**Εθνικό Μετσόβιο Πολυτεχνείο**
**Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών**

Μ Α Θ Η Μ Α
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ (ΡΟΗ Υ)

Ε Ρ Γ Α Σ Ι Α
2Η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Σ Π Ο Υ Δ Α Σ Τ Ε Σ
ΚΑΡΑΜ ΚΩΝΣΤΑΝΤΙΝΟΣ
ΚΟΛΙΟΣ ΑΠΟΣΤΟΛΟΣ

2022–23

**Ζήτημα 2.1** *(Lab2_1.asm)*

Ο κώδικας του προγράμματος είναι ο εξής:

```
.include "m328PBdef.inc" ; ATmega328P microcontroller definitions

.equ FOSC_MHZ=16 ; Microcontroller operating frequency in MHZ
.equ DEL_mS=500 ; Delay in mS (valid number from 1 to 4095)
.equ DEL_NU=FOSC_MHZ*DEL_mS ; delay mS routine: (1000* DEL NU+6) cycles

; INTERRUPT INT1
.org 0x0
rjmp reset
.org 0x4
rjmp ISR1

reset:
; Interupt on rising edge of INT1 pin
ldi r24,(1<<ISC11) | (1<< ISC10)
sts EICRA, r24
; Enable the INT1 interrupt (PD3)
ldi r24, (1 << INT1)
out EIMSK, r24

sei ; Sets the Global Interrupt Flag

; Init Stack Pointer
ldi r24, LOW (RAMEND)
out SPL, r24
```

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

; Init PORTB and PORTC as output
ser r26
out DDRB, r26
out DDRC, r26

; Init PORTD as input
clr r27
out DDRD,r27

ldi r20,0

loop1:
clr r26

loop2:
out PORTB, r26

ldi r24, low (DEL_NU)
ldi r25, high (DEL_NU) ; Set delay (number of cycles)
rcall delay_mS

inc r26

cpi r26, 16 ; compare r26 with 16
breq loop1
rjmp loop2


;delay of 1000* F1+6 cycles (almost equal to 1000* Fl cycles)
delay_mS:

; total delay of next 4 insruction group = 1+ (249*4-1) 996 cycles
ldi r23, 249 ; (1 cycle)

loop_inn:

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
```

```
ISR1: ; Interupt Routine
push r25
push r24
in r24, SREG        ; Save r24, r25, SREG
push r24

INT_1:
ldi r24,(1<<INTF1)
out EIFR,r24
ldi r24,low(80)
ldi r25,high(80) ; Set delay (number of cycles)
rcall delay_mS
in r24,EIFR
sbrc r24,1
rjmp INT_1

in r23,PIND
andi r23,0x80       ; Mask PD7
cpi r23,0x00        ; 0=pathmeno
breq exodos

cpi r20,0x1F ;31 dec
brne cont
ldi r20, 0x00
cont:
ldi r21, 0x01
add r20, r21
out PORTC,r20

exodos:
pop r24
out SREG ,r24 ; Restore r24, r25, SREG
pop r24
pop r25
reti
```

Θα εξηγήσουμε απλά την λειτουργία του προγράμματος. Στο πρόγραμμα του μετρητή μας ζητήθηκε να προστεθεί μία ρουτίνα εξυπηρέτησης διακοπής INT1, η οποία θα μετρά το πλήθος των διακοπών INT1 από 0 έως 31 και θα το απεικονίζει σε δυαδική μορφή στα Led PC4 με PC0.
Όταν πατηθεί το κουμπί PD3 εκτελούμε την ρουτίνα εξυπηρέτησης της διακοπής INT1. Αρχικά για την καταπολέμηση του φαινομένου του σπινθηρισμού, μηδενίζουμε (θέτουμε ίσο με 1) το ψηφίο INTF1 και αφού εφαρμόσουμε μία μικρή καθυστέρηση (5 msec), ελέγχουμε ξανά το ψηφίο, έως ότου αυτό σταθεροποιηθεί στο λογικό 0, οπότε έχουν σταματήσει οι αναπηδήσεις.

Ο αριθμός των συνολικών διακοπών αποθηκεύεται στον καταχωρητή r20, ο οποίος ελέγχεται σε κάθε διακοπή πριν αυξηθεί εάν είναι ίσος με το 31. Αν όχι αυξάνεται, διαφορετικά μηδενίζεται για να ξεκινήσει η μέτρηση από την αρχή.
Πριν γίνει η αύξηση του παραπάνω μετρητή των διακοπών, ελέγχεται μέσα στη ρουτίνα εξυπηρέτησης της διακοπής εάν είναι πατημένο το κουμπί PD7. Εάν είναι πατημένο τότε η ρουτίνα τερματίζει διαφορετικά συνεχίζει την μέτρηση των διακοπών.

**Ζήτημα 2.2** *(Lab2_2.asm)*

Ο κώδικας του προγράμματος είναι ο εξής:

```
.include "m328PBdef.inc" ; ATmega328P microcontroller definitions

.equ FOSC_MHZ=16 ; Microcontroller operating frequency in MHZ
.equ DEL_mS=600 ; Delay in mS (valid number from 1 to 4095)
.equ DEL_NU=FOSC_MHZ*DEL_mS ; delay mS routine: (1000* DEL NU+6) cycles

; INTERRUPT INT1
.org 0x0
rjmp reset
.org 0x2
rjmp ISR0

reset:
; Interupt on rising edge of INT0 pin
ldi r24,(1<<ISC01) | (1<< ISC00)
sts EICRA, r24
; Enable the INT0 interrupt (PD2)
ldi r24, (1 << INT0)
out EIMSK, r24

sei ; Sets the Global Interrupt Flag

; Init Stack Pointer
ldi r24, LOW (RAMEND)
out SPL, r24
ldi r24, HIGH (RAMEND)
out SPH, r24

; Init PORTC as output
ser r26
out DDRC, r26

; Init PORTB as input
clr r27
```

```
out DDRB,r27

ldi r20,0   ;input from PORTD
ldi r19,0x01    ;r19=1
ldi r21,0   ;counter of zeros
ldi r18, 0x00   ; LED output

loop1:
clr r26

loop2:
out PORTC, r26

ldi r24, low (DEL_NU)
ldi r25, high (DEL_NU) ; Set delay (number of cycles)
rcall delay_mS

inc r26

cpi r26, 32 ; compare r26 with 32
breq loop1
rjmp loop2


;delay of 1000* F1+6 cycles (almost equal to 1000* Fl cycles)
delay_mS:

; total delay of next 4 insruction group = 1+ (249*4-1) 996 cycles
ldi r23, 249 ; (1 cycle)

loop_inn:

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: ; Interupt Routine
push r25
push r24
in r24, SREG        ; Save r24, r25, SREG
push r24

INT_1:
```

```
ldi r24,(1<<INTF1)
out EIFR,r24
ldi r24,low(80)
ldi r25,high(80) ; Set delay (number of cycles)
rcall delay_mS
in r24,EIFR
sbrc r24,1
rjmp INT_1

;1st iteration
in r20, PINB
ror r20     ; C flag <- LSB
brcs next_1 ;if C = 1 then jump to next_1
add r21, r19

;2nd
next_1:
ror r20     ; C flag <- LSB
brcs next_2 ;if C = 1 then jump to next_1
add r21, r19

;3d
next_2:
ror r20     ; C flag <- LSB
brcs next_3 ;if C = 1 then jump to next_1
add r21, r19

;4th
next_3:
ror r20     ; C flag <- LSB
brcs next_4 ;if C = 1 then jump to next_1
add r21, r19

;5th
next_4:
ror r20     ; C flag <- LSB
brcs next_5 ;if C = 1 then jump to next_1
add r21, r19

