Συστήματα Μικρουπολογιστών

Εργαστηριακή Άσκηση

Μοίρας Αλέξανδρος Α.Μ.: el18081

Μαρίνος Δημήτριος Α.Μ.: el18135

Παπαδημητρίου Ευθύμιος Α.Μ.: el18129

Ο κώδικας των παρακάτω ασκήσεων περιέχει αναλυτικά σχόλια και συνοδεύεται από μία παράγραφο ανά άσκηση που εξηγεί συνοπτικά τη λειτουργία του κώδικα.

Ζήτημα 4.1:

```
.include "m16def.inc"

reset:
	ldi r24, low(RAMEND) ; Αρχικοποίηση στοίβας στο τέλος της RAM
	out SPL, r24
	ldi r24, high(RAMEND)
	out SPH, r24
	ser r24 ; Θύρα A ως έξοδος
	out DDRA, r24
	clr r24 ; Θύρα B ως είσοδος
	out DDRB, r24
	ldi r26, 0x01 ; Αναμμένο LED
	ldi r28, 0x01 ; Flag φοράς περιστροφής

main:
	out PORTA, r26 ; Άναμμα τρέχοντος LED
	in r16, PINB ; Ανάγνωση PINB
	andi r16, 0x01 ; Απομόνωση LSB
	cpi r16, 0x01 ; Αν είναι 1 (πατημένο) σταμάτα την κίνηση κάνοντας jump στη main
;χωρίς να ολισθήσει το αναμμένο LED
	breq main
	cpi r28, 0x01 ; Έλεγχος αν πρέπει να γίνει δεξιά (r28=0) ή αριστερή (r28=1)
;περιστροφή
	breq left

right:
	lsr r26 ; Δεξιά περιστροφή
	cpi r26, 0x01 ; Έλεγχος αν το αναμμένο LED είναι στο LSB
	brne main ; Αν όχι συνέχισε κανονικά τη λειτουργία
	ldi r28, 0x01 ; Αλλιώς άλλαξε τη φορά περιστροφής ενημερώνοντας το flag
	jmp main ; και μετά συνέχισε τη λειτουργία

left:
	lsl r26 ; Αριστερή περιστροφή
	cpi r26, 0x80 ; Έλεγχος αν το αναμμένο LED είναι στο MSB
	brne main ; Αν όχι συνέχισε κανονικά τη λειτουργία
	ldi r28, 0x00 ; Αλλιώς άλλαξε τη φορά περιστροφής ενημερώνοντας το flag
	jmp main ; και μετά συνέχισε τη λειτουργία
```

Το πρόγραμμα, αφού φτιάξει τον stack pointer, ορίσει τη θύρα A ως έξοδο και τη B ως είσοδο και αρχικοποιήσει το LED στο LSB, συνεχώς απεικονίζει το LED στην τρέχουσα θέση του, διαβάζει το PINB και ελέγχει αν έχει πατηθεί το PB0. Αν όχι μετακινεί το LED μία θέση είτε αριστερά είτε δεξιά αναλόγως του flag περιστροφής (αρχικά είναι στο 1 για αριστερή περιστροφή και μόλις φτάσει στην τέρμα δεξιά θέση το LED θα γίνει 0 ώστε να σημάνει δεξιά περιστροφή, αντιστοίχως όταν φτάσεις στην τέρμα αριστερή θέση θα ξαναγίνει 1). Αν ναι προχωρά στην επόμενη επανάληψη χωρίς να ολισθήσει το LED.

Ζήτημα 4.2:

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

char a,b,c,d, ccomp, f0, f1, output;

int main(void)
{
    DDRB=0xFF; //PORTB ως έξοδος
       DDRA=0x00; //PORTA ως είσοδος

    while (1)
    {
              output = 0x00; //αρχικοποίηση/επαναφορά output στο 0 για να γίνει or με τα
//f0,f1

              a=PINA & 0x01; //Ανάγνωση A

              b=PINA & 0x02; //Ανάγνωση B
              b = b >> 1; //ολίσθηση του bit B στο LSB (Επειδή διαβάζεται από το bit 1)

              c=PINA & 0x04; //Ανάγνωση C
              c = c >> 2; //ολίσθηση του bit C στο LSB
              ccomp = ~c; //Υπολογισμός C'
              ccomp = ccomp & 0x01; //Μάσκα γιατί έγιναν flip όλα τα bit και θέλουμε μόνο
//το 1ο


