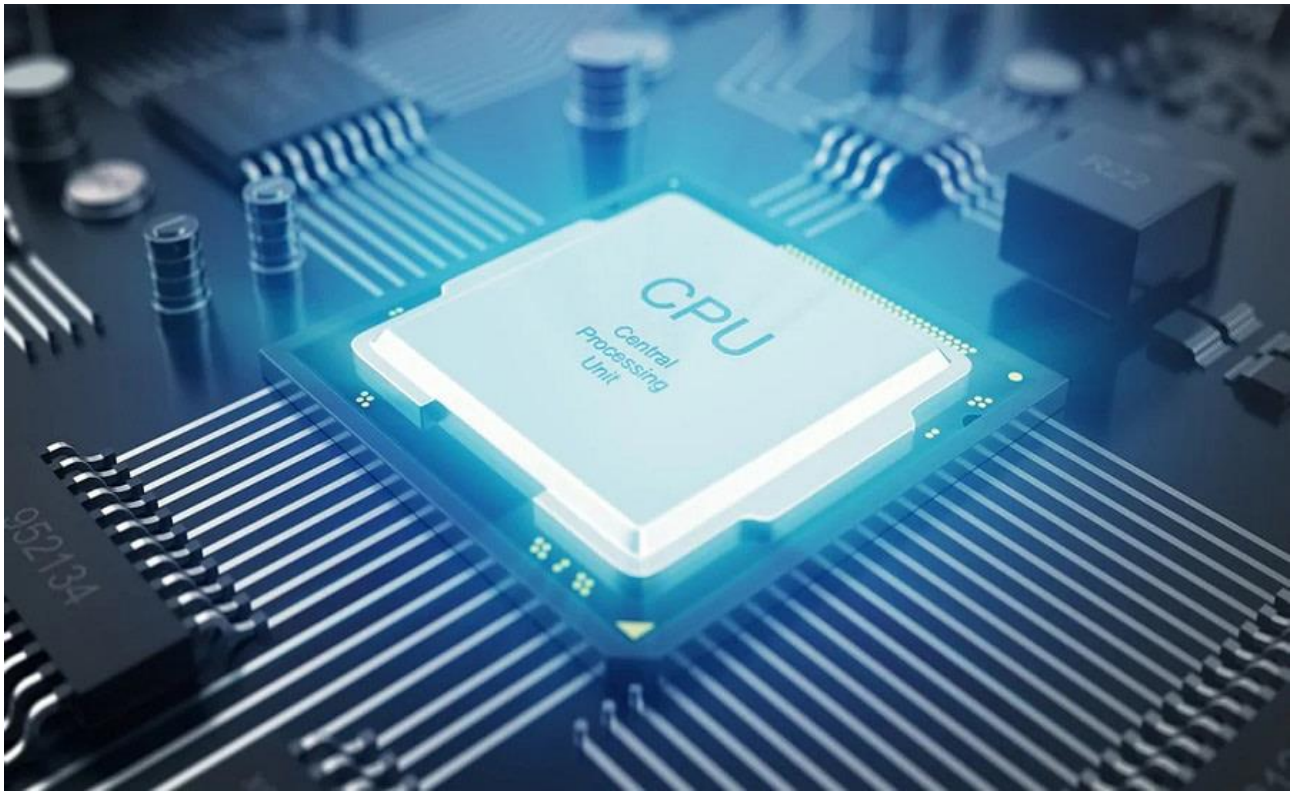
Ψηφιακά Συστήματα HW-1
Εργασία 2021-2022

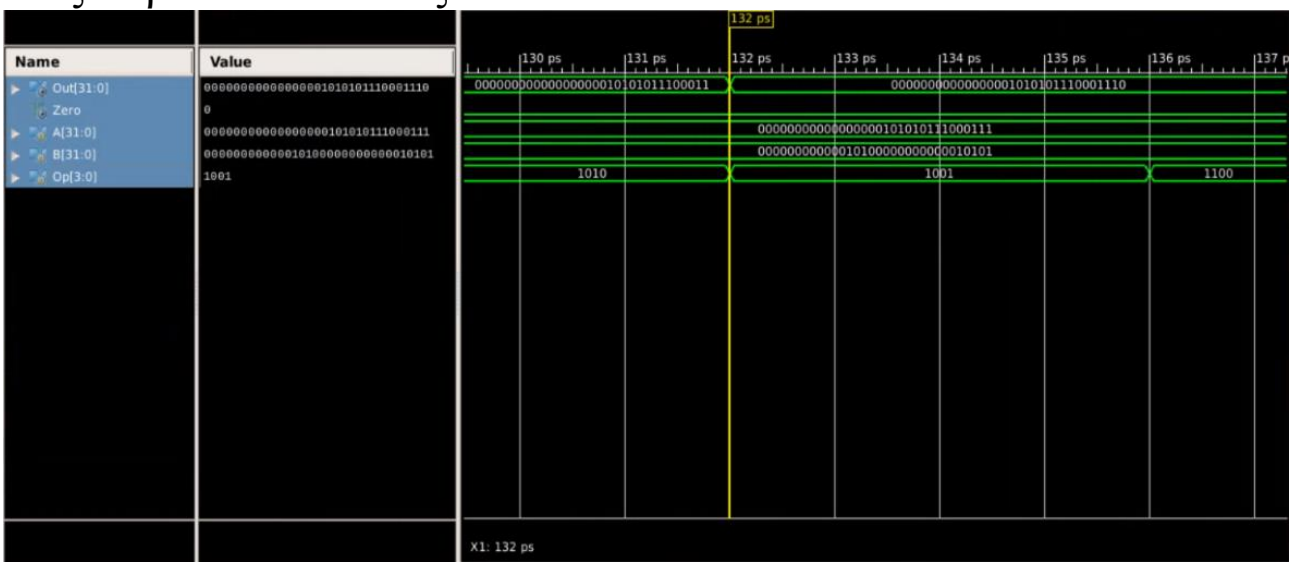


Ονοματεπώνυμο: Καραμητόπουλος Παναγιώτης
ΑΕΜ: 9743
email: karamitopp@ece.auth.gr

19.01.22

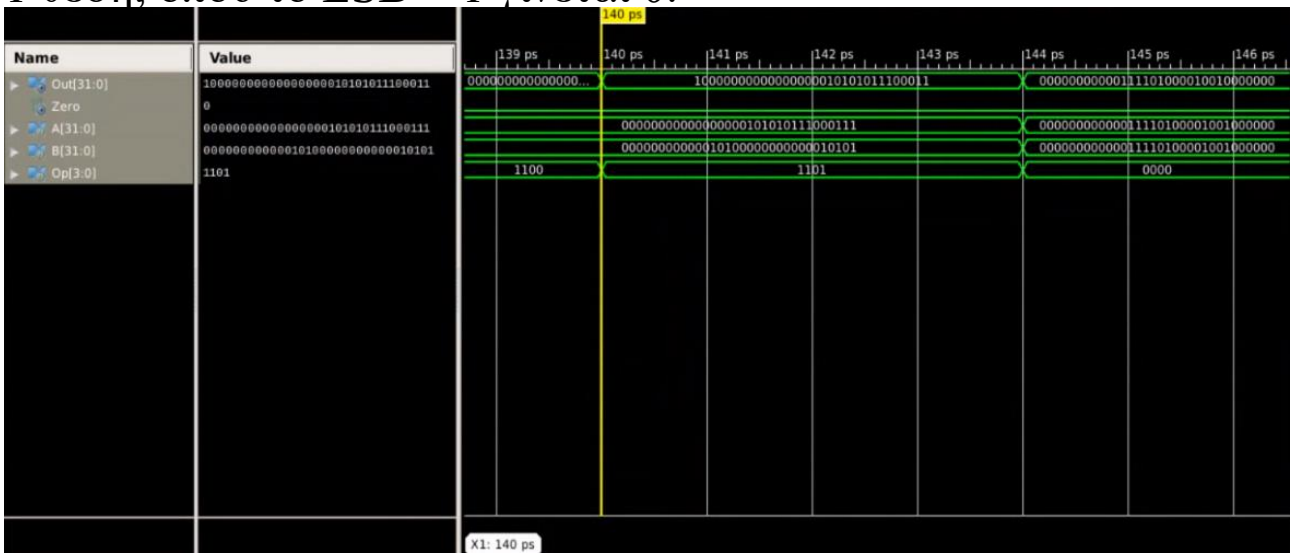
PART 1

A) ALU: Για την υλοποίηση της ALU ορίζουμε της μεταβλητές εισόδου και εξόδου και ανάλογα με το opcode που παίρνουμε ως είσοδο, λέμε στην ALU να κάνει την ανάλογη πράξη. Η συνθήκη που ορίζει την έξοδο Zero της ALU είναι: όταν η έξοδος out της ALU είναι 0 τότε Zero = 1, αλλιώς Zero = 0. Επιλέγουμε 3 περιπτώσεις-πράξεις για να εκτελέσει η ALU, οι οποίες φαίνονται στις παρακάτω εικόνες:



Εικόνα 1: Λογική ολίσθηση αριστερά κατά 1 θέση

Επεξήγηση: Παίρνουμε το διάνυσμα της εισόδου Α και παρατηρούμε ότι γίνεται, όντως, λογική ολίσθηση αριστερά κατά 1 θέση, όπου το LSB = 1 γίνεται 0.



Εικόνα 2: Κυκλική Ολίσθηση του Α δεξιά κατά 1 θέση

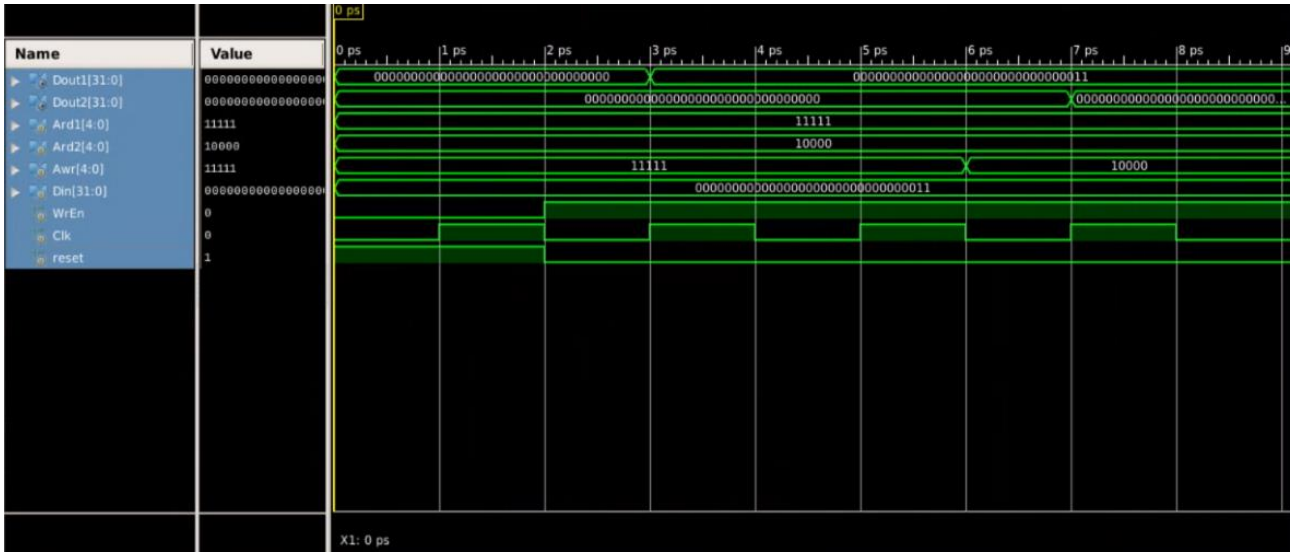
Επεξήγηση: Παίρνουμε το διάνυσμα της εισόδου A και παρατηρούμε ότι το LSB γίνεται MSB και τα υπόλοιπα bit πάνε 1 θέση πίσω.

