ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2022-2023

Αναφορά

2η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"

Ανάπτυξη κώδικα για το μικροελεγκτή ATmega328PB και προσομοίωση της εκτέλεσης του στο αναπτυξιακό περιβάλλον MPLAB X

Ομάδα 1

Ξανθόπουλος Παναγιώτης (03119084)

Παπαναστάσης Αθανάσιος (03113197)

Ζήτημα 2.1

Στο πρόγραμμα ορίζουμε τις παραμέτρους (συχνότητα επεξεργαστή, καθυστερήσεις σε ms) και τις συναρτήσεις για το μετρητή, όπως δίνονται από την εκφώνηση. Ορίζουμε επίσης τη ρουτίνα εξυπηρέτησης για τη διακοπή INT1 (θέση διακοπής 0x4), η οποία ενεργοποιείται στην ανερχόμενη ακμή.

Στη συνάρτηση init θέτουμε την PORTD ως είσοδο καθώς ελέγχουμε αν είναι πατημένο το PD7 ώστε να μην αυξάνουμε τον μετρητή των διακοπών. Επίσης, θέτουμε την PORTC ως έξοδο καθώς χρησιμοποιούμε τα led της PORTC ως έξοδο για να μετρήσουμε σε δυαδικό σύστημα το πλήθος των διακοπών.

Η συνάρτηση μετρητή έχει δοθεί από την εκφώνηση.

Στη ρουτίνα εξυπηρέτησης διακοπής, αφού αποθηκευτούν στη στοίβα οι τιμές των r24, r25 και SREG, υλοποιείται ο αλγόριθμος αποφυγής της αναπήδησης με τη συνάρτηση debounce. Στη συνάρτηση, ουσιαστικά υλοποιείται σε assembly το σχήμα που δόθηκε από την εκφώνηση.

Στη συνέχεια, ελέγχουμε αν έχει πατηθεί το PD7. Αν ναι, πηγαίνουμε κατευθείαν στο τέλος όπου επιστρέφουμε τις τιμές των καταχωρητών που αποθηκεύτηκαν στη στοίβα και επιστρέφουμε από την διακοπή. Αν όχι, αυξάνουμε τον μετρητή. Σε περίπτωση που έγινε 32, τον μηδενίζουμε πρώτα και μετά βγάζουμε το περιεχόμενό του στην PORTC. Τέλος, επιστρέφουμε τις τιμές των καταχωρητών που αποθηκεύτηκαν στη στοίβα και επιστρέφουμε από την διακοπή.

```
.include "m328PBdef.inc"

.equ FOSC_MHZ=16
.equ DEL_mS=500
.equ DEL_NU = FOSC_MHZ*DEL_mS

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

reset:
   ldi r24, low(RAMEND)
   out SPL,r24
   ldi r24, high(RAMEND)
   out SPH,r24

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

   ldi r24, (1 << INT1)
   out EIMSK, r24    ;INT1 is enabled at rising edge
```

```
sei

init:
   ser r26
   out DDRB, r26     ;PORT B as output
   out DDRC, r26     ;PORT C   as output
   clr r26
   out DDRD, r26     ;PORT D as output
   clr r22           ;r22 is interrupt counter, initialized at 0

loop1:
   clr r26           ;counter code (given)

loop2:
   out PORTB, r26

   ldi r24, low(DEL_NU)
   ldi r25, high(DEL_NU)
   rcall delay_mS

   inc r26

   cpi r26, 16
   breq loop1
   rjmp loop2



delay_mS:            ;delay of 1000*DEL_NU + 6 cycles
   ldi r23, 249    ;default is 249, for simulator set to 10
loop_inn:
   dec r23
   nop
   brne loop_inn

   sbiw r24, 1
   brne delay_mS

   ret



ISR1:
   push r25                  ;save r24, r25 because we use them later
   push r24
   in r24, SREG          ;save SREG
   push r24

debounce:                  ;debounce avoidance algorithm (given)
   ldi r24, (1 << INTF1)
   out EIFR, r24

   ldi r24, low(16*5)
   ldi r25, high(16*5)
   rcall delay_mS

   in r24, EIFR
   lsr r24
   lsr r24
   brcs debounce
```

```
check_PD7:
   in r24, PIND              ;check if PD7 is pressed, if yes, dont increase counter
                             ;and skip straight to the end
   lsl r24
   brcc end                  ;brcc for lab, brcs for simulator

   inc r22                   ;increase counter
   cpi r22, 32               ;if 32, set to 0
   brne next

   clr r22

next:
   out PORTC, r22            ;output

end:
   pop r24                   ;return saved values
   out SREG, r24
   pop r24
   pop r25

   reti
```

Ζήτημα 2.2

Στο πρόγραμμα αυτό, χρησιμοποιούμε ξανά τον μετρητή της εκφώνησης, με τις εξής διαφορές: Η καθυστέρηση ανάμεσα στα βήματα είναι 600ms, ο μετρητής φτάνει μέχρι το 32 και η έξοδος του μετρητή φαίνεται στην PORTC. Επίσης, χρησιμοποιούμε την διακοπή INT0, ξανά σε rising edge. Τέλος, θέτουμε και την PORTB ως είσοδο.

