18 Απριλίου 2021

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Ο Ρ Γ Α Ν Ω Σ Η Υ Π Ο Λ Ο Γ Ι Σ Τ Ω Ν

(Η Ρ Υ 302)

Ε Ρ Γ Α Σ Ι Α #1

ΚΑΤΣΙΜΠΑΣ ΠΕΤΡΟΣ

2016030038

Σκοπός της Εργαστηριακής Άσκησης

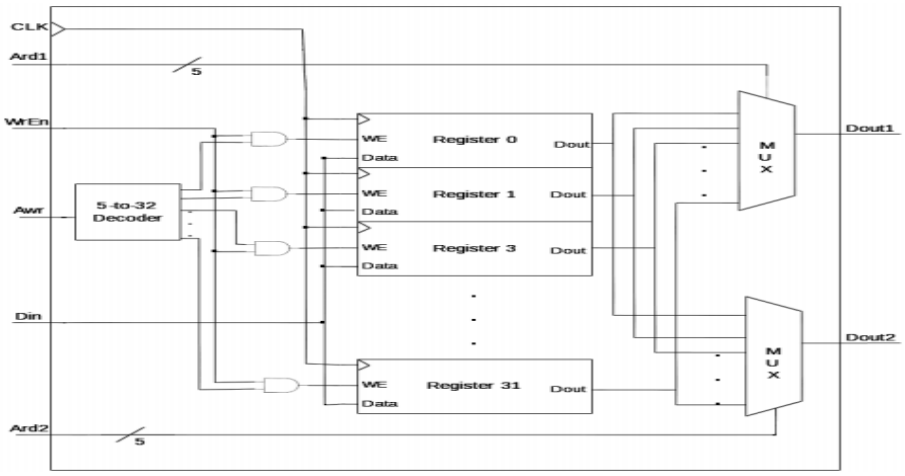
Ο σκοπός της εργαστηριακής άσκησης ήταν η εξοικείωση με την γλώσσα VHDL και το εργαλείο Xilinx ISE καθώς και η κατανόηση της λειτουργίας των συνδέσεων κομματιών(components) στην υλοποίηση ενός επεξεργαστή μονού κύκλου. Η εργαστηριακή άσκηση διαχωρίστηκε σε 3 φάσεις όπως φαίνεται παρακάτω:

1η Φάση εργασίας (ALU, Register File)

Η 1η φάση εμπεριείχε τη κατασκευή μίας μονάδας αριθμητικών και λογικών πράξεων(Arithmetic Logic Unit) καθώς και ένα αρχείο καταχωρητών (Register File) με 32 καταχωρητές.

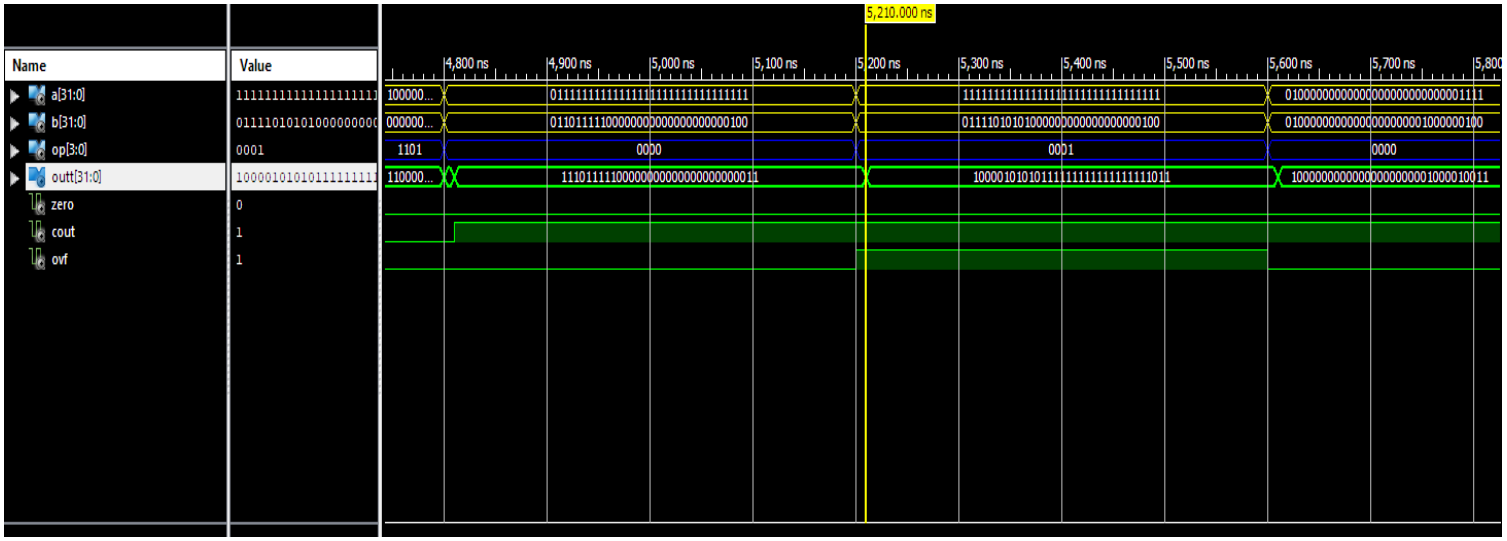
Η σχεδίαση της *ALU* αλλά και η συμπεριφορά της (πράξεις) έγινε ακολουθώντας τις οδηγίες της εκφώνησης. Όλες οι πράξεις υλοποιήθηκαν με χρήση των standard βιβλιοθηκών της VHDL αλλά και της βιβλιοθήκης ”numeric” που εμπεριέχει όλους τους αριθμητικούς και λογικούς τελεστές για την πραγματοποίηση των πράξεων μεταξύ των δεδομένων (είσοδοι – έξοδοι). Σημαντικό να ειπωθεί είναι η υλοποίηση shift (logic or arithmetic) κατά μία θέση δεξιά ή αριστερά, καθώς και rotate κατά μία θέση δεξιά ή αριστερά, για τις παρελκόμενες πράξεις. Ακόμη, αξίζει να αναφερθεί η λογική των 3 σημάτων(flags), δηλαδή για το Zero, το οποίο γίνεται ΄1΄ εάν το αποτέλεσμα μιας πράξης είναι μηδέν, για το Ovf, το οποίο ελέγχει το αποτέλεσμα μίας πράξης και στην περίπτωση που δύο θετικοί αριθμοί ισούνται με αρνητικό αποτέλεσμα ή δύο αρνητικοί αριθμοί ισούνται με θετικό αντίστοιχα(λόγω 2’s complement) ενεργοποιείται το σήμα και τέλος, για το Cout, το οποίο γίνεται ‘1’ εφόσον μία πρόσθεση εμπεριέχει μεγάλους αριθμούς και ειδικότερα, αν και των δύο τελεστών το MSB είναι ‘1’.