Εικόνα 3: Λογικό "ΚΑΙ"

Επεξήγηση: Παρατηρούμε ότι το αποτέλεσμα της λογικής πράξης "AND" είναι 0, οπότε η έξοδος Zero = 1, ενώ στις δύο προηγούμενες περιπτώσεις το αποτέλεσμα των πράξεων ήταν διάφορο του μηδενός οπότε Zero = 0.

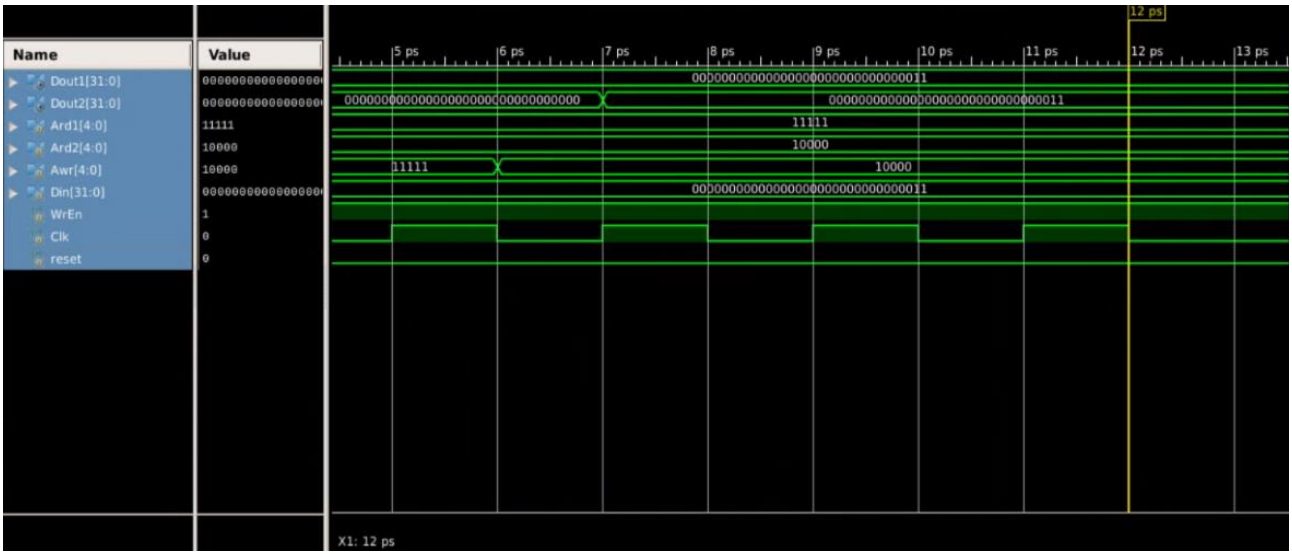
B1) Register: Στον καταχωρητή έχουμε προσθέσει ασύγχρονο σήμα reset ώστε να αρχικοποιείται στο 0.

B2) Register File: Όπως και στο B1) ερώτημα έτσι και εδώ προσθέσαμε ασύγχρονο σήμα reset, ώστε να αρχικοποιούνται όλοι οι καταχωρητές στο 0. Επιπλέον, για την υλοποίηση του Register File, χρησιμοποιήσαμε: 32 καταχωρητές των 32 bit, 2 MUX 32 to 1(32 bit το κάθε input, 32 bit το output), ένα Decoder 5 to 32 και 32 πύλες AND, 2 εισόδων. Για να μην αλλάζει ποτέ η τιμή του R_0 , στην αντίστοιχη πύλη AND θα προσθέσουμε μία έξτρα είσοδο, η οποία θα είναι μόνιμα το λογικό 0. Έτσι, το WE του Register0, θα παραμένει στο 0, ανεξάρτητα των τιμών των δύο άλλων εισόδων της πύλης and. Παραθέτουμε 2 περιπτώσεις λειτουργίας του Register File στις παρακάτω εικόνες:



Εικόνα 4

<u>Επεξήγηση(εικόνα 4):</u> Αρχικά γίνεται reset στο Register File, οπότε αποθηκεύεται 32'b0 στους 32 καταχωρητές και ορθώς παίρνουμε 0 στις εξόδους Dout1 και Dout2. Την χρονική στιγμή 3ps, έρχεται θετικός παλμός στο clock, η είσοδος WrEn είναι 1, το Awr είναι 5'b11111, οπότε εγγράφεται στον καταχωρητή 31 η λέξη: 32'b11. Το Ard1 είναι 5'b11111 οπότε θα διαβάσουμε στην έξοδο Dout1 την λέξη που μόλις γράψαμε στον καταχωρητή 31, ενώ το Ard2 είναι 5'b10000, οπότε θα διαβάσουμε στην έξοδο Dout2 την λέξη που είναι αποθηκευμένη στον καταχωρητή 16, η οποία είναι το 0 (μετα από reset του Register File).



Εικόνα 5

Επεξήγηση(εικόνα 5): Τη χρονική στιγμή 6ps, το Awr είναι 5'b10000, οπότε στον επόμενο θετικό παλμό του clock που είναι τη χρονική στιγμή 7ps, έχουμε WrEn = 1, εγγράφεται στον καταχωρητή 16 η λέξη: 32'b11. Το Ard1 συνεχίζει να είναι 5'b11111, οπότε θα διαβάσουμε στην έξοδο Dout1 την λέξη που έχει αποθηκευτεί στον καταχωρητή 31 και το Ard2 συνεχίζει να είναι 5'b10000, οπότε θα διαβάσουμε στην έξοδο Dout2 την λέξη που μόλις αποθηκεύτηκε στον καταχωρητή 16, η οποία είναι το 32'b11.

PART 2

Α) Σε αυτό το μέρος, εκτός από τις παρατηρήσεις για την απλοποίηση της αποκωδικοποίησης των εντολών, θα αναλυθεί ξεχωριστά και η κάθε «ομάδα» πράξεων, πράγμα το οποίο θα χρειαστεί τόσο στο part2, αλλά κυρίως στο part3, όπου θα σχεδιαστεί η fsm.

Παρατηρούμε ότι οι πράξεις με opcode = Instr[31:26] = 100000 πραγματοποιούνται στην ALU. Τα 4 τελευταία (από δεξιά) bit του func είναι όμοια με το opcode της αντίστοιχης πράξης της ALU που σχεδιάστηκε στο part1. Τα πρώτα δύο bits του func είναι 1, άρα δεν χρειάζονται για την αποκωδικοποίηση της εντολής. Για την υλοποίηση των πράξεων με opcode = 100000 θα πρέπει να επιλεχτεί RF_B_sel = 0, έτσι ώστε στην έξοδο RF_B του Register File να διαβάζουμε τον καταχωρητή με διεύθυνση rt, όπου rt = Instr[15:11]. Στην συνέχεια, θα πρέπει να είναι ALU_Bin_sel = 0, έτσι ώστε στον MUX να επιλεχτεί το σήμα RF_B, το οποίο πρέπει να αποτελεί την δεύτερη είσοδο της ALU. Το ALU_func ισούται με το Instr[3:0], (όπως ειπώθηκε παραπάνω δεν θα χρησιμοποιήσουμε τα bit Instr[5:4], αφού είναι όμοια για όλες τις πράξεις που υλοποιεί η ALU). Τέλος το αποτέλεσμα της πράξης

(σήμα ALU_out) θα αποθηκευτεί στον καταχωρητή του Register File με διεύθυνση rd = Instr[20:16]. Για να γίνει αυτό θα πρέπει να τεθεί RF_WrEn = 1 και RF_WrData_sel = 0, ώστε ο MUX να οδηγήσει το ALU_out στην είσοδο Write_data του Register File. Με την έλευση θετικής ακμής του ρολογιού, το αποτέλεσμα της ALU-πράξης θα αποθηκευτεί στον επιθυμητό καταχωρητή του RF.

Εντολές “li”, “lui” με opcode = Instr[31:26] = 111000 ή 111001: Θα πρέπει η έξοδος RF_A της βαθμίδας decode να είναι μηδενική, άρα πρέπει συνεχώς να διαβάζεται ο καταχωρητής 0 που έχει αποθηκευμένο μόνιμα το 0, για αυτό θα είναι rs = Instr[25:21] = 5’b0. Έπειτα, αφού το RF_A θα είναι μηδενικό, θα χρησιμοποιηθεί η βαθμίδα ALU_stage, ώστε να προσθέτει (ALU_func = 4’b0) το RF_A, που είναι το 32’b0, με το 32 bit Immed, που προέρχεται από την μονάδα cloud. Τέλος, το αποτέλεσμα της πράξης (σήμα ALU_out) θα αποθηκευτεί στον καταχωρητή του Register File με διεύθυνση rd = Instr[20:16]. Για να γίνει αυτό θα πρέπει να τεθεί RF_WrEn = 1 και RF_WrData_sel = 0, ώστε ο MUX να οδηγήσει το ALU_out στην είσοδο Write_data του Register File. Με την έλευση θετικής ακμής του ρολογιού, το αποτέλεσμα της ALU-πράξης θα αποθηκευτεί στον επιθυμητό καταχωρητή του RF.

Το 32bit Immed προέρχεται από την μονάδα “cloud” που έχει ως είσοδο το Instr[15-0] και κάνει ανάλογα με το opcode της εκάστοτε πράξης ZeroFill ή SignExtend.

Εντολές “beq”, “bne” με opcode = Instr[31:26] = 000000 ή 000001: Ως έξοδοι RF_A και RF_B της βαθμίδας decode θα πρέπει να είναι οι έξοδοι των καταχωρητών του RF με διεύθυνση rs = Instr[25:21], και rd = Instr[20:16]. Γι' αυτό θα πρέπει να επιλεχτεί RF_B_sel = 1, ώστε στην έξοδο RF_B του Register File να διαβάζουμε τον καταχωρητή με διεύθυνση rd. Έπειτα, θα πρέπει να γίνει ALU_Bin_sel = 0, έτσι ώστε στο MUX να

επιλεχτεί το σήμα RF_B, το οποίο πρέπει να αποτελεί τη δεύτερη είσοδο της ALU. Το ALU_func ισούται με 4'b0001, θα αφαιρεί από το RF_A το RF_B, αν το αποτέλεσμα (ALU_out) είναι μηδέν τότε RF[rs] = = RF[rd] και Zero = 1, αλλιώς θα είναι Zero = 0. Στην συνέχεια, αναλόγως αν ικανοποιείται ή όχι η συνθήκη της εντολής, θα επιλέγεται κατάλληλα το PC_sel στην βαθμίδα IF_STAGE (απαραίτητο να γίνει PC_LdEn = 1).

Εντολή "b" με opcode = Instr[31:26] = 111111: Πρέπει να γίνει PC_sel = 1. Η 32bit λέξη SignExtend(Imm) << 2 θα παραχθεί κατάλληλα από την μονάδα cloud της βαθμίδας decode (απαραίτητο να γίνει PC_LdEn = 1).

Εντολές "addi", "andi", "ori" με opcode = Instr[31:26] = 110000, 110010, 110011: Η έξοδος RF_A της βαθμίδας "decode" οδηγείται στην ALU. Ως δεύτερη είσοδος της ALU θα πρέπει να είναι το 32 bit Immed, το οποίο προέρχεται από την μονάδα cloud του decode που έχει ως είσοδο το Instr[15-0] και κάνει, ανάλογα με το opcode της εκάστοτε πράξης, ZeroFill ή SignExtend. Επομένως, θα πρέπει να τεθεί ALU_Bin_sel = 1, το ALU_func θα πρέπει να τεθεί ίσο με 4'b0, 4'b10, 4'b11 για την πράξη "addi", "andi" και "ori" αντίστοιχα. Παρατηρούμε ότι το ALU_func ισούται για αυτές της εντολές με το Instr[29:26]. Τέλος, το αποτέλεσμα της πράξης (σήμα ALU_out) θα αποθηκευτεί στον καταχωρητή του Register File με διεύθυνση rd = Instr[20:16]. Για να γίνει αυτό θα πρέπει να τεθεί RF_WrEn = 1 και RF_WrData_sel = 0, ώστε ο MUX να οδηγήσει το ALU_out στην είσοδο Write_data του Register File. Με την έλευση θετικής ακμής του ρολογιού, το αποτέλεσμα της ALU-πράξης θα αποθηκευτεί στον επιθυμητό καταχωρητή του RF.

