Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Ηλεκτρονικής και Υπολογιστών

ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ HARDWARE 1

ΑΝΑΦΟΡΑ ΕΡΓΑΣΤΗΡΙΑΚΩΝ ΑΣΚΗΣΕΩΝ

Χειμερινό εξάμηνο 2023/24

Δεϊρμεντζόγλου Ιωάννης
A.E.M.: 10015
Email: deirmentz@ece.auth.gr

Περιεχόμενα

ΑΣΚΗΣΗ 1

Για την 1η άσκηση ζητούμενο ήταν η υλοποίηση μιας αριθμητική/λογική μονάδα (Arithmetic Logic Unit (ALU)) η οποία είναι υπεύθυνη για την εκτέλεση αριθμητικών λειτουργιών, όπως πρόσθεση και αφαίρεση, καθώς και λογικών λειτουργιών. Οι πράξεις που υποστηρίζει η ALU είναι :Logic AND , Logic OR, Πρόσθεση ,Αφαίρεση , Μικρότερο από , Λογική ολίσθηση δεξιά κατά op2 bits, Λογική ολίσθηση αριστερά κατά op2 bits, Αριθμητική ολίσθηση δεξιά κατά op2 bits και Λογική XOR. Η υλοποίηση του κυκλώματος βρίσκεται στο αρχείο alu.v.

Περιγραφή υλοποίησης

Για την υλοποίηση της άσκησης δημιουργήθηκε το module με τις εισόδους και τις εξόδους όπως ζητούνται από την άσκηση. Για τις εισόδους χρησιμοποιήθηκε τύπος wire και reg για τις εξόδους όπως και σε όλες τις υπόλοιπες ασκήσεις . Στην συνέχεια, ορίστηκαν οι παράμετροι για το σήμα **alu_op**, το οποίο καθορίζει και την πράξη που θα εκτελέσει η ALU . Χρησιμοποιήθηκε ένα always block και μια case (**Πολυπλέκτης ALU**) εντολή για τον υπολογισμό της πράξης που ορίστηκε από το alu_op και ανάλογα με την τιμή του εκτελείται και η αντίστοιχη πράξη. Στην λίστα ευαισθησίας του block μπήκαν τα op1,op2,alu_op. Τέλος , για τον υπολογισμό του zero χρησιμοποιήθηκε ένα always block έχοντας στην λίστα ευαισθησίας το σήμα result και ανάλογα ορίζεται με βάση το εάν το αποτέλεσμα είναι ίσο με μηδέν ή όχι.

ΑΣΚΗΣΗ 2

Για την 2η άσκηση ζητούμενο ήταν σε πρώτη φάση η σχεδίαση ενός κυκλώματος αριθμομηχανής που χρησιμοποιεί την ALU που δημιουργήθηκε στην άσκηση 1 .

Περιγραφή υλοποίησης

Calculator

Για την υλοποίηση της άσκησης δημιουργήθηκε το module με τις εισόδους και τις εξόδους όπως ζητούνται από την άσκηση. Στην συνέχεια, ορίστηκαν κάποια nets(wire) που θα συνδεθούν με την ALU μέσα στο module calc και με τύπο reg ο 16 bit συσσωρευτής accumulator για να κρατά την τρέχουσα τιμή της αριθμομηχανής. Στην συνέχεια, ανατίθεται (assign) στο op1_calc ένα σήμα 32-bit που είναι μια έκδοση με επέκταση πρόσημου του accumulator 16-bit . Το σήμα op1_calc είναι αυτό που θα συνδεθεί στο port op1 της alu. Με παρόμοιο τρόπο δημιουργείται και ένα σήμα 32-bit που είναι μια έκδοση με επέκταση πρόσημου των εισόδων του διακόπτη (16-bit) και ανατίθεται στο op2_calc που θα συνδεθεί στο port op2 της alu. Έπειτα , με ένα always block στο sensitivity list του οποίου τοποθετείται η θετική ακμή του ρολογιού (posedge clk) ενημερώνεται ο accumulator . Με μια if else συνθήκη εάν το σήμα εισόδου btnd είναι στο λογικό 1 τότε στον συσσωρευτή αποθηκεύεται το αποτέλεσμα της ALU (result_calc που συνδέεται στο output result της ALU) . Εάν το btnu είναι 1 (πάτημα) τότε , ο accumulator τίθεται στο 0 , όπως ζητείται και από την άσκηση. Με βάσει τα wires εισόδου btnl , btnc , btnr και μέσω του module decoder (που έχει ως είσοδο τα παραπάνω σήματα) προκύπτει το alu_op_calc το οποίο συνδέεται στο port alu_op της alu. Ο decoder υλοποιείται στο αρχείο decoder.v με structural Verilog όπως έχει διδαχθεί στο μάθημα. Έχει εισόδους (wire) τα btnrD, btnlD, btncD και έξοδο το 4bito σήμα alu_opD (υλοποιείται ένα συνδυαστικό κύκλωμα για κάθε bit όπως φαίνεται και από τα σχήματα στην εκφώνηση της εργασίας) . Τέλος , με ένα always block το σήμα εξόδου led παίρνει την τιμή που υπάρχει στον accumulator σε κάθε θετική ακμή του ρολογιού.

Testbench

Το 2ο ζητούμενο της άσκησης ήταν η δημιουργία ενός testbench για τον έλεγχο της σωστής λειτουργίας του συνόλου της αριθμομηχανής που περιλαμβάνεται στα αρχεία ALU.v , calc.v , decoder.v .

