Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών
Εργαστήριο Μικροϋπολογιστών

Έκθεση 3ης Εργαστηριακής Άσκησης

Στοιχεία:
Ομάδα: **39**
Μέλη: **Ευάγγελος Μυργιώτης, Αριστοτέλης Γρίβας**

Ζήτημα 3.1 :

Ο κώδικας χρησιμοποιεί την ρουτίνα εξυπηρέτησης του INT για να ενημερώσει τον r27 (ανάλογα αν πρέπει να γίνει ανανέωση ή όχι). Εντός της main, γίνεται έλεγχος του r27 και του PC5, και ανάβει τα λαμπάκια ανάλογα αν πρέπει να γίνει ανανέωση ή όχι. Το αν πρέπει να γίνει ανανέωση ή όχι εξαρτάται από το αν εντοπίσουμε το PB0 αναμμένο.

Τα λαμπάκια σβήνουν αυτόματα μέσω της ρουτίνας διακοπής όταν ο Timer κάνει overflow που γίνεται μετά από 4 δευτερόλεπτα όταν το θέσουμε στο 3036 και με την ταχύτητα που αυξάνεται.

Κώδικας σε Assembly :

```
.include "m328PBdef.inc"
.equ FOSC_MHZ=16
.equ DEL_ms=600
.equ Del_NU=FOSC_MHZ*DEL_ms

.org 0x0
rjmp reset
.org 0x4
rjmp ISR1
.org 0x1A
rjmp ISR_TIMER1_OVF
sei                     ;  timer1

reset:

ldi r24,(1 << ISC11) | (1 << ISC10)      ;enable interrupt on rising edge of INT1
sts EICRA, r24

ldi r24, (1 << INT1)                     ;enable INT1 interrupt
out EIMSK, r24

ldi r24, (1<<TOIE1)                      ;TCNT1
sts TIMSK1, r24                  ;timer1

ldi r24 ,(1<<CS12) | (0<<CS11) | (1<<CS10) ; CK/1024
sts TCCR1B, r24
```

```
clr r27
sei                                     ;enable interrupts


ldi r24, LOW(RAMEND)
out SPL, r24
ldi r24, HIGH(RAMEND)
out SPH, r24

clr r30                                 ;PORTD input and PORTb output
out DDRD, r30
out DDRC,r30
ser r30
out DDRB, r30


main:

    lsr r27                     ;if LSB of r27 is 0 then do nothing
    brcs start
    in r21,PINC
    lsl r21
    lsl r21
    lsl r21
    brcs again

 check:
    in r21,PINC
    lsl r21
    lsl r21
    lsl r21
    brcc check


 start:
    ldi r24, HIGH(3036) ;  TCNT1
    sts TCNT1H, r24       ;  will overflow after 4 secs and turn leds off
    ldi r24, LOW(3036)
    sts TCNT1L, r24


    in r25, PORTB                   ;store input in r25
    lsr r25                         ;check LSB of input
    brcc no_refresh           ;if it is 1 then we need to open all leds for 500ms


    ldi r28, 0xFF     ;if refreshed,open all leds for 500ms

    out PORTB, r28            ;r28 = 0xFF as output

    ldi r24, low(16*500)    ;500ms delay
    ldi r25, high(16*500)
    rcall delay_ms
```

```
no_refresh:
    ldi r28, 0x01      ;else we need to open LSB led of portB for 4s
    out PORTB, r28             ;r28 = 1 as output


      again:
      brcs main
      clr r27
      rjmp main




delay_ms:                  ;delay routine used in previous exercises too

ldi r23, 249

loop_inn:

dec r23
nop
brne loop_inn
sbiw r24, 1
brne delay_ms
ret


ISR1:

    push r24
    push r25
    in r24, SREG
    push r24

    ldi r24, (1 << INTF1)
    out EIFR, r24

    ldi r24, low(16*5)                 ;start spin8
    ldi r25, high(16*5)
    rcall delay_ms

    in r24, EIFR
    lsr r24
    lsr r24
    brcs ISR1                          ;end spin8

    ldi r24, HIGH(3036) ;  TCNT1
    sts TCNT1H, r24        ; 3 sec
    ldi r24, LOW(3036)
    sts TCNT1L, r24
```

```
    ldi r27, 0x01           ;r27 = 00000001
    in r25, PORTB                   ;store input in r25
    lsr r25                         ;check LSB of input
    brcc done                       ;if MSB LED is closed then continue to main
    ldi r27, 0x03           ;else r27 = 00000011 and continue to main


done:
    pop r24
    out SREG, r24
    pop r25
    pop r24
    sei
    rjmp main



ISR_TIMER1_OVF:
    push r24
    push r25
    in r24, SREG
    push r24

    ldi r24, 0x00  ;turn off leds
    out PORTB, r24

    pop r24
    out SREG, r24
    pop r25
    pop r24
    reti
```

Υλοποίηση σε C :

Η βασική διαφορά είναι ότι μέσα στην ρουτίνα διακοπής του INT1 διαχειριζόμαστε το άνοιγμα των led.

