

## Θεόδωρος Μπάρκας 2016030050 LAB2

### Σκοπός άσκησης και παραλλαγές του προηγούμενου εργαστηρίου

Σκοπός της άσκησης ήταν να υλοποιήσουμε ένα Seven Segment Display τον 8 δεκαδικών ψηφίων με πολυπλεξία στο χρόνο. Εκμεταλλευόμενοι μόνο τις εξόδους PortC και PortB .Δηλαδή συνολικά  $8+7=15$  bits.

Για να λυθεί το πρόβλημα κάθε SSD πρέπει να ανάβει με συχνότητα τουλάχιστον 30Hz για να είναι ευχάριστο οπτικά .Καθώς πρέπει να διανείμουμε( χρονικά ) το φόρτο σε συνολικά 8 SSD πρέπει να επιλέξουμε μια συχνότητα  $8*30\text{Hz}=240\text{Hz}$  τουλάχιστον. Στο δικό μου πρόγραμμα επιλέγω ένα Interrupt το οποίο θεωρητικά να συμβαίνει 250 φορές το δευτερόλεπτο δηλαδή η συχνότητα μου είναι  $250\text{ Hz} > 240\text{ Hz}$ .

Όμως 250 φορές το Δευτερόλεπτο συνεπάγεται Interrupts κάθε 4 ms.

Στο προηγούμενο εργαστήριο είχαμε φτιάξει Interrupt του 1ms.

Όπως είναι προφανές αρκεί να πολλαπλασιάσω με 4 το χρόνο του πότε γίνεται το Interrupt με 4 ή με ισοδύναμο τρόπο να πολλαπλασιάσω το step ( ρόλοι ) του counter επί 4 .Επομένως είναι προφανές ότι αρκεί να κάνω τον Prescaler του Counter μου 4 φορές μεγαλύτερο. Αφού  $f_{\text{new}}=f_{\text{old}}/4$ .

Επομένως κρατάμε τα Steps του Counter στα 156 από την προηγούμενη φορά και κάνουμε τον N , 4 φορές μεγαλύτερο από την προηγούμενη φορά  $N=4*64=256 \rightarrow \text{CS02}..\text{CS00}=110$

CS22	CS21	CS20	Description
1	1	0	$\text{clk}_{T2}/256$ (From prescaler)

Τότε έχουμε ότι η νέα συχνότητα είναι περίπου 4ms.

Συγκεκριμένα:

$$f_{\text{interrupt}}=f_{\text{ck}} / (N*\text{Steps}) \rightarrow$$

$$f_{\text{interrupt}}=10\text{Mhz}/256*156=250,300641026$$

$$T_{\text{interrupt}}=0.399359999$$

Αποτελέσματα πολύ κοντά και εντός των προδιαγραφών που θέσαμε στην αρχή για συχνότητα  $>250\text{Hz}$ .

## Κώδικας με Σχόλια

Ο Καταχωρητής X χρησιμοποιείτε για να ορίζουμε σε ποια θέση της μνήμης αναφερόμαστε όταν χρησιμοποιούμε εντολές όπως το LD ,ST. Στην Υλοποίηση μου αρχίζω από την αρχή της RAM και αποθηκεύω προς τα κάτω.

```
;we Store in Ram
ldi XL,LOW(SRAM_START)      ; initialize X pointer
ldi XH,HIGH(SRAM_START)
```

Συγκεκριμένα στην αρχικοποίηση που κάνω βάζω τα 8 πρώτα bytes στη μνήμη από την μικρότερο address στο μεγαλύτερο σε μορφή BCD, όπου τον αριθμό τον αναπαριστούν τα 4 τελευταία Bits του κάθε Byte.

Επίσης το μικρότερο address της μνήμης αντιστοιχεί στο λιγότερο σημαντικό δεκαδικό ψηφίο που γίνεται Display.

