ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εργαστήριο Μικροϋπολογιστών

$2^{η}$ εργαστηριακή άσκηση

Μικροελεγκτής AVR

Αναστάσιος Λαγός - 03113531
Αντώνιος Δημήτριος Αλικάρης - 03118062

Άσκηση 1

Γενική Ιδέα

Αφού διαβάσουμε τα 4 LSB της θύρας C και αποθηκεύοντας το καθένα σε έναν ξεχωριστό καταχωρητή, τα μεταφέρουμε όλα στο LSB και εκτελούμε διαδοχικά τις αντίστοιχες λογικές πράξεις. Τέλος εμφανίζουμε το αποτέλεσμα στην έξοδο.

Κώδικας σε assembly

```
.org 0x0
rjmp reset

reset:
        ; set r24 to 0x00
        clr r24
        ; set PORTC as input
        out DDRC , r24
        ; set r24 to 0xFF
        ser r24
        ;set PORTB as output
        out DDRB , r24

main:
        ;r16 = F0
        ;r17 = F1
        ;r18 = A
        ;r19 = B
        ;r20 = C
        ;r21 = D

        in r18 , PINC          ;get the input
        andi r18, 0x01         ;get PC0

        in r19, PINC             ;get the input
        andi r19, 0x02          ;get PC1
        lsr r19                 ;shift right

        in r20, PINC
        andi r20, 0x04
        lsr r20
        lsr r20

        in r21, PINC
        andi r21, 0x08
        lsr r21
        lsr r21
        lsr r21

```

```
        ; F0 = (A'B + B'CD)'

        mov r24, r18 ; r24 <- A
        com r24 ; r24 <- A'
        and r24, r19 ; r24 <- A'B
        mov r16, r24 ; F0 <- A'B

        mov r24, r19 ; r24 <- B
        com r24                 ;r24 <- B'
        and r24, r20        ; r24 <- B'C
        and r24, r21        ; r24 <- B'CD

        or r16, r24                 ; F0 <- (A'B + B'CD)
        com r16                         ; F0 <- (A'B + B'CD)'
        andi r16, 0x01        ; get the lsb

        ; F1 = (AC)(B + D)

        mov r24, r18        ;r24 <- A
        and r24, r20        ; r24 <- AC
        mov r17, r24        ; F1 <- AC

        mov r24, r19        ;r24 <- B
        or r24, r21                ;r24 <- B + D

        and r17, r24        ; F1 <- (AC)(B + D)
        andi r17, 0x01        ; get the lsb
        lsl r17                         ; shift left

        or r16, r17
        out PORTB, r16        ;show on LEDs

        rjmp main

```

Κώδικας σε c

```c
#include <avr/io.h>

//Initialize variables
char F0,F1,A,B,C,D;

int main() {

        //Set PORTC as input
        DDRC = 0x00;
        //Set PORTB as output
```

```
        DDRB = 0xFF;

        while(1) {

                A = PINC & 0x01;                //Get PC0
                B = (PINC & 0x02) >> 1; //Get PC1 and shift to LSB
                C = (PINC & 0x04) >> 2;         //Get PC2 and shift to LSB
                D = (PINC & 0x08) >> 3;         //Get PC3 and shift to LSB

                //Perform bitwise operation for F0 = (A'B + B'CD)'
                F0 = ~((~A & B) | (~B & C & D));
                //Keep the LSB
                F0 = F0 & 0x01;

                //Perform bitwise operation for F1 = (AC)(B + D)
                F1 = (A & C) & (B | D);
                //Keep the LSB
                F1 = F1 & 0x01;
                //Shift one to the left
                F1 = F1 << 1;
                //Merge F0-F1 and output
                PORTB = F0 | F1;

        }


}

```

Άσκηση 2

Γενική ιδέα

Τρέχουμε το πρόγραμμα μετρητή και μόλις γίνει διακοπή INT1 το πρόγραμμα πηγαίνει στην ρουτίνα εξυπηρέτησης της διακοπής, όπου ελέγχει αν τα dip switches A7 και A6 είναι ON, τότε αυξάνει τον καταχωρητή που μετράει τις διακοπές και βγάζει το αποτέλεσμά του στην έξοδο.

