Εργαστήριο Μικροϋπολογιστών: 2η Εργαστηριακή Άσκηση (2022)

Ομάδα 64

Ιωάννου Κωνσταντίνος AM:031-19840

Μυτιληναίος Γιώργος AM:031-19841

```
01:.include "m328PBdef.inc"
02:     .equ FOSC_MHZ=16
03: .org 0x0
04:     rjmp reset
05: .org 0x4
06:     rjmp ISR1
07: .org 0x1A
08:     rjmp ISR_TIMER1_OVF
09:
10: reset:
11:     ldi r24, (1<<ISC11)|(1<<ISC10)
12:     sts EICRA, r24
13:     ldi r24, (1<<INT1)
14:     out EIMSK, r24
15:     sei
16:     ser r25
17:     out DDRB ,r25
18:     clr  r25
19:     out DDRD ,r25
20:     out DDRC ,r25
21:      clr r26    // r26: 0 of all led is off / 1 if led is on
22:
23: start:
24:     sei
25:     clr r25
26:     in r25 ,PINC
27:     cli
28:     andi r25 , 0b000100000
29:     cpi  r25 , 0b000100000
30:     breq start
31:     jmp ISR1
32:
33: timer_1:
34:     sei
35:     ldi r24,(1<<CS12)|(0<<CS11)|(1<<CS10)
36:     sts TCCR1B ,r24
37:     cli
38:     ldi r24,HIGH(3035)
39:     sts  TCNT1H,r24
40:     ldi r24,LOW(3035)
41:     sts  TCNT1L,r24
42:     ldi r24,(1<<TOIE1) ;TCNT1
43:     sts TIMSK1 ,r24    ; timer1
44:     sei
45:     cli
46:     jmp  L1
```

Ζήτημα 3.1.asm:

Στην άσκηση 1 χρησιμοποιείτε ο μετρητής για να ελέγξουμε πόση ώρα θα είναι ανοικτά τα LEDS. Αρχικά ενεργοποιείτε ο διακόπτης INT1 κι θέτεται το PORTB output. Στη συνέχεια στην start (γραμμή 23) ελέγχει το PC5, αν το εντοπίσει τρέχει την ρουτίνα ISR1 (γραμμή 31). Η ρουτίνα ISR1 (γραμμή 73) ελέγχει αρχικά αν έχει ξαναπατηθεί ένα από τα δύο κουμπιά ελέγχοντας τον καταχωρητή r26. Αν δεν έχει ξαναπατηθεί τότε κάνει jump στην ρουτίνα timer1 (γραμμή 33) όπου θέτει από ποια τιμή θα αρχίσει να μετράει ο timer κι όταν φτάσει στην μέγιστη τιμή(διάρκεια 4sec) κάνει άλμα στην ISR_TIMER1_OVF (γραμμή 64) για να κλείσει τα LEDs κι να τελειώσει την ρουτίνα. Αφού τελειώσει η timer1 γίνεται άλμα στην L1 (γραμμή 79) όπου ανάβει το PB1, θέτει άσσο τον καταχωρητή r26 κι γυρνάει στην start. Στην περίπτωση που πατιέται το PD5 ή το INT1 $2^{η}$ φορά η ISR1 θα κάνει άλμα στην timer2 όπου θα θέσει την τιμή του timer (θα μείνει ανοικτό για 500ms) κι ανάβει όλα τα LEDS του PORTB. Μετά κάνει άλμα στην D_loop (γραμμή 85) όπου κάθεται για 500msec κι μετά θα πάει στην timer1 (γραμμη 87).

```
48: timer_2:
49:     ser  r25
50:     out PORTB ,r25
51:     cli
52:     ldi r24,(1<<CS12)|(0<<CS11)|(1<<CS10)
53:     sts TCCR1B ,r24
54:     ldi r24,HIGH(58000)
55:     sts  TCNT1H,r24
56:     ldi r24,LOW(58000)
57:     sts  TCNT1L,r2
58:     ldi r24,(1<<TOIE1) ;TCNT1
59:     sts TIMSK1 ,r24    ; timer1
60:     sei
61:     ldi r22 , 0
62:     jmp D_loop  // for some delay 0_5 sec
63:
64: ISR_TIMER1_OVF:
65:         clr r25
66:         out PORTB ,r25
67:         ldi r26,0
68:         ldi r22 ,1
69:         reti
70:
71:
72:
73: ISR1:
74:         cpi  r26 ,  0
75:         brne  timer_2
76:         jmp   timer_1
77:
78:
79: L1:
80:     ldi r26 , 1
81:     ldi r30 ,0b0001
82:     out PORTB ,r30
83:     jmp start
84:
85: D_loop:
86:     cpi r22 , 0
87:         brne timer_1
88:         jmp D_loop
```

