ZX Forum #04

November 19, 1997

New 40 best procedures - scrolling the screen, merging two pictures, inverting the screen, rotating characters, replacing attributes, filling a closed path, calculating addresses on the screen, copying part of the screen, etc.

(c) Kolotov Sergey, Shadrinsk, SerzhSoft, July, 1997.

NEW "40 BEST Routines"

In 1992, the pages of ZX-REVIEW published an abbreviated translation of the book "40 Best Routines" by J. Hardman and E. Hugheson. The publication received enthusiastic responses from readers, as many aspiring Sinclair users were finally able to overcome the difficulties of machine code with its help. But now, exactly five years later, long-time readers of ZX-REVIEW are surprised to see the familiar title in the first issue of 1997.

All the numerous arguments presented by INFOCOM are certainly convincing in many ways. But in my (and not only my) opinion, they do not give the publishers of the popular magazine the right to waste precious pages of ZX-REVIEW on printing material that has already been published.

Instead of the "RETRO" section, it would be more useful and relevant to introduce a "REMAKE" section. And in it, not just old material should be printed, but it should be examined more deeply, providing examples of more efficient program implementation and individual procedures.

Take, for example, the same "40 procedures". Yes, this work is very useful for beginners. But more experienced assembly language experts will immediately notice that the larger the volume of the procedure provided by the authors, the more flaws it inevitably has, the less efficiently it is implemented, and the more "extra" commands... Yes, some of the "best procedures" are reduced by more than half! And also: the program comments are located after the listings themselves, which makes it very difficult to understand "what exactly happens when executing a procedure at a certain stage". Of course, a description is given like “register C is copied to register B,” but the meaning behind this is not always clear...

Next, a library will be presented, in which graphic procedures are integrated, rewritten according to the principle of the "40 best". The memory occupied by the library has been reduced from 1444 bytes to 861 bytes! Each procedure is thoroughly commented in its own listing, so to speak, "without leaving the cash register".

see Listing and hex dump.

Many procedures require certain pre-defined values - constants - to work. A special memory area addressed by the label CONSTS is allocated for these values. In this case, CONSTS points to address 23296, but of course, this address can be changed to any other. The length of the constant area is 8 bytes. When executing any of the procedures, none of the constants are changed. Otherwise, they would have to be called variables...

In procedures that manipulate the coordinates of points on the screen, the counting is done differently from BASIC - from top to bottom, not from bottom to top. This counting of coordinates is much more convenient and is used in many other computers. Now you can specify the Y coordinate from 0 to 191 (instead of 175), i.e. it becomes possible to specify the coordinates of those points on the

screen that are in the two lower rows reserved for error messages. When counting from bottom to top, the maximum Y-coordinate is Y=175, and it is impossible to reach the bottom two rows at all.

BRIEF DESCRIPTION OF PROCEDURES

1. ASRL_LF -
The procedure shifts the entire screen of attributes (colors) to the left. The right column is filled with the attribute from the cell at CONSTS (23296). The length of the procedure is 22 bytes (previously 23 bytes). Translated to address 62000 (HEX: #F230).

2. ASRL_RG -
Shifts the entire attribute screen to the right. The left column is filled with the attribute from CONSTS. The length of the procedure is 21 bytes (previously 19 bytes). Located at address 62022 (#F246).

3. ASRL_UP -
Shifts the entire attribute screen upwards. The bottom row of attributes is filled from CONSTS. The length of the procedure is 19 bytes (previously 21 bytes). Address: 62043 (#F25B).

4. ASRL_DN -
moving the entire attribute screen down. The top line is filled with the attribute from CONSTS. Length: 20 bytes (21 bytes). Address: 62062 (#F26E).

5. SSRL_LF
Shifts the screen left one character (graphics). The right column of character cells is cleared. Length 20 bytes (21 bytes). Address: 62082 (#F282).

6. SSRL_RG
Shifts the screen to the right by one character. The left column of character cells is cleared. Length: 19 bytes (previously 22 bytes). Address: 62102 (#F296).

7. SSRL_UP -
Shifts the entire screen upwards by one character. The bottom row of character cells is cleared. Length: 55 bytes (previously 68 bytes). Address: 62121 (#F2A9).

8. SSRL_DN -
Shifts the entire screen downwards by one character. The top row of character cells is cleared. Length: 55 bytes (previously 73 bytes). Address: 62176 (#F2E0).

9. PSRL_LF -
Shifts the entire screen to the left by one pixel (graphics). The right column of pixels is cleared. Length: 16 bytes (previously 17 bytes). Address: 62231 (#F317).

10. PSRL_RG -
shift the entire screen to the right by one pixel. The left column of pixels is cleared. Length 17 bytes (17 bytes). Address: 62247 (#F327).

11. PSRL_UP -
shift the entire screen up by one line of pixels. The bottom row of pixels is cleared. Length 38 bytes (91 bytes). Address: 62264 (#F338).

12. PSRL_DN -
shift the entire screen down by one line of pixels. The top row of pixels is cleared. Length 38 bytes (90 bytes). Address: 62302 (#F35E).

13. SCR_MRG -
merge two images (graphics) using the OR principle. A two-byte constant at address CONSTS should contain the address of the second image in memory (overlay). The result is placed on the screen. Length of the procedure 17 bytes (previously - 21 bytes). Placement address: 62340 (#F384).

14. SCR_INV -
invert the screen (graphics) using the NOT principle. All pixels change their value to the opposite. Length of the procedure 12 bytes (previously - 18 bytes). Address: 62357 (#F395).

15. SINV_UD -
invert the character vertically. An arrow pointing upwards becomes an arrow pointing downwards, and vice versa. The address of the character to be modified should be contained in a two-byte variable at address CONSTS. Length of the procedure 20 bytes (remains the same). Placement address: 62369 (#F3A1).

16. SINV_LR -
invert the character horizontally. An arrow pointing left becomes an arrow pointing right, and vice versa. The address of the character to be modified should be contained in a two-byte variable at address CONSTS. Length of the procedure 17 bytes (previously - 19 bytes). Placement address: 62389 (#F3B5).