Έχουμε ενεργοποιήσει τα global interrupt εντός της ρουτίνας, οπότε υπάρχει ενδεχόμενο να ξανακάνουμε interrrupt ενώ ήμαστε εντός της ρουτίνας. Για να μπορέσουμε να διαχειριστούμε τις αναδρομικές κλήσεις έχουμε μια global μεταβλητή check, η οποία ενημερώνεται σε 1 στην αρχή κάθε διακοπής ενώ ενημερώνεται σε 0 στην έξοδο κάθε κλήσης. Ο έλεγχος του check γίνεται εντός ενός loop, το οποίο είναι υπεύθυνο για την ένδειξη της ανανέωσης ανάβοντας τα led για 0.5s. Στην τελευταία κλήση interrupt το loop θα ολοκληρωθεί και μετά από 0.5s θα σβήσουν τα led και θα ενημερωθεί το check. Επιστρέφοντας λοιπόν στην προηγούμενες interrupt, αφού το check θα είναι μηδέν θα τελειώσουν κατευθείαν χωρίς να περιμένουν ms παραπάνω.

Να σημειωθεί ότι χρησιμοποιούμε την sei μόνο όταν το check έχει ενημερωθεί κατάλληλα και λάβουμε υπόψη τον σπινθηρισμό.

Κώδικας σε C :

```
#define F_CPU 16000000UL
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>


unsigned char check = 0;
unsigned char check2 = 0;



ISR(TIMER1_OVF_vect){
	PORTB = 0x00;
	sei();
}

ISR(INT1_vect) {
TCNT1 = 3036;
while(1){
   EIFR = (1 << INTF0);
   _delay_ms(5);
  // int a = EIFR &0x02;
   if(!(EIFR &0x01)) break;
}
check = 1;		//ENTERED INTERR
sei();

TCNT1 = 3036;
   if((PINB & 0x01)){
	PORTB = 0b00111111;
	for(int i = 0; i < 500; i++){

	   if(check == 0){
	      break;
	   }
	   _delay_ms(1);
	}
   }
	PORTB = 0x01;   //close leds
	check = 0;
	check2 = 0;

	    //EIFR.1 = 0

}
```

```
int main()
{
    EICRA=(1 << ISC11) |(1 << ISC10);
    EIMSK= (1 << INT1);

    TIMSK1 = ( 1 << TOIE1);

    TCCR1B =  (1<<CS12) | (0<<CS11) | (1<<CS10);

    DDRB |= 0xFF;           //PORTB output
    DDRD |= 0x00;           //PORTD input
    DDRC |= 0x00;

    while(1){
        sei();              //enable interrupts
        check2 = 1;
        if( !(PINC & 0x20)){
            while(!(PINC & 0x20)){

            }

         _delay_ms(5);

             if((PINB & 0x01)){
                while( !(PINC & 0x20)){
                //PORTB = 0b00111111;
                _delay_ms(5);
                TCNT1 = 3036;
                }
                PORTB = 0b00111111;
                for(int i = 0; i < 500; i++){

                    if(  ( !(PINC & 0x20)) || (check2 == 0) ){
                    break;
                    }
                    _delay_ms(1);
        }

    }

            PORTB = 0x01;
            TCNT1 = 3036;


        }
        sei();
    }
    sei();
```

```
}
```

Ζήτημα 3.2 :

Στο mode που είμαστε λόγω των TTCR1A, TCCR1B το Top είναι default 256. Υπολογίζοντας λοιπόν τα κατάλληλα ποσοστά του 256, τα αποθηκεύουμε στο table, για γρήγορη πρόσβαση. Στην assembly κάνουμε πρόσβαση του πίνακα μέσω του r24, που λειτουργεί ως pointer στον πίνακα, όπου επειδή οι τιμές του table είναι αποθηκευμένες σε 16-bit λέξεις του προσθαφαιρούμε ανά 2 για την πρόσβαση του πίνακα.

