

---

Ενσωματωμένα Συστήματα Μικροεπεξεργαστών

Project 2

Ημερομηνία Παράδοσης: 16 Οκτωβρίου 2021

Αμπλιανίτης Κωνσταντίνος

2017030014

---

## Σκοπός της Άσκησης:

Σκοπός της δεύτερης εργαστηριακής άσκησης είναι η εξοικείωση με την χρήση της μνήμης. Με seven segment displays θα πρέπει να απεικονίζονται δεδομένα τα οποία έχουν γίνει store στην μνήμη.

## Περιγραφή της τεχνολογίας που χρησιμοποιήθηκε:

Για την υλοποίηση της συγκεκριμένης εργαστηριακής άσκησης χρησιμοποιήθηκε το Microchip Studio (version 7.0.2542) της Microchip Technology.

## Περιγραφή επίλυσης της άσκησης:

Για την επίλυση της άσκησης χρησιμοποιήθηκαν οι αρχές του πρώτου εργαστηρίου με κάποιες προσθήκες. Αρχικά δεσμεύτηκαν από την μνήμη (SRAM) 9 byte, 8 για το portC δηλαδή για την οθόνη που θα ανάβει κάθε φορά και 1 για την συνθήκη του reset. Επιπλέον έγινε δέσμευση ακόμα 10 byte, τα οποία είναι υπεύθυνα για την αποθήκευση των καταστάσεων 0-9 σε seven-segment. Αυτό έγινε χρησιμοποιώντας το directive .dseg(data segment) όπως φαίνεται παρακάτω.

```
.dseg
; reserve the blocks of memory that we will need to store the addresses and the values of the numbers
port_C_segment: .byte 9 ;eight for the memory allocation and one for the situation that will need the reset
seven_seg_status: .byte 10
```

Όσον αφορά το code segment το πρώτο πράγμα που έγινε ήταν η φόρτωση των καταστάσεων στην μνήμη. Αυτό έγινε με τον εξής τρόπο. Αρχικά έγινε load η διεύθυνση μνήμης του χώρου που δεσμεύτηκε ως seven\_seg\_status με σκοπό να αποθηκευτούν οι καταστάσεις 0-9 στη μορφή της 7 segment display. Έπειτα υπολογιζόταν η τιμή στην οποία ανταποκρινόταν κάθε κατάσταση ( πχ 0  $\rightarrow$  00000011) και μέσω της εντολής st και του καταχωρητή Z (ένας από του τρεις που είναι υπεύθυνοι για την εισαγωγή στην/εξαγωγή από την μνήμη) πραγματοποιούνταν η αποθήκευση στη μνήμη.

```
; now i initialize the values i want into sram
; bring the address that is going to store the statuses of the numbers
ldi ZL, LOW(seven_seg_status)
ldi ZH, HIGH(seven_seg_status)

ldi r16, 0b00000011
st Z+, r16 ; store 0

ldi r16, 0b10011111
st Z+, r16 ; store 1

ldi r16, 0b00100101
st Z+, r16 ; store 2

ldi r16, 0b00001101
st Z+, r16 ; store 3

ldi r16, 0b10011001
st Z+, r16 ; store 4
```

Σημαντική παρατήρηση είναι ότι στην σύνταξη της εντολής `st` χρησιμοποιείται ο `Z+` ως όρισμα. Με αυτόν τον τρόπο μετά την αποθήκευση σε συγκεκριμένο block μνήμης η διεύθυνση του `Z` θα είναι η διεύθυνση του αμέσως επόμενου block. Αυτό βοηθάει στο να μην πανωγράφεται η μνήμη κατά την διάρκεια του store.

