**26/5/2019**

# 4ο Εργαστήριο Οργάνωσης Υπολογιστών
# ΟΛΟΚΛΗΡΩΣΗ ΕΠΕΞΕΡΓΑΣΤΗ ΠΟΛΛΑΠΛΩΝ ΚΥΚΛΩΝ (DATAPATH & CONTROLPATH) ΚΑΙ ΠΡΟΣΘΗΚΗ ΕΞΑΙΡΕΣΕΩΝ
# 5/4/2019

Ομάδα LAB31239632

| |
|---|
| *Αλέξανδρος Πουπάκης 2016030061* |
| *Βαγγέλης Κιούλος 2016030056* |


## Σκοπός εργαστηριακής άσκησης

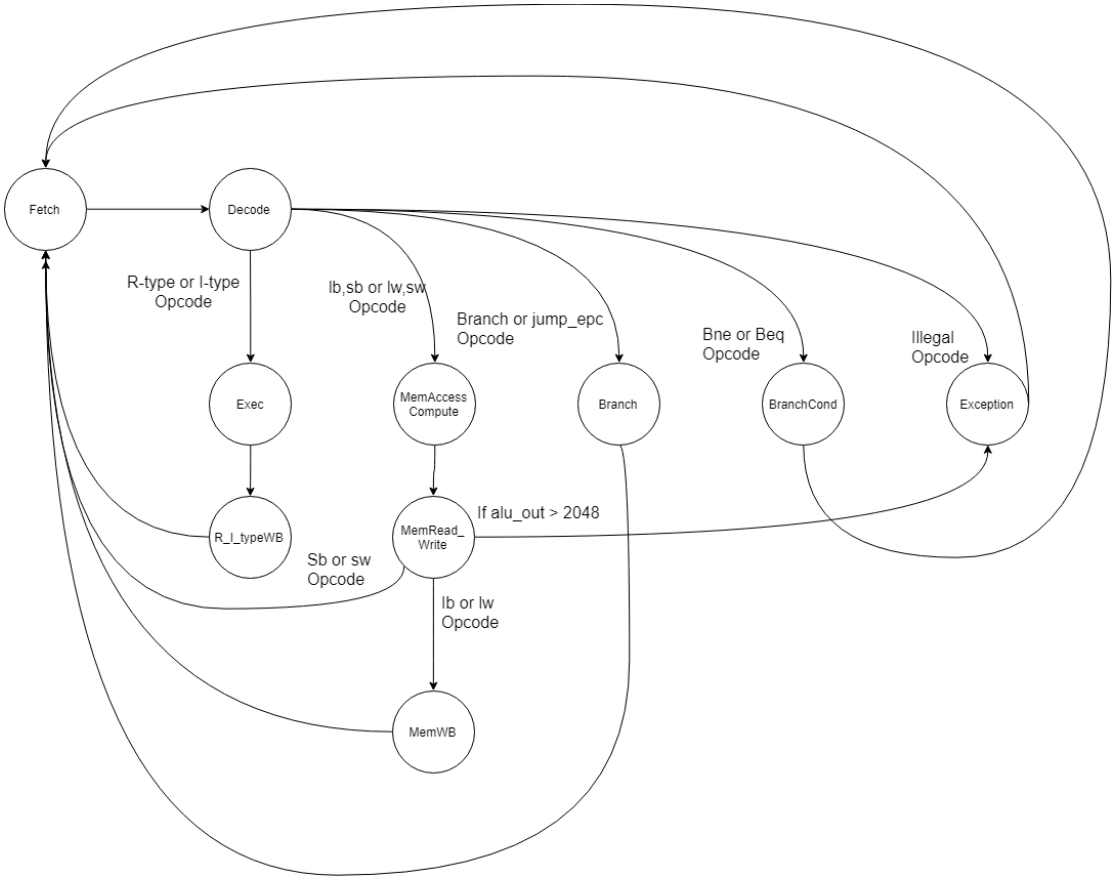
Στο εργαστήριο αυτό έπρεπε να τροποποιήσουμε το υπάρχον Datapath ώστε από single-cycle να γίνει multi-cycle, να υλοποιήσουμε το Control, να προσθέσουμε Exceptions και τέλος να τα συνδέσουμε μεταξύ τους και με τη μνήμη ώστε να προκύψει ένας πλήρης επεξεργαστής. Η επαλήθευση του επεξεργαστή θα γίνει μέσω εκτέλεσης του δοθέντος προγράμματος αναφοράς, καθώς και αυτά του προηγούμενου εργαστηρίου.