Περιγραφή υλοποίησης testbench

Η κλίμακα χρόνου τίθεται 1ns/1ps και δημιουργείται το module calc_tb. Αρχικά, δημιουργούνται nets και μεταβλητές που θα συνδεθούν όταν γίνει instantiate το module calc και τα πλήκτρα εισόδου αρχικοποιουνται στο 0 με ένα initial block και δημιουργείται και το ρολόι με όπως διδάχθηκε στο μάθημα με ένα initial και ένα always block. Επίσης , με initial block αρχικοποιείται αρχικά στο 1 το btnu (που ουσιαστικά είναι το reset της αριθμομηχανής) και μετα από καθυστέρηση 10 time units τοποθετείται στο 0. Αντίστοιχα

, με ένα initial block το πλήκτρο btnu τίθεται στο 0 και μετα από 10 time units στο 1 ώστε να αποθηκεύεται το αποτέλεσμα της ALU στον accumulator. Στην συνέχεια , καλείται το module calc οπού είσοδοι και εξόδιο συνδέονται σε nets και μεταβλητές με παρόμοια ονομασία για να αναγνωρίζονται εύκολα. Τέλος, με ένα initial block εκτελούνται οι πράξεις με την σειρά που ορίζεται από τον πίνακα της εκφώνησης, εκτυπώνονται τα αποτελέσματα των πράξεων σε σχέση με τα αναμενόμενα για επιβεβαίωση της ορθής λειτουργίας και παράγονται τα αντίστοιχα διαγράμματα της προσομοίωσης. Τα αποτελέσματα λαμβάνονται παρακολουθώντας την έξοδο του calculator **led**.

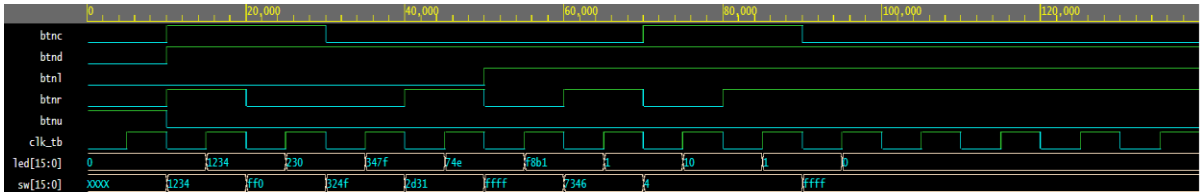
Αποτελέσματα :

Παρακατω παρατίθενται τα αποτελέσματα που προέκυψαν :

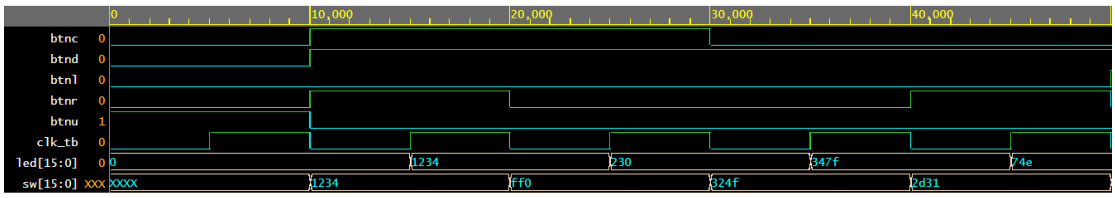
```
Test Case 0: Expected Result 0x0, Actual Result 0000
Test Case 1: Expected Result 0x1234, Actual Result 1234
Test Case 2: Expected Result 0x0230, Actual Result 0230
Test Case 3: Expected Result 0X347F, Actual Result 347f
Test Case 4: Expected Result 0x074e, Actual Result 074e
Test Case 5: Expected Result 0xf8b1, Actual Result f8b1
Test Case 6: Expected Result 0x0001, Actual Result 0001
Test Case 7: Expected Result 0x0010, Actual Result 0010
Test Case 8: Expected Result 0x0001, Actual Result 0001
Test Case 9: Expected Result 0x0000, Actual Result 0000
```

Σχήμα 2.1 : Αποτελέσματα πράξεων με την αριθμομηχανή

Όπως , παρατηρείται τα αποτελέσματα είναι ιδιά με τα αναμενόμενα που παρέχονται από τον πίνακα και από τις κυματομορφες προσομοίωσης που παρατίθενται στην συνέχεια η ALU φαίνεται να λειτουργεί σωστά.



Σχήμα 2.2 : Κυματομορφές προσομοίωσης



Σχήμα 2.3 : Κυματομορφές προσομοίωσης

ΑΣΚΗΣΗ 3

Για την 3η άσκηση ζητούμενο ήταν η δημιουργία ενός αρχείου καταχωτητών 32x32 bit είναι άλλο ένα σημαντικό στοιχείο κάθε συστήματος επεξεργαστή. Το
αρχείο καταχωρητών αποθηκεύει τις τιμές των καταχωρητών που χρησιμοποιούνται από τον επεξεργαστή RISC-V. Η υλοποίηση της άσκησης βρίσκεται στο αρχείο **regfile. v**.