Υστερα αρχικοποιώ στις επόμενες θέσεις της μνήμης το εκάστοτε segment 6 down to 0 -> a down to g με τη σωστή σειρά στα addresses (από το 0 στο 9 )το οποίο είναι πολύ σημαντικό μέρος της λύσης καθώς χρησιμοποιείτε παρακάτω στο **DECODER μέρος του κωδικά** (σε συνάρτηση με το BCD αριθμό που προβάλλεται ).

## **LOADING μέρος κωδικά**

Loading block gives us in r18 the Number we gonna display is bsd AND at R17 the memory location of The Decimal Dedit we display AND in r19 the Port C outputs

Πιο συγκεκριμένα αρχικοποιήσαμε το r17 στη αρχή του προγράμματος (δεν χρησιμοποιείτε σε άλλο σημείο του κωδικά) στην τιμή 0x07.Έτσι ώστε τη πρώτη φορά που γίνεται interrupt να γίνεται Branch στο if1: label στο οποίο η μνήμη ορίζεται σωστά και αρχικοποιούνται όλες οι τιμές.

π.χ. ο καταχωρητής X να δείχνει στην SRAM\_START ,τον r17 στο Memory Location (για να ανάψει το Least Significant Δεκαδικό του Display που φτιάχνουμε) και οι λοιπές μεταβλητές που φαίνονται στο κωδικά και αναλύονται και στα σχόλια.

Κατά τα άλλα μέσα στο **LOADING μέρος του κωδικά** φορτώνονται οι τιμές από τη μνήμη στο r18 που έχουν να κάνουν με το εκάστοτε BCD που προβάλλουμε καθώς και αλλάζουν οι τιμές που είπαμε παραπάνω.

### **Κώδικας Loading**

```
;loading block start
inc r17
cpi r17,0x08 ;gives Zflag = 1 when r17-0x07=0 ;;first time program enters here is imporetant to branch so that everything works as designed
breq if1
LSL r19 ;R19 Is the output for C port
ld r18,X+ ;r18 has the value of the Number we gonna display is bsd
rjmp endif1
if1:
    ldi r19,0x01;R19 Is the output for C port
    ldi r17,0x00
    subi XL,8 ; X pointer register only low part ;
    ld r18,X+ ;r18 has the value of the Number we gonna display is bsd
endif1:
;loading block end
;Loading block gives us in r18 the Number we gonna display is bsd AND at R17 the memory location of The Decimal Dedit we display AND in r19 the Port C outputs
;dont touch r17,r19,through all the program
;dont touch r18 until you succesfully decode it and get it out
```

## **DECODER μέρος κωδικά**

Στο Decoder αρχικά αρχικοποιούμε τον pointer X στη θέση μνήμης  $X=SRAM\_START+8$  .

Μετά εκμεταλλευόμενοι ότι έχουμε αρχικοποιήσει τα segments του SSD στη RAM που έχουν οριστεί έτσι ώστε  $X+r18$  (r18 the Number we gonna display is bsd) να μας δεινή τα ανάλογα Segments και τα Configuration που θα πρέπει να έχουν αυτά για τον εκάστοτε BCD που προβάλλουμε,απλώς κάνουμε Load τη συγκεκριμένη θέση μνήμης στον καταχωρητή r20.

```

    add XL,r18 ;r18 is the number we want to show
    ld r20,X

```

Ύστερα κλείνουμε το PORTC που δίνει τις ανόδους έτσι ώστε να μην έχουμε λάθος προβολή στο εκάστοτε SSD .Περνάμε στο PORTB την τιμή r20 που είναι η σωστή κωδικοποίηση για το εκάστοτε Segment και κατόπιν ανοίγουμε ξανά με σωστή άνοδο αυτή τη φορά το PORTC .

```

;START OF OUT
;out C port r16 close all Anodes -ADD STUFFS-
ldi r16, 0b00000000 ;Anodes for SSDs
out PORTC, r16
;out B port r20 6 down to 0 -> a down to g -ADD STUFFS-
out PORTB, r20
;out C port r19 7 down to 0 -> MSDecimal(7) to LSDecimal(0) anodes -ADD STUFFS-
out PORTC, r19
;-END OF OUT

```

Τέλος βάζουμε πάλι στη RAM την τιμή την οποία είχε πριν αρχίσει να τρέχει το DECODER μέρος του κωδικά.