## Προεργασία

Με έτοιμο το Datapath του προηγούμενου εργαστηρίου, παράχθηκε ο παρακάτω πίνακας με τις τιμές των σημάτων ελέγχου.

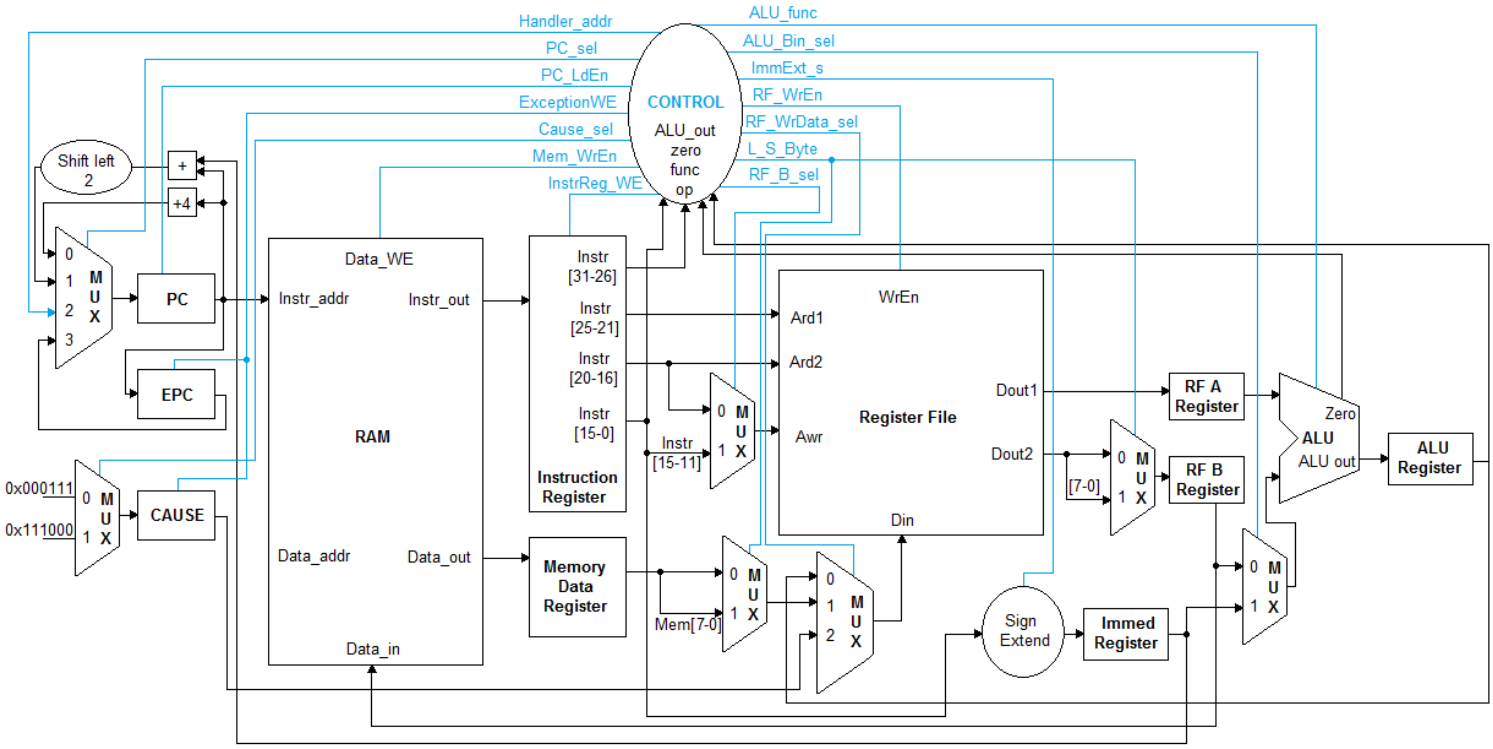
| | **R-TYPE** | **I-TYPE** | b | beq | bne | lb | lw | sb | sw |
|---|---|---|---|---|---|---|---|---|---|
| PC_sel | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| PC_LdEn | 1 | 1 | 1 | zero == 0? | zero != 0? | 1 | 1 | 1 | 1 |
| RF_WrEn | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| RF_WrData_sel | 0 | 0 | **d** | **d** | **d** | 1 | 1 | **d** | **d** |
| RF_B_sel | 0 | **d** | **d** | 1 | 1 | **d** | **d** | 1 | 1 |
| ImmExt_s | **d** | *** | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ALU_Bin_sel | 0 | 1 | **d** | 0 | 0 | 1 | 1 | 1 | 1 |
| ALU_func | func | func | **d** | 110001 | 110001 | 110000 | 110000 | 110000 | 110000 |
| Mem_WrEn | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | | | | | | | | | |
| | | *** | (li, addi) -> 1, (nandi, ori) -> 0 | | | | | | |

