4η Ομάδα Ασκήσεων

AVR

Μάθημα Συστήματα Μικροεπεξεργαστών

Ακ. Έτος *2019-2020 (Εαρινό Εξάμηνο)*

Πανεπιστήμιο *Εθνικό Μετσόβιο Πολυτεχνείο(NTUA)*

Σχολή: *Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών (ECE)*

Ημερομηνία *7/06/2020*

Στοιχεία Φοιτητή:			
Όνομα:	*Στέφανος-Σταμάτης*	*Αγγελική -Διονυσία*	*Δημήτριος*
Επώνυμο	*Αχλάτης*	*Γαρουφαλιά*	*Γκέγκας*
Εξάμηνο	*8ο*	*8ο*	*8ο*
Α.Μ.	*03116149*	*03116152*	*03116004 / el16004*
e-mail:	*sachlatis@gmail.com*	*angelikigaroufalia@gmail.com*	*dimitrisgegas01@gmail.com*

Ομαδική Εργασία

Άσκηση 1

Παρακάτω παρατίθεται ο κώδικας όπως ακριβώς εκτελέστηκε στον προσομοιωτή.

```
.include "m16def.inc"
reset:
    ser r24
    out DDRB , r24  ; initialize PORTB for output
    clr r24
    out DDRC, r24   ; initialize PORTC for input
    ldi r26, 1  ; αρχικοποιούμε τον r26 με 1

increase:   ; αριστερή ολίσθηση bit
    in r27, PINC    ; φόρτωση εισόδου και συνθήκη ελέγχου του push button PC2
    ror r27 ; με δεξια ολίσθηση της εισόδου και έλεγχος του
    ror r27
    ror r27
    brcs increase   ; κρατουμένου που αντιστοιχεί στο LSB δηλαδή το PC2
    out PORTB, r26  ; εμφάνιση εξόδου στο PORTB
    ;rcall wait_msec ;για την προσομοίωση
    lsl r26     ;αριστερή ολίσθηση
    cpi r26, 128    ;έλεγχος αν είναι μικρότερος του 128
    brlo increase   ; αν ναι, τότε πάμε στο increase
    rjmp decrease   ; αν όχι, πάμε στο decrease

decrease:   ;αντίστοιχα για δεξιά ολίσθηση του bit
    in r27, PINC
    ror r27
    ror r27
    ror r27
    brcs decrease
    out PORTB, r26
    lsr r26
    cpi r26,2   ; έλεγχος αν είναι μεγαλύτερο ή ίσο του 2 επειδή τυπώνω στην επομενη φαση
    brge decrease ; αν ειναι μεγαλυτερος ή ισος του δυο μενω στο ντικρισ
    rjmp increase ; αν οχι, δλδ αν ειναι μικρότερο του 2 πηγαινε στο ντικρις
```

Σχόλια πάνω στον κώδικα:

Αρχικά ο προσομοιωτής δουλεύει για μια ευρεία κατηγορία AVR και γι αυτό στην αρχή πρέπει να προσθέσουμε το κατάλληλο include έτσι ώστε να δουλέψουμε για τον AVR του εργαστηρίου. Ορισμένες διαφορές μεταξύ αυτών των AVR είναι οι θέσεις εξόδου, διαφορα keywords και άλλα, έτσι τοποθετούμε το .include “m16def.inc” για να δουλέψουμε στον AVR του εργαστηρίου.

Στην αρχή κάνουμε κάποιες απαραίτητες αρχικοποιήσεις. Πιο αναλυτικα παίρνουμε τον καταχωρητή r24 (8-bits) και τον γεμίζουμε με 8 μηδενικά για να το “δεσουμε” με τον DDRB έτσι ώστε και τα 8 bit της θύρας B να προγραμματιστούν ως ακροδέκτες εξόδου. Στην συνέχεια, παίρνουμε τον καταχωρητή r24 (8-bits) και τον γεμίζουμε με 8 άσσους για να το “δέσουμε” με τον DDRC έτσι ώστε και τα 8 bit της θύρας C να προγραμματιστούν ως ακροδέκτες εισόδου, βέβαια εμάς μας ενδιαφέρει μόνο ο τρίτος ακροδέκτης να είναι για είσοδο αλλά δεν μας πειράζει να έχουμε περισσότερους. Στην συνέχεια φορτώνουμε τον αριθμό 1 δηλαδή το 00000001 στον καταχωρητή r26, ο οποίος θα είναι “δείχνει” πιο led θα είναι ανοιχτό κάθε φορά. Φορτώνουμε τον αριθμό 00000001 έτσι ώστε την πρώτη φορά που θα δείξει τα led για να ανοίξουν θα ανοίξει το lsb led, δηλαδή το πρώτο.

