Εργαστήριο Μικροϋπολογιστών

2η Εργαστηριακή Άσκηση

Παναγιώτης Ντάγκας el18018

Βικέντιος Βιτάλης el18803

Ζήτημα 2.1

Assembly Code:

```
.include "m16def.inc"

start:
    ser r24                 ;Αρχικοποίηση της PORTB
    out DDRB,r24            ;Θέτουμε την B ως έξοδο
    clr r24
    out DDRC,r24            ;θέτουμε την C ως είσοδο

    in r25,PINC             ;Διαβάζουμε την είσοδο
    mov r26,r25
    com r25
    andi r25,0x01           ;Στο LSB του r25 κρατάμε την μεταβλητή A’
    mov r24,r26
    andi r24,0x02           ;Στο LSB του r24 κρατάμε την μεταβλητή B
    lsr r24
    and r25,r24             ;Αποθήκευση του A'B στον καταχωρητή r25
    com r24
    andi r24,0x01           ;Στο LSB του r24 κρατάμε την μεταβλητή B'
    mov r23,r26
    andi r23,0x04           ;Στο LSB του r23 κρατάμε την μεταβλητή C
    lsr r23
    lsr r23
    and r24,r23             ;Αποθηκεύουμε το B'C στον καταχωρητή r24
    mov r23,r26
    andi r23,0x08           ;Στο LSB του r23 κρατάμε την μεταβλητή D
    lsr r23
    lsr r23
    lsr r23
    and r24,r23             ;Αποθηκεύουμε το B'CD στον καταχωρητή r24
    or r25,r24              ;Αποθηκεύουμε το A'B+B'CD στον καταχωρητή r25
    com r25
    andi r25,0x01           ;Αποθηκεύουμε το F0 res στο LSB του καταχωρητή r25
    mov r22,r25

    mov r25,r26
    andi r25,0x01           ;Στο LSB του r25 κρατάμε την μεταβλητή A
    mov r24,r26
    andi r24,0x04           ;Στο LSB του r24 κρατάμε την μεταβλητή C
    lsr r24
    lsr r24
    and r25,r24             ;Αποθηκεύουμε το AC στον καταχωρητή r25
```

```
    mov r24,r26
    andi r24,0x02          ;Στο LSB του r24 κρατάμε τη μεταβλητή B
    lsr r24
    mov r23,r26
    andi r23,0x08          ;Στο LSB του r23 κρατάμε τη μεταβλητή D
    lsr r23
    lsr r23
    lsr r23
    or r24,r23             ;Αποθηκεύουμε το B+D στον r24
    and r25,r24            ;Αποθηκεύουμε το (AC)(B+D) στον r25
    lsl r25                ;Κρατάμε το F1 στο δεύτερο LSB του r25
    or r25,r22             ;000000F1F0:Στον r25 το F0 στο LSB και F1 στο 2ο LSB
    out PORTB,r25          ;Εμφανίζουμε τα αποτελέσματα στην έξοδο
    rjmp start             ;Επανεκκίνηση του προγράμματος
```

Οι εντολές lsr/lsl αποτελούν λογικές ολισθήσεις δεξιά/αριστερά αντίστοιχα.

C Code:

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

char F0, F1, x, A, B, C, D; //Ορίζουμε τις 8bit μεταβλητές που χρησιμοποιούμε

int main(void)
{
       DDRB = 0xFF; //Ορίζουμε την portB ως έξοδο
       DDRC = 0x00; //Ορίζουμε την portC ως είσοδο
    while (1)
    {
              x = PINC; //Για κάθε μεταβλητή char που ορίσαμε, κρατάμε στο LSB
της, την αντίστοιχη μεταβλητή της εισόδου, κάνοντας κάθε φορά τις απαραίτητες
ολισθήσεις για να φτάσει στο LSB
              A = x & 0x01;
              B = (x & 0x02) >> 1;
              C = (x & 0x04) >> 2;
              D = (x & 0x08) >> 3;

              F0 = ~((~A&B) | (~B&C&D)) & 0x01; //Κρατάμε το αποτέλεσμα της
//F0 = (A'B + B'CD)' στο LSB της μεταβλητής F0
              F1 = (A&C)&(B|D) & 0x01; //Κρατάμε το αποτέλεσμα της
//F1 = (AC)(B+D) στο LSB της μεταβλητής F1

              F1 = (F1 << 1)|F0; //Κρατάμε το αποτέλεσμα της F1 στο 2ο LSB της
//μεταβλητής F1
              PORTB = F1; //Εμφανίζουμε το αποτέλεσμα στην έξοδο
    }
}
```

Ζήτημα 2.2

```
.include "m16def.inc"

.org 0x0                    ;Ορίζουμε το σημείο εκκίνησης του προγράμματος
rjmp start
.org 0x4                    ;Θέση μνήμης ρουτίνας εξυπηρέτησης
rjmp ISR1                   ;Για τη διακοπή INT1 θα γίνει άλμα στην ετικέτα
                            ;ISR1


start:
       ldi r21, LOW(RAMEND)  ;Κύριο πρόγραμμα
       out SPL, r21          ;Θέτουμε δείκτη στοίβας στην RAM
       ldi r21, HIGH(RAMEND)
       out SPH, r21
       clr r20              ;Μετρητής για διακοπές
       ldi r24, (1 << ISC11)|(1 << ISC10)
       out MCUCR, r24       ;Διακοπή INT1 όταν το σήμα έχει ανερχόμενη ακμή
       ldi r24, (1 << INT1)
       out GICR, r24        ;INT1 να προκαλεί διακοπές. Οι υπόλοιπες αγνοούνται