Έπειτα σχεδιάστηκε η ακόλουθη FSM για το Control.

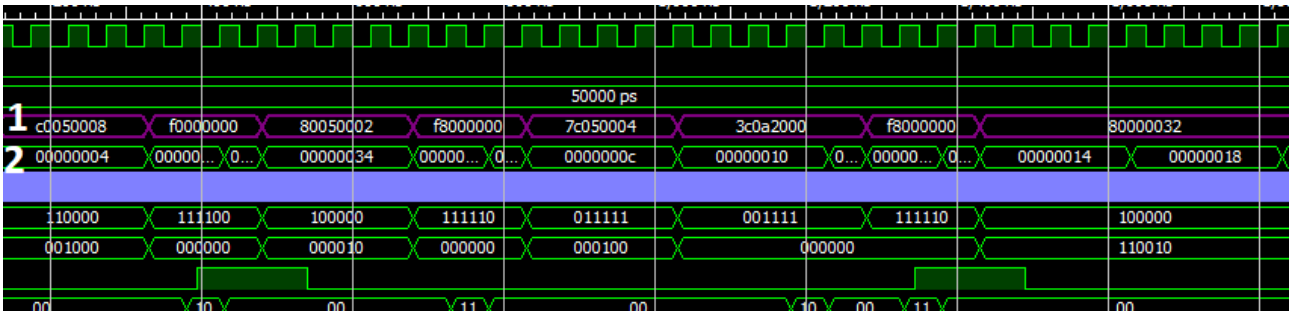



## Περιγραφή

Αρχικά υλοποιήθηκε η FSM στο αρχείο CPU_Control.vhd. Έπειτα, έπρεπε να γίνουν διάφορες αλλαγές στο Datapath έτσι ώστε αφενός να λειτουργεί σε multi-cycle επεξεργαστή και αφετέρου να υποστηρίζει Exceptions. Με τις αλλαγές αυτές και την FSM που περιγράφθηκε προηγουμένως, συνδέσαμε την μνήμη, το Control και το Datapath σε ένα top module με όνομα processor.vhd. Έτσι, προκύπτει το ακόλουθο ολοκληρωμένο Datapath, συνδεδεμένο κατάλληλα με το Control.