Περιγραφή υλοποίησης

Για την υλοποίηση της άσκησης δημιουργήθηκε το module με τις εισόδους και τις εξόδους όπως ζητούνται από την άσκηση. Αρχικά , δηλώθηκαν οι 32 καταχωρητες και στην συνέχεια αρχικοποιηθηκαν με μηδενικά μέσω ενός initial block και μιας for όπως ζητούνταν. Στην συνέχεια , με ένα always block στο sensitivity list του οποίου τοποθετείται την θετική ακμή του ρολογιού (posedge clk) διαβάζονται οι 2 τιμές των καταχωρητών των οποίων οι διευθύνσεις αποτελούν είσοδο στο module regfile. Επίσης , εάν το write είναι 1 και η διεύθυνση για την εγγραφή (net writeReg) διάφορη του 0 ,τότε τα δεδομένα από το writeData εγγράφονται στην αντίστοιχη διεύθυνση δίνεται ως είσοδος. Για την περιπτωση οπού η διεύθυνση εγγραφής είναι ίδια με κάποια από τις διευθύνσεις ανάγνωσης με μια if για κάθε περιπτωση η εκάστοτε έξοδος (readData1 αν το writeReg ταυτίζεται με το readReg1 , readData2 αν το writeReg ταυτίζεται με το readReg2) ανανεώνεται μετα την εγγραφή του writeData στην συγκεκριμένη διεύθυνση.

ΑΣΚΗΣΗ 4

Για την 4η άσκηση ζητούμενο ήταν η υλοποίηση του τμήματος της διαδρομής δεδομένων το οποίο αποτελείται τις εσωτερικές λειτουργικές μονάδες, τους καταχωρητες και τους πολυπλέκτες που χρησιμοποιούνται για την υλοποίηση μεμονωμένων εντολών. Η υλοποίηση της άσκησης βρίσκεται στο αρχείο datapath.v .

Περιγραφή υλοποίησης

Για την υλοποίηση της άσκησης δημιουργήθηκε το module με τις εισόδους και τις εξόδους όπως ζητούνται από την άσκηση. Αρχικά, δηλωθήκαν οι 3 παράμετροι που θα χρησιμοποιηθούν στο module : INITIAL_PC που χρησιμοποιείται για τον καθορισμό της αρχικής τιμής του PC όταν το datapath επαναφέρεται στην αρχική κατάσταση με το σύγχρονο σήμα rst , PC _UPDATE=4 που προστίθεται στον PC όταν το πρόγραμμα προχωρά στην επόμενη εντολή στην μνήμη, SIGN_EXTEND_VALUE = 20 που χρησιμοποιείται για την δημιουργία των εντολών Immediate Generation (επέκταση πρόσημου) από το σήμα instr. Έπειτα, δηλώνονται τα reg 5bit readReg1_data , readReg2_data, writeReg_data που είναι οι μεταβλητές που συνδέονται στα ports εισόδου του module regfile που καλείται μέσα στο datapath. Αρχικοποιουνται μέσω ενός initial block και σε κάθε θετική ακμή του ρολογιού ανανεώνονται όπως φαίνεται και από

το σχήμα τις εκφώνησης : υπολογίζονται ως εξής **readReg1_data = instr[19:15] , readReg2_data = instr[24:20], writeReg_data = instr[11:7]** . Στην συνέχεια , με παρόμοιο τρόπο αρχικοποιουνται και ανανεώνονται σε κάθε θετική ακμή του clk (initial and always block) δημιουργούνται τα σήματα:

- **imm={{SIGN_EXTEND_VALUE{instr[31]}}, instr[31:20]}** ,
- **immS = {{SIGN_EXTEND_VALUE {instr [31]}}, instr [31:25], instr [11:7]}** και
- **immB = {{SIGN_EXTEND_VALUE {instr [31]}}, instr [7], instr [31:25], instr [11:8], 1'b0}**

Τα σήματα προκύπτουν με βάσει το εγχειρίδιο που δόθηκε για την εργασία και συγκεκριμένα την σελίδα 148 . Σημειώνεται ότι στο σήμα immB που προκύπτει από εντολές τύπου B εφαρμόζεται αριστερή ολίσθηση κατά 1 όπως ζητείται από την άσκηση για τον υπολογισμό του branch offset για την πραγματοποίηση διακλάδωσης.

Με χρήση ενός always block για κάθε θετική ακμή του ρολογιού (posedge clk) ή αλλαγή σήμα rst φανερώνεται το σήμα Program Counter. Μέσα στο block υλοποιείται μια if-else if εντολή συνθηκών στην οποία , εάν το rst είναι στο λογικό 1 ο PC επανέρχεται στην αρχική του τιμή (INITIAL_PC). Εάν πάλι το σήμα loadPC είναι στο λογικό 1 , τότε ανάλογα με το σήμα PCScr υπολογίζεται η νέα τιμή του PC όπως περιγράφεται στην εκφώνηση της 4^{ης} άσκησης . Έπειτα, καλείται το module regfile που υλοποιήθηκε στην 3^{η} άσκηση (αρχείο 32x32) καταχωρητών και τα 2 ports εξόδου συνδέονται στις μεταβλητές outReadData1, outReadData2 . Με κάθε θετική ακμή ρολογιού (always block) και με εντολές if else εάν το σήμα ALUScr είναι 1 το op2_ALU τίθεται στην τιμή outReadData2 που θα συνδεθεί στο port op2 του module alu που καλείται παρακάτω. Στην άλλη περιπτωση το op2_alu γίνεται ίσο με το σήμα ImmI που αναφέρθηκε παραπάνω. Στην συνέχεια , καλείται το module της alu που υλοποιήθηκε στην άσκηση 1 και συνδέονται τα ports εισόδου op1 και op2 με outReadData1 από το αρχείο των καταχωρητών και το op2_alu που προέκυψε με βάσει την λογική που περιγραφικέ πριν. Το ALUCtrl συνδέεται με το port εισόδου alu_op και το αποτέλεσμα της alu, aluResult σε κάθε θετική ακμή του ρολογιού το dAddress παίρνει την τιμή aluResult και το dWriteData = outReadData2. Τέλος , για κάθε θετική ακμή του ρολογιού εγγράφεται στους καταχωρητες είτε το αποτέλεσμα της alu, είτε το αποτέλεσμα ανάγνωσης μνήμης (if else)(Πολυπλέκτης) με βάση το σήμα MemtoReg. Τα δεδομένα που γράφονται στους καταχωρητές πρέπει επίσης να συνδεθούν στην έξοδο **WriteBackData** του κυκλώματος.