Ακολουθεί η ίδια διαδικασία για τους BCD αριθμούς. Στην συγκεκριμένη περίπτωση χρησιμοποιείται ο καταχωρητής `Y` για το γράψιμο στην μνήμη. Η αποθήκευση των αριθμών γίνεται μέσω μίας επαναληπτικής διαδικασίας και της εντολής `st` που χρησιμοποιήθηκε με τον ίδιο τρόπο όπως αναλύθηκε και παραπάνω.

```
;reinitialise the r16 as long as its last value is of no use
;registers that will help me with the bcd depiction
ldi r16, 0x01
ldi r17, 0x0A

; load in Y the address of the port_C_segment to store the bcd encoding
ldi YL, LOW(port_C_segment)
ldi YH, HIGH(port_C_segment)

; create a loop to store the numbers to the port_C_segment of the memory that i reserved
store_loop: ; bcd loop basically
    st Y+, r16
    inc r16
    cp r16,r17 ; comparison that will help get out of the loop
    brne store_loop
```

Μετά την αποθήκευση τόσο των καταστάσεων όσο και των bcd αριθμών πραγματοποιούνται οι αρχικοποιήσεις. Γίνονται οι αρχικοποιήσεις των Data Direction Registers των A,C ως εξόδους, όπως ζητείται από την εκφώνηση. Έπειτα γίνεται αρχικοποίηση της πρώτης κατάστασης που θα εμφανίζεται. Για την εναλλαγή των καταστάσεων δημιουργείται ένας rotating register όπου υποδεικνύει ποιά από τις 8 οθόνες θα είναι ενεργή (ενεργό στο 1). Με κάθε αλλαγή κατάστασης γίνεται shift left μέσω της εντολής `rol`.

Τέλος είναι απαραίτητη η ρύθμιση του timer. Θα χρησιμοποιηθεί πάλι ο `Timer0`. Η επιλογή του prescaler έγινε με τον τύπο:

$$P_{val} = \frac{P_{clk}}{TOV_{clk} * Max\_Val}$$

Για ρολόι 10MHz και θέλοντας 240 interrupts το δευτερόλεπτο προκύπτει ότι ο Prescaler που πρέπει να βάλουμε είναι ο 250 (ο αμέσως επόμενος διαθέσιμος από το 164). Υπολογίζεται ότι για 240 interrupts το δευτερόλεπτο χρειάζεται ο timer να κάνει overflow κάθε 4.1ms (Υπολογίζεται όπως και στο προηγούμενο εργαστήριο) συνεπώς ο timer αρχικοποιείται στην τιμή 96. Τέλος ενεργοποιούμε τα αντίστοιχα interrupts (TOV, TOIE). Τελευταίο βήμα είναι η ατέρμονη επαναληπτική διαδικασία ώστε να περιμένει ο CPU στα διαστήματα μεταξύ των interrupts.

Για τον interrupt handler του timer γίνονται τα εξής. Αρχικά γίνεται shift left του `rotate_reg` μέσω της εντολής `rol` για να ανοίξει ο επόμενος από τις 8 7-segment displays.

```
timer_ovfr:
    rol rotate_reg
    ld address_adder, Y+
    cp address_adder, r17 ; address_adder will become 9 same as the r17 because it's stored on memory

    breq reset_address

changes:
    ; change the values
    ; process to load the value of the rotate reg
    ldi ZL, LOW(seven_seg_status)
    ldi ZH, HIGH(seven_seg_status)

    add ZL, address_adder

    ld r2, Z
    out PORTA, r2
    out PORTC, rotate_reg
    ; reset the timer
    ldi r19, 0x60 ;timer seted to 96
    out TCNT0, r19
    sei
    rjmp final_loop
```

Ενδεικτικά παρουσιάζονται τα εξής αποτελέσματα. Αρχική κατάσταση, πρώτη αλλαγή και αλλαγή μετά από  $\frac{1}{30}$  του δευτερολέπτου ώστε να φανεί η παρουσία ιδίων εξόδων με την αρχική.