### Κώδικας Decoder

```

;decoder Start
ldi r16,8
ldi XL, LOW(SRAM_START)
add XL,r16 ; initialize X pointer to SRAM_START+8

add XL,r18 ;r18 is the number we want to show
ld r20,X

;START OF OUT
;out C port r16 close all Anodes -ADD STUFFS-
ldi r16, 0b00000000 ;Anodes for SSDs
out PORTC, r16
;out B port r20 6 down to 0 -> a down to g -ADD STUFFS-
out PORTB, r20
;out C port r19 7 down to 0 -> MSDecimal(7) to LSDecimal(0) anodes -ADD STUFFS-
out PORTC, r19
;-END OF OUT

ldi XL, LOW(SRAM_START) ;getting the X pointer where it was before decoder block
add XL,r17
inc XL ;inc because we used "ld r18,X+" so X was left one count after sram_start+r17
;decoder End

```

\*(To DDRB,DDRC αρχικοποιηθήκαν εκτός του Handler πριν καν το πρόγραμμα μπει στο waitloop )

## TESTING

Παρακάτω βλέπουμε τις αρχικοποιήσεις που έχω κάνει

```
;initialization start /numbers that we want to display in the memory
;initialization of numbers to display
ldi r16,0x05
st x+,r16
ldi r16,0x06
st x+,r16
ldi r16,0x07
st x+,r16
ldi r16,0x08
st x+,r16
ldi r16,0x09
st x+,r16
ldi r16,0x01
st x+,r16
ldi r16,0x02
st x+,r16
ldi r16,0x03
st x+,r16
```

Αρχικά βλέπουμε ότι ο χρόνος που κάναμε για να φτάσουμε στο πρώτο Interrupt δηλαδή ισοδύναμα το  $f\_interrupt=3,9967\text{ ms}$  που είναι πολύ κοντά στα  $4\text{ms}$  . Εντός των ορίων  $f > 240\text{hz}$ .

The screenshot shows a disassembly window for a file named 'main.asm'. The assembly code is as follows:

```
;decoder Start
ldi r16,8
ldi XL, LOW(SRAM_START)
add XL,r16 ; initialize X pointer to SRAM_START+8

add XL,r18 ;r18 is the number we want to show
ld r20,X

;START OF OUT
;out C port r16 close all Anodes -ADD STUFFS-
ldi r16, 0b00000000 ;Anodes for SSDs
out PORTC, r16
;out B port r20 6 down to 0 -> a down to g -ADD STUFFS-
out PORTB, r20
;out C port r19 7 down to 0 -> MSDecimal(7) to LSDecimal(0) anodes -ADD STUFFS-
out PORTC, r19
;-END OF OUT

ldi XL, LOW(SRAM_START) ;getting the X pointer where it was before decoder block
add XL,r17
inc XL ;inc because we used "ld r18,X+" so X was left one count after sram_start+r17
;decoder End

reti
```

On the right side, the 'Processor Status' window is open, showing the following values:

Name	Value
Program Counter	0x0000153
Stack Pointer	0x045D
X Register	0x0061
Y Register	0x0000
Z Register	0x0000
Status Register	0x0000
Cycle Counter	39967
Frequency	10,000 MHz
Stop Watch	3.996,70 $\mu\text{s}$

Below the status window, the 'Registers' window is open, showing the values of registers R00 through R09, all of which are 0x00.

Ύστερα βάζοντας παραπάνω break points ελέγχουμε αν είναι σωστή η λειτουργία της αλλαγής των LED από δεκαδικό σε δεκαδικό και πάμε στο δεύτερο interrupt έτσι ώστε να έχουν ήδη πάρει τιμή οι καταχωρητές PORTC,PORTB για να δούμε αν γίνεται σωστά η αλλαγή.