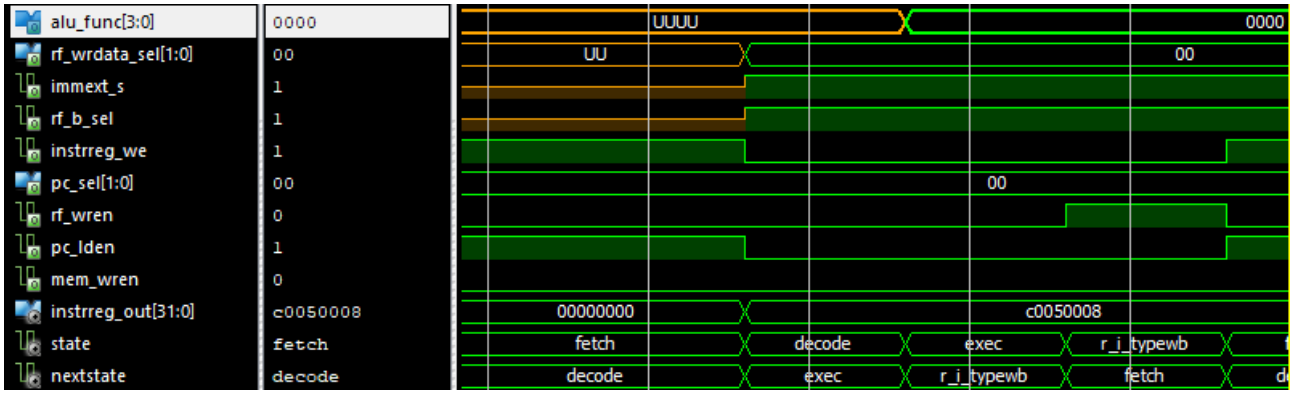

## Κυματομορφές - Προσομοίωση




Παραπάνω, παρατηρούμε τις κυματομορφές που προέκυψαν από την εκτέλεση του προγράμματος του τετάρτου εργαστηρίου. Αρχικά στην σειρά με την ένδειξη (1)(μοβ χρώμα), παρατηρούμε τα περιεχόμενα του instruction register, δηλαδή τις εντολές του προγράμματος μας και την σειρά με την οποία εκτελούνται. Βλέπουμε ότι, πρώτα, εκτελείται η εντολή “addi r5,r0,8”(c0050008) και ο καταχωρητής pc έχει την τιμή 0x00, όπως φαίνεται στην σειρά 2. Στην συνέχεια, έχουμε μια παράνομη εντολή (f0000000) στην διεύθυνση 0x04 καθώς το opcode δεν υπάρχει στο δοσμένο instruction set, οπότε, η τιμή του pc θα γίνει 0x30, που είναι η διεύθυνση του handler στην περίπτωση άγνωστης εντολής. Επομένως, στον καταχωριστή exception pc, θα αποθηκευτεί η διεύθυνση της εντολής που προκάλεσε το exception και θα εκτελεστεί η εντολή “move_cause r5”(80050002), η οποία καταχωρεί την τιμή του cause register στην τιμή του καταχωρητή r5.

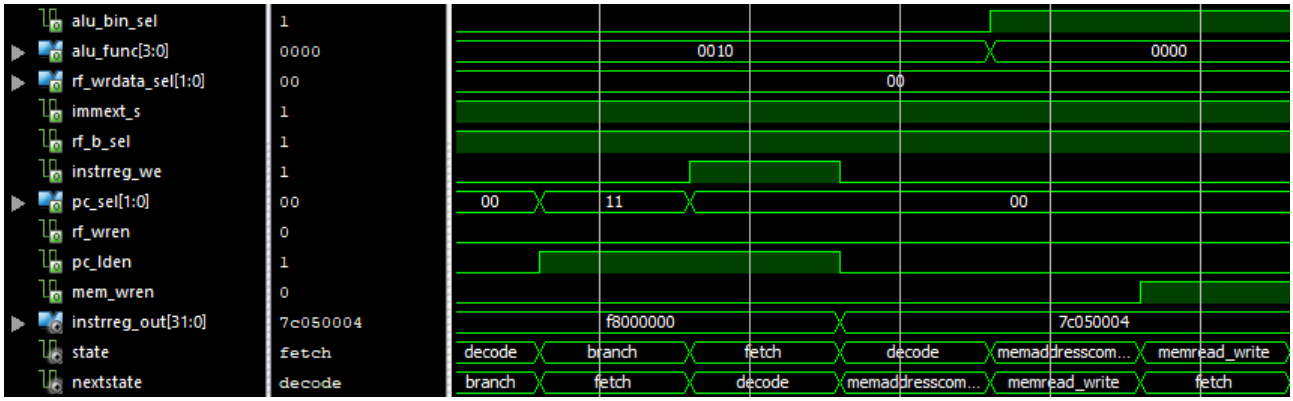
Έπειτα, εκτελείται η εντολή jump_epc (f8000000) η οποία βρίσκεται στην διεύθυνση 0x34 και καταχωρεί την τιμή του exception pc, στον καταχωρητή pc. Επομένως, η επόμενη εντολή που εκτελείται είναι η εντολή “sw r5,4(r0)”(7c050004), η οποία βρίσκεται στην διεύθυνση 0x08 και θα αποθηκεύσει την τιμή του καταχωρητή r5, δηλαδή την τιμή που είχε ο cause register, στην διεύθυνση 0x404. Στην συνέχεια θα εκτελεστεί η εντολή “lw r10, 8192(r0)”(3c0a2000), η οποία βρίσκεται στην διεύθυνση 0x0c και θα προκαλέσει exception καθώς η διεύθυνση στην οποία θέλουμε να αποθηκεύσουμε δεν υπάρχει. ‘Επομένως, η διεύθυνση του pc θα γίνει 0x040, που είναι η τιμή του handler στην περίπτωση ανάγνωσης λανθασμένης εντολής. Επομένως, το πρόγραμμα θα μεταβεί στην διεύθυνση 0x040 όπου εκτελείται η εντολή jump_epc. Τέλος, ο καταχωρητής epc, καταχωρεί την τιμή του, δηλαδή 0x10 στον καταχωρητή pc οπότε το πρόγραμμα εκτελεί την εντολή “nand r0,r0,r0”(80000032).