ΑΣΚΗΣΗ 5

Για την 5η άσκηση ζητούμενο ήταν η δημιουργία μιας μονάδας ελέγχου πολλαπλών κύκλων , η οποία εκτελεί κάθε instruction σε 5 κύκλους ρολογιού. Στην συνέχεια , ζητούνταν η υλοποίηση ενός testbench που θα υποστηρίζει εντολές Register – Register, ALU Immediate, προσπέλασης μνήμης (LW , SW) και διακλάδωσης. Τα αρχεία με την υλοποίηση του κώδικα είναι τα multicycle.v και multicycle _tb.v

Περιγραφή υλοποίησης multicycle

Για την υλοποίηση της άσκησης δημιουργήθηκε το module με τις εισόδους και τις εξόδους όπως ζητούνται από την άσκηση. Αρχικά, δηλωθήκαν οι παράμετροι που θα χρησιμοποιηθούν στο module και παρατίθενται στους παρακάτω πίνακες. Πιο συγκεκριμένα , τα 7 LSB της εντολής (είσοδος instr στο module) αποτελούν το opcode όπως φαίνεται και από το εγχειρίδιο και το funct3 προκύπτει από τα 12,13,14 bit της εντολής instr . Το funct7 προκύπτει από τα 7 MSB του instr. Επίσης, πρέπει να δηλωθούν και οι παράμετροι για το σήμα της ALU πράξης, ALUCtrl. Ακολουθούν οι πίνακες :

ALU Operation	Παράμετρος
Logic AND	0000
Logic OR	0001
Addition	0010
Subtraction	0110
Less Than	0111
Logic Left Shift	1000
Logic Right Shift	1001
Arithmetic Right Shift	1010
Logic XOR	1101

Πίνακας 5.1 : Παράμετροι σήματος ALUCtrl

Opcode Values	Instruction [0:6]
I Type Immediate	0010011
I Type Load	0000011
S Type	0110011
R Type	0100011
SB Type	1100011

Πίνακας 5.2 : Παράμετροι για τα 7 LSB bit της εντολής (σήμα opcode)

Operation	Funct3
BEQ	000
BNE	001
ADD, SUB	000
LESS	010
XOR	100
OR	110
AND	111
Logic and Arithmetic RSHFT	101
Logic Left Shift	001

Πίνακας 5.2 : Παράμετροι για τα 12,13,14 bit της εντολής (σήμα funct3)

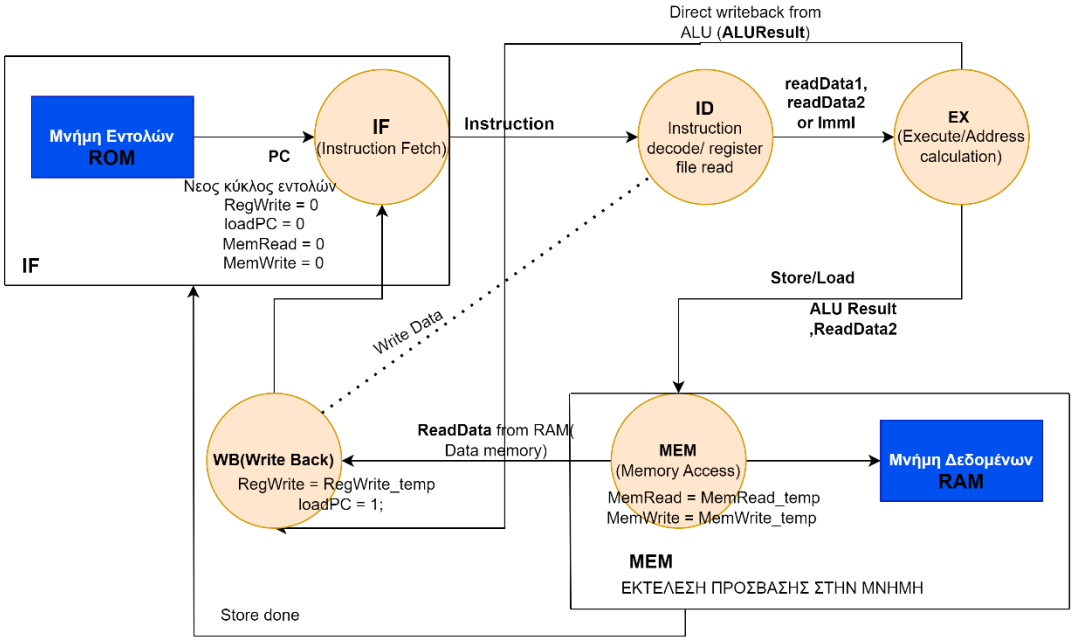
Οι παραπάνω τιμές ανατίθενται στα αντίστοιχα σήματα μέσω ενός initial και ενός always() block που εκτελείται για κάθε θετική ακμή του ρολογιού ή για αλλαγή της εντολής instr. Επίσης, το σήμα ALUCtrl παίρνει τιμές μέσω ενός always και μια case για το σήμα opcode. Ανάλογα, με το value του opcode καθορίζεται ο τύπος της εντολής και στην συνέχεια με εντολή if , else if,else για την τιμή του funct3 επιλέγεται και η τιμή για το ALUCtrl και συνεπώς η πράξη που εκτελεί η ALU μέσα στο datapath.