Εντολές lw, lb με opcode = Instr[31:26] = 000011, 001111: Αρχικά για την εύρεση της διεύθυνσης της θέσης μνήμης, όπου θα γίνει ανάγνωση του περιεχομένου της, θα χρησιμοποιήσουμε την ALU_STAGE. Στην ALU εισέρχεται στην 1η είσοδο η έξοδος

RF_A (έξοδος του καταχωρητή με διεύθυνση rs = Instr[25:21] του Register File) της βαθμίδας DEC_STAGE, ενώ πρέπει να γίνει ALU_Bin_sel = 1, για να εισέλθει στην 2η είσοδο της ALU, το 32bit Immed. Η πράξη που πρέπει να εκτελέσει η ALU είναι πρόσθεση οπότε πρέπει ALU_func = 0000. Στην συνέχεια η έξοδος ALU_out εισέρχεται στην θύρα διευθυνσιοδότησης της μνήμης, όπου τα bits ALU_out[11:2] αποτελούν την διεύθυνση της θέσης μνήμης που θα γίνει η ανάγνωση. Έχοντας το MEM_WrEn στο 0, στην επόμενη θετική ακμή του ρολογιού, θα γίνει ανάγνωση του περιεχομένου της θέσης μνήμης. Η έξοδος της μνήμης εισέρχεται μέσα σε μία επιπρόσθετη μονάδα, που ανάλογα με το opcode της πράξης αποφασίζει ή όχι αν θα κάνει ZeroFill(31 downto 8) στην έξοδο της μνήμης. Στην συνέχεια, για να εγγραφεί το περιεχόμενο MEM_out στον καταχωρητή με διεύθυνση rd = Instr[20:16] του Register File, επιλέγεται RF_WrData_sel = 1, έτσι ώστε στην είσοδο εγγραφής του RF να συνδεθεί η έξοδος MEM_out και στην επόμενη θετική ακμή του ρολογιού, θα εγγραφεί στον κατάλληλο καταχωρητή του Register File.

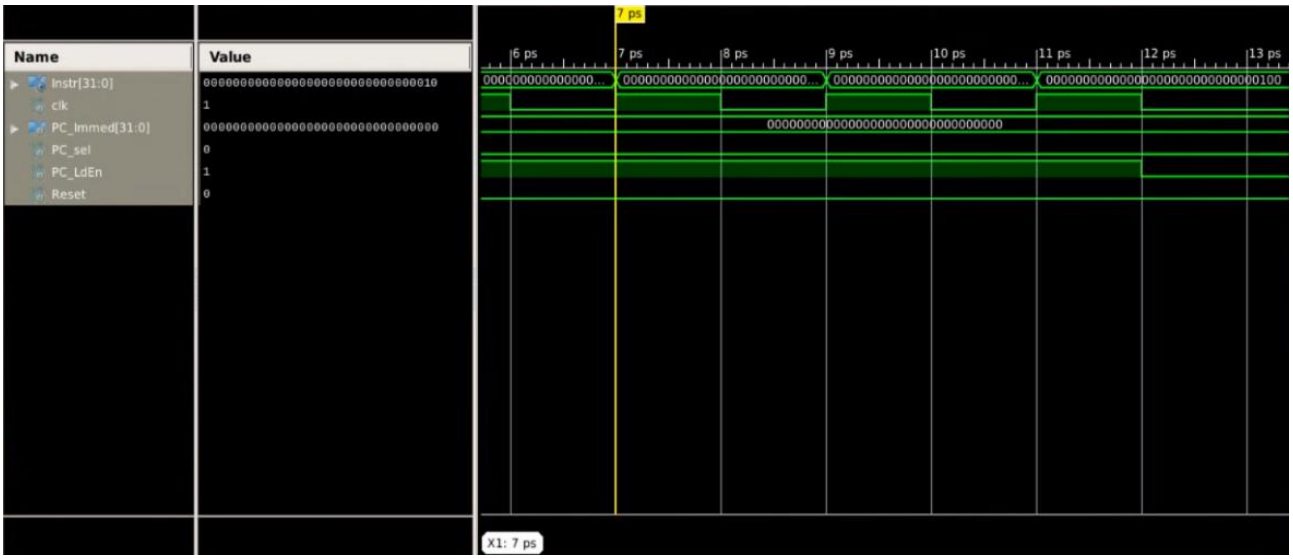
<u>Εντολές sw, sb με opcode = Instr[31:26] = 011111, 000111:</u> Αρχικά για να αναγνωσθεί στην έξοδο RF_B του DEC_STAGE, ο καταχωρητής του Register File, με διεύθυνση rd = Instr[20:16], επιλέγεται RF_B_sel = 1. Στην συνέχεια, για την εύρεση της διεύθυνσης της θέσης μνήμης όπου θα γίνει εγγραφή, θα χρησιμοποιήσουμε την ALU_STAGE. Στην ALU εισέρχεται στην 1η είσοδο η έξοδος RF_A (έξοδος του καταχωρητή με διεύθυνση rs = Instr[25:21] του Register File) της βαθμίδας DEC_STAGE, ενώ πρέπει να γίνει ALU_Bin_sel = 1, για να εισέλθει στην 2η είσοδο της ALU, το 32bit Immed. Η πράξη που πρέπει να εκτελέσει η ALU είναι πρόσθεση, οπότε πρέπει ALU_func = 0000. Έπειτα, η έξοδος ALU_out εισέρχεται στην θύρα διευθυνσιοδότησης της μνήμης, όπου τα bits ALU_out[11:2] αποτελούν την διεύθυνση της θέσης μνήμης που θα γίνει η εγγραφή. Η έξοδος RF_B, πριν εισέλθει στην είσοδο δεδομένων της μνήμης, διέρχεται από μια επιπρόσθετη μονάδα ελέγχου, όπου

αποφασίζει αν θα γίνει ή όχι ZeroFill(31 downto 8). Έχοντας το MEM_WrEn = 1, στην επόμενη θετική ακμή του ρολογιού, θα γίνει εγγραφή του περιεχομένου του καταχωρητή rd στην κατάλληλη θέσης μνήμης.

B1)

Εικόνα 6

<u>Επεξήγηση:</u> Αρχικά κάνουμε reset τον καταχωρητή PC, το PC_Out γίνεται 0 και στην επόμενη θετική ακμή του ρολογιού, τα bit [11:2] του PC_Out είναι 0, οπότε θα διαβαστεί από την μνήμη ROM η 1η γραμμή που είναι το 32'b1. Γίνεται PC_LdEn = 0, οπότε συνεχίζει το PC_Out να είναι 0 και η έξοδος της μνήμης να είναι 32'b1.



Εικόνα 7

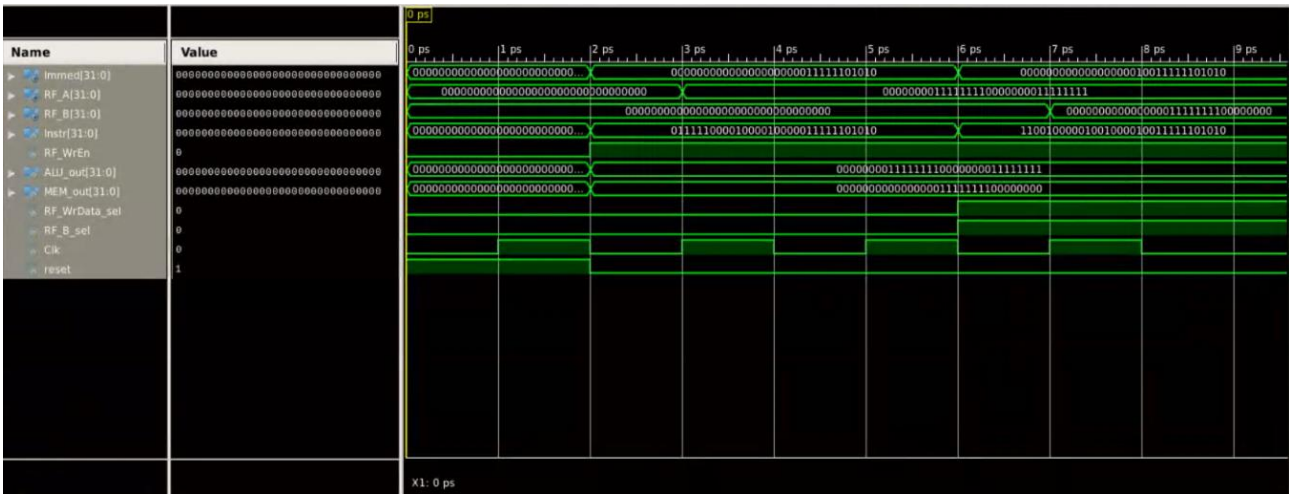
Τη χρονική στιγμή 7ps, έχουμε θετική ακμή του ρολογιού, το PC_LdEn ήδη έχει γίνει 1 από τη στιγμή 4ps, άρα το PC_Out αυξάνεται κατά 4, δηλαδή γίνεται 4 (32'b100). Όμως, ως διεύθυνση για την μνήμη θα χρησιμοποιηθούν τα bit [11:2] το οποίο αντιστοιχεί στην διεύθυνση 10'b1. Στην έξοδο της μνήμης, παίρνουμε τη 2η σειρά του αρχείου "rom.data" που είναι το 32'b10 και η διαδικασία συνεχίζεται.

Εικόνα 8

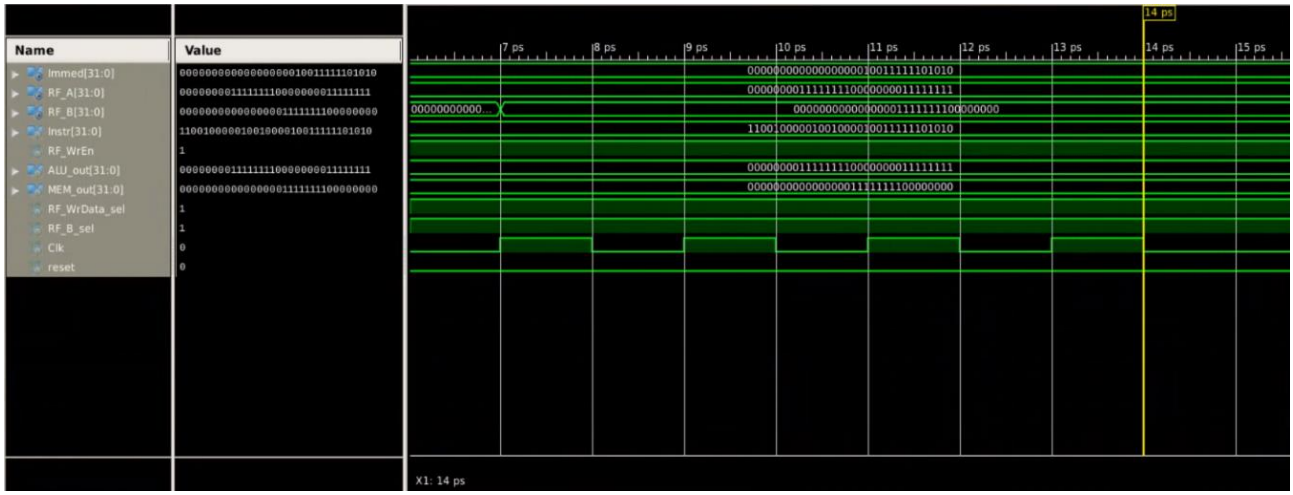
Τη χρονική στιγμή 52ps, κάνουμε reset. Το PC_Out γίνεται 0 και στην επόμενη θετική ακμή του ρολογιού, στην έξοδο της μνήμης θα ξαναπάρουμε την 1η σειρά του αρχείου "rom.data" και η διαδικασία επαναλαμβάνεται.