**R/I-type instructions:**



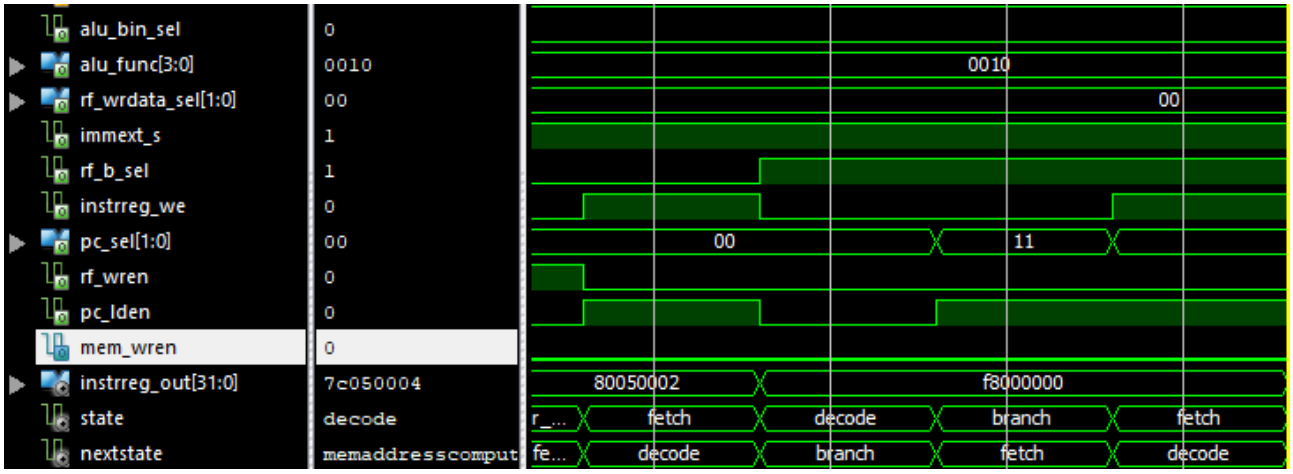

Παραπάνω, βλέπουμε την αλλαγή των καταστάσεων της control FSM, όταν εκτελούμε μια i-type εντολή(addi). Παρατηρούμε ότι, η πρώτη κατάσταση της FSM, είναι η κατάσταση στην οποία έχουμε instruction fetch και οι έξοδοι της fsm είναι “insrReg_we = 1” για να γράψουμε την εντολή στον instruction register, “pc_sel = 00” για να δώσουμε στον pc την τιμή “pc+4”, “rf_wren =0” καθώς δεν θέλουμε να γράψουμε στο register file, “pc_lden = 1” για να δώσουμε την τιμή που θέλουμε στο pc και “mem_wren =0” καθώς δεν θέλουμε να γράψουμε τα δεδομένα μας στην μνήμη. Στην συνέχεια, μεταβαίνουμε στην κατάσταση decode, όπου οι έξοδοι της fsm είναι “rf_b_sel =1” καθώς η εντολή μας είναι i-type και “immext_s = 1” αφού χρειάζεται να κάνουμε επέκταση πρόσημου. Στην συνέχεια, μεταβαίνουμε στην κατάσταση exec, όπου έχουμε “alu_func = 0000” για να εκτελέσουμε την πράξη που χρειάζεται και “alu_bin_sel = 1” καθώς κάνουμε πράξη με immediate. Τέλος, μεταβαίνουμε στην κατάσταση r_i_typeWB, οπου έχουμε “rf_wren = 1”, για να γράψουμε το αποτέλεσμα της πράξης στο register file.

