ΟΡΓΑΝΩΣΗ ΥΠΟΛΟΓΙΣΤΩΝ
ΗΡΥ 312

ΕΡΓΑΣΤΗΡΙΟ 3
Αναφορά Εργαστηριακής Άσκησης

Ομάδα Εργασίας **LAB31220188**

α/α	Α.Μ.	Ονοματεπώνυμο
1	2011030010	Χριστοδουλου Θεοφιλος
2	2011030009	ΚΑΡΙΜΠΙΔΗΣ ΔΙΟΝΥΣΗΣ

Στο 3ο εργαστήριο δημιουργήσαμε το ολοκληρωμένο datapath καθώς και το control ώστε να υλοποιούνται όλες οι εντολές που μας δόθηκαν κατά το 2ο εργαστήριο.

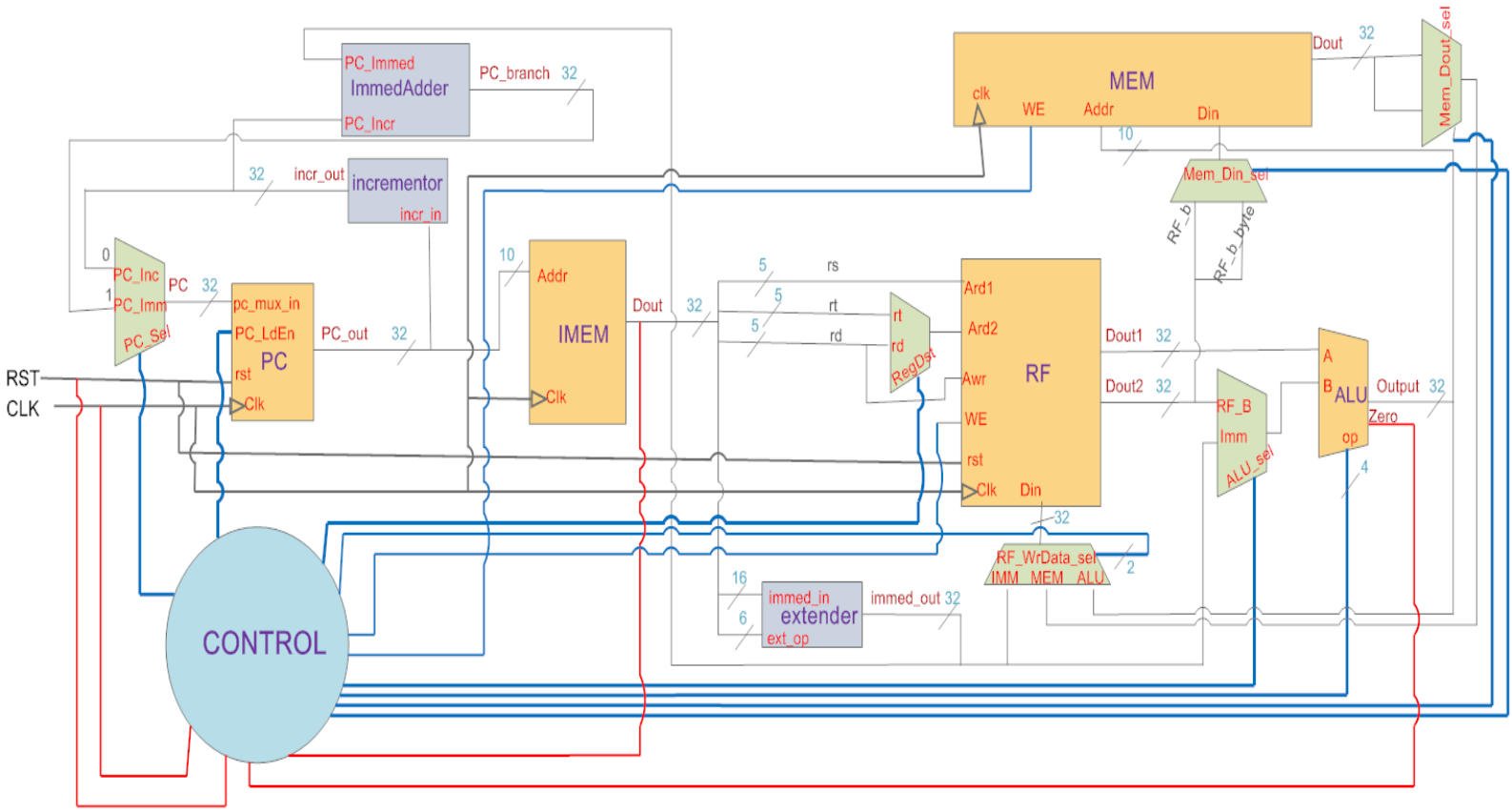
DATAPATH

Για να δημιουργήσουμε το ολοκληρωμένο datapath ουσιαστικά κάναμε τις κατάλληλες συνδέσεις στα 4 stages που είχαμε φτιάξει ήδη απο το προηγούμενο εργαστήριο, ενώ για την υλοποίησή μας δε χρειάστηκαν επιπρόσθετοι registers.