17. SROTATE -
rotation of a symbol clockwise by 90 degrees. The address of the symbol to be rotated should be stored in a two-byte variable at address CONSTS. Length: 26 bytes (previously 42 bytes). Address: 62406 (#F3C6).

18. ACHANGE -
changing the attribute values of all screen symbols. Bitwise operation. The mask of bits should be stored in the cell at address CONSTS: the bits that are set in the mask remain unchanged in the attributes, while the bits that have a zero value in the mask will have a zero value in the attributes (AND operation). The byte at address CONSTS+1 should contain the bits to be included in all screen attributes, i.e. if any bit in this byte is set, it will be set in all attributes (OR operation). Length: 16 bytes (previously 21 bytes). Address: 62432 (#F3E0).

19. AREPLC -
searching for attributes with a specific value and replacing each found attribute with a new value. The value to be replaced (what to search for) should be stored in the cell at address CONSTS. The value to replace it with should be stored in the cell at address CONSTS+1. Length: 18 bytes (previously 22 bytes). Address: 62448 (#F3F0).

20. PAINT -
filling a specific area of the screen bounded by a line of pixels. The starting point is specified by placing its X coordinate at address CONSTS, and its Y coordinate at address CONSTS+1. If the Y coordinate is greater than 191 or the point at the specified coordinates is already set, the program will terminate. This procedure is not relocatable due to calls to the POINT procedure. The stack is actively used during the filling process - it stores the coordinates of the filling lines. When filling a large area with a complex shape, more free space is required in the RAM - between the end of the BASIC program and the address set by the CLEAR statement (the content of the RAMTOP system variable). If there is not enough memory space, a failure may occur. The procedure occupies 88 bytes, and together with the POINT procedure - 123 bytes, which is more than twice as short as the 1992 procedure (263 bytes!). Address: 62466 (#F402).

21. POINT
drawing points on the screen at specified coordinates and checking the state of these points (ON/OFF). Attention! This procedure can only be used from machine codes (calling it from BASIC will not work). Before calling the procedure, the X coordinate (0..255) must be set in register E, and the Y coordinate (0..191) of the point to be checked must be set in register D. The procedure will set the HL register pair to the address of the byte on the screen where the point is located, and the C register to the mask of the point in this byte (one bit set to one). Depending on whether the point is enabled or not, the zero flag is set: Z - point is not enabled, NZ - point is enabled. If the point is enabled (visible), the A register (accumulator) will have the same value as the C register, and if the point is not enabled, the A register will be cleared. The B register is always set to zero when exiting the procedure. Length: 35 bytes (would have taken about 70 bytes in the original). Address: 62554 (#F45A).

22. PFIGURE -
drawing any previously defined figure (template) on the screen. The coordinates of the starting point are set similarly to the PAINT procedure. The template is defined in the BASIC string variable A$ (it can be changed to any other by slightly adjusting the assembly listing or dump). The string of characters has the following format (slightly different from the original):

"5" - decrease the X-coordinate

"6" - increase the Y-coordinate

(counting from top to bottom)

"7" - decrease the Y-coordinate

"8" - increase the X-coordinate

"0" - place a point.

Any other characters are ignored. If the string variable does not exist or does not contain any information, the program terminates. There is no control for the exit of the initial Y-coordinate, as part of the figure may still be visible.

Therefore, the check for going beyond the screen boundaries is introduced in the template formation loop. The 'wrap-round' capability is preserved, i.e. when the X-coordinate goes beyond the left part of the screen, the template appears on the right, and vice versa.

The procedure is not relocatable. Length of PFIGURE: 98 bytes, and together with the used subroutine POINT - 133 bytes, which is still much smaller than the original (196 bytes). Address: 62589 (#F47D). If the call to the POINT procedure is "expanded", PFIGURE will become relocatable and will occupy approximately 125 bytes!23. PSCALER - copying a part of the screen to another area of the same screen with possible enlargement of the copy along the X and/or Y axes. The following constants are used:

Address Name Comment

CONSTS X1_OLD one of the two initial X-coordinates of the rectangle

CONSTS+1 Y1_OLD one of the two initial Y-coordinates of the rectangle

CONSTS+2 X2_OLD one of the two initial X-coordinates of the rectangle

CONSTS+3 Y2_OLD one of the two initial Y-coordinates of the rectangle

CONSTS+4 XSCALE scaling factor for X

CONSTS+5 YSCALE scaling factor for Y

CONSTS+6 X_NEW coordinates of the top left corner of the screen area,

CONSTS+7 Y_NEW where the copying is performed.

The coordinates of the initial rectangle for copying are specified in the constants X1_OLD, Y1_OLD, X2_OLD, Y2_OLD, and can be arranged in any order. The procedure itself will determine the smallest and largest coordinates.

Emergency exit from the procedure occurs in the following cases:
1. XSCALE=0 - the scale of enlargement along the X axis is zero
2. YSCALE=0 - the scale of enlargement along the Y axis is zero
3. Y_NEW>191 - the new Y coordinate goes beyond the screen boundaries
4. Y1_OLD>191 - the old Y1 coordinate goes beyond the screen boundaries
5. Y2_OLD>191 - the old Y2 coordinate goes beyond the screen boundaries.

As in the original, the program does not have a control that would check the possibility of placing a new image on the screen. If this is not possible, a failure may occur.

During the execution of the procedure, it first pushes the bit image of the copied screen rectangle onto the stack and only then redraws it in a new place, enlarging it if necessary. Therefore, if there is not enough space on the stack, a hang, reset, or error may occur, just like in the PAINT procedure.
If you want the copied part of the screen to have the same size as the given one, you need to set the scale to 1:1 - assign the value of one to the constants XSCALE and YSCALE. For double size, there should be twos, and so on.

The procedure is not movable due to the use of the POINT subroutine. PSCALER takes up 174 bytes, and together with POINT - 209 bytes. In any case, this is much less than the original - 335 bytes! The placement address is 62687 (#F4DF).