Δηλαδή χρησιμοποιούμε δύο καταχώρησες ως flags ,τον r26 ως flag για το αν θα πάμε ρουτίνα timer_4sec ή στην ρουτίνα timer_05sec.Συγκεκριμένα όταν r26 είναι 0 , σημαίνει ότι το led είναι κλειστό οπότε πάει στο timer_04sec και ανάβει το PB1 , ενώ θέτει το r26 ώς 1 ώστε αν πατηθεί πάλι διακοπή ή το PC5 πριν «σκάσει» ο Timer1 με r26 , να πάει στην ρουτίνα timer_05sec να ανάψει τα leds του PORTB .Εδω «αναλαμβάνει» ο r22 ο οποίος όσο είναι ίσος με 0 , φροντίζει να μένει το πρόγραμμα στη D_loop για να μένουν όλα τα leds ανοιχτά για 0,5 sec , μετά από αυτό το χρονικό διάστημα που «σκάει» ο timer θέτουμε r22 = 1 ώστε να φύγουμε από την loop και να ενεργοποιήσουμε την ρουτίνα timer_4sec ώστε να ανάψει το PB1 για άλλα 4 sec , ανανενώνοντας προφανώς τον χρόνο.

```
01: #define F_CPU  16000000UL
02: #include <stdio.h>
03: #include <stdlib.h>
04: #include<avr/io.h>
05: #include<util/delay.h>
06: #include <avr/io.h>
07: #include <avr/interrupt.h>
08:
09: int x = 0x0; // x = 0 if led is off / x = 1 if
led is on
10: int y;       //y = 0 dont go to timer_1 wait
/ y =1 go now to timer_1
11: void Timer_4();
12: void Timer_05();
13:
14: void ISR1(TIMER1_OVF_vect) {
15:
16:     x=0x0;
17:     y =0x1;
18:     PORTB=0x00;
19:
20: }
21:
22: void Timer_05(){
23:
24:     TIMSK1 =  (1<<TOIE1);
25:     TCCR1B =
(1<<CS12)|(0<<CS11)|(1<<CS10);
26:     cli();
27:     TCNT1H = 57700;
28:     TCNT1L = 57700;
29:     sei();
30:     PORTB=0xFF;
31:     y = 0x00 ;
32:     while(1) {
33:     if( y == 0x01) {
34:         Timer_4();
35:             }
36:         }
37:     }
38: }
39:
40: void Timer_4(){
41:
42:     TIMSK1 =  (1<<TOIE1);
43:     TCCR1B =
(1<<CS12)|(0<<CS11)|(1<<CS10);;
44:     cli();
45:     TCNT1H = 3035;
46:     TCNT1L = 3035;
47:     sei();
48:     PORTB=0x01;
49:     x= 0x01;
50: }
51:
52: ISR(INT1_vect) {
53:
54:     sei();
55:     if(x==0) {   Timer_4() ; }
56:     if (x==1) { Timer_05() ; }
57:    }
```

Ζήτημα 3.1.c:

Η έκδοση στην C δουλεύει το ίδιο όπου υπάρχει μια μεταβλητή x για να ελέγχεται αν είναι ή όχι η πρώτη φορά που έχει πατηθεί το κουμπί. Ανάλογα την τιμή τρέχει την συνάρτηση void Timer_4(); ή void Timer_05 (γραμμή 52). Ακριβώς όπως πριν η μεταβλητή **x** ελέγχει αν υπάρχει κάποιο led αναμμένο στο PORTB ώστε να ανοίξει όλα τα led για 0,5 sec ή απλώς να ανοίξει το PB1 για 4 sec. Ενώ η μεταβλητή y είναι αυτή που καθορίζει ότι θα μείνουν όλα τα led της PORTB ανοιχτά για μόλις 5 sec και έπειτα θα πάει στη συνάρτηση Timer_4().

```
59: int main()
60: {
61: //Interrupt on rising edge of INTO and INT1 pin
62:     EICRA= (1 << ISC11) | ( 1 << ISC10);
63: //Enable the INTO interrupt (PD2), INT1 interrupt
(PD3)
64:     EIMSK= (1 << INT1);
65:     sei ();
66:         DDRB=0xFF;
67:         DDRC=0xFF;
68:         PINC =0x00;
69:
70:     while(1){
71:
72:        if(PINC == 0x20){
73:              if(x==0) {   Timer_4() ; }
74:              else /* if (x==1) */ { Timer_05() ; }
75:            }
76:
77:     }
78:     return (EXIT_SUCCESS);
79: }
```

