2023

Αναφορά 2ης Εργαστηριακής Άσκησης

Εργαστήριο μικροϋπολογιστών

ΕΜΜΑΝΟΥΗΛ ΞΕΝΟΣ (ΑΜ):03120850
ΠΑΠΑΔΟΠΟΥΛΟΣ ΣΠΥΡΙΔΩΝ (ΑΜ): 03120033

Ζήτημα 2.1

Σε αυτό το ζήτημα έγινε επέκταση του προγράμματος του μετρητή που φαίνεται στο σχήμα 2.1 έτσι ώστε να μετράει τον αριθμό των διακοπών INT1 από το 0 έως το 31 και όταν πατηθεί το PD6 η μέτρηση των διακοπών να παγώνει. Το αποτέλεσμα αυτής της μέτρησης θα φαίνεται στα led PC4-PC0. Ο κώδικας που υλοποιεί τα παραπάνω είναι ο ακόλουθος:

```
.org 0x00
rjmp reset
.org 0x004
rjmp ISR1


reset:
    ldi r16, 0

.include "m328PBdef.inc"
.equ FOSC_MHZ=16
.equ DEL_mS=500
.equ DEL_NU=FOSC_MHZ*DEL_mS

.equ DEL_Inter=5
.equ DEL_INT=FOSC_MHZ*DEL_Inter

.def interrupts=r16
ldi interrupts, 0

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

;Init PORTD as input
    clr r16
    out DDRD, r16

;Init PORTC as output
    ser r26
    out DDRC, r26

;Init PORTB as output
    ser r26
    out DDRB, r26
```

```
;Setting interrupt INT1
    ldi r26, (1 << ISC11) | (1 << ISC10)
    sts EICRA, r26

    ldi r26, (1<<INT1)
    out EIMSK, r26

    sei

loop1:
    clr r26
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:

    ldi r23, 249
loop_inn:
    dec r23
    nop
    brne loop_inn
    sbiw r24,1
    brne delay_ms
    ret

ISR1:
    in r17, SREG
    push r17
    push r24
    push r25

    debounce:   ;Debounce handling
    ldi r24, (1<<INTF1)
    out EIFR, r24
```

```
    ldi r24, low (DEL_INT)
    ldi r25, high (DEL_INT)
    rcall delay_mS
    sbic EIFR, INTF1
    rjmp exit

    sbis PIND, 6
    rjmp exit
    inc interrupts
    sbrc interrupts, 5
    clr interrupts
    out PORTC, interrupts
    exit:
    pop r25
    pop r24
    pop r17
    out SREG, r17
    reti
```

Στο παραπάνω πρόγραμμα αυτό που αξίζει να σχολιασθεί είναι η ρουτίνα εξυπηρέτησης της διακοπής. Σε αυτή αρχικά βάζουμε στο stack το περιεχόμενο του καταχωρητή κατάστασης και τους καταχωρητές r24 και r25 που είναι οι καταχωρητές που χρησιμοποιούνται για την καθυστέρηση που χρησιμοποιείται ώστε να αποφευχθεί το φαινόμενο του σπινθηρισμού. Έπειτα μηδενίζουμε το INTF0 και καλούμε delay 5ms και στη συνέχεια ελέγχουμε αν το bit INTF1 είναι 1. Σε περίπτωση που είναι 1 τότε σημαίνει ότι κλήθηκε μία άλλη ρουτίνα οπότε τερματίζουμε την ρουτίνα αφού κάνουμε pop τους καταχωρητές. Αν δεν έχει γίνει άλλο interrupt ελέγχουμε αν το PD6 είναι πατημένο. Αν είναι τότε κάνουμε pop τους καταχωρητές από την στοίβα και βγαίνουμε από την ρουτίνα. Αν δεν είναι τότε αυξάνουμε τον μετρητή που έχουμε για τα interrupts και ελέγχουμε αν είναι ίσως με 32. Αν είναι τότε τον μηδενίζουμε. Έπειτα εξάγουμε τον μετρητή στην PORTC και βγαίνουμε από το interrupt αφού κάνουμε pop τους καταχωρητές που έχουμε εισάγει στο stack.

