ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εργαστήριο Μικροϋπολογιστών

5η Εργαστηριακή Άσκηση

Ονοματεπώνυμα: Μπέτζελος Χρήστος, Γιαννιός Γεώργιος-Ταξιάρχης
Α.Μ. : 031 16 067, 031 16 156
Ομάδα : Α 16
Εξάμηνο: 7ο

Ζήτημα 5.1

(α) Γράψαμε πρόγραμμα σε assembly για τον ATmega16 στο σύστημα EasyAvr6 το οποίο έστελνε στην UART ένα string (σύνολο χαρακτήρων που τερματίζεται με τον χαρακτήρα '\0') το οποίο αποθηκεύτηκε στην RAM στην αρχή του προγράμματος σας.

```
#include "m16def.inc"

start:

;Αρχικοποίηση δείκτη στοίβας

 ldi r24 , low(RAMEND)

 out SPL , r24

 ldi r24 , high(RAMEND)

 out SPH , r24


;Φόρτωση του μηνύματος στη μνήμη από τη διεύθυνση 0600

 clr r27

 ldi r26, 0x60

 ldi r24, 'm'

 st X+, r24

 ldi r24, 'i'

 st X+, r24

 ldi r24, 'c'

 st X+, r24

 ldi r24, 'r'
```

```
 st X+, r24
 ldi r24, 'o'
 st X+, r24
 ldi r24, '\n'
 st X+, r24
 ldi r24, '\0'
 st X+, r24

 rcall usart_init
 clr r27
 ldi r26, 0x60

;Εμφάνιση string
looop:
        ld r24, X+
        cpi r24,'\0'
        breq exit
        rcall usart_transmit
        rjmp looop


usart_init:
 clr r24 ; initialize UCSRA to zero
 out UCSRA ,r24
 ldi r24 ,(1<<RXEN) | (1<<TXEN) ; activate transmitter/receiver
 out UCSRB ,r24
 ldi r24 ,0 ; baud rate = 9600
 out UBRRH ,r24
 ldi r24 ,51
 out UBRRL ,r24
 ldi r24 ,(1 << URSEL) | (3 << UCSZ0) ; 8-bit character size,
```

```
 out UCSRC ,r24 ; 1 stop bit
 ret

usart_transmit:
 sbis UCSRA ,UDRE ; check if usart is ready to transmit
 rjmp usart_transmit ; if no check again, else transmit
 out UDR ,r24 ; content of r24
 ret

usart_receive:
 sbis UCSRA ,RXC ; check if usart received byte
 rjmp usart_receive ; if no check again, else read
 in r24 ,UDR ; receive byte and place it in
 ret ; r24

exit:
 ;rjmp exit
```

(β) Γράψαμε σε C πρόγραμμα για τον ATmega16 στο σύστημα EasyAvr6 το οποίο διαβαζε έναν αριθμό από 0 έως 8 από την UART και άναβε το αντίστοιχο αριθμό LED της PORTB, αρχίζοντας από το LSB. Θεωρήσαμε ότι το 0 σβήνει όλα τα LED. Το πρόγραμμα μας απαντάει κάθε φορά που διαβάζει έναν αριθμό με μήνυμα της μορφής “Read X”, όπου X ο αριθμός που στάλθηκε. Σε περίπτωση ανάγνωσης μη έγκυρου αριθμού εκτυπώνεται μήνυμα “Invalid Number”

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

//Συναρτήσεις για ασύρματη επικοινωνία

void usart_init(void);
void usart_transmit(char);
char usart_receive(void);
void usart_transmit_string (char *);

void main(void)
```

```
{
      SP = RAMEND;
      char c;
      char *read="Read ";
      char *invalid="Invalid Number\n";
      DDRB=0xFF; // Αρχικοποίηση PORT B σαν έξοδος

      usart_init();




      while(1){
            c=usart_receive();
            _delay_ms(100);
            if(c>='0' && c<='8'){
                  usart_transmit_string(read);
                  usart_transmit(c);
            usart_transmit('\n');
                  if(c=='0') PORTB=0x00;
                  else {
                        c=c-0x30;
                        c=c-1;
                        PORTB=0x01<<c;
                  }
                  _delay_ms(100);
            }
            else if(c!='\n') {
                  usart_transmit_string(invalid);
                  _delay_ms(100);
            }
      }
}


void usart_init(void){
 UBRRH=0x00;
 UBRRL=51;
 UCSRA=0x00;
 UCSRB=(1<<RXEN) | (1<<TXEN);
 UCSRC=(1 << URSEL) | (3 << UCSZ0);
}

void usart_transmit(char x){
 while ((UCSRA & 0x20)==0);
 UDR=x;
}

char usart_receive(void){
 while ((UCSRA & 0x80)==0);
 return UDR;

