Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών
Εργαστήριο Μικροϋπολογιστών

Έκθεση 2ης Εργαστηριακής Άσκησης

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

Ζήτημα 2.1 :

Ο κώδικας τρέχει περιμένει στην main μέχρι να δεχτεί interrupt. Όταν γίνει διακοπή διαχειρίζεται το Σπινθηρισμό με τον εξής κώδικα σύμφωνα με τον διάγραμμα του σχήματος 2.4:

```
ldi r24, (1 << INTF1)              ;EIFR <- 1
out EIFR, r24

ldi r24, low(16*5)                 ;wait 5ms
ldi r25, high(16*5)
rcall delay_ms

in r24, EIFR                ;check if EIFR.1 = 0
lsr r24                            ;if it is continue to subroutine
lsr r24                            ;else go back to ISR1(button was pressed multiple times)
brcs ISR1
```

Θα χρησιμοποιήσουμε την r27 (τον οποίον μηδενίζουμε στην αρχή) για να μετράμε τον αριθμό των interrupt. Για αυτό τον λόγο όταν μπαίνουμε στην ρουτίνα διακοπής δεν θα βάλουμε το περιεχόμενο του r27 στο stack καθώς θέλουμε το περιεχόμενο του να μην χάνεται μεταξύ των διακοπών.

Κώδικας :

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


.org 0x0
clr r27
rjmp reset
.org 0x4
rjmp ISR1


reset:

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

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

```
sei                                     ;enable interrupts


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

clr r30                         ;PORTD as input
out DDRD, r30
ser r30                                 ;PORTC as output
out DDRC, r30

main:
   rjmp main

delay_ms:                       ;delay routine used in ISR1

ldi r23, 249

loop_inn:

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




ISR1:                                   ;INT1`subroutine
   push r24
   push r25
   in r24, SREG
   push r24

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

   ldi r24, low(16*5)                   ;wait 5ms
   ldi r25, high(16*5)
   rcall delay_ms

   in r24, EIFR                 ;check if EIFR.1 = 0
   lsr r24                              ;if it is continue to subroutine
   lsr r24                              ;else go back to ISR1(button was pressed multiple times)
   brcs ISR1

   in r25, PIND                         ;check PIND
   com r25                              ;negative logic
   lsl r25                              ;if we press button PD7,print the same number and stop
counting
   brcs done
   subi r27, -1                 ;else increase r27 by one
   cpi r27, 32                          ;if it reaches 32,clear r27(r27 =0)
   brne done                            ;else continue and print it
   clr r27
done:
   out PORTC, r27                       ;print r27
```

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

Ζήτημα 2.2 :

Ο κώδικας κάνει καθυστερήσεις των 600 ms και αυξάνει το r26 από το 0 εώς το 31. Αποθηκεύουμε στο r24 το περιεχόμενο του PORTB, και το αντιστρέφουμε (λόγω αντίστροφης λογικής). Ύστερα κάνουμε shift ένα ένα τα bit και αποθηκεύουμε ποσά 1 υπήρχαν στο r26. Μετά, βάζουμε τόσα 1 στο r27 (ξεκινώντας απο το lsb) σύμφωνα με το r26 και το εμφανίζουμε στο PORTC για μισό δευτερόλεπτο και έπειτα συνεχίζεται η κανονική λειτουργιά του μετρητή από τον αριθμό που είχε μείνει.

Κώδικας :

```
.include "m328PBdef.inc"
.equ FOSC_MHZ=16            ;operating freqency of microcontroller
.equ DEL_ms=600                     ;delay in ms(600 ms for this exercise)
.equ Del_NU=FOSC_MHZ*DEL_ms   ;input for delay_ms routine

.org 0x0
rjmp reset
.org 0x2
rjmp ISR0

reset:

ldi r24,(1 << ISC01) | (1 << ISC00)          ;interrupt on rising edge of INT0 pin
sts EICRA, r24

ldi r24, (1 << INT0)                    ;enable INT0 interrupt = PD2
out EIMSK, r24
sei                                     ;enable interrupts



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