Ζήτημα 2.2

Στο δεύτερο ζήτημα υλοποιήθηκε πρόγραμμα το οποίο όταν δεν υπάρχει διακοπή να μετράει από το 0 έως το 31 με καθυστέρηση 1sec σε κάθε αλλαγή και το αποτέλεσμα φαίνεται στα led PC4-PC0. Όταν υπάρχει διακοπή INT0 τότε στην PORTC θα πρέπει να ανάβουν τόσα LED όσα το πλήθος των button PB4-PB0 που είναι πατημένα. Παρακάτω φαίνεται ο κώδικας που υλοποιεί τα παραπάνω:

```
.org 0x00
rjmp reset
.org 0x02
rjmp ISR0


reset:

;.include "m328PBdef.inc"
.equ FOSC_MHZ=16
.equ DEL_mS=1000
.equ DEL_NU=FOSC_MHZ*DEL_mS

.equ DEL_Inter=10
.equ DEL_INT=FOSC_MHZ*DEL_Inter


;Init PORTC as output
    ser r26
    out DDRC, r26

;Init PORTB as input
    clr r26
    out DDRB, r26

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

;Setting interrupt INT0
    ldi r26, (1 << ISC01) | (1 << ISC00)
    sts EICRA, r26

    ldi r26, (1<<INT0)
    out EIMSK, r26
```

```
    sei

loop1:
    clr r26
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:

    ldi r23, 249
loop_inn:
    dec r23
    nop
    brne loop_inn
    sbiw r24,1 brne delay_ms
    ret



ISR0:
    in r17, SREG
    push r17
    push r24
    push r25

    debounce:   ;Debounce handling
    ldi r24, (1<<INTF0)
    out EIFR, r24
    ldi r24, low (DEL_INT)
    ldi r25, high (DEL_INT)
    rcall delay_mS
    sbic EIFR, INTF0
    rjmp exit
```

```
    ldi r19, 0  ;counter
    ldi r18, 5
    in r17, PINB
    com r17
    count:
    ror r17
    brcc zero
    lsl r19
    inc r19
    zero:
        dec r18
        brne count
    out PORTC, r19
    exit:
    pop r25
    pop r24
    pop r17
    out SREG, r17
    reti
```

Από τον παραπάνω κώδικα μπορεί κανείς να παρατηρήσει ότι η μόνη αλλαγή που έγινε στο «κύριο» μέρος του προγράμματος σε σχέση με το σχήμα 2.1 είναι η αλλαγή της παραμέτρου DEL_mS από 500 σε 1000 ώστε να έχουμε καθυστέρηση 1 sec μεταξύ των εναλλαγών και η αλλαγή cpi r26, 16 σε cpi r26, 32 ώστε η μέτρηση να γίνεται από το 0 ως το 31. Στην ρουτίνα εξυπηρέτησης της εξωτερικής διακοπής INT0 αρχικά κάνουμε όσο έχουν ήδη αναφερθεί στην προηγούμενη άσκηση σχετικά με το stack και την μέθοδο αντιμετώπισης του σπινθηρισμού. Στην συνέχεια χρησιμοποιούμε τον καταχωρητή r19 ώστε να κρατάμε το τρέχον αποτέλεσμα και τον r18 ώστε να πάρουμε τα πρώτα 5 bits της θύρας PORTB. Έτσι στην αρχή κάνουμε μία δεξιά ολίσθηση μέσω του κρατουμένου και ελέγχουμε αν το κρατούμενο είναι 0 ή 1. Αν το κρατούμενο είναι μηδέν τότε πηγαίνουμε στην ετικέτα zero, αλλιώς κάνουμε μία αριστερή ολίσθηση στο περιεχόμενο του καταχωρητή r19 και του προσθέτουμε ένα, ώστε το τελευταίο του ψηφίο να είναι 1. Έτσι για κάθε 1 που θα δούμε στα PB0-PB4 προσθέτουμε και ένα 1 στο τέλος του r19 και η ρουτίνα εξυπηρέτησης συνεχίζετε στην ετικέτα zero. Στην ετικέτα zero μειώνουμε τον καταχωρητή r18 κατά 1. Αν αυτός δεν είναι 0 μετά την μείωση τότε σημαίνει ότι έχουμε κι άλλο bits να ελέγξουμε, αλλιώς εξάγουμε το αποτέλεσμα στο PORTC και ολοκληρώνουμε την ρουτίνα εξυπηρέτησης της εξωτερικής διακοπής INT0 κάνοντας pop τις τιμές που είχαμε βάλει στην στοίβα και επιστρέφοντας στην κανονική ροή του προγράμματος.

Ζήτημα 2.3

Στο ζήτημα αυτό μας ζητήθηκε να υλοποιήσουμε την ένα πρόγραμμα το οποίο όταν γίνεται μία εξωτερική διακοπή INT1 να ανάβει το PB0 για 3 sec, ενώ αν πατηθεί ξανά το INT1 εντός αυτών των τριών δευτερολέπτων ο χρόνος να ανανεώνεται και να ανάβουν όλα τα led της θύρας PORTB για 0.5 sec. Αν εντός αυτών των 0.5 sec πατηθεί ξανά το PD3 ο χρόνος που τα led μένουν ανοιχτά ανανεώνεται.