       sei                  ;Ενεργοποίηση διακοπών

       clr r26              ;Ο r26 αρχικοποιείται στην τιμή 00000000

       out DDRA, r26        ;Αρχικοποίηση της PORTA ως έξοδο (του βασικού
                            ;μετρητή)
       ser r26
       out DDRC, r26        ;Αρχικοποίηση της PORTC ως έξοδο(μετρητής εξόδου)
       clr r26
       out PORTC, r26
loop:
       out PORTC, r26       ;Δείξε την τιμή του μετρητή r26 στη θύρα εξόδου LED
       ldi r24, low(100)    ;Φόρτωσε τον r25:r24 με το 100
       ldi r25, high(100)
       ;rcall wait_msec     ;Καλείται η ρουτίνα χρονοκαθυστέρησης 1msec
       inc r26              ;Αύξησε τον βασικό μετρητή r26 κατά 1
       rjmp loop            ;Επανέλαβε

ISR1:                       ;Εδώ κάνει άλμα η ρουτίνα εξυπηρέτησης INT1
        in r23, PINA
        andi r23, 0xC0      ;Απομονώνουμε τα PA7 & PA6 (0xC0=192 δεκαδικό)
        cpi r23, 0xC0       ;Αν PA7 != 1 ή PA6 != 1 δεν κάνει τίποτα
        brne return
        inc r20             ;Αύξησε τον μετρητή r20 κατά 1
        out PORTB, r20      ;Εμφάνιση αποτελέσματος στην έξοδο PORTB
return:
        reti                ;Επαναφορά του PC στο σημείο πριν της διακοπής
```

Ζήτημα 2.3

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

unsigned char x, y, counter, dips; //Ορίζουμε τις 8bit μεταβλητές
ISR (INT0_vect) {                  //Ρουτίνα εξυπηρέτησης για την INT0
       dips = 0x00;                //Πλήθος διακοπτών που είναι ON
       x = PINA & 0x04;            //Η μεταβλητή x αποθηκεύει την είσοδο PINA
       //και απομονώνεται το PA2
       y = PINB;                   //Η μεταβλητή y αποθηκεύει την είσοδο PINB
       while (y!=0) {
              counter += y & 0x01; //Σέ κάθε επανάληψη απομονώνουμε το LSB του
//τωρινού y
              y = y>>1;            //Αν αυτό είναι ON, τότε ο μετρητής αυξάνεται
//κατά 1, αλλιώς μένει ίδιος στη συνέχεια το y ολισθαίνεται αριστερά κατά μία
//θέση οπότε κάθε φορά που μετράμε άσσο αυτός χάνεται, αφού με την ολίσθηση
//αντικαθίσταται από ένα 0 στο MSB. Έτσι, όταν θα έχουν μετρηθεί όλοι οι άσσοι
//του y αυτό θα γίνει 0 και ο βρόχος θα σταματήσει.
       }
       if (x == 0x00)       { //Αν το x=0, δηλαδή αν το PA2 είναι OFF
              while(counter > 0) { //Για όσες φορές όσοι και οι άσσοι
                     dips = dips<<1; //Ολίσθηση δεξιά κατά μια θέση
                     dips = dips + 0x01; //Προσθήκη νέου άσσου στο LSB
                     counter -= 1; //Μείωση μετρητή κατά 1
              }
       }
       else { //Αν το PA2 είναι ON
              dips = counter; //Εμφάνισε τον μετρητή των άσσων στην έξοδο
       }
       PORTC = dips; //Εμφανίζουμε το αποτέλεσμα στην θύρα C
}

int main(void)
{
       DDRC = 0xFF;         //Θέτουμε την θύρα C ως έξοδο
       DDRB = 0x00;         //Θέτουμε την θύρα B ως είσοδο
       DDRA = 0x00;         //Θέτουμε την θύρα A ως είσοδο
       GICR = (1<<INT0);    //Μόνο το σήμα INT0 προκαλεί διακοπές, αγνόηση των
//υπόλοιπων
       MCUCR = (1<<ISC00) | (1<<ISC01); //Προκαλείται διακοπή μόνο όταν το
//σήμα έχει ανερχόμενη ακμή(πάτημα κουμπιού)
       asm("sei");   //Ενεργοποίηση διακοπών
    while (1)
    {
              asm("nop"); //Αναμονή για διακοπή
    }
}
```

Είναι σημαντικό να αναφέρουμε πως σε αυτή τη σειρά έγινε χρήση του Μικροελεγκτή AVR ATmega16. Σε αυτόν τον Μικροελεγκτή υπάρχουν δύο βασικές σημαίες:

(1)Η επιλογή του επιπέδου ενεργοποίησης διακοπής

(2)Η επίτρεψη της επιθυμητής εισόδου διακοπής

Η πρώτη ενεργοποιείται γράφοντας στον καταχωρητή MCUCR (διεύθυνση $35) και στα τέσσερα λιγότερα σημαντικά ψηφία κατάλληλες τιμές. Η δεύτερη ενεργοποιείται γράφοντας στον καταχωρητή GICR (διεύθυνση $3B) την τιμή 1 στο ψηφίο που αντιστοιχεί στην είσοδο διακοπής που επιθυμούμε να επιτρέψουμε.

Για τις διακοπές οφείλουμε να αναφέρουμε επίσης πως ανάλογα την ζητούμενη διακοπή (INT0,INT1) και την επιθυμητή ακμή της διακοπής, την ρυθμίζουμε με την βοήθεια των εντολών ISC00,ISC01,ISC10,ISC11. Τέλος, η διεύθυνση της INT0 είναι η 0x2 και η διεύθυνση της INT1 η 0x4.