So, you have familiarized yourself with the new implementation of the "best procedures". But I advise you not to be deceived - some procedures can certainly be further shortened... However, in doing so, something will inevitably have to be sacrificed: execution speed, movability, or simply time spent. I hope that the programs presented in this article will be useful to you, at worst, they may just give you some ideas... Beginners can try comparing the procedures of 1992 with the new ones, study the principles of creating more efficient programs, techniques for integrating multiple different procedures into one large library... Experienced programmers, perhaps, will get a lot of pleasure, slyly laughing at this work. But that's also a plus: to entertain people is a very necessary thing! So I wish all readers ENJOYABLE WORK WITH YOUR FAVORITE SPECCY!

```
; ------------------------------------------------------------;
; 40 New Best Routines (graphic)                              ;
; (c) SerzhSoft, Shadrinsk, may-june, 1997                    ;
; old length: 1444 bytes        new length:  861 bytes        ;
; ------------------------------------------------------------;
        ORG 62000                   ; ;assembly address
        CONSTS:  defs 8             ; ;constant buffer address (8 bytes)
        VAR:                        ; VAR 23296
        23296:

; ;-----------------------------------------
; Shift attributes left (22<=23)
; ;-----------------------------------------
ASRL_LF:
            LD DE, #5800        ; DE=address of the first attribute byte
LP_ASLF:    LD H, D             ; copied DE to HL
            LD L,E              ; and increased HL by one:
            INC HL              ; HL=address of the second attribute byte
            LD BC,#001F         ; <attribute line length> - 1
            LDIR                ; shift the attribute line to the left
            LD    A,(CONSTS)    ; fill color after shift
            LD    (DE),A        ; set a new attribute
            INC    DE           ; move to the next line from below
            LD    A,D           ; if the attributes have already run out,
            CP    #5B           ; if we come across the printer buffer,
            JR C, LP_ASLF       ; then STOP, otherwise shift further
            RET                 ; exit the procedure

; ;-----------------------------------------
; ;Shift attributes to the right (21<=23)
; ;-----------------------------------------
ASRL_RG:
            LD    DE,#5AFF      ; address of the last attribute byte
LP_ASRG:    LD    H,D           ;copied DE to HL -
            LD    L,E           ; last byte of the attribute line
            DEC    HL           ; last byte of the attribute line
            LD    BC,#001F      ; <attribute line length> - 1
            LDDR                ; shift the attribute line to the right
            LD    A,(CONSTS)    ; fill color after shift
            LD    (DE),A        ; set a new attribute
            DEC    DE           ; go to next line from top
            BIT    3,D          ; if we are still in attributes,
            JR NZ, LP_ASRG      ; then we repeat the cycle for the next one. lines
            RET                 ; exit procedure
```

```
; ;-----------------------------------------
; Shift attributes up (19<=21)
; ;-----------------------------------------
ASRL_UP:
                LD HL,#5820          ; address of the second attribute line
                LD     DE,#5800      ; address of the first attribute line
                LD BC,#02E0          ; move: 23 lines of 32 bytes
                LDIR                 ; move the bottom 23 lines up
                LD     A,(CONSTS)    ; color to fill the bottom line
LP_ASUP:        LD     (DE),A        ; set a new attribute
                INC     E            ; if you filled in the entire last line
                JR     NZ,LP_ASUP    ; (E=0), then we interrupt the cycle
                RET                  ; exit procedure

; ;-----------------------------------------
; ;Shift attributes down (20<=21)
; ;-----------------------------------------
ASRL_DN:
                LD     HL,#5ADF      ; address of the end of the second line from the bottom
                LD     DE,#5AFF      ; address of the end of the lowest line
                LD BC,#02E0          ; move: 23 lines of 32 bytes
                LDDR                 ; move the top 23 lines down
                LD     A,(CONSTS)    ; color to fill the top line
LP_ASDN:        LD     (DE),A        ; set a new attribute
                DEC     E            ; if we reached the very first byte
                JR NZ,LP_ASDN        ; attribute area (E=0), then STOP
                LD     (DE),A        ; and set this byte
                RET                  ; exit procedure

; ;-----------------------------------------
; ;Shift left one character (20<=21)
; ;-----------------------------------------
SSRL_LF:
                LD DE,#4000          ; start of graphics area
LP_SSLF: LD H,D                      ; address of the first
                LD     L,E           ; byte line
                INC HL               ; address of the second byte of the line
                LD     BC,#001F      ; how many bytes to shift
                LDIR                 ; shift line left by 1 byte
                XOR     A            ; clear the flags
                LD     (DE),A        ; to the last (right) byte of the line
                INC     DE           ; go to next line (bottom)
                LD     A,D           ; if attributes
                CP     #58           ; "not seen yet"
                JR     C,LP_SSLF     ; then we repeat the cycle for the next one. lines
                RET                  ; exit procedure
```

```
; ;-----------------------------------------
; ;Shift right one character (19<=22)
; ;-----------------------------------------
SSRL_RG:
                LD DE,#57FF          ; ;last byte of the graphics area
LP_SSRG:        LD      H,D          ; last byte address
                LD      L,E          ; current line
                DEC     HL           ; address of the next to the last byte
                LD      BC,#001F     ; shift: 31 bytes
                LDDR                 ; shift graphics line to the right
                XOR     A            ; clear the flags
                LD      (DE),A       ; first (left) byte of the current line
                DEC     DE           ; go to next line above
                BIT     6,D          ; if we have not yet “came across” the ROM,
                JR      NZ,LP_SSRG   ; then we continue to spin the cycle
                RET                  ; exit procedure

; ;-----------------------------------------
; ;Shift up one character (55<=68)
; ;-----------------------------------------
SSRL_UP:
                LD      DE,#4000     ; beginning of screen area
LP_SSU1:        PUSH    DE           ; save the line address on the stack
                LD      BC,#0020     ; in line - 32 bytes
                LD      A,E          ; The DE register contains the address
