Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εργαστήριο Μικροϋπολογιστών

7ο εξάμηνο, Ακαδημαϊκή περίοδος 2025-2026

2η Εργαστηριακή Άσκηση

Χρήση εξωτερικών διακοπών στον Μικροελεγκτή AVR

Εργαστηριακή Αναφορά

Φοιτητές: Παπαδάτος Αναστάσιος -> **ΑΜ:** 03122847
Σέρτζιο Γκούρι -> **ΑΜ:**03122827

Ομάδα: 30

Ημερομηνία: 28/10/2025

Άσκηση 1:

Παρακάτω φαίνεται ο κώδικας assembly που φτιάξαμε:

```
.include "m328PBdef.inc"

;Constant variables that assembler reads first
.equ FOSC_MHZ=16                        ;Microcontroller op frequency
.equ DEL_mS=500                         ;Delay in mS(only valid for 1 to 4095)
.equ DEL_miliS=5                        ;Delay in MS
.equ DEL_NU=FOSC_MHZ*DEL_mS             ;delay_mS routine: (1000*DEL_NU + 6)cycles the
                                        extra 6 cycles dont bother us
.equ DEL_nanoU=FOSC_MHZ*DEL_miliS       ;delay_MS routine:

;Initialize the vector table stored in FLASH
.org 0x0
rjmp reset
.org 0x4
rjmp ISR1

.def intr_counter=r16
.def pause_check=r17

reset:
;Init stack pointer
       ldi r24, LOW(RAMEND)
       out SPL,r24
       ldi r24, HIGH(RAMEND)
       out SPH,r24

       ldi intr_counter, 0              ;Initialize interrupt counter to 0

;Init of PORTD as read only
       clr r18                          ;Set r18 bits to 0
       out DDRD, r18                    ;Set portD as read only
       ser r18                          ;set r18 bits to 1
       out PORTD, r18                   ;Pull-up resistors so we dont have floating pins

;Init of PORTB as output only
       out DDRC, r18                    ;set portC as output only

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

;Enable the INT1 interrupt(PD3)
       ldi r24, (1 << INT1)
       out EIMSK, r24
       sei                              ;Set the Global interrupt Flag

main:
       in r18, PIND                     ;read portD
       lsr r18                          ;logical shift right to check 1st bit
       lsr r18
       brcc wait                        ;If carry flag is 0 then pd1 is pressed so wait
       sei
       rjmp main
```

```
wait:
	cli
	rjmp main

ISR1:
;set INTF1 bit to 0
intf1_check:
	ldi r21, (1<<INTF1)		; set intf1 to 0 so we can check
	out EIFR,r21			; Set the bit intf1 to 0
	ldi r24, low(DEL_nU)
	ldi r25, high(DEL_nU)		;set delay
	rcall delay_MS

;check INTF1 bit
	in r21,EIFR
	lsr r21
	lsr r21
	brcs intf1_check

	lsl intr_counter		;shift intr_counter 1 place to light-up PC1-PC5
	out PORTC, intr_counter

	lsr intr_counter		;place intr_counter to first position
	inc intr_counter		;Intrerrupt occured so increase int_counter
	mov r20, intr_counter
	subi r20, 31			;subtract 31
	breq before_reset		; if carry flag is 0 then jump to reset
	reti

before_reset:
	ldi intr_counter, 0		;Initialize interrupt counter to 0
	reti

;delay of 1000F1_6 cycles
delay_mS:
;total delay of next 4 instruction group= 1+(249*4-1)= 996 cycles
	ldi r23, 249
loop_inn:
	dec r23
	nop
	brne loop_inn
	sbiw r24,1
	brne delay_mS
	ret

;delay of 1000F1_6 cycles
delay_miliS:
;total delay of next 4 instruction group= 1+(249*4-1)= 996 cycles
	ldi r23, 249
loop_in:
	dec r23
	nop
	brne loop_in
	sbiw r26,1
	brne delay_miliS
	ret
```

Σχόλια/Περιγραφή του κώδικα:

Στον παραπάνω κώδικα η διαδικασία με την οποία υλοποιούμε τον κώδικα είναι απλή. Όσο είμαστε στην main ελέγχουμε την συνθήκη για το μπουτόν **PD1**. Για να το κάνουμε αυτό χρησιμοποιούμε τον καταχωρητή **r18** για να διαβάζει τα την κατάσταση των **PIND**. Φορτώνουμε την τιμή του **I/0 καταχωρητή PIND** στον **r18**, κάνουμε shift right των καταχωρητή **r18** 2 φορές για να ελέγξουμε την τιμή του bit **PD1**. Επειδή το πάτημα των κουμπιών είναι αρνητικής λογικής, σε περίπτωση που το carry flag είναι 0 τότε πηγαίνουμε στην wait και απενεργοποιούμε τις διακοπές. Σε διαφορετική περίπτωση ξαναπηγαίνουμε στην main για να ελέγχουμε διαρκώς αν ικανοποιείται η παραπάνω συνθήκη. Μόλις πατηθεί το μπουτόν **PD3** προκαλείται διακοπή. Μέσα στην διακοπή προσπαθούμε να απαλείψουμε τον σπινθηρισμό με την συνάρτηση **intf1_check**. Στην αρχή, μόλις μπούμε στην ρουτίνα διακοπής, θέτουμε στον καταχωρητή **r21** την τιμή 1, στην θέση στην οποία βρίσκεται το bit του **INTF1**. Βάζουμε την τιμή 1 γιατί έτσι μηδενίζεται το bit του καταχωρητή σε αυτή τη θέση. Έπειτα αντιγράφουμε την τιμή του καταχωρητή **r21** στον καταχωρητή **EIFR** και καλούμε μια delay για να τσεκάρουμε αν έχει μεταβεί το bit **INTF1** από 0 σε 1. Για να το κάνουμε αυτό φορτώνουμε τον **EIFR** στον **r21** και τον κάνουμε shift 2 θέσεις δεξιά. Αν το carry flag έχει γίνει 1 τότε έχουμε σπινθηρισμό και πρέπει να ξαναμηδενίσουμε τον **INTF1** αλλιώς δεν έχουμε σπινθηρισμό και μεταβαίνουμε στην συνάρτηση κανονικά. Μέσα στην κύρια συνάρτηση χρησιμοποιούμε τον **intr_counter** για να μετρήσουμε το πλήθος των διακοπών. Έχουμε αρχικοποιήσει τον συγκεκριμένο καταχωρητή στην τιμή 0, οπότε θα μετρήσουμε απ το 0-31. Αρχικά τον κάνουμε shift left για να φωτίσουμε τα bits **PC1-PC5** και ύστερα φωτίζουμε τα bits 0-31. Έπειτα κάνουμε shift right για να φέρουμε τον καταχωρητή στην αρχική θέση, τον αυξάνουμε κατά 1, τον συγκρίνουμε με την τιμή 31 και σε περίπτωση που το carry flag είναι 0, κάνουμε reset τον **intr_counter** στο 0 και ξανακάνουμε την παραπάνω διαδικασία.

Άσκηση 2:

Α) Παρακάτω φαίνεται ο κώδικας assembly που φτιάξαμε:

```
.include "m328PBdef.inc"                ;Atmega328PB microcontroller definitions

.equ FOSC_MHZ=16                        ;Microcontroller frequency- 16MHz
.equ DEL_mS=1000                        ;Delay in mS
.equ DEL_NU=FOSC_MHZ*DEL_mS             ;delay_mS routine

;Init Stack Pointer
       ldi r23, LOW(RAMEND)
       out SPL, r24
       ldi r24, HIGH(RAMEND)
       out SPH, r24

; Init PORTC as output
       ser r26
```

```
        out DDRC, r26

loop1:
        clr r26
loop2:
        lsl r26
        out PORTC, r26
        lsr r26
        ldi r24, low(DEL_NU)
        ldi r25, high(DEL_NU)             ;set delay
        rcall delay_mS

        inc r26
        cpi r26, 31                       ;compare with 31
        breq loop1
        rjmp loop2

;delay of 1000*f1+6 cycles
delay_mS:
;total delay of next 4 instruction group= 1+ (249*4-1)=996 cycles
        ldi r23, 249
loop_inn:
        dec r23
        nop
        brne loop_inn
        sbiw r24,1
        brne delay_mS
        ret
```

Σχόλια/Περιγραφή του κώδικα:

Ο παραπάνω κώδικας έχει πιο απλή υλοποίηση και είναι μια προσπάθεια μέτρησης απ το 0-31 και ο φωτισμός τους στο **PORTC**. Για αρχή θέτουμε τον καταχωρητή **r26** στη τιμή 0 και έπειτα τον κάνουμε shift left για να φωτίσουμε τα bits **PC1-PC5**. Στη συνέχεια τον κάνουμε shift right για να τον φέρουμε στην αρχική θέση, καλούμε μια delay και έπειτα αυξάνουμε τον **r26** κατά 1. Ελέγχουμε αν έχει ξεπεράσει την τιμή 31 και σε περίπτωση που έχει, ξεκινάει απ την αρχή κάνοντας reset τον καταχωρητή **r26** σε 0, αλλιώς ακολουθάει την διαδικασία απ την οποία κάνει shift left και φωτίζει τα bits **PC1-PC5.** Η περίπτωση με το reset δίνεται απ την συνάρτηση **loop1** ενώ η περίπτωση φωτισμού των LEDs δίνεται τη συνάρτηση **loop2**.

Β) Παρακάτω φαίνεται ο κώδικας assembly που φτιάξαμε:

```
.include "m328PBdef.inc"                  ;Atmega328PB microcontroller definitions

.equ FOSC_MHZ=16                          ;Microcontroller frequency- 16MHz
.equ DEL_mS=2000                          ;Delay in mS
.equ DEL_NU=FOSC_MHZ*DEL_mS               ;delay_mS routine

.org 0x0
```

```
rjmp reset
.org 0x2
rjmp ISR0

reset:
;Init Stack Pointer
	ldi r23, LOW(RAMEND)
	out SPL, r24
	ldi r24, HIGH(RAMEND)
	out SPH, r24

;Init of PORTB as read only
	clr r26				;Set r26 bits to 0
	out DDRB, r26			;Set portD as read only
	ser r26				;set r26 bits to 1
	out PORTD, r26			;Pull-up resistors so we dont have floating pins

;Init PORTC as output
	ser r26
	out DDRC, r26

;Interrupt on rising edge of INT1 pin
	ldi r24,(1 << ISC01) | ( 1 << ISC00)
	sts EICRA, r24

;Enable the INT0 interrupt(PD2)
	ldi r24, (1 << INT0)
	out EIMSK, r24
	sei				;Set the Global interrupt Flag

main:
	rjmp main

ISR0:
	ldi r16,0
	ldi r25,4
	in r24,PINB			;r24 reads portb
	com r24				;leds are negative logic
	andi r24,0x1E			;mask r24 so we can read the right pins
	lsr    r24			;shift right to bring PB4-PB1 to PB3-PB0

check:
	lsr    r24			;check for PB1,PB2,PB3,PB4
	brcs counter			;if c=1 then its pressed so go to counter

decrease_function:
	dec r25
	brne check		;if r25 is not 0 go to check else you have finished
	out PORTC, r16			;light up bits
	reti

counter:
	lsl r16				;shift left by 1
	ori r16, 0x01			;enable that pin
	rjmp decrease_function

;delay of 1000*f1+6 cycles
delay_mS:
```

```
;total delay of next 4 instruction group= 1+ (249*4-1)=996 cycles
       ldi r23, 249
loop_inn:
       dec r23
       nop
       brne loop_inn
       sbiw r24,1
       brne delay_mS
       ret
```

Σχόλια/Περιγραφή του κώδικα:

Στον παραπάνω κώδικα προσπαθούμε μέσα στη ρουτίνα εξυπηρέτησης διακοπής να ανάψουμε ένα LED απ το **PORTC**, ξεκινώντας απ το LSB, όσο το πλήθος των μπουτόν **PB1-PB4** που είναι πατημένα εκείνη τη στιγμή. Για να το κάνουμε αυτό ξεκινάμε αρχικοποιόντας 2 καταχωρητές **r16** και **r25** στις τιμές 0 και 4 αντίστοιχα. Ξεκινάμε απ το 0 με τον **r16** για να μετρήσουμε το πλήθος των πατημένων μπουτόν και ξεκινάμε απ το 4 με τον **r25** για να τσεκάρουμε και για τα 4 μπουτόν, να έχουμε δηλαδή 4 επαναλήψεις. Αρχίζουμε λοιπόν και χρησιμοποιούμε τον καταχωρητή **r24** για διαβάσει το **PORTB**, το φέρνουμε σε θετική λογική και έπειτα κάνουμε **mask** με την τιμή **0x1E** για να ελέγξουμε τα bit **PB1-PB4**. Στη συνέχεια κάνουμε shift δεξιά για να φέρουμε τους **PB1-PB4** στην θέση **PB0-PB3** και μετά αρχίζουμε τον έλεγχο. Ο έλεγχος αρχίζει με τη συνάρτηση **check** μέσα στην οποία κάνουμε κι άλλο shift right για να δούμε την τιμή του carry flag. Άμα το carry flag είναι 1, το μπουτόν είναι πατημένο οπότε πάμε στην συνάρτηση **counter**, κάνουμε shift τον output καταχωρητή μας μια θέση αριστερά και κάνουμε την λογική πράξη **ori 0x01**. Στη συνέχεια πηγαίνουμε στην συνάρτηση **decrease_function** για να μειώσουμε τον μετρητή των επαναλήψεων **r25**. Αν αυτός ο μετρητής έχει γίνει 0 τότε βγάζουμε στο **PORTC** την τιμή του **r16**, αλλιώς τσεκάρουμε το επόμενο bit πηγαίνοντας στην συνάρτηση **check**.

Άσκηση 3:

Παρακάτω φαίνεται ο κώδικας **assembly** που φτιάξαμε:

```
.include "m328PBdef.inc"

;Initialize the vector table stored in FLASH
.org 0x0
rjmp start
.org 0x4
rjmp ISR1

start:
;Init stack pointer
       ldi r24, LOW(RAMEND)
       out SPL,r24
       ldi r24, HIGH(RAMEND)
       out SPH,r24
```

```
;Init of PORTD as read only
       clr r24                          ;set r18 bits to 0
       out DDRD, r24                    ;set portD as input
       ser r24                          ;set r18 bits to 1
       out PORTD, r24                   ;pull-up resistors so we dont have floating pins

;Init of PORTB as output only
       ser r24
       out DDRB, r24                    ;set portB as output only

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

;Enable the INT1 interrupt(PD3)
       ldi r24, (1 << INT1)
       out EIMSK, r24

       sei
                                        ;set the Global interrupt Flag
       clr r24
       clr r25
       clr r26
       clr r27


main:
       clr r16
       out PORTB,r16                    ;turn LEDS off

loop_renew:
       mov r28,r26
       mov r29,r27
       or r28,r29
       breq loop
       ldi r16, 0b00111111
       out PORTB,r16
       rcall delay_1ms
       sbiw r26,1
       rjmp loop_renew

loop:
       mov r28,r24
       mov r29,r25
       or r28,r29
       breq main
       ldi r16, 0b00001000
       out PORTB,r16
       rcall delay_1ms
       sbiw r24,1
       rjmp loop_renew

ISR1:
       in r20, SREG
       push r20
       push r28
       push r29
```

```
rst:
	ldi r20, (1 << INTF1)				;for button debounce
	out EIFR, r20					;clear INTF1
	rcall delay_1ms
	in r16, EIFR
	sbrc r16, INTF1					;skip if INTF1 = 0
	rjmp rst

	in r17, PORTB
	cpi r17,0
	brne renew
	clr r26
	clr r27
	ldi r24,LOW(4000)
	ldi r25,HIGH(4000)
	rjmp end_isr
renew:
	ldi r26,LOW(1000)
	ldi r27,HIGH(1000)
	ldi r24,LOW(3000)
	ldi r25,HIGH(3000)
end_isr:
	pop r29
	pop r28
	pop r20
	out SREG, r20
	reti

delay_1ms:
ldi r28,LOW(15992)		;1 cycle
ldi r29,HIGH(15992)		;1 cycle
delay:
sbiw r28,4			;2 cycle
brne delay			;2 cycle or 1 if last
;at this stage 15993 cycles
reti				;4 cycles
;+3 cycles or rcall so total 16000 cycles = 1 ms delay
```

Σχόλια/Περιγραφή του κώδικα:

Έχουμε 2 word καταχωρητές, τους **r24,r25** (αντιπροσωπεύουν τον χρόνο που θα είναι αναμένο το led PB3 της θύρας PORTB) και **r26,r27** (αντιπροσωπεύουν τον χρόνο που θα είναι αναμένα όλα τα led της θύρας PORTB). Τους αρχικοποιούμε με διαφορετικές τιμές ανάλογα με το αν τα LEDS του PORTB είναι σβηστά η όχι τη στιγμή που έγινε η διακοπή. Αν ήταν σβηστά αρχικοποιούμε (μέσα στη διακοπή) τους **r26,r27** στο **0** ενώ τους **r24,r25** στην τιμή **4000 (ms)**, ενώ αν δεν ήταν σβηστά σημαίνει πως κάνουμε ανανέωση του χρόνου και έτσι αρχικοποιούμε (μέσα στη διακοπή) τους **r26,r27** στο **1000** ενώ τους **r24,r25** στην τιμή **3000 (ms)**.
Στην main τρέχουμε ένα loop που ελέγχει τις τιμές των 2 word καταχωρητών **r24,r25** και **r26,r27**. Πρώτα τσεκάρουμε τον **r26,r27**. Αν ΔΕΝ είναι μηδενισμένος σημαίνει πως μένει χρόνος που πρέπει να ανάψουμε όλα τα led

της θύρας PORTB και έτσι τα ανάβουμε, αφαιρούμε 1 από τον τον **r26,r27** και κάνουμε πάλι jump στην αρχή για να ξανατσεκάρουμε. Αν ο **r26,r27** είναι μηδενισμένος, τσεκάρουμε τον **r24,r25.** Αν ΔΕΝ είναι μηδενισμένος σημαίνει πως μένει χρόνος που πρέπει να ανάψουμε το led PB3 της θύρας PORTB και έτσι το ανάβουμε, αφαιρούμε 1 από τον τον **r24,r25** και κάνουμε πάλι jump στην αρχή για να ξανατσεκάρουμε. Αν και οι 2 word καταχωρητές είναι μηδενισμένοι τότε κάνει jump στην main η οποία σβήνει τα τα led της θύρας PORTB και ελεγχει τις τιμές των 2 word καταχωρητών.

Παρακάτω φαίνεται ο κώδικας **C** που φτιάξαμε:

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

volatile int counter = 0;

ISR (INT1_vect)            //external INT1 ISR
{
       if(counter == 0) counter = 4000;
       else{
       PORTB = 0b00111111;
       _delay_ms(1000);
       PORTB = 0b00001000;
       counter = 3000;
       }
}

int main(void)
{
       //interrupt on rising edge of INT1 pin
       EICRA = (1<<ISC11) | (1<<ISC10);
       //enable the INT1 interrupt (PD3)
       EIMSK = (1<<INT1);
       //enable global interrupts
       sei();

       DDRB = 0xFF;         //set PORTB as output

    while(1)
    {
              PORTB = 0x00;

              while(counter>0)
              {
                     PORTB = 0b00001000;
                     counter -= 1;
                     _delay_ms(1);
              }
    }
}
```

Σχόλια/Περιγραφή του κώδικα:

Έχουμε μια volatile μεταβλητή **counter** όπου τον αρχικοποιούμε με διαφορετικές τιμές ανάλογα με το αν ήταν μηδενισμένος την στιγμή που έγινε η διακοπή ή όχι. Αν ήταν μηδενισμένος τη στιγμή της διακοπής τον αρχικοποιούμε στην τιμή 4000 (ms), ενώ αν δεν ήταν μηδενισμένος σημαίνει πως κάνουμε ανανέωση του χρόνου και έτσι ανάβουμε όλα τα LED του PORTB για 1 sec και τον αρχικοποιούμε στην τιμή 3000 (ms).
Στην main ενεργοποιούμε τις διακοπές και τρέχουμε ένα loop που ελέγχει την τιμή της μεταβλητής counter. Αν είναι 0 τότε σβήνουμε τα LED του PORTB ενώ αν είναι μεγαλύτερη του 0, τότε ανάβει το led PB3 της θύρας PORTB, μειώνουμε την τιμή της μεταβλητής counter κατά 1 και καλούμε 1ms delay.