

6 Παράδειγμα 1: Λειτουργία ενός LED

Για να ενεργοποιηθεί/απενεργοποιηθεί ένα LED με κάποια συγκεκριμένη τιμή στην περίοδο, π.χ. $T=10\text{ms}$, θα πρέπει πρώτα να οριστεί το “Direction” του συγκεκριμένου Pin ως “Output”. Αρχικά, το PORT που θα χρησιμοποιηθεί είναι το PORTD καθώς οι τέσσερες πρώτοι ακροδέκτες (PINs) του συνδέονται με LEDs, πάνω στην πλακέτα (όπως φαίνεται και στο Board Overview). Για να οριστεί ένα PIN ως “Output” πρέπει να τεθεί το ψηφίο (bit) 5 του Register DIR (Data Direction) με ‘1’. Τώρα μπορεί να δοθεί η τιμή ‘0’ ή ‘1’ στον Register Out (Output Value) και να “ενεργοποιείται” ή να “απενεργοποιείται”, το LED αντίστοιχα.

Σημείωση: Στη φάση του Simulation οι χρόνοι του “delay” όπως και των “timers”, (που θα δούμε στη συνέχεια) δεν είναι αντιπροσωπευτικοί του χρόνου που αναμένουμε. Μόνο με την εκτέλεση του κώδικα πάνω στην πλακέτα μπορούμε να αντιληφθούμε τους πραγματικούς χρόνους. Για το λόγο αυτό, μπορούμε να επιλέγουμε σχετικά μικρούς χρόνους, για να αποφύγουμε μεγάλες καθυστερήσεις αναμονής.

Ο κώδικας της υλοποίησης του παραδείγματος παρουσιάζεται παρακάτω:

```
#include <avr/io.h>
#include <util/delay.h>
#define del 10
int main(void){
    //PIN is output
    PORTD.DIR |= 0b00000010; //PIN1_bm
    //LED is off
    PORTD.OUT |= 0b00000010; //PIN1_bm
    while (1) {
        //on
        PORTD.OUTCLR= 0b00000010; //PIN1_bm
        _delay_ms(del); //wait for 10ms
        //off
        PORTD.OUT |= 0b00000010; //PIN1_bm
        _delay_ms(del); //wait for 10ms
    }
}
```

Σχήμα 0.11 : Κώδικας Παραδείγματος 1

Μπορείτε να χρησιμοποιήσετε τον κώδικα του παραδείγματος, για να εκτελέσετε τη διαδικασία του “Simulation” στο “Microchip Studio”. Ανοίξτε το I/O παράθυρο (Debug ⇒ Windows ⇒ I/O) και ξεκινήστε το Debugging. Εκτελέστε βήμα-βήμα τις εντολές (η συνάρτηση delay δεν εκτελείται βηματικά) και παρατηρήστε τις τιμές που παίρνουν οι καταχωρητές (Registers) του PORTD.

Σημείωση: Για να δείτε τις τιμές των δηλωμένων σταθερών PIN1_bm και άλλων που θα δούμε στη συνέχεια, μπορείτε να ανατρέξετε στο header file iom4808.h, το οποίο βρίσκεται στον φάκελο που έχετε εγκαταστήσει το Microchip Studio :

π.χ. Path, “~\Studio\7.0\packs\atmel\ATmega_DFP\1.6.364\include\avr\iom4808.h”

7 Παράδειγμα 2: Διακοπή (Interrupt)

Οι διακόπτες (switches) που μπορούν να χρησιμοποιηθούν βρίσκονται στο PORTF και είναι τα PIN5 και PIN6 (ανατρέξτε στο Board Overview). Για τη χρήση ενός διακόπτη (switch), το pullup enable bit που του αντιστοιχεί πρέπει να είναι ενεργοποιημένο (ανατρέξτε σελ. 158 στο ATmega4808 DataSheet). Επίσης, το σημείο του παλμού στο οποίο θα ενεργοποιηθεί η μονάδα διαχείρισης της διακοπής (interrupt) πρέπει να οριστεί. Εδώ η ενεργοποίησή της και στις δύο άκρες του παλμού επιλέγεται (ανατρέξτε σελ. 158 στο ATmega4808 DataSheet). Με την ενεργοποίηση των συγκεκριμένων ψηφίων (bits) του PIN5, το σύστημα μπορεί να δεχτεί διακοπές (interrupts) όταν πατηθεί ο διακόπτης (switch).