Και αφού πάρει και ο PORTB την σωστή τιμή ο PORT C και ο PORT B ξαναδείχνουν σωστά (κάτω εικόνα)

The left screenshot shows the assembly window with the following code:

```
out PORTC, r16  
;out B port r20 6 down to 0 -> a dow  
out PORTB, r20  
;out C port r19 7 down to 0 -> MSDec  
out PORTC, r19  
;-END OF OUT  
  
ldi XL, LOW(SRAM_START) ;getting the  
add XL,r17  
inc XL ;inc because  
;decoder End  
  
reti
```

The right screenshot shows the assembly window with the following code:

```
ldi r16, 0b00000000 ;Anodes for SSL  
out PORTC, r16  
;out B port r20 6 down to 0 -> a dc  
out PORTB, r20  
;out C port r19 7 down to 0 -> MSDe  
out PORTC, r19  
;-END OF OUT  
  
ldi XL, LOW(SRAM_START) ;getting th  
add XL,r17  
inc XL ;inc because  
;decoder End  
  
reti
```

Below the assembly windows are two I/O register windows. The left window shows the state of the I/O ports:

Name	Value
I/O I/O Port (PORTB)	
I/O I/O Port (PORTC)	
I/O I/O Port (PORTD)	
JTAG Interface (JTAG)	
Serial Peripheral Interface (...)	

The right window shows the state of the I/O ports:

Name	Value
I/O I/O Port (PORTB)	
I/O I/O Port (PORTC)	
I/O I/O Port (PORTD)	
JTAG Interface (JTAG)	
Serial Peripheral Interface (...)	

# TESTING LOADING μέρος του κωδικά

Υστερα βλέπουμε ότι και ο κύκλος γίνεται σωστά επομένως αποδεικνύουμε τη σωστή λειτουργικότητα του **LOADING μέρος του κωδικά** .Παρατηρήστε Cycle Counter,Stop Watch .

## 1ο Άνοιγμα της πρώτης ανόδου (λιγότερου σημαντικοί δεκαδικού ψηφίου)

### PORTC

```
;out C port r16 close all Anodes -ADD STUFFS-
ldi r16, 0b00000000 ;Anodes for SSDs
out PORTC, r16
;out B port r20 6 down to 0 -> a down to g -ADD STUFFS-
out PORTB, r20
;out C port r19 7 down to 0 -> MSDecimal(7) to LSDecimal(0) anodes -ADD STUFFS-
out PORTC, r19
;-END OF OUT

ldi XL, LOW(SRAM_START) ;getting the X pointer where it was before decoder block
add XL,r17
inc XL ;inc because we used "ld r18,X+" so X was left one count after sram_start+r17
;decoder End

reti
```

Name	Value
I/O Port (PORTA)	
I/O Port (PORTB)	
I/O Port (PORTC)	
I/O Port (PORTD)	
JTAG Interface (JTAG)	

Name	Address	Value	Bits
PO...	0x35	0x01	00000000
PI...	0x33	0x01	00000000
DD...	0x34	0xFF	11111111

Z Register

Status Register

Cycle Counter 39967

Frequency 10,000 MHz

Stop Watch 3.996,70 μs

Registers

R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0x00
R18	0x05
R19	0b00000001
R20	0b00100100
R21	0x00
R22	0x00

### PORTB

```
;START OF OUT
;out C port r16 close all Anodes -ADD STUFFS-
ldi r16, 0b00000000 ;Anodes for SSDs
out PORTC, r16
;out B port r20 6 down to 0 -> a down to g -ADD STUFFS-
out PORTB, r20
;out C port r19 7 down to 0 -> MSDecimal(7) to LSDecimal(0) anodes -ADD STUFFS-
out PORTC, r19
;-END OF OUT

ldi XL, LOW(SRAM_START) ;getting the X pointer where it was before decoder block
add XL,r17
inc XL ;inc because we used "ld r18,X+" so X was left one count after sram_start+r17
;decoder End

reti
```