Γ) Για την υλοποίηση της βαθμίδας χρησιμοποιήσαμε τα εξαρτήματα που φαίνονται στο σχήμα 2 της εκφώνησης του part2 και τη μονάδα "cloud", που δέχεται ως είσοδο το Instr[15:0] και, ανάλογα με το opcode της πράξης, κάνει ZeroFill, SignExtend ή SignExtend και/με ολίσθηση "<< 2". Για τις εντολές που δεν χρησιμοποιούν το Immed, η μονάδα "cloud" σχεδιάστηκε να βγάζει μηδέν. Επιπλέον, στη βαθμίδα DECSTAGE προστέθηκε μία είσοδος RF_WrEn, το οποίο ενεργοποιεί και απενεργοποιεί την εγγραφή στο Register File. Παραθέτουμε ένα παράδειγμα προσομοίωσης της βαθμίδας DECSTAGE, το οποίο φαίνεται στις

παρακάτω εικόνες:



Εικόνα 9: Παράδειγμα 1



Εικόνα 10:Συνέχεια παράδειγμα 1

<u>Επεξήγηση:</u> Αρχικά γίνεται reset στο Register File, οπότε αποθηκεύεται 32'b0 στους 32 καταχωρητές, τα σήματα ελέγχου για τα πρώτα 2ps είναι όλα στο 0 και ορθώς παίρνουμε 0 στις εξόδους RF_A και RF_B. Στα 2ps, έχουμε:

- Instr = 011111_00001_00001_0000011111110101, όπου:
  - opcode = Instr[31:26],

- rs = Instr[25:21],
- rd = Instr[20:16],
- rt = Instr[15:11],
- Imm = Instr[15:0]

- ALU_out = 32’b11111111000000011111111
- MEM_out = 32’b00000000011111111
- RF_WrEn = 1
- RF_WrData_sel = 0

Το opcode είναι 011111, το immed που εισέρχεται στη βαθμίδα cloud είναι 16'b0000011111101010. Άρα το 32 bit Immed εξόδου του cloud είναι SignExtend(immed) (βάζει 16 μηδενικά στο immed). Η πληροφορία του ALU_Out γραφεται όταν έρθει θετική ακμή του ρολογιού στον καταχωρητή με διεύθυνση 00001 του Register File, (αφού: write_register -> Instr[20:16] = 00001, και RF_WrEn = 1). Στην εξόδο RF_A διαβάζουμε το περιεχόμενο του καταχωρητή με διεύθυνση read_register1 (Instr[25:21]) = 00001 (καταχωρητής 1), το οποίο πριν την παραπάνω εγγραφή είναι μηδενικό, και μετά από την εγγραφή ισούται με το ALU_Out.
Στην εξόδο RF_B διαβάζουμε το περιεχόμενο του καταχωρητή με διεύθυνση read_register2 (Instr[15:11]) = 00000 (καταχωρητής 0), ο οποίος έχει πάντοτε αποθηκευμένο το μηδέν.

Την χρονική στιγμή 6ps γίνεται:

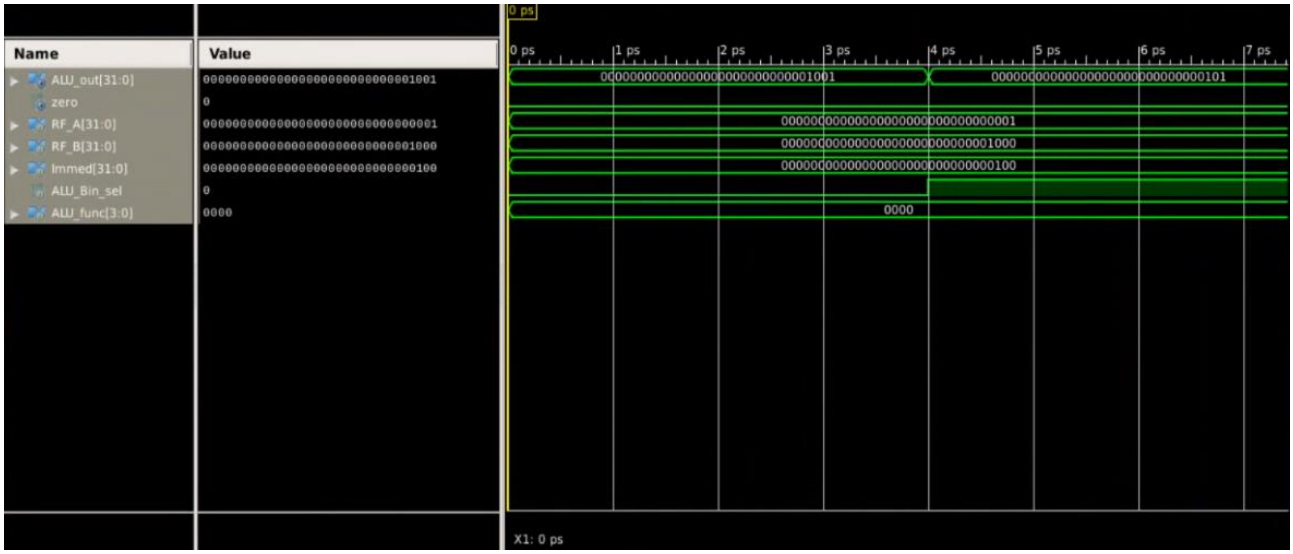
- Instr = 110010_00001_00100_0010011111101010
- write_register->00100 (μπαίνει και στο mux, RF_B_sel = 1)
- read_register2 -> 00100
- read_register1 -> 00001
- RF_WrData_sel = 1
- RF_B_sel =1
- RF_WrEn = 1 (παραμένει σταθερό)

Επομένως στην είσοδο δεδομένων του Register File θα συνδεθεί το MEM_out, και στην είσοδο ανάγνωσης 2 θα συνδεθεί το Instr[20:16] = rd. Στην έξοδο RF_A διαβάζουμε το περιεχόμενο

του καταχωρητή με διεύθυνση read_register1 (Instr[25:21]) = 00001 (καταχωρητής 1), όπου δεν έχει γίνει κάποια αλλαγή. Ενώ αρχικά (τη χρονική στιγμή 6ps) στην έξοδο RF_B διαβάζουμε το περιεχόμενο του καταχωρητή, με διεύθυνση read_register2 (Instr[15:11]) = 00100, ο οποίος μετά το reset έχει αρχικοποιηθεί στο 0,...την στιγμή 7ps που έρχεται θετική ακμή του ρολογιού αποθηκεύεται στον καταχωρητή αυτόν το MUX_out, (αφού write_register->00100).

Το opcode είναι 110010, το immed είναι 0000011111101010
Άρα το 32bit Immed εξόδου είναι ZeroFill(immed) (η μονάδα cloud προσθέτει 16 μηδενικά στο immed).

Δ) Για την υλοποίηση της βαθμίδας χρησιμοποιήσαμε μόνο τα 2 εξαρτήματα που φαίνονται στην σχήμα 3 της εκφώνησης του part2. Επιπλέον, προσθέσαμε ένα επιπλέον σήμα εξόδου (1bit) “zero”, το οποίο είναι η έξοδος “Zero” της ALU.

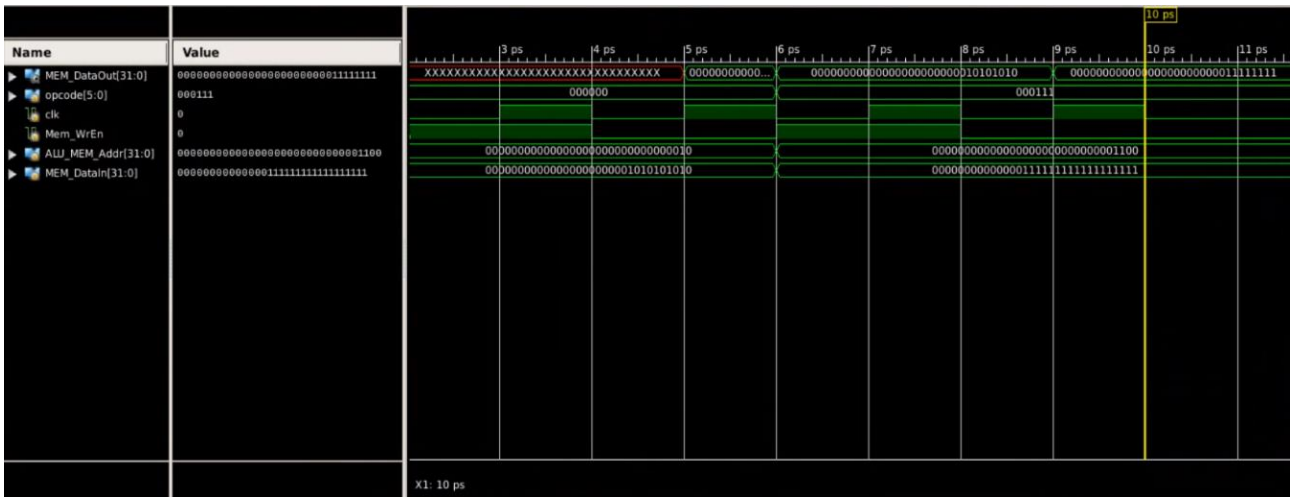


Εικόνα 11: Προσομοίωση 1 της βαθμίδας ALU-STAGE

Επεξήγηση: Αρχικά, το RF_A είναι 32'b1, το RF_B είναι 32'b1000, το Immed είναι 32'b100, το ALU_Bin_sel είναι 0, άρα η 2η είσοδος της ALU είναι το RF_B. Το ALU_func είναι 4'b0000, οπότε θα γίνει πρόσθεση των RF_A και RF_B και το αποτέλεσμα θα είναι 32'b1001. Τη χρονική στιγμή 4ps, το ALU_Bin_sel γίνεται 1 άρα η 2η είσοδος της ALU είναι το Immed. Οι υπόλοιπες είσοδοι της ALU παραμένουν σταθερές,

οπότε θα γίνει πρόσθεση μεταξύ του RF_A και του Immed και το αποτέλεσμα είναι 32'b101. Τέλος, οι έξοδοι της ALU είναι διάφορες του μηδενός οπότε το zero = 0.

Ε) <u>MEM-STAGE:</u> Στη βαθμίδα προστέθηκε: μία είσοδος 6 bit "opcode", η οποία είναι το Instr[31:26], και μια μονάδα που δέχεται ως είσοδο την έξοδο της μνήμης και, ανάλογα με το opcode της πράξης, αποφασίζει αν κάνει ZeroFill(31 down to 8) στην έξοδο της μνήμης RAM ή όχι. Παραθέτουμε ένα παράδειγμα λειτουργίας της βαθμίδας MEM-STAGE:



Εικόνα 12: Παράδειγμα εκτέλεσης της βαθμίδας MEM-STAGE

<u>Επεξήγηση:</u> Αρχικά, ALU_MEM_Addr = 32'b10 και MEM_DataIn = 32'b1010101010, MEM_WrEn = 1. Με την έλευση θετικής ακμής του ρολογιού, εγγράφεται στη διεύθυνση ALU_MEM_Addr το MEM_DataIn. Στη συνέχεια, MEM_WrEn = 0, οπότε στη επόμενη θετική ακμή του ρολογιού θα πάρουμε στην έξοδο της βαθμίδας MEM-STAGE το MEM_DataIn, χωρίς ZeroFill(31 down to 8), γιατί opcode = 6'b0. Τη χρονική στιγμή 6ps, opcode = 6'b111 και στην έξοδο της βαθμίδας θα πραγματοποιηθεί ZeroFill(31 down to 8).

PART 3

Αρχικά συνδέσαμε τις 4 βαθμίδες που σχεδιάσαμε στο part2 σύμφωνα με τις προδιαγαφές του, με την προσθήκη μιας μονάδας cloud, που με βάση το opcode αποφασίζει αν θα γίνει ή όχι ZeroFill ή SignExtend ή SignExtend << 2 στο Instr[15-0], ώστε να μετατραπεί σε 32bit Immed. Ακόμη, προσθέσαμε και μία δεύτερη μονάδα στην βαθμίδα MEM_STAGE που με βάση το opcode της εντολής αποφασίζει αν θα κάνει ή όχι ZeroFill(31 downto 8) στην είσοδο ή στην έξοδο δεδομένων της μνήμης.