Όταν γίνει η διακοπή (interrupt), συγκεκριμένη συνάρτηση η οποία θα το διαχειριστεί, ενεργοποιείται. Αυτή η συνάρτηση είναι μια ρουτίνα διαχείρισης διακοπών (ISR routine) η οποία παίρνει ως όρισμα μια διεύθυνση (interrupt vector) το οποίο αντιστοιχεί σε μία διακοπή. Για παράδειγμα, το interrupt vector για τη διακοπή του PINF είναι το `PORTF_PORT_vect`. Στο αρχείο `iom4808.h` υπάρχουν όλα τα interrupt vectors που μπορούν να χρησιμοποιηθούν από το συγκεκριμένο board.

Ο κώδικας της υλοποίησης του παραδείγματος, παρουσιάζεται παρακάτω:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define del 10
int x=0; //logic flag

int main() {

    PORTD.DIR |= 0b00000010; //PIN is output
    PORTD.OUT |= 0b00000010; //LED is off
    //pullup enable and Interrupt enabled with sense on both edges
    PORTF.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc;
    sei(); //enable interrupts
    while (x==0) {

        PORTD.OUTCLR= 0b00000010; //on

    }
    PORTD.OUT |= 0b00000010; //off
    cli(); //disable interrupts
}

ISR(PORTF_PORT_vect){
    //clear the interrupt flag
    int y = PORTF.INTFLAGS;
    PORTF.INTFLAGS=y;
    x=1;
}
```

Σχήμα 0.12 : Κώδικας Παραδείγματος 2

Στο σύστημα υπάρχει ένα LED και ένας διακόπτης (switch). Σύμφωνα με την κανονική λειτουργία του προγράμματος, το LED ενεργοποιείται και απενεργοποιείται με περίοδο $T=10\text{ms}$. Όταν πατηθεί ο διακόπτης ενεργοποιείται η διακοπή (interrupt) και αλλάζει η τιμή μιας τυχαίας μεταβλητής (int x) ώστε να σταματήσει η λειτουργία του LED και του προγράμματος.

Για να ενεργοποιηθεί η διακοπή (interrupt) το ψηφίο (bit) 5 του register INTFLAGS στο PORTF πρέπει να πατηθεί, εφόσον το πρόγραμμα είναι σε κάποιο breakpoint. Μόλις το ψηφίο (bit) αλλάξει από '0' σε '1' και το πρόγραμμα συνεχίσει με την επόμενη εντολή (STEP OVER), θα ενεργοποιηθεί η ρουτίνα διαχείρισης διακοπών (interrupt routine).

8 Παράδειγμα 3: Χρονιστής (Timer)

Για να λειτουργήσει ο χρονιστής TCA0 timer σε κανονική λειτουργία (normal mode) και να ενεργοποιηθεί διακοπή (interrupt), όταν φτάσει σε μία προβλεπόμενη τιμή θα πρέπει:

- Να δοθεί η τιμή '0' στον CTRLB register (Normal Mode).
- Να δοθεί η τιμή '0' στον CNT register (θέτουμε τον χρονιστή – timer στο μηδέν).
- Να δοθεί η προβλεπόμενη τιμή στον CMP0 register.
- Να ενεργοποιηθούν οι διακοπές (interrupts) μέσω του INTCTRL register.
- Να τεθεί η συχνότητα ρολογιού (clock frequency).
- Τέλος, να ενεργοποιηθεί ο timer μέσω του CTRLA register.

Σημείωση: Για περισσότερες λεπτομέρειες μπορείτε να ανατρέξετε στη σελ. 243, στο ATmega4808 DataSheet.

Ο TCA0 θα αρχίσει να μετράει και όταν θα φτάσει την τιμή που τέθηκε στον CMP0 θα ενεργοποιηθεί η ISR routine με όρισμα το vector TCA0_CMP0_vect. Αυτό είναι το interrupt vector που έχει οριστεί για τον συγκεκριμένο χρονιστή/μετρητή (timer/counter).

Ο κώδικας της υλοποίησης του παραδείγματος, παρουσιάζεται παρακάτω:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#define ped 20

int x=0;

int main() {

    PORTD.DIR |= 0b00000010; //PIN is output
    PORTD.OUTCLR= 0b00000010; //LED is on
    //(σελ 219, 224, 205) 16-bit counter high and low
    TCA0.SINGLE.CNT = 0; //clear counter
    //Normal Mode (TCA_SINGLE_WGMODE_NORMAL_gc σελ 207)
    TCA0.SINGLE.CTRLB = 0;
    //When CMP0 reaches this value -> interrupt //CLOCK FREQUENCY/1024
    TCA0.SINGLE.CMP0 = ped;
    TCA0.SINGLE.CTRLA = 0x7<<1; //TCA_SINGLE_CLKSEL_DIV1024_gc σελ 224
    TCA0.SINGLE.CTRLA |=1; //Enable
    TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm; //Interrupt Enable (=0x10)
    sei(); //begin accepting interrupt signals
    while (x==0) {

        ;

    }
    PORTD.OUT |= PIN1_bm; //LED is off
    cli();
}

ISR(TCA0_CMP0_vect){
    TCA0.SINGLE.CTRLA = 0; //Disable
    //clear flag
    int intflags = TCA0.SINGLE.INTFLAGS;
    TCA0.SINGLE.INTFLAGS=intflags;
    x=1;
}
```

