

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

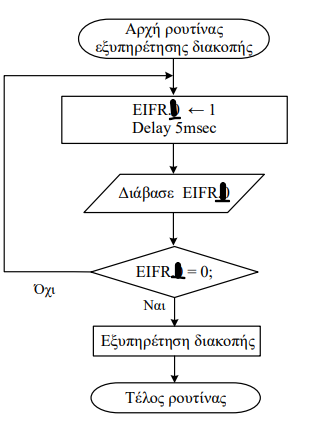
push r25

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, 63 ; 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**

**Ζήτημα 2.3 - assembly**

**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;

}