**Load/Store instructions:**



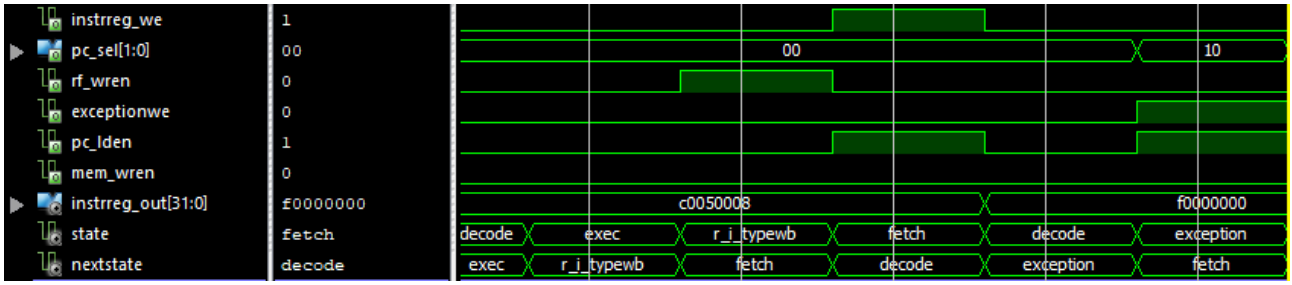

Παραπάνω, βλέπουμε την αλλαγή των καταστάσεων της control FSM, όταν εκτελούμε την εντολή sw. Μετά την κατάσταση decode, μεταβαίνουμε στην κατάσταση memAddressCompute, όπου έχουμε ως έξοδο “alu_bin_sel = 1” και “alu_func = 0000” για να υπολογίσουμε την διεύθυνση στην οποία θέλουμε να αποθηκεύσουμε τα δεδομένα μας. Τέλος, μεταβαίνουμε στην κατάσταση memRead_write, όπου έχουμε ως έξοδο “mem_wren = 1” για να γράψουμε τα δεδομένα μας στην μνήμη(mem_wren = 0 για τις εντολές load, στις οποίες ακολουθεί η κατάσταση memWb για την εγγραφή των δεδομένων από την μνήμη στους καταχωρητές).

**Branch instructions:**




Παραπάνω, βλέπουμε την αλλαγή των καταστάσεων της control FSM, όταν εκτελούμε την εντολή branch. Μετά την κατάσταση decode, μεταβαίνουμε στην κατάσταση branch, που ως έξοδο έχουμε “pc_sel = 01”(ή 11 για την εντολή jump_epc), για να δώσουμε στο pc την τιμή “pc+4+immed*4”.

**Exceptions:**




Παραπάνω, βλέπουμε την αλλαγή των καταστάσεων της control FSM, όταν προκύψει κάποιο exception. Παρατηρούμε ότι μετά την κατάσταση decode, μεταβαίνουμε στην κατάσταση exception , όπου ως έξοδο έχουμε “exceptionWE = 1” για να γράψουμε την διεύθυνση της εντολής που προκάλεσε το exception στον καταχωρητή epc, “pc_sel = 10” και “pc_lden = 1” για να πάρει ο pc την διεύθυνση του exception handler και τέλος, αν έχουμε την εντολή move_cause, έχουμε “cause_sel = 0” η “cause_sel = 1” για να δώσουμε στον cause register την τιμή που θέλουμε ανάλογα με το είδος του exception.

## Συμπεράσματα

- Οι εντολές χρειάζονται τέσσερεις με πέντε κύκλους για να εκτελεστούν ανάλογα με το είδος τους.
- Για την υλοποίηση των exceptions χρειάζονται οι καταχωρητές exception pc και cause.
- Για την παραγωγή των σημάτων ελέγχου δημιουργήθηκε η control FSM.
- Για την μετατροπή του datapath από single-cycle σε multi-cycle χρειάστηκε να προστεθούν οι κατάλληλοι καταχωρητές ανάμεσα στις βαθμίδες.