ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΜΠ

‘’ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ”

Μπακούρος Αριστείδης 03113138
Ορφανουδάκης Φίλιππος 03113140
Ρουσσάκης Γεώργιος 03113041

5η Σειρά Ασκήσεων

Ζήτημα 5.1

Στο πρώτο ζητούμενο χρειάστηκε να αξιοποιήσουμε τη ρουτίνα του μετρητή που αναφέρεται στην εκφώνηση και να εισάγουμε κατάλληλα τη ρουτίνα διακοπής INT1. Συγκεκριμένα έπρεπε στην πύλη A(PA0-PA7) να απεικονίζεται το πλήθος των διακοπών που εισάγονται μέσω του πλήκτρου PD3 εφόσον είναι πατημένο το PD7 αλλιώς να μην καταχωρείται διακοπή. Επίσης ζητήθηκε η δημιουργία μιας ρουτίνας για την αποφυγή φαινομένων σπινθηρισμού.

```
.include "m16def.inc"
.def temp = r17
.org 0x0
rjmp main
.org 0x4                                ;;Διεύθυνση της ρουτίνας INT1
rjmp ISR1

main:
ldi temp,high(RAMEND)                   ;; Αρχικοποίηση της στοίβας
out SPH, temp
ldi temp,low(RAMEND)
out SPL, temp
ldi r24 ,( 1 << ISC11) | ( 1 << ISC10)  ;; συνθήκες για την int1
out MCUCR, r24
ldi r24 ,( 1 << INT1)
out GICR, r24
sei
ser r26
out DDRA, r26                           ;; αρχικοποίηση της εξόδου A
                                        ;;που απεικονίζει τις διακοπές
out DDRB, r26                           ;;αρχικοποίηση της εξόδου B
                                        ;;που απεικονίζει το μετρητή
clr r26
clr r23
out DDRD, r26                           ;;αρχικοποίηση εισόδου D
loop:
out PORTB, r26                          ;; εκτύπωση στη θύρα B
ldi r24, low(200)                       ;;φόρτωση καθυστέρησης 0.2sec
ldi r25, high(200)                      ;;στους r24,25 για τη wait
rcall wait_msec                         ;;κάλεσμα καθυστέρησης
inc r26                                 ;;αύξηση r26 που χρησιμοποιούμε
                                        ;;για το μέτρημα των διακοπών
rjmp loop

wait_usec:
```

```
        sbiw r24 ,1
        nop
        nop
        nop
        nop
        brne wait_usec
        ret
wait_msec:
        push r24
        push r25
        ldi r24 , low(998)
        ldi r25 , high(998)
        rcall wait_usec
        pop r25
        pop r24
        sbiw r24 , 1
        brne wait_msec
        ret                 ;
ISR1:                                   ;;ρουτίνα εξυπηρέτησης διακοπής
        rcall protect
        push r26                        ;;σώσε το περιεχόμενο των r26
        in r26, SREG                    ;;και SREG
        push r26
        in temp,PIND                    ;;θέσε τη θύρα εξόδου των LED
        sbrs temp,7                     ;;έλεγχος αν είναι πατημένο το PD7
        rjmp ISR1_exit                  ;;αν όχι φύγε
        inc r23
        out PORTA, r23
        ldi r24, low(998)               ;;φόρτωσε τους r24,25 με καθυστέρηση 1sec
        ldi r25, high(998)
ISR1_exit:
        pop r26
        out SREG, r26
        pop r26
        sei
        reti
protect:                                ;;
        ldi temp,0x80
        out GIFR,temp
        ldi r24,0x05                    ;;εισαγωγή μικρής καθυστέρησης για
        ldi r25,0x00                    ;;αποφυγή σπινθηρισμού
        rcall wait_msec
        in temp,GIFR                    ;;έλεγχος και επιβεβαίωση ότι υπάρχει
        sbrc temp,7                     ;;μία μόνο διακοπή
        rjmp protect
        ret
```

Ζήτημα 5.2

- Στο ζήτημα αυτό κληθήκαμε να χρησιμοποιήσουμε την ρουτίνα διακοπής **INT0** που ενεργοποείται με το **PD2**. Πιο συγκεκριμένα έχουμε το κύριο πρόγραμμα μας που μετράει προς τα πάνω συνεχώς **mod256** , και όταν ενεργοποιήσουμε την διακοπή τότε πρέπει στην έξοδο μας -PORTC- να απεικονίζουμε τόσα led αρχίζοντας απο το LSB όσοι διακόπτες του PORTA είναι ενεργοποιημένοι.

- Το σκεπτικό είναι να κανουμε 8 φορες τη loop για την είσοδο μας και όποτε βρίσκουμε 1 σε bit της εισόδου μας να έχουμε έναν καταχωρητή και να τον κάνουμε **shift left** και ύστερα **increase.**

- Το κύριο πρόγραμμα μας είναι απλά ένας καταχωρητής που αυξάνεται με μια καθυστέρηση 200msec.