Επίσης δημιουργήσαμε ακόμη 2 ίδιους πολυπλέκτες, ένας για τις εντολές store και ένας για τις load, τους οποίους και ενσωματώσαμε στο MEMSTAGE. Ο **store_mux** παίρνει ως *είσοδο* την έξοδο RF_B απο την register file, και στην *έξοδό* του βγάζει ολόκληρο το σήμα αυτό για τις περιπτώσεις sw ή τα 8LSB με zerofill για τις περιπτώσεις sb και η έξοδος οδηγείται στην μνήμη.

Αντίστοιχα λειτουργεί και ο **load_mux** για τις περιπτώσεις lw και lb όπου ως *είσοδο* παίρνει την έξοδο της μνήνης και η έξοδός του οδηγείται σε έναν πολυπλέκτη που διαλέγει τι θα γράψει στην RF.

Οι συνδέσεις των επιμέρους στοιχείων έγιναν όπως φαίνονται αναλυτικά στο ακόλοθυθο block diagram..



Με κόκκινες γραμμές εμφανίζονται οι είσοδοι του Control, ενώ με μπλε οι έξοδοί του.

Control

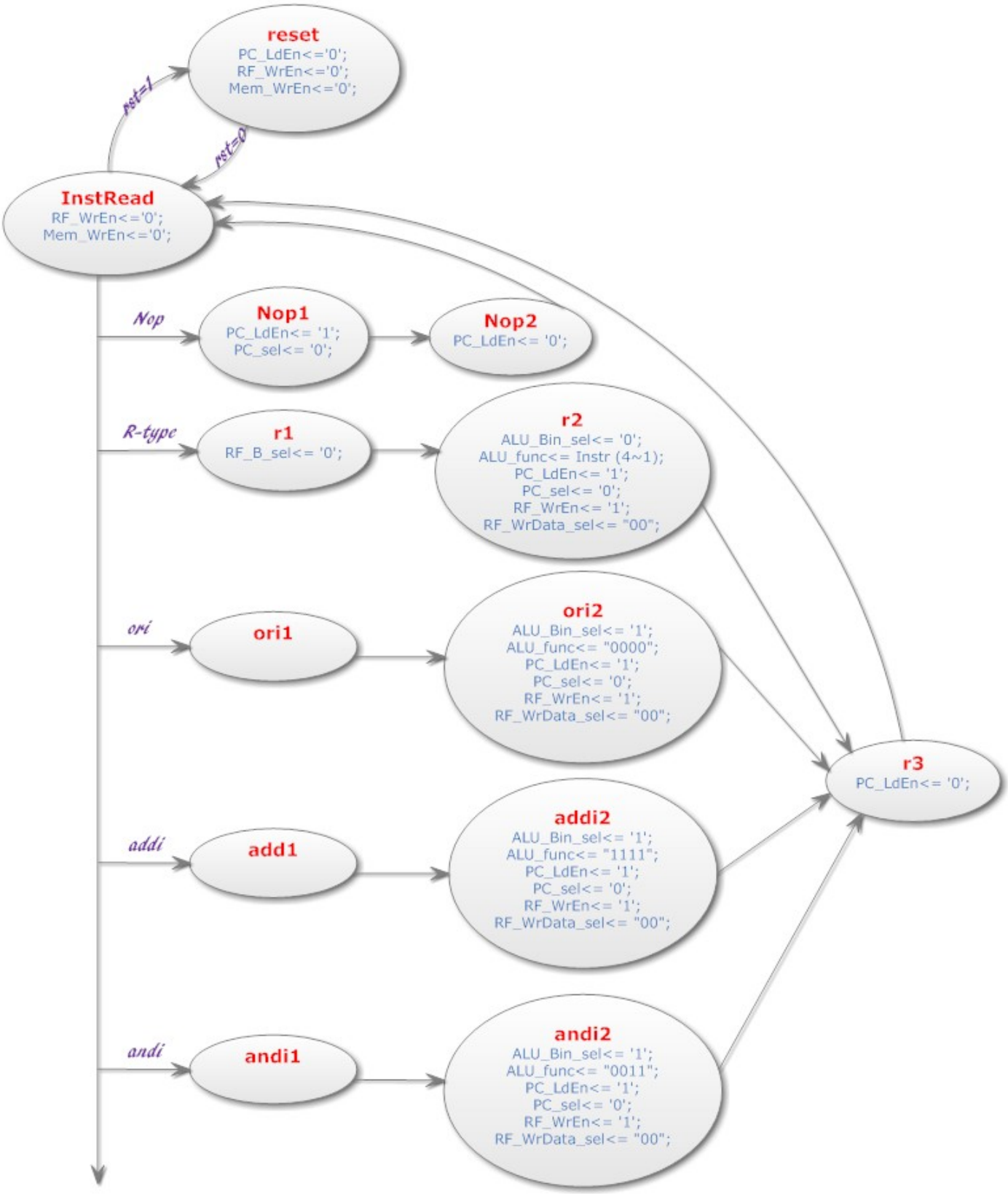
Το control module δίνει στο σύστημα όλα τα καθοριστικά σήματα για την εκτέλεση μιας εντολής. Όπως φαίνεται και παραπάνω, αυτά είναι τα:

PC_Sel, PC_LdEn,
RegDst, RF_WrData_sel, (RF) WE,
ALU_sel, ALU_op,
Mem_Din_sel,Mem_Dout_sel , (MEM)WE

Ως εισόδους παίρνει το *reset*, το *clk*, την 32-bit εντολή (*instr*) που διαβάστηκε από την Instruction Memory και το σήμα *Zero* της ALU.

Στην ουσία, πρόκειται για μία μηχανή πεπερασμένων καταστάσεων (FSM), που επεξεργάζεται τα σήματα που δέχεται ως εισόδους για να παράγει τα κατάλληλα σήματα εξόδου που θα οδηγήσουν στην εκτέλεση της εκάστοτε εντολής.

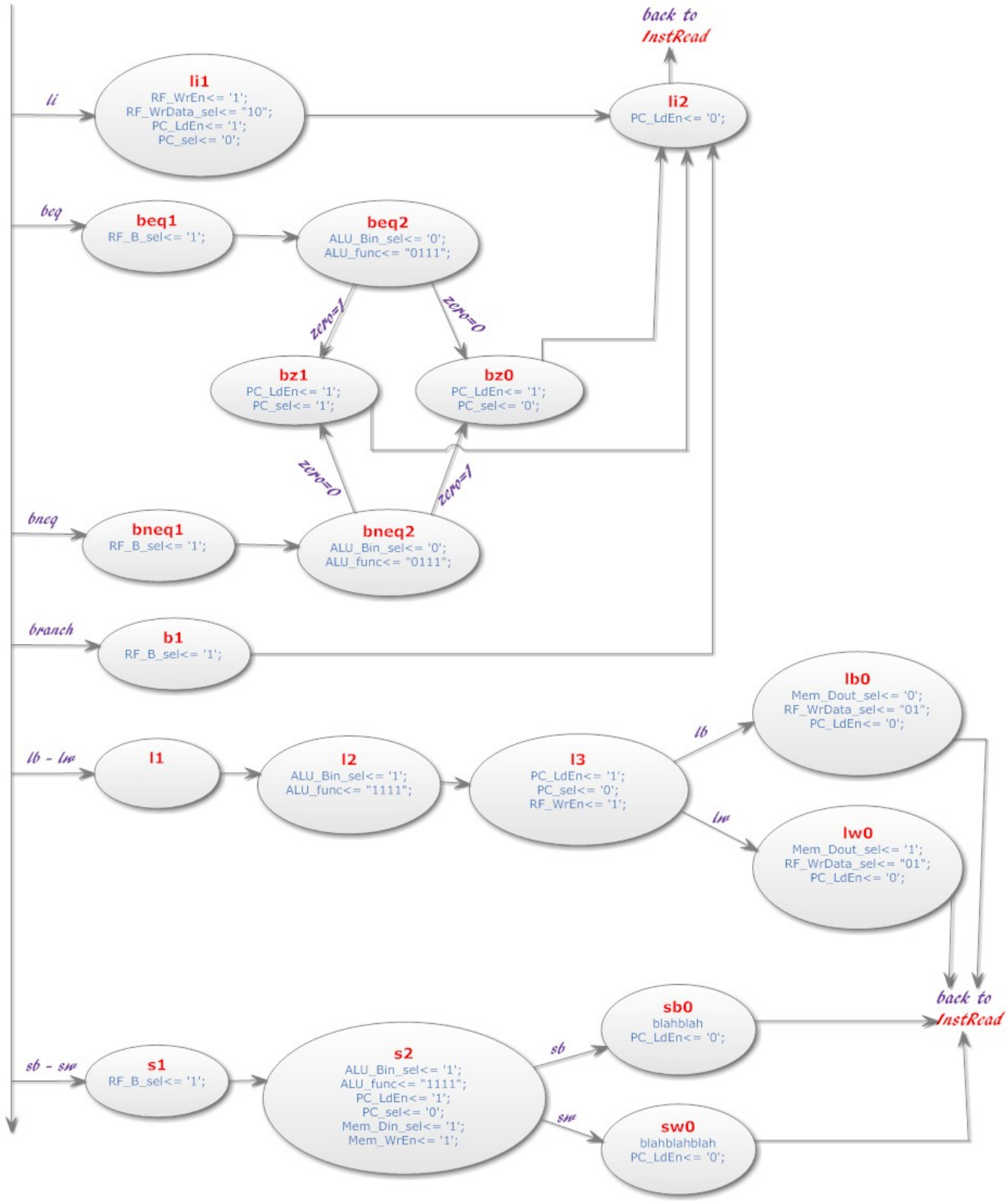
Η λειτουργία της FSM παρουσιάζεται συνοπτικά στο διάγραμμα που ακολουθεί..