Κώδικας σε Assembly :

```
.include "m328PBdef.inc"

.org 0x00
reset:

ldi r24 , (1<<WGM10) | (1<<COM1A1);COM1A overrides PORTx
sts TCCR1A, r24

ldi r24 ,(1<<WGM12) | (1 << CS11)
sts TCCR1B, r24


 ldi r24, 0x80              ;DUty cycle 50%,TOP = 0x80
 sts OCR1AL, r24        ;    Bottom = 0
 clr r24
 sts OCR1AH, r24

 ldi r24, LOW(RAMEND)
out SPL, r24
ldi r24, HIGH(RAMEND)
out SPH, r24
```

```
ser r24
out DDRB, r24


ldi r29, 0x0C

clr r24
out DDRD, r24

 ldi Zh, high(Table*2)
 ldi Zl, low(Table*2+12)

 clr r18
   ; add Zl, r29
   ; adc Zh, r18
    lpm
    mov r20, r0
   ; ldi r20, 128
    sts OCR1AL, r20
    clr r20
    sts OCR1AH, r20


main:
     ldi Zh, high(Table * 2)
     ldi Zl, low(Table * 2 )



    in r24, PIND
    com r24
    cpi r24, 0x02
    breq increase
    cpi r24, 0x04
    breq decrease
    rjmp main

increase:

    checki:
    ldi r24, low(16*5)    ;500ms delay
    ldi r25, high(16*5)
    rcall delay_ms
    in r24, PIND
    com r24
    cpi r24, 0x02
    breq checki

    cpi r29, 24
    breq no_increase
    subi r29, -2
no_increase:
    clr r18
    add Zl, r29
```

```
    adc Zh, r18
    lpm
    mov r20, r0


    sts OCR1AL, r20
    clr r20
    sts OCR1AH, r20

    rjmp main
decrease:

    checkd:
    ldi r24, low(16*5)    ;500ms delay
    ldi r25, high(16*5)
    rcall delay_ms
    in r24, PIND
    com r24
    cpi r24, 0x04
    breq checkd

     cpi r29, 0
    breq no_decrease
    subi r29, 2
no_decrease:
    clr r18
    add ZI, r29
    adc Zh, r18
    lpm
    mov r20, r0

    sts OCR1AL, r20
    clr r20
    sts OCR1AH, r20

    rjmp main


    delay_ms:                ;delay routine used in previous exercises too

ldi r23, 249

loop_inn:

dec r23
nop
brne loop_inn
sbiw r24, 1
brne delay_ms
ret


.org 0x2000
Table:
```

```
.DW  0x0006, 0x001A,0x002E, 0x0043 ,0x0057 ,0x006C,0x0080, 0x0094 , 0x00A9 , 0x00BD , 0x00D2 , 0x00E6 ,0x00FA

;.DW  0x06, 0x1A,0x2E, 0x43 ,0x57 ,0x6C,0x80, 0x94 , 0xA9 , 0xBD , 0xD2 , 0xE6 ,0xFA
;.DW  0x1A06, 0x432E,0x6C57, 0x9480 ,0xBDA9 ,0xE6D2,0x00FA
```

Κώδικας σε C :

```
  #define F_CPU 16000000UL
#include "avr/io.h"
#include "util/delay.h"
int main(){

unsigned char duty;
unsigned char i;
unsigned char x, y;
unsigned char a[13] = {5,26,46,67,87,108,128,148,169,190,210,230,251};


TCCR1A = (1 << WGM10) | (1 << COM1A1);
TCCR1B = (1 << WGM12) | (1 << CS11);

DDRB |=  0b00111111;
DDRD |=  0b00000000;

duty = 128;
i = 6;
OCR1AL = duty;


while(1){

   x = ~PIND;

   if((x & 0x02)){

      if(i != 12){
         i +=1;
      }
      duty = a[i];
      OCR1AL = duty;
      while(1){
         y = PIND;
         OCR1AL = duty;
         if(y & 0x02) break;
      }
   }

   if((x & 0x04)){
      if(i != 0){
         i -=1;
```

```
        }
        duty = a[i];
        OCR1AL = duty;
        while(1){
            y = PIND;
            OCR1AL = duty;
            if(y & 0x04) break;
        }
    }
    else {
         duty = a[i];
         OCR1AL = duty;
    }

    }
}
```

Ζήτημα 3.3 :

Στους παρακάτω κώδικες ελέγχουμε εντός της main πιο κουμπί έχουμε πατήσει και κάνουμε συνέχει loop μέχρι να αφήσουμε το κουμπί.

Ανάλογα πιο κουμπί έχουμε πατημένο θέτουμε την τιμή του ICR1 για να έχουμε την συχνότητα που θέλουμε καθώς σε αυτό το mode το ICR1 είναι το TOP. Επίσης βάζουμε το OCR1A στην μισή τιμή του ICR1 καθώς θέλουμε το Duty cycle να είναι 50%.