Ένα από τα ζητούμενα για την δημιουργία της μονάδας ελέγχου ήταν η δημιουργία ενός FSM 5 καταστάσεων όπως περιγράφεται από την άσκηση. Για την υλοποίηση του σε Verilog ακολουθήθηκε η διαδικασία όπως είχε παρουσιαστεί στις διαφάνειες του μαθήματος. Υλοποιήθηκε με 2 block always ένα για το NEXT_STATE_LOGIC και ένα για το CURRENT_STATE_LOGIC μέσα στο οποίο γίνονται οι μεταβάσεις από την μια κατάσταση στην άλλη όπως φαίνεται και από το παρακάτω σχήμα : Τα 5 στάδια φαίνονται παρακάτω και διατρέχονται με σειρά : IF->ID->EX->MEM->WB κατά την εκτέλεση μιας εντολής σε συνολικά 5 κύκλους ρολογιού (επομένως ένας κύκλος για κάθε κατάσταση). Σημειώνεται ότι στην αρχή του module καθορίζονται οι τιμές των παραμέτρων για κάθε κατάσταση ως εξής :

State	Value
IF	000
ID	001
EX	010
MEM	011
WB	100
ERROR	101

Πίνακας 5.3 : Καταστάσεις FSM

Η κατάσταση ERROR χρησιμοποιείται για την ανίχνευση σφάλματος και για διαπιστωθεί αν αλλάζουν σωστά οι καταστάσεις.
Ακολούθως παρατίθεται το σχηματικό διάγραμμα



Σχήμα 5.1 : Σχηματικό διάγραμμα FSM

Για τα σήματα MemRead, MemWrite και RegWrite ζητούνταν να τίθενται μόνο κατά την διάρκεια του σταδίου MEM και WB του FSM αντίστοιχα. Για αυτόν τον λόγο δημιουργήθηκαν τα σήματα MemRead_temp, MemWrite_temp και RegWrite_temp και με ένα always block που εκτελείται για θετικές ακμές του ρολογιού στο οποίο τίθενται τιμές όπως ζητείται από την εκφώνηση. Στην συνέχεια , κατά την εκτέλεση του σταδίου Memory Access και Write Back στο
CURRENT_STATE_LOGIC που αναφέρθηκε παραπάνω, οι τιμές των temp σημάτων τίθενται στα σήματα ελέγχου. Αυτή η διαδικασία δεν ακολουθείται για τα σήματα PCScr και MemtoReg καθώς δεν έχει σημασία να τίθενται σε συγκεκριμένο στάδιο -κατάσταση του FSM και απλά χρησιμοποιείται ένα με ένα always block που εκτελείται για θετικές ακμές του ρολογιού στο οποίο το PCScr τίθεται στο 1 εάν opcode == SB_TYPE_OPCODE && funct3 == FUNCT3_BEQ && Zero == 1 και το MemtoReg εάν το opcode είναι διαφορετικό από SB_TYPE_OPCODE και S_TYPE_OPCODE . Τέλος , το σήμα loadPC τίθεται σε '1' κατά τη διάρκεια της κατάστασης '**WriteBack**' κάθε εντολής.

Το datapath συνδέεται στο module multicycle όπως περιγράφεται στην άσκηση. Η παρακάτω εικόνα δείχνει την σύνδεση με τα ports όπως υποδεικνύεται από την εκφώνηση της άσκησης.

```
// Calling datapath module
datapath myDataPath (.clk(clk),
                     .rst(rst),
                     .instr(instr),
                     .PCSrc(PCSrc),
                     .ALUSrc(ALUSrc),
                     .RegWrite(RegWrite),
                     .MemtoReg(MemtoReg),
                     .ALUCtrl(ALUCtrl),
                     .loadPC(loadPC),
                     .dReadData(dReadData),
                     .PC(PCw),
                     .Zero(Zerow),
                     .dAddress(dAddressw),
                     .dWriteData(dWriteDataw),
                     .WriteBackData(WriteBackDataw));
```

Σχήμα 5.2 : Σύνδεση datapath στο multicycle

Testbench

Περιγραφή υλοποίησης testbench

Η κλίμακα χρόνου τίθεται 1ns/1ps και δημιουργείται το module calc_tb. Αρχικά, δημιουργούνται nets και μεταβλητές που θα συνδεθούν όταν γίνει instantiate τα modules INSTRUCTION_MEMORY, DATA_MEMORY και multicycle. Αρχικά, δημιουργείται το ρολόι και το σήμα reset όπως διδάχθηκε στο μάθημα με ένα initial και ένα always block. Στην συνέχεια , γίνονται instantiate τα modules που προαναφέρθηκαν και συνδέονται τα ports με παρόμοιας ονομασίας σήματα . Σημειώνεται ότι , αν υπάρχει mismatch ανάμεσα στα bits κάποιας εισόδου μεταξύ πολλαπλών modules, τότε μπορείς να προστίθενται τα 9 LSB των σημάτων ως είσοδο αντί για όλα τα 32 bits. Αυτό συμβαίνει για τα 9 LSB του PC οπού μπαίνουν ως είσοδος στην διεύθυνση της ROM και του port dAddress που μπαίνει ως διεύθυνση στην είσοδο του module
DATA_Memory. Ακολουθούν οι εικόνες από τα instantiation των modules και τα αποτελέσματα των προσομοιώσεων. Τα σήματα addrRAM και addrROM αντιστοιχούν στα 9 LSB του PC και dAddress αντίστοιχα

```
multicycle myMulticycle (
  .clk(clk),
  .rst(rst),
  .instr(Instruction),
  .dReadData(dReadData),

  .PC(PC),
  .dAddress(dAddress),
  .dWriteData(dWriteData),
  .MemRead(MemRead),
  .MemWrite(MemWrite),
  .WriteBackData(WriteBackData)
);
```