Register	Address	Value	Hex Value	Bit Pattern	Status
I/O PINC	0x33				Program Counter: 0x0000003E
I/O DDRC	0x34				Stack Pointer: 0x0000
I/O PORTC	0x35				X Register: 0x0000
<hr/>					
I/O PINA	0x39	0x9F			Y Register: 0x0061
I/O DDRA	0x3A	0xFF			Z Register: 0x006A
I/O PORTA	0x3B	0x9F			Status Register: 0x00000000
					Cycle Counter: 100
					Frequency: 10.000 MHz
					Stop Watch: 10.60 us

4

### Πρώτο Transition (delay 4.1ms)

I/O	PINC	0x33	0x02	■ ■ ■ ■ ■ ■ ■ ■	Program Counter	0x00000052
I/O	DDRC	0x34	0xFF	■ ■ ■ ■ ■ ■ ■ ■	Stack Pointer	0x07FE
I/O	PORTC	0x35	0x02	■ ■ ■ ■ ■ ■ ■ ■	X Register	0x0000
					Y Register	0x0062
					Z Register	0x006B
I/O	PINA	0x39	0x25	■ ■ ■ ■ ■ ■ ■ ■	Status Register	■ ■ ■ ■ ■ ■ ■ ■
I/O	DDRA	0x3A	0xFF	■ ■ ■ ■ ■ ■ ■ ■	Cycle Counter	40978
I/O	PORTA	0x3B	0x25	■ ■ ■ ■ ■ ■ ■ ■	Frequency	10.000 MHz
					Stop Watch	4,097.80 μs

### $\frac{1}{30}$ s επιβεβαίωση κατάστασης 1

I/O	PINC	0x33	0x01	■ ■ ■ ■ ■ ■ ■ ■	Program Counter	0x00000052
I/O	DDRC	0x34	0xFF	■ ■ ■ ■ ■ ■ ■ ■	Stack Pointer	0x07F0
I/O	PORTC	0x35	0x01	■ ■ ■ ■ ■ ■ ■ ■	X Register	0x0000
					Y Register	0x0060
					Z Register	0x006A
I/O	PINA	0x39	0x9F	■ ■ ■ ■ ■ ■ ■ ■	Status Register	■ ■ ■ ■ ■ ■ ■ ■
I/O	DDRA	0x3A	0xFF	■ ■ ■ ■ ■ ■ ■ ■	Cycle Counter	327700
I/O	PORTA	0x3B	0x9F	■ ■ ■ ■ ■ ■ ■ ■	Frequency	10.000 MHz
					Stop Watch	32,770.50 μs

Όπως φαίνεται στην εικόνα ότι μετά  $\approx 33ms$  (λόγω στρογγυλοποιήσεων χάνεται η απόλυτη ακρίβεια) η οθόνη θα βρίσκεται στην ίδια κατάσταση που βρισκόταν στην αρχικοποίηση της. Αυτό γίνεται κάθε  $\approx 33ms$  και ισχύει για όλες τις καταστάσεις.

*Παρατήρηση:* Υπολογίζεται ότι σε 1s δεδομένου ότι λαμβάνονται 240 interrupts/s ο συνολικός αριθμός που χρειάζεται για τις ανανεώσεις της οθόνης στο συγκεκριμένο διάστημα είναι  $\approx 14 * 240 = 3360$  εντολές (περίπου λόγω εντολών διακλάδωσης που θα γίνονται κάθε 8 αλλαγές οθόνης). Συνεπώς το ποσοστό των κύκλων που απασχολούν το CPU σε ένα δευτερόλεπτο για την ανανέωση της οθόνης είναι  $\frac{3360}{10^7} = 0.033\%$ . Ποσό που μας δείχνει την μεγάλη ποσότητα των διαθέσιμων ελεύθερων εντολών ώστε να χρησιμοποιηθούν για άλλες λειτουργίες.

### Συμπεράσματα

Μέσα από την δεύτερη εργαστηριακή άσκηση έγινε η πρώτη επαφή με την αποθήκευση στην μνήμη και την ανάκληση δεδομένων από την. Επιπλέον, φάνηκε πόσο καλή διαχείριση πόρων μπορεί να γίνει μιας και με την χρήση δύο ports υπάρχει η δυνατότητα να ελεγχθούν 10 καταστάσεις σε 8 7segment displays.