Σχήμα 0.13 : Κώδικας Παραδείγματος 3

Ενεργοποιείται το LED και ο χρονιστής (timer). Όταν ο χρονιστής φτάσει την τιμή που έχει τεθεί αρχικά, τότε:

- Ενεργοποιείται η ρουτίνα διαχείρισης της διακοπής.
- Αλλάζει μια τυχαία μεταβλητή (x), για να μπορεί να ελεγχθεί μέσα στην συνάρτηση main αν εκτελέστηκε η ρουτίνα διαχείρισης της διακοπής ώστε το πρόγραμμα να συνεχίσει με την επόμενη λειτουργία.
- Το πρόγραμμα ολοκληρώνεται απενεργοποιώντας το LED.

Σημείωση: Μπορείτε να χρησιμοποιείτε τα *breakpoints* για να δείτε εάν ένα *interrupt* ενεργοποιείται.

Η τιμή του καταχωρητή CPRO (value) ορίζει το χρονικό διάστημα μέχρι να κάνει διακοπή (interrupt) ο χρονιστής (timer), (γενικά να τελειώσει και να αρχίσει από την αρχή). Ο τύπος για τον υπολογισμό είναι ο παρακάτω:

$$f_{timer} = \frac{f_{system}}{N_{prescaler}}$$

$$value = T * f_{timer}$$

Ο ATmega4808 λειτουργεί σε μέγιστο F=20 MHz και το $N_{prescaler}$ παίρνει την τιμή που έχουμε ορίσει εμείς για συχνότητα ρολογιού (clock frequency). Για παράδειγμα στον κώδικά μας δώσαμε τιμή value=20 άρα ο χρόνος του χρονιστή (timer) είναι:

$$f_{timer} = \frac{20MHz}{1024} = 19,531KHz$$

$$T = \frac{value}{f_{timer}} = 1,024ms$$

9 Παράδειγμα 4: Λειτουργία του Μετατροπέα Αναλογικού σε Ψηφιακό (Analog to Digital Converter – ADC)

Ο Μετατροπέας Αναλογικού σε Ψηφιακό (Analog to Digital Converter – ADC) είναι ένα σύστημα το οποίο μετατρέπει αναλογικά σήματα που έρχονται ως είσοδοι στα κατάλληλα PINs, όπως ήχος, ρεύμα, φως, κτλ., σε ψηφιακά σήματα. Επομένως, οι τιμές των αναλογικών σημάτων μπορούν να διαβαστούν πλέον από τον μικροελεγκτή και να αξιοποιηθούν κατάλληλα. Ο ADC που εμπεριέχεται στην πλακέτα δέχεται από το PIN7 του PORTD μετρήσεις. Επίσης, τα βασικά modes λειτουργίας του ADC είναι δύο. Το free-running mode στο οποίο ο ADC συνεχώς δέχεται τιμές και τις μετατρέπει σε ψηφιακά σήματα. Η δεύτερη και η προκαθορισμένη λειτουργία του (single-conversion mode) εκτελεί μόνο μία μετατροπή όποτε ζητηθεί στον κώδικα και μετά σταματά.

