS to indicate a string
                ; result is expected.

;; FUNC-SYN
L0B15: BIT    7,(HL)    ; test FLAGS checking syntax?

POP    HL         ; restore RESULT of expression in brackets.
RET    Z          ; return if checking syntax.    >>

LD     HL,L0BBA    ; else the routine INS-RSLT
PUSH   HL         ; is pushed on the machine stack

EX     DE,HL      ; HL now points to table entry.
INC    HL         ; point to address low byte.
LD     E,(HL)     ; pick up the low byte.
INC    HL
LD     D,(HL)     ; pick up the high byte.
PUSH   DE         ; push routine address on stack.

LD     HL,($022)   ; load HL with argument from RESULT
                ; either integer or string pointer.
RET     ; indirect jump to routine and then
                ; to INS-RSLT .

; ---

;; FUNC-ERR
L0B27: POP    HL    ; balance stack.
        JP    L08AE    ; jump back to INS-ERR

; -----

;; ARRAY
L0B2B: CALL   L0049    ; routine BRACKET evaluates expression
        JR    L0B35    ; skip to V-SYN

; ---

;; STRING
L0B30: RES    6,(IY+$01) ; FLAGS signal string result.
        RST    18H      ; NXT-CH-SP

;; V-SYN
L0B35: POP    HL      ; * restore pointer to first letter
        BIT    7,(IY+$01) ; check FLAGS
        RET    Z      ; return if checking syntax
                ; but continue in run-time

```

; also called from NEXT and LET
; HL points to first letter of variable in the command.

```
:: LV-FIND
L0B3B: LD    C,(HL)      ; C first character
      INC    HL
      LD     A,(HL)      ; A second character

      PUSH   HL          ; save pointer to character 2

      CP     $DA         ; is second character '(' ?
      JR     NZ,L0B5C     ; forward if not to LV-ENCODE with strings and
                          ; simple numeric variables.
```

; an array

```
      PUSH   BC          ; save BC on stack
      LD     BC,($4026)   ; fetch character address CH_ADD
      PUSH   BC          ; and save that on stack as well.

      CALL   L0025        ; routine EVAL-EXPR evaluates the
                          ; expression after the current '('
                          ; disturbing CH_ADD

      POP    HL          ; restore original value of CH_ADD
      LD     ($4026),HL   ; and backdate CH_ADD system variable.
      POP    BC          ; restore the letter in BC.

      LD     HL,$4000     ; address system variable ERR_NR
      BIT    7,(HL)       ; test if $FF has been disturbed by eval_expr.
      JR     NZ,L0B6B     ; forward if not to V-RUN.

      LD     (HL),$02     ; else insert the code for subscript error
      POP    HL          ; balance the stack
      RET                     ; return with error set.    >>
```

; ---

; encode the variable type into bits 5-7 of the letter.

```
:: LV-ENCODE
L0B5C: RES    5,C         ; presume type string
      CP     $0D         ; is second character '$' ?
      JR     Z,L0B6B     ; forward if so to V-RUN

      SET    6,C         ; presume long-named numeric.

      CALL   L0D18        ; routine ALPHANUM test second character.

      JR     C,L0B6B     ; forward if so to V-RUN

      SET    5,C         ; else mark as simple numeric or for/next
```

```
:: V-RUN
L0B6B: LD     HL,($4008)   ; point HL to the first variable from VARS.
```

```
:: V-EACH
L0B6E: LD     A,(HL)      ; fetch letter/marker
      AND     $7F         ; reset bit 7 to allow simple numeric variables
                          ; to match against FOR-NEXT variables.
      JP     Z,L0CD0      ; if character was $80 then forward to ERROR-02
```

```

; Variable not found.

CP   C           ; else compare to first letter in command
JR   NZ,L0B93     ; forward if no match to V-NEXT

RLA           ; rotate A to left and then
ADD  A,A         ; double to test bits 5 and 6.

JP   M,L0BA4     ; forward to STK-VAR with
; all single letter numeric variables
; including for/next and arrays.

JR   NC,L0BB8     ; forward to STR-RSLT with string.

; that leaves long-named variables (mask 010xxxxx)
; that have to be matched in full.

POP  DE         ; take a copy of pointer.
PUSH DE         ; to 2nd character in BASIC area.
PUSH HL         ; save 1st letter pointer in vars area.

;; V-MATCHES
L0B81: INC  HL     ; point to next vars character.
LD   A,(DE)      ; fetch each BASIC char in turn.
INC  DE         ; advance BASIC pointer
CP   (HL)        ; compare to character in variable
JR   Z,L0B81     ; back if the same to V-MATCHES

OR   $80        ; try a match on inverted character.
CP   (HL)        ; compare to variable
JR   NZ,L0B92     ; forward to V-GET-PTR without full
; match.

LD   A,(DE)      ; check that the end of name in BASIC
; has been reached.

CALL L0D18       ; routine ALPHANUM checks that no
; more letters follow.

JR   NC,L0B9B     ; forward to V-FOUND-1 with a full
; match on an inverted long name.

; else continue the search

;; V-GET-PTR
L0B92: POP  HL     ; fetch the pointer.

;; V-NEXT
L0B93: PUSH BC     ; save B and C

CALL L0624       ; routine NEXT-ONE points DE at next
; variable

EX  DE,HL       ; switch pointers.
POP  BC         ; retrieve B and C.
JR   L0B6E      ; back for another search to V-EACH.

; ---

;; V-FOUND-1
L0B9B: POP  DE     ; drop saved var pointer

```

```

;; V-FOUND-2
L0B9C: POP    DE            ; drop pointer to second character

;; V-FOUND-3
L0B9D: INC     HL            ; advance to value.
      LD      E,(HL)        ; fetch low byte to E
      INC     HL            ;
      LD      D,(HL)        ; fetch high byte to D.
      EX      DE,HL         ; value to HL
      JR      L0BBA         ; forward to INS-RSLT

; ---

; simple 011xxxxx, array 101xxxxx, for/next 111xxxxx

;; STK-VAR
L0BA4: JR      C,L0B9C      ; back to V-FOUND-2 above with simple
                        ; and FOR/NEXT variables.

;; SV-ARRAYS
      EX      (SP),HL       ; save pointer to letter on stack discarding
                        ; the second letter pointer
      LD      HL,($4022)    ; fetch argument within brackets from RESULT

      RLC     H             ; test the high byte.

      POP     DE            ; retrieve pointer to letter
      JR      NZ,L0BBE      ; forward to ERROR-03 subscript error
                        ; if subscript > 255

      INC     DE            ; point to dimensions value - 1 byte
      LD      A,(DE)        ; fetch the max subscription

      CP      L             ; compare to low byte of argument.
      JR      C,L0BBE      ; forward if higher than max subscription
                        ; to ERROR-03.