Μετά προχωράμε στο κομμάτι “increase” όπου αρχικά ελέγχουμε αν έχει πατηθεί το τρίτο κουμπί της θύρας C, το PC2. Αν έχει πατηθεί μένουμε σε μια λούπα όπου ελέγχουμε συνέχεια αν το κουμπί αυτό έχει πατηθεί. Όταν το κουμπί αυτό αφεθεί προχωράμε και φορτώνουμε το περιεχόμενο του r26 στην θύρα B και τότε ανάβει το led που αναδεικνύει το περιεχόμενο του καταχωρητή r26.Στην συνέχεια κάνουμε αριστερή ολίσθηση στον καταχωρητή r26 όπου ουσιαστικά μεταφέρει τον ασσο μια θέση αριστερότερα, έτσι ώστε την επόμενη φορά να ανοίξει το αμέσως αριστερότερο led. Έπειτα κάνουμε έλεγχο αν το r26 έχει τον αριθμό 128 δηλαδή το 10000000, αν ναι τότε πάμε στην διαδικασία decrease αλλιώς ξαναμπαίνουμε στο increase.
Όταν μπούμε στο decrease αρχικά ελέγχουμε αν έχει πατηθεί το τρίτο κουμπί της θύρας C, το PC2. Αν έχει πατηθεί μένουμε σε μια λούπα όπου ελέγχουμε συνέχεια αν το κουμπί αυτό έχει πατηθεί. Όταν το κουμπί αυτό αφεθεί προχωράμε και φορτώνουμε το περιεχόμενο του r26 στην θύρα B και τότε ανάβει το led που αναδεικνύει το περιεχόμενο του καταχωρητή r26. Στην συνέχεια κάνουμε δεξιά ολίσθηση στον καταχωρητή r26 όπου ουσιαστικά μεταφέρει τον ασσο μια θέση δεξιότερα, έτσι ώστε την επόμενη φορά να ανοίξει το αμέσως δεξιότερο led. Έπειτα κάνουμε έλεγχο αν το r26 έχει τον αριθμό 2 δηλαδή το 00000010, αν ναι τότε πάμε στην διαδικασία increase αλλιώς ξαναμπαίνουμε στο increase.

Αξίζει να σημειωθεί ότι στο πρόγραμμα μας δεν έχουμε χρησιμοποιήει καθόλου διαδικασία χρονοκαθυστέρησης που θα απαιτούσε και την ύπαρξη stack. Ο λόγος που έγινε αυτό είναι επειδή τρέξαμε το πρόγραμμα σε προσομοιωτή αν το τρέχαμε στο εργαστήριο θα έπρεπε να βάζαμε μια χρονοκαθυστέρηση έτσι ώστε ο χρήστης να έβλεπε το αποτέλεσμα του AVR.

Άσκηση 2

```
#include <avr/io.h>
int main(void)
{
   DDRB=0x00;            //as input
   DDRA=0xFF;            //as output
   while (1)
   {
      char IN= PINB;                    //reading input
      char A=(IN >> 0) & 1;             //getting from 8bits input the A,B,C,D
      char B=(IN >> 1) & 1;             //masking with the 0x01 after shifting the
      char C=(IN >> 2) & 1;             //bits to get the nth bit at the LSB position
      char D=(IN >> 3) & 1;

      char F0 = ~((A & (~B) )| (B & (~C) & D));   //creating the output logic function
      F0=F0 &1;                         //only the LSB
      char F1 =((A|C)&(B|D));
      F1=F1 &1;

      PORTA=F0 +(F1<<1);    //export output at portA with F0 in LSB and F1 in the 2nd LSB
   }
}
```

Το παραπάνω πρόγραμμα προσομοιώνει ένα λογικό κύκλωμα με 4 εισόδους, A,B,C,D όπου εισάγονται από τα 4LSB της θύρας B και 2 εξόδους , F0,F1 που εξάγονται στα 2LSB της θύρας A.

Άσκηση 3

Ακολουθεί αρχικά ο κώδικας της άσκησης σε C. Οι σημειωμένες γραμμές, αντιστοιχούν στα σημεία που έχουν μπει breakpoints.