                ADD     A,C          ; top line. In the register
                LD      L,A          ; HL needs to get the address
                LD      A,D          ; lines, lying below with a step of 8.
                JR      NC,GO_SSUP   ; To do this, add to the register E
                ADD     A,#08        ; We add 32 and enter it in L. If if there is
GO_SSUP:        LD      H,A          ; overflow, then H=D+8
                LDIR                 ; carry one line (32 bytes)
                POP     DE           ; restore the address of the beginning of the line
                LD      A,H          ; check if it's time for us to scroll
                CP      #58          ; (moved all 23 rows)
                JR      NC,LP_SSU2   ; if yes, go to clear
                INC     D            ; ----------------------------------;
                LD      A,D          ; DOWN_DE
                AND     #07          ; standard sequence
                JR      NZ,LP_SSU1   ; of commands to move to the next line
                LD      A,E          ; in the screen area
                ADD     A,#20        ; (for register DE)
                LD      E,A
                JR      C,LP_SSU1    ; input: DE - line address
                LD      A,D          ; output: DE - line address below
                SUB     #08          ; accumulator is used
                LD      D,A
```

```
                JR      LP_SSU1       ; ---------------------------------;
LP_SSU2:        XOR     A             ; clearing the accumulator
LP_SSU3:        LD      (DE),A        ; and using it -
                INC     E             ; clearing one line of the image
                JR      NZ,LP_SSU3    ; total: 32 bytes
                LD      E,#E0         ; jump to the next
                INC     D             ; (lower) line of the image
                BIT     3,D           ; filled the entire last row?
                JR      Z,LP_SSU2     ; if not, continue filling
                RET                   ; exit the procedure

; ;-----------------------------------------
; ;Shift down one character (55<=73)
; ;-----------------------------------------
SSRL_DN:
                LD DE,#57FF     ; address of last graphics byte
LP_SSD1:        PUSH    DE            ; stored line end address
                LD      BC,#0020      ; length of one image line
                LD      A,E           ; in register HL
                SUB     C             ; we get the address
                LD      L,A           ; end of the line
                LD      A,D           ; lying above
                JR      NC,GO_SSDN    ; initial step
                SUB     #08           ; in 8 pixels (lines):
GO_SSDN:        LD      H,A           ; HL=copy from; DE=where
                LDDR                  ; transfer one line of graphics
                POP     DE            ; restore the end-of-line address
                BIT     6,H           ; if we are no longer on the screen,
                JR      Z,LP_SSD2     ; then we move on to cleaning
                LD      A,D           ; ---------------------------------;
                DEC     D             ; UP_DE
                AND     #07           ; standard sequence
                JR      NZ,LP_SSD1    ; commands to go to the line
                LD      A,E           ; up in the screen area
                SUB     #20           ; (for DE register)
                LD      E,A
                JR      C,LP_SSD1     ; at the input: DE - line address
                LD      A,D           ; output: DE - line address above
                ADD     A,#08         ; accumulator is used
                LD      D,A
                JR      LP_SSD1       ; ---------------------------------;
LP_SSD2:        XOR     A             ; clear accumulator
LP_SSD3:        LD      (DE),A        ; clear one line of the image:
                DEC     E             ; decrement E
                JR NZ,LP_SSD3         ; jump to LP_SSD3 if not zero (31 bytes)
                LD      (DE),A        ; clear the first byte of the line
                LD      E,#1F         ; move to the next (upper) line
```

```
                DEC     D           ; decrement D
                BIT     6,D         ; have we reached the ROM?
                JR      NZ,LP_SSD2  ; if not, continue clearing
                RET                 ; return from the procedure

; ;-----------------------------------------
; ;Shift left by one pixel (16<=17)
; ;-----------------------------------------
PSRL_LF:
                LD      HL,#57FF    ; address of the last byte of the graphics
LP_PSL1:        OR      A           ; reset the carry flag CF
                LD      B,#20       ; 32 bytes in one line
LP_PSL2:        RL      (HL)        ; CF<-[shifted byte]<-CF (to the left)
                DEC     HL          ; move to the previous byte of the line
                DJNZ    LP_PSL2     ; shift loop for one line
                BIT     6,H         ; are we still on the screen?
                JR      NZ,LP_PSL1  ; if yes, shift the next line
                RET                 ; exit the procedure

; ;-----------------------------------------
; ;Shift right by one pixel (17)
; ;-----------------------------------------
PSRL_RG:
                LD      HL,#4000    ; address of the first byte of the graphics
                LD      C,#C0       ; shift - 192 lines
LP_PSR1:        OR      A           ; CF=0 for an empty column on the left
                LD      B,#20       ; number of bytes in one line
LP_PSR2:        RR      (HL)        ; shift one byte to the right
                INC     HL          ; next byte of the image line
                DJNZ    LP_PSR2     ; shift the entire line - 32 bytes
                DEC     C           ; decrease the line counter
                JR      NZ,LP_PSR1  ; if all lines have been shifted, then STOP
                RET                 ; exit the procedure
```

```
; ;-----------------------------------------
:                ; ;Shift up one pixel (38<=91)
; ;-----------------------------------------
PSRL_UP:
                LD      DE,#4000      ; address of the first byte of the graphics
                LD      H,D           ; copied start address
                LD      L,E           ; of the graphics line to HL
                LD      BC,#0020      ; single line size
                INC     H             ; ---------------------------------;
                LD      A,H           ; DOWN_HL
                AND     #07           ; standard sequence
                JR      NZ,GO_PSUP    ; of commands to jump to the line
                LD      A,L           ; below in the screen area
                ADD     A,C           ; (for register HL)
                LD      L,A           ; (here ADD A,C instead of ADD A,#08)
                JR      C,GO_PSUP     ; input: HL - line address
                LD      A,H           ; output: HL - line address below
                SUB     #08           ; accumulator is used