Κώδικας

```
.org 0x0        ;η αρχή του κώδικα reset
rjmp reset
.org 0x4        ;η εξυπηρέτηση της INT1
rjmp ISR1

reset:
        ldi r24,(1<<ISC11)|(1<<ISC10)
        out MCUCR,r24        ;η διακοπή INT1 να προκαλείται με σήμα θετικής
        ↪ ακμής
```

```
        ldi r24,(1<<INT1)
        out GICR,r24        ;ενεργοποίηση διακοπής INT1
        sei
        ldi r24,low(RAMEND)
        out SPL,r24
        ldi r24,high(RAMEND)
        out SPH,r24


;counting programm given
start:
    clr r26
        out DDRA,r26        ;Αρχικοποίηση της PORTA για είσοδο
        ser r26
        out DDRC,r26        ;Αρχικοποίηση της PORTC για έξοδο
        out PORTA,r26        ;ενεργοποίηση αντιστάσεων πρόσδεσης
        clr r16
        clr r26

loop:
        out PORTC,r26        ;δείξε την τιμή του μετρητή στην θύρα εξόδου των
        ↪ LED
        inc r26
        rjmp loop

ISR1:
        push r26        ;σώζει το περιεχόμενο του r26
        in r26,SREG        ;σώζει το περιεχόμενο των sreg
        push r26
        ser r26
        out DDRB,r26        ;η θύρα B ως έξοδος
        in r26,PINA                ;διαβάζει την A
        subi r26,192
        brlo noincr        ;αν δεν είναι 1 τα PA7 και PA6 πήγαινε στο noincr
        inc r16                        ;αλλιώς αύξησε τον μετρητή
        out PORTB,r16        ;και βγάλτον στην έξοδο
noincr:
        pop r26
        out SREG,r26        ;επανάφερε την τιμη των sreg
        pop r26                        ;και του r26
        reti                        ; και επίστρεψε
```

Άσκηση 3

Γενική Ιδέα

Το πρόγραμμα είναι στο άπειρο βρόχο και περιμένει διακοπή από τον χρήστη. Μόλις ανιχνεύσει διακοπή πηγαίνει στην ρουτίνα εξυπηρέτησης της διακοπής και αφού ελέγξει αν το PA2 είναι ON τότε μετράει πόσα switches είναι ανοιχτά στην Θύρα B και ανάβει στην έξοδο τόσους διακόπτες σύμφωνα με την εξίσωση $n = 2^{count} - 1$. Αν το PA2 είναι OFF τότε στην έξοδο βγάζει τον δυαδικό αριθμό των ανοιχτών switches της θύρας B.

Κώδικας

```
#include <avr/io.h>
#include <avr/interrupt.h>

volatile int check, count, v;
volatile int bit_count(volatile int v);


#define NOP(){__asm__ __volatile__("nop");} //does nothing. needed inside
↪  while(1) compiler optimizations skip it

// INT0 interrupt service routine
ISR(INT0_vect) {
        check = PINA & 0x04;        //get PA2
        if (check == 0x00) {        //if not set
                v = PINB;
                count = bit_count(v); //count the number of set bits
                PORTC = (1 << count) - 1; //light up 'count' # of LEDs
        }
        else {
                v = PINB;
                count = bit_count(v);
                PORTC = count;        //output the number of set bits
        }
        return;
}

// function that counts the number of set bits in a byte
volatile int bit_count(volatile int v) {
        volatile int count = 0x00;
        int hex_lsb;
        int loop_count = 7;
        while(loop_count >= 0) {
                hex_lsb = v & 0x01;
                if (hex_lsb == 0x01) {
                        count++;
                }
                v = v >> 1;
```

```
37                  loop_count--;
38          }
39          return count;
40  }
41
42
43  int main(void) {
44
45          DDRA, DDRB, DDRD = 0x00; // make PORTA, PORTB, PORTD as inputs
46          DDRC = 0xff;                            // make PORTC as output
47          //disable global interrupts
48          cli();
49
50          GICR = 0x40;          //enable INT0
51          MCUCR = 0x03;          // enable on rising edge
52          //enable global interrupts
53          sei();
54
55          while(1) {NOP();}
56  }
```