      ADD     HL,HL         ; double the subscript 0 - 510
      ADD     HL,DE         ; add to variable pointer
                        ; now point to location before required cell.
                        ; if the first element is 0 then still pointing
                        ; at the max subscription byte.
      JR      L0B9D         ; back to V-FOUND-3 above.

; ---

; string type mask 100xxxxx

;; STR-RSLT
L0BB8: POP     DE           ; drop pointer to var.
      INC     HL            ; advance to first character of string.

;; INS-RSLT
L0BBA: LD      ($4022),HL   ; insert value/pointer into RESULT
      RET                ; return.

; ---

;; ERROR-03
L0BBE: RST     08H         ; ERROR restart
      DEFB    $02         ; subscript error

```



```

; -----
; THE 'INTEGRAL FUNCTIONS TABLE'
; -----
; Table of functions to be parsed and addresses.
; Parsed by LOOK-VARS.
; Inversion is with $80 (string argument)
; and with $CO (numeric argument).
; The TL$, "Truncate Left string", of "CABBAGE" is "ABBAGE".

;; TAB-FUNC
L0BC0: DEFB  $35,$2A,$2A,$F0 ; PEEK (+$C0)
        DEFW  L0C24          ; $0C24

        DEFB  $28,$2D,$37,$CD ; CHR$ (+$C0)
        DEFW  L0C28          ; $0C28

        DEFB  $28,$34,$29,$AA ; CODE (+$80)
        DEFW  L0C24          ; $0C24

        DEFB  $37,$33,$E9      ; RND (+$C0)
        DEFW  L0BED           ; $0BED

        DEFB  $39,$31,$8D      ; TL$ (+$80)
        DEFW  L0C38           ; $0C38

        DEFB  $3A,$38,$F7      ; USR (+$C0)
        DEFW  L06F0           ; $06F0

        DEFB  $38,$39,$37,$CD ; STR$ (+$C0)
        DEFW  L0C10           ; $0C10

        DEFB  $26,$27,$F8      ; ABS (+$C0)
        DEFW  L0DF2           ; $0DF2

        DEFB  $00              ; zero end-marker

; -----
; THE 'RND' FUNCTION
; -----
; e.g. LET LOTTERYNUMBER = RND (49) produces a random number in the range
; 1 to 49.
; the routine has two stages -
; First the seed is fetched and manipulated in such a way that it cycles through
; every value between 0 and 65535 in a pseudo-random way before repeating the
; sequence. If the seed fetched is zero it is set to 65536-77.
; The multiplicand used is 77 and any overflow is subtracted from the
; register result.

;; RND
L0BED: PUSH  HL              ; * save the integer parameter e.g. 49.
        LD   HL,($401C)      ; fetch the 'seed' from SEED.
        LD   DE,$004D        ; place 77 in DE
        LD   A,H             ; test the seed
        OR   L               ; for value zero
        JR   Z,L0C03         ; forward if zero.

        CALL L0D55           ; routine MULT16 multiplies seed by 77
                                ; BC contains zero or overflow

        AND  A              ; clear carry flag.
        SBC  HL,BC          ; subtract any overflow from lower 16 bits

```

```

    JR    NC,L0C05      ; forward if no carry to RND-3

    INC    HL          ; increase seed value.
    JR    L0C05        ; forward to RND-3

; ---

;; RND-2
L0C03: SBC    HL,DE      ; form number $FFB3 if seed is zero.

;; RND-3
L0C05: LD     ($401C),HL ; store new value of SEED

; now multiply the new seed by the argument to give result-1 in BC.

    POP    DE          ; * restore argument

    CALL   L0D55        ; routine MULT16 multiplies HL by DE
                        ; returning in BC, for the example, 0-48

    LD     H,B          ; transfer BC
    LD     L,C          ; to HL - the result register.
    INC    HL           ; increment - make range start with 1.
    RET                     ; return

; -----
; THE 'STR$' FUNCTION
; -----
; the function produces a string comprising the characters that would appear
; if the numeric argument were printed.
; So seven characters e.g. "-10000" terminated by the null character ($01)
; is the maximum amount of characters required.
; Note. that for this reason the ZX80, unlike the ZX81 and ZX Spectrum, is able
; to have four tabstops across the 32 character screen.

;; str$
L0C10: EXX
    LD     BC,$0007      ; 7 characters required at most.
    RST    30H          ; routine BC-SPACES
    JR     NC,L0C34      ; forward to NULL-STR if not enough
                        ; memory.

    PUSH   DE           ; * save start of new space

    EXX                ; switch in other set
    LD     B,H          ; transfer argument to BC
    LD     C,L          ; register.

    CALL   L06A1        ; OUT-NUM-1 prints at this DE in WKG Space.

    EXX                ; switch back

    LD     A,$01        ; prepare the terminating ""
    LD     (DE),A       ; and place at end of string.

;; POP-RET
L0C22: POP    HL        ; * restore result pointer.
    RET                     ; return.

; -----
; THE 'CODE' AND 'PEEK' FUNCTIONS

```

```

; -----
; Two functions in one subroutine.
; CODE with HL pointing to start of string.
; and also,
; PEEK with HL pointing to a memory address.
; The return value is in HL.

;; CODE
;; PEEK
LOC24: LD    L,(HL)      ; parameter is in HL.
      LD    H,$00      ;
      RET                      ; return with result in HL.

; -----
; THE 'CHR$' FUNCTION
; -----
; this function returns the null-terminated single-character string that
; corresponds to the integer argument e.g. CHR$(38) returns "A".

;; chr$
LOC28: LD    BC,$0002    ; two locations required.
      LD    A,L          ; character to A.
      RST    30H         ; BC-SPACES creates two locations
                        ; in WORKSPACE
      JR     NC,LOC34     ; forward to NULL-STR if no room.

;; NULL-PTR
LOC2F: LD    (HL),$01    ; insert the "" terminator at last new location
      DEC    HL          ; decrease the pointer.
      LD    (HL),A       ; insert the character.
      RET                      ; return with HL pointing to string.

; ---

;; NULL-STR
LOC34: LD    HL,LOC2F + 1 ; point to the null string at NULL-PTR + 1
                        ; in the above code.
      RET                      ; return.

; -----
; THE 'TL$' FUNCTION
; -----
; This limited string slicing function returns the tail of a string starting
; at the second character and the null string otherwise.
; It requires no string workspace.

;; tl$
LOC38: LD    A,(HL)      ; fetch first character of string
      DEC    A           ; decrement it.
      RET    Z           ; return if was CHR$ 1 - the null string.

      INC    HL          ; else increase the string pointer
      RET                      ; return with HL pointing at result.

; -----
; THE 'LET' ROUTINE
; -----
; This subroutine is called from the FOR command and the CLASS-02 routine
; to create the variable.

;; LET

```

```

LOC3D: BIT    7,(IY+$00)    ; test ERR_NR
        RET    Z            ; return if not $FF

```

; proceed if no errors so far.