Για το Register File, αναγκαίο 1ο βήμα ήταν η σχεδίαση ενός καταχωρητή 32bit, ένα component από το οποία θα λαμβάναμε 32 instances, αλλά και η σχεδίαση πολυπλέκτη και αποκωδικοποιητή (5 to 32). Στη δημιουργία αυτή χρησιμοποιήσαμε το for-generate της VHDL για τους καταχωρητές, εξαιρώντας τον R0 ο οποίος έπρεπε να έχει σταθερή τιμή ‘0’. Έτσι, υλοποιήσαμε το Register File σύμφωνα με το διάγραμμα της σελίδας 4 της εκφώνησης.



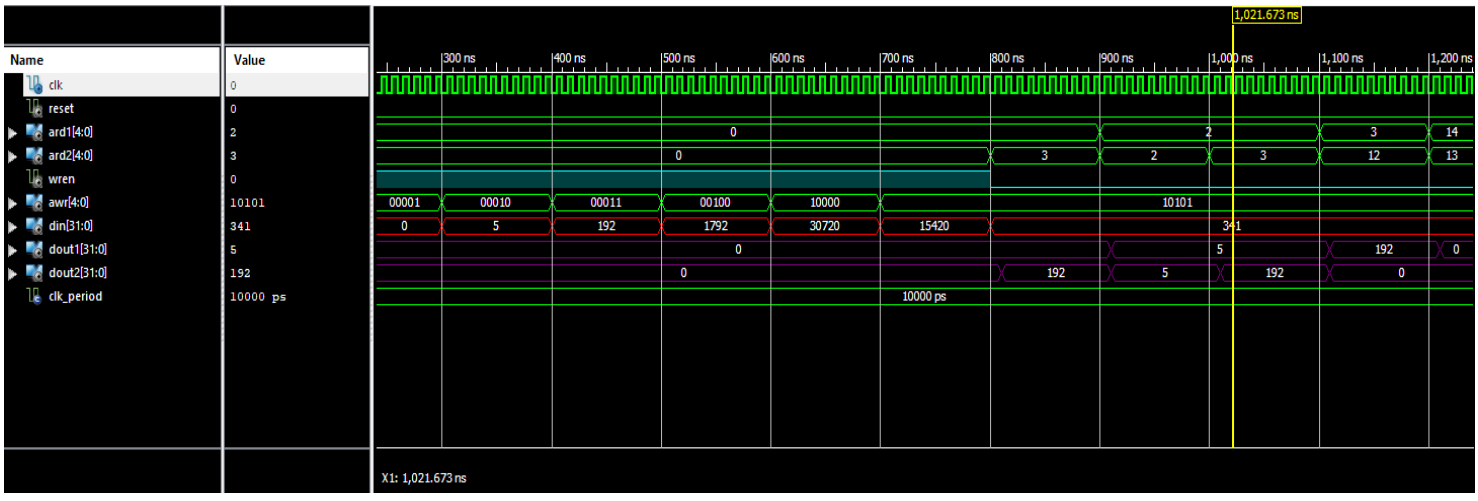
Waveforms

ALU



Παρατίθεται ενδεικτικά η κυματομορφή Εισόδου-Εξόδου της ALU. Μας ζητήθηκε η καθυστέρηση 10 ns στην έξοδο(Out), η οποία γίνεται αντιληπτή τη χρονική στιγμή 5,200ns, αλλά και η λειτουργικότητα των σημάτων Carry-out, Overflow. Παρατηρείται, επίσης, με κίτρινο χρώμα τα σήματα εισόδου(A , B) και το σήμα επιλογής πράξης 'Op-code' . (Για όλες τις πράξεις ανατρέξτε στο testbench του Module)

REGISTER_FILE



Παρατίθεται ενδεικτικά η κυματομορφή Εισόδου-Εξόδου του Αρχείου Καταχωρητών. Παρατηρούμε πως αρχικά η ενεργοποίηση εγγραφής είναι ενεργή (*wren = '1'*) επομένως τα δεδομένα (***din***) εγγράφονται στον καταχωρητή που δείχνει το **awr** και διαβάζονται από τους καταχωρητές που δείχνουν τα ***ard1, ard2***. Έτσι μετά τα 800ns όταν ζητάμε να πάρουμε τα δεδομένα του καταχωρητή «3» λαμβάνουμε το 192 που είχε γραφθεί και όχι την τιμή που ήταν στο din τη δεδομένη χρονική στιγμή, με το *wren = '0'*.

