2η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"

Δρυς-Πεντζάκης Οδυσσεύς el19192
Μαρκαντωνάτος Γεράσιμος el19149

Ζήτημα 2.1

Στο ζήτημα αυτό όταν καλείται interrupt INT1 η ρουτίνα διακοπής μετράει τον αριθμό διακοπών. Για να αποφύγουμε την αναπήδηση όταν λαμβάνουμε διακοπή ελέγχουμε συνεχώς την τιμή του INTF1 μέχρι να γίνει 0 έτσι όσες φορές και αν πατηθεί με τη πίεση του κουμπιού θα μετρήσει για μια. Επίσης ελέγχουμε την τιμή του PD7 κάθε φορά που γίνεται διακοπή ώστε αν είναι πατημένο να μην εκτελεστεί η ρουτίνα διακοπής. Προσέχουμε επίσης μέσα στην διακοπή να καλέσουμε την delay αφού κάνουμε push τους καταχωρητές έτσι ώστε να εκτελεστεί η delay με σωστές τιμές στον διπλό καταχωρητη r25.

```
.include "m328PBdef.inc"

.equ FOSC_MHZ=16

.equ DEL_mS=500

.equ DEL_NU= FOSC_MHZ*DEL_mS

.org 0x0
rjmp reset

.org 0x4
rjmp ISR1

  ;not ours

  reset:
        ldi r23,(1<< ISC11)|(1<<ISC10)
        sts EICRA, r23

        ldi r23, (1<<INT1)
        out EIMSK, r23

        sei
```

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

   ser r26
   out DDRB, r26
   ser r28
   out DDRC, r28
   clr r20
   out PORTC, r20


loop1:
    clr r26
loop2:
    out PORTB, r26

    ldi r24, low(500*16)
    ldi r25, high(500*16)
    rcall delay_mS

   inc r26

   cpi r26, 16
   breq loop1
   rjmp loop2

delay_mS:

    ldi r23, 249
loop_inn:
    dec r23
    nop
    brne loop_inn

    sbiw r24, 1
    brne delay_mS

    ret

ISR1:
```

```
    push r23
    push r24
    in r24, SREG
    push r24
start:
        ldi r16, (1 << INTF1)
        out EIFR, r16
        ldi r24, low(5*16)
        ldi r25, high(5*16)
        rcall delay_mS

        in r16, EIFR
        cpi r16,0
        brne start
    rjmp routine


  routine:
    in r23, PIND
    com r23
    andi r23, 0x80
    cpi r23, 0x00
    brne endint
    in r24, PORTC
    cpi r24, 0x1F
    breq int_full
    inc r24
    out PORTC, r24
   endint:
    pop r23
    pop r24
    out SREG, r24
    pop r24

    reti

  int_full:

    ldi r23, 0x00
    out PORTC, r23
    pop r23
    pop r24
    out SREG, r24
    pop r24

    reti
```

Ζήτημα 2.2

Στο ζήτημα αυτό μας ζητείται να απεικονίσουμε στο PORTC το πλήθος κουμπιών PB που έχουν πατηθεί όταν γίνεται διακοπή INT0. Παίρνουμε είσοδο το PINB, την αποθηκεύουμε σε καταχωρητή και κάνοντας shift right απομονώνουμε το τελευταίο bit και ελέγχουμε αν είναι 1. Στην περίπτωση που είναι 1 αυξάνουμε έναν καταχωρητή που έχουμε βάλει να μετράει τους άσσους και έπειτα κάνουμε πάλι shift right τον καταχωρητή που έχει την είσοδο αλλιώς κάνουμε σκέτα το shift right. Τέλος θέτοντας το carry ενός τρίτου καταχωρητή ως 1 μέσω του rol γεμίζουμε αυτόν τον τρίτο καταχωρητή με όσους άσσους όσους έχουμε μετρήσει πριν και βγάζουμε το αποτέλεσμα.

