Управляющая программа для микрокомпьютера SX-28 для экспериментов с импульсным разрядом.

```
; =====================================================================
; Author:  Oleg Baskakov
; PRODUCING OF THE DISCHARGE PULSES AND PULSES FOR AD COMBATA.
; For every discharge palse two pulses for AD combata are produced.
; First at the same time when discharge pulse and second one is shifted.
; Useful pulse is a second pulse. But because there is a noise pulse
; at the time of the begining of discharge which causes acidental runing of AD combata
; we force this pulse of noise by another pulse. These false data
; counting by AD combata will be removed by the C++ program which will processed
; data. It is much more easy way than strugle against nois pulses.
; =====================================================================
; ======= Assembler directives ========
; uses: SX28AC, 2 pages of program memory, 8 banks of RAM, high speed osc.
;       operating in turbo mode, with 8-level stack & extended option reg.
        DEVICE  SX28L,OSCXTMAX
        DEVICE  turbo,stackx_optionx
        ID      'SX_Disch'      ;program ID label
        FREQ    50000000
; ======= RESET ========
;    RESET   reset_entry        ;set reset/boot address. Input point of the program.
; *********************
    SCAN        equ    rb.7     ; Input, scan signal. High level for the forward scan,
                                ; low level for the back scan.
    DisPulse    equu   rb.6     ; Output, to fire a discharge pulse. These pulses occur after dividing of
the
                                ; frequency of HeNe laser and expanded in time.
    HeNe        equ    rb.5     ; Input, HeNe laser. This pin is used for the detection of the edge
interruption.
    ADPulse     equ    rb.4     ; Output, pulses for AD combata shifted in time with respect to the
                                ; discharge pulses.
    UsedScan    equ    rb.2     ; Output, SCAN signal to AD. It is needed for the sinchronization of the
                                ; performance of AD and Sx-chip.
                                ; Now it is a trigger for AD combata and signal to handle PC program.
                                ; It is used instead of SCAN line because from on this pin a much less
                                ; pulse noise than on SCAN pin.
; UsedScan output is not used because after 25.10.01 something happened with electronics and SX chip could
not
; drive AD with this signal. This signal was replaced by scan signal in the electic curcits.
    ADready     equ    rb.0     ; Input, waiting for the start of AD.
    cycles      equ    -120     ; negative number, which determine the time in SX cycles between next
                                ; RTCC interuption.
    org         8
    divider     ds     1        ; the DisPulses occur with a 1:divider ratio of HeNe frquency.
    ShiftCount  ds     1        ; number of the HeNe pulses to be skipped after beginning of the forward
scan
                                ; to fire the first discharge pulse. It is a ring counter which takes a
value in
                                ; the range 0 < ShCount < divider-1.
    SkipCount   ds     1        ; number of the HeNe pulses between two next discharge pulses.
                                ; It is a ring counter which takes a value in the range 0 < ShCount <
divider-1.
    Cdis        ds     1        ; counter responsible for the width of the discharge pulse.
    DisWidth    ds     1        ; width of the discharge pulse.
    Cdel        ds     1        ; counter responsible for the delay of the AD pulse relative to the start
of the
                                ; discharge pulse.
    ADDelay     ds     1        ; delay of the AD pulse.
    CADw        ds     1        ; counter responsible for the width of the AD pulse.
    ADWidth     ds     1        ; width of the AD pulse.
    ADpulsewas  ds     1        ; flag which determines either or not the AD pulse was at the current
discharge cycle.
                                ; =0 if it was not yet.
    Work1       ds     1
   IsNoisePulse ds     1    ; =1 if there is a first, "noise" pulse for AD combata.
   OneTwoPulses ds     1    ; =1 - two pulses, =0 - one pulse (without "noise" pulse).

; ===========  TIMER INTERRUPTION ===========
    org         0           ; It must always start at address of the interruption.
; This routine is responsible for the pulse widths and for the starting of the AD shifted pulses.
    snb         ADPulse     ; is there AD pulse now ? AD pulse can be "noise" or
                            ; useful.
    jmp         :AD_pulse_width  ; Yes. AD pulse exists. Go to manage the pulse widthes.
    snb         ADpulsewas.0     ; Was AD pulse already earlier ?
    jmp         :Dis_pulse_width ; Yes. So let us work with discharge pulse.
    decsz       CDel             ; No. AD pulse is not yet started.
                                 ; So, we must make delay of the AD start.
    jmp         :Dis_pulse_width ; It is not time to start AD pulse. So let us continue
                                 ; to work with discharge pulse.
    setb        ADPulse          ; Is it time to start AD.
    setb        ADpulsewas.0
    jmp         :Dis_pulse_width ; OK with AD start. Go to discharge pulse.