Στη συνέχεια, κατά την σύνδεση της μονάδας ελέγχου στο datapath, παρουσιάστηκαν τα εξής προβλήματα:

1ο πρόβλημα: Καθώς μεταβαλλόταν το Instr, δεν προλάβαινε να ολοκληρωθεί μια εντολή, άλλαζε το Instr και παρήγαγε η εντολή λάθος αποτελέσματα. Για αυτόν τον λόγο αποφασίστηκε να τοποθετηθεί ανάμεσα στην βαθμίδα IF και DECODE μια διάταξη δύο καταχωρητών σε σειρά, στην ουσία η διάταξη αυτήν είναι ένας καταχωρητής ολίσθησης.

Το 2ο πρόβλημα είναι ότι το Instr παρέμενε με την παραπάνω προσθήκη σταθερό για 4 κύκλους ρολογιού, ενώ κάποιες εντολές επιτυγχάνονταν σε λιγότερες από 4 καταστάσεις, μεταβαίνοντας στην επόμενη κατάσταση που ήταν η i_fetch, χωρίς να αλλάξει το Instr, με αποτέλεσμα οι επόμενες εντολές να παράγουν λανθασμένα αποτελέσματα. Για την επίλυση αυτού του προβλήματος αποφασίστηκε όλες οι εντολές με εξαίρεση τις “b”, “beq” και “bne” να μεταβαίνουν από 4 καταστάσεις για να ολοκληρωθούν. Οι καταστάσεις nop, nop2, nop3, δεν ενεργοποιούν κανένα σήμα, και οι τρεις προστέθηκαν για να διέρχεται το σήμα Instr = 32’b0 από τέσσερις καταστάσεις για να ολοκληρωθεί.

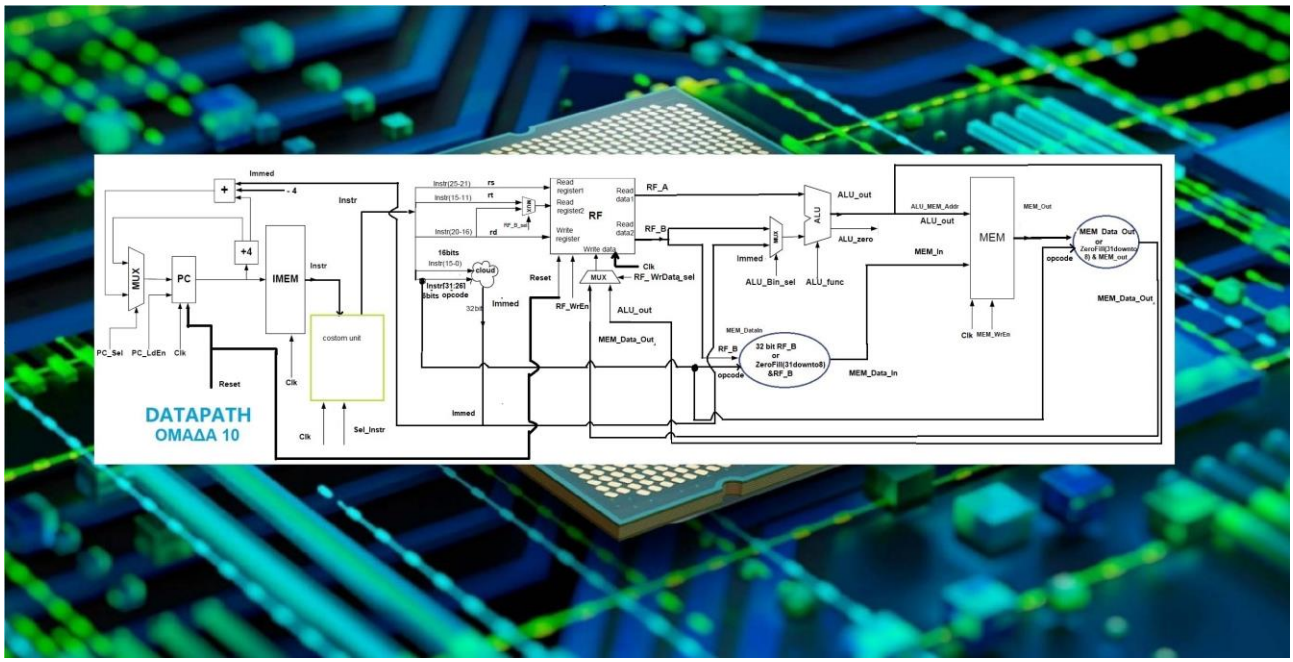
Στην συνέχεια, παρουσιάστηκε 3ο πρόβλημα με τις εντολές branch και αυτό γιατί στα: CONTROL, DECODE, ALU, MEM_STAGE, μετά την προσθήκη των 2 καταχωρητών αργεί να αλλάξει το σήμα Instr, σε σχέση με την βαθμίδα IF_STAGE. Γι'αυτό αποφασίστηκε ανάμεσα στην βαθμίδα IF και DECODE να τοποθετηθεί το εξάρτημα που περιγράφεται από τον παρακάτω κώδικα, ο οποίος είναι ενσωματωμένος μέσα στο module Datapath.v

```
//Custom Unit ==============
        reg [31:0] Ins, Input;
       wire [31:0]  Instr;
        always @(*)
        begin
             if(Sel_Instr)
                   In = Instr;
             else
                   In = Input;
        end
        always @(posedge clock)
        begin
                   Ins <= Instr;
              Input <= Ins;
        end
       //============================
```

Στην μονάδα αυτήν όταν θα φτάσει σήμα με opcode branch εντολής (η συνθήκη της beq, bne είναι ορθή, ή φτάσει το σήμα της εντολής b) τότε γίνεται Sel_Instr = 1, (συνδέεται η βαθμίδα decode απευθείας με την βαθμίδα IFSTAGE), αλλιώς Sel_Instr = 0, δηλαδή θα ξανασυνδέεται η διάταξη των δύο καταχωρητών (καταχωρητής ολίσθησης) με το υπόλοιπο σύστημα. Η κατάσταση nop4 προστέθηκε για να γίνει σωστά (με την κατάλληλη χρονική καθυστέρηση) η μετάβαση του σήματος Instr στην εντολή b μέσω τις πρόσθετης μονάδας (costom unit, πιο κατανοητή η λειτουργία αυτής της μονάδας θα γίνει σε παρακάτω - 3ο παράδειγμα στις

τελικές προσομοιώσεις). Η μονάδα αυτήν αποτελείται από έναν MUX και δύο καταχωρητές σε σειρά - (καταχωρητή ολίσθησης).

Το 4ο και τελευταίο πρόβλημα ήταν ότι στις εντολές branch το σήμα PC_out στην βαθμίδα IF_STAGE ήταν μεγαλύτερο κατά 4, από ότι θα έπρεπε, αφού κάθε εντολή (και οι εντολές branch) ξεκινάει από την κατάσταση i_fetch, στην οποία το PC_out αυξάνεται κατά 4. Όμως εμείς θέλουμε στην επόμενη μετά την κατάσταση i_fetch, την κατάσταση branch το PC_out να αυξηθεί κατά 4 + Immed, γι' αυτό στον αθροιστή που υπολογίζει την τιμή PC + 4 + Immed για τις εντολές διακλάδωσης, αποφασίσαμε να ενσωματώσουμε έναν αφαιρέτη κατά 4 (αφαιρείται το + 4 που προέξυψε από την προηγούμενη κατάσταση i_fetch), ώστε αν τελικά πραγματοποιηθεί η εντολή διακλάδωσης το σήμα PC_out να ισούται με το αναμενόμενο (στο τελικό αρχείο incrementor_pc_immediate.v έχει ενσωματωθεί η παραπάνω αφαίρεση κατά 4, η παραπάνω συνθήκη θα γίνει πιο κατανοητή στο παράδειγμα 3 της τελικής προσομοίωσης).



Εικόνα 13: Σχηματικό Διάγραμμα Datapath.

Στην συνέχεια θα αναλύσουμε την λειτουργία της FSM ελέγχου. Η «καρδιά» της FSM είναι ο καταχωρητής κατάστασης, που περιέχει την παρούσα κατάσταση, “currState”, του συστήματος.

Ένα μπλοκ Συνδυαστικής Λογικής υπολογίζει ποια θα είναι η επόμενη κατάσταση, “nextState”, αμέσως μετά την επόμενη θετική ακμή του ρολογιού, σαν συνάρτηση της παρούσας κατάστασης και των εισόδων του κυκλώματος ελέγχου, στην προκειμένη περίπτωση του op, δηλαδή του Instr[31:26]. Οι έξοδοι του κυκλώματος ελέγχου, δηλαδή τα σήματα ελέγχου του datapath, είναι επίσης συναρτήσεις της παρούσας κατάστασης και μερικά από αυτά μπορεί να είναι συναρτήσεις ορισμένων εισόδων του κυκλώματος ελέγχου πχ. ALU_func = Instr[3:0], και ALU_zero.

Αρχικά, ορίσαμε ποιες είναι οι καταστάσεις της FSM (15 + 4 “nop” καταστάσεις), δίνοντας τους ονόματα όπως i_fetch, decode_rr, mem_addr, alu_exec κλπ και μια κωδικοποίηση όπως 00001, 00010, 00011, κλπ. Αυτό το κάνουμε με την εντολή "parameter" της Verilog. Η διαφορά της "parameter" από την "define" είναι ότι η τιμή για τα συμβολικά ονόματα που ορίζονται με τη βοήθεια της "parameter" μπορεί να αλλάξει αργότερα έτσι το εργαλείο σύνθεσης θα μπορέσει να βελτιστοποιήσει, πιθανόν, την κωδικοποίηση των καταστάσεων.
Στην συνέχεια, μέσα στο always @(posedge clock) block υλοποιείται ο καταχωρητής κατάστασης, ο οποίος ακριβώς πριν την έλευση της θετικής ακμής του ρολογιού βλέπει ποια είναι η τιμή “nxtState” και μόλις έρθει η θετική ακμή του ρολογιού την αναθέτει στο “currState”.

Στο επόμενο always block (γραμμές 64-150) όπου χρησιμοποιούνται εντολές “case” και “if”, υπολογίζεται η επόμενη κατάσταση “nxtState” με βάση την παρούσα κατάσταση “currState” και των εισόδων Instr, ALU_zero (και του op, όπου op = Instr[31:26]).

Στο always block (γραμμές 152-265), χρησιμοποιώντας εντολές “if”, περιγράφουμε τις εξόδους της fsm, δηλαδή τα σήματα ελέγχου για το datapath.

ΕΝΤΟΛΕΣ - ΚΑΤΑΣΤΑΣΕΙΣ

Για την εκτέλεση των εντολών με opcode = 100000, περνάμε από τις εξής καταστάσεις: i_fetch < decode_rr < alu_exec < alu_wb.

Για την εκτέλεση των εντολών με opcode = 111000, opcode = 111001, περνάμε από τις εξής καταστάσεις: i_fetch < decode_rr < alu_li < alu_wb.

Για την εκτέλεση των εντολών με opcode = 110000, opcode = 110010, opcode = 110011, περνάμε από τις εξής καταστάσεις: i_fetch < decode_rr < alu_i < alu_wb.

Για την εκτέλεση της εντολής με opcode = 111111, περνάμε από τις εξής καταστάσεις: i_fetch < branch <nop4 < change_state< change_state2.

Για την εκτέλεση των εντολών με opcode = 000000, opcode = 000001, περνάμε από τις εξής καταστάσεις: i_fetch < decode_bb_alu_sub < alu_sub <
if(true) => branch < change_state_beq_bne < change_state < change_state2
else => nop3

Για την εκτέλεση των εντολών με opcode = 000011, opcode = 001111, περνάμε από τις εξής καταστάσεις: i_fetch < decode_rr < mem_addr < mem_rd.

Για την εκτέλεση των εντολών με opcode = 000111, opcode = 011111, περνάμε από τις εξής καταστάσεις: i_fetch < decode_rr < mem_addr < mem_wr.