```
.include "m328PBdef.inc"

.equ FOSC_MHZ=16

.equ DEL_mS=500

.equ DEL_NU= FOSC_MHZ*DEL_mS

.org 0x0
rjmp reset

.org 0x2
rjmp ISR0

  reset:
        ldi r23,(1<< ISC01)|(1<<ISC00)
        sts EICRA, r23

        ldi r23, (1<<INT0)
        out EIMSK, r23

        sei


main:
   ldi r24, LOW(RAMEND)
   out SPL, r24
   ldi r24, HIGH (RAMEND)
   out SPH, r24
   ser r26
   out DDRC, r26
   clr R26
   out DDRB, R26
```

```
loop1:
    clr r26
loop2:
    out PORTC, r26

    ldi r24, low(16*600)
    ldi r25, high(16*600)
    rcall delay_mS

   inc r26

   cpi r26, 0x1F
   breq loop1
   rjmp loop2

delay_mS:

    ldi r23, 249
loop_inn:
    dec r23
    nop
    brne loop_inn

    sbiw r24, 1
    brne delay_mS

    ret

ISR0:
    push r23
    push r24
    in r24, SREG
    push r24
    in r16, PINB
    com r16
    clr r26
    ldi r25,0x06
loop:
    mov r24,r16
    andi r24,0x01
    cpi r24,0x00
    breq next
    inc r26
next:
    ror r16
    dec r25
    cpi r25,0x00
    breq nextt
```

```
    jmp loop
nextt: ldi r19,0x00
lights:
    cpi r26,0
    breq end
    sec
    rol r19
    dec r26
    rjmp lights
end:
    out PORTC,r19
    ldi r24, low(16*500)
    ldi r25, high(16*500)
    rcall delay_mS
    pop r23
    pop r24
    out SREG, r24
    pop r24

    reti
```

Ζήτημα 2.3

Σε αυτό το ζήτημα μας ζητείται να υλοποιήσουμε ένα σύστημα που να ελέγχει το άναμμα και το σβήσιμο ενός φωτιστικού σώματος. Όταν πατάμε το PD3, δηλαδή ενεργοποιούμε την INT1, θα ανάβει το PB0 για 4 δευτερόλεπτα και αν πατήσουμε το PD3 ενώ είναι ήδη αναμμένο το PB0 να ανάψουν όλα τα λαμπάκια PB για μισό δευτερόλεπτο και μετά να ανάψει το λαμπάκι PB0 για άλλα 4 δευτερόλεπτα. Το θέμα σε αυτή την άσκηση ήταν να μπορεί το σύστημά μας να δεχτεί διακοπή ενώ εκτελεί ήδη μια μια ρουτίνα διακοπής, πιθανότατα ενώ εκτελεί delay, για αυτό και στα κομμάτια της ρουτίνας εξυπηρέτησης όπου μπορεί να γίνει και άλλο interrupt ενεργοποιούμε πάλι τις διακοπές μέσω της εντολής sei (ίδια εντολή και σε assembly και σε c).

2.3 assembly

```
.include "m328PBdef.inc"

.equ FOSC_MHZ=16

.equ DEL_mS=500
```

```
.equ DEL_NU= FOSC_MHZ*DEL_mS ;to interrupt se petaei se shmeia tou kwdika analoga
th timh tou pb0.

.org 0x0
rjmp reset

.org 0x4
rjmp ISR1

  reset:
        ldi r23,(1<< ISC11)|(1<<ISC10)
        sts EICRA, r23

        ldi r23, (1<<INT1)
        out EIMSK, r23

        sei

    ser r26
    out DDRB, r26
    ldi r24, LOW(RAMEND)
   out SPL, r24
   ldi r24, HIGH (RAMEND)
   out SPH, r24
    clt
main:
   rjmp main

ISR1:

    brts all_lights
    rjmp lights_on

all_lights:
    ser r20
    out PORTB, r20
    ldi r24, low(16*500)
    ldi r25, high(16*500)
    rcall delay_mS
    ldi r20, 0x01
    out PORTB, r20
    ldi r24, low(16*3500)
    ldi r25, high(16*3500)
    rcall delay_mS
    clr r20
    out PORTB, r20
    clt
    reti
```

```
lights_on:
    set
    ldi r20, 0x01
    out PORTB, r20
    ldi r24, low(16*4000)
    ldi r25, high(16*4000)
    rcall delay_mS
    clr r20
    out PORTB, r20
    clt
    reti


delay_mS:
    sei
    ldi r23, 249
loop_inn:
    dec r23
    nop
    brne loop_inn

    sbiw r24, 1
    brne delay_mS

    ret
```

2.3 c

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
ISR (INT1_vect){
```

```
    sei();
    if (PORTB==0x01){
        PORTB=0xFF;
        _delay_ms(500);
        PORTB=0x01;
        _delay_ms(3500);
        PORTB=0x00;

    }

    else{
        PORTB=0x01;
        _delay_ms(4000);
        PORTB=0x00;
        }
    reti();
}


int main(void) {
    EICRA=(1<< ISC11)|(1<< ISC10);
    EIMSK=(1<<INT1);
    sei();
    DDRB=0xFF;
    PORTB=0x00;
    while (1) {
}
}
```