Χριστοδούλου Θεόφιλος **A.M.: 2011030010**

Καριμπίδης Διονύσης **A.M.: 2011030009**

Μάρτιος 2014



Χριστοδούλου Θεόφιλος **A.M.: 2011030010**

Καριμπίδης Διονύσης **A.M.: 2011030009**

Μάρτιος 2014

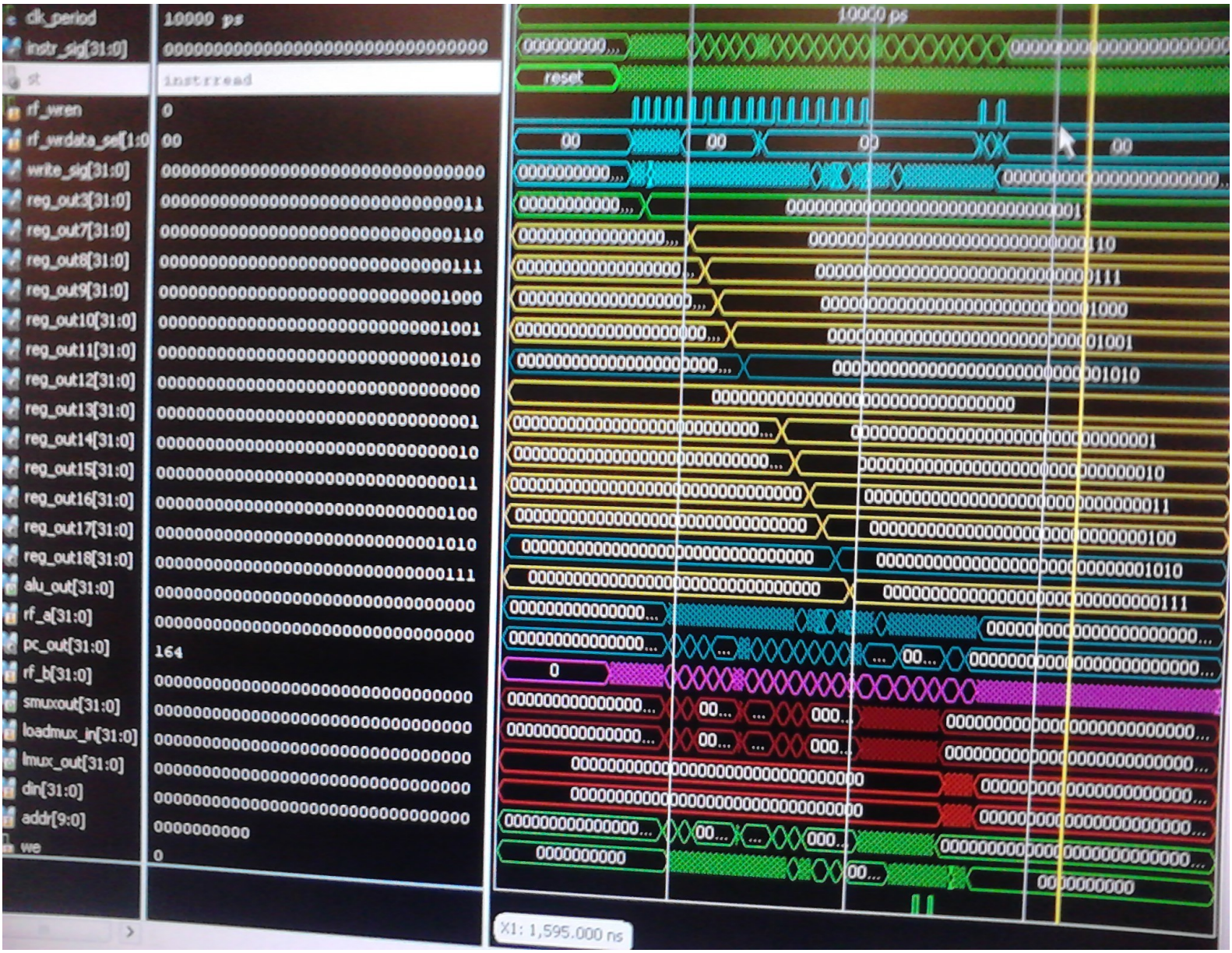
CRASH TEST

Τα επιμέρους components είχαν δοκιμαστεί και ελεγχθεί εκτενώς στο προηγούμενο εργαστήριο (εκτός από τους 2 νέους πολυπλέκτες), όπου αποδείξανε την ορθή λειτουργία τους, ακόμα και υπό αντίξοες συνθήκες.

Το ζητούμενο, λοιπόν, ήταν ο σωστός χρονισμός των σημάτων του Control, ώστε να οδηγούνται σωστά τα κατάλληλα σήματα εντός του datapath.