$2^{η}$ Φάση εργασίας

Στην 2η φάσης της εργαστηριακής άσκησης υλοποιήσαμε τμήματα ενός non-pipelined επεξεργαστή βασισμένου σε υποσύνολο της αρχιτεκτονικής συνόλου εντολών CHARIS (CHAnia Risc Instruction Set), που αποτελείται από τις εξής βαθμίδες : ανάκλησης εντολών (IF), αποκωδικοποίησης εντολών (DEC), εκτέλεσης εντολών (EX), πρόσβασης μνήμης (MEM). Οι βαθμίδες αυτές έχουν μία κοινή μνήμη (RAM) την οποία κληθήκαμε να ενσωματώσουμε στη σχεδίασή μας με σκοπό την εξοικείωσή μας με τις block μνήμες και τη διαχείρισή τους μέσα στη σχεδίαση.

Μελετήστε την κωδικοποίηση των εντολών του CHARIS

Με την μελέτη αποφανθήκαμε ότι οι μαθηματικές και οι λογικές πράξεις καθώς και κάποιες εντολές ολίσθησης, είναι R τύπου και διακρίνονται από το function που έχουν. Τις υπόλοιπες εντολές που είναι τύπου I και J τις διακρίνουμε με τον αν χρειάζεται να κάνουμε SignExtend,το Immed ή ZeroFill,το Immed στους αντίστοιχους καταχωρητές.

Υλοποίηση κύριας μνήμης 2048x32

Για την υλοποίηση της κύριας μνήμης χρησιμοποιήθηκε ο κώδικας που παραχωρήθηκε με την εκφώνηση της εργαστηριακής άσκησης ενώ για την αρχικοποίησή τη χρησιμοποιήθηκε το αρχείο rom.data που παραχωρήθηκε.

Βαθμίδα ανάκλησης εντολών (IF)

Για την υλοποίηση της βαθμίδας ανάκλησης εντολών χρησιμοποιήθηκε ένας καταχωρητής 32 bits όπως εκείνοι στο προηγούμενη φάση, ένας αθροιστής/αυξητής που υπολογίζει την τιμή PC+4 (Program Counter+4) αλλά και ένας αθροιστής/αυξητής που υπολογίζει την τιμή PC+4+Immediate όπου immediate ακέραιος αριθμός για την εξυπηρέτηση των εντολών διακλάδωσης. Τέλος, χρησιμοποιήθηκε ένας πολυπλέκτης 2-σε1 για την επιλογή του επιθυμητού αυξητή. Τα modules αυτά συνέθεσαν σε structural VHDL τη βαθμίδα ανάκλησης εντολών (Instruction Fetch).

Βαθμίδα αποκωδικοποίησης εντολών (DECODE)

Για την υλοποίηση της βαθμίδας αποκωδικοποίησης των εντολών χρησιμοποιήθηκε το Αρχείο Καταχωρητών από τη 1η φάση ,το οποίο περιέχει τους 32 καταχωρητές του επεξεργαστή, ένας πολυπλέκτης 2-σε-1 για την επιλογή της προέλευσης (Κύρια Μνήμη ή Μονάδα Αριθμητικών & Λογικών Πράξεων) των δεδομένων προς εγγραφή στο Αρχείο Καταχωρητών και τέλος ένα module “ImmModule” που εμπεριέχει τη λογική για την κατάλληλη μετατροπή του immediate που μπορεί να εμπεριέχεται σε μια εντολή του επεξεργαστή, από 16 σε 32 bits, με τρόπο που θα εξυπηρετεί την εκάστοτε λειτουργία/εντολή. Για παράδειγμα, στην εντολή sll ο immediate πρέπει στον καταχωρητή RS να δεχτεί zero fill, σύμφωνα με το ISA, πράγμα που το «αποφασίζει» το ImmModule με την κατάλληλη λογική. Τα modules αυτά συνέθεσαν σε structural VHDL τη βαθμίδα αποκωδικοποίησης των εντολών (Instruction Decoding).

Βαθμίδα εκτέλεσης εντολών (EX)