Όσον αφορά την main() , αφού ενεργοποιήσουμε τις διακοπές και τον TIMER_1 , μπαίνει σε loop μέχρι να πατηθεί το PD3 διακοπή ή να πατηθεί το PC5.

```
01: .include "m328PBdef.inc"
02: .equ FOSC_MHZ=16
03: .equ DEL_MS=500
04: .equ DEL_NU=FOSC_MHZ*DEL_MS
05:       clr r4
06:       clr r5
07:       clr r30
08:       clr r31
09:
10: .def temp = r20
11:       ser temp
12:       out DDRB, temp   // B output
13:       clr temp
14:       out DDRD, temp   // D input
15:
16:       ldi temp ,(1<<COM1A1) | (1<<WGM10)
17:       sts TCCR1A, temp
18:       ldi temp, (1<<CS11)|(1<WGM12)
19:       sts TCCR1B, temp
20:
21:       ldi zl, low(Table*2+12)      // zl =r30
22:       ldi zh, high(Table*2)      // zh = r31
23:
24:       lpm r4 , Z+  ; load lower byte of var into
r4
25:       lpm r5 , z   ; load upper byte of var into
r5
26:       sts OCR1Al , r4  // r4 = 0x80
27:       sts OCR1Ah,  r5  // r5 = 0x00 start from
50/100
28:       ldi temp, 0b01 //0b01000000
29:       out PORTB, temp
30:
31: main_:
32:       in temp, PIND
33:       com temp
34:       cpi temp, 2   // PD1
35:       breq add_
36:       cpi temp, 4 // PD2
37:       breq sub_
38:       rjmp  exit_
```

Ζήτημα 3.2.asm:

Στην άσκηση 2 χρησιμοποιώντας τον πίνακα στην γραμμή 82. Αρχικά ενεργοποιείτε το PWM με prescale 8 και διαμορφώνεται ο δείκτης z για να δείχνει την μέση του πίνακα για να έχει PWM 50% στην αρχή του προγράμματος (γραμμές 16-25). Έπειτα στην main_ ελέγχεται ποιο κουμπί έχει πατηθεί. Αν πατηθεί το PD1 κάνει άλμα στην add_. Όταν μπαίνει για πρώτη φορά στην add_ (και sub_) αρχικά ελέγχεται σε ποια θέση στην μνήμη δείχνει ο z. Στην περίπτωση της add_ όταν ο καταχωρητής r26 είναι ίσος με 0xDC τότε θα κάνει άλμα στην exit_, αν δεν βρίσκεται σε οριακή τιμή τότε αυξάνει τον δείκτη z κατά 2 κι φορτώνει την τιμή στον r4 και μετά στον OR1A (γραμμές 54-62) . Το ίδιο για την sub με διαφορά ότι μειώνετε ο z κατά 2. Στην περίπτωση που δεν πατηθεί κανένα κουμπί τότε γίνεται άλμα στην exit_.

```
40: sub_:
41:         ldi r26 , 0x0
42:         add r26 ,r4
43:         cpi r26 ,0x0005   // if is 2/100 stop dec dc
44:          breq exit_
45:          subi zl, 2
46:          lpm r4 , Z+ ;load lower byte of var into r4
47:          ldi r26 , 0x0
48:          add r26 ,r4
49:          cpi r26 ,0x0
50:          breq sub_
51:          sts OCR1Al, r4
52:           jmp exit_
53:
54: add_:
55:         ldi r26 , 0x0
56:         add r26 ,r4
57:        cpi r26 ,0x00DC   // if is 98/100 stop inc  dc
58:        breq exit_
59:        adiw zl , 1
60:         lpm r4 , Z+  ; load lower byte of var into r4
61:        sts OCR1Al, r4
62:        jmp exit_
63:
64: exit_:
65:         sts 0x0089,r5    // Store direct to data space
66:           sts 0x0088,r4
67:        ldi r24, low(DEL_NU) ;
68:        ldi r25, high(DEL_NU)
69:        rcall delay_ms
70:        rjmp main_
71:
72: delay_mS:
73:        ldi r23, 249
74: loop_inn:
75:        dec r23
76:        nop
77:        brne loop_inn
78:        sbiw r24, 1
79:        brne delay_mS
80:        ret
81:
82: Table: // 13 τιμές για το DC απο 2/100 εως 98/100 με βήμα  8/100
83: .DW 0x0005,0x0016,0x0028,0x003A,0x004C
84: .DW 0x005F,0x0080,0x0085,0x0094,0x00A6
85: .DW 0x00B8,0x00CA,0x00DC
```