Το παραπάνω πραγματοποιήθηκε επιτυχώς και πλέον έχουμε έναν μικρό επεξεργαστή που μπορεί με απόλυτη επιτυχία να εκτελέσει τις εντολές του CHARIS4, όπως αυτές παρουσιάστηκαν στο 2ο εργαστήριο.

Παρακάτω φαίνεται η **επιτυχής εκτέλεση ολόκληρου του δοκιμαστικού προγράμματος** που δόθηκε στο εργαστήριο, σε μία από τις καλύτερες φωτογραφίες που έχουν τραβηχτεί μέσα σε αυτό!
(Τα χρώματα προστέθηκαν για κατηγοριοποίηση των σημαντικών σημάτων ανά εντολή)



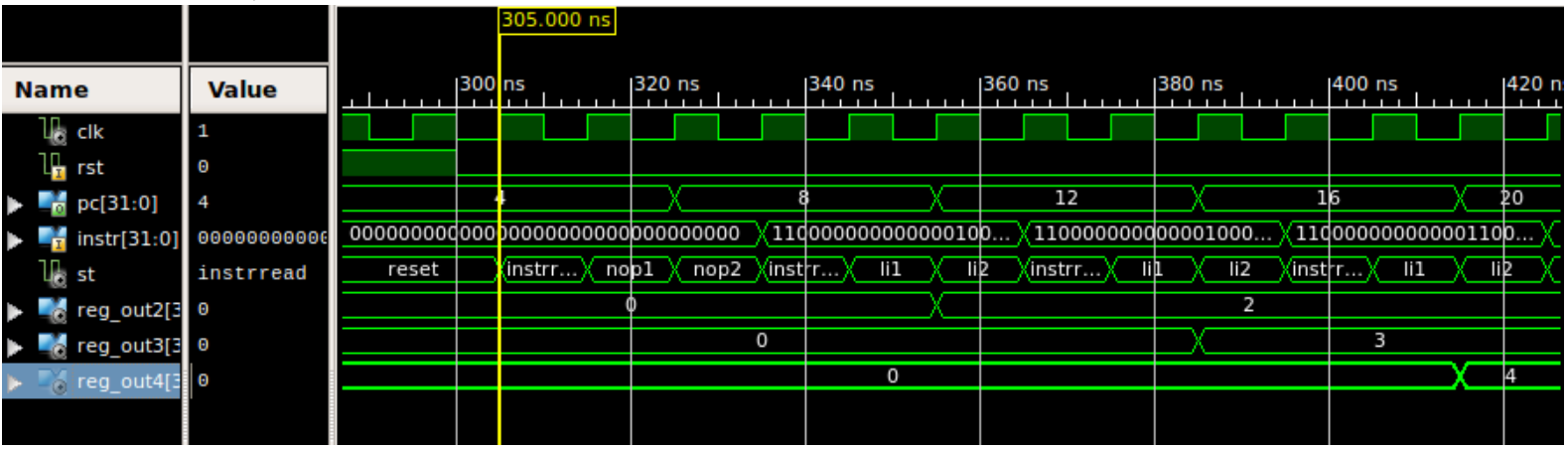
Χριστοδούλου Θεόφιλος **A.M.: 2011030010**