}

void usart_transmit_string (char *message){
```

```
 int i=0;
 while (message[i]!='\0'){
      usart_transmit(message[i]);
      i++;
 }
}
```

Ζήτημα 5.2

Στο πλαίσιο της άσκησης μελετήσαμε δύο τρόπους χειρισμού του ADC. Ο πρώτος τρόπος ήταν με την χρήση της διακοπής ολοκλήρωσης της μετατροπής του ADC. Η διακοπή αυτή (ADC) μετέφερε τον έλεγχο στην διεύθυνση 0x1C, αν είναι ενεργοποιημένη η αντίστοιχη διακοπή (από το bit ADIE του ADCSRA) καθώς και οι γενικές διακοπές. Για να ξεκινήσει μια μετατροπή αρκεί να γραφεί 1 στο bit ADSC του καταχωρητή ADCSRA. Ο δεύτερος τρόπος είναι το πρόγραμμα να αναμένει να ολοκληρωθεί η μετατροπή. Η αναμονή αυτή γίνεται μέσω προγράμματος, ελέγχοντας το bit ADSC του ADCSRA το οποίο γίνεται 0 μόλις ολοκληρωθεί η μετατροπή (polling).

(α) Γράψαμε πρόγραμμα σε assembly για τον ATmega16 στο σύστημα EasyAvr6 το οποίο θα ξεκινάει μια μετατροπή του ADC και θα αυξάνει έναν μετρητή ο οποίος θα εμφανίζεται στα LED της PORTB κάθε 200ms. Η ανάγνωση των δεδομένων του ADC έγινε μέσα στην ρουτίνα εξυπηρέτησης της διακοπής ολοκλήρωσης μετατροπής του ADC και τα δεδομένα αυτά μετατράπηκαν σε τάση και να εκτυπώθηκαν στην UART με ακρίβεια δύο δεκαδικών ψηφίων (δεν χρειάζεται στρογγυλοποίηση). Η τάση δίνεται από τον τύπο:

$$V_{IN} = \frac{ADC}{1024} V_{REF} \text{ όπου}$$

όπου VIN η τάση στο pin A0,ADC η τιμή που διαβάζεται από τον ADC (αριθμός 10bit από 0-1023) VREF η τάση αναφοράς που με την δεδομένη ρουτίνα αρχικοποίησης έχει οριστεί σαν Vcc=5V.

Περιγραφή κώδικα : Στην άσκηση αυτή, μιας και το αποτέλεσμα μας (Vin) έπρεπε να έχει δύο δεκαδικά ψηφία έπρεπε να πολλαπλασιάσουμε με το 100 και στην συνέχεια με το Vref που ήταν 5.

Ο πολλαπλασιασμός έγινε προσθέτοντας τον ίδιο αριθμό 500 φορές σε μια loop.Επείδη το 500 δεν χωρούσε σε έναν καταχωρητή, (8 bit -> 255),γράψαμε 2 loops που εκαναν πρόσθεση 250 φορές του low καταχωρητη.

Το τελικό αποτέλεσμα αποθηκεύτηκε σε τρεις καταχωητες(high-median-low),οπου οι δύο πρώτοι γράφοταν μόνο όταν προέκυπτε κρατούμενο στον προηγούμενο τους.

Τρεις καταχωρητές χρειάστηκαν γιατι το μέγιστο αποτέλεσμα 1024*500 απαιτούσε το πολύ 8+8+3 bits.Στη συνέχεια κάναμε την διαίρεση με 1024 ώς εξής
α)Αγνοήσαμε τον low καταχωρητη (== 8 bits δεξιά== διαίρεση με 256) και

Β)Κάναμε με τη χρήση των εντολων (ror,lsr) shift τα δυο LSB του high στον median(==διαίρεση με 4)

Πλέον το αποτέλεσμα μας (9 bits) είναι στα 8 του median και στο LSB του high.

Με το διάγραμμα ροής της 3ης Εργαστηριακής βρήκαμε τις μονάδες,τις εκατοντάδες και τις δεκάδες .Στη περίπτωση που το LSB του high ήταν 1, έπρεπε να προσθέσουμε το το 256 ξεκινώντας από μονάδες και προσθέτοντας (αν χρειαστεί) κρατούμενο

```
.include "m16def.inc"


.def zero = r16

.def count = r27

.def temp = r24

.def flag = r19

.def low_ = r21

.def median_ = r22

.def high_ = r23

.def ekat = r29

.def dek = r28


.org 0x00

jmp reset


.org 0x1c

jmp ADC_interrupt

reti



reset:

 ldi r26, low(RAMEND)
```

```
 out SPL, r26
 ldi r26,high(RAMEND)
 out SPH, r26
 ser temp
 out DDRB,temp
 ldi count,0x00
 sei
 rcall usart_init
 rcall adc_init


start:

 out PORTB, count
 ldi r24 ,low(300)
 ldi r25 ,high(300)
 rcall wait_msec

 sbi ADCSRA, ADSC
 rjmp start

ADC_interrupt:
 set
 clr low_
 clr median_
 clr high_
 clr ekat
 clr dek
 clr zero