Τέλος, αν έρθει Instr = 32’b0 περνάμε από τις εξής καταστάσεις i_fetch < nop < nop2 < nop3

Οι καταστάσεις nop, nop2, nop3, nop4, δεν ενεργοποιούν κανένα σήμα, οι 3 πρώτες προστέθηκαν για να διέρχεται το σήμα Instr = 32’b0 από τέσσερις καταστάσεις για να ολοκληρωθεί, η nop4 προστέθηκε για να γίνει σωστά η μετάβαση του σήματος Instr στην εντολή b (μέσω τις πρόσθετης μονάδας (costom unit)).

Ανάλυση σημάτων ελέγχου FSM για το datapath ανά κάθε μία κατάσταση.

1) Αν η τρέχουσα κατάσταση είναι η i_fetch γίνονται οι εξής αναθέσεις: PC_LdEn = 1, PC_Sel = 0, και απενεργοποιούνται τυχόν ξεχασμένα σήματα enable RF_WrEn = 0, Mem_WrEn = 0,
αλλιώς τίθεται PC_LdEn = 0.
2) Αν η τρέχουσα κατάσταση είναι η decode_rr γίνονται οι εξής αναθέσεις: RF_WrEn = 0, RF_WrData_sel = 0, και RF_B_sel = 0.
3) Αν η τρέχουσα κατάσταση είναι η decode_bb_alu_sub γίνονται οι εξής αναθέσεις: ALU_Bin_sel = 0, ALU_func = 0001, RF_B_sel = 1.
4) Αν η τρέχουσα κατάσταση είναι η alu_li γίνονται οι εξής αναθέσεις: ALU_func = 0000, ALU_Bin_sel = 1, RF_WrEn = 0, RF_B_sel = 0 και RF_WrData_sel = 0.
5) Αν η τρέχουσα κατάσταση είναι η alu_exec γίνονται οι εξής αναθέσεις: ALU_func = Instr[3:0], ALU_Bin_sel = 0.
6) Αν η τρέχουσα κατάσταση είναι η alu_i γίνονται οι εξής αναθέσεις: ALU_func = Instr[29:26], ALU_Bin_sel = 1 και RF_B_sel = 1.
7) Αν η τρέχουσα κατάσταση είναι η alu_wb γίνονται οι εξής αναθέσεις: RF_WrEn = 1, RF_B_sel = 0 και RF_WrData_sel = 0.
8) Αν η τρέχουσα κατάσταση είναι η alu_sub γίνονται οι εξής αναθέσεις: ALU_func = 0001, ALU_Bin_sel = 0 και RF_B_sel = 1.

9) Αν η τρέχουσα κατάσταση είναι η mem_addr γίνονται οι εξής αναθέσεις: ALU_func = 0000 και ALU_Bin_sel = 1.

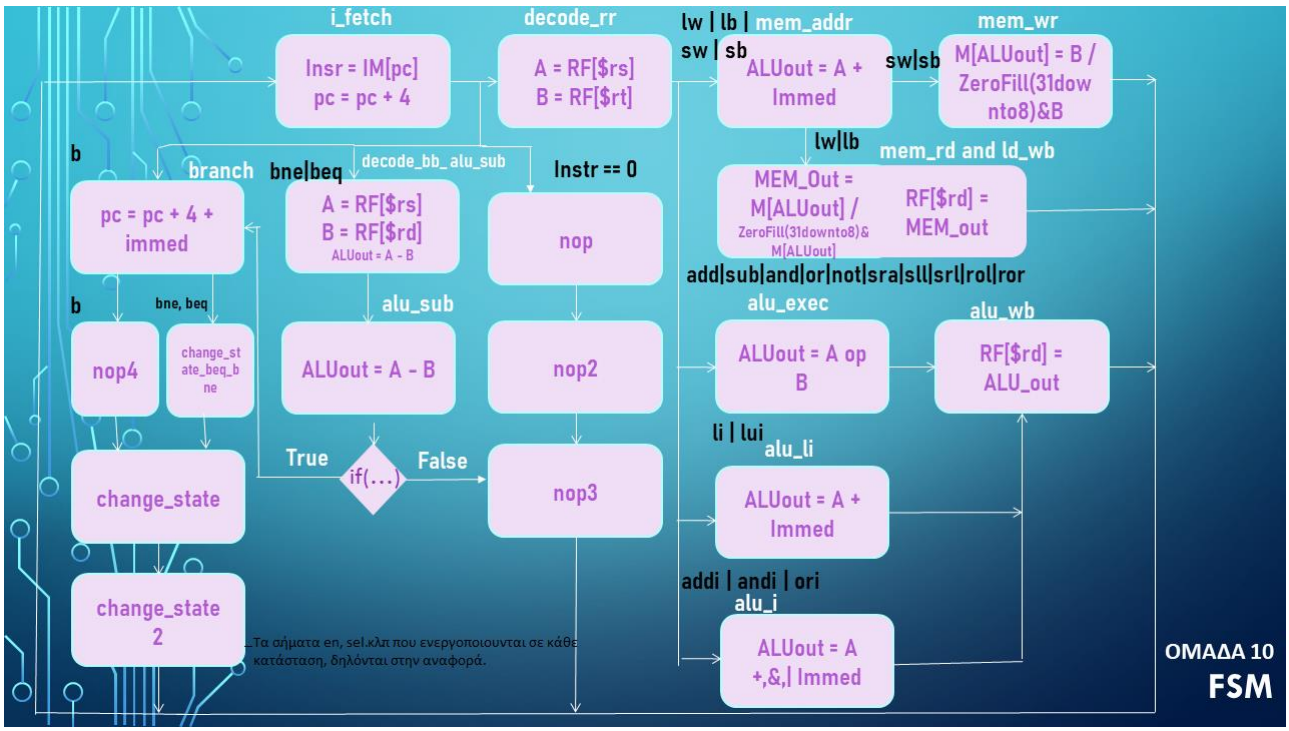
10) Αν η τρέχουσα κατάσταση είναι η mem_wr γίνονται οι εξής αναθέσεις: Mem_WrEn = 1 και RF_B_sel = 1.

11) Αν η τρέχουσα κατάσταση είναι η mem_rd γίνονται οι εξής αναθέσεις: Mem_WrEn = 0 και RF_B_sel = 1, RF_WrEn = 1, και RF_WrData_sel = 1. Η παρούσα κατάσταση συγχωνεύθηκε με την ld_wb που φαίνεται στο σχήμα της εκφώνησης (εικόνα datapath.png).

12) Αν η τρέχουσα κατάσταση είναι η change_state, change_state2 ή change_state_beq_bne γίνονται η εξής αναθέση: Sel_Instr = 1, αλλιώς Sel_Instr = 0. Αυτές οι καταστάσεις προσθέθηκαν (στις εντολές b, bne, beq) για την ορθή μετάβαση του σήματος Instr μέσω της costom unit.

13) Αν η τρέχουσα κατάσταση είναι η branch γίνονται η εξής αναθέση: PC_Sel = 1 και PC_LdEn = 1, αλλιώς PC_Sel = 0.

14) Αν η τρέχουσα κατάσταση είναι η nop, nop2, nop3, nop4 δεν γίνεται καμία ανάθεση, αυτές οι καταστάσεις προστέθηκαν για να μην ενεργοποιείται κανένα σήμα ελέγχου.



Εικόνα 14: Σχηματικό Διάγραμμα FSM

Για όλες τις προσομοιώσεις χρησιμοποιήθηκε clock με περίοδο 2ps. Το οποίο υλοποιήθηκε μέσα στο αρχείο test bench με όνομα Test_A.v.

```
always
  begin
   clock = 0;
  #0.001;
  clock = 1;
  #0.001;
  end
```

Έπειτα στο αρχείο Test_A.v γίνεται Reset έτσι ώστε να αρχικοποιηθούν στο 32'b0 όλοι οι καταχωρητές του Datapath.
Για εκπαιδευτικούς λόγους προσθέσαμε στο αρχείο Test_A.v επιπλέον σήματα τύπου wire ούτως ώστε να βλέπουμε στις προσομοιώσεις τα εσωτερικά σήματα των βαθμίδων, προκειμένου να διαπιστώσουμε ότι ο συνολικός επεξεργαστής λειτουργεί με την επιθυμητή συμπεριφορά.

Ακολουθούν στο φάκελο simulations
key: aN7en4WHkOmV9jmKOoLu4A
screenshots με τις κυματομορφές προσομοίωσης (όλες οι περιπτώσεις από το rom.data) και 4 screenshots με την χρήση της εντολής $display(), όπου τυπώνεται το περιεχόμενο των καταχωρητών 0-20, καθώς και το περιεχόμενο της μνήμης στην θέση 2 και 3. Συγκρίνοντας της προσομοιώσεις με το αρχείο Program.txt παρατηρούμε ότι όλες οι εντολές πραγματοποιούνται επιτυχώς.

Ακολουθούν τέσσερα παραδείγματα από τις παραπάνω προσομοιώσεις των εντολών.

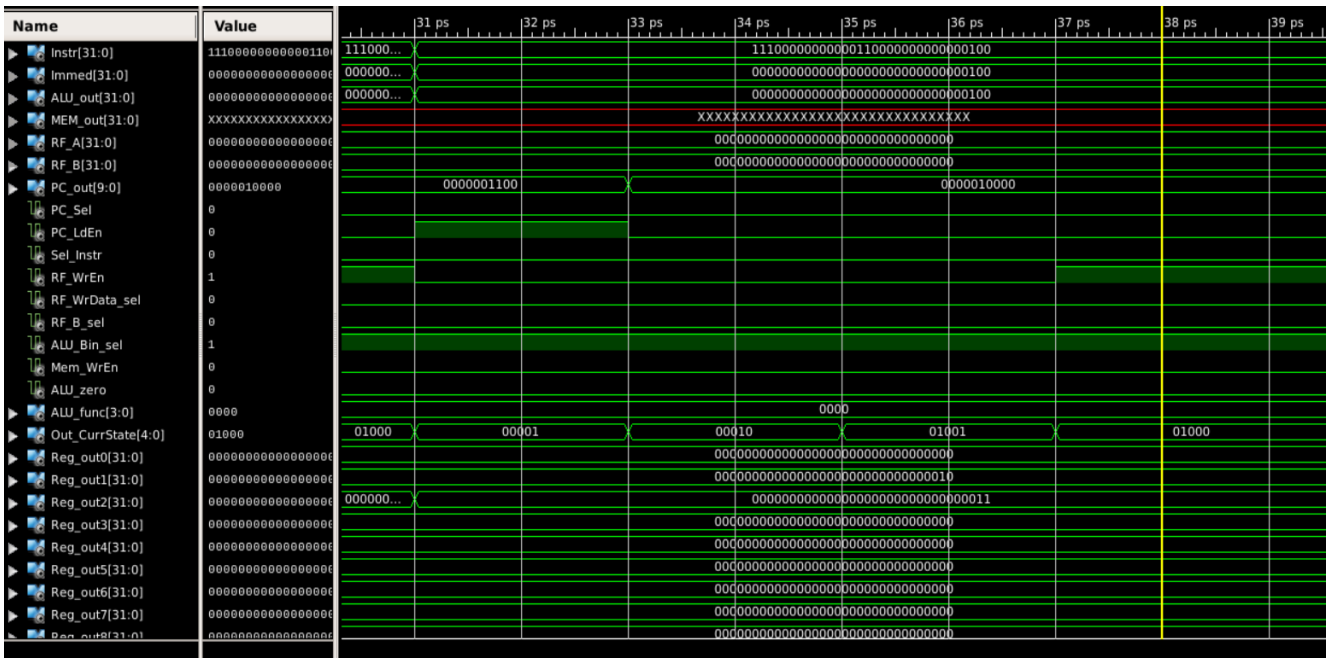
1) Η εντολή li $3, 4 => $3 = 4 //screenshot_4 στον φάκελο screenshots.