Στο κομμάτι της υλοποίησης του προβλήματος αυτού αρχικά θα αναφέρουμε την γενική ιδέα βάση της οποίας γράφτηκε τόσο ο κώδικας σε assembly όσο και σε C και μετά θα μπούμε στις ιδιαιτερότητες κάθε υλοποίησης. Στην υλοποίηση μας έχουμε ένα flag που αρχικοποιείται στο 0 και το οποίο γίνεται ίσο με 1 όταν μπαίνουμε στην ρουτίνα εξυπηρέτησης της διακοπής , προκειμένου να μας δείχνει ότι έχει πραγματοποιηθεί ήδη μία διακοπή. Επίσης στην ρουτίνα εξυπηρέτησης της διακοπής έχουμε κάποιους counter, έναν για το PB0 ο οποίος παίρνει την τιμή 3000 κάθε φορά που γίνεται μία διακοπή και ένα για όλα τα φώτα που παίρνει την τιμή 500 κάθε φορά που ενεργοποιείται μία διακοπή και το flag έχει την τιμή 1. Στο κύριο πρόγραμμα ελέγχουμε αν οι δύο μετρητές είναι 0 ή όχι. Αν κάποιος μετρητής δεν είναι 0 τότε τα αντίστοιχα led ανάβουν και αυτός μειώνεται κατά 1.Αφού ανάψουν τα led πραγματοποιείται μία καθυστέρηση ενός msec, ώστε αθροιστικά να έχουμε για όλα τα led 500msec καθυστέρηση και για το PB0 3 sec καθυστέρηση. Γενικά πρώτα ελέγχεται ο μετρητής για όλα τα led και αν είναι αυτός 0 τότε ελέγχεται ο μετρητής για το PB0. Αν και οι δύο μετρητές είναι 0 τότε το flag γίνεται 0 και όλα τα led σβήνουν. Αυτή είναι η βασική λογική του κώδικά μας. Στην συνέχεια θα αναφέρουμε ότι χρειάζεται για να γίνει κατανοητή κάθε ξεχωριστή υλοποίηση. Ξεκινάμε με την assembly

Κώδικας σε assembly:

```
.include "m328PBdef.inc"

.equ FOSC_MHZ=16           ; Microcontroller frequency in MHz
.equ DEL_mS=500              ; (valid number from 1 to 4095)
.equ DEL_NU=FOSC_MHZ*DEL_mS   ; Delay_ms routine: (1000*DEL_NU+6)
.equ DEL_mS2=1              ; (valid number from 1 to 4095)
.equ DEL_NU2=FOSC_MHZ*DEL_mS2   ; Delay_ms routine: (1000*DEL_NU2+6)
.equ DEL_mS3=1              ; (valid number from 1 to 4095)
.equ DEL_NU3=FOSC_MHZ*DEL_mS3   ; Delay_ms routine: (1000*DEL_NU2+6)
.def counter=r17
.def temp=r18
.def all_lights=r20        ; designates that all lights must be lit
.def all_light_counter=r26
.def light_counter=r28


.org 0x0
    rjmp reset
.org 0x04
    rjmp ISR1
```

```
reset:
    clr all_light_counter
    clr r27
    clr light_counter
    clr r29
;initialize stack pointer
    ldi r24,LOW(RAMEND)
    out SPL,r24
    ldi r24,HIGH(RAMEND)
    out SPH,r24
;set interrupts
    ldi r24,(1<<ISC10) | (1<<ISC11)
    sts EICRA,r24

;enable the INT1 (PD2)
    ldi r24,(1<< INT1)
    out EIMSK, r24
;Init PORTB as output
    ser temp
    out DDRB , temp
    clr flag
    sei
loop1:
    clr all_lights
    out PORTB,all_lights
if2:
    adiw light_counter,1    ; check if interrupt is pressed at least once
    sbiw light_counter,1
    breq end_if2
if3:                ; check if interrupt is pressed at least twice
    adiw all_light_counter,1
    sbiw all_light_counter,1
    breq else_if3
    ser temp
    out PORTB,temp        ; if we are here all lights must be lit
    sbiw all_light_counter,1    ; decrease all_light_counter by 1
    rjmp end_if3
else_if3:            ; if all_lights=0 then only PB0 should be lit
    sbiw light_counter,1    ; decrease light_counter by 1
    ldi temp,1
    out PORTB,temp
end_if3:
    ldi r24,low(DEL_NU3)    ; a 1 ms delay
    ldi r25,high(DEL_NU3)
    rcall delay_ms
```

```
    rjmp if2
end_if2:
    rjmp loop1

    ISR1:

    ldi temp,(1<<INTF1)       ; set EIFR to 0
    out EIFR,temp
    push r24
    push r25
    ldi r24,low(DEL_NU2)
    ldi r25,high(DEL_NU2)
    rcall delay_ms
    pop r24
    pop r25
    sbic EIFR,INTF1
    reti
    cpi all_lights,0x00    ; if this is a second interrupt all lights must be lit
    breq only_PB0
    ldi all_light_counter,low(500)
    ldi r27,high(500)
only_PB0:
    ldi light_counter,low(3000)
    ldi r29,high(3000)
    ser all_lights
    reti



delay_ms:
    push r24
    push r25
    ldi r23,245   ; 1 cycle
    rjmp loop_inn  ; 2 cycles
my_delay:
    ldi r23,249
loop_inn:
    dec r23
    nop
    brne loop_inn
    sbiw r24,1
    brne my_delay
    pop r24
    pop r25
    ret
```