;6th
next_5:
ror r20     ; C flag <- LSB
brcs next_6 ;if C = 1 then jump to next_1
add r21, r19

next_6:
```

```
ldi r18, 0x00

check:
cpi r21, 0x00       ;cpi zeros counter to 0
breq fin
add r18, r19
rol r18
subi r21,1
rjmp check

fin:
ldi r21, 0x00
ror r18
out PORTC, r18
ldi r24,low(8000)
ldi r25,high(8000)
rcall delay_mS  ; show how many are pressed for 0.5 sec
pop r24
out SREG ,r24 ; Restore r24, r25, SREG
pop r24
pop r25
reti
```

Οι μικρές αλλαγές που ζητήθηκαν σε σχέση με το πρόγραμμα του σχήματος 2.1, έγιναν με τις τροποποιήσεις στις γραμμές 4 (600 msec καθυστέρηση), 54 (μέτρηση έως το 32) και 46 (εμφάνιση αποτελέσματος μέτρησης στην PORTC).
Ορίζουμε την ρουτίνα εξυπηρέτησης για την διακοπή INT0 στη γραμμή 74 και χρησιμοποιούμε την μέθοδο για την αποφυγή του σπινθηρισμού όπως και στο προηγούμενο ερώτημα. Με το που κληθεί η ρουτίνα εξυπηρέτησης της διακοπής, διαβάζουμε τη θύρα PORTB και αποθηκεύουμε την τιμή στον καταχωρητή r20. Έπειτα κάνουμε ror το περιεχόμενο του καταχωρητή, μετακινώντας με αυτόν τον τρόπο στο C flag του καταχωρητή το LSB του. Έπειτα ελέγχουμε με την εντολή brcs εάν το C flag είναι 1, που σημαίνει ότι το κουμπί δεν είναι πατημένο και άρα προχωράμε στην περιστροφή του επόμενου ψηφίου της εισόδου. Εάν κάποιο C flag είναι ίσο με 0 (το κουμπί είναι πατημένο), τότε αυξάνουμε τον μετρητή των μηδενικών (πατημένων κουμπιών) r21 κατά 1 (με την βοήθεια του r19 που ισούται με τη μονάδα). Αφού ελέγξουμε και τα έξι κουμπιά της εισόδου, με τη βοήθεια του καταχωρητή r18 προσθέτουμε τόσες φορές όσες είναι και τα μηδενικά (r21) έναν άσσο στο LSB του, και τον περιστρέφουμε με την rol μία θέση αριστερά κάθε φορά. Τέλος, ακριβώς πριν εμφανίσουμε το περιεχόμενο του καταχωρητή r18 στην έξοδο PORTC, τον περιστρέφουμε μία φορά δεξιά μέσω της ror, λόγω της τελευταίας αριστερής περιστροφής του. Εμφανίζουν το επιθυμητό αποτέλεσμα για μισό δευτερόλεπτο και έπειτα το πρόγραμμα συνεχίζει τη λειτουργία του.

**Ζήτημα 2.3** *(Lab2_3.asm και Lab2_3.c)*

Ο κώδικας του προγράμματος σε Assembly είναι ο εξής:

```
.include "m328PBdef.inc" ; ATmega328P microcontroller definitions

.equ FOSC_MHZ=16 ; Microcontroller operating frequency in MHZ
.equ DEL_mS=4000 ; Delay in mS (valid number from 1 to 4095)
.equ DEL_NU=FOSC_MHZ*DEL_mS ; delay mS routine: (1000* DEL NU+6) cycles

; INTERRUPT INT1
.org 0x0
rjmp reset
.org 0x4
rjmp ISR1

reset:
; Interupt on rising edge of INT1 pin
ldi r24,(1<<ISC11) | (1<< ISC10)
sts EICRA, r24
; Enable the INT1 interrupt (PD3)
ldi r24, (1 << INT1)
out EIMSK, r24

sei ; Sets the Global Interrupt Flag

; Init Stack Pointer
ldi r24, LOW (RAMEND)
out SPL, r24
ldi r24, HIGH (RAMEND)
out SPH, r24