Στο παράδειγμα, όταν ο ADC θα διαβάζει μετρήσεις που είναι κάτω από μία τιμή (για παράδειγμα RES=10) θα ενεργοποιείται μια διακοπή (interrupt). Στη διακοπή (interrupt) θα ανάβει ένα LED για T=5ms και μετά θα επιστρέφουμε στην αρχική ροή και θα περιμένουμε ξανά την μέτρηση του ADC.

Για την ενεργοποίηση του ADC:

- 1 Πρώτα το resolution ορίζεται σε 10-bit ή 8-bit → Resolution Selection bit (RESSEL) στον Control A register (ADCn.CTRLA).
- 2 Έπειτα, το Free-Running Mode ενεργοποιείται → '1' στο Free-Running bit (FREERUN) στο ADCn.CTRLA. Στην περίπτωση που ζητείται να εκτελεστεί η προκαθορισμένη λειτουργία του ADC (single-conversion mode), δηλαδή να εκτελεστεί μία μόνο μετατροπή όποτε ζητηθεί, αυτό το ψηφίο (bit) δεν ενεργοποιείται και παραμένει '0'.
- 3 Ορίζεται το ψηφίο (bit) με το οποίο θα συνδεθεί ο ADC → MUXPOS bit field στο MUXPOS register (ADCn.MUXPOS).
- 4 Ο ADC ενεργοποιείται → Γράφουμε '1' στο ENABLE bit στο ADCn.CTRLA.
- 5 Δίνεται η επιλογή της ενεργοποίησης του Debug Mode ώστε ο ADC να μην σταματά ποτέ να τρέχει και να λαμβάνει τιμές.

Με τη λειτουργία του Window Comparator Mode ελέγχονται όλες οι τιμές που εισάγονται από τον ADC με μια δοθείσα τιμή, δηλαδή ένα κατώφλι (threshold). Για να ενεργοποιηθεί αυτή η λειτουργία πρέπει να ακολουθηθούν τα παρακάτω βήματα:

- 1 Πρώτα εισάγεται το κατώφλι (threshold) στον καταχωρητή ADCn.WINLT και/ή ADCn.WINHT.
- 2 Ενεργοποιείται η λειτουργία δημιουργίας διακοπών (interrupts) → Window Comparator Interrupt Enable bit (WCOMP) στον Interrupt Control register (ADCn.INTCTRL).
- 3 Ορίζεται το επιθυμητό Mode (στην περίπτωσή μας θέλουμε διακοπές – interrupt όταν RESULT<THRESHOLD) → WINCM bit field στον ADCn.CTRLE.

Εφόσον προγραμματιστεί κατάλληλα ο ADC και οι λειτουργίες του, απομένει να γραφτεί το λογικό '1' στο Start Conversion Bit (STCONV) στον Command Register (ADCn.COMMAND), το οποίο εκκινεί το conversion. Οι τιμές καταγράφονται στον καταχωρητή RES (Result). Αυτή η εντολή πρέπει να εκτελεστεί μόνο εφόσον προγραμματιστεί εξ' ολοκλήρου ο ADC με τους παραπάνω τρόπους που εξηγήθηκαν. Στο Free-Running Mode αρκεί να ενεργοποιηθεί μία φορά αυτή η εντολή ώστε συνεχώς να γίνονται μετατροπές νέων τιμών από αναλογικό σε ψηφιακό. Στην προκαθορισμένη λειτουργία (single-conversion mode) κάθε φορά που απαιτείται μία μετατροπή νέων τιμών, πρέπει να εκτελείται η εντολή Start Conversion ώστε ο ADC να μετατρέψει το καινούριο σήμα εισόδου από αναλογικό σε ψηφιακό.

Συμβουλή: Ανατρέξτε στο ATmega4808 DataSheet και διαβάστε καλά τις σελίδες 394-421. Αναγράφονται αναλυτικά όλες οι πιθανές λειτουργίες που μπορείτε να χρησιμοποιήσετε για να κάνετε Analog to Digital Conversion και πως να τις προγραμματίσετε.