Καριμπίδης Διονύσης **A.M.: 2011030009**

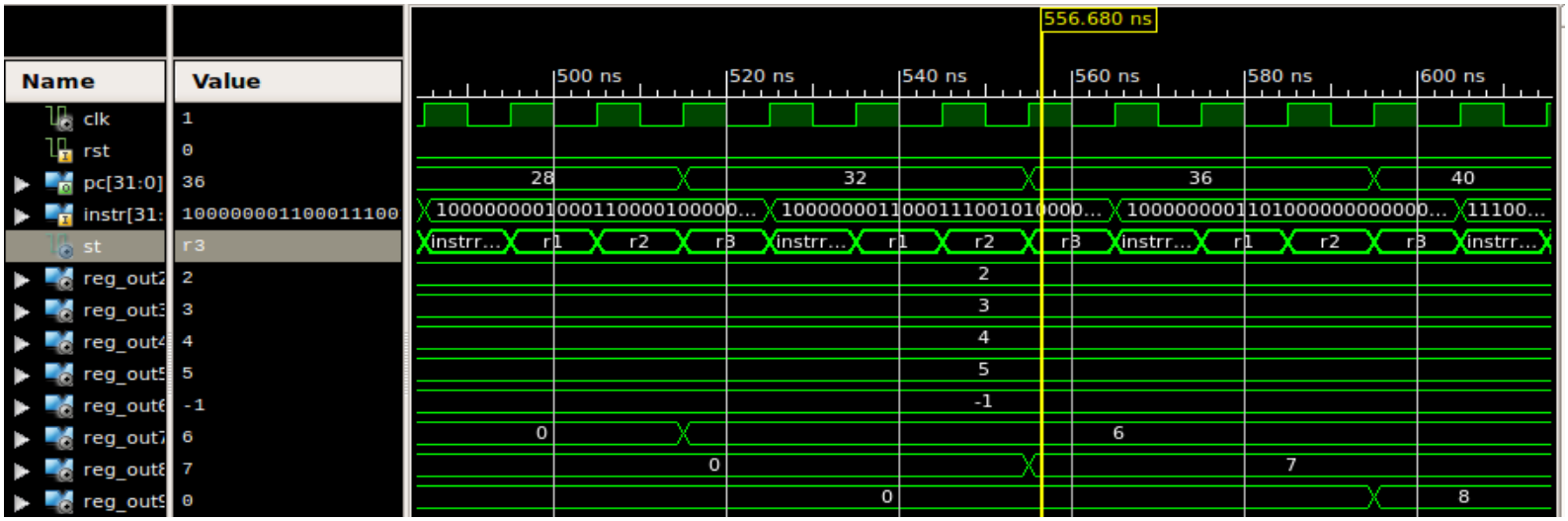
Μάρτιος 2014

Και πιο συγκεκριμένα οι επιμέρους εντολές..

- ✔ Οι 3 πρώτες li, μετά το reset..
  li $1, 2-- $1 = 2
  li $2, 3-- $2 = 3
  li $3, 4-- $3 = 4

