Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

2η Σειρά Ασκήσεων

Μάθημα: Εργαστήριο Μικροϋπολογιστών

Εξάμηνο: 7ο

Ονοματεπώνυμα: Αλεξοπούλου Γεωργία, Γκενάκου Ζωή

Ζήτημα 2.1

Α) Στον κώδικα του μετρητή του σχήματος 2.1, να προστεθεί ρουτίνα εξυπηρέτησης της εξωτερικής διακοπής INT1 (PD3), σε γλώσσα Assembly, η οποία να απαριθμεί το πλήθος των διακοπών INT1 από 0 έως 31. Όταν φτάσει 31 να αρχίζει ξανά από το μηδέν. Όσο είναι πατημένο το μπουτόν PD6 (λογικό 0) η μέτρηση των διακοπών παγώνει. Όταν το μπουτόν PD6 αφεθεί ξανά, η μέτρηση συνεχίζεται από το σημείο που είχε μείνει. Το πλήθος των εξωτερικών διακοπών να απεικονίζεται, σε δυαδική μορφή, στα leds PC4-PC0.

Β) Επειδή υπάρχει η πιθανότητα να εμφανιστεί το φαινόμενο του σπινθηρισμού, με το πάτημα του μπουτόν PD3 του προηγούμενου παραδείγματος, να προστεθούν οι κατάλληλες εντολές στη ρουτίνα εξυπηρέτησης της εξωτερικής διακοπής INT1, ώστε να αποφευχθεί το φαινόμενο αυτό.

```
.include "m328pbdef.inc"
.equ FOSC_MHZ=16
.equ delay=500
.equ F1=FOSC_MHZ*delay

.org 0x00
    rjmp reset  ; Reset Handler
```

```
.org 0x04
    rjmp ISR1   ; ISR1 Handler

reset:
    ; Initialize the stack pointer
    ldi r24, low(RAMEND)
    out SPL, r24
    ldi r24, high(RAMEND)
    out SPH, r24

    ; Init PORTC as output
    ser r26
    out DDRC, r26

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

    ; Init PORTD as input
    clr r26
    out DDRD,r26

    ; Configure external interrupt INT1 (PD3) for falling edge
trigger
    ldi r24, (1 << ISC11) | (1 << ISC10)
    sts EICRA, r24

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

    ; Enable global interrupts
    sei

loop1:
    clr r27

loop2:
```

```
    ; Set all pins of Port B to 0
    out PORTB, r27

    ; Load delay value
    ldi r24, LOW(F1)
    ldi r25, HIGH(F1)
    nop
    rcall delay_mS

    ; Increment r27
    inc r27

    ; Compare r27 with 32
    cpi r27, 32
    brne loop2  ; Branch if r27 is equal to 32
    rjmp loop1  ; Repeat loop2

delay_mS:
    ; Delay function using nested loops
    ldi r23, 249
    loop_inn:
        dec r23
        nop
        brne loop_inn
        sbiw r24, 1
        brne delay_mS
    ret

ISR1:
    push r21

    push r22
    push r23
    push r24
    push r25
    in r21, SREG
    push r21

    loop:
```

```
; Load delay value
ldi r24, low(F1)
ldi r25, high(F1)
ldi r22, (1 << INTF1)

; Clear the interrupt flag for INT1
out EIFR, r22

; Call delay function
rcall delay_mS

; Read the EIFR register to check INTF1 flag
in r22, EIFR
andi r22, 2

; Branch if INTF1 flag is set
brne loop

; Read PIND to check a specific bit (bit 6)
in r19, PIND
andi r19, 64

; Branch if the bit is not set
breq END

; Increment r20
inc r20

; Compare r20 with 32
cpi r20, 32

; Branch if r20 is not equal to 32
brne NEXT

; Clear r20
clr r20

NEXT:
; Output r20 to Port C
```

```
out PORTC, r20

END:
; Restore the SREG
pop r21
out SREG, r21

; Pop registers
pop r25
pop r24
pop r23
pop r22
pop r21

; Enable global interrupts
sei

reti
```