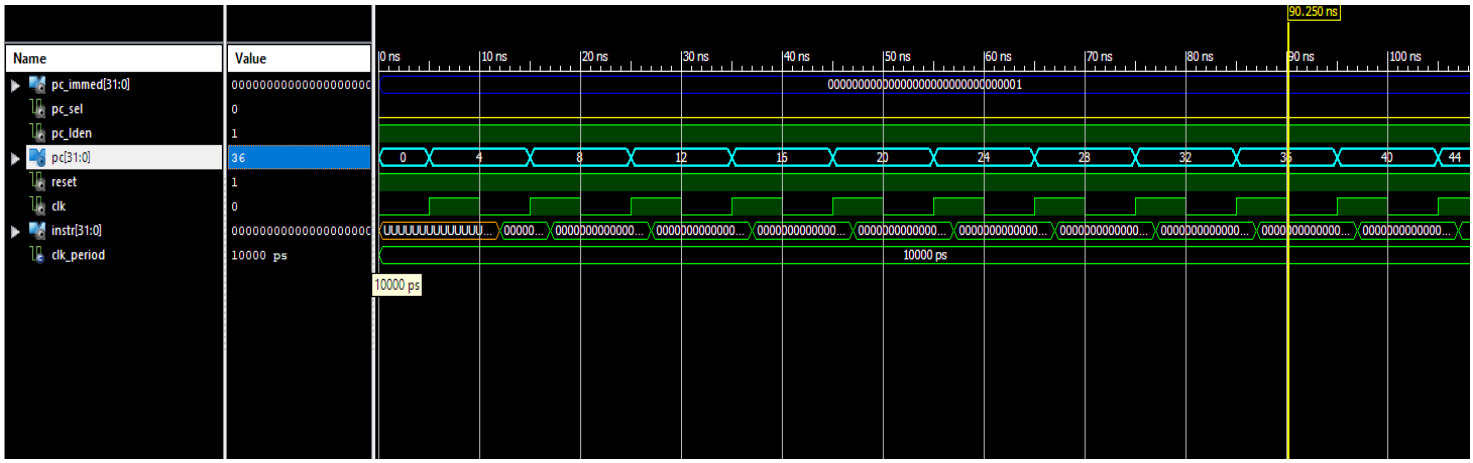
Για την υλοποίηση της βαθμίδας εκτέλεσης εντολών χρησιμοποιήθηκε η Μονάδα Αριθμητικών και Λογικών Πράξεων (ALU) από τη 1η φάση και ένας πολυπλέκτης 2-σε-1 για την επιλογή του δεύτερου τελεστέου της ALU μεταξύ κάποιου καταχωρητή ή κάποιας immediate τιμής. Τα modules αυτά συνέθεσαν σε structural VHDL τη βαθμίδα εκτέλεσης των εντολών (Instruction Execution) καθώς δε χρειάστηκε περαιτέρω λογική (πέραν κάποιον σημάτων-εισόδων για select στον πολυπλέκτη και για καθορισμό της λειτουργίας της ALU) εφόσον η περισσότερη λογική εμπεριέχεται μέσα στο ALU Module για την εκτέλεση των αριθμητικών και λογικών εντολών.

Βαθμίδα πρόσβασης μνήμης (MEM)

Για την υλοποίηση της βαθμίδας Πρόσβασης Μνήμης δε χρειάστηκε κάποιο module παρά η ίδια η μνήμη του βήματος Β και λίγη λογική ώστε να προσθέσουμε offset 0x400 στην είσοδο της βαθμίδας που αφορά την διεύθυνση της μνήμης ώστε τα δεδομένα να γράφονται στο σωστό section της μνήμης εφόσον έχουμε κοινή μνήμη για τον κώδικα (text) και τα δεδομένα (data). Η βαθμίδα αυτή ονομάζεται Memory Access Module – MEM).