Ο κώδικας της υλοποίησης του παραδείγματος παρουσιάζεται παρακάτω:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

int main(){

    PORTD.DIR |= PIN1_bm; //PIN is output
    //initialize the ADC for Free-Running mode
    ADC0.CTRLA |= ADC_RESSEL_10BIT_gc; //10-bit resolution
    ADC0.CTRLA |= ADC_FREERUN_bm; //Free-Running mode enabled
    ADC0.CTRLA |= ADC_ENABLE_bm; //Enable ADC
    ADC0.MUXPOS |= ADC_MUXPOS_AIN7_gc; //The bit //Enable Debug Mode
    ADC0.DBGCTRL |= ADC_DBGRUN_bm; //Window Comparator Mode
    ADC0.WINLT |= 10; //Set threshold
    ADC0.INTCTRL |= ADC_WCMP_bm; //Enable Interrupts for WCM
    ADC0.CTRLB |= ADC_WINCM0_bm; //Interrupt when RESULT < WINLT
    sei();
    ADC0.COMMAND |= ADC_STCONV_bm; //Start Conversion
    while(1){

        ;

    }

}

ISR(ADC0_WCOMP_vect){
    int intflags = ADC0.INTFLAGS;
    ADC0.INTFLAGS = intflags;
    PORTD.OUTCLR= PIN1_bm; //LED is on
    _delay_ms(5);
    PORTD.OUT |= PIN1_bm; //LED is off
}
```

Σχήμα 0.14 : Κώδικας Παραδείγματος 4

Σημείωση: Για την προσομοίωση της λειτουργίας του ADC πρέπει να γράφεται εσείς την τιμή εισόδου στον καταχωρητή RES (Result) χειροκίνητα.

Εργαστηριακή Άσκηση 01:

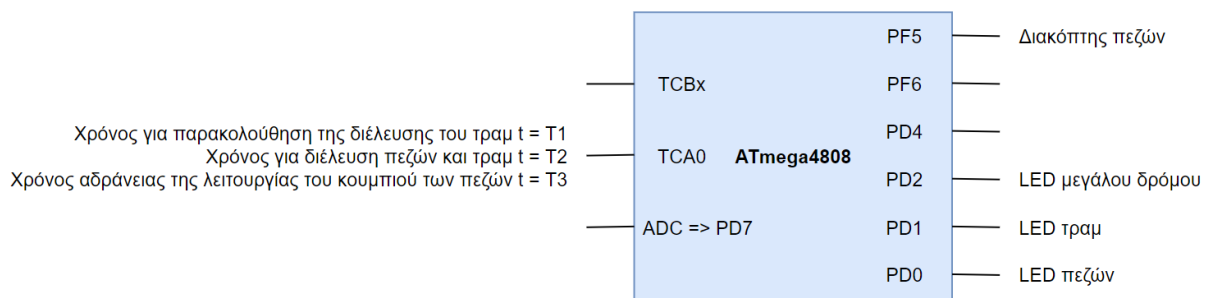
Φανάρια Κυκλοφορίας (Traffic Lights)

1 Περιγραφή

Σκοπός της Εργαστηριακής Άσκησης είναι η προσομοίωση της λειτουργία μιας διασταύρωσης αυτοκινητόδρομου, η οποία αποτελείται από ένα μεγάλο δρόμο και έναν κάθετο σιδηρόδρομο. Στον μεγάλο δρόμο υπάρχει ένα φανάρι για τα αυτοκίνητα και ένα φανάρι για τους πεζούς, που ανάβει μόνο μετά από πίεση ενός κουμπιού (push button). Τραμ περνάει από τον σιδηρόδρομο ανά τακτά χρονικά διαστήματα $t = T1$ και διακόπτει την κανονική κυκλοφορία, δηλαδή το κόκκινο φανάρι για το μεγάλο δρόμο και το πράσινο για τους πεζούς ενεργοποιείται για ένα χρονικό διάστημα $t = T2$. Επίσης, όταν πατηθεί το κουμπί των πεζών, ανάβει το κόκκινο φανάρι για τα αυτοκίνητα, για τον μεγάλο δρόμο μαζί με το πράσινο φανάρι για τους πεζούς, για το ίδιο χρονικό διάστημα $t = T2$. Μετά το πέρας του διαστήματος $t = T2$, το κόκκινο φανάρι των πεζών και το πράσινο φανάρι του μεγάλου δρόμου ενεργοποιούνται. Για να μπορέσει να πατηθεί ξανά το κουμπί των πεζών (δηλαδή να ενεργοποιηθεί το πράσινο φανάρι για τους πεζούς) πρέπει να έχει περάσει ένα χρονικό διάστημα $t = T3$. Στην περίπτωση που πατηθεί το κουμπί του φαναριού των πεζών και είναι ήδη ενεργό (είτε έχει πατηθεί ήδη μια φορά είτε περνάει το τραμ στο σιδηρόδρομο) η λειτουργία που εκτελείται δεν πρέπει να διακοπεί και τα ενεργά χρονικά διαστήματα πρέπει να διατηρηθούν μέχρι την ολοκλήρωσή τους.