Ο κώδικας εξηγείται περιληπτικά στα σχόλια, ωστόσο αξίζει να σχολιαστούν σε βάθος ορισμένα τμήματά του.
Αρχικά, προσδιορίζουμε τις διευθύνσεις μνήμης του reset handler και της ρουτίνας διακοπής INT1. Εντός του reset, αρχικοποιούμε τον Stack Pointer στην κορυφή της μνήμης SRAM, διαμορφώνουμε κατάλληλα τις θύρες εισόδου (PORTD) και εξόδου (PORTB, PORTC), καθώς και την εξωτερική διακοπή INT1, ώστε να ενεργοποιείται με rising και falling trigger του ακροδέκτη PD3. Οι ετικέτες loop1 και loop2 “καθαρίζουν” και καθορίζουν τον counter των ακροδεκτών της θύρας B, αξιοποιώντας και τη ρουτίνα καθυστέρησης delay_mS. Εντός της ετικέτας ISR1, πραγματοποιείται η υλοποίηση της ρουτίνας παροχής υπηρεσιών διακοπής INT1, αξιοποιώντας τα δεδομένα εισόδου της θύρας D και ενημερώνοντας κατάλληλα τα δεδομένα εξόδου της θύρας C.
Μέσα στο interrupt γίνονται push και pop οι καταχωρητές r21-r25 και του SREG για να διασφαλιστεί ότι το ISR δεν παρεμβαίνει στην κατάσταση αυτών των καταχωρητών, οι οποίοι ενδέχεται να χρησιμοποιούνται και από τον κύριο κώδικα. Οι καταχωρητές επαναφέρονται στις αρχικές τους τιμές στο τέλος της ετικέτας ISR1, μετά το πέρας της οποίας ενεργοποιείται και το global interrupt flag.
Ένα από τα μεγαλύτερα ζητήματα που ανακύπτουν με τη χρήση των INT0/INT1 αποτελεί το πρόβλημα του σπινθηρισμού, κατά το οποίο γίνεται λάθος καταγραφή των δεδομένων εισόδου κατά το falling/rising edge του PD3. Για τη διασφάλιση της σωστής λειτουργίας του κώδικα, αξιοποιούμε έναν βρόχο («βρόχο») για συνεχή έλεγχο για INTF1, που είναι η σημαία που υποδεικνύει την εμφάνιση εξωτερικής διακοπής στο INT1. Ο βρόχος ελέγχει επανειλημμένα αυτήν τη σημαία και εκτελεί τις ακόλουθες ενέργειες εάν έχει οριστεί η σημαία. Η σημαία αυτή διαγράφεται με την εγγραφή της τιμής 1 στον EIFR.

Αφού ελεγχθεί η εγκυρότητα των input data του interrupt, ελέγχουμε την είσοδο του ακροδέκτη PD6. Ο έλεγχος αυτός επιτυγχάνεται με την εντολή “andi r19, 64”, κατά την οποία απομονώνεται το 7ο ψηφίο της θύρας εισόδου PORTD, το οποίο στη συνέχεια συγκρίνεται με το ψηφίο 0. Τέλος, το πλήθος των εξωτερικών διακοπών που έχουμε προκαλέσει τυπώνεται σε binary σύστημα στη θύρα εξόδου PORTC.
Αξίζει να σχολιαστεί πως, μετά την επεξεργασία, ο κώδικας επαναφέρει την αρχική κατάσταση των καταχωρητών και ενεργοποιεί καθολικές διακοπές πριν επιστρέψει από το ISR χρησιμοποιώντας την εντολή “reti”.