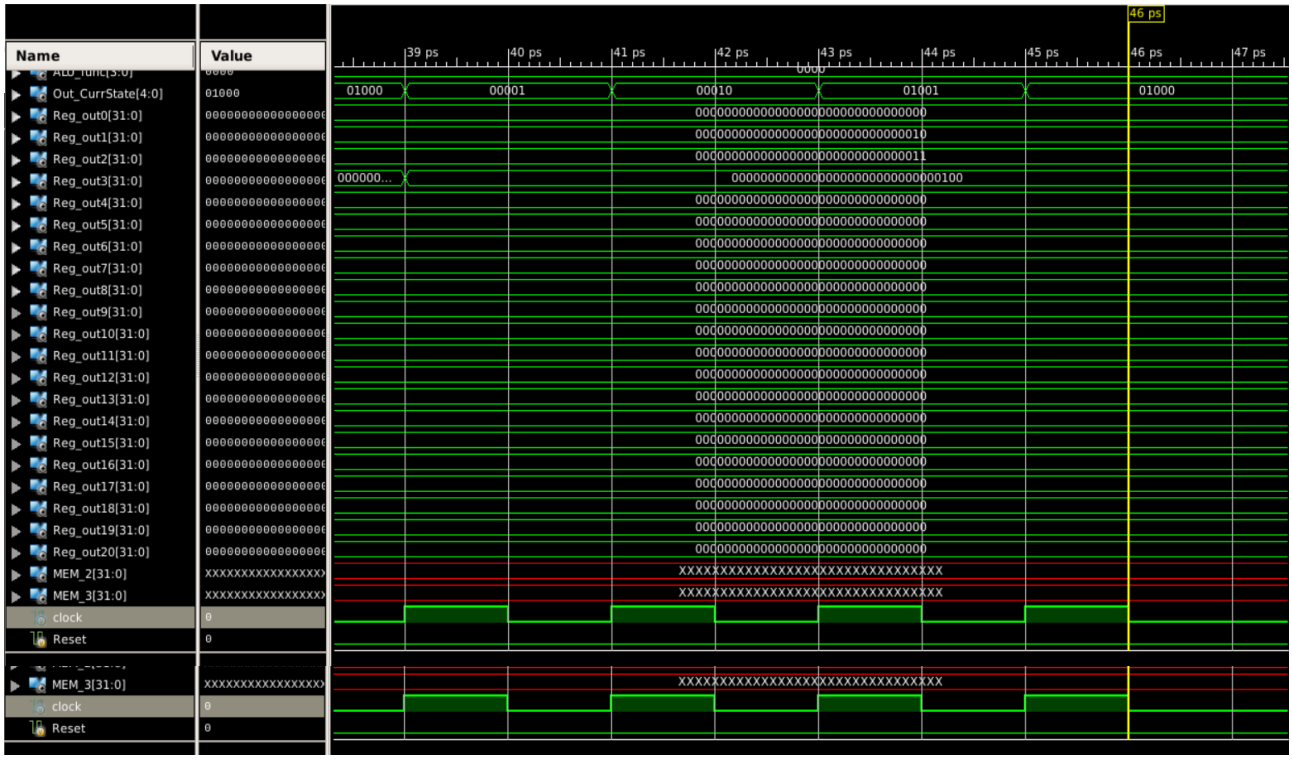
Όπως έχουμε πει ήδη για την εκτέλεση της εντολης αυτής (opcode = 111000) περνάμε από τις εξής καταστάσεις: i_fetch < decode_rr < alu_li < alu_wb.

Όντως παρατηρούμε στην παρακάτω φωτογραφία ότι για την εντολή αυτή η αλληλουχία των καταστάσεων (σήμα Out_CurrState[4:0])είναι

00001 (i_fetch) < 00010 (decode_rr)< 01001 (alu_li)< 01000 (alu_wb), και ενεργοποιούνται τα κατάλληλα σήματα όπως ορίσαμε παραπάνω στην Ανάλυση σημάτων ελέγχου FSM.

PC_LdEn = 1, PC_Sel = 0 > RF_B_Sel = 0 > ALU_Bin_Sel = 1, ALU_func = 4'b0000 (πράξη πρόσθεση (με το 0 του καταχωρητή 0)) > RF_WrData_sel = 0, RF_WrEn = 1 (ανάμεσα στα «>» είναι τα σήματα που ενεργοποιούνται σε κάθε κατάσταση))


Name
Value
Instr[31:0]
Immed[31:0]
ALU_out[31:0]
MEM_out[31:0]
RF_A[31:0]
RF_B[31:0]
PC_out[9:0]
PC_Sel
PC_LdEn
Sel_Instr
RF_WrEn
RF_WrData_sel
RF_B_sel
ALU_Bin_sel
Mem_WrEn
ALU_zero
ALU_func[3:0]
Out_CurrState[4:0]
Reg_out0[31:0]
Reg_out1[31:0]
Reg_out2[31:0]
Reg_out3[31:0]
Reg_out4[31:0]
Reg_out5[31:0]
Reg_out6[31:0]
Reg_out7[31:0]
31 ps
32 ps
33 ps
34 ps
35 ps
36 ps
37 ps
38 ps
39 ps
0000001100
0000010000
01000
00001
00010
01001



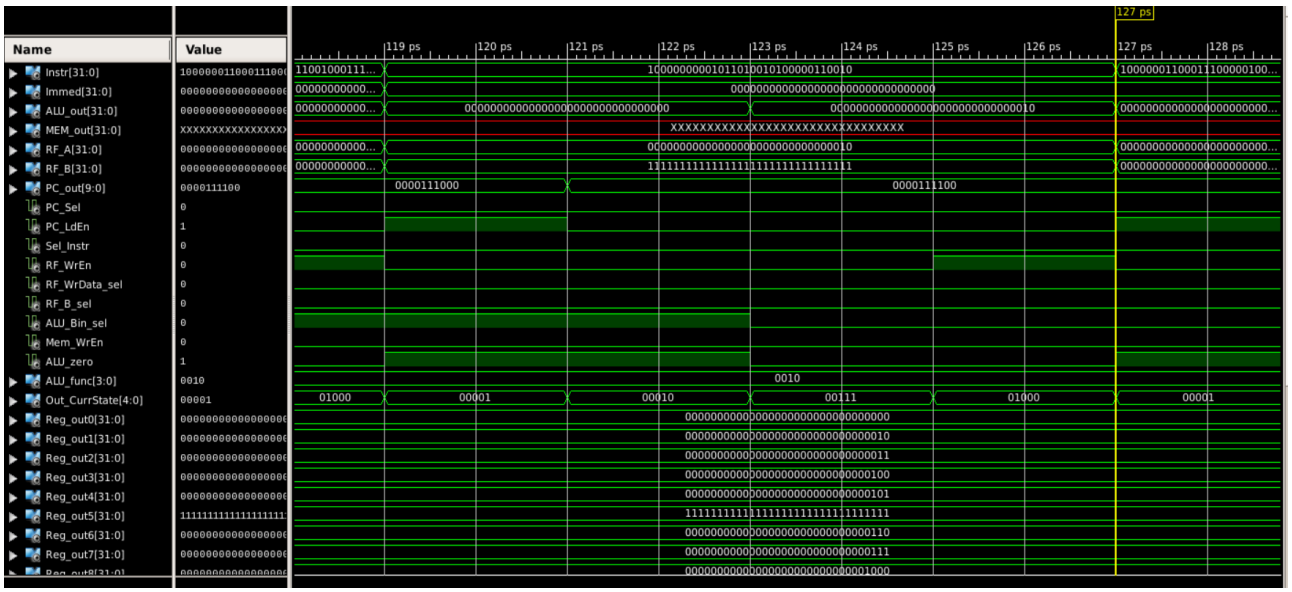
Στην συνέχεια παρατηρούμε ότι η εκτέλεση της εντολής έχει πραγματοποιηθεί με επιτυχία αφού στον καταχωρητή 3 έχει αποθηκευτεί η τιμή 4.

2) Εντολή and $13, $1, $5 => $13 = 2

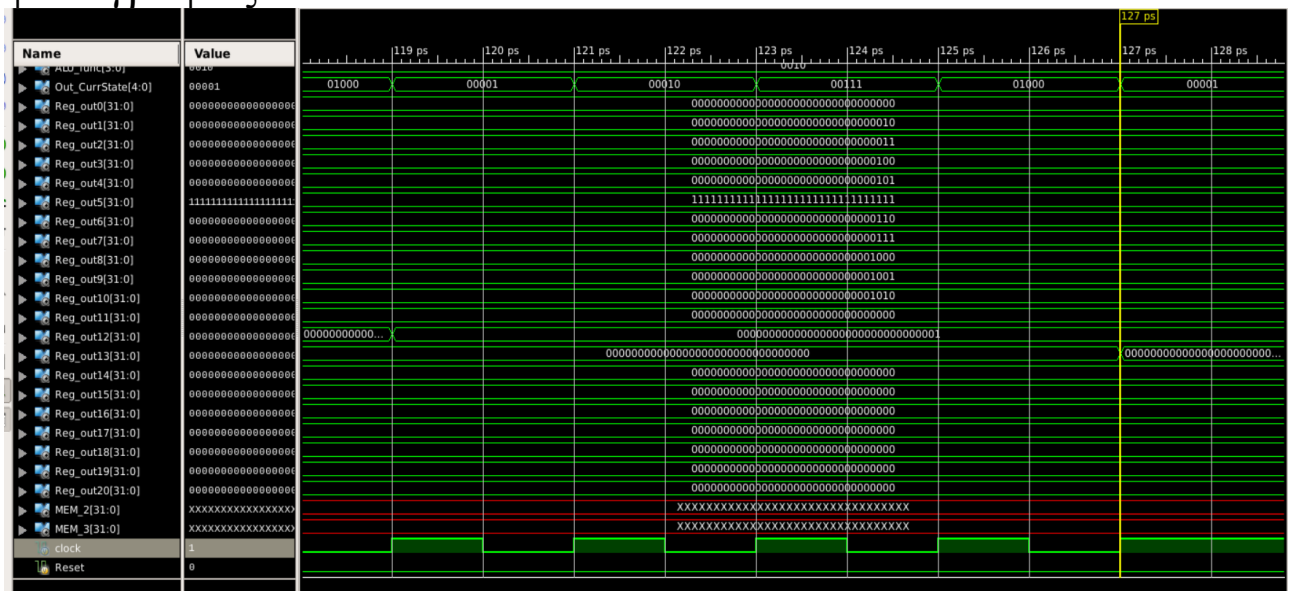
Όπως έχουμε πει ήδη για την εκτέλεση της εντολης αυτής (opcode = 100000) περνάμε από τις εξής καταστάσεις: i_fetch < decode_rr < alu_exec < alu_wb.

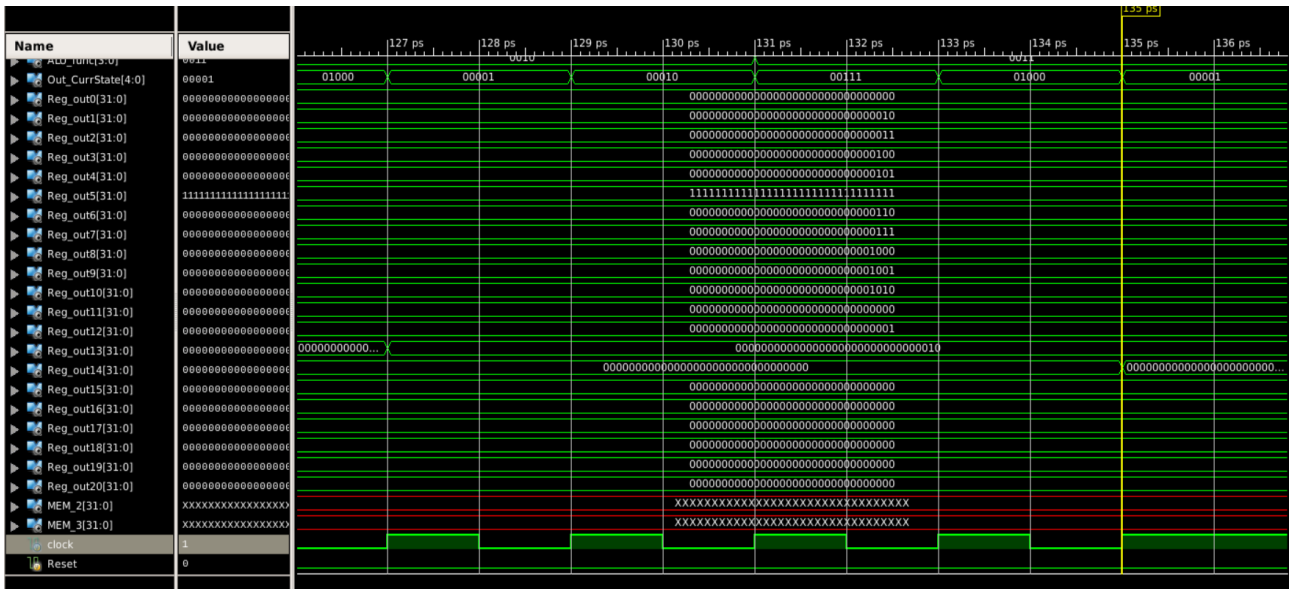
Όντως παρατηρούμε στην παρακάτω φωτογραφία ότι για την εντολή αυτή η αλληλουχία των καταστάσεων (σήμα Out_CurrState[4:0]) είναι:

00001 (i_fetch) < 00010 (decode_rr) < 00111 (alu_exec) < < 01000 (alu_wb) και ενεργοποιούνται τα κατάλληλα σήματα όπως ορίσαμε παραπάνω στην Ανάλυση σημάτων ελέγχου FSM.



