

ΑΘΗΝΑ 10 Οκτωβρίου 2024

**2η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ**

**ΓΙΑ ΤΟ ΜΑΘΗΜΑ “Εργαστήριο Μικροϋπολογιστών”**

**ΟΜΑΔΑ 23**

**Συνεργάτες**

Νικόλαος Αναγνώστου Νικόλαος Λάππας

03121818 03121098

**Ζήτημα 2.1**

Α) Στο συγκεκριμένο ερώτημα καλούμαστε να φτιάξουμε ένα πρόγραμμα σε avr assembly στο οποίο να απαριθμούμε το πλήθος των διακοπών INT1 από 0 έως 63. Για τον σκοπό αυτό έχουμε χρησιμοποιήσει έναν μετρητή, τον counter = r16, τον οποίο αρχικοποιούμε στην τιμή μηδέν, και μέχρι να ξεπεράσει το 63 τυπώνουμε τον αριθμό των διακοπών (PD3) που έχουν προκύψει στα led PC5-PC0. Έχουμε ενσωματώσει στον κώδικα την απόρριψη των διακοπών όσο είναι πατημένο το button PD5 (στον κώδικα με κόκκινο χρώμα). Ο κώδικας που ακολουθεί δεν περιέχει το τμήμα κώδικα που δίνει η εκφώνηση.

.include "m328pbdef.inc"

.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

**.equ delay\_for\_int1 = FOSC\_MHZ \* 5 ; 5msec**

.def counter = r16 ; counter for external interrupts

.org 0x0

rjmp reset

.org 0x4

rjmp isr1

reset:

; stack initialisation

ldi r24,LOW(RAMEND)

out SPL,r24

ldi r24,HIGH(RAMEND)

out SPH,r24

clr r24

; PORTs initialisation

ser r26

out DDRB, r26 ; init PORTB as output

ser r26

out DDRC, r26 ; init PORTC as output

clr r26

out DDRD, r26 ; init PORTB as input

;Interrupt on rising edge of INT1 pin

ldi r24, (1 << ISC11) | (1 << ISC10)

sts EICRA, r24

;Enable the INT1 interrupt

ldi r24, (1 << INT1)

out EIMSK, r24

sei ; Enable general flag of interrupts

clr r24

ldi counter, 0 ; initialize counter for interrupts

;External interrupt 1 service routine

isr1:

push r23

push r24

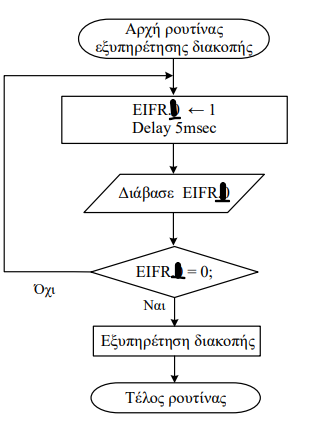
push r25

push r26

in r25, SREG

push r25 ; save r23, r24, r25, r26, SREG to stack

**; -----------------------------------------------------------**

 **; get rid of bounce phenomenon (B)**

**clear\_flag:**

**ldi r24, (1 << INTF1)**

**out EIFR, r24 ; Clear external interrupt 0 flag**

**ldi r24, low(delay\_for\_int1)**

**ldi r25, high(delay\_for\_int1); set delay (number of cycles)**

**rcall delay\_mS ; delay 5msec**

**; check if EIFR.1 1st bit is zero**

**; if yes, then skip the next operation (jmp or rjmp)**

**; and continue with the interruption routine**

**sbic EIFR, 1**

**jmp clear\_flag ; 3 cycles**

**;rjmp clear\_flag ; 2 cycles**

**;-----------------------------------------------------------**

in r26, PIND ; Read the state of PORTD

sbrs r26, 5 ; Skip the next instruction if PD5 is set

rjmp dont\_count ; Jump to 'dont\_count'

inc counter ; Increase counter

cpi counter, 64 ; If counter > 63, reset

breq reset

dont\_count:

out PORTC, counter ; if counter <= 63, print the answer

; Retrieve r23, r24, r25, r26, SREG from stack

pop r25

out SREG, r25

pop r26

pop r25

pop r24

pop r23

reti ; Return from interrupt

Β) Για το ερώτημα αυτό προσαρμόσαμε τον παραπάνω κώδικα έτσι ώστε να αποφεύγεται το φαινόμενο του σπινθηρισμού. Συγκεκριμένα, θέλουμε να αποτρέψουμε την καταμέτρηση συνεχόμενων διακοπών PD3 εάν αυτές προέκυψαν σε χρόνο μικρότερο του 5 msec. Για τον σκοπό αυτό ακολουθούμε το δοσμένο λογικό διάγραμμα (κατάλληλα προσαρμοσμένο ώστε να ανταποκρίνεται στην διακοπή INT1), κώδικας ο οποίος φαίνεται με bold γραμματοσειρά παραπάνω. Ορίσαμε μέσω της .equ την μεταβλητή **delay\_for\_int1** να περιέχει τον χρόνο που θέλουμε (5 msec) και στην συνέχεια γράψαμε την ρουτίνα **clear\_flag,** η οποία πρακτικά μηδενίζει το EIFR.1 (external interrupt flag register of INT1), δημιουργεί καθυστέρηση 5msec, και εξετάζει αν σε αυτόν τον χρόνο έχει γίνει ξανά set το EIFR.1. Αν ναι τότε επαναλαμβάνει την ίδια διαδικασία αγνοώντας την νέα αυτή διακοπή, και αν όχι, τότε προχωράει με την ρουτίνα εξυπηρέτησης της διακοπής.

**Ζήτημα 2.2**

Στο 2.2 Α) ζητήθηκε η υλοποίηση ενός απλού μετρητή από το 0 μέχρι το 31 ,με ενδιάμεση καθυστέρηση 2000ms (η οποία επιτεύχθηκε με ακρίβεια λόγω του δικού μας χρονομετρητή από την 1η σειρά που είχε και έχει απόλυτη ακρίβεια).Το πρόγραμμα αυτό είναι τόσο απλό που δεν κρίνεται ότι θέλει άλλη εξήγηση.

Στο 2.2 Β) ζητείται να έχουμε ρυθμίσει σωστά το πρόγραμμα μας ώστε να αντιδρά σε interrupts INT0 και όταν το κάνει να μετρά πόσα κουμπιά είναι πατημένα από τα PB3-PB0 και να ανάβει τόσα ακριβώς σε πλήθος στα PC0-PC3 ξεκινώντας από το PC0.Όταν η διακοπή ολοκληρώνεται και επιστρέφουμε στο μετρητή ,εκείνος έπρεπε να συνεχίζει.

Η διαδικασία στιγματίζεται από το ότι 1) στην αρχή κάναμε push το sreg (και στο τέλος κανουμε pop 2)χρησιμοποιούμε τη final για να υπολογίσουμε πόσα και ποια λαμπάκια θα πρέπει να ανοίξουν κάθε φορά και 3) εφευρέθηκε ένας αυτόματος μηχανισμός στον οποίο καθορίζεται σταδιακά το final,και αυτός είναι ο εξής: ”δες το πρώτο bit του critical και μετά κάντο λογικό or με το final,αν το bit ήταν 1 τότε μεταπήδησε στον έλεγχο για το ν+1οστο bit της final(δηλαδη στον έλεγχο για το αν το ν+1οστο bit της portc πρέπει να γίνει 0)” και αύξησε το counter που μετράει μέχρι 4 φορές, αν το bit είναι 0 τότε συνέχισε τον έλεγχο για το αν το ν-οστο bit του final πρέπει να είναι μηδέν και αύξησε το counter κατά ένα.”

Αυτή η διαδικασία παρομοιάστηκε στον κώδικα με brain και για το όλο σκεπτικό χρησιμοποιήθηκε ένας καταχωρητής που ονομάστηκε neuron ,ονομασία που ακολουθεί αυτή την προσομοίωση .

Ακολουθεί ο κώδικας για την όλη υλοποίηση (με κόκκινα όσα είναι (σημαντικά^Β\_ερώτημα)):

.include "m328pbdef.inc"

.equ M = 16 ; Mhz

.equ req = 2000 ;requested msec

.def counter = r17

.def critical = r31

.def neuron = r30

.def final = r29

.def counter1 = r28

.org 0x0

rjmp reset

.org 0x2

rjmp handler\_INT0

reset:

;stack initialisation

ldi r24,LOW(RAMEND)

out SPL,r24

ldi r24,HIGH(RAMEND)

out SPH,r24

clr r24

clr r25

;setting PORTC as input

ser r16

out DDRC,r16

clr r16

;PINB as inputs

out DDRB,r16

out DDRD,r16

ldi r16,0

STS EICRA,r16

ldi r16,1

out EIMSK,r16

clr r16

;timer initialisation

ldi r24,LOW(req)

ldi r25,HIGH(req)

sei

loop:

clr counter

main:

out PORTC,counter

rcall wait\_x\_msec

inc counter

cpi counter,32

brne main

rjmp loop

handler\_INT0:

IN R1,SREG

PUSH R1

rcall handler\_0\_brain

andi final,0x0F

out PORTC,final

POP R1

OUT SREG,R1

reti

handler\_0\_brain: ;preparing registers for the initialisation

in critical,PINB

com critical

andi critical,0x0F

clr final

clr counter1 ;will count up to four times

rcall first\_bit\_setup

ret

first\_bit\_setup:

mov neuron,critical

andi neuron,0x01

or final,neuron

inc counter1

cpi counter1,4

breq log\_out

cpi neuron,0x01

breq second\_bit\_setup

lsr critical

rjmp first\_bit\_setup

second\_bit\_setup:

mov neuron,critical

andi neuron,0x02

or final,neuron

inc counter1

cpi counter1,4

breq log\_out

cpi neuron,0x02

breq third\_bit\_setup

lsr critical

rjmp second\_bit\_setup

third\_bit\_setup:

mov neuron,critical

andi neuron,0x04

or final,neuron

inc counter1

cpi counter1,4

breq log\_out

cpi neuron,0x04

breq fourth\_bit\_setup

lsr critical

rjmp third\_bit\_setup

fourth\_bit\_setup:

mov neuron,critical

andi neuron,0x08

or final,neuron

out PORTC,final

log\_out:

ret

**Ζήτημα 2.3 – assembly**

Σε αυτή την άσκηση προσομοιάζεται το άναμμα και το σβήσιμο ενός φωτιστικού σώματος κάνοντας χρήση της διακοπής INT1 (PD3).Έπρεπε σε κάθε πάτημα του PD3 να ανάβει το PB0 για 5 δεύτερα αλλά με την exception ότι αν τυχών η διακοπή INT1 ενεργοποιούταν την στιγμή που είτε το PB0 μόνο ήταν αναμμένο ,είτε τα 6 led PB5-PB0 ήταν αναμμένα η χρονομέτρηση έπρεπε να ξαναρχίζει με το εξής τρόπο: να ανάψουν όλα τα led PB5-PB0 για μισό δεύτερο και κατόπιν να ανάβει μόνο το PB0 για 4,5 δεύτερα και μετά να σβήνει και να τελειώνει η διακοπή. Αν ενδιάμεσα επαναληφθεί διακοπή να γίνεται η ίδια διακοπή.

Να σημειωθεί πως για να λειτουργήσει κάτι τέτοιο σωστά σε assembly έπρεπε με κάποιο τρόπο να επιβληθεί στο πρόγραμμα να «ξεχνάει» τις προηγούμενες διακοπές…Αυτό έγινε κάνοντας σε κάθε διακοπή pop r1 δύο φορές ώστε η στοίβα να μην δείχνει στην πραγματική διεύθυνση επιστροφής και κατόπιν με rcall της handler της INT1 και αμέσως κάτω από αυτό rjmp στην main μία ατέρμονη loop που περιμένει τις διακοπές.

Ο κώδικας :

.include "m328pbdef.inc"

.equ req = 5000

.def epoptis = r23

.org 0x0

rjmp reset

.org 0x4

pop r1

pop r1

rcall handler\_INT1

rjmp main

reset:

ldi r16,LOW(RAMEND)

out SPL,r16

ldi r16,HIGH(RAMEND)

out SPH,r16

;interrupt enable

ldi r16,0x08

STS EICRA,r16

ldi r16,0x02

out EIMSK,r16

clr r16

out DDRD,r16

ser r16

out DDRB,r16

main:

clr epoptis

sei ;enabling interrupts

out PORTB,epoptis

rjmp main

handler\_INT1:

sei

cpi epoptis,1

breq exception

cpi epoptis,63

breq exception

ldi r24,LOW(5000)

ldi r25,HIGH(5000)

rjmp normal\_function

exception:

ldi r24,LOW(500)

ldi r25,HIGH(500)

ldi epoptis,63

out PORTB,epoptis

rcall wait\_x\_msec

ldi r24,LOW(4500)

ldi r25,HIGH(4500)

rjmp normal\_function

normal\_function:

ldi epoptis,1

out PORTB,epoptis

rcall wait\_x\_msec

ret

;loop

wait\_x\_msec:

ldi r26,LOW(15984);1 cycle

ldi r27,HIGH(15984);1 cycle

helper:

sbiw r26,4 ;2 cycles

brne helper ;2 cycles or 1 cycle for the last iteration

;15984 -> helper consumes 15983 cycles

;so after helper we consume totally 15985 cycles

sbiw r24,1 ;2 cycle

breq last\_msec ;1 cycle but if last msec 2 cycles

;for all msec except from the last -> 15985 + 2 + 1 = 15988 cycles

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop ;10 cycles

;extra 10 cycles -> 15998

brne wait\_x\_msec ;2 cycles total 16000 cycles with this operation

last\_msec:

;in the last iteration (last msec) we have 15989 cycles

nop

nop

nop

nop

;extra 4 cycles -> 15993 cycles

ret ;4 cycles

;with ret and rcall we calculated exactly 16000 cycles again

;so in both cases we end up having 16000 cycles -> 1 msec \* (desired time)

**Ζήτημα 2.3 – c**

Η συγκεκριμένη άσκηση προσομοιάζει το άναμμα και το σβήσιμο ενός φωτιστικού σώματος κάνοντας χρήση της διακοπής INT1 (PD3). Συγκεκριμένα, έχουμε έναν μετρητή που κάθε φορά που ενεργοποιείται μια διακοπή τον αρχικοποιούμε στην τιμή μηδέν εντός της ρουτίνας εξυπηρέτησης της διακοπής καθώς, επίσης, θέτουμε στην τιμή 1 ένα flag που μας δείχνει ότι έχει προκύψει μια διακοπή. Αφού ενεργοποιήσουμε τις διακοπές στην κατερχόμενη ακμή του ρολογιού και ενεργοποιήσουμε τις πύλες εισόδου-εξόδου, μπαίνουμε μέσα σε ένα while(1) loop στο οποίο εξετάζουμε αν έχει προκύψει διακοπή (interrupt\_flag = 1), και αν ναι τότε ανάβουμε για 5 δευτερόλεπτα το led PB0. Αν μέσα στον χρόνο αυτό έχει ενεργοποιηθεί ξανά η διακοπή, τότε και μόνο τότε εισερχόμαστε στο κομμάτι του κώδικα που παρατίθεται με bold, όπου ανάβουμε τα led PB0-PB5 για 0.5 δευτερόλεπτα και αφήνουμε ανοιχτό μετά το PB0 μόνο για ακόμα 4.5 δευτερόλεπτα.

Κάτι σημαντικό που παρατηρήσαμε είναι ότι το #define που φαίνεται με κόκκινο χρώμα για την F\_CPU πρέπει να ορίζεται πιο πάνω από τα #include (ειδικά το #include <util/delay.h>) επειδή η συνάρτηση \_delay\_ms() χρησιμοποιεί την συχνότητα του επεξεργαστή που της ορίζουμε κάθε φορά και εμείς θέλουμε να χρησιμοποιεί την συχνότητα του μικροεπεξεργαστή ATmega328PB που είναι 16MHz.

#define F\_CPU 16000000UL // needs to be here before #include <util/delay.h>

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

volatile uint8\_t interrupt\_flag = 0;// indicates interrupt occurence

volatile uint16\_t counter = 0; // counter for timing

// External INT1 interrupt routine (PD3)

ISR(INT1\_vect)

{

interrupt\_flag = 1; // interrupt occurs, open led PB0

counter = 0; // reset counter

EIFR = (1 << INTF1); // Clear the flag of interrupt INTF1

}

int main(void)

{

// Interrupt on falling edge of INT1 pin

EICRA = (1 << ISC11) | (0 << ISC10);

// Enable the INT1 interrupt mask (PD3)

EIMSK = (1 << INT1); // Mask for external interrupt INT1

sei(); // Enable global interrupts

DDRB = 0xFF; // Initialize PORTB as output

// remember to check if the following are necessary

DDRD &= ~(1 << PD3); // Initialize PD3 as input

PORTD |= (1 << PD3); // Initialize pull-up resistor of PD3

while (1)

{

if(interrupt\_flag)

{

interrupt\_flag = 0;

while(counter < 5000) // till 5 seconds pass

{

PORTB = 0x01; // open led PB0

\_delay\_ms(1); // delay 1 msec

counter++;

**// if interrupt flag is set again**

**if(interrupt\_flag)**

**{**

**interrupt\_flag = 0;**

**counter = 500; // reset the timer of 5 seconds**

**/\*counter=500 and not 0 because we have a delay of 0.5 seconds**

**so we need to adjust the counter as if it counted these seconds too\*/**

**PORTB = 0x3F; // open PB5-PB0**

**\_delay\_ms(500); // delay 0.5 sec**

**PORTB = 0x01; // open only PB0**

**}**

}

// Turn off PB0 LED of PORTB after 5sec

PORTB = 0x00;

}

}

return 0;

}