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

char x;

int main(void)
{
   DDRB = 0xFF;          //Θέτουμε την έξοδο στο PORTB.
   DDRA = 0x00;          //Θέτουμε την είσοδο στο PORTA

   x = 1;               //Αρχικά ανάβει το led0

   while(1) {
        if((PINA & 0x01) == 1){      //Έλεγχος πατήματος SW0
            while((PINA & 0x01) == 1); //Έλεγχος επαναφοράς SW0
            if(x==1)
                x = 128;             //Έλεγχος υπερχείλισης
            else x = x >> 1;
        }

        if((PINA & 0x02) == 2){        //Έλεγχος πατήματος SW1
            while((PINA & 0x02) == 2);  //Έλεγχος επαναφοράς SW1
            if(x==128)
                x = 1;               //Έλεγχος υπερχείλισης
            else x = x << 1;
        }

        if((PINA & 0x04) == 4){        //Έλεγχος πατήματος SW2
            while((PINA & 0x04) == 4);  //Έλεγχος επαναφοράς SW2
            x = 1;           //Μετακίνηση αναμμένου LED στο led0 (lsb)
        }

        if((PINA & 0x08) == 8){        //Έλεγχος πατήματος SW3
            while((PINA & 0x08) == 8);  //Έλεγχος επαναφοράς SW3
            x = 128;         //Μετακίνηση αναμμένου LED στο led7 (msb)
        }

        PORTB = x;      //Έξοδος στην PORTB
   }
   return 0;
}
```

Περιγραφή Κώδικα

Ως θύρα εξόδου ορίζουμε την PORT B, ενώ ως θύρα εισόδου την PORT A.
Αρχικά, ανάβει το led0 που είναι συνδεδεμένο με το bit0 της θύρας εξόδου B.
Τα Push Buttons αντιστοιχούν στα pin της θύρας A, ως εξής:
SW0 -> pinA0
SW1 -> pinA1
SW2 -> pinA2
SW3 -> pinA3
Κάθε φορά είναι αναμμένο ένα led, καθώς επίσης υποθέτουμε ότι τα Push Buttons δεν θα είναι πατημένα ταυτόχρονα, αλλά πατιέται ένα κάθε φορά.

Ζητούμενο της άσκησης είναι, επίσης, οι αλλαγές στην κατάσταση των led να γίνονται κατά την επαναφορά του εκάστοτε διακόπτη. Αυτή τη λειτουργία εξυπηρετούν οι εντολές "*while((PINA & 0x08) == 8);",* έτσι ώστε το πρόγραμμα να αναμένει την επαναφορά του διακόπτη πριν συνεχίσει τη λειτουργία του παρακάτω στον κώδικα.

Όσον αφορά τη λειτουργία του κάθε διακόπτη, αυτές έχουν ως εξής:
SW0: Το αναμμένο led ολισθαίνει μία θέση δεξιά με την εντολή "x = x >> 1;". Πριν εκτελεστεί αυτή η εντολή, υπάρχει πάντα έλεγχος για το αν το αναμμένο led αντιστοιχεί στο bit0 (x==1), οπότε αντί για δεξιά ολίσθηση θα πραγματοποιηθεί η ανάθεση "x = 128;" ώστε να ανάψει το led στο msb, αφού η κίνηση πρέπει να είναι κυκλική σύμφωνα με την εκφώνηση.
SW1: Το αναμμένο led ολισθαίνει μια θέση αριστερά με την εντολή "x = x << 1;". Πριν εκτελεστεί η εντολή, και πάλι υπάρχει ο αντίστοιχος έλεγχος με πριν, απλά αυτή τη φορά ελέγχεται αν το αναμμένο led είναι στη θέση msb (x==128), οπότε και μπαίνει η ανάθεση "x=1;", για να ανάψει το led0.
SW2: Με την επαναφορά αυτού του διακόπτη, το αναμμένο led από όπου και αν βρίσκεται, μετακινείται στη θέση του led0, δηλαδή στο lsb.
SW3: Σε αυτή την περίπτωση συμβαίνει κάτι αντίστοιχο με την ακριβώς προηγούμενη, μόνο που το αναμμένο led μετακινείται στη θέση που αντιστοιχεί στο msb, δηλαδή στο led7.

Η μεταβλητή που περιέχει την κατάσταση των led, η x, έχει οριστεί ως char, καθώς αυτός ο τύπος μεταβλητής αποτελείται από 8bit, όσα είναι και τα led των οποίων την κατάσταση θέλουμε να χειριστούμε μέσω των διακοπτών (Push Buttons).

Τέλος, οι συνθήκες για το αν τα Push Buttons είναι πατημένα ή όχι, για παράδειγμα "if((PINA & 0x04) == 4)", δυαδικά μεταφράζονται ως: (00000100 && 00000100) == (00000100), και αυτός είναι ο λόγος που στο δεξί μέρος της συνθήκης δεν είναι ο αριθμός 1, γιατί αλλιώς η συνθήκη θα ίσχυε μόνο για το πρώτο Push Button.