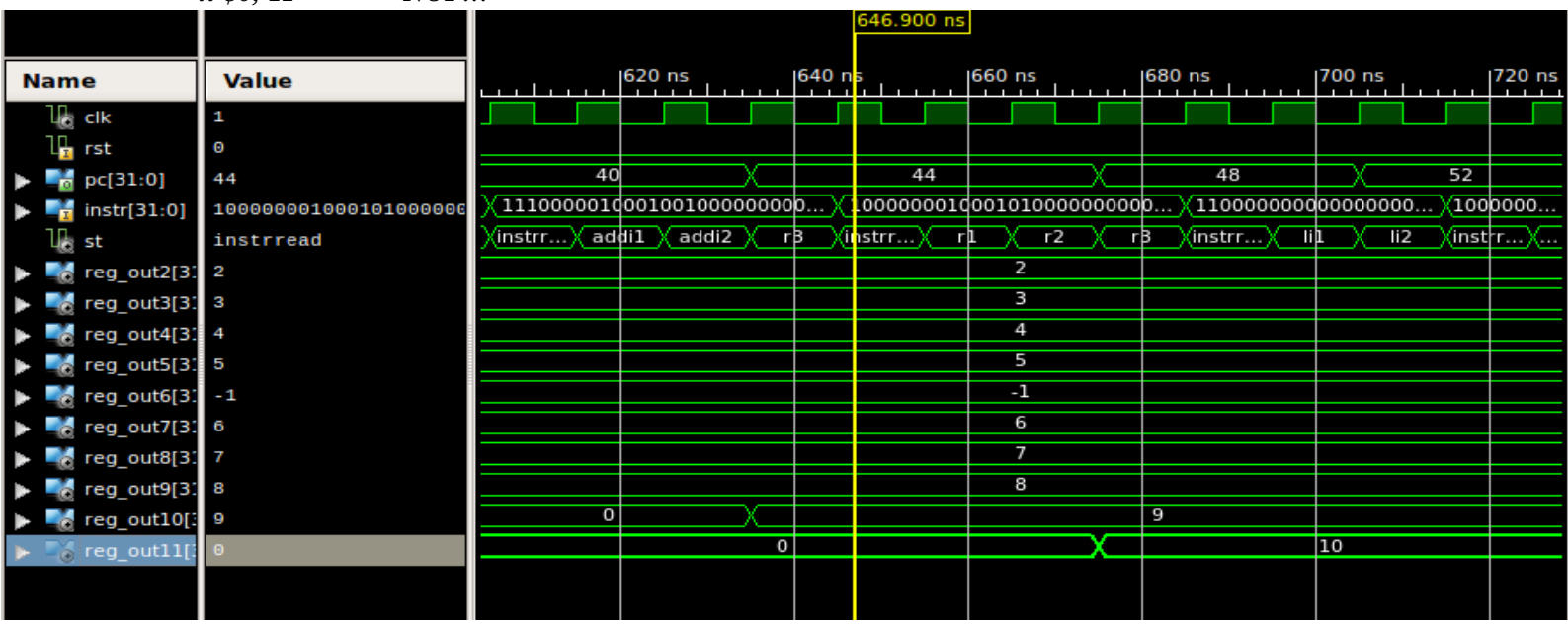


- ✔ 3 R-type εντολές..
  add $6, $2, $2 -- $6 = 6
  sub $7, $6, $5 -- $7 = 7
  shl $8, $3 -- $8 = 8



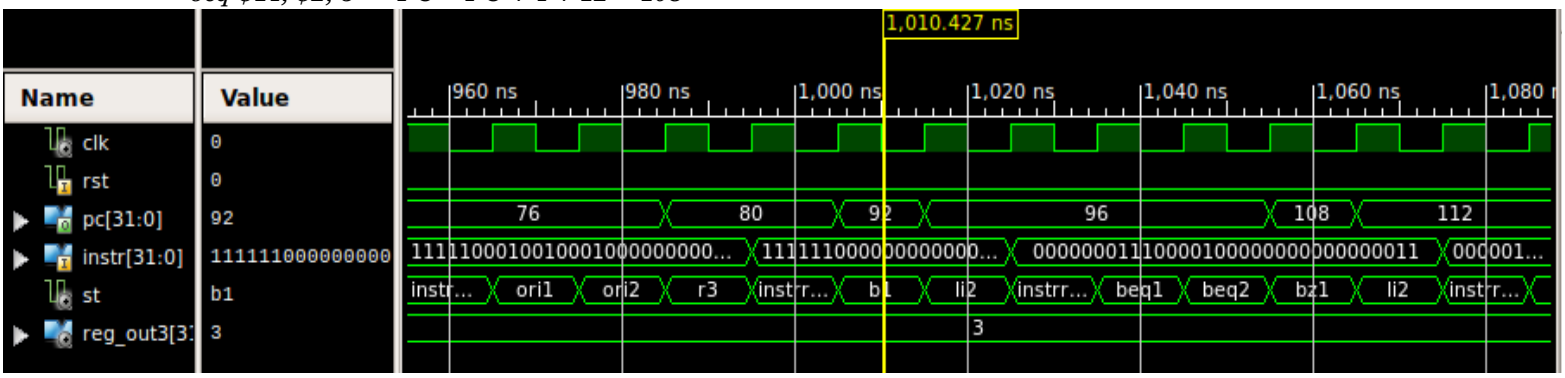
✔ Ακολουθούν...

addi $9, $4, 4 -- $9 = 9
rol $10, $4 -- $10 = 10
li $0, 11 -- NOP!!!