Name	Value
I/O Port (PORTA)	
I/O Port (PORTB)	
I/O Port (PORTC)	
I/O Port (PORTD)	
JTAG Interface (JTAG)	

Name	Address	Value	Bits
PO...	0x38	0x24	00000000
PINB	0x36	0x24	00000000
DD...	0x37	0x7F	01111111

Z Register

Status Register

Cycle Counter 39967

Frequency 10,000 MHz

Stop Watch 3.996,70 μs

Registers

R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0x00
R18	0x05
R19	0b00000001
R20	0b00100100
R21	0x00
R22	0x00

## 2ο Άνοιγμα της πρώτης ανόδου (λιγότερου σημαντικοί δεκαδικού ψηφίου) PORTC

The screenshot shows the Keil IDE with assembly code for initializing PORTC. The code includes comments in Greek and instructions to configure PORTC and PORTB. The I/O window shows the status of various I/O ports, including PORTC, PORTB, PORTD, and JTAG Interface (JTAG). The Registers window shows the status of registers R00 through R21.

```
ld r20,X

;START OF OUT
;out C port r16 close all Anodes -ADD STUFFS-
ldi r16, 0b00000000 ;Anodes for SSDs
out PORTC, r16
;out B port r20 6 down to 0 -> a down to g -ADD STUFFS-
out PORTB, r20
;out C port r19 7 down to 0 -> MSDecimal(7) to LSDecimal(0) anodes -ADD STUFFS-
out PORTC, r19
;-END OF OUT

ldi XL, LOW(SRAM_START) ;getting the X pointer where it was before decoder block
add XL,r17
inc XL ;inc because we used "ld r18,X+" so X was left one count after sram_start+r17
;decoder End

retl
```

I/O

Name	Value
I/O Port (PORTB)	
I/O Port (PORTC)	
I/O Port (PORTD)	
JTAG Interface (JTAG)	
Serial Peripheral Interface (...)	

Name	Address	Value	Bits
PO...	0x35	0x01	00000000
PI...	0x33	0x01	00000000
DD...	0x34	0xFF	11111111

Registers

Register	Value
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0x00
R18	0x05
R19	0b00000001
R20	0b00100100
R21	0x00

## PORTB

The screenshot shows the Keil IDE with assembly code for initializing PORTB. The code includes comments in Greek and instructions to configure PORTB and PORTC. The I/O window shows the status of various I/O ports, including PORTA, PORTB, PORTC, and JTAG Interface (JTAG). The Registers window shows the status of registers R00 through R21.

```
add XL,r18 ;r18 is the number we want to show
ld r20,X

;START OF OUT
;out C port r16 close all Anodes -ADD STUFFS-
ldi r16, 0b00000000 ;Anodes for SSDs
out PORTC, r16
;out B port r20 6 down to 0 -> a down to g -ADD STUFFS-
out PORTB, r20
;out C port r19 7 down to 0 -> MSDecimal(7) to LSDecimal(0) anodes -ADD STUFFS-
out PORTC, r19
;-END OF OUT

ldi XL, LOW(SRAM_START) ;getting the X pointer where it was before decoder block
add XL,r17
inc XL ;inc because we used "ld r18,X+" so X was left one count after sram_start+r17
;decoder End

retl
```

I/O

Name	Value
I/O Port (PORTA)	
I/O Port (PORTB)	
I/O Port (PORTC)	
I/O Port (PORTD)	
JTAG Interface (JTAG)	

Name	Address	Value	Bits
PO...	0x38	0x24	00000000
PINB	0x36	0x24	00000000
DD...	0x37	0x7F	01111111

Registers

Register	Value
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0x00
R18	0x05
R19	0b00000001
R20	0b00100100
R21	0x00