**Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΑριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης**

**Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών**

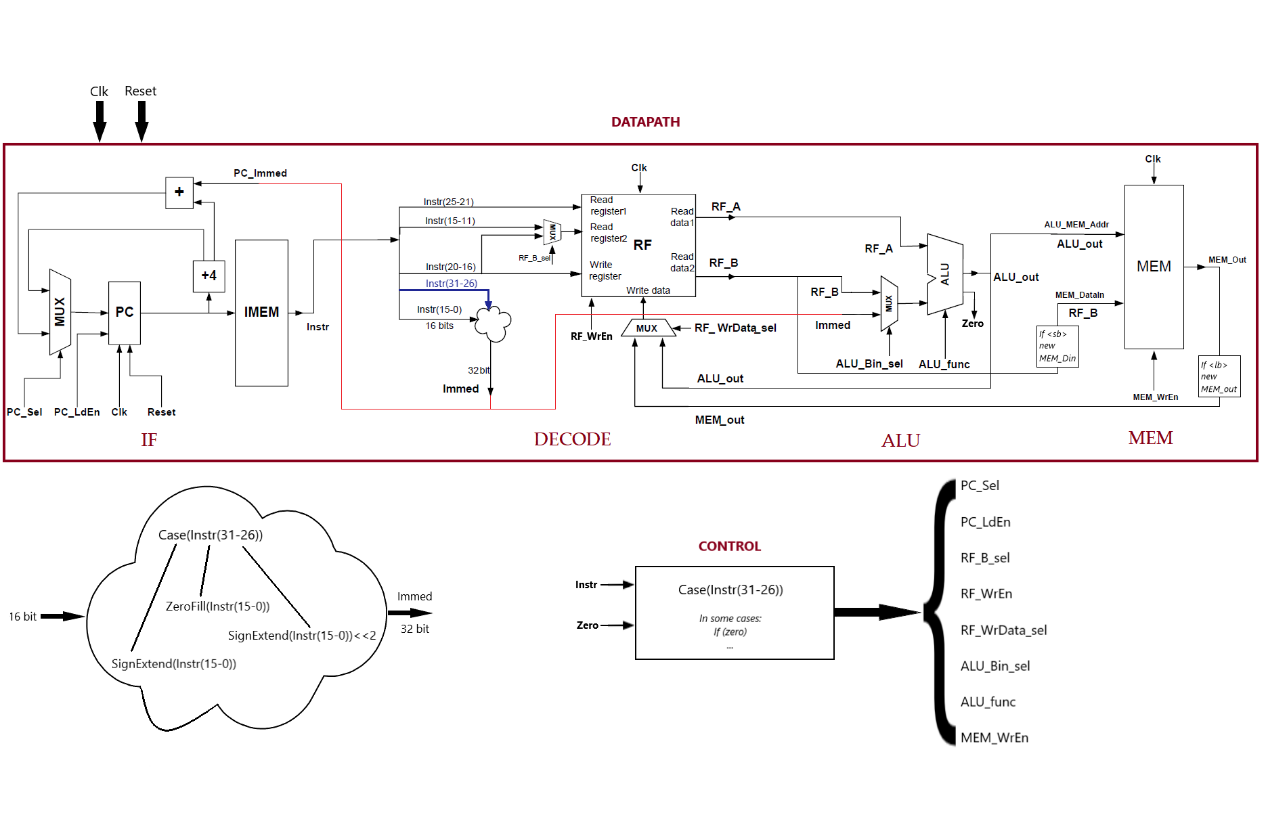
**Ψηφιακά συστήματα HW-1**

Σχεδίαση επεξεργαστή πολλαπλών κύκλων

Project Report

Λιούπης Θεόδωρος 9733

Μπαρμπαρούσης Δημήτριος 9775



Η υλοποίηση του project πραγματοποιήθηκε χρησιμοποιώντας τα εργαλεία της Xilinx σε γλώσσα Verilog. Αρχικά, δημιουργήσαμε τα submodules του επεξεργαστή και στην συνέχεια, προσομοιώσαμε την λειτουργεία του εκτελώντας τις εντολές του αρχείου *rom.data*.

Ο επεξεργαστής αποτελείται από τα παρακάτω modules:

* Processor (top module)
  1. Datapath
     1. Ifstage
     2. Decstage
     3. Alustage
     4. Memstage
  2. Control

Για την σύνθεση των παραπάνω modules ήταν απαραίτητο να δημιουργηθούν επιπλέον βοηθητικά modules όπως:

* Register modules
* Register File module
* Mux modules
* Alu module
* Memory modules

1. **Datapath Module**

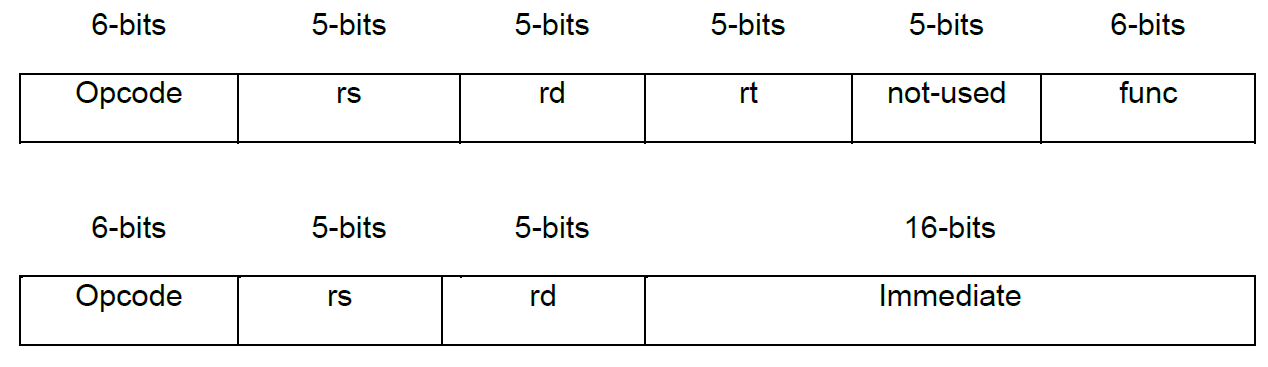
Το συγκεκριμένο module συνδέει τις τέσσερις βαθμίδες ifstage, decstage, alustage και memstage μεταξύ τους. Δέχεται σαν είσοδο τα σήματα ελέγχου του κάθε stage και σαν έξοδο δίνει την τρέχουσα εντολή, το αποτέλεσμα του alustage, την έξοδο της μνήμης RAM και το περιεχόμενο του καταχωρητή rd (αποτέλεσμα εντολής).

**A1. IFSTAGE**

Το module αυτό είναι υπεύθυνο για το Instruction Fetch. Αποτελείται από τον PC register, έναν πολυπλέκτη που ελέγχει την είσοδο του PC και την μνήμη ROM που περιέχει τις εντολές προς εκτέλεση. Δέχεται σαν είσοδο το PC\_Immed, τα σήματα ελέγχου PC\_Sel, PC\_LdEn, Clk, Reset και στην έξοδο του δίνει την εντολή προς εκτέλεση (