Ζήτημα 2.2

Α) Να υλοποιηθεί κώδικας Assembly για το μικροελεγκτή ATmega328PB, αντίστοιχο με τον κώδικα του μετρητή του σχήματος 2.1, με τη διαφορά ότι η μέτρηση θα είναι 0 έως 31, η απεικόνιση θα γίνεται στα leds PC4-PC0 και ο χρόνος καθυστέρησης μεταξύ των εναλλαγών θα κυμαίνεται στα 1000 mS.
Β) Η ρουτίνα εξυπηρέτησης της εξωτερικής διακοπής INT0 (PD2) να τροποποιηθεί έτσι ώστε όταν ενεργοποιείται να ανάβει τόσα LEDs της θύρας PORTC αρχίζοντας από τα LSB όσο το πλήθος των 5 μπουτόν PB4 - PB0 που είναι πατημένα.

```
.include "m328PBdef.inc"
.equ FOSC_MHZ=16
.equ delay=1000
.equ F1=FOSC_MHZ*delay

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

reset:
    ; Initialize the stack pointer
    ldi r24, low(RAMEND)
    out SPL, r24
    ldi r24, high(RAMEND)
    out SPH, r24

    ; Set all pins of Port C as outputs
    ser r26
    out DDRC, r26

    ; Set only the lower 5 bits of PORTB as inputs (PB0-PB4)
```

```
    clr r26
    out DDRB, r26

    ; Configure external interrupt INT0 for rising edge trigger
    ldi r24, (1 << ISC01) | (1 << ISC00)
    sts EICRA, r24

    ; Enable external interrupt INT0
    ldi r24, (1 << INT0)
    out EIMSK, r24

    ; Enable global interrupts
    sei

loop1:
    clr r26
loop2:
    ; Set all pins of Port C to 0
    out PORTC, r26

    ; Load delay value
    ldi r24, low(F1)
    ldi r25, high(F1)
    nop
    rcall delay_mS

    ; Increment r26
    inc r26

    ; Compare r26 with 32
    cpi r26, 32
    brne loop2
    rjmp loop1

delay_mS:
    ; Delay function using nested loops
    ldi r23, 249
    loop_inn:
        dec r23
```

```
        nop
        brne loop_inn
        sbiw r24, 1
        brne delay_mS
    ret

ISR0:
    ; Push registers onto the stack to preserve their values
    push r20
    push r21
    push r22
    push r23
    push r24
    push r25

    ; Save the state of the status register (SREG) and push it onto
the stack
    in r20, SREG
    push r20

    ; Clear r20 (used for bit manipulation)
    clr r20

    ; Read the state of the PINB register (Port B input)
    in r20, PINB

    ; Invert all bits in r20 (complement)
    com r20

    ; Initialize r21 for loop control (5 iterations for PB4-PB0)
    ldi r21, 5

    ; Clear r22 (used for accumulating bits)
    clr r22

    loop3:
        ; Rotate right through carry (bit 0 becomes the carry flag)
        ror r20
```

```
        ; Check the carry flag
        brcc NEXT

        ; If the carry flag is set, rotate left through carry and
accumulate in r22
        rol r22

        NEXT:
        ; Decrement the loop counter r21
        dec r21

        ; Repeat the loop while r21 is not zero
        brne loop3

    ; Output the accumulated value (r22) to Port C
    out PORTC, r22

    ; Load the delay values
    ldi r24, LOW(F1)
    ldi r25, HIGH(F1)

    ; Call the delay function
    rcall delay_mS

    ; Restore the registers and status register
    pop r20
    out SREG, r20
    pop r25
    pop r24
    pop r23
    pop r22
    pop r21
    pop r20

    ; Enable global interrupts
    sei

    ; Return from the interrupt
    reti
```

Όπως και στο *Ζήτημα 2.1*, ο παραπάνω κώδικας ξεκινά με τον reset handler, ο οποίος προετοιμάζει τον Stack Pointer και διαμορφώνει τη θύρα PORTC ως έξοδο και τα 5 LSBs της θύρας PORTB ως είσοδο. Ρυθμίζει επίσης την εξωτερική διακοπή INT0 για ενεργοποίηση σε rising edge και επιτρέπει global interrupts. Και πάλι, δύο ένθετοι βρόχοι (loop1 και loop2) ελέγχουν το μοτίβο εξόδου της θύρας C, με καθυστέρηση που παρέχεται από τη συνάρτηση delay_mS.
Αντίστοιχα με προηγουμένως, η ετικέτα ISR0 υλοποιεί τη ρουτίνα εξυπηρέτησης της εξωτερικής διακοπής INT0. Προς διασφάλιση των τιμών των καταχωρητών r21-r25 και του SREG, γίνεται ξανά push και pop αυτών εντός του βρόχου.
Στη συνέχεια, τα δεδομένα εισόδου διαβάζονται από τη θύρα PORTB, τα bit της οποίας αντιστρέφονται. Εντός του loop3 γίνεται η καταμέτρηση των set bits της θύρας B και μετά το πέρας του βρόχου αυτού, το αποτέλεσμα τυπώνεται στη θύρα C (στο δεκαδικό σύστημα).