                LD      H,A           ; ---------------------------------;
GO_PSUP:        PUSH    HL            ; save the bottom line address
                LDIR                  ; image transfer from bottom to top
                POP     DE            ; DE - bottom line address
                LD      A,H           ; we are still in the graphics area
                CP      #58           ; or have we encountered attributes?
                JR      C,LP_PSU1     ; if still graphics, repeat
                XOR     A             ; clear the accumulator and use it to
LP_PSU2:        LD      (DE),A        ; clear the
                INC     E             ; bottom line of the image
                JR      NZ,LP_PSU2    ; after moving the screen up
                RET                   ; exit the procedure
```

```
; ;-----------------------------------------
; ;Shift down one pixel (38<=90)
; ;-----------------------------------------
PSRL_DN:
                LD DE,#57FF      ; address of the last byte of the graphics
LP_PSD1:        LD H,D           ; copied the address of the last
                LD L,E           ; byte of the line into HL
                LD      BC,#0020      ; width of one image line
                LD      A,H           ; ---------------------------------;
                DEC     H             ; UP_HL
                AND     #07           ; standard sequence
                JR      NZ,GO_PSDN    ; commands to go to the line
                LD      A,L           ; up in the screen area
                SUB     C             ; (for HL register)
                LD      L,A           ; (here SUB C instead of SUB #08)
                JR      C,GO_PSDN     ; at the input: HL - line address
                LD      A,H           ; output: HL - line address above
                ADD     A,#08         ; accumulator is used
                LD      H,A           ; ---------------------------------;
GO_PSDN:        PUSH    HL            ; save the top line address
                LDDR                  ; copy 1 line from top to bottom
                POP     DE            ; top line address has become current
                BIT     6,H           ; until we reach the ROM
                JR      NZ,LP_PSD1    ; continue the loop through the lines
                XOR     A             ; clear the accumulator
LP_PSD2:        LD      (DE),A        ; clear the top line
                DEC     E             ; of the image after shifting
                JR      NZ,LP_PSD2    ; the entire screen down
                LD      (DE),A        ; clear the very first byte
                RET                   ; exit the procedure
```

```
; ;-----------------------------------------
; ;Merge images (17<=21)
; ;-----------------------------------------
SCR_MRG:
                LD HL,(CONSTS)   ; took the image address from the cell
                LD DE,#4000      ; screen area address
LP_SCRM:    LD A,(DE)        ; screen image byte
                OR (HL)          ; "merged" with the picture byte in memory
                LD (DE),A        ; and placed back on the screen
                INC HL           ; next byte of the image in memory
                INC DE           ; next byte of screen area
                LD A,D           ; end check
                CP #58           ; screen area
                JR C,LP_SCRM     ; if not finished, then repeat
                RET              ; exit the procedure

; ;-----------------------------------------
; ;Invert screen (12<=18)
; ;-----------------------------------------
SCR_INV:
                LD     HL,#57FF      ; last byte of the screen area
LP_SCRI:    LD     A,(HL)        ; take image byte from screen
                CPL              ; invert it
                LD     (HL),A        ; and put it back
                DEC    HL            ; moving to the beginning of the area
                BIT    6,H           ; if "crosses" the beginning,,
                JR     NZ,LP_SCRI    ; then STOP, otherwise loop
                RET              ; exit the procedure

; ;-----------------------------------------
; ;Invert character vertically (20)
; ;-----------------------------------------
SINV_UD:
                LD     HL,(CONSTS)   ; take the address from the variable
                LD     D,H           ; save this
                LD     E,L           ; address in DE
                LD     B,#08         ; in a symbol - 8 bytes
LP_SIU1:    LD     A,(HL)        ; take one byte of the character
                PUSH   AF            ; and push it onto the stack
                INC    HL            ; move to next character byte
                DJNZ   LP_SIU1       ; repeat the cycle for eight bytes
                LD     B,#08         ; how many bytes we will read
LP_SIU2:    POP    AF            ; pop a byte from the stack and back
                LD     (DE),A        ; in nom order we write into the symbol
                INC    DE            ; next character byte
                DJNZ   LP_SIU2       ; rotate the loop eight times
                RET              ; exit the procedure
```

```
; ;-----------------------------------------
; ;Invert character horizontally (17<=19)
; ;-----------------------------------------
SINV_LR:
                LD      HL,(CONSTS)   ; take the address from the variable
                LD      B,#08         ; modify: 8 bytes
LP_SIL1:        LD      A,#01         ; set zero bit A to 1
LP_SIL2:        RR      (HL)          ; rotate the symbol byte to the right
                RLA                   ; and accumlator - to the left (via CF)
                JR      NC,LP_SIL2    ; until the zero bit is in CF
                LD      (HL),A        ; write the changed byte
                INC     HL            ; next character byte
                DJNZ    LP_SIL1       ; repeat the cycle 8 times
                RET                   ; exit procedure

; ;-----------------------------------------
; ;Rotate the character clockwise (26<=42)
; ;-----------------------------------------
SROTATE:
                LD      HL,(CONSTS)   ; take the address from the variable
                LD      B,#08         ; 8 vertical columns per character
LP_SRO1:        PUSH    HL            ; save address on stack
                LD      A,#80         ; turned on the 7th bit in the accumulator
LP_SRO2:        RR      (HL)          ; rotate the character bytes to the right
                RRA                   ; and one bit from each byte
                INC     HL            ; gradually fill the accumulator
                JR      NC,LP_SRO2    ; so far 7 on. bit won't hit CF
                POP     HL            ; restore the symbol address
                PUSH    AF            ; vertical character column - onto the stack
                DJNZ    LP_SRO1       ; rotate the loop according to the number of columns
                LD      B,#08         ; columns became lines - bytes
LP_SRO3:        POP AF                ; pop a byte from the stack
                LD      (HL),A        ; and this is a new line of the symbol
                INC HL                ; next character byte
                DJNZ    LP_SRO3       ; repeat by number of lines (8 bytes)
                RET                   ; exit procedure
```

```
; ;-----------------------------------------
; ;Attribute change (16<=21)
; ;-----------------------------------------
ACHANGE:
                LD      HL,(CONSTS)   ; L - mask (AND), H - additive (OR)
                LD      DE,#5AFF      ; last byte of the attribute area
LP_ACHN:        LD      A,(DE)        ; take the current attribute value
                AND     L             ; discarded extra bits
                OR      H             ; add necessary bits
                LD      (DE),A        ; and write it back to the original location
                DEC     DE            ; moving to the beginning of the attributes
                BIT     3,D           ; are there more attributes left?
                JR      NZ,LP_ACHN    ; if not, then loop
                RET                   ; exit procedure

; ;-----------------------------------------
; ;Attribute replacement (18<=22)
; ;-----------------------------------------
AREPLC:
                LD      DE,(CONSTS)   ; E - what to look for, D - what to replace
                LD      HL,#5AFF      ; last byte of the attribute area
LP_ARPL:        LD      A,(HL)        ; take the current attribute value
                CP      E             ; is it the one we're looking for?
                JR      NZ,GO_ARPL    ; no, skip the change
                LD      (HL),D        ; yes, change it to the new value
GO_ARPL:        DEC HL                ; move towards the beginning of the attribute area
                BIT     3,H           ; are there more attributes left?
                JR      NZ,LP_ARPL    ; if not, then loop
                RET                   ; exit procedure

; ;-----------------------------------------
; ;Painting the outline (123<=263)
; ; 123=88+35 - together with the POINT procedure
; ;-----------------------------------------
PAINT:          LD      HL,(CONSTS)   ; coordinates of the starting point
                LD      A,H           ; check the Y coordinate for the output
                CP      #C0           ; off screen:
                RET     NC            ; if Y>=192, then exit
                SBC     A,A           ; because CF=1, then SBC A,A gives A=#FF -
                PUSH    AF            ; this will be the end of stack pointer
                PUSH    HL            ; save the coordinates of the first point
LP_PNT1:        POP     DE            ; take X,Y of the next point from the stack
                INC     D             ; if Y=#FF, then the stack is exhausted,
                RET     Z             ; and then we exit the procedure
                DEC     D             ; restore original Y value
                CALL POINT            ; check the point with coordinates (E,D)
                JR      NZ,LP_PNT1    ; if it is enabled, then go to the next
```

```
                EX AF,AF            ; ;A'=0, CF=0 - auxiliary. signs
LP_PNT2:        LD      A,E         ; take X coordinate
                OR      A           ; if it is equal to zero,
                JR      Z,GO_PNT1   ; then jump back
                DEC     E           ; otherwise - decrease the X coordinate
                CALL POINT          ; and check the previous point
                JR      Z,LP_PNT2   ; if "no obstacle", repeat
LP_PNT3:        INC     E           ; move to the point on the right (X=X+1)
                JR      Z,LP_PNT1   ; if X>255, then next. point from stack
GO_PNT1:        CALL    POINT       ; check the next right point
                JR      NZ,LP_PNT1  ; if enabled, then next. from the stack
                LD      A,(HL)      ; if the point is not set,
                OR      C           ; then take a byte from the screen, turn it on
                LD      (HL),A      ; the desired bit and put it back
                LD      A,D         ; check the Y coordinate:
                OR      A           ; if it is zero,
                JR      Z,GO_PNT4   ; then we don't check while lying down. above the line
                DEC     D           ; go to line above (Y=Y-1)
                CALL    POINT       ; checking the overlying point
                JR      Z,GO_PNT2   ; if not enabled, then transition
                EX AF,AF            ; take auxiliary flags
                LD      A,B         ; allowed to save the point in the stack
                JR      GO_PNT3     ; continue
GO_PNT2:        EX AF,AF            ; take auxiliary flags
                INC     A           ; if A>0, then it is prohibited
                DEC     A           ; to save the coordinates of the new
                JR      NZ,GO_PNT3  ; points in the stack -> jump over
                LD      A,C         ; otherwise, we prohibit saving the coordinates
                PUSH    DE          ; but push one onto the stack
GO_PNT3:        EX AF,AF            ; save auxiliary flags
                INC     D           ; return to the bottom line
GO_PNT4:        LD      A,D         ; check the Y coordinate:
                CP      #BF         ; if - last (lower does not happen),
                JR      NC,LP_PNT3  ; then go to next. point to the right
                INC     D           ; otherwise - go down to the line below
                CALL    POINT       ; check the underlying point
                JR      Z,GO_PNT5   ; if not enabled, then transition
                EX AF,AF            ; take auxiliary flags
                AND     A           ; allowed to remember the point on the stack
                JR      GO_PNT6     ; continue
GO_PNT5:        EX AF,AF            ; take auxiliary flags
                JR      C,GO_PNT6   ; if you can't save, then go
                SCF                 ; prohibit storing point on stack
                PUSH    DE          ; but we push one point onto the stack
GO_PNT6:        EX AF,AF            ; saved auxiliary flags
                DEC     D           ; return to the top line
                JR      LP_PNT3     ; go to next point on the right
```

```
; ;-----------------------------------------
; ;Checking the status of the point and calculating the address on the screen (35<=70)
; ;-----------------------------------------
; if the point is turned off, then ZF=1 (Z)
; otherwise ZF=0 (NZ)
POINT:      LD    B,#07       ; frequently used mask (#07)
            LD    A,D         ; take Y-coordinate
            RRA               ; divide it by 8
            SCF               ; and start forming
            RRA               ; high byte
            RRA               ; pixel addresses
            AND   #5F         ; in the screen (register H):
            LD    H,A         ; %010yyyyy
            XOR   E           ; next we form
            AND   B           ; low byte
            XOR   E           ; addresses
            RRCA              ; pixel
            RRCA              ; on the screen
            RRCA              ; (register L):
            LD    L,A         ; %yyyxxxxx
            LD    A,D         ; finish
            XOR   H           ; formation
            AND   B           ; high byte
            XOR   H           ; pixel addresses
            LD    H,A         ; in the screen (register H)

POINTLP:    LD    A,E         ; begin to form
            AND   B           ; pixel mask in byte
            LD    B,A         ; images (corresponding
            LD    A,#80       ; bit is on). Turn on the 7th bit
            JR    Z,GO_PNT    ; if this is just what you need,
LP_PNT:     RRCA              ; then we jump over the shift
            DJNZ  LP_PNT      ; bit on right
GO_PNT:     LD    C,A         ; save the mask on. bit in reg. C
            AND   (HL)        ; check the pixel on the screen
            RET               ; exit procedure
```

```
; ;-----------------------------------------
; Building templates (98<=196)
; 98+35=133 - together with the POINT procedure
; ;-----------------------------------------

PFIGURE:    LD DE,(CONSTS)       ; coordinates of the starting point
            LD HL,(23627)        ; start address of BASIC variables
LP_PFG1:    LD A,(HL)            ; first byte of some variable
            INC HL               ; go to next byte
            LD BC,#0012          ; size of loop variable FOR...NEXT
            CP #E0               ; found the FOR...NEXT loop variable?
            JR NC,GO_PFG2        ; if yes, then go to next. AC
            CP #80               ; run out of BASIC variables?
            RET Z                ; if yes, then exit the procedure
            LD C,#05             ; number length AC with one character
            JR NC,GO_PFG3        ; array or number lane from several Sim.
            CP #60               ; variable number with one character in the name?
            JR NC,GO_PFG2        ; yes, move to next variable
            CP "A"               ; character variable name (A$)
            JR Z,GO_PFG4         ; hurray, we finally found it!!!
GO_PFG1:    LD C,(HL)            ; we get
            INC HL               ; size of the study
            LD B,(HL)            ; variable
            INC HL               ; in bytes and,
GO_PFG2:    ADD HL,BC            ; adding to the address,
            JR LP_PFG1           ; move on to the next variable
GO_PFG3:    BIT 5,A              ; array variable?
            JR Z,GO_PFG1         ; yes, let's jump over it
LP_PFG2:    BIT 7,(HL)           ; check for the end of the name number. lane
            INC HL               ; next name byte
            JR NZ,GO_PFG1        ; name ended, transition
            JR LP_PFG2           ; continue to view the name
GO_PFG4:    LD C,(HL)            ; took the length of the found
            INC HL               ; string variable
            LD B,(HL)            ; with data according to the template
LP_PFG3:    INC HL               ; next pattern data character
            LD A,B               ; checking to see if we have exhausted
            OR C                 ; Are we all data according to a template?
            RET Z                ; if yes (length=0), then exit
            DEC BC               ; reduced length
            LD A,(HL)            ; took the pattern data symbol
            CP "0"               ; and not "should I put an end to it"?
            JR NZ,GO_PFG6        ; if not, go to continuation
            LD A,D               ; y-coordinate of the current point
            CP #C0               ; if extends beyond the bottom edge
            JR NC,LP_PFG3        ; screen, then we don’t depict the dot
            PUSH HL              ; otherwise, save some
```

```
                PUSH BC             ; registers so as not to spoil
                CALL POINT          ; call the point checking procedure
                LD A,(HL)           ; based on calculated values
                OR C                ; draw a dot on the screen
                LD (HL),A           ; using the fact that HL=address,
                POP BC              ; and register C contains the dot mask
                POP HL              ; restore saved registers
                JR LP_PFG3          ; processing next. pattern symbol
GO_PFG6:        SUB "5"             ; move the pen to the left?
                JR NZ,GO_PFG7       ; if not, then leave everything as is
                DEC E               ; otherwise - reduce x-coordinate
GO_PFG7:        DEC A               ; are we moving down?
                JR NZ,GO_PFG8       ; no, transition
                INC D               ; yes, increase the y-coordinate
GO_PFG8:        DEC A               ; direction "up"?
                JR NZ,GO_PFG9       ; no, let's jump over
                DEC D               ; yes, reduce the y-coordinate
GO_PFG9:        DEC A               ; maybe you need to move to the right?
                JR NZ,LP_PFG3       ; no, go to next. symbol template
                INC E               ; yes, increase the x-coordinate
                JR LP_PFG3          ; and move on to the next one. symbol template

; ;-----------------------------------------
; Screen enlargement and copying (174<=335)
; 174+35=209 - together with the POINT procedure
; ;-----------------------------------------
PSCALER:        LD HL,(CONSTS+4)    ; ;scale of magnification in x and y
                INC L               ; x-coordinate check
                DEC  L              ; to zero value
                RET  Z              ; if equal to 0, then error (output)
                INC  H              ; y-coordinate check
                DEC  H              ; to zero value
                RET  Z              ; if equal to 0, then error (output)
                LD   HL,(CONSTS+6) ; new x-,y-coordinates (“to”)
                LD   A,#BF          ; maximum possible y-coordinate
                CP   H              ; checking the new y-coordinate
                RET  C              ; if not on the screen - exit
                LD   HL,(CONSTS)    ; x1-,y1-coordinates ("from")
                CP   H              ; check if y1 is included in the screen
                RET  C              ; if behind the screen, then exit
                LD   DE,(CONSTS+2) ; x2-,y2-coordinates ("from")
                CP   D              ; is y2 in the screen?
                RET  C              ; if not, then exit the procedure
                LD   A,E            ; coordinate x2
                CP   L              ; compared with coordinate x1
                JR   NC,GO_PSC1     ; if L<E, then everything is fine
                EX   DE,HL          ; otherwise - swapped them
```

```
GO_PSC1:  LD A, D          ; y2 coordinate
          CP   H           ; compared with coordinate y1
          JR NC, GO_PSC2   ; if H<D, then jump over
          LD   D,H         ; otherwise, we change
          LD   H,A         ; their places
GO_PSC2:  LD A,D           ; ;largest of y-coordinates
          SUB H            ; subtract the smaller y-coordinate
          INC  A           ; and add 1
          EXX              ; switched to alternative registers
          LD   B,A         ; y offset value (height)
          EXX              ; back to main registers
          LD   A,E         ; largest of x-coordinates
          SUB  L           ; subtract the smaller x-coordinate
          INC  A           ; added one
          EXX              ; changed the set of registers
          LD   C,A         ; x offset value (width)
          EXX              ; back to basics set of registers
          PUSH AF          ; threw one byte onto the stack (any)
          INC  SP          ; - this is necessary for completion
          LD   C,#08       ; number of bits in one byte
LP_PSC1:  LD   A,E         ; stored in an alternative register
          EX AF,AF         ; coord. x end of line
LP_PSC2:  PUSH HL          ; remember registers HL, BC on the stack
          PUSH BC          ; so as not to spoil
          CALL POINT       ; calling the point checking procedure
          POP  BC          ; restoring registers
          POP  HL          ; registers BC, HL from the stack
          ADD  A,#FF       ; if A>0, then the CF flag will turn on
          RR   B           ; “place” this bit into register B
          DEC  C           ; Decrement the bit counter
          JR   NZ,GO_PSC3  ; if not zero, then jump over
          PUSH BC          ; otherwise, throw it onto the stack
          INC  SP          ; register B (1 byte only)
          LD   C,#08       ; and set the bit counter
GO_PSC3:  LD   A,E         ; current x-coordinate
          DEC  E           ; moving along the line to the left
          CP   L           ; end of line check
          JR   NZ,LP_PSC2  ; rotate the loop along the line
          EX AF,AF         ; restore the value
          LD   E,A         ; x-coordinates from alternate A
          LD   A,D         ; current y-coordinate
          DEC  D           ; moving up the lines
          CP   H           ; was that the last line?
          JR   NZ,LP_PSC1  ; if not, then cycle along the lines
          LD   A,#08       ; number of bits in a byte
          SUB  C           ; A=number of filled bits in reg. B
          JR   NZ,GO_PSC4  ; if not zero, then jump over
```

```
            LD      A,C         ; A=C=8 - number of bits in a byte
            DEC     SP          ; remove the last one from the stack
            POP     BC          ; byte thrown there
GO_PSC4:    LD      C,A         ; how many bits of data are in the sequence? byte
            LD      DE,(CONSTS+6) ; new x-,y-coordinates (“to”)
LP_PSC3:    LD      A,E         ; save the x-coordinate of the beginning
            EX AF,AF            ; ; image lines in A'
            EXX                 ; switch to alternate registers
            LD      E,C         ; this will be a counter of points by x
LP_PSC4:    EXX                 ; back to basics set of registers
            EX AF,AF            ; transition to alternative flag. register
            RLC     B           ; flag CF - output/not_out. point
            EX AF,AF            ; returned to normal flags
            PUSH    BC          ; save the data byte and count. bits
            LD      HL,(CONSTS+4) ; scale of magnification in x and y
            LD      B,H         ; maintained the scale of increase
            LD      C,L         ; in registers C and B (x and y)
            PUSH DE             ; save the coordinates (cycle along the lines)
LP_PSC5:    PUSH    DE          ; save the coordinates (cycle by points)
LP_PSC6:    PUSH    HL          ; save registers HL and BC
            PUSH    BC          ; before calling the POINT procedure
            CALL    POINT       ; calculation of address on screen and mask
            LD      A,C         ; point mask (bit enabled)
            POP     BC          ; restored BC from the stack
            EX AF,AF            ; check the alternative flag CF
            JR      C,GO_PSC5   ; if it is turned on, then jump over
            EX AF,AF            ; save this CF flag
            CPL                 ; invert the point bit mask
            AND     (HL)        ; and use it to reset the pixel
            JR      GO_PSC6     ; transition to continuation
GO_PSC5:    EX AF,AF            ; make the CF flag alternate again
            OR      (HL)        ; turn on the pixel
GO_PSC6:    LD (HL),A           ; writing the modified byte to the screen
            POP HL              ; ;restore HL (scale count)
            INC E               ; jump to next point on the screen line
            DEC     L           ; decrease the x scale counter
            JR      NZ,LP_PSC6  ; not yet zero, continue the cycle
            LD      L,C         ; restore the x-scale value.
            POP     DE          ; restore the coordinates of the beginning of the line
            INC     D           ; move to next line on screen
            DEC     H           ; decrease the y scale counter
            JR      NZ,LP_PSC5  ; and spin until it reaches 0
            LD      H,B         ; restore the y-scale value
            POP     DE          ; restore the coordinates of the point's origin
            LD      A,E         ; go to the beginning of the next
            ADD     A,L         ; rectangle representing
            LD      E,A         ; one image point (right)
```

```
                POP     BC          ; restore byte dx and counter
                DEC     C           ; decrement the bit counter in byte B
                JR      NZ,GO_PSC7  ; if there are still bits, then go
                DEC     SP          ; otherwise, read from the stack
                POP     BC          ; next data byte to register B
                LD      C,#08       ; set the bit counter
GO_PSC7:        EXX                 ; jump to alternate registers
                DEC     E           ; decrease the counter of dots in a line
                JR      NZ,LP_PSC4  ; if there are still points, then we spin
                EXX                 ; back to basics set of registers
                EX AF,AF            ; restore from alternative
                LD      E,A         ; registers x-coordinate of the stitch
                LD      A,D         ; go to the beginning of the next straight line
                ADD     A,H         ; square representing one
                LD      D,A         ; image point (down)
                EXX                 ; jump to alternate registers
                DEC     B           ; decrement the sprite line counter
                EXX                 ; back to basics set of registers
                JR      NZ,LP_PSC3  ; cycle if the lines are not over
                RET                 ; exit procedure
```

original article : https://zxpress.ru/article.php?id=5429

Code optimized from the book
40 Best Machine Code Routines for the ZX Spectrum by John Hardman and Andrew Hewson

https://ia800604.us.archive.org/view_archive.php?archive=/1/items/World_of_Spectrum_June_2017_Mirror/World%20of%20Spectrum%20June%202017%20Mirror.zip&file=World%20of%20Spectrum%20June%202017%20Mirror/sinclair/books/123/40BestMachineCodeRoutinesForTheZXSpectrum.pdf