Στη ρουτίνα εξυπηρέτησης διακοπής, μετά τη συνάρτηση debounce, υλοποιείται η συνάρτηση start, η οποία καταχωρεί στον r22 την είσοδο του PORTB και ελέγχει κάθε bit αν είναι 0 ή 1. Αν είναι 0 (δηλαδή το κουμπί είναι πατημένο), τότε ο r21 αυξάνεται κατά 1, αλλιώς η εντολή για την αύξηση αγνοείται, λόγω της εντολής sbrs r22,x, όπου x η θέση του bit που ελέγχουμε. Μετά μηδενίζουμε τον r22.

Στη συνέχεια, στη συνάρτηση loop ελέγχουμε αν ο r21 είναι 0, οπότε δεν έχει πατηθεί κανένα κουμπί και συνεχίζει το πρόγραμμα με την υλοποίηση της συνάρτησης output, αλλιώς αλλάζουμε το lsb του r22, το μετακινούμε αριστερά και μειώνουμε κατά 1 την τιμή του r21. Η διαδικασία επαναλαμβάνεται μέχρι το r21 γίνει 0, οπότε και όσα buttons είχαν μετρηθεί με τον r21 τόσα συνεχόμενα bits έχουν γίνει 1.

Στη συνάρτηση output, βλέπουμε το περιεχόμενο του r22 στην PORTC και δημιουργεί καθυστέρηση για 1 δευτερόλεπτο, ώστε να προλάβουμε να παρατηρήσουμε τα led πάνω στην πλακέτα. Τέλος, επιστρέφουμε τις τιμές των καταχωρητών που αποθηκεύτηκαν στη στοίβα και επιστρέφουμε από την διακοπή.

```
.include "m328PBdef.inc"

.equ FOSC_MHZ=16
.equ DEL_mS=600
.equ DEL_NU = FOSC_MHZ*DEL_mS

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

reset:
   ldi r24, low(RAMEND)
   out SPL,r24
   ldi r24, high(RAMEND)
   out SPH,r24

   ldi r24, (1 << ISC01) | (1 << ISC00)
   sts EICRA, r24

   ldi r24, (1 << INT0)
   out EIMSK, r24    ;INT0 is enabled at rising edge

   sei

init:
   ser r26
   out DDRC, r26     ;PORT C as output
   clr r26
   out DDRB, r26     ;PORT B   as input


loop1:
   clr r26                ;counter code (given)

loop2:
   out PORTC, r26

   ldi r24, low(DEL_NU)
   ldi r25, high(DEL_NU)
   rcall delay_mS

   inc r26

   cpi r26, 32
   breq loop1
   rjmp loop2



delay_mS:                 ;delay of 1000*DEL_NU + 6 cycles
   ldi r23, 249    ;default is 249, for simulator set to 10
loop_inn:
   dec r23
   nop
   brne loop_inn

   sbiw r24, 1
   brne delay_mS

   ret
```

```
ISR0:
   push r25                         ;save r24, r25 because we use them later
   push r24
   in r24, SREG          ;save SREG
   push r24

debounce:                           ;debounce avoidance algorithm (given)
   ldi r24, (1 << INTF0)
   out EIFR, r24

   ldi r24, low(16*5)
   ldi r25, high(16*5)
   rcall delay_mS

   in r24, EIFR
   lsr r24
   brcs debounce

start:
   in r22, PINB
   clr r21     ;counter

   sbrs r22, 0             ;sbrc for simulator, sbrs for lab
   inc r21

   sbrs r22, 1
   inc r21

   sbrs r22, 2
   inc r21

   sbrs r22, 3
   inc r21

   sbrs r22, 4
   inc r21

   sbrs r22, 5
   inc r21

   clr r22

loop:
   cpi r21, 0x00
   breq output

   bclr 0
   lsl r22
   ori r22, 0x01

   dec r21
   rjmp loop

output:
   out PORTC, r22

   ldi r24, low(16*1000)        ;the output stays visible for 1 sec
   ldi r25, high(16*1000)       ;for validation purposes
   rcall delay_ms
```

```
end:
   pop r24                    ;return saved values
   out SREG, r24
   pop r24
   pop r25

   reti
```

Ζήτημα 2.3

Στο πρόγραμμα, χρησιμοποιούμε την INT1 σε rising edge. Ορίζουμε την PORTB ως έξοδο. Στη συνέχεια, στη συνάρτηση loop χρησιμοποιούμε μια καθυστέρηση για 3.5 δευτερόλεπτα και σβήνουμε όλα τα led της PORTB.