File: ask2_lcd.c

```
01: #define F_CPU 16000000UL
02: #include <stdio.h>
03: #include <stdlib.h>
04: #include <avr/io.h>
05: #include <util/delay.h>
06:
07: int main(){
08:       unsigned char duty[13] = {5, 22, 40, 58, 76, 95, 128,
133, 148, 166, 184, 202, 220};
09:       TCCR1A = (1<<WGM10) | (1<<COM1A1);
10:       TCCR1B = (1<<WGM12) | (1<<CS11);
11:       DDRB = 0b00111111;
12:       DDRD = 0b00000000;
13:       int x, i=6;
14:       while(1)
15:       {
16:        x=PIND;
17:
18:       if((x==0b11111101) && (duty[i] != 220))
19:       {
20:             i++;
21:             _delay_ms(100);
22:       }
23:
24:       if((x == 0b11111011) && (duty[i] != 5))
25:       {
26:             i--;
27:             _delay_ms(100);
28:       }
29:                OCR1A = duty[i];
30:       }
31:
32:       return (EXIT_SUCCESS);
33: }
```

Ζήτημα 3.2.C:

Όπως ακριβώς στην asm αφού ορίσουμε έναν πίνακα με τις τιμές που θέλουμε να φορτώσουμε ώστε να μεταβάλετε η φωτεινότητα του PB2.

```
01: .include "m328PBdef.inc"
02: .equ FOSC_MHZ=16
03: .equ DEL_MS=500
04: .equ DEL_NU=FOSC_MHZ*DEL_MS
05: .org 0x00
06:     rjmp reset
07:         .def temp = r20
08:         .def duty = r21
09:         clr r28
10:
11:  reset:
12:         ser temp
13:         out DDRB, temp   // B output
14:         clr temp
15:     out DDRD, temp   // D input
16:         ldi temp ,(0<<WGM10) | (1<<WGM11) |
(1<<COM1A1)
17:         sts TCCR1A, temp
18:         ldi temp, (1<<WGM13)|(1<<WGM12) |
(0<<CS12) |   (1<<CS11) | (0<<CS10)
19:         sts TCCR1B, temp
20:         rjmp main_
21:
22: main_:
23:         in temp, PIND
24:         com temp
25:         cpi temp, 0b01  // PD0
26:         breq PWM_125
27:         cpi temp, 0b010 // PD1
28:         breq PWM_250
29:         cpi temp, 0b100 // PD2
30:         breq PWM_500
31:         cpi temp , 0b1000 //PD3
32:         breq PWM_1000
33:     ldi r24, HIGH(0)
34:     sts OCR1AH, r24
35:     ldi r24, LOW(0)
36:     sts OCR1AL, r24
37:         ldi r18 , LOW(0x0)
38:         ldi r19 , HIGH(0x0)
39:         sts ICR1L , r18
40:         sts ICR1H,  r19
41:         rjmp  main_
```

Ζήτημα 3.3.asm:

Η άσκηση 3 χρησιμοποιεί τον στην σελίδα 4 για να βρεθούν οι τιμές της ICR1A και OCR1A. Αρχικά ορίζεται το FAST PWM με prescaler 8 (γραμμες 16-19). Μετά ελέγχει άμα έχει πατηθεί κανένα κουμπί, στην περίπτωση που δεν έχει πατηθεί κανένα τότε ICR1A και OCR1A ίσο με 0. Αν πατηθεί το PD0 για παράδειγμα γίνεται άλμα στην PWM_125. Εκεί δίνεται στον καταχωρητή ICR1A και OCR1A οι τιμές 1600 και 8000 αντίστοιχα (γραμμές 43-52). Είναι σημαντικό να σημειωθεί πως όταν φορτώνονται οι τιμές στις ICR1A και OCR1A πρέπει να προσέχουμε όταν τις φορτώνουμε να μπαίνει πρώτα η τιμή της High κι μετά της Low για να μην υπάρχει πρόβλημα με την temp.Επίσης σημειώνουμε ότι κάθε φορά αλλάζουμε τους καταχωρητές OCR1A και ICR1A σύμφωνα με την σχέση

$$\frac{ICR1A}{OCR1A} = dc = \frac{50}{100}$$

Ακόμη για να υπολογίσουμε την του TOP για τις διάφορες τιμές του fpwm που ζητούνται λύνουμε:

$$fpwm = \frac{16 * 10^6}{8 * (1 + TOP)}$$

```
43: PWM_125:
44:     ldi r19 , HIGH(16000)
45:         sts ICR1H,  r19
46:         ldi r18 , LOW(16000)
47:         sts ICR1L , r18
48:         ldi r19, HIGH(8000)
49:         sts OCR1Ah , r19
50:         ldi r18 , LOW(8000)
51:         sts OCR1Al,  r18
52:         jmp main_
53:
54: PWM_250:
55:     ldi r19, HIGH(8000)
56:         ldi r18 , LOW(8000)
57:         sts OCR1Ah , r19  // r4 = 0x80
58:         sts OCR1Al,  r18 // r5 = 0x00 start from
50/100
59:         ldi r18 , LOW(4000)
60:         ldi r19 , HIGH(4000)
61:         sts ICR1L , r18
62:         sts ICR1H,  r19
63:         jmp main_
64:
65: PWM_500:
66:     ldi r19, HIGH(4000)
67:         ldi r18 , LOW(4000)
68:         sts OCR1Ah , r19  // r4 = 0x80
69:         sts OCR1Al,  r18 // r5 = 0x00 start from
50/100
70:         ldi r18 , LOW(2000)
71:         ldi r19 , HIGH(2000)
72:         sts ICR1L , r18
73:         sts ICR1H,  r19
74:         jmp main_
75:
76: PWM_1000:
77:     ldi r18 , LOW(1999)
78:         ldi r19, HIGH(1999)
79:         sts OCR1Al , r18  // r4 = 0x80
80:         sts OCR1Ah,  r19 // r5 = 0x00 start from
50/100
81:     ldi r18 , LOW(1000)
82:         ldi r19 , HIGH(1000)
83:         sts ICR1L , r18
84:         sts ICR1H,  r19
85:         jmp main_
```

```
01: #define F_CPU 16000000UL
02: #include <stdio.h>
03: #include <stdlib.h>
04: #include <avr/io.h>
05: #include <util/delay.h>
06:
07: int main(int argc, char** argv) {
08:     unsigned char duty = 128;
09:         TCCR1A = (1<<WGM11) |
(1<<COM1A1);
10:     DDRB |=(1<<PB1);
11:         DDRD = 0;
12:         TCCR1B = (1<<WGM13) | (1<<WGM12)
| (1<<CS11);
13:     int x;
14:         while(1){
15:
16:         x=PIND;
17:
18:         if(x == 0b11111110) // 125 Hz !!!!
19:         {
20:               ICR1 = 16000;
21:               OCR1A=8000;
22:         }
23:
24:         else if(x == 0b11111101) // 250 Hz
25:         {
26:               ICR1 = 8000;
27:               OCR1A= 4000;
28:         }
29:
30:         else if(x == 0b11111011) // 500 Hz !!!
31:         {
32:               ICR1 = 4000;
33:               OCR1A= 2000;
34:         }
35:
36:         else if(x == 0b11110111) //  1000 Hz
37:         {
38:                ICR1 = 2000;
39:               OCR1A= 1000;
40:         }
41:
42:         else
43:         {
44:                   ICR1 = 0;
45:                   OCR1A= 0;
46:         }
47:         }
48:     return (EXIT_SUCCESS);
49: }
```

Ζήτημα 3.3.C:

Εργαζόμαστε ακριβώς όπως και στην assemply.