loop:
```

```
in r24, ADCSRA
sbrc r24, ADSC
rjmp loop
in r24,ADCL
in r25,ADCH
clr flag

convert1:
add low_, r24
adc median_, r25
adc high_, zero
inc flag
cpi flag,250
brlo convert1

clr flag

convert2:
add low_, r24
adc median_, r25
adc high_, zero
inc flag
cpi flag,250
brlo convert2


clr low_
lsr high_
ror median_
lsr high_
ror median_
```

```
ekato:
 cpi median_,100
 brlo decc
 inc ekat
 subi median_, 100
 rjmp ekato

decc:
 cpi median_,10
 brlo mon
 inc dek
 subi median_, 10
 rjmp decc

 ;ekat=r29, dek=r28, mon=median_=r22

mon:
 sbrs high_, 0
 rjmp screen

check_mon:
 subi median_, -6
 cpi median_, 10
 brsh mon_big_than_10
 rjmp check_dec

mon_big_than_10:
 subi median_, 10
 inc dek
```

```
check_dec:
 subi dek, -5
 cpi dek, 10
 brsh dec_big_than_10
 rjmp check_ekat

dec_big_than_10:
 subi dek, 10
 inc ekat

check_ekat:
 subi ekat, -2

screen:
 subi median_,-0x30
 subi dek,-0x30
 subi ekat,-0x30

 mov temp,ekat
 rcall usart_transmit

 ldi r24,'.'
 rcall usart_transmit

 mov temp,dek
 rcall usart_transmit

 mov temp,median_
 rcall usart_transmit

 ldi temp,'\n'
```

```
 rcall usart_transmit

 inc count

 reti


usart_init:
 clr r24 ; initialize UCSRA to zero
 out UCSRA ,r24
 ldi r24 ,(1<<RXEN) | (1<<TXEN) ; activate transmitter/receiver
 out UCSRB ,r24
 ldi r24 ,0 ; baud rate = 9600
 out UBRRH ,r24
 ldi r24 ,51
 out UBRRL ,r24
 ldi r24 ,(1 << URSEL) | (3 << UCSZ0) ; 8-bit character size,
 out UCSRC ,r24 ; 1 stop bit
 ret

usart_transmit:
 sbis UCSRA ,UDRE ; check if usart is ready to transmit
 rjmp usart_transmit ; if no check again, else transmit
 out UDR ,r24 ; content of r24
 ret

usart_receive:
 sbis UCSRA ,RXC ; check if usart received byte
 rjmp usart_receive ; if no check again, else read
 in r24 ,UDR ; receive byte and place it in
 ret ; r24
```

```
ADC_init:
 ldi r24,(1<<REFS0) ; Vref: Vcc
 out ADMUX,r24 ;MUX4:0 = 00000 for A0.
 ;ADC is Enabled (ADEN=1)
 ;ADC Interrupts are Enabled (ADIE=1)
 ;Set Prescaler CK/128 = 62.5Khz (ADPS2:0=111)
 ldi r24,(1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
 out ADCSRA,r24
 ret



wait_msec:
 push r24
 push r25
 ldi r24 , low(998)
 ldi r25 , high(998)
 rcall wait_usec
 pop r25
 pop r24
 sbiw r24 , 1
 brne wait_msec
 ret

wait_usec:
 sbiw r24 ,1
 nop
 nop
 nop
 nop
```

brne wait_usec

ret

(β) Γράψαμε σε C πρόγραμμα για τον ATmega16 στο σύστημα EasyAvr6 το οποίο θα ξεκινάει μια μετατροπή, θα περιμένει να ολοκληρωθεί η μετατροπή και θα εκτυπώνει την τάση στην UART με ακρίβεια δύο δεκαδικών ψηφίων.

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

void usart_init(void);
void usart_transmit(char);
void ADC_init(void);


void main(void)
{
	SP = RAMEND;

	unsigned long long int x, y;

	ADC_init();
	usart_init();

	while(1){
		ADCSRA = ADCSRA | (1<<ADSC);
		while(ADCSRA & (1<<ADSC));

		x = ADCL & 0xFF;
	   y = ADCH & 0x03;
	   x = x + (y * 256);
	   x = 100* x * 5 /1024;

	   y = x / 100 ;

	   usart_transmit(y + 0x30);
	   usart_transmit(',');

	   y = x - 100*y;
		y=y/10;

	   usart_transmit(y + 0x30);

		y = x % 10;

		usart_transmit(y + 0x30);

	   usart_transmit('\n');
	}

}

void usart_init(void){
	UBRRH=0x00;
```

```
      UBRRL=51;
      UCSRA=0x00;
      UCSRB=(1<<RXEN) | (1<<TXEN);
      UCSRC=(1 << URSEL) | (3 << UCSZ0);
}

void usart_transmit(char x){
      while ((UCSRA & 0x20)==0);
      UDR=x;
}


void ADC_init(void) {
      ADMUX = (1<<REFS0);
      ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}
```