- Περισσότερες λεπτομέρειες στον κώδικα και τα σχόλια του

```
.include "m16def.inc"
.def reg = r24
.def temp = r26

.org 0x0
rjmp reset
.org 0x2                    ;Address of the routine2-INT0
rjmp routine2
reti

reset:
;----------------.INITIALIZE STACK.-----------------------;
          ldi temp,high(RAMEND) ;Initialize stack
          out SPH, temp
          ldi temp,low(RAMEND)
          out SPL, temp

;---------.MAKE THE PROGRAM SESNITIVE TO INTERRUPTS INT0.--------;

          ldi reg, (1 << ISC00) | (1 << ISC01)  ;Set the options
for interrupt INT0
          out MCUCR, reg
          ldi reg, (1 << INT0)
          out GICR, reg
          sei             ;Enable interrupts
```

```
;------------------.SET OUTPUT AND INPUT.----------------------;
           ser temp
           out DDRC, temp   ;PORTC is output (interrupts' counter)
           out DDRB, temp   ;PORTB is output (counter from 1 to 255)
           ldi temp, 0           ;PORTD is the input (Interrupt)
           out DDRD, temp
           out DDRA,temp
           ldi temp, 0xFF              ;Set the Counter

;--------------.ROUTINE OF THE MAIN PROGRAMM-COUNTING.------;
loop:
           inc temp         ;Increase the counter
           out PORTB, temp  ;Display to PORTA
           ldi r24, low(200)     ;Wait for 0.10 sec (100 msec)
           ldi r25, high(200)
           rcall wait_msec
           rjmp loop  ;Continuous operation (when the counter
reaches 255 it resets)



;--------------.PREVENTING SPARKLING- INT0.---------------;

routine2:
           ldi r24, low(5)
           ldi r25, high(5)      ;Small delay to check and avoid
sparkling
jumpee:
           ldi reg, (1 << INTF0)  ;Check for misses when the PD2
button is pressed
           out GIFR, reg         ;Make sure that only one interrupt
will be processed
           rcall wait_msec
           in reg, GIFR
           sbrc reg,6
           rjmp jumpee           ;End loop

;-------------.MAIN ROUTINE OF INT0.---------------;

           push temp
           in temp, SREG
```

```
           push temp
           ldi r20,0            ;register where we store the ouput
           ldi r19,8            ;Loop counter-8 loops one for each
bit of the input
           in r18,PINA       ;Take input
again:
           lsr r18             ;logic shift right our input
           brcc next         ; chech if the bit is 1 or 0
           lsl r20              ; shift left our counter
           inc r20            ;increase by 1
next:
           dec r19         ; decrease loop counter
           cpi r19,0
           brne again

           out PORTC,r20 ; print
           pop temp
           out SREG,temp
           pop temp
           reti

;------------.WAIT ROUTINES.--------------------;
wait_usec:
      sbiw r24 ,1
      nop
      nop
      nop
      nop
      brne wait_usec
      ret


wait_msec:
      push r24
      push r25
      ldi r24 , low(998)
      ldi r25 , high(998)
      rcall wait_usec
      pop r25
      pop r24
      sbiw r24 , 1
      brne wait_msec
```

```
      ret
```