```

        PUSH   BC          ; save start val

        LD     HL,($4020)   ; fetch location of letter in BASIC from DEST

        CALL   L0B3B        ; routine LV-FIND will set error

        LD     HL,$4000     ; ERR_NR
        LD     A,(HL)
        CP     $02         ; compare to 2 - subscript out of range

        JR     Z,L0C22      ; back to POP-RET if so    >>>

```

; continue with variable not found or OK.

```

        RLA              ; test for $FF??
        BIT    6,(IY+$01)  ; test bit 6 FLAGS - affects zero flag only.
                        ; zero if string NZ if numeric
        JR     C,L0C93     ; forward if error was $FF to L-EXISTS

```

; continue if variable does not exist.

```

        LD     (HL),$FF    ; cancel the error as variable will be created.

        JR     Z,L0CA3     ; forward to L-STRING with string var.

```

; continue with numeric INTEGER variable

```

        LD     HL,($4020)   ; pick up destination from DEST
        LD     BC,$0002     ; set default space for integer contents
                        ; will be 3 including letter

```

; L-EACH-CH

```

LOC62: INC     BC          ; pre-increment character count.
        INC     HL          ; increment character pointer in BASIC or
                        ; workspace.
        LD     A,(HL)       ; fetch the character.

        CALL   L0D18        ; routine ALPHANUM check if "[0-Z]"
        JR     C,L0C62      ; loop back if so to L-EACH-CH

        CP     $DA         ; is character '(' ?
        JR     Z,L0CD0      ; forward if so to ERROR-02 - var not found.
                        ; e.g. perhaps a function has been misspelled.

        RST     30H         ; BC-SPACES creates room for new INTEGER
                        ; variable at D-FILE - 1, the variables
                        ; end-marker.
        JR     NC,L0C22     ; back to POP-RET if not enough room

        PUSH   DE          ; save first new location    ***
        LD     HL,($4020)   ; fetch DEST the pointer to letter in command

        DEC     BC          ; reduce count by
        DEC     BC          ; the three bytes
        DEC     BC          ; for simple integer.

        DEC     DE          ; point to destination

```

```

LD    A,B        ; check if this is a one-character
OR    C          ; variable name from reduced count.

LD    A,$40      ; prepare mask 010xxxxx
JR    Z,LOC87    ; forward to L-SINGLE if is simple numeric.

LDIR          ; else copy all but one characters of name.
LD    A,(HL)     ; fetch last character
OR    $80        ; invert it
LD    (DE),A     ; place at last destination

LD    A,$60      ; prepare mask 011xxxxx

;; L-SINGLE
LOC87: POP    HL        ; restore first new location      ***

CALL    LOC89      ; routine L-MASK inserts masked letter.

EX    DE,HL      ;
DEC    DE        ;
                ; and continue to initialize variable contents.

; this branch is taken from below to overwrite contents.

;; L-NUMERIC
LOC8D: POP    HL        ; restore variable value
EX    DE,HL      ; HL points last location

LD    (HL),D     ; insert high byte.
DEC    HL        ; decrement the pointer.
LD    (HL),E     ; and insert low-byte value
RET          ; return. with HL addressing the value. >>>>

; ---

;; L-EXISTS
LOC93: JR    NZ,LOC8D    ; back to L-NUMERIC to overwrite variable
                ; if numeric type.

POP    HL        ; restore string

CALL    LOC84      ; routine L-LENGTH evaluates length of OLD
                ; string

LD    HL,($4022)  ; fetch string pointer from RESULT
DEC    HL        ; decrement to point to letter.
CALL    L0624      ; routine NEXT-ONE calculate space to delete
JP     L0666      ; routine RECLAIM-2

; now continue into L-STRING to evaluate length of new string.

; ---

;; L-STRING
LOC83: POP    HL        ; restore pointer to contents.

;; L-LENGTH
LOC84: LD    A,$01    ; the search will be for the quote character.
LD    BC,$0001    ; initialize length to one.

;; L-COUNT

```

```

LOCA9: CP    (HL)        ; is addressed character null ?
      INC    HL          ; increase pointer.
      INC    BC          ; increase length.

      JR     NZ,LOCA9    ; loop back to L-COUNT till terminating
                          ; quote found.

      PUSH   HL          ; save pointer to end - null terminator.

      RST    30H         ; routine BC-SPACES creates room at end.
      EX     DE,HL       ; transfer end to DE.

      POP    HL          ; retrieve pointer to null terminator in E-LINE.

      RET     NC          ; return if no room was available.

      LDDR                   ; else copy string to the variables area.
      EX     DE,HL       ; HL now points to letter -1
      INC     HL          ; adjust
      LD     A,$A0        ; prepare mask %10100000

;; L-MASK
LOCB9: EX     DE,HL       ; save variable pointer in DE.
      LD     HL,($4020)   ; fetch destination in prog/e-line area
                          ; from system variable DEST
      XOR     (HL)        ; XOR mask with the letter.
                          ; Note. All letters have bit 5 set. The
                          ; preparation of masks must accommodate this.
      EX     DE,HL       ; variable pointer to HL,
      PUSH   AF          ; save masked letter

      CALL    LOD0D       ; routine REC-V80 reclaims
                          ; the previous $80 variables end-marker.

      POP    AF          ; pop the letter.
      DEC     HL          ; point to the letter in the variables area.
                          ; which is now one location lower than it was
                          ; a moment ago.
      LD      (HL),A      ; insert masked letter.

      LD      HL,($400C)   ; use D_FILE value
      LD      ($400A),HL  ; to update new E_LINE
      DEC     HL          ; step back.
      LD      (HL),$80    ; and insert the new variable $80 end-marker.
      RET                                ; return.

```

; ---

;; ERROR-02

LOCD0: POP HL ;

```

      RST    08H         ; ERROR restart
      DEFB   $01         ; variable name not found.

```

; -----

; THE 'DIM' COMMAND ROUTINE

; -----

; This routine creates a one-dimensional numeric array with up to
; 256 subscripts. Each is initialized to the integer zero.
; Note. array subscripts begin at zero. On later ZX computers subscripts began
; at 1 and there were no limits to the dimensions and subscripts other than
; memory.

```

;; DIM
LOCD3: AND    B            ; check high byte of parameter.
                        ; a maximum of 255 subscripts possible.
        JP    NZ,LOBBE     ; back to ERROR-03 - subscript error.

        PUSH  BC           ; save max subscript
        LD    H,B          ; transfer
        LD    L,C          ; to HL.

        INC   HL           ; increment to make range 1-256 from 0-255
        INC   HL           ; increment for letter and subscript byte
        ADD   HL,HL        ; double - allocates two bytes per integer
                        ; and two for the letter and subscript.

        LD    B,H          ; transfer count
        LD    C,L          ; to BC

        RST   30H         ; BC-SPACES
        JP    NC,L0C22     ; back to POP-RET if out of memory

        DEC   HL           ; point to last new location
        LD    D,H          ; transfer to DE
        LD    E,L          ; - the destination.
        DEC   DE           ; make DE one less than source.
        DEC   BC           ; reduce count
        DEC   BC           ; by two.
        LD    (HL),$00     ; insert a zero at source.