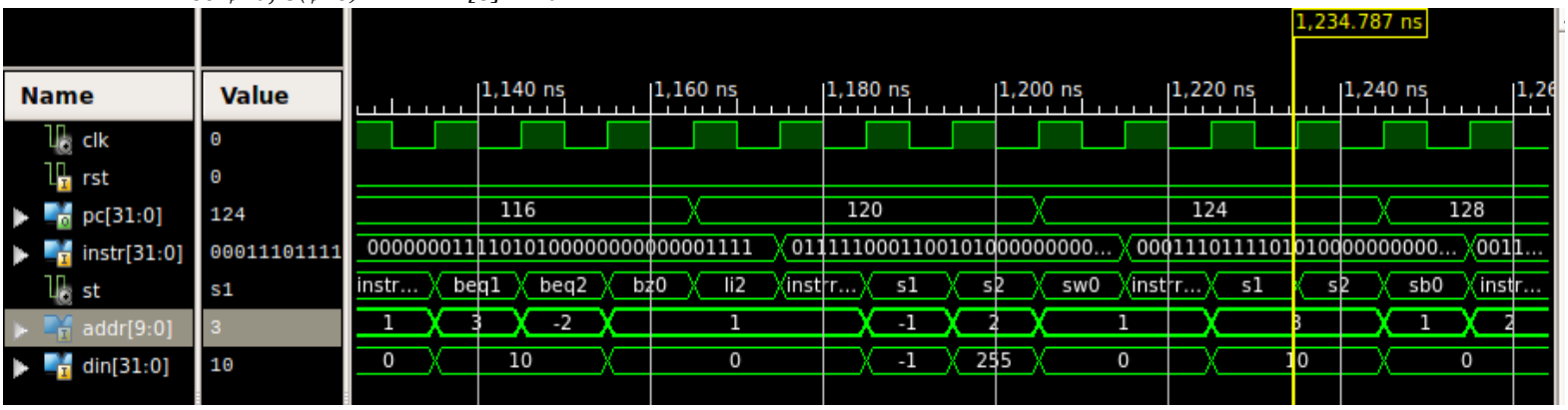
✔ Μία ori, μία branch και μία beq μετ' εμποδίων..

ori $17, $4, 2 -- $17 = 7, PC = 72
b 3 -- PC = PC + 4 + 12 = 92
nop
nop
nop
beq $14, $2, 3 -- PC = PC + 4 + 12 = 108



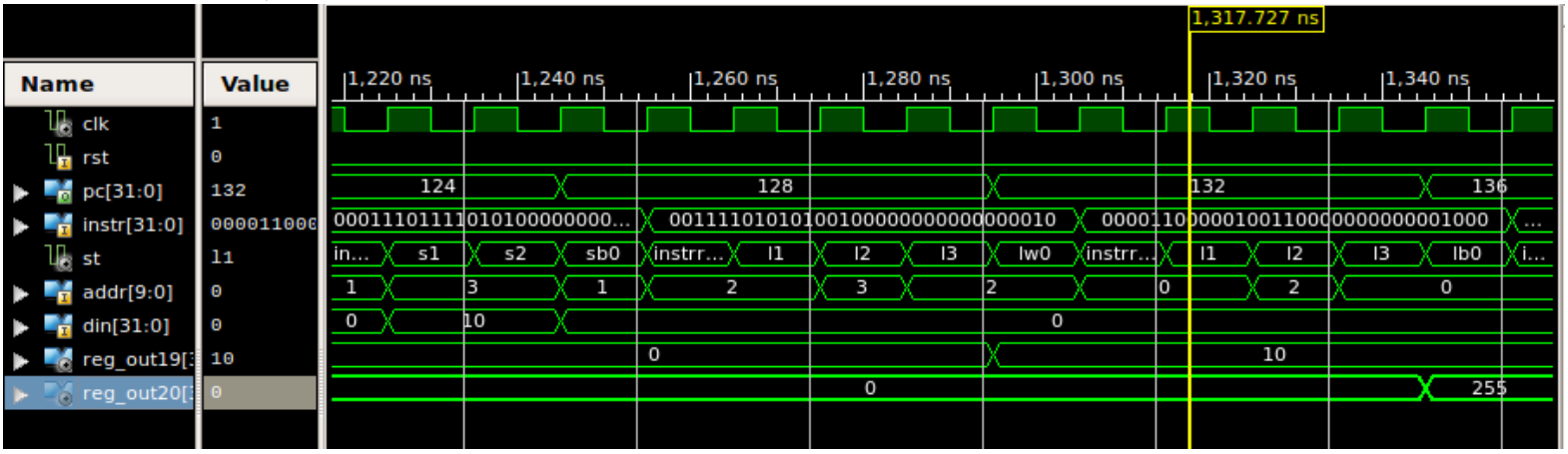
✔ Ακόμα μία beq, ακολουθούμενη από 2 store..

beq $15, $10, 16
sw $5, 4($3) -- MEM[2] = -1
sb $10, 8($15) -- MEM[3] = 10



✔ Τέλος, μία ακόμη store και 2 load..

sb $10, 8($15) -- MEM[3] = 10
lw $18, 2($10) -- $18 = MEM[3] = 10
lb $19, 8($0) -- $19 = MEM[2] = 255



Εξαιρετικό, ε?