```
clr r26                                 ;PORTB as input
out DDRB, r26
ser r26                                         ;PORTC as output
out DDRC, r26

loop1:
clr r26

loop2:
out PORTC, r26                  ;print number

ldi r24, low(DEL_NU)
ldi r25, high(DEL_NU)           ;set delay(number of cycles)
rcall delay_ms
inc r26                         ;increase r26 and check if it is 32
cpi r26, 32
breq loop1                      ;if it is 32 clear it
rjmp loop2                      ;else move on to the next number



delay_ms:               ;total delay of 1000*DEL_NU + 6 cycles

ldi r23, 249            ;1 cycle


loop_inn:               ;total delay(1 + 249*4 - 1 = 996 cycles)

dec r23                         ;1 cycle
nop                     ;1 cycle
brne loop_inn           ;1 or 2 cycles
sbiw r24, 1             ;2 cycles
brne delay_ms                   ;1 or 2 cycles
ret                     ;4 cycles


ISR0:               ;INT0 subroutine
push r26
push r24
push r25


ldi r27, 1      ;register for interrupt output
clr r26         ;register where we store number of buttons pressed
ldi r25, 7      ;number of loops needed(7-1=6 to check all the buttons)

in r24, PINB    ;Buttons pressed
com r24                 ;complementary due to negative logic

loop:
dec r25                   ;decrease number of loops left
cpi r25, 0
breq number    ;after 6 loops we have the number in r26
lsr r24            ;else rotate input register right and check carry
brcc loop          ;if carry is 0 loop again
inc r26            ;else if carry is 1 increase r26 as we found button pressed
rjmp loop          ;and loop again
```

```
number:

cpi r26, 0      ;if r26 = 0,we've pressed 0 buttons and we clear output register
breq zero
cpi r26, 1      ;if r26 = 1,1 button pressed and we print r27 which has initial value 1
breq done
lsl r27         ;else we open one more led from lsb to msb by shifting left and
subi r27, -1    ;adding 1(we move all the leds left once and open the LSB by
dec r26                 ; adding 1)
rjmp number  ;and loop again


zero:
    clr r27
done:
out PORTC, r27          ;r27 output


pop r25                 ;restore stack
pop r24
pop r26



push r25        ;and wait extra 500 ms
push r24
in r24, SREG
push r24



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


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

Ζήτημα 2.3 :

Περιγραφή υλοποίησης σε Assembly :

Ο r27 καθορίζει τον κώδικα που θα τρέξει το πρόγραμμα στην main.

Όσο το r27 = 00000000, η main θα κάνει συνέχει loop με κλειστά τα λαμπάκια.

Όταν κάνουμε ένα interrupt θα αλλάξουμε την κατάσταση του r27 με τον εξής τρόπο: Θα κάνουμε το πρώτο bit του r27 1 και αν υπήρχε λαμπάκι ανοικτό θα κάνουμε και το δεύτερο bit του r27 1.

Επιστρέφοντας στην main με την **rjmp main αντί για reti** (έχοντας ενεργοποιήσει τα global interrupt ξανά πρώτα) θα τσεκάρουμε τα δυο πρώτα bit του r27. Αν r27 = 00000001 θα ανάψουμε το πρώτο λαμπάκι για 4 δεύτερα, ενώ αν το r27 = 00000011, θα ανάψουμε όλα τα λαμπάκια για 0.5 δεύτερα ενώ το πρώτο θα παραμείνει ανοιχτό για άλλα 3.5 δεύτερα.

Σε κάθε περίπτωση αν περάσουν 4 δευτερόλεπτα και δεν υπάρξουν interrupt θα κλείσουμε τα λαμπάκι και θα κάνουμε το r27 = 0 και θα συνεχίζουμε να περιμένουμε για το επόμενο interrupt.

Αν εντός των 4 δευτερολέπτων υπάρξει interrupt θα ενημερώσουμε το r27 εντός της ρουτίνας κάνοντας τα δυο πρώτα bit ίσα με 1.