        LDDR                     ; block fill locations with zero.

        POP   BC           ; restore number of subscripts
        LD    (HL),C       ; and place in location before data.
        LD    A,$80        ; prepare mask %100
        JR    L0CB9        ; back to L-MASK

; -----
; THE 'RESERVE' ROUTINE
; -----
; A continuation of the BC-SPACES RESTART.
; the number of bytes required is on the machine stack.

;; RESERVE
L0CF3: LD    HL,($400A)     ; fetch start of WKG Space from E_LINE
        PUSH HL            ; preserve location.

        LD    HL,($400C)    ; fetch location after WKG Space from D_FILE
        DEC   HL            ; point to last byte of WKG space.

        CALL  L05D5         ; routine MAKE-ROOM creates the space after
                        ; last byte sliding D-FILE up and updating
                        ; D_FILE, DF_EA and DF_END

        INC   HL            ; increase address
        INC   HL            ; by two bytes

        POP   BC           ; retrieve E_LINE which may have been updated
                        ; by pointers
        LD    ($400A),BC    ; restore E_LINE
        POP   BC           ; restore the number of bytes required.

        EX    DE,HL        ; switch - DE points to first

```

```

        INC    HL        ; make HL point to last new byte
        SCF          ; signal success
        RET          ; return

;-----
; THE 'RECLAIM THE EDIT LINE' SUBROUTINE
;-----
; Interestingly, Hugo Davenport refers to this subroutine in the manual
; by its Nine Tiles source code label X_TEMP.
; The second entry point deletes the old variables end-marker when creating
; a new variable immediately after this position.

;; REC-EDIT
L0D0A: LD    HL,($400C)    ; D_FILE

;; REC-V80
L0D0D: LD    DE,($400A)    ; E_LINE
        JP    L0663        ; RECLAIM-1

;-----
; THE 'ALPHA' SUBROUTINE
;-----

;; ALPHA
L0D14: CP    $26          ; compare to 'A'
        JR    L0D1A        ; forward to ALPHA-2 to compare
                        ; against 'Z'

;-----
; THE 'ALPHANUM' SUBROUTINE
;-----
; The zx80 character set makes this routine as straightforward as the one above
; as there is no gap between numerals and alphabetic characters.

;; ALPHANUM
L0D18: CP    $1C          ; compare to '0' - carry set if less

;; ALPHA-2
L0D1A: CCF          ; change to carry reset if less.
        RET    NC        ; return if less than '0'

        CP    $40        ; compare to character after 'Z'
        RET          ; return with carry set if in the
                        ; range '0' - 'Z'

;-----
; THE 'ARITHMETIC OPERATORS AND COMPARISONS' TABLE
;-----
; This table is indexed with the operator * 2 to access the address of the
; associated routine.

;; TAB-OPS
L0D1F: DEFW    L0D39        ; $00 subtract
        DEFW    L0D3E        ; $01 addition
        DEFW    L0D44        ; $02 multiply
        DEFW    L0D90        ; $03 division
        DEFW    L0DB5        ; $04 and
        DEFW    L0DBC        ; $05 or
        DEFW    L0D70        ; $06 to-power
        DEFW    L0DC3        ; $07 nos-eql
        DEFW    L0DCC        ; $08 no-grtr
        DEFW    L0DCD        ; $09 no-less

```



```

        DEFW  L0DD9      ; $0A str-eql
        DEFW  L0DDF      ; $0B str-grtr
        DEFW  L0DDE      ; $0C str-less

; -----
; THE 'SUBTRACTION' OPERATION
; -----
; offset $00 : subtract
; This operation simply uses the Z80's 16-bit register subtract instruction
; which sets the overflow flag if the lower 15 bits overflow.

;; subtract
L0D39: AND  A           ; clear carry flag.
      SBC  HL,DE        ; 16 bit subtraction.
      JR   L0D41        ; forward to RSLT-TEST

; -----
; THE 'ADDITION' OPERATION
; -----
; offset $01 : add
; This operation simply uses the Z80's 16-bit register add instruction
; which sets the overflow flag in the manner above.

;; addition
L0D3E: AND  A           ; clear carry flag.
      ADC  HL,DE        ; 16 bit addition.

;; RSLT-TEST
L0D41: RET  PO          ; return if no twos-complement arithmetic
                        ; overflow.

;; ERROR-06
L0D42: RST  08H        ; ERROR restart
      DEFB $05         ; arithmetic overflow.

; -----
; THE 'MULTIPLICATION' OPERATION
; -----
; offset $02 : multiply
; the multiplication operation converts the two numbers HL and DE to positive
; integers, saving the result sign in the accumulator. If the positive result
; is above 32767 then an error code is produced else result is converted
; to the required sign, if necessary, as dictated by the accumulator.

;; multiply
L0D44: CALL L0DED       ; routine PREP-MD

      PUSH BC          ; save priority/operation
      EX  AF,AF'       ; save result sign

      CALL L0D55        ; routine MULT16

      JR   NZ,L0D8D     ; forward with overflow to POP6
                        ; clear the stack and produce ERROR-06

;; MULT-2
L0D4E: POP  BC          ; restore priority/operation
      EX  AF,AF'       ; restore result sign.
      RRA              ; test sign bit.
      RET  NC          ; return if result positive.

      JP   L0DF6        ; exit via routine TWOS-COMP

```

```

;-----
; THE 'SIXTEEN BIT MULTIPLICATION' ROUTINE
;-----
; Binary long multiplication by shifting and addition at the appropriate place
; if the multiplier bit is set.
; This important subroutine is called from the multiply routine, the to-power
; routine and twice from the RND function routine.
; It multiplies the 16 bit multiplier, HL, by the 16-bit multiplicand DE.
; Since the highest number the ZX80 can hold is 32767, the routine detects
; any overflow above this, resetting the zero flag - NZ with overflow.
; However if overflow occurs the routine does not abort, as does say the
; Spectrum, but continues to calculate the 32-bit result in B, C, H, L.
; Use is made of this by the RND routine.

```

```

;; MULT16

```

```

L0D55: LD  B,H      ; transfer HL to BC
      LD  C,L      ; register.

      LD  A,$10     ; count 16 bits.
      LD  HL,$0000  ; initialize result register.

```

```

;; MULT-LP

```

```

L0D5C: ADD  HL,HL    ; shift result left.
      RL  C          ; shift multiplier
      RL  B          ; to the left.
      ; and capture any overflow.
      JR  NC,L0D67   ; skip addition if no carry to MULT-SKIP.

      ADD  HL,DE     ; else add in multiplicand for this bit
      JR  NC,L0D67   ; forward if no overflow.

      INC  BC        ; capture overflow in BC

```

```

;; MULT-SKIP

```

```

L0D67: DEC  A        ; decrement bit count.
      JR  NZ,L0D5C   ; loop back for all 16 bits to MULT-LP.