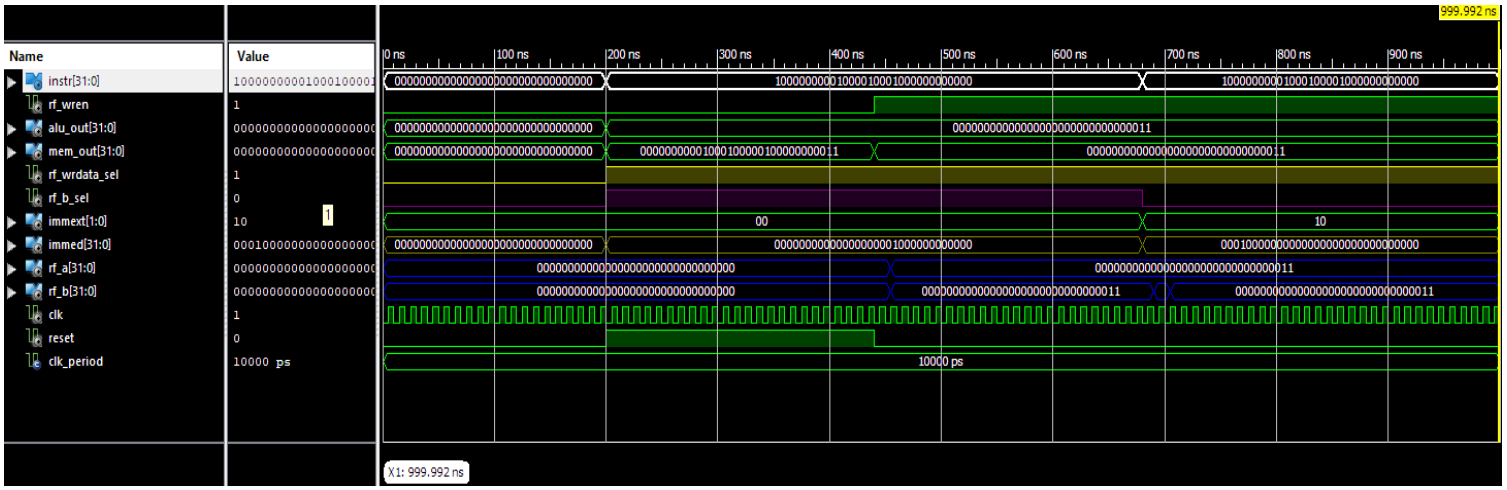
Waveforms

IFSTAGE



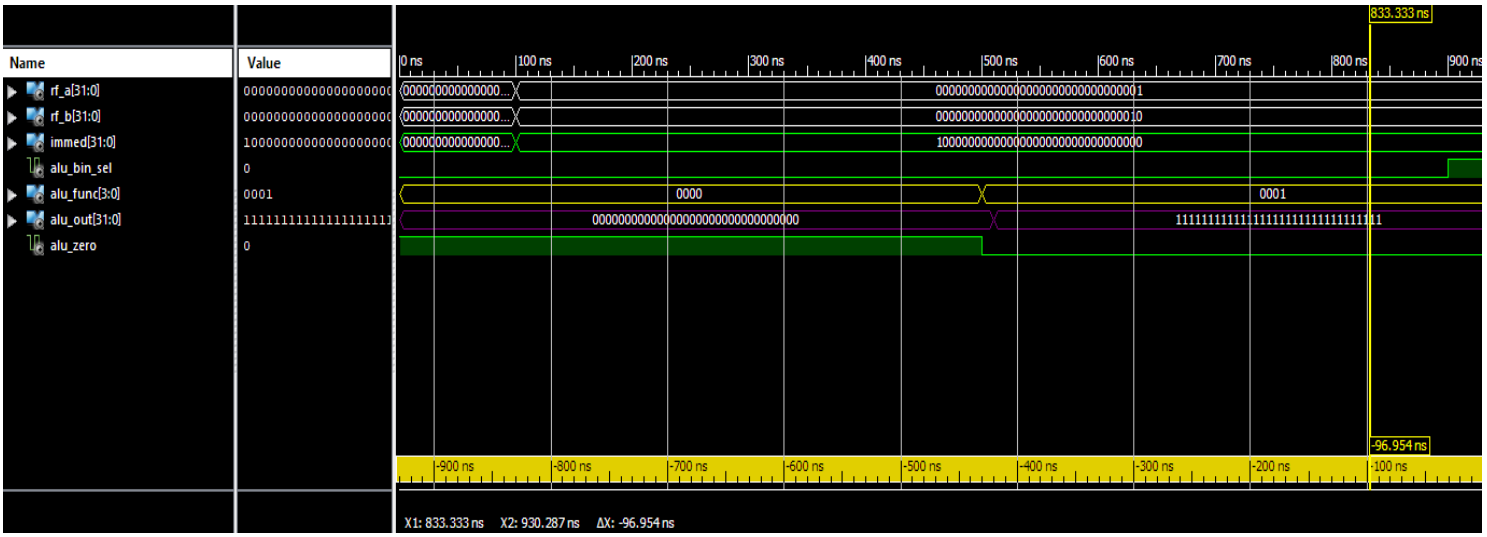
Με μπλε χρώμα αναπαριστάται η είσοδος της βαθμίδας (***immediate***). Ύστερα με κίτρινο η είσοδος ελέγχου (***pc_sel***) του 32bit-ου, 2 σε 1 πολυπλέκτη που καθορίζει την τιμή που θα εισχωρηθεί στον καταχωρητή PC. Με πράσινο επίσης φαίνεται το σήμα του ελέγχου εγγραφής του καταχωρητή PC (***pc_lden***) το οποίο ενεργοποιεί η απενεργοποιεί την δυνατότητα εγγραφής στον ίδιο. Τέλος με κυανό χρώμα αναπαριστάται η έξοδος της βαθμίδας IFSTAGE δύνοντας είτε την τιμή PC + 4 ή την τιμή PC + 4 + Immediate.