Εικόνα 5.1: Multicycle in testbench

```
// Instantiate RAM Memory
DATA_MEMORY RAM_Memory (.clk(clk),
                        .we(weRAM),
                        .addr(addrRAM),
                        .din(dWriteData),
                        .dout(dReadDataRAM)) ;
```

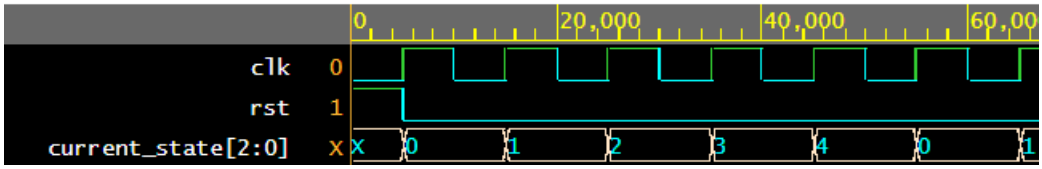
Εικόνα 5.2: RAM in testbench

```
// Instantiate ROM Memory
INSTRUCTION_MEMORY ROM_Memory (.clk(clk),
                               .addr(addrROM),
                               .dout(Instruction))
```

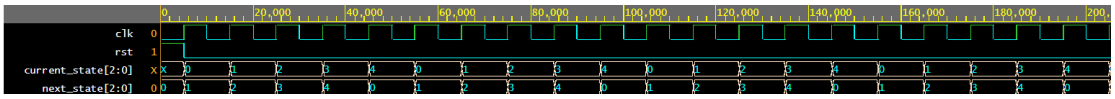
Εικόνα 5.3: ROM in testbench

Αποτελέσματα

Αρχικά , παρουσιάζονται τα states του FSM σε σχέση με το clock.



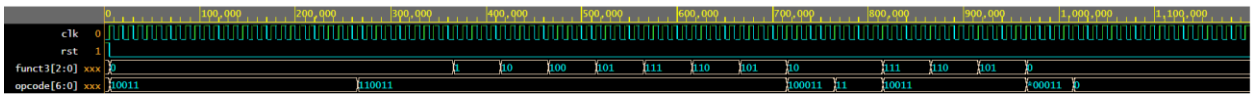
Σχήμα 5.4 : States FSM for current state



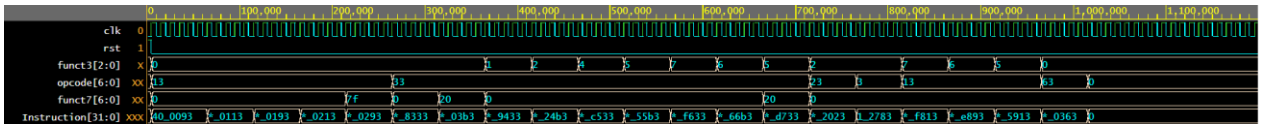
Σχήμα 5.5 : States FSM for current state and next state

Παρατηρείται πως η καταστάσεις διαρκούν 5 κύκλους ρολογιού όπως ζητείται από την άσκηση.

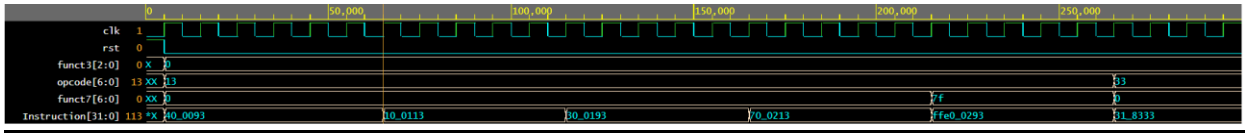
Στην συνέχεια παρουσιάζονται τα σήματα opcode, funct3 όπως υπολογίζονται από την μνήμη ROM και το αρχείο rom_bytes.data που περιέχει 8bit τιμές και με την διεύθυνση addr ως είσοδο σχηματίζει τα 32 bit της εντολής instruction με την σειρά : Στην αρχή οι εντολές τύπου I-Immediate (opcode=7'h13), στην συνέχεια οι εντολές τύπου Register/Register-RType (opcode=7'h33), μετα η εντολή LW (S-Type)(opcode=7'h23), ακολουθεί η εντολή LW (opcode=7'h3), οι υπόλοιπες εντολές I-Immediate (opcode=7'h13) και τέλος οι εντολές SB-Type (opcode=7'h63). Παρακατω παρουσιάζονται οι αντίστοιχες προσομοιώσεις.



