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: .equ DEL_mS = 500
04: .equ DEL_NU = DEL_mS*FOSC_MHZ
05:
06: .org 0x0
07: rjmp reset
08: .org 0x4
09: rjmp ISR1
10:
11: reset:
12:     ldi r24, (1<<ISC11)|(1<<ISC10)
13:     sts EICRA, r24
14:     ldi r24, low(ramend)
15:    ;Stack pointer
16:     out SPL, r26
17:     ldi r26, high(ramend)
18:     out SPH,r26
19:     in r24, EIMSK
20:     ldi r24, (1<<INT1)
21:     out EIMSK, r24
22:     sei
23:
24: ;Init PortB-C as output
25:     ser r26
26:     out DDRB, r26
27:     out DDRC, r26
28:
29: loop1:
30:     clr r26
31: loop2:
32:     out PORTB, r26
33:     ldi r24, low(DEL_NU);
34:     ldi r25, high(DEL_NU)             ;Number of cycles
35: rcall delay_mS
36:     inc r26
37:     cpi r26, 16
38:     breq loop1
39: rjmp loop2
File: ask2.1.asm
41: delay_mS:
42:     ldi r23, 249
43: loop_inn:
44:     dec r23
45:     nop
46:     brne loop_inn
47:     sbiw r24, 1
48:     brne delay_mS
49: ret
```

Ζήτημα 2.1:

Αρχικά ενεργοποιούμε την διακοπή INT1 στην reset. Στη γραμμή 12 θέτουμε πότε θα ενεργοποιείτε η διακοπή INT1 και στην 21 επιτρέπουμε την INT1 να λειτουργεί στο πρόγραμμα. Έπειτα το πρόγραμμα τρέχει ολόιδια με το Σχήμα 2.1.

```
51: ISR1:
52:     push r24
53: start_:
54:     ldi r28, (1<<INTF1)
55:     out EIFR, r28
56:     ldi r24, low(DEL_NU);
57:     ldi r25, high(DEL_NU)     ;Number of cycles
58:     rcall delay_mS
59:     in r22, EIFR
60:     andi r22, 2
61:     brne start_
62:     in r24, PIND
63:     com r24
64:     sbrc r24,7
65:     rjmp exit
66:     inc r27
67:     sbrc r27, 6
68:     clr r27
69:     out PortC, r27
70: exit:
71:         pop r24
72: reti
```

Όταν πατηθεί το PD3 (που αντιστοιχεί στην INT1) το θα πρόγραμμα θα πάει στην 0x4 θέση στην μνήμη (γραμμή 9) κι θα κάνει άλμα στην ISR1. Εδώ κάθε φορά που μπαίνει ο επεξεργαστής θα αυξάνει την τιμή του καταχωρητή r27 κατά ένα (γραμμή 66). Στην περίπτωση που γίνει διακοπή κι ο r27 είναι αρχικά ίσος με 31 (0b11111) τότε θα γίνει 32 (0b10000) με την εντολή sbrc (γραμμή 56) δεν θα παραλείψει την γραμμή 57 αφού το 6ο bit 32 θα είναι άσσος οπότε ο r27 θα μηδενιστεί κι το μέτρημα θα αρχίσει από εκεί. Πατώντας το PD7 την εντολή sbrc (γραμμή 64) δεν θα παραλείψει την γραμμή 64 κι θα κάνει άλμα στο τέλος της ρουτίνας. Για να αποφύγουμε τον σπινθηρισμό μηδενίζουμε τον καταχωριτή EIFR για ένα μικρό χρονικό διάστημα (γραμμές 54

```
.include "m328PBdef.inc"

    .equ FOSC_MHZ=16
    .equ DEL_mS = 50
    .equ DEL_NU=FOSC_MHZ * DEL_mS

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

reset:
ldi r24 , (1 << ISC01) | (1 << ISC00)
sts EICRA , r24
ldi r24, (1<< INT0)
out EIMSK ,r24
ldi r24, (1<<INTF0)
out EIFR, r24
sei



  ldi r24,LOW(RAMEND)
  out SPL ,r24
  ldi r25,HIGH(RAMEND)
  out SPH ,r25
  clr r26
  out DDRB, r26
  ser r26
  out DDRC ,r26
  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 r23,120
     nop
     nop
     nop

   loop_inn:
   dec r23
   nop
   brne loop_inn
   sbiw r24 ,1
   brne delay_mS //
   ret
```

Ζήτημα 2.2:

Στο αρχικό κύματι του προγράμματος παίρνουμε το πρόγραμμα του μετρητή που είδαμε στις διαφάνειες και το τροποποιούμε τόσο στην σταθερά DEL_mS ώστε να έχουμε το επιθυμητό delay 600ms όσο και στον καταχωρητή r26 που πλέον του φορτώνουμε την τιμή 32 αφου ο το πρόγραμμα μας θέλουμε να μετράει από 0 εως 31.Ακόμη ενεργοποιούμε τις διακοπές (sei) με είσοδο διακοπής την INT0 δηλαδή το πάτημα του μπουτον PD2.