      LD  A,H        ; test for a
      AND  $80       ; negative result.

      OR  B          ; test for any
      OR  C          ; intermediate overflow

      RET            ; return with zero flag set
      ; for success.

```

```

;-----
; THE 'TO-POWER' OPERATION
;-----

```

```

; offset $06 : to-power
; This routine raises HL to the power DE, by performing a multiplication
; for each unit of the power. For the integer range supported this is quite
; adequate with 2**14 returning the result without any noticeable delay
; and 1**32767 blacking the screen out for no more than a second.
; Note also that
; 0 ** 0 = 1.
; 0 ** +n = 0.
; 0 ** -n = arithmetic overflow.

```

```

;; to-power

```

```

L0D70: BIT  7,D      ; test if second number negative.

```

```

JR    NZ,L0D42    ; back to ERROR-06 if so.

XOR    A          ; initialize sign flag
CALL   L0DF2      ; routine ABS - makes HL positive.
                ; A holds 1 if HL was negative else 0.

AND    E          ;
EX     AF,AF'     ; save result

PUSH   BC         ; save priority/operation

LD     B,D        ; transfer power
LD     C,E        ; to BC

EX     DE,HL      ; transfer number to DE
LD     HL,$0001   ; initialize result.

;; POWER-LP
L0D81: DEC    BC    ; decrement power counter.
      BIT    7,B    ; check when zero passed.

JR     NZ,L0D4E    ; back when finished to MULT-2
                ; to test result.      >>

PUSH   BC         ; save counter.

CALL   L0D55      ; routine MULT16

POP    BC         ; restore counter.
JR     Z,L0D81    ; loop while no overflow exists from
                ; the multiplication to POWER-LP.

;; POP6
L0D8D: POP    BC    ; restore priority/operation
JR     L0D42     ; back to ERROR-06 - arithmetic overflow.

; -----
; THE 'DIVISION' OPERATION
; -----
; offset $03 : division
; Binary long division by shifting and subtraction at the appropriate place,
; setting correct quotient bit if the subtraction goes.
; dividend (HL) / divisor (DE) = quotient (HL)

;; division
L0D90: LD     A,D    ; test divisor for zero
      OR     E      ; avoiding division by zero.

JR     Z,L0D42     ; to ERROR-06 - arithmetic overflow
                ; if so.

CALL   L0DED      ; routine PREP-MD converts HL and DE to 15-bit
                ; integers and records the result sign in A.

PUSH   BC         ; save the priority/operation.

RRA                     ; sets carry if a negative result.

ADC    HL,HL      ; pick up the carry in HL, (bit 15 was reset)
LD     A,H        ; transfer modified dividend to
LD     C,L        ; registers A and C.

```

```

        LD    HL,L0000    ; initialize 'accumulator' to zero.
        LD    B,$10      ; sixteen bits including sign bit.

;; DIV-1
L0DA2: ADC    HL,HL        ;
        SBC    HL,DE        ; subtract divisor.
        JR     NC,L0DA9     ; skip forward if subtraction goes to DIV-2.

        ADD    HL,DE        ; add back divisor.

;; DIV-2
L0DA9: RL     C            ; as dividend bits are shifted out, the
        RLA                     ; result bits are shifted in.
        DJNZ   L0DA2        ; back for all 16 bits.

; note after 16 bits the final RLA retrieves the sign

        LD     H,A          ; transfer result in A and C
        LD     L,C          ; to HL
        INC    HL           ; increment

        POP    BC           ; restore priority/operation.
        RET    C            ; return if .

        JR     L0DF6        ; else forward to TWOS-COMP.

```

```

; -----
; THE 'BITWISE AND' OPERATION
; -----
; offset $04 : and

```

```

;; and
L0DB5: LD     A,H          ;
        AND    D           ;
        LD     H,A         ;

        LD     A,L         ;
        AND    E           ;
        LD     L,A         ;

        RET                     ;

```

```

; -----
; THE 'BITWISE OR' OPERATION
; -----
; offset $05 : or

```

```

;; or
L0DBC: LD     A,H          ;
        OR     D           ;
        LD     H,A         ;

        LD     A,L         ;
        OR     E           ;
        LD     L,A         ;

        RET                     ;

```

```

; -----
; THE 'THREE NUMERIC COMPARISON' OPERATIONS
; -----
; offsets $07 - nos-eql, $08 - no-grtr, $09 - no-less.

```

```

;
; for example, PRINT 2=2 gives result -1 (true)

;; nos-eql
L0DC3: AND  A      ; prepare to subtract.
      SBC  HL,DE    ; subtract the two numbers.

;; SET-RSLT
L0DC6: LD   HL,$FFFF ; prepare true result.
      RET  Z        ; return true result, $FFFF, in HL
                        ; if remainder was zero.

      INC  HL        ; else increment to $0000
      RET                    ; return false result, zero in HL.

; ---

;; no-grtr
L0DCC: EX   DE,HL    ; swap values and continue into ...

;; no-less
L0DCD: AND  A      ; prepare for true subtraction
      SBC  HL,DE    ; subtract using registers
      LD   A,H      ; fetch MSB
      RLA          ; test the sign bit without affecting P/V flag
      JP   PO,L0DD6 ; skip to TEST-HL with no overflow

      CCF          ; complement the carry flag

;; TEST-HL
L0DD6: SBC  HL,HL    ; result HL will be $0000 false or $FFFF true
                        ; with carry.
      RET          ; return

; -----
; THE 'THREE STRING COMPARISON' OPERATIONS
; -----
; offsets $0A - str-eql, $0B - str-grtr, $0C - str-less.

;; str-eql
L0DD9: CALL L0DE4    ; routine STR-CMP
      JR   L0DC6     ; to SET-RSLT

; ---

;; str-grtr
L0DDE: EX   DE,HL    ; swap the two string pointers

;; str-less
L0DDF: CALL L0DE4    ; routine STR-CMP
      JR   L0DD6     ; back to TEST-HL

; -----
; THE 'STRING COMPARISON' SUBROUTINE
; -----

;; STR-CMP
L0DE4: LD   A,(DE)    ; fetch character of 2nd string.
      CP   (HL)      ; compare to first.
      RET  NZ        ; return with mismatch, carry flag
                        ; shows the comparison.

```

```

    DEC  A           ; test for the null string chr$ 1.
    RET  Z           ; return as both strings have
                        ; terminated - an exact match.

    INC  DE          ; else increase
    INC  HL          ; both the string pointers.

    JR   LODE4       ; and loop back to STR-CMP till one
                        ; of the two conditions is met.

; -----
; THE 'PREPARE TO MULTIPLY OR DIVIDE' SUBROUTINE
; -----

;; PREP-MD
LODED: XOR  A        ; initialize a sign flag.
      CALL LODF1      ; call PREP-1 to prepare one number
                        ; and continue into routine to prepare
                        ; the other number.