; Init PORTB and PORTC as output
ser r26
out DDRB, r26


; Init PORTD as input
clr r27
out DDRD,r27

main:
ldi r18,0x00        ;cnt
ldi r17,0x01    ;assos
rjmp main
```

```
;delay of 1000* F1+6 cycles (almost equal to 1000* Fl cycles)
delay_mS:

; total delay of next 4 insruction group = 1+ (249*4-1) 996 cycles
ldi r23, 249 ; (1 cycle)

loop_inn:
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

ISR1: ; Interupt Routine
push r25
push r24
in r24, SREG        ; Save r24, r25, SREG
push r24

INT_1:
ldi r24,(1<<INTF1)
out EIFR,r24
ldi r24,low(80)
ldi r25,high(80) ; Set delay (number of cycles)
rcall delay_mS
in r24,EIFR
sbrc r24,1
rjmp INT_1

sei         ;epitrepoume na ginei diakoph panw se diakoph

add r18,r17
cpi r18,0x01        ;ama einai 0 h afairesh tote mpainw prwth fora ara sto
bre tha kalesw 4s delay alliws tha kalesw 0.5
brne cont1

cont:       ;4 sec
ldi r19,0x01
out PORTB,r19
ldi r24, low (DEL_NU)
ldi r25, high (DEL_NU) ; Set delay (number of cycles)
rcall delay_mS
ldi r19,0x00
out PORTB,r19
ldi r18,0x00
```

```
jmp cont2

cont1:      ;0.5 sec
ldi r19,0xFF
out PORTB,r19
ldi r24, low (8000)
ldi r25, high (8000) ; Set delay (number of cycles)
rcall delay_mS
ldi r19,0x01
out PORTB,r19
ldi r24, low (DEL_NU)
ldi r25, high (DEL_NU) ; Set delay (number of cycles)
rcall delay_mS
ldi r19,0x00
out PORTB,r19
ldi r18,0x00

cont2:
pop r24
out SREG ,r24 ; Restore r24, r25, SREG
pop r24
pop r25
reti
```

Ο κώδικας του προγράμματος σε γλώσσα C είναι ο εξής:

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

int flag = 0;

ISR (INT1_vect) //External INT1 ISR
{
    flag++;
    sei();

    if (flag>1){
    PORTB = 0xFF;
    _delay_ms (500) ;
    PORTB = 0x01;
    _delay_ms (4000) ;
    PORTB= 0x00;
    flag = 0;
```

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

int main()
{
    // Interrupt on rising edge of INT0 and INT1 pin
    EICRA = (1 << ISC11) | ( 1 << ISC10);
    //Enable the INT0 interrupt (PD2), INT1 interrupt (PD3)
    EIMSK = (1 << INT1);
    sei(); // Enable global interrupts

    DDRB=0xFF; //Set PORTB as output
    PORTB=0x00;

    while(1)
    {
    PORTB = 0x00;
    flag = 0;
    }
}
```

Θα εξηγήσουμε απλά την λειτουργία του προγράμματος. Αρχικά το πρόγραμμα αρχικοποίηση τον μετρητή r18 στο μηδέν και περιμένει έως ότου πατηθεί κάποια διακοπή. Όταν πατηθεί το κουμπί PD3 εφαρμόζουμε μία μικρή καθυστέρηση για τον σπινθηρισμό, όπως και στα παραπάνω προβλήματα. Την πρώτη φορά που θα πατηθεί το κουμπί θα αυξηθεί κατά ένα ο καταχωρητής r18 έτσι ώστε εάν ξανά προκληθεί διακοπή, να κάνει άλμα και να εκτελέσει την ρουτίνα cont1 η οποία ανάβει για μισό δευτερόλεπτο όλα τα Led και έπειτα ξανά ανανεώνει τον χρόνο των τεσσάρων δευτερολέπτων. Μόλις τα 4 δευτερόλεπτα τελειώσουν, τα λαμπάκια σβήνουν και ο καταχωρητής r18 ξαναμηδενίζεται. Στην υλοποίηση σε γλώσσα C, αντί του καταχωρητή r18 χρησιμοποιήθηκε η μεταβλητή flag.