Στην συνέχεια παρατηρούμε, την έξοδο ALU_out η οποία έχει γίνει 2, με τα κατάλληλα σήματα ελέγχου σε κάθε κατάσταση (PC_LdEn = 1, PC_Sel = 0 > RF_B_Sel = 0 > ALU_Bin_Sel = 0, ALU_func = [Instr[3:0]= 4'b0010 (πράξη Λογικό ΚΑΙ) > RF_WrData_sel = 0, RF_WrEn = 1) η εκτέλεση της εντολής έχει πραγματοποιηθεί με επίτυχια αφού στον καταχωρητή 13 έχει αποθηκευτεί η τιμή 2 όπως βλέπουμε στις παρακάτω φωτογραφίες.





3) Εντολή b 3 => PC = PC + 4 + (4 – 4) + 12 = 96 //screenshot 21

(opcode = 111111) περνάμε όπως έχουμε πει παραπάνω από τις εξής καταστάσεις: i_fetch < branch <nop4 < change_state< change_state2.

Παρατηρούμε στην παρακάτω φωτογραφία ότι για την εντολή αυτή η αλληλουχία των καταστάσεων (σήμα Out_CurrState[4:0]) είναι πράγματι:

00001 (i_fetch) <01110(branch)<01111(nop4) <10100 (change_state) <10101 (change_state2)

Το σήμα PC_out (PC) είναι αρχικά $(0001010000)_2 = (80)_{10}$, μετά την κατάσταση i_fetch (PC_LdEn = 1, PC_Sel = 0) το PC γίνεται PC+4 => $(0001010100)_2 = (84)_{10}$ στην επόμενη κατάσταση την branch (PC_LdEn = 1, PC_Sel = 1) μέσω της αλλαγής που κάναμε στον αθροιστή PC+4+Immed , έτσι ώστε να γίνεται PC+4+Immed – 4, το PC θα αυξηθεί πάλι κατά 4, αλλά στην συνέχεια θα γίνει πρόσθεση με το (Immed – 4) οπότε το PC με την ολοκλήρωση της εντολής θα είναι $(1100000)_2 = (96)_{10}$.

Έπειτα αφού αρχικά έγινε στην κατάσταση i_fetch, PC_LdEn = 1, στον καταχωρητή ολίσθησης που προσθέσαμε, εισέρχεται το 0 (από το αρχείο rom.data), όμως λόγω της εντολής branch (συγκεκριμένα b 3) οι

επόμενες τρεις σειρές (σειρές μηδενικών) που ακολουθούν στο αρχείο rom.data πρέπει να παραλειφθούν από την διαδικασία. Έτσι μέσω της κατάστασης nop4 (καθυστέρηση ενός κύκλου, έτσι ώστε να παραχθεί από την βαθμίδα IFSTAGE το κατάλληλο Instr, αυτό που προκύπτει για PC = 96) και των δύο κατάστασεων change_state όπου θέτουμε για 2 κύκλους ρολογιού (ένας σε κάθε change_state) Sel_Instr = 1, στην είσοδο της βαθμίδας decode εισέρχεται το Instr της εντολής beq $14, $2, 3 όπου με παρομοιο τρόπο εκτελείται και αυτήν (για την beq $14, $2, 3 βλ. screenshots 22 στο φάκελο simulations).

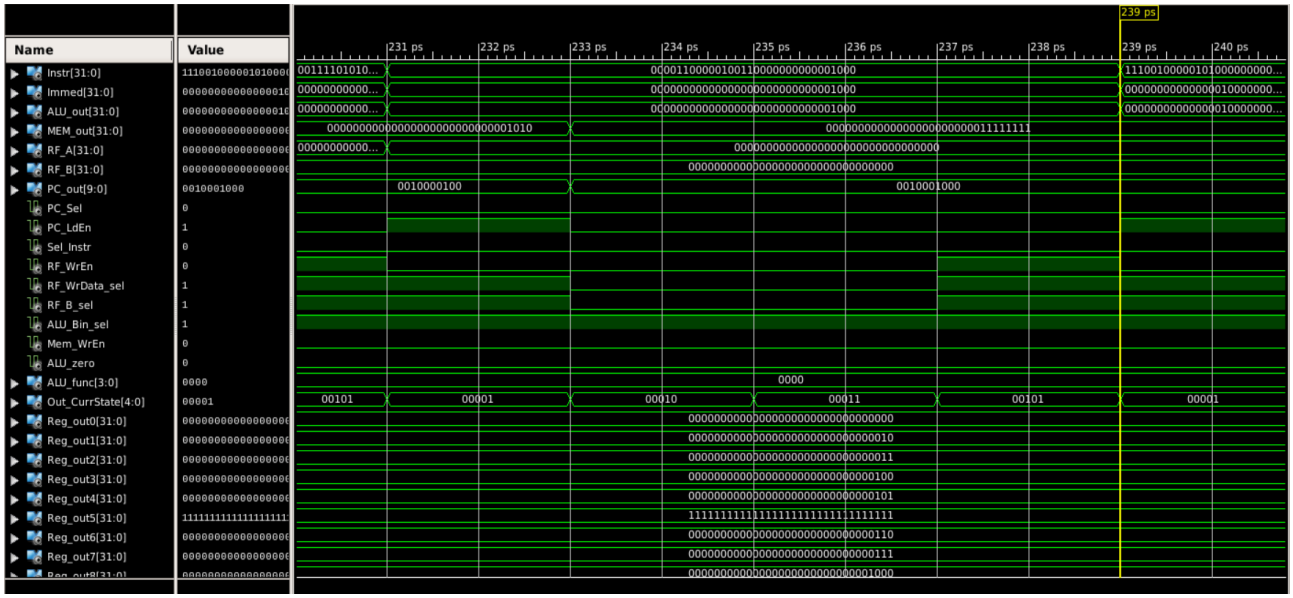
4) Εντολή lb $19, 8($0) => $19=MEM[2]=255 //screenshot 28

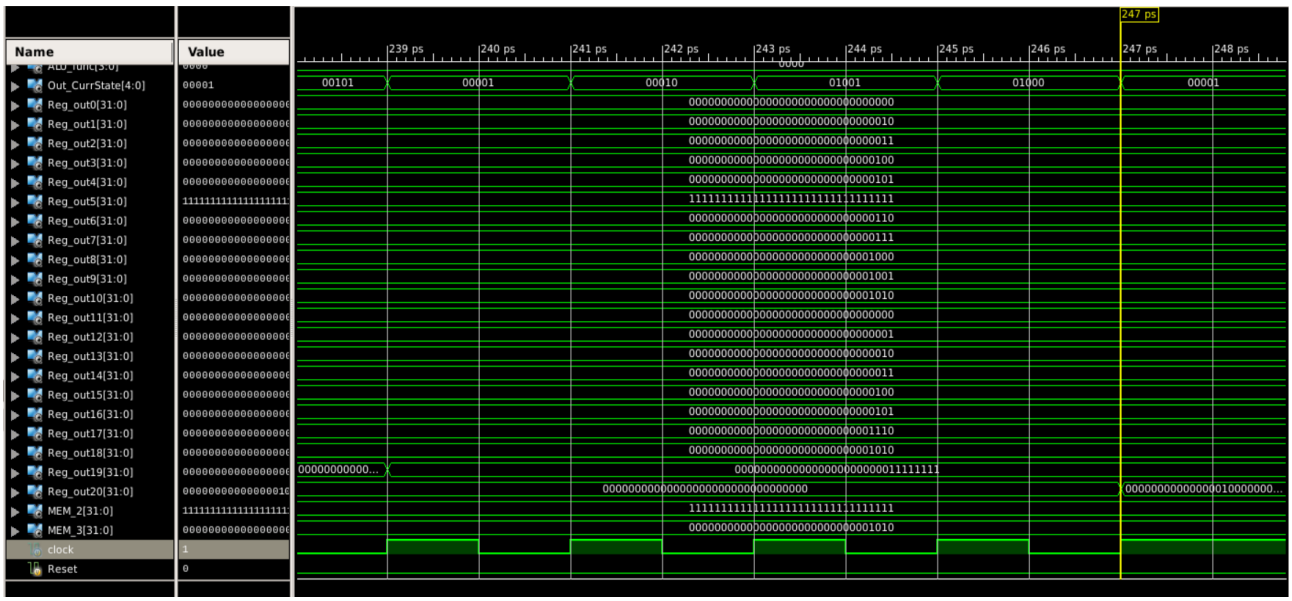
Για την εκτέλεση της εντολής αυτής (opcode = 000011) περνάμε από τις εξής καταστάσεις: i_fetch < decode_rr < mem_addr < mem_rd.

Παρατηρούμε στην παρακάτω φωτογραφία ότι για την εντολή αυτή η αλληλουχία των καταστάσεων (σήμα Out_CurrState[4:0]) είναι:

00001 (i_fetch) < 00010 (decode_rr) < 00011 (mem_addr) < 00101 (mem_rd) και ενεργοποιούνται τα κατάλληλα σήματα όπως: PC_LdEn = 1, PC_Sel = 0, RF_WrEn = 0, Mem_WrEn = 0 > RF_B_Sel = 0, RF_WrData_sel = 0 > ALU_Bin_Sel = 1, ALU_func = 4'b0000 (πράξη +) > RF_WrData_sel = 1, RF_WrEn = 1).

Για την παραπάνω εντολή η επιπρόσθετη μονάδα ελέγχου στο MEM_stage κάνει ZeroFill(31 downto 8) στο MEM[2].





Στο παραπάνω screenshot μπορούμε να δούμε ότι η εκτέλεση όλων των εντολών έχει πραγματοποιηθεί με επίτυχια αφού στον καταχωρητές και στις θέσεις μνήμης έχουν αποθηκευτεί τα αναμενόμενα.

Τέλος προσθέσαμε στο αρχείο rom.data την γραμμή 11100100000101000000000000000001, για να ελέγξουμε και την εντολή lui. Σύμφωνα με την προσομοίωση (screenshot 29) η εντολή lui πράγματι εκτελείται ορθά, αφού αποθηκεύεται στον καταχωρητή 20 του RF το 10000000000000000 (= 65536 στο δεκαδικό).

Βιβλιογραφία-Πηγές

1)Διαφάνειες Μαθήματος Ψηφιακά Συστήματα HW-1, Βασίλειος Παυλίδης, Νικόλαος Ταμπουρατζής.

2) Βασική FSM Ελέγχου του Επεξεργαστή, Σειρά Ασκήσεων 11, HY-225: Οργάνωση Υπολογιστών, Τμ. Επ. Υπολογιστών Πανεπιστήμιο Κρήτης, S. Lyberis, G. Sapountzis, and M. Katevenis