DECSTAGE



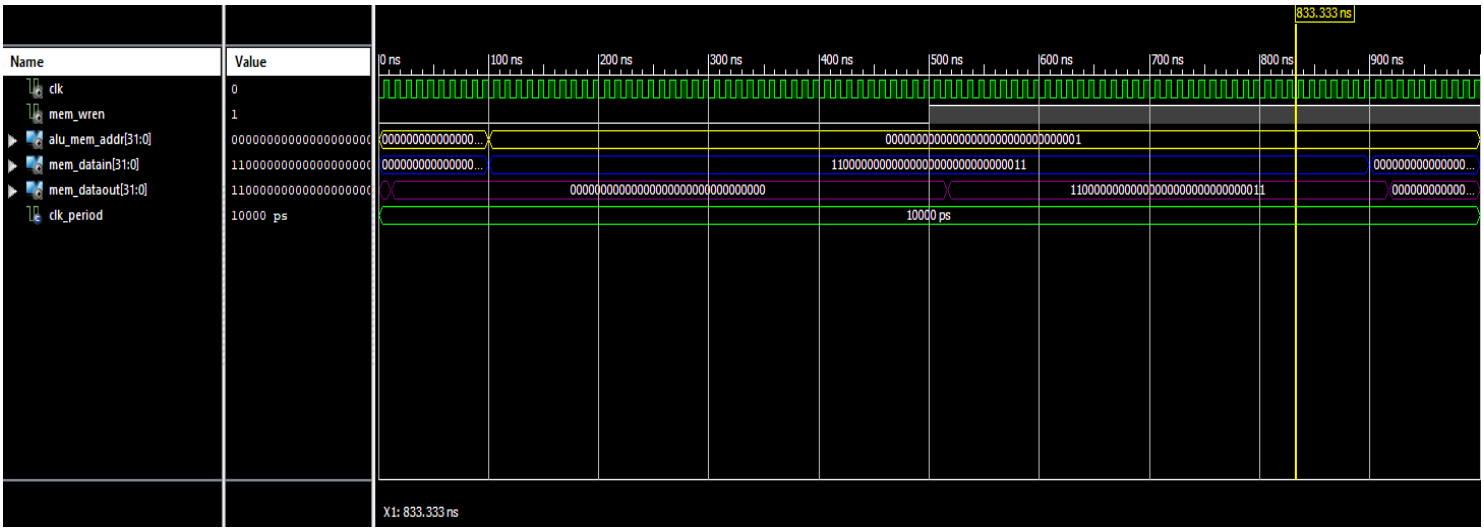
Με άσπρο χρώμα αναπαριστάται το instruction προς αποκωδικοποίηση. Το ***rf_b_sel*** ορίζει ποιος θα είναι ο δεύτερος καταχωρητής προς ανάγνωση επιλέγοντας ανάμεσα στους rd και rt. Το ***rf_wrdata_sel*** ορίζει αν θα γραφτούν δεδομένα στην RF από το αποτέλεσμα των πράξεων της ALU ή από την μνήμη. ***RF_wren*** ύστερα είναι το σήμα ενεργοποίησης δυνατότητας εγγραφής στους καταχωρητές του register file. Με μπλε χρώμα επίσης προσομοιώνεται η έξοδος των δύο καταχωρητών προς το EXSTAGE.