Για την υλοποίηση της Εργαστηριακής Άσκησης 1, θεωρείστε ότι:

- Το φανάρι είναι πράσινο όταν το LED είναι ανοιχτό (λογικό '0') και κόκκινο όταν είναι σβηστό (λογικό '1'). Τα τρία PIN του PORTD που χρησιμοποιούνται είναι τα PIN0, PIN1 και PIN2. Για την προσομοίωση του τραμ, το αντίστοιχο LED (PIN1) θα είναι ενεργοποιημένο όταν το τραμ περνάει από το σημείο αυτό του σιδηρόδρομου και απενεργοποιημένο σε διαφορετική περίπτωση.
- Με διακοπή (interrupt) πρέπει να υλοποιηθεί το πάτημα του κουμπιού των πεζών. Η χρήση του PIN5 του PORTF θα χρησιμοποιηθεί.
- Ο χρονιστής/μετρητής TCA0 μπορεί να χωριστεί σε δύο μικρότερους χρονιστές/μετρητές των 8-bit ο καθένας, με το σετ των εντολών TCA0.SPLIT. Ο ένας από αυτούς θα χρησιμοποιηθεί για τα δύο χρονικά διαστήματα $t = T2$ και $t = T3$. Ο δεύτερος για το χρονικό διάστημα $t = T1$ το οποίο ορίζει κάθε πότε περνάει τραμ, με την αντίστοιχη λειτουργία να ενεργοποιείται. Ορίστε εσείς τις μονάδες χρόνου, που σας διευκολύνουν, για την προσομοίωση, αλλά εξηγήστε τον τρόπο σκέψης και τη διαδικασία του υπολογισμού των τιμών (values).



Σχήμα 1.1 : Απεικόνιση ATmega4808, Εργαστηριακής Άσκησης 01

Χρήσιμη οδηγία: Δείτε τα παραπάνω παραδείγματα 1, 2, και 3.

2 Ερωτήματα Εργαστηριακής Άσκησης 1

- 1 Υλοποιήστε την λειτουργία της διαχείρισης των φαναριών του μεγάλου δρόμου και των πεζών, χρησιμοποιώντας όλες τις διαδικασίες που εξηγήθηκαν.
- 2 Προσθέστε την λειτουργία διαχείρισης του τραμ και του χρονικού διαστήματος ανά το οποίο περνάει από το συγκεκριμένο σημείο του σιδηρόδρομου.

3 Αναφορά Εργαστηριακής Άσκησης 1

- 1 Αναπτύξτε τον κώδικα σας και βεβαιωθείτε πως όλες οι λειτουργίες επιτελούνται, όπως προβλέπεται.
- 2 Παραδίδετε αναλυτική περιγραφή, με τη λειτουργία του κώδικά σας, μαζί με διάγραμμα ροής. Εξηγήστε τον τρόπο με τον οποίο ο κώδικάς σας εκτελεί όλες τις διαδικασίες, της εφαρμογής που αναλύθηκε παραπάνω, καθώς και τα πιθανά στοιχεία (interrupts, timers, κ.α.), που επιλέξατε και χρησιμοποιήσατε.

*** Σημείωση: Παραδίδετε τον κώδικα ανά ομάδα και τα αναλυτικά σχόλια, για τις εντολές που χρησιμοποιήσατε.