Στην assembly ο καταχωρητής r20(all lights) είναι το flag που καθορίζει αν έχει πατηθεί ήδη ένα interrupt εντός των τριών δευτερολέπτων του πρώτου. Επειδή οι αριθμοί 500 και 3000 απαιτούν πάνω από 8 bit για τους μετρητές για όλα τα φώτα και για το PB0 χρησιμοποιούμε ζευγάρια καταχωρητών. Για το PB0 τους r28(light counter) και r29 και για όλα τα led τους r26(all light counter) και r27. Επιπλέον για να ελέγξουμε αν κάποιος από τους διπλούς καταχωρητές αυτούς είναι 0 ή όχι προσθέτουμε χρησιμοποιώντας την adiw μία μονάδα και μετά αφαιρούμε μία μονάδα από τον διπλό καταχωρητή χρησιμοποιώντας την sbiw και χρησιμοποιούμε την breq για να ελέγξουμε αν το αποτέλεσμα της αφαίρεσης αυτή είναι 0 ή όχι. Αυτές είναι οι κύριες ιδιαιτερότητες του κώδικα σε assembly σε σχέση με την βασική ιδέα της υλοποίησης. Τέλος, προφανώς εντός της ρουτίνας εξυπηρέτησης της διακοπής ελέγχουμε για το φαινόμενο του σπινθηρισμού όπως κάναμε και στα προηγούμενα ερωτήματα.

Κώδικας σε C:

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

unsigned int light_counter;
unsigned int all_light_counter;
int all_flag;
int on = 0xFF;
int off= 0x00;

ISR (INT1_vect)
{
    EIFR=0xFF;
    _delay_ms(3);
    if(EIFR!=0xFF){
        if(all_flag){
            all_light_counter=500;
        }
        all_flag=1;
        light_counter=3000; // indicating 3000 ms
    }
}

int main() {
    EICRA =(1<< ISC10) | (1<< ISC11);
```

```
    EIMSK = (1<< INT1);
    sei();
    DDRB=0xFF;  //PORTB is output
    light_counter=0;
    all_light_counter=0;
    all_flag=0;
    while(1){
        while(light_counter!=0){
            if(all_light_counter!=0){
                PORTB=0xFF;
                all_light_counter--;
            }
            else{
                PORTB=0x01;
            }
            _delay_ms(1);
            light_counter--;
        }
        all_flag=0;
        PORTB=0x00;
    }

}
```

Ο κώδικας σε C αναδεικνύει πλήρως την λογική του αλγορίθμου που χρησιμοποιήσαμε για την επίλυση του συγκριμένου προβλήματος. Ο light counter είναι ο μετρητής για τον χρόνο που πρέπει να μείνει αναμμένο ακόμα το PB0, ενώ το all light counter είναι ο μετρητής για τον χρόνο που πρέπει να μείνει αναμμένα όλα τα led. Το all flag είναι το flag που παίρνει τιμή ένα εάν έχει πατηθεί η διακοπή και βρισκόμαστε εντός των τριών δευτερολέπτων που μένει αναμμένο το PB0. Προφανώς και στην C εντός της ρουτίνας εξυπηρέτησης της διακοπής ελέγχουμε και για το φαινόμενο του σπινθηρισμού προκειμένου να αποφύγουμε το ανεπιθύμητο άναμμα των led.