Σχήμα 5.6 : Σήμα opcode και funct3 για επιβεβαίωση ορθής προσπέλασης

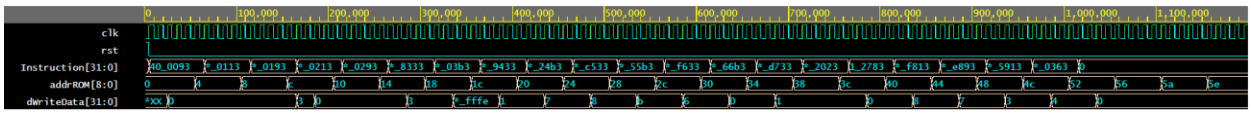


Σχήμα 5.7 : Εντολή Instruction που προκύπτει από την έξοδο ROM

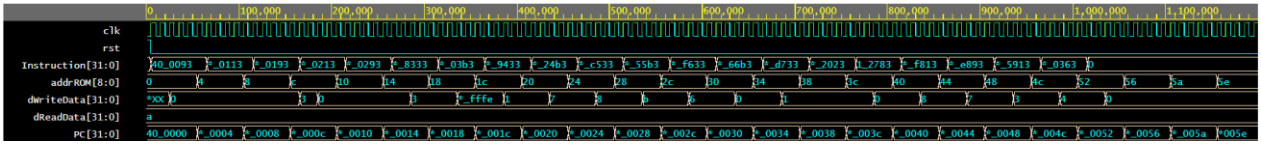


Σχήμα 5.8 : Σήμα opcode και funct7 και Εντολή Instruction που προκύπτει από την έξοδο ROM

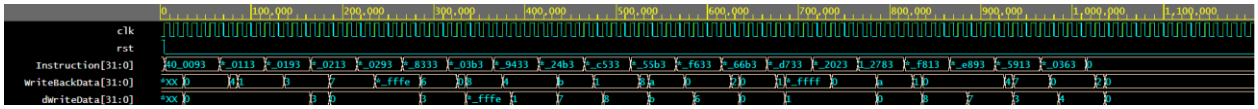
Τέλος, παρουσιάζονται παρακάτω οι κυματομορφες προσομοίωσης



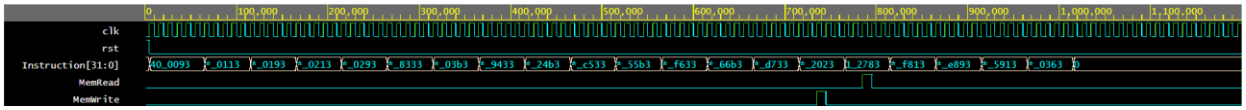
Σχήμα 5.9 : Κυματομορφές σημάτων



Σχήμα 5.10 : Κυματομορφές σημάτων



Σχήμα 5.11 : Κυματομορφές σημάτων WriteBackData,dWriteData



Σχήμα 5.12 : Κυματομορφές σημάτων Memwrite,Memread

Για το αρχείο top_proc_tb.v, δηλαδή το testbench, ισχύουν τα εξής:

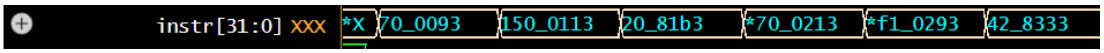
- Αρχικά, πραγματοποιούνται τα instantiations τόσο του module top_proc όσο και των DATA_MEMORY (ram.v) και INSTRUCTION_MEMORY (rom.v).
- Ακολουθεί το clock generation block, στο οποίο δημιουργείται ρολόι με περίοδο 20 time units.
- Στη συνέχεια, το σήμα rst τίθεται ίσο με 1, δηλαδή γίνεται reset του επεξεργαστή στην αρχική του κατάσταση.
- Έπειτα, εφαρμόζεται καθυστέρηση 20 time units και το σήμα rst τίθεται σε 0.
- Τέλος, προστίθεται καθυστέρηση 2700 time units, ώστε να δοθεί επαρκής χρόνος στον επεξεργαστή για την επεξεργασία των εντολών. Ο αριθμός αυτός προκύπτει από την παρατήρηση ότι η πρώτη εντολή χρειάζεται 5 κύκλους ρολογιού για να εκτελεστεί, ενώ κάθε επόμενη χρειάζεται 1 επιπλέον κύκλο, δηλαδή συνολικά 132 κύκλοι.
  Με περίοδο ρολογιού 20 time units, η συνολική καθυστέρηση υπολογίζεται ως:
  (132×20)+20=2660 time units
  Τα επιπλέον 20 time units αφορούν το αρχικό delay πριν ξεκινήσει η εκτέλεση των εντολών.

Όσον αφορά το αρχείο rom_bytes.data, αυτό περιέχει 512 σειρές, κάτι που σημαίνει ότι περιλαμβάνει συνολικά 128 εντολές. Το συμπέρασμα αυτό προκύπτει από το γεγονός ότι κάθε εντολή αποτελείται από 32 bits, ενώ κάθε γραμμή του αρχείου περιέχει 8 bits.

Μετά την εκτέλεση των κωδίκων, πρέπει να παρατηρηθούν οι κυματομορφές σε συνδυασμό με τις εντολές του αρχείου rom_bytes.data. Κάνοντας decoding της πρώτης εντολής, προκύπτουν τα εξής αποτελέσματα:

Assembly	Binary	Hexadecimal	Format	Instruction Set	Manual
addi x1, x0, 7	0000 0000 0111 0000 0000 0000 1001 0011	0x00700093	I-type	RV32I	addi

Πίνακας: Decoding της πρώτης εντολής



Εικόνα: Στιγμιότυπο από τις κυματομορφές, όπου απεικονίζεται το σήμα instr

Από τις κυματομορφές παρατηρείται ότι ο επεξεργαστής πραγματοποιεί **σωστά το fetch της πρώτης εντολής** από τη μνήμη.

Αντίστοιχα, για τη δεύτερη εντολή πραγματοποιείται decoding και προκύπτουν τα εξής αποτελέσματα:

- Ο επεξεργαστής εκτελεί και πάλι **σωστά το fetch της δεύτερης εντολής**, όπως επιβεβαιώνεται από τις κυματομορφές.