;; PREP-1
LODF1: EX  DE,HL     ; switch numbers at each pass

; -----
; THE 'ABS' FUNCTION
; -----
; finds the absolute value of a signed integer.
; Negative numbers are twos complemented.
; e.g. minus 1 ($FFFF) is first 'ones complemented' to $0000 then incremented.

;; abs
LODF2: BIT  7,H      ; test sign of HL.
      RET  Z         ; return if positive.

      INC  A         ; sets bit 0 if result is negative.
                        ; two negatives will reset bit 0 when this
                        ; routine is used to prepare for multiplication.
                        ; 'a minus times a minus gives a plus'.

;; TWOS-COMP
LODF6: EX  AF,AF'    ; save running flag.

      LD   A,H       ; fetch high byte
      CPL           ; complement it
      LD   H,A       ; put back

      LD   A,L       ; fetch low byte
      CPL           ; complement
      LD   L,A       ; put back

      INC  HL        ; twos complement

      EX  AF,AF'     ; restore running flag.
      RET           ; return.

; -----
; THE 'SPARE' SECTION
; -----

; Start of Spare bytes

; End of Spare bytes.

```

```
;-----  
; THE 'CHARACTER SET'  
;-----
```

```
:: char-set
```

```
; $00 - space character      CHR$(0)
```

```
L0E00: DEFB  %00000000  
        DEFB  %00000000  
        DEFB  %00000000  
        DEFB  %00000000  
        DEFB  %00000000  
        DEFB  %00000000  
        DEFB  %00000000
```

```
; $01 - Character: ""      CHR$(1)
```

```
        DEFB  %00000000  
        DEFB  %00010100  
        DEFB  %00010100  
        DEFB  %00000000  
        DEFB  %00000000  
        DEFB  %00000000  
        DEFB  %00000000  
        DEFB  %00000000
```

```
; $02 - Character: mosaic   CHR$(2)
```

```
        DEFB  %11110000  
        DEFB  %11110000  
        DEFB  %11110000  
        DEFB  %11110000  
        DEFB  %11110000  
        DEFB  %11110000  
        DEFB  %11110000  
        DEFB  %11110000
```

```
; $03 - Character: mosaic   CHR$(3)
```

```
        DEFB  %00000000  
        DEFB  %00000000  
        DEFB  %00000000  
        DEFB  %00000000  
        DEFB  %11111111  
        DEFB  %11111111  
        DEFB  %11111111  
        DEFB  %11111111
```

```
; $04 - Character: mosaic   CHR$(4)
```

```
        DEFB  %11110000  
        DEFB  %11110000  
        DEFB  %11110000  
        DEFB  %11110000  
        DEFB  %00000000  
        DEFB  %00000000  
        DEFB  %00000000  
        DEFB  %00000000
```

```
; $05 - Character: mosaic   CHR$(5)
```

DEFB %00001111
DEFB %00001111
DEFB %00001111
DEFB %00001111
DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00000000

; \$06 - Character: mosaic CHR\$(6)

DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %11110000
DEFB %11110000
DEFB %11110000
DEFB %11110000

; \$07 - Character: mosaic CHR\$(7)

DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00001111
DEFB %00001111
DEFB %00001111
DEFB %00001111

; \$08 - Character: mosaic CHR\$(8)

DEFB %00001111
DEFB %00001111
DEFB %00001111
DEFB %00001111
DEFB %11110000
DEFB %11110000
DEFB %11110000
DEFB %11110000

; \$09 - Character: mosaic CHR\$(9)

DEFB %10101010
DEFB %01010101
DEFB %10101010
DEFB %01010101
DEFB %10101010
DEFB %01010101
DEFB %10101010
DEFB %01010101

; \$0A - Character: mosaic CHR\$(10)

DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %10101010
DEFB %01010101

DEFB %10101010
DEFB %01010101

; \$0B - Character: mosaic CHR\$(11)

DEFB %10101010
DEFB %01010101
DEFB %10101010
DEFB %01010101
DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00000000

; \$0C - Character: uk pound CHR\$(12)

DEFB %00000000
DEFB %00011110
DEFB %00100001
DEFB %01111000
DEFB %00100000
DEFB %00100000
DEFB %01111111
DEFB %00000000

; \$0D - Character: '\$' CHR\$(13)

DEFB %00000000
DEFB %00001000
DEFB %00111110
DEFB %01001000
DEFB %00111110
DEFB %00001001
DEFB %00111110
DEFB %00001000

; \$0E - Character: ':' CHR\$(14)

DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00001000
DEFB %00000000
DEFB %00000000
DEFB %00001000
DEFB %00000000

; \$0F - Character: '?' CHR\$(15)

DEFB %00000000
DEFB %00111110
DEFB %01000001
DEFB %00000110
DEFB %00001000
DEFB %00000000
DEFB %00001000
DEFB %00000000

; \$10 - Character: '(' CHR\$(16)

DEFB %00000000
DEFB %00000100

DEFB %00001000
DEFB %00001000
DEFB %00001000
DEFB %00001000
DEFB %00000100
DEFB %00000000

; \$11 - Character: ')' CHR\$(17)

DEFB %00000000
DEFB %00010000
DEFB %00001000
DEFB %00001000
DEFB %00001000
DEFB %00001000
DEFB %00010000
DEFB %00000000

; \$12 - Character: '-' CHR\$(18)

DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00111110
DEFB %00000000
DEFB %00000000
DEFB %00000000

; \$13 - Character: '+' CHR\$(19)

DEFB %00000000
DEFB %00000000
DEFB %00001000
DEFB %00001000
DEFB %00111110
DEFB %00001000
DEFB %00001000
DEFB %00000000

; \$14 - Character: '*' CHR\$(20)

DEFB %00000000
DEFB %00000000
DEFB %00101010
DEFB %00011100
DEFB %00001000
DEFB %00011100
DEFB %00101010
DEFB %00000000

; \$15 - Character: '/' CHR\$(21)

DEFB %00000000
DEFB %00000000
DEFB %00000010
DEFB %00000100
DEFB %00001000
DEFB %00010000
DEFB %00100000
DEFB %00000000

; \$16 - Character: '=' CHR\$(22)

```
DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00111110
DEFB %00000000
DEFB %00111110
DEFB %00000000
DEFB %00000000
```

; \$17 - Character: '>' CHR\$(23)

```
DEFB %00000000
DEFB %00000000
DEFB %00010000
DEFB %00001000
DEFB %00000100
DEFB %00001000
DEFB %00010000
DEFB %00000000
```

; \$18 - Character: '<' CHR\$(24)

```
DEFB %00000000
DEFB %00000000
DEFB %00000100
DEFB %00001000
DEFB %00010000
DEFB %00001000
DEFB %00000100
DEFB %00000000
```

; \$19 - Character: ':' CHR\$(25)

```
DEFB %00000000
DEFB %00000000
DEFB %00001000
DEFB %00000000
DEFB %00000000
DEFB %00001000
DEFB %00001000
DEFB %00010000
```

; \$1A - Character: ';' CHR\$(26)