EXSTAGE



Με άσπρο χρώμα έχουμε τις δύο εισόδους των καταχωρητών ***A*** και ***B***. Αναλόγως το κίτρινο σήμα ***alu_func*** εκτελείται η ανάλογη πράξη από το ISA, ενώ με μωβ έχουμε την έξοδο-αποτέλεσμα της πράξης.

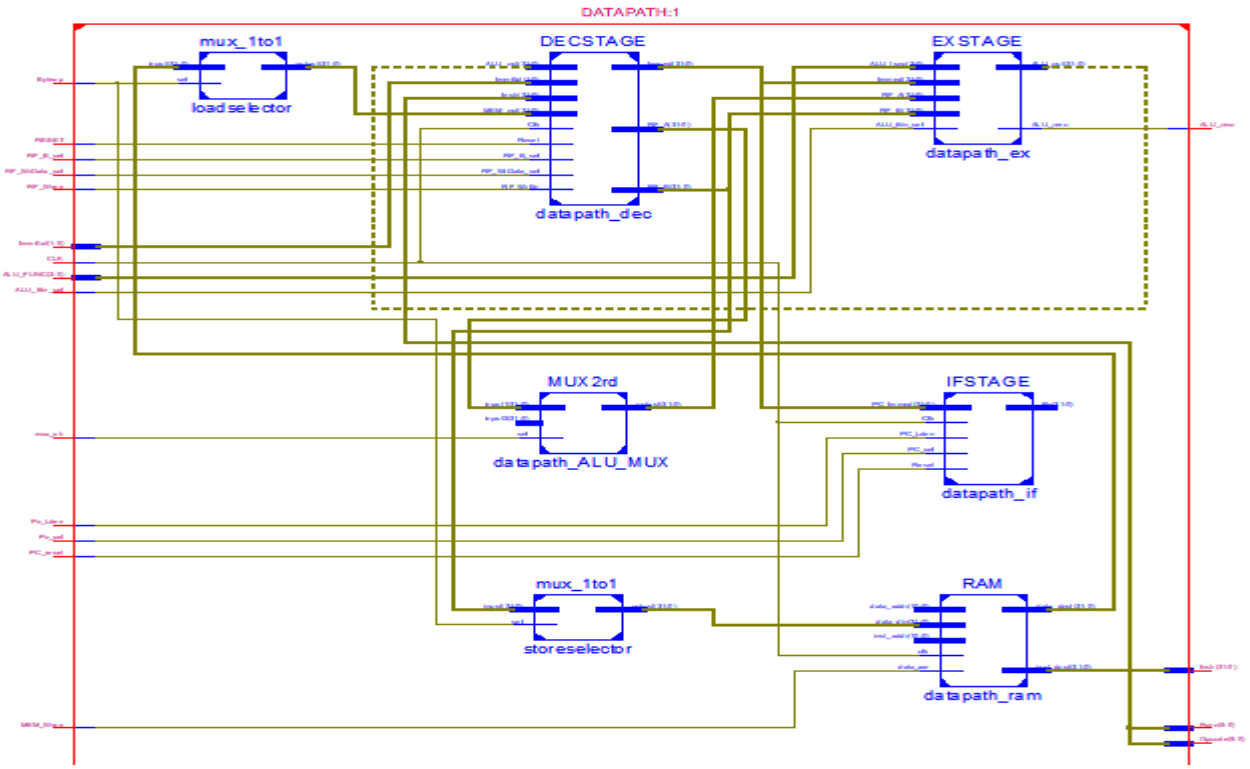
MEMSTAGE



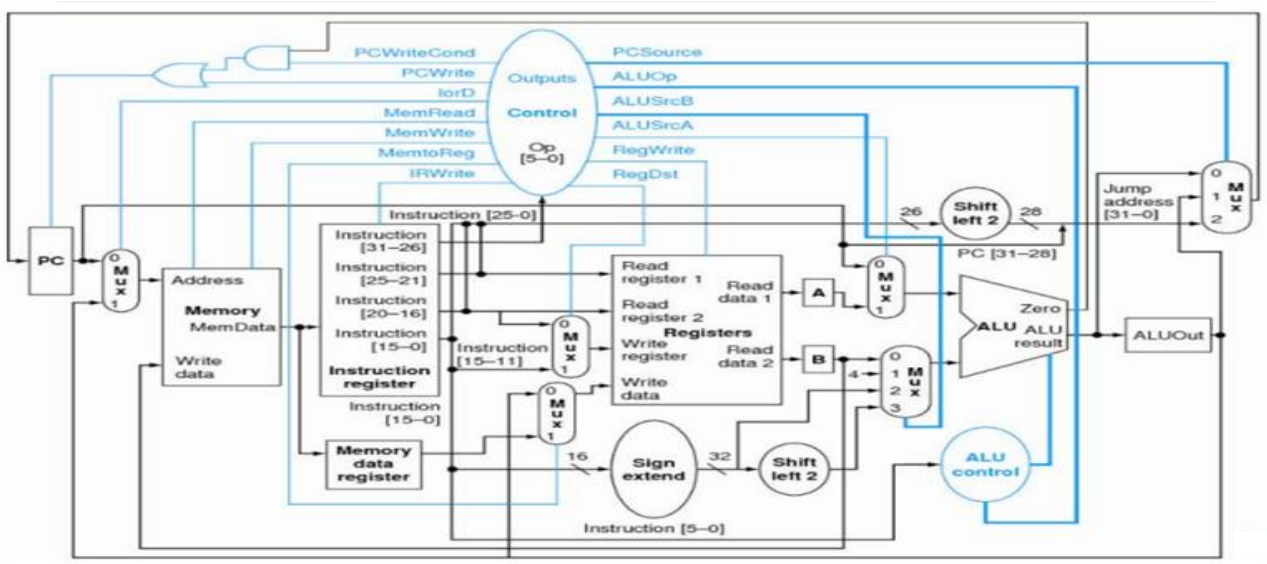
Με κίτρινο χρώμα βλέπουμε την είσοδο διευθύνσεων της μνήμης η οποία θυμίζουμε πως έχει αυξηθεί η τιμή της κατά 1024. Με μπλε χρώμα φαίνεται η είσοδος δεδομένων της μνήμης. Το σήμα με το άσπρο χρώμα είναι υπεύθυνο για το κλείδωμα και ξεκλείδωμα της δυνατότητας εγγραφής στη μνήμη. Τέλος με μωβ φαίνεται η έξοδος των δεδομένων της μνήμης.

3η Φάση εργασίας

Σε αυτή την φάση της εργαστηριακής άσκησης ενώσαμε τις δομικές βαθμίδες του επεξεργαστή CHARIS (Instruction Fetch, RAM Memory, Instruction Decode, Execution/ALU) ώστε να δημιουργούν ένα ολοκληρωμένο single cycle data path. Εκτός των συνδέσεων των «καλωδίων», προστέθηκε επιπλέον λογική όπου χρειάστηκε, όπως πολυπλέκτες και άλλες πιο σύνθετες βαθμίδες ώστε να πετύχουμε τη λογική των lb/lw και sb/sw εντολών. Η υλοποίηση φαίνεται στο παρακάτω σχήμα:



Για την ολοκλήρωση του επεξεργαστή υλοποιήσαμε και ένα CONTROL, ώστε να παράγουμε τα σωστά σήματα ελέγχου για κάθε instruction/cycle. Με τη βοήθεια μιας FSM εξετάζουμε, δηλαδή, το opcode – func της εντολής και ενεργοποιούμε τα κατάλληλα σήματα. Παρατίθεται η υλοποίηση της στο παρακάτω σχήμα:



Προκειμένου να δοκιμάσουμε την ορθή λειτουργία του επεξεργαστή μονού κύκλου, δόθηκαν κάποια προγράμματα αναφοράς και αρχικοποιημένο αρχείο μνήμης. Οι εντολές δόθηκαν σε binary μορφή στο αρχείο μνήμης και το CONTROL διαλέγοντας τα κατάλληλα σήματα ελέγχου «τρέξαμε» τις εντολές για να ελέγξουμε τη λειτουργία του επεξεργαστή, επεξεργαζόμενοι το δοθέν test bench όπου χρειάστηκε.