:AD_pulse_width
    test        IsNoisePulse     ; is it "noise" pulse ?
    jz          :not_noise_pulse
    clrb        ADPulse          ; stop "noise" pulse.
    clr         IsNoisePulse
    jmp         :Dis_pulse_width ; Go to verify a discharge pulse.
:not_noise_pulse
    decsz       CADw             ; Is it time to stop AD pulse ?
    jmp         :Dis_pulse_width ; No.
    clrb        ADPulse          ; Yes.
:Dis_pulse_width
    test        CDis
    jz          :out
; It is two old instructions.
;        sb        DisPulse     ; Is a discharge pulse now ?
;        jmp       :out         ; No. Go home.
; It is 7 new instructions.  13.11.01. They have been included to prevent of the influence of
; the pulse noise to give false disision about existing of the pulse discharge.
    mov         Work1,  #5
:CheckDis
    snb         DisPulse         ; Is a discharge pulse now ?
    jmp         :ContinueDis
    decsz       Work1
    jmp         :CheckDis
    jmp         :out             ; No. Go home.
:ContinueDis
    decsz       Cdis             ; It is time to stop discharge pulse ?
    jmp         :out             ; No. Go home.
```

```
    clrb        DisPulse        ; Yes, stop discharge.
;     mov        !option, #%11001000    ; disable interruption from RTCC.
:out  mov       w,  #cycles
    retiw
; ***********  END OF TIMER INTERRUPTION ****
; ===========  RESET ENTRY POINT ============
reset_entry    PAGE        start        ;Set page bits and then
start
    clr         rtcc
    mov         !option,  #%11001000    ; disable interruption from RTCC.
; ========== Adjusting of the parameters ======================
; All time terms are in the terms between the two RTCC interuptions.
    mov         divider,  #1
    mov         ShiftCount,  #1
    mov         DisWidth,  #4  ;#10    ;#4
    mov         ADDelay,  #2   ;#8    ;#2
    mov         ADWidth,  #1
    mov         OneTwoPulses,  #1
; ************************************************************
;========== Set B Port options ================================
    mode        $F              ; direction.
    mov         !rb,  #%10100001 ; B Port directions, 0=OUT 1=IN.
    mode        $A              ; rising edges
    mov         !rb,  #$00      ; at the HeNe input pin and all other pins.
    mode        $9
    mov         !rb,  #$00      ; clearing the pending register.
    mode        $D              ; level
    mov         !rb,  #$FF      ; TTL
    mov         rb,   #$00      ; B Port low levels
    mode        $B
    mov         !rb,  #%11111111 ; disable interrupts from port B.
; Next two instructoins should decrease of an influence of the noise.
    mode        $C              ; Schmitt triggers (0).
    mov         !rb,  #%01011110
; Next two instructions may be not nessesary.
    mode        $E
    mov         !rb,  #%01111111 ; enable pullup register for SCAN.
; ************************************************************
    clr        IsNoisePulse
; Next block is inserted on 25.10.01
;========== Waiting for the command from PC that =============
;=== Next cycle wait for the start of C++ program which must send high level ==========
;=== signal on the ADready pin. The C++ program do it with function AD_Out_Aux(1). ====
:what_does_AD_do
    sb          ADready         ; The chip can go out from this cycle when
    jmp         :what_does_AD_do ; PC program put 1 at ADready pin with AD_Out_Aux(1).
; Next two instructions are nessery if a next version of this program will
; use the interuptions from the b port. But they are kept in this version.
    mode        $9
    mov         !rb,  #$00      ; clearing the pending register.
; ************************************************************
;========== Waiting for the beginning of the forward scan ====
;not_shifted_scan
;     mov        ShiftCount,  #1
    snb         ADready         ; Has AD work been finished ? If C++ program is finished
                                ; then SX go to sleep.
    jmp         :shifted_scan
    mode        $B
    mov         !rb,  #%11111110 ; enable interruption on rb.0 pin. In the future
                                ; SX wakes up from rb.0 signal wich will send by
                                ; the "out(1)" button in C++ program.
    mode        $9
    mov         !rb,  #$00      ; clearing the pending register.
    sleep
:shifted_scan
    mov         !option,  #%11001000    ; disable interruption from RTCC.
    clrb        DisPulse        ; Stop discharge.
;============================================================
    mode        $9
; This cycle detects the start of the scan by occuring 1 in the pending bit
; corresponding to the SCAN. This method can be implemented because
; at the thime of the begining of the scan there is no discharge and, as
; a consequence, no pulse noise.
:start_scan
    clr         w
    mov         !rb,  w         ; swapping and clearing of the pending bits.
    and         w,  #%10000000
    jz          :start_scan

    setb        UsedScan        ; permition for AD to take data.   25.10.01
                                ; This signal triggers AD combata.
;------------------------------------------------------------
; Another realization of the above part of the program.
; It employ a checking of the high level of the SCAN bit instead of
; checking the corresponding pending bit.