```
DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00001000
DEFB %00001000
DEFB %00010000
```

; \$1B - Character: '.' CHR\$(27)

```
DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00000000
DEFB %00000000
```

DEFB %00001100
DEFB %00001100
DEFB %00000000

; \$1C - Character: '0' CHR\$(28)

DEFB %00000000
DEFB %00011100
DEFB %00100010
DEFB %01000001
DEFB %01000001
DEFB %00100010
DEFB %00011100
DEFB %00000000

; \$1D - Character: '1' CHR\$(29)

DEFB %00000000
DEFB %00001100
DEFB %00010100
DEFB %00000100
DEFB %00000100
DEFB %00000100
DEFB %00011110
DEFB %00000000

; \$1E - Character: '2' CHR\$(30)

DEFB %00000000
DEFB %00111110
DEFB %01000001
DEFB %00000001
DEFB %00111110
DEFB %01000000
DEFB %01111111
DEFB %00000000

; \$1F - Character: '3' CHR\$(31)

DEFB %00000000
DEFB %00111110
DEFB %01000001
DEFB %00000110
DEFB %00000001
DEFB %01000001
DEFB %00111110
DEFB %00000000

; \$20 - Character: '4' CHR\$(32)

DEFB %00000000
DEFB %00001100
DEFB %00010100
DEFB %00100100
DEFB %01000100
DEFB %01111111
DEFB %00000100
DEFB %00000000

; \$21 - Character: '5' CHR\$(33)

DEFB %00000000

DEFB %01111111
DEFB %01000000
DEFB %01111110
DEFB %00000001
DEFB %01000001
DEFB %00111110
DEFB %00000000

; \$22 - Character: '6' CHR\$(34)

DEFB %00000000
DEFB %00111110
DEFB %01000000
DEFB %01111110
DEFB %01000001
DEFB %01000001
DEFB %00111110
DEFB %00000000

; \$23 - Character: '7' CHR\$(35)

DEFB %00000000
DEFB %01111111
DEFB %00000001
DEFB %00000010
DEFB %00000100
DEFB %00001000
DEFB %00001000
DEFB %00000000

; \$24 - Character: '8' CHR\$(36)

DEFB %00000000
DEFB %00111110
DEFB %01000001
DEFB %00111110
DEFB %01000001
DEFB %01000001
DEFB %00111110
DEFB %00000000

; \$25 - Character: '9' CHR\$(37)

DEFB %00000000
DEFB %00111110
DEFB %01000001
DEFB %01000001
DEFB %00111111
DEFB %00000001
DEFB %00111110
DEFB %00000000

; \$26 - Character: 'A' CHR\$(38)

DEFB %00000000
DEFB %00111110
DEFB %01000001
DEFB %01000001
DEFB %01111111
DEFB %01000001
DEFB %01000001
DEFB %00000000

; \$27 - Character: 'B' CHR\$(39)

DEFB %00000000
DEFB %01111110
DEFB %01000001
DEFB %01111110
DEFB %01000001
DEFB %01000001
DEFB %01111110
DEFB %00000000

; \$28 - Character: 'C' CHR\$(40)

DEFB %00000000
DEFB %00011110
DEFB %00100001
DEFB %01000000
DEFB %01000000
DEFB %00100001
DEFB %00011110
DEFB %00000000

; \$29 - Character: 'D' CHR\$(41)

DEFB %00000000
DEFB %01111100
DEFB %01000010
DEFB %01000001
DEFB %01000001
DEFB %01000010
DEFB %01111100
DEFB %00000000

; \$2A - Character: 'E' CHR\$(42)

DEFB %00000000
DEFB %01111111
DEFB %01000000
DEFB %01111100
DEFB %01000000
DEFB %01000000
DEFB %01111111
DEFB %00000000

; \$2B - Character: 'F' CHR\$(43)

DEFB %00000000
DEFB %01111111
DEFB %01000000
DEFB %01111100
DEFB %01000000
DEFB %01000000
DEFB %01000000
DEFB %00000000

; \$2C - Character: 'G' CHR\$(44)

DEFB %00000000
DEFB %00011110
DEFB %00100001
DEFB %01000000

DEFB %01000111
DEFB %00100001
DEFB %00011110
DEFB %00000000

; \$2D - Character: 'H' CHR\$(45)

DEFB %00000000
DEFB %01000001
DEFB %01000001
DEFB %01111111
DEFB %01000001
DEFB %01000001
DEFB %01000001
DEFB %00000000

; \$2E - Character: 'I' CHR\$(46)

DEFB %00000000
DEFB %00111110
DEFB %00001000
DEFB %00001000
DEFB %00001000
DEFB %00001000
DEFB %00111110
DEFB %00000000

; \$2F - Character: 'J' CHR\$(47)

DEFB %00000000
DEFB %00000010
DEFB %00000010
DEFB %00000010
DEFB %01000010
DEFB %00100010
DEFB %00011100
DEFB %00000000

; \$30 - Character: 'K' CHR\$(48)

DEFB %00000000
DEFB %01000010
DEFB %01000100
DEFB %01111000
DEFB %01000100
DEFB %01000010
DEFB %01000001
DEFB %00000000

; \$31 - Character: 'L' CHR\$(49)

DEFB %00000000
DEFB %01000000
DEFB %01000000
DEFB %01000000
DEFB %01000000
DEFB %01000000
DEFB %01111111
DEFB %00000000

; \$32 - Character: 'M' CHR\$(50)

DEFB %00000000
DEFB %01000001
DEFB %01100011
DEFB %01010101
DEFB %01001001
DEFB %01000001
DEFB %01000001
DEFB %00000000

; \$33 - Character: 'N' CHR\$(51)

DEFB %00000000
DEFB %01100001
DEFB %01010001
DEFB %01001001
DEFB %01000101
DEFB %01000011
DEFB %01000001
DEFB %00000000

; \$34 - Character: 'O' CHR\$(52)

DEFB %00000000
DEFB %00111110
DEFB %01000001
DEFB %01000001
DEFB %01000001
DEFB %01000001
DEFB %00111110
DEFB %00000000

; \$35 - Character: 'P' CHR\$(53)

DEFB %00000000
DEFB %01111110
DEFB %01000001
DEFB %01000001
DEFB %01111110
DEFB %01000000
DEFB %01000000
DEFB %00000000

; \$36 - Character: 'Q' CHR\$(54)

DEFB %00000000
DEFB %00111110
DEFB %01000001
DEFB %01000001
DEFB %01001001
DEFB %01000101
DEFB %00111110
DEFB %00000000

; \$37 - Character: 'R' CHR\$(55)

DEFB %00000000
DEFB %01111110
DEFB %01000001
DEFB %01000001
DEFB %01111110
DEFB %01000100
DEFB %01000010

DEFB %00000000

; \$38 - Character: 'S' CHR\$(56)

DEFB %00000000

DEFB %00111110

DEFB %01000000

DEFB %00111110

DEFB %00000001

DEFB %01000001

DEFB %00111110

DEFB %00000000

; \$39 - Character: 'T' CHR\$(57)

DEFB %00000000

DEFB %01111111

DEFB %00001000

DEFB %00001000

DEFB %00001000

DEFB %00001000

DEFB %00001000

DEFB %00000000

; \$3A - Character: 'U' CHR\$(58)

DEFB %00000000

DEFB %01000001

DEFB %01000001

DEFB %01000001

DEFB %01000001

DEFB %01000001

DEFB %00111110

DEFB %00000000

; \$3B - Character: 'V' CHR\$(59)

DEFB %00000000

DEFB %01000001

DEFB %01000001

DEFB %01000001

DEFB %00100010

DEFB %00010100

DEFB %00001000

DEFB %00000000

; \$3C - Character: 'W' CHR\$(60)

DEFB %00000000

DEFB %01000001

DEFB %01000001

DEFB %01000001

DEFB %01001001

DEFB %01010101

DEFB %00100010

DEFB %00000000

; \$3D - Character: 'X' CHR\$(61)

DEFB %00000000

DEFB %00100001

DEFB %00010010