Πρόγραμμα αναφοράς #1

00: addi r5, r0, 8

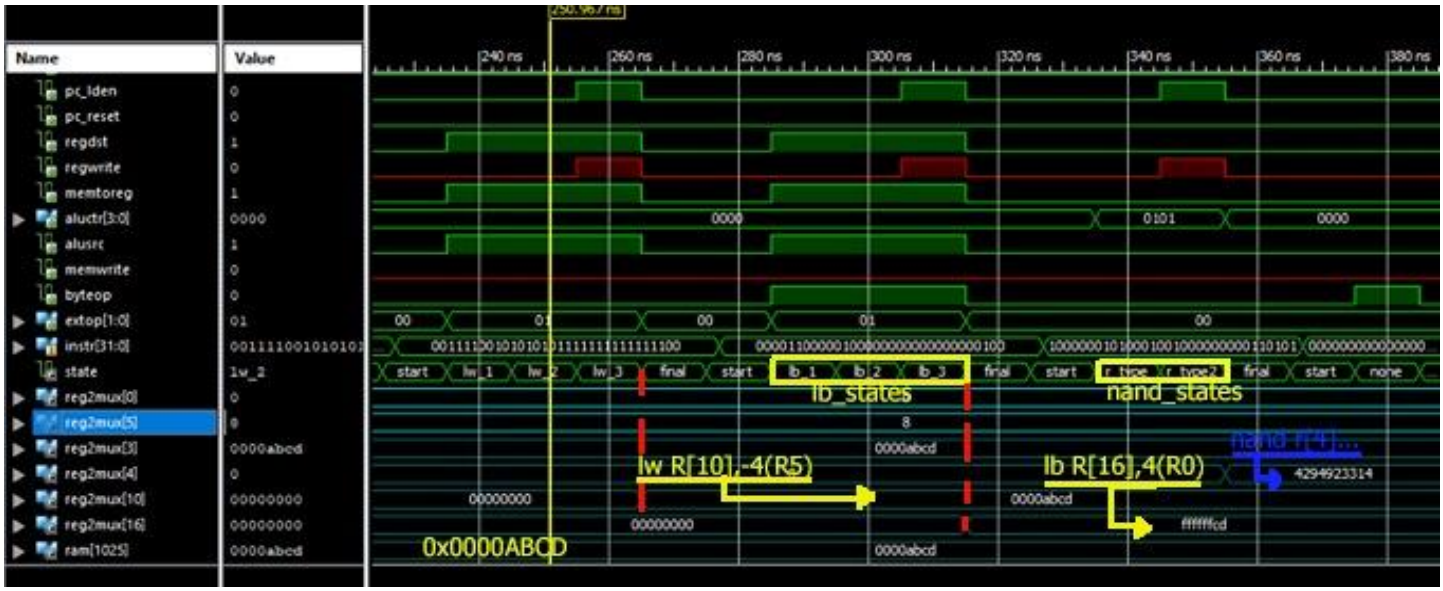
04: ori r3, r0, 0xABCB

08: sw r3, 4(r0) // γράφει στην διεύθυνση 0x4 => 0x404 την τιμή 0x0000ABCD

0C: lw r10, -4(r5) // διαβάζει από την διεύθυνση 0x4 => 0x404 την τιμή 0x0000ABCD

10: lb r16, 4(r0) // διαβάζει byte από την διεύθυνση 0x4 => 0x404 την τιμή 0xFFFFFFCD

14: nand r4, r10, r16

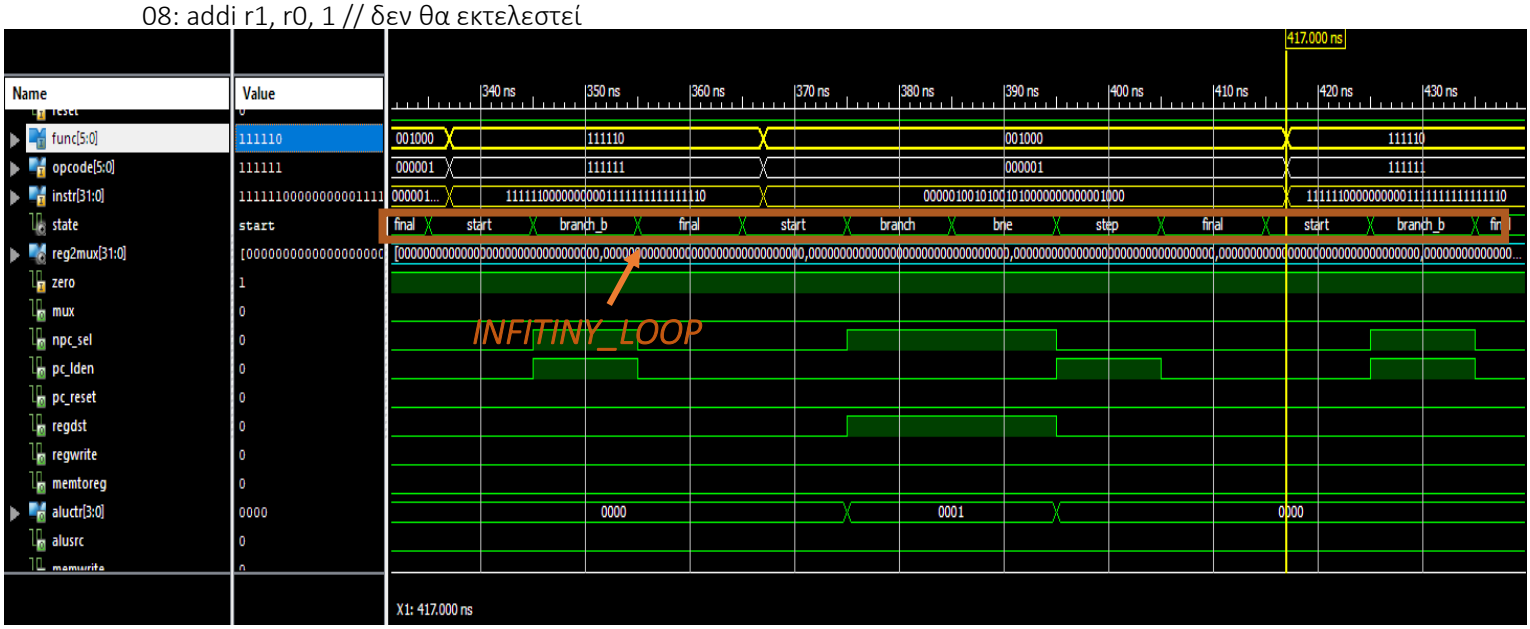


Πρόγραμμα αναφοράς #2

00: bne r5, r5, 8 // αποτυχημένη διακλάδωση

04: b -2 // branch (PC=04 + 4 -2*4 = 00) infinite loop!

08: addi r1, r0, 1 // δεν θα εκτελεστεί



Ευχαριστώ για τον χρόνο σας!