```
//////////////////////////////////////////
;ρουτίνα εξυπηρέτησης

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

  Do_It:
  clr r18
  in r20 ,PINB  ;ανάγνωση αριθμού απο την θύρα B
  ;com r20
  clr r30 ; Εδώ θα κρατάμε το πλήθος των μπουτον της PortB που ειναι πατημένα

  ldi r31, 8 //6
  ldi r28 , 1
  jmp loopX

 L2: inc r30
   nop
   nop
     dec r30
   breq CON
   clr r18
  out PORTC ,r18;PC
  ldi r18 ,1 //

  LEDON:
   dec r30
   breq CON
    lsl r18
   ori  r18 , 1
   jmp LEDON



 CON:
  out PORTC,r18
  ldi r24,low(DEL_NU)
  ldi r25,high(DEL_NU)
  rcall delay_mS

   pop r24
   out SREG ,r24
   pop r24
   pop r25
   sei
   reti







  loopX:
   mov r29 ,r20
   and  r29,r28 ;τσεκάρουμε ενα - ενα τα ψηφία απο LSB σε MSB
   cpi r29 , 0  ; εξτάζουμε αν έχουμε μονάδα σε αυτή την θέση ή οχι
  ; αν ισχύει το παραπάνω  r29==0  skip την επόμενη γραμμή
   breq L1
   inc r30

  L1:
   lsl r28 ; παμε μια θέση αριστερά την μονάδα στον r28
   dec r31
   breq  L2 ; Αν τσεκάρεις με όλα τα buttons του portB βγες απο το loop
   jmp loopX
```

Σκοπός είναι όταν γίνεται η διακοπή στην portC να ανάβουν τόσα led όσο και το πλήθος των μπουτον που ηταν πατημένα στο portB την στιγμή της διακοπής. Για παράδειγμα αν η portB έχει 00010011 τότε η portC πρέπει να εμφανίζει όσο διαρκεί η διακοπή στην έξοδο 00000111.Ο αλγόριθμος είναι απλός κοιτάζουμε ένα ένα τα 8 bit της θύρας B και αποθηκεύουμε το πλήθος από αυτά που είναι ανάμενα. Στη συνέχεια με την εντολή lsl στον βρόγχο LEDON , εκτελούμε επαναλήψεις όσο το πλήθος των led on του portB και βάζουμε κάθε φορά έναν 1 στο LSB και κάνουμε lsl .Στην περίπτωση που στην θύρα B δεν έχουμε πατημένα μπουτον δεν μπαίνουμε στον βρόγχο LEDON και απλώς στην θύρα C εμφανίζεται το δυαδικό μηδέν.

```
01: .include "m328PBdef.inc"
02:
03: .equ FOSC_MHZ = 16
04: .equ DEL_mS = 500
05: .equ DEL_NU = DEL_mS*FOSC_MHZ
06:
07: .org 0x0
08: rjmp reset
09: .org 0x4
10: rjmp ISR1
11:
12: reset:
13:     ldi r24, (1<<ISC11)|(1<<ISC10)
14:     sts EICRA, r24
15:     ldi r24, (1<<INT1)
16:     out EIMSK, r24
17:     ser r24
18:     out DDRB, r24
19:    clr r20
20:     sei
21: main:
22:     rjmp main
23:
24: delay_mS:
25:     ldi r23, 120
26:     nop
27:     nop
28: loop_inn:
29:     dec r23
30:     nop
31:     brne loop_inn
32:     sbiw r24, 1
33:     brne delay_mS
34: ret
```

Ζήτημα 2.3:

Ο σκοπός του κώδικα είναι να περιμένει να εντοπιστεί διακοπή INT1 κι να ανοίξει το LED PB0 για 4 sec. Αν μέχρι να τελειώσει η ρουτίνα της διακοπής πατηθεί ξανά η διακοπή INT1 (PD3) θα ανοίξουν όλα τα LED του PORTB για 500 ms κι έπειτα θα ανανεωθεί ο χρόνος που το PB0 είναι ανοιχτό. Το πρόγραμμα στα αριστερά ξεκινάει κι πάει στην reset στην οποία διαμορφώνει τα χαρακτηρίστηκα της INT1 (το ίδιο με την άσκηση 1) κι μετά μπαίνει στην main κι περιμένει