Στη ρουτίνα εξυπηρέτησης διακοπής, αφού γίνει ο έλεγχος για το σπινθηρισμό, αρχικά γίνονται όλα τα led της PORTB φωτεινά. Στη συνέχεια, δημιουργούμε καθυστέρηση για 0.5 δευτερόλεπτα και μετά αφήνουμε αναμμένο μόνο το led που αντιστοιχεί στο lsb.

Στη συνάρτηση end θέτουμε τους καταχωρητές r24:r25 στην τιμή που αντιστοιχεί σε καθυστέρηση 3,5 ώστε κατά την επιστροφή από την ρουτίνα εξ. διακοπής να έχει ανανεωθεί ο μετρητής .

```
.include "m328PBdef.inc"

.equ FOSC_MHZ=16
.equ DEL_1=3500
.equ DEL_NU1 = FOSC_MHZ*DEL_1
.equ DEL_2=500
.equ DEL_NU2 = FOSC_MHZ*DEL_2

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

reset:
   ldi r24, low(RAMEND)
   out SPL,r24
   ldi r24, high(RAMEND)
   out SPH,r24

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

   ldi r24, (1 << INT1)
   out EIMSK, r24
                              ;INT1 is enabled at rising edge
   sei
```

```
init:
   ser r26              ;PORT B is set as output
   out DDRB, r26

loop:
   ldi r24, low(DEL_NU1)   ;main loop is: turn off PORTB every 3.5 sec
   ldi r25, high(DEL_NU1)  ;if an interrupt comes, it adds 0.5 sec delay and
   rcall delay_mS        ;renews the 3.5 sec counter

   clr r26
   out PORTB, r26

   rjmp loop



delay_mS:               ;delay of 1000*DEL_NU_X + 6 cycles
   ldi r23, 249     ;default is 249, for simulator set to 15
loop_inn:
   dec r23
   nop
   brne loop_inn

   sbiw r24, 1
   brne delay_mS

   ret



ISR1:
   in r24, SREG          ;save only SREG because r25 and r24 are renewed
   push r24

debounce:                        ;debounce avoidance algorithm (given)
   ldi r24, (1 << INTF1)
   out EIFR, r24

   ldi r24, low(16*5)
   ldi r25, high(16*5)
   rcall delay_mS

   in r24, EIFR
   lsr r24
   brcs debounce

   ser r24
   out PORTB, r24

   ldi r24, low(DEL_NU2)
   ldi r25, high(DEL_NU2)
   rcall delay_mS

   ldi r24, 0x01
   out PORTB, r24
```

```
end:
   pop r24
   out SREG, r24

   ldi r24, low(DEL_NU1)        ;renew counter
   ldi r25, high(DEL_NU1)

   reti
```

Για τον κώδικα C αυτού του ζητήματος, αρχικά ορίζουμε τη μεταβλητή counter, η οποία καθορίζει την καθυστέρηση που θα χρησιμοποιήσουμε και είναι τύπου volatile, ώστε να μπορεί να αλλάξει η τιμή της και μέσα από τη ρουτίνα εξυπηρέτησης διακοπής. Σε αντιστοιχία με τον κώδικα assembly ορίζουμε τη συνάρτηση ISR, η οποία θέτει σε λειτουργία όλα τα led, δημιουργεί καθυστέρηση 0.5 δευτερολέπτου, απενεργοποιεί όλα τα led, εκτός από το lsb και ανανεώνει τη μεταβλητή counter στην αναγκαία τιμή για την επόμενη καθυστέρηση των 3.5 δευτερολέπτων.

Στο κύριο πρόγραμμα ενεργοποιούμε την εξωτερική διακοπή INT1 σε rising edge και θέτουμε την PORTB ως έξοδο και τα led να είναι σβησμένα. Με τη χρήση μιας επανάληψης while εξασφαλίζουμε ότι το πρόγραμμα δε θα σταματήσει να εκτελείται και με μια δεύτερη εμφωλευμένη, δημιουργούμε καθυστέρηση 1 ms, μειώνοντας τη μεταβλητή counter μέχρι να μηδενιστεί. Τότε τερματίζεται η εσωτερική επανάληψη, απενεργοποιούμε τα led και ανανεώνουμε την τιμή της μεταβλητής counter. Ο counter αρχικοποιείται στο 3500 (3496 ώστε να αντισταθμίζονται κάποιες καθυστερήσεις και ο χρόνος να είναι ακριβής) ώστε η εσωτερική while να διαρκεί 3500 επαναλήψεις x 1ms delay = 3.5 δευτερόλεπτα.

```
#define F_CPU 16000000UL

#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/delay.h>

volatile int counter = 3496UL;

ISR(INT1_vect) {
   PORTB = 0xFF;
   _delay_ms(500);
   PORTB = 0x01;
   counter = 3496;
}

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

   DDRB = 0xFF;
   PORTB = 0x00;
```

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

    DDRB = 0xFF;
    PORTB = 0x00;

    while (1) {
        while (1) {
            _delay_ms(1);
            counter = counter - 1;
            if (counter == 0) {
                break;
            }
        }
        PORTB = 0x00;
        counter = 3496;
    }
}
```