Να σημειωθεί ότι θέτουμε και τον TIMER στο 0 κάθε φορά που αλλάζουμε κουμπί. Καθώς αφού αλλάζουμε το top υπάρχει ενδεχόμενο ο Timer να έχει ήδη ξεπεράσει το νέο Top, οπότε αν δεν το κάνουμε μηδέν θα πρέπει να περιμένουμε να κάνει overflow ο timer που είναι ιδιαίτερα χρονοβόρο.

Κώδικας :

```
.include "m328PBdef.inc"


reset:

ldi r24 ,(1<<WGM11) | (0<<WGM10) | (1<<COM1A1)  ;COM1A overrides PORTx
sts TCCR1A, r24

ldi r24 ,(1<<WGM12) | (1<<WGM13) | (1<< CS12) |(1<< CS10)
sts TCCR1B, r24
```

```
 ldi r24, 0x80              ;DUty cycle 50%,TOP = 0x80
 sts OCR1AL, r24        ;    Bottom = 0
 clr r24
 sts OCR1AH, r24

 sts ICR1L, r24
 sts ICR1H, r24

clr r24
out DDRD, r24
ser r24
out DDRB,r24


main:
    ldi r24, LOW(RAMEND)
    out SPL, r24
    ldi r24, HIGH(RAMEND)
    out SPH, r24


   ldi r24, HIGH(0) ;  TCNT1
    sts TCNT1H, r24
    ldi r24, LOW(0)
    sts TCNT1L, r24

    in r24, PIND
    com r24
    cpi r24, 0x01
    breq Button0
    cpi r24, 0x02
    breq Button1
    cpi r24, 0x04
    breq Button2
    cpi r24, 0x08
    breq Button3
    rjmp main

Button0:

 ldi r24, 62
 sts OCR1AL, r24
 clr r24
 sts OCR1AH, r24
 ldi r24, 124               ;TOP
 sts ICR1L, r24
 clr r24
 sts ICR1H, r24

 in r24, PIND
 com r24
 cpi r24, 0x01
```

```
 breq Button0
 rjmp exit
Button1:
  ldi r24, 31
 sts OCR1AL, r24
 clr r24
 sts OCR1AH, r24
 ldi r24, 62                ;TOP
 sts ICR1L, r24
 clr r24
 sts ICR1H, r24

 in r24, PIND
 com r24
 cpi r24, 0x02
 breq Button1
 rjmp exit
    rjmp exit


Button2:
 ldi r24, 16
 sts OCR1AL, r24
 clr r24
 sts OCR1AH, r24
 ldi r24, 31                ;TOP
 sts ICR1L, r24
 clr r24
 sts ICR1H, r24

 in r24, PIND
 com r24
 cpi r24, 0x04
 breq Button2
 rjmp exit
    rjmp exit
Button3:
     ldi r24, 8
 sts OCR1AL, r24
 clr r24
 sts OCR1AH, r24
 ldi r24, 15                ;TOP
 sts ICR1L, r24
 clr r24
 sts ICR1H, r24

 in r24, PIND
 com r24
 cpi r24, 0x08
 breq Button3
 rjmp exit

exit:
```

```
    rjmp main
```

Κώδικας C :

```
#define F_CPU 16000000UL
#include "avr/io.h"
#include "util/delay.h"

int main(){

unsigned char duty;
//unsigned char i;
unsigned char x;


TCCR1A = (1 << WGM11) | (0 << WGM10)| (1 << COM1A1);
TCCR1B = (1 << WGM12) | (1 << WGM13) | (1 << CS12) |(1<< CS10);

DDRB |=  0b00111111;
DDRD |=  0b00000000;

duty = 128;
OCR1AL = duty;


while(1){
    x = ~PIND;

    if( x & 0x01){
        TCNT1=0;

      while( x & 0x01 ){
      OCR1AL = 62;
      ICR1 = 124;
      x = ~PIND;
      }

    }


    else if( x & 0x02 ){
        TCNT1=0;

      while( x & 0x02 ){
```

```
      OCR1AL = 31;
      ICR1 = 62;
      x = ~PIND;
     }

  }

  else if( x & 0x04 ){
     TCNT1=0;

    while( x & 0x04 ){
    OCR1AL = 15;
     ICR1 = 31;
    x = ~PIND;
    }
  }

  else if( x & 0x08 ){
     TCNT1=0;

    while( x & 0x08 ){
    OCR1AL = 7;
     ICR1 = 15;
     x = ~PIND;
    }
  }

  else{
  OCR1AL = 0;
  ICR1 = 0;
  TCNT1 = 0;
  }
  }
}
```