Με την ίδια διαδικασία ελέγχθηκαν και οι υπόλοιπες εντολές. Τα αποτελέσματα παρουσιάζονται **αναλυτικά στον παρακάτω πίνακα**.

Assembly	Binary	Hexadecimal	Format	Instruction Set	Manual
addi x2, x0, 21	0000 0001 0101 0000 0000 0001 0001 0011	0x01500113	I-type	RV32I	addi

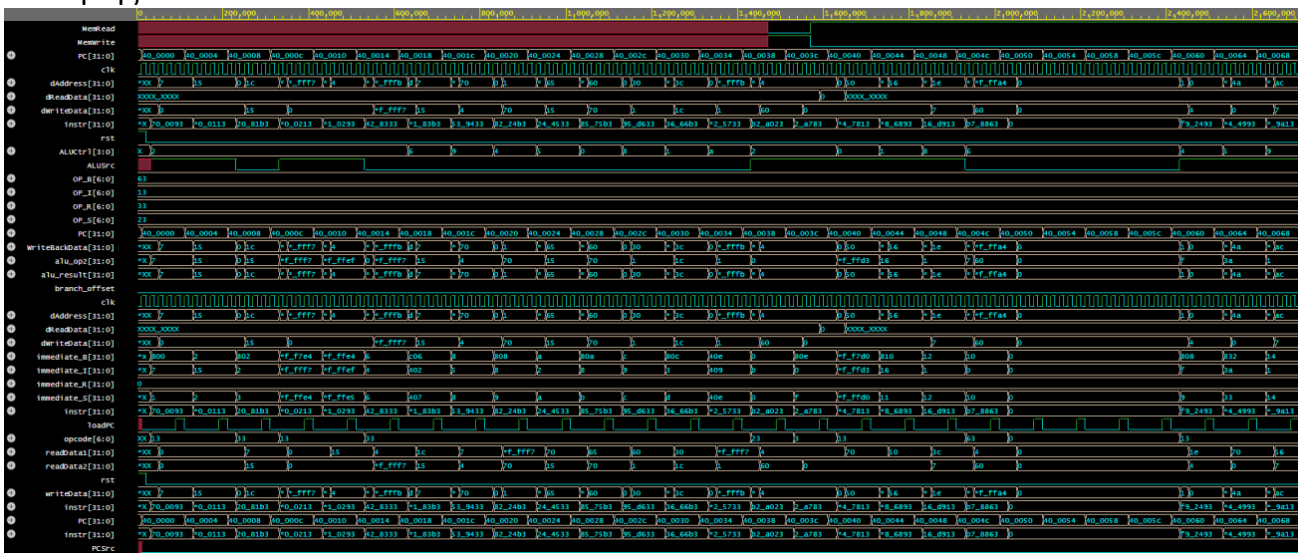
Εικόνα: Decoding 2ου instruction

Παρατηρώντας και τις κυματομορφές, βλέπουμε πως και πάλι πως ο processor έχει κάνει σωστά fetch το δεύτερο instruction από τη μνήμη. Με την ίδια λογική ελέγχθηκαν και οι υπόλοιπες εντολές, οι οποίες παρουσιάζονται αναλυτικά στο παρακάτω πίνακα:

Παρακάτω, παρατίθενται και ένα επιπλέον στιγμιότυπο από τις κυματομορφές τις άσκησης 5.

Binary	Hexadecimal	Assembly
00000000011100000000000010010011	0x00700093	addi x1, x0, 7
00000001010100000000000100010011	0x01500113	addi x2, x0, 21
00000000001000001000000110110011	0x002081b3	add x3, x1, x2
11111111011100000000001000010011	0xff700213	addi x4, x0, -9
11111110111100010000001010010011	0xfef10293	addi x5, x2, -17
00000000010000101000001100110011	0x00428333	add x6, x5, x4
01000000001000011000001110110011	0x402183b3	sub x7, x3, x2
00000000010100111001010000110011	0x00539433	sll x8, x7, x5
00000000100000100010010010110011	0x008224b3	slt x9, x4, x8
00000000001001000100010100110011	0x00244533	xor x10, x8, x2
00000000100001010111010110110011	0x008575b3	and x11, x10, x8
00000000100101011101011000110011	0x0095d633	srl x12, x11, x9
00000000001101100110011010110011	0x003666b3	or x13, x12, x3
01000000100100100101011100110011	0x40925733	sra x14, x4, x9
00000000101100101010000000100011	0x00b2a023	sw x11, 0(x5)
00000000000000010101001111000011	0x0002a783	lw x15, 0(x5)
11111101001101000111100000010011	0xfd347813	andi x16, x8, -45
00000001011010000110100010010011	0x01686893	ori x17, x16, 22
00000000000101101101100100010011	0x0016d913	srli x18, x13, 1
00000000101101111000100001100011	0x00b78863	beq x15, x11, 16
00000000111110010010010010010011	0x00f92493	slti x9, x18, 15
00000011101001000100100110010011	0x03a44993	xori x19, x8, 58
00000000000110001001101000010011	0x00189a13	slli x20, x17, 1
01000000001001111101001010010011	0x4027d293	srai x5, x15, 2

Παρακάτω, παρατίθενται και ένα επιπλέον στιγμιότυπο από τις κυματομορφές τις άσκησης 5.



Εικόνα : Στιγμιότυπο κυματομορφών άσκησης 5

Σημείωση

Για την συνολική υλοποίηση χρησιμοποιήθηκε το EDA Playground. Το project βρίσκεται στον ακόλουθο σύνδεσμο και στα αρχεία που επισυνάπτονται μαζί με την αναφορά:
https://edaplayground.com/x/VJWR