Ζήτημα 2.3

Να υλοποιηθεί αυτοματισμός που να ελέγχει το άναμμα και το σβήσιμο ενός φωτιστικού σώματος. Όταν πατάμε το push button PD3 (δηλαδή με την ενεργοποίηση της INT1) να ανάβει το led PB0 της θύρας PORTB (που υποθέτουμε ότι αντιπροσωπεύει το φωτιστικό σώμα). Το led θα σβήνει μετά από 4 sec, εκτός και αν ενδιάμεσα υπάρξει νέο πάτημα του PD3, οπότε και ο χρόνος των 4 sec θα ανανεώνεται. Κάθε φορά που γίνεται ανανέωση να ανάβουν όλα τα led της θύρας PORTB (PB7-PB0) για 0.5 sec, μετά να σβήνουν εκτός από το led PB0 που παραμένει συνολικά για 4 sec εκτός και αν ανανεωθεί. Ο κώδικας να δοθεί σε assembly και C.

Assembly:

```
.include "m328PBdef.inc"
.equ FOSC_MHZ=16
.equ basic_delay=3000
.equ double_delay=500
.equ F1=FOSC_MHZ*basic_delay
.equ F2=FOSC_MHZ*double_delay

.org 0x00
     rjmp reset
.org 0x04
     rjmp ISR1

reset:
```

```
      ldi r24, low(RAMEND)
      out SPL, r24
      ldi r24, high(RAMEND)
      out SPH, r24
      ser r26
      out DDRC, r26
      out DDRB, r26
      ldi r24, (1 << ISC11) | (1 << ISC10)
      sts EICRA, r24
      ldi r24, (1 << INT1)
      out EIMSK, r24
      sei
      loop1:
      clr r20
      rjmp loop1

delay_mS:

    ldi    r23, 249       ; (1 cycle)
loop_inn:
    dec r23               ; 1 cycle
    nop                   ; 1 cycle
    brne loop_inn         ; 1 or 2 cycles

    sbiw r24 ,1           ; 2 cycles
    brne delay_mS         ; 1 or 2 cycles

    ret                   ;4 cycles


ISR1:

cpi r20,0
breq first_interrupt

more_interrupts:
ser r22
out PORTB,r22
ldi r24, low(F2)
```

```
ldi r25, high(F2)
rcall delay_mS
ldi r22,1
out PORTB,r22
ldi r24, low(F1)
ldi r25, high(F1)
ldi r23, 249
sei
reti


first_interrupt:
ldi r21,1
out PORTB,r21
ser r20
sei
ldi r24, low(F1)
ldi r25, high(F1)
rcall delay_mS
ldi r21,0
out PORTB,r21
reti
```

Στο *Ζήτημα 2.3*, όπως και στο *Ζήτημα 2.1*, ο κώδικας ξεκινά με τον reset handler, ο οποίος προετοιμάζει τον Stack Pointer και διαμορφώνει τη θύρα PORTB ως έξοδο. Ο κώδικας ρυθμίζει μια εξωτερική διακοπή στον ακροδέκτη INT1 (PD3) και εκτελεί τις ακόλουθες ενέργειες όταν συμβεί η διακοπή:

Όταν γίνει επαναφορά του μικροελεγκτή, αρχικοποιεί ορισμένες ρυθμίσεις και στη συνέχεια εισάγει έναν βρόχο που διαγράφει συνεχώς έναν καταχωρητή (r20), που στην δικιά μας περίπτωση χρησιμοποιείται ως flag, για να ξέρουμε αν το LED είναι ανοιχτό ή όχι. Όταν ενεργοποιείται η εξωτερική διακοπή (INT1), ελέγχει την τιμή στον καταχωρητή r20.