```
36: ISR1:
37:     ldi r28, (1<<INTF1)
38:     out EIFR, r28
39:     ldi r24, low(DEL_NU);
40:     ldi r25, high(DEL_NU)
41: rcall delay_mS
42:     in r22, EIFR
43:     andi r22, 2
44:     brne ISR1
45:
46:     cpi r20, 0
47:     breq start_
48:     sei
49:     ldi r20, 8
50:     ldi r24, low(DEL_NU)
51:     ldi r25, high(DEL_NU)
52:     ser r26
53:     out PORTB, r26
54:     rcall delay_mS
55:     ldi r30, 1
56:     out PORTB, r30
57:     reti
58: start_:
59:     ldi r20, 8
60: loop_:
61:     sei
62:     ldi r26, 1
63:     out PORTB, r26
64:     ldi r24, low(DEL_NU)
65:     ldi r25, high(DEL_NU)
66:     rcall delay_mS
67:     cli
68:     dec r20
69:     cpi r20, 0
70:     brne loop_
71:     clr r20
72:     out PORTB, r20
73: reti
```

Όταν πατηθεί το PD3 το πρόγραμμα θα κάνει την ρουτίνα ISR1. Για να αποφύγει τον σπινθηρισμό θα μηδενίσει τον καταχωρητή EIFR (γραμμή 37-38) για ένα μικρό χρονικό διάστημα κι μετά θα αρχίσει να ελέγχει ξανά για διακοπές. Στον καταχωρητής r20 αποθηκεύετε πόσες φορές έχει γίνει η loop_ ώστε το LED PB0 να είναι ανοιχτό για ακριβώς 4 sec. Στην γραμμή 20 με το cpi το πρόγραμμα ελέγχει άμα ο r20 είναι μεγαλύτερος του 0. Αν ισχύει αυτό θα σημαίνει πως το PB0 είναι είδη ανοιχτό κι πρέπει να το ανανεώσουμε αλλιώς κάνει άλμα στην start_. Η start_ δίνει στον r20 την τιμή 8. Έπειτα πάει στην loop_ όπου ανάβει την PB0 και περιμένει για 500 ms. Θα το κάνει αυτό 8 φορές (όσο η τιμή του r20) που αθροίζει σε 4 sec. Στην περίπτωση που πατηθεί το PD3 πριν τελειώσουν τα 4 sec τότε το breq στη γραμμή 47 δεν θα κάνει άλμα. Στην συνέχεια ανανεώσει τον r20, θα ανάψει όλα τα LED του PORTB κι θα περιμένει για 500 ms. Πριν τελειώσει την ρουτίνα θα αφήσει ανοιχτό μόνο το PB0 για να αποφύγει την περίπτωση που βγαίνει από την ρουτίνα κι γυρνάει στην αρχική μα βρίσκεται μέσα στο delay οπότε τα LED θα έμεναν ανοιχτά για περισσότερο από 500 ms

```
01: #define F_CPU  16000000UL
02: #include <stdio.h>
03: #include <stdlib.h>
04: #include<avr/io.h>
05: #include<avr/interrupt.h>
06: #include<util/delay.h>
07:
08: int x = 0;
09: ISR (INT1_vect)
10: {
11:     sei ();
12:     if( x > 0 )
13:     {
14:         x=0x4;
15:         PORTB=0xFF;
16:         _delay_ms(500);
17:         exit(0);
18:     }
19:     x=0x4;
20:     while(x > 0){
21:         PORTB=0x01;
22:        _delay_ms(250);
23:         x-=0x1;
24:     }
25:     PORTB=0x00;
26: }
27:
28: int main(int argc, char** argv)
29: {
30: //Interrupt on rising edge of INTO and INT1
pin
31:     EICRA= (1 << ISC11) | ( 1 << ISC10);
32: //Enable the INTO interrupt (PD2), INT1
interrupt (PD3)
33:     EIMSK= (1 << INT1);
34:     sei ();
35:     DDRB=0xFF;
36:        DDRC=0xFF;
37:        PORTC=0x00;
38:     while (1)
39:     {
40:         PORTB=0x00;
41:         _delay_ms(1);
42:         PORTB=0x00;
43:     }
44:     return (EXIT_SUCCESS);
45: }
```

Στο C κομμάτι της άσκησης πάλι ορίζονται οι ιδιότητες του INT1 (γραμμές 31-33) κι μετά περιμένει στην while (γραμμή 38). Αφού πατηθεί το PD3 τρέχει την συνάρτηση ISR (γραμμή 9) αντί για r20 εδώ χρησιμοποιείτε ο x όπου είναι global μεταβλητή ώστε να μπορούν να την χρησιμοποιούν όλες οι διακοπές. Αν το x = 0 του δίνεται η τιμή 4 (γραμμή 19) κι ανάβει το PB0 πάλι για 4 sec. Πατώντας το PD3 πριν τελειώσουν τα 4 sec το if (γραμμή 12) θα δει ότι το x δεν είναι 0 και θα ανανεώσει το x, θα ανοίξει όλα τα LED του PORTB για 500 ms και θα κάνει exit(γραμμές 14-17)