Για κάθε interrupt εξασφαλίζουμε την ορθή ενημέρωση καθώς όταν βγαίνουμε από την ρουτίνα δεν επιστρέφουμε στο σημείο του κώδικα όπου έγινε κλήση της interrupt άλλα πηγαίνουμε στην αρχή της main με την **rjmp main.**

Κώδικας 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

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
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
ser r30
out DDRB, r30

main:
   lsr r27          ;if LSB of r27 is 0 then do nothing
   brcc again
   lsr r27                  ;else check second LSB
   brcs refresh             ;if it is 1 then we need to open all leds for 500ms
```

```
    ldi r28, 0x01            ;else we need to open LSB led of portB for 4s
    out PORTB, r28           ;r28 = 1 as output
    ldi r24, low(16*4000)   ;delay 4s
    ldi r25, high(16*4000)
    rcall delay_ms
    ldi r28, 0x00            ;close LEDS if 4s passed
    out PORTB, r28
    rjmp again               ;clear r27 and continue

    refresh:
    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
    ldi r28, 1               ;now open LSB led for 3.5s more
    out PORTB, r28
    ldi r24, low(16*3500)   ;delay 3.5s
    ldi r25, high(16*3500)
    rcall delay_ms
    ldi r28, 0x00            ;close all leds if 3.5ms passed and continue
    out PORTB, r28



    again:
      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 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
```

Περιγραφή υλοποίησης σε C :

Στο πρόγραμμα αυτό θα περιμένουμε στην main και θα διαχειριστούμε την κατάσταση των led αποκλειστικά μέσα στην interrupt.

Συγκεκριμένα, μέσα στην ρουτίνα διακοπής θα ανοίγουμε το πρώτο led και θα τσεκάρουμε κάθε 1ms αν κάποιος έχει πατήσει το PD3. Η διαδικασία αυτή θα επαναληφθεί 4000 φορές, σύνολο 4000ms (= 1s) όπου μετά θα ξαναπάμε στην main. Αντί να επιτρέπουμε τις διακοπές μέσα στην διακοπή, θα βλέπουμε αν υπάρχει νέο αίτημα διακοπής κάθε ms. Αν υπάρξει θα ανάψουμε όλα τα λαμπάκια για μισό δευτερόλεπτο και θα ανανεώσουμε την λούπα των 4000 φορών, ώστε να μείνουν άλλες 3500 επαναλήψεις όπου μόνο το 1 ένα λαμπάκι είναι ανοικτό. Όσο όλα τα λαμπάκια είναι ανοιχτά θα συνεχίζουμε να κοιτάμε κάθε ms αν έχει υπάρξει και άλλη διακοπή και απλά για κάθε ερχόμενη διακοπή, θα ξεκινάμε από την αρχή να μετράμε τα 500 ms όπου όλα τα λαμπάκια θα είναι ανοιχτά.

Να σημειωθεί ότι στην εξής υλοποίηση τα interrupt εκτελούνται με την λογική όταν υπάρχει λογικό 1 στο pd3 και όχι όταν εντοπίσουμε θετικό μοναδιαίο βήμα σε αντίθεση με την υλοποίηση της assembly.

Λόγω του ότι όλα τα interrupt διαχειρίζονται εντός μιας ρουτίνας, οπότε στο τέλος της θα μηδενίσουμε το EIFR.1 για να μην γίνει και άλλη κλήση ρουτίνας αμέσως μετά.

Κώδικας C :

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

ISR(INT1_vect)
{
for(int i = 0; i < 4000; i++){              //for checking every 1 ms if we have new interrupt
      if(PIND & 0x08 == 1){                 //if new interrupt is detected
```

```
          for(int j = 0; j < 500; j++){       //open all LEDS for 500 ms while checking for new
interrupts
          if(PIND & 0x08 == 1) j=0;          //for every new interrupt we detect, j = 0 (we print all
leds for 500 ms again)
          PORTB = 0xFF;
          _delay_ms(1);
         }
          PORTB = 0x01;                    //after all leds go out
          i = 499;                        //i = 499 and we have 3500 more loops to print LSB LED for
3500 ms
          continue;
       }
       else {
          PORTB = 0x01;                    //print LSB
          _delay_ms(1);
        }
   }
   PORTB = 0x00;              //close leds
   EIFR = (1 << INTF0);       //EIFR.1 = 0

}


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

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

   while(1){
      sei();             //enable interrupts
   }
}
```