; ************************************************************
;:start_scan
;        sb       SCAN
;        jmp      :start_scan
;        setb     UsedScan       ; permition for AD to take data.   25.10.01
;========== Skip ShiftCount pulses of HeNe laser =============
; This realization does not work well. The problems in SX chip. It does not work correctly.
;        mov      Work1,ShiftCount
;        mode     $9
;:skip_pulses                    ; skip HeNe pulses ShiftCount times.
;:wait_for_pulse                 ; waiting for the next HeNe pulse.
;        clr      w
;        mov      !rb,  w        ; swapping and clearing of the pending bits.
;        and      w,  #%00100000
;        jz       :wait_for_pulse
;        decsz    Work1
;        jmp      :skip_pulses
;------------------------------------------------------------
; Another realization of the above part of the program
;        mov      Work1,  #1     ; experiment.
;        mov      Work1,  #24    ; experiment.

    mov         Work1,  ShiftCount
:skip_pulses                    ; skip HeNe pulses ShiftCount times.
:high_HeNe
    snb         HeNe
    jmp         :high_HeNe
:low_HeNe
```

```
    sb          HeNe
    jmp         :low_HeNe
    decsz       Work1
    jmp         :skip_pulses
; ************************************************************

;=========== Timer enable ===================================
:fire_discharge
    clr         rtcc            ; clear timer.
    mov         !option,  #%10001000   ; enable interruption from RTCC.
; ************************************************************
;=========== Fire discharge pulse ===========================
    mov         SkipCount,  divider
    test        OneTwoPulses    ; if zero then a noise pulse is not needed.
    jnz         :noispulse
    mov         rb,  #%01000100  ; setting only the DisPulse bit
                                 ; and leaving UsedScan at high level.
    jmp         :ok
:noispulse
    mov         rb,  #%01010100  ; setting the DisPulse bit and a "noise" AdPulse
                                 ; and leaving UsedScan at high level.
    mov         IsNoisePulse,  #1 ; "noise" pulse start
:ok
    mov         Cdis,  DisWidth
    inc         Cdis
    mov         Cdel,  ADDelay
    inc         Cdel
    mov         CADw,  ADWidth
    inc         CADw
    clr         ADpulsewas
    mov         rtcc,  #$FE     ; initialization of the timer to quickly start the forming
                                ; AD pulse. The number #$FE results rtcc interuption after
                                ; next two instruction cycles. And therefore a counting of
                                ; the all delays and widths starts almost immediately after
                                ; begining of the discharge pulse (in the interuption
                                ; procedure.
; ************************************************************
;=========== Varification of a forward scan ==================
:skip_pulses_1                  ; skip HeNe pulses SkipCount times.
    sb          SCAN
    jmp         :new_scan_prep ; forward scan is over. Go to prepare a new scan.
; ************************************************************
;=========== Waiting for the next HeNe pulse ================
; It should be noted that, first, divider=1 means no skipping of the HeNe pulses
; and, second, at the moment of the start of the bellow loop to count skipped
; HeNe pulses the discharge has already burned.
;------------------------------------------------------------
; This realization of the skipping of the HeNe pulses does not
; work because pending bits in B port are loos in the interruption
; routine
;       mode    $9
;:wait_for_pulse_1               ; waiting for the next HeNe pulse.
;       clr     w
;       mov     !rb,        w   ; swapping and clearing of the pending bits.
;       and     w,  #%00100000
;       jz      :wait_for_pulse_1
;       decsz   SkipCount
;       jmp     :skip_pulses_1
;       jmp     :fire_discharge ; a needed number of HeNe pulses is skipped so let us
;                               ; immediatly fire discharge at the present HeNe pulse.
;------------------------------------------------------------
; Another realization of the above part of the program which
; produces the skipping of a needed number of HeNe pulses.
:high
    snb         HeNe
    jmp         :high
:low
    sb          HeNe
    jmp         :low
    decsz       SkipCount
    jmp         :skip_pulses_1
    jmp         :fire_discharge  ; a needed number of HeNe pulses is skipped so let us
;                                ; immediatly fire discharge at the present HeNe pulse.
; ************************************************************
;=========== Preparation to next scan ========================
:new_scan_prep
    mov         !option,  #%11001000  ; disable interruption from RTCC.
    mode        $9
    mov         !rb,  #0        ; clearing the pending bits. They could contain the old data.
    mov         rb,  #%00000000  ; stop all pulses.
;       clrb        UsedScan        ; prohibition for AD to take data.    25.10.01
;       clrb        DisPulse        ; stop discharge pulse.
;       clrb        ADPulse         ; stop AD pulse.
;       inc         ShiftCount

;       cja         ShiftCount,divider,:not_shifted_scan
;       jmp         :shifted_scan
    jmp         :not_shifted_scan
; ************************************************************
end
```