              d=PINA & 0x08; //Ανάγνωση D
              d = d >> 3; //ολίσθηση του bit D στο LSB

              f0 = (a & b & ccomp) | (c & d); //Υπολογισμός F0'
              f0 = ~f0; // Υπολογισμός F0 αντιστρέφοντας το F0' και εφαρμόζοντας
              f0 = f0 & 0x01; // μάσκα για να κρατήσουμε το τελευταίο bit

              f1 = (a | b) & (c | d); //Υπολογισμός F1
              f1 = f1 << 1; //και τοποθέτηση του στο 2ο LSB αφού θέλουμε να είναι στο bit
//1 της εξόδου

              output = output | f0; //F0 στο LSB της εξόδου
              output = output | f1; //F1 στο 2ο LSB της εξόδου

              PORTB = output; //Έξοδος σε PORTB
    }
```

```
}
```

Ορίζουμε την θύρα A ως είσοδο και τη B ως έξοδο. Διαβάζουμε τα A, B, C, D από το PINA χρησιμοποιώντας τις κατάλληλες μάσκες και εφαρμόζουμε κατάλληλο αριθμό ολισθήσεων ώστε να τα φέρουμε στο LSB. Υπολογίζουμε το συμπλήρωμα του C και ύστερα εκτελούμε τις λογικές πράξεις για τον υπολογισμό των f0 και f1. Ολισθαίνουμε το f1 μια θέση αριστερά καθώς θέλουμε να απεικονιστεί στο bit1 της εξόδου και ύστερα βάζουμε στο output που είναι αρχικά 0 το f0 or f1 που είναι το αποτέλεσμα που θα εκτυπώσουμε στην έξοδο.

Ζήτημα 4.3:

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

int led;

int main(void)
{
	DDRA=0xFF; //PORTA ως έξοδος
	DDRC=0x00; //PORTC ως είσοδος
	led=1; // Αρχική θέση LED στο LSB

    while (1)
    {
		if((PINC&0x01)==1) { // Έλεγχος πατήματος push-button SW0
			while((PINC&0x01)==1); // Έλεγχος επαναφοράς push-button SW0
			if(led==128) // Αν είσαι στο MSB led μετακινήσου στο LSB (κυκλική
//περιστροφή)
				led=1;
			else
				led = led << 1; // Αριστερή ολίσθηση
		}

		if((PINC&0x02)==2) { // Έλεγχος πατήματος push-button SW1
			while((PINC&0x02)==2); // Έλεγχος επαναφοράς push-button SW1
			if(led==1) // Αν είσαι στο LSB led μετακινήσου στο MSB (κυκλική
//περιστροφή)
			led=128;
			else
			led = led >> 1; // Δεξιά ολίσθηση
		}

		if((PINC&0x04)==4) { // Έλεγχος πατήματος push-button SW2
			while((PINC&0x04)==4); // Έλεγχος επαναφοράς push-button SW2
			led=128; //Μετακίνηση αναμμένου LED στη θέση MSB
		}

		if((PINC&0x08)==8) { // Έλεγχος πατήματος push-button SW3
			while((PINC&0x08)==8); // Έλεγχος επαναφοράς push-button SW3
			led=1; //Μετακίνηση αναμμένου LED στην αρχική του θέση LSB
		}
```

```
            PORTA=led; // Έξοδος σε PORTA
    }
}
```

Ορίζουμε την θύρα C ως είσοδο και την A ως έξοδο και την αρχική θέση του LED στο LSB. Έπειτα το πρόγραμμα συνεχώς διαβάζει την είσοδο ελέγχοντας αν πατήθηκε κάποιος από τους διακόπτες SW0-SW4 (χρησιμοποιώντας κατάλληλη μάσκα στην είσοδο και ελέγχοντας αν το bit της αντίστοιχης θέσης του διακόπτη είναι 1) ώστε να εκτελέσει την αντίστοιχη λειτουργία. Αν δεν πατηθεί κάποιος διακόπτης εκτυπώνει το LED στο PORTA. Αν πατηθεί αναμένει έως ότου αφεθεί πάλι ο διακόπτης (μπαίνοντας σε ένα while loop στο οποίο μένει έως ότου ξαναγίνει 0 το bit που αντιστοιχεί στον διακόπτη που πατήθηκε). Τότε εκτελείται η κατάλληλη λειτουργία για κάθε διακόπτη, το LED εκτυπώνεται στην έξοδο και επαναλαμβάνεται η όλη διαδικασία.