Ζήτημα 5.3

Για την άσκηση αυτή υλοποιήθηκε αυτοματισμός που ελέγχει ένα φωτιστικό σώμα. Το σώμα άναβε είτε με το πάτημα του push button PD3 (που ουσιαστικά ενεργοποιούσε την διακοπή INT1) είτε με το push button PA7 (υποθέτουμε ότι αντιστοιχεί σε αισθητήρα κίνησης). Η αναπαράσταση του σώματος γίνεται στο led του PB0. Κάθε φορά που πατιέται κάποιο εκ των PD3, PA7 τότε όλη η σειρά των led των PB0-PB7 ανάβει για 0.5 secs. Μετά το πέρας του μισού δευτερολέπτου παραμένει ανοικτό μόνο το PB0 για 4 secs συνολικά, εκτός και αν ξαναπατηθεί κάποιο από τα PA7,PD3 οπότε έχουμε ανανέωση του χρόνου και έτσι θα ανάψει ξανά όλη η σειρά των led PB0-PB7 και μετά τα νέα 0.5 secs θα μείνει ανοικτό μόνο το PB0 για 4 secs. Παρακάτω φαίνεται ο κώδικας μαζί με τα απαραίτητα σχόλια για την κατανόησή του. Να σημειωθεί ότι για τη μέτρηση των 4 secs χρησιμοποιήθηκε ο χρονιστής Timer1 (TCNT1, διαλέξαμε τον 16-bit χρονιστή καθώς μόνο αυτός ήταν δυνατόν να μετρήσει 4 secs).

```
.include "m16def.inc"
.def temp=r16
.def mule=r17  ; used for PA7 input
.def clock_h=r18 ;for the timer
.def clock_l=r19 ;for the timer
.def leds=r20 ; used for keeeping PB0 lit after 0.5 secs
.def int_mule=r22; used for PD3 input
.org 0x00
      jmp main   ; Reset Handler
.org 0x04
      jmp ISR1   ; IRQ1  Handler
.org 0x10
      jmp timer1_rout
main:
      ldi temp,high(RAMEND)
      out SPH,temp
      ldi temp,low(RAMEND)
      out SPL,temp               ; stack init
;-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-
```

```
      ldi r24 ,( 1 << ISC11) | ( 1 << ISC10)
      out MCUCR, r24
      ldi r24 ,( 1 << INT1)
      out GICR, r24              ;INT1 and protect from sparkling
;setup

      ldi temp,0x05              ; Frequency Divisor = 1024
      out TCCR1B,temp
      ldi temp,0x04              ; Enable Timer1
      out TIMSK,temp
      sei

      clr temp
      out DDRD,temp              ; pullup resistance for input
      ser temp
      out DDRB,temp

      ldi clock_h,0x85      ; clock_h:clock_l = 0x85EE
;{=65536-4*7812.5}, where 7812.5Hz = cycles/sec =~ 8MHz/1024
      ldi clock_l,0xEE      ;
      ldi leds,0x01              ; PB0
loop:
      in mule,PINA               ; porta -> input
      sbrs mule,7               ; If PA7==1 exit loop and do stuff
;with leds
      rjmp loop
      ldi temp,0xFF
      out PORTB,temp            ; light up PB0-PB7 leds
      ldi r24,low(500)
      ldi r25,high(500)         ; delay 500 ms
      rcall wait_msec
      out PORTB,leds             ; led PB0 on due to PA7
      out TCNT1H,clock_h         ; 4secs
      out TCNT1L,clock_l
      in mule,PINA
      sbrc mule,7                ; if PA7==0, ignore next command
      clr r21
      out PORTB,r21               ; turn PB0 led off
      rjmp loop
;-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-
ISR1:
```

```
      rcall protect              ;prevent sparkling
      push r26
      in r26, SREG
      push r26
      ldi temp,0xFF              ; on refresh, light em up
      out PORTB,temp
      ldi r24,low(500)
      ldi r25,high(500)         ; delay 500 ms
      rcall wait_msec
      out PORTB,leds             ; led on
      out TCNT1H,clock_h         ; reset clock
      out TCNT1L,clock_l
      pop r26
      out SREG, r26
      pop r26
      sei
      in int_mule, PIND
      sbrc int_mule,3          ; if PD3==0, ignore next command
      reti
      clr r22
      out PORTB,r22            ;turn PB0 led off
    reti

timer1_rout:
      clr leds
      out PORTB,leds             ; timer overflow -> lights out
      ldi leds,0x01              ; led PB0 needs to light up again
      sei
      reti

wait_usec:
      sbiw r24 ,1           ; 2 cycles (0.250 micro sec)
      nop                   ; 1 cycles (0.125 micro sec)
      nop                   ; 1 cycles (0.125 micro sec)
      nop                   ; 1 cycles (0.125 micro sec)
      nop                   ; 1 cycles (0.125 micro sec)
      brne wait_usec   ; 1 or 2 cycles (0.125 or 0.250 micro sec)
      ret                   ; 4 cycles (0.500 micro sec)

wait_msec:
      push r24         ; 2 cycles (0.250 micro sec)
      push r25         ; 2 cycles
```

```
     ldi r24 , 0xe6   ; load register r25:r24 with 998 (1 cycles -
0.125 micro sec)
     ldi r25 , 0x03   ; 1 cycles (0.125 micro sec)
     rcall wait_usec  ; 3 cycles (0.375 micro sec), total delay
998.375 micro sec
     pop r25               ; 2 cycles (0.250 micro sec)
     pop r24               ; 2 cycles
     sbiw r24 , 1     ; 2 cycles
     brne wait_msec   ; 1 or 2 cycles (0.125 or 0.250 micro sec)
     ret                   ; 4 cycles (0.500 micro sec)

protect:
     ldi temp,0x80         ;0b1000 0000
     out GIFR,temp         ;Setting zero INTF1
     ldi r24,0x05
     ldi r25,0x00
     rcall wait_msec       ;wait 5 msec
     in temp,GIFR          ;Check if INTF1==1
     sbrc temp,7
     rjmp protect          ;If INTF1==1 loop
     ret                        ;If INTF1==0 return
```