- Εάν το r20 είναι μηδέν, πηγαίνει στην ετικέτα «Frst_Int», δηλαδή ότι το LED ανάβει για πρώτη φορά και ορίζει την έξοδο στο PORTB σε 1 (ανάβει το LED), εισάγει μια καθυστέρηση και κλείνει πάλι το LED.
- Εάν το r20 δεν είναι μηδέν, πηγαίνει στην ετικέτα «more» ανάβει όλα τα LED για 0.5s και στην συνέχεια ανάβει πάλι το PB0.

C:

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

volatile int pointer = 0; // Variable to track the state of the LED
int counter;

ISR(INT1_vect){ // Interrupt Service Routine for External Interrupt 1

    if(pointer == 0){ // If the LED is currently off
        pointer = 1; // Set the flag to indicate LED is on
        PORTB = 0x01; // Turn on the LED connected to PORTB pin 0
        sei(); // Enable global interrupts
        for(counter=1; counter<=3000; counter++){
                _delay_ms(1); // Delay for 3000 milliseconds (3
seconds)
        }
        PORTB = 0x00; // Turn off the LED
    }
    else{ // If the LED is currently on
        PORTB = 0xFF; // Turn on all the LEDs connected to PORTB
        for(counter=1; counter<=500; counter++){
                _delay_ms(1); // Delay for 500 milliseconds (0.5
seconds)
        }
        PORTB = 0x01; // Turn on the LED connected to PORTB pin 0
        counter=1;
    }

}

int main(){

    EICRA = (1 << ISC11) | (1 << ISC10); // Configure External
Interrupt 1 to trigger on rising edge
    EIMSK = (1 << INT1); // Enable External Interrupt 1
```

```
    sei(); // Enable global interrupts

    DDRB = 0xFF; // Configure PORTB as output

    while(1){
        PORTB = 0x00; // Turn off the LED (assuming all other PORTB
pins are also off)
        pointer=0;
    }
}
```

Για το πρόγραμμα στην C, ακολουθεί παρόμοια υλοποίηση, δηλαδή έχουμε πάλι ένα flag το οποίο μας καθορίζει αν το LED είναι αναμμένο ή όχι και εκτελεί αντίστοιχα τις απαραίτητες ενέργειες.
Αξίζει να σταθούμε στο σημείο που εισάγουμε τις καθυστερήσεις. Ενώ με την συνάρτηση `_delay_ms()` θα μπορούσαμε να εισάγουμε απευθείας την θεμιτή καθυστέρηση, (4 ή 0.5s), χρησιμοποιούμε εναν επαναληπτικό βρόχο που καλούμε όσες φορές χρειάζεται μια καθυστέρηση 1ms `_delay_ms(1).`Εάν ο χρόνος που διαρκεί η εκτέλεση μιας ρουτίνας εξυπηρέτησης διακοπής είναι πολύ μικρός, ενδέχεται να εμφανιστεί το φαινόμενο της αναπήδησης, λόγω σπινθηρισμού στον πιεστικό διακόπτη που προκαλεί την εξωτερική διακοπή. Συγκεκριμένα, ένας πιεστικός διακόπτης μπορεί να δώσει περισσότερα του ενός σήματα διακοπής τόσο κατά την πίεση του όσο και κατά την απελευθέρωση του, οπότε μετά την ολοκλήρωση της ρουτίνας εξυπηρέτησης της διακοπής μπορεί να καταφθάσουν και άλλα σήματα διακοπής που οφείλονται στο ίδιο πάτημα ή απελευθέρωση του πιεστικού διακόπτη. Ο επαναληπτικός βρόχος περιλαμβάνει πολλές κλήσεις συναρτήσεων και επιβάρυνση ελέγχου βρόχου, κάτι που μπορεί να το κάνει πιο αργό σε σύγκριση με το ενσωματωμένο _delay_ms(3000), το οποίο λύνει το πρόβλημα του σπινθηρισμού.