```

        DEFB  %00001100
        DEFB  %00001100
        DEFB  %00010010
        DEFB  %00100001
        DEFB  %00000000

; $3E - Character: 'Y'      CHR$(62)

        DEFB  %00000000
        DEFB  %01000001
        DEFB  %00100010
        DEFB  %00011100
        DEFB  %00001000
        DEFB  %00001000
        DEFB  %00001000
        DEFB  %00000000

; $3F - Character: 'Z'      CHR$(63)

        DEFB  %00000000
        DEFB  %01111111
        DEFB  %00000010
        DEFB  %00000100
        DEFB  %00001000
        DEFB  %00010000
        DEFB  %01111111
L0FFF: DEFB  %00000000

.END                      ;TASM assembler directive.

;-----
;
;-----
; The 'Character set'
;-----
;
;
; $00 $01 $02 $03 $04 $05 $06 $07 $08 $09 $0A $0B $0C $0D $0E $0F
; nul gra gra gra gra gra gra gra gra gra £ $ : ?
;
;
; $10 $11 $12 $13 $14 $15 $16 $17 $18 $19 $1A $1B $1C $1D $1E $1F
; ( ) - + * / = > < ; , . 0 1 2 3
;
;
; $20 $21 $22 $23 $24 $25 $26 $27 $28 $29 $2A $2B $2C $2D $2E $2F
; 4 5 6 7 8 9 A B C D E F G H I J
;
;
; $30 $31 $32 $33 $34 $35 $36 $37 $38 $39 $3A $3B $3C $3D $3E $3F
; K L M N O P Q R S T U V W X Y Z
;
;-----
;
;-----
; THE 'ZX80 KEYBOARD'
;-----
;
;      [] mosaic graphic      £ currency symbol
;
;
; NOT AND THEN TO <= V ^ => HOME RUBOUT
;+---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
;|  ||  ||  ||  ||  ||  ||  ||  ||  ||  ||  |
;| 1 ||2 ||3 ||4 ||5 ||6 ||7 ||8 ||9 ||0 |
;+---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
;
;
; NEW LOAD SAVE RUN CONT REM IF INPUT PRINT

```

```
;-----+-----+-----+-----+-----+-----+-----+-----+-----+
;| || | || | || | || | || "|| $|| (|| )|| *|
;|Q ||W ||E ||R ||T ||Y ||U ||I ||O ||P |
;-----+-----+-----+-----+-----+-----+-----+-----+-----+
;
; LIST STOP DIM FOR GOTO POKE RAND LET EDIT
;-----+-----+-----+-----+-----+-----+-----+-----+-----+
;| || | || | || | || | || **|| -|| +|| =||NEW|
;|A ||S ||D ||F ||G ||H ||J ||K ||L ||LINE|
;-----+-----+-----+-----+-----+-----+-----+-----+-----+
;
; CLEAR CLS GOSUB RET NEXT BREAK
;-----+-----+-----+-----+-----+-----+-----+-----+-----+
;| || :|| ;|| ?|| /|| OR|| <|| >|| ,|| £ |
;|SHIFT||Z ||X ||C ||V ||B ||N ||M ||. ||SPACE|
;-----+-----+-----+-----+-----+-----+-----+-----+-----+
;
;
;
;-----
;
; THE 'SYSTEM VARIABLES'
;-----
; Note. the names of the System Variables are taken from the original
; Nine Tiles Assembly Listing.
;
; 1 16384 $4000 IY+$00 ERR_NR One less than report code.
; X1 16385 $4001 IY+$01 FLAGS Various Flags to control BASIC System.
; 7 1-Syntax off 0-Syntax on
; 6 1-Numeric result 0-String result
; 5 1-Evaluating function (not used)
; 3 1-K cursor 0-L cursor
; 2 1-K mode 0-L mode.
; 0 1-No leading space 0-Leading space.
; 2 16386 $4002 IY+$02 PPC Line number of current line.
; N2 16388 $4004 IY+$04 P_PTR. Position in RAM of [K] or [L] cursor.
; 2 16390 $4006 IY+$06 E_PPC Number of current line with [>] cursor.
; X2 16392 $4008 IY+$08 VARS Address of start of variables area.
; X2 16394 $400A IY+$0A E_LINE Address of start of Edit Line.
; X2 16396 $400C IY+$0C D_FILE Start of Display File.
; X2 16398 $400E IY+$0E DF_EA Address of the start of lower screen.
; X2 16400 $4010 IY+$10 DF_END Display File End.
;
; X1 16402 $4012 IY+$12 DF_SZ Number of lines in lower screen.
; 2 16403 $4013 IY+$13 S_TOP. The number of first line on screen.
; 2 16405 $4015 IY+$15 X_PTR Address of the character preceding
; the [S] marker.
; 2 16407 $4017 IY+$17 OLDPPC Line number to which continue jumps.
; N1 16409 $4019 IY+$19 FLAGX. More flags.
; 7 1-K mode 0-L mode.
; 6 1-Numeric result 0-String result
; 5 1-Inputting 0-Editing
; N2 16410 $401A IY+$1A T_ADDR Address of next item in syntax table.
; U2 16412 $401C IY+$1C SEED The seed for the random number.
; U2 16414 $401E IY+$1E FRAMES Count of frames shown since start-up.
; N2 16416 $4020 IY+$20 DEST Address of variable in statement.
; N2 16418 $4022 IY+$22 RESULT. Value of the last expression.
; X1 16420 $4024 IY+$24 S_POSN_X Column number for print position.
; X1 16421 $4025 IY+$25 S_POSN_Y Line number for print position.
; X2 16422 $4026 IY+$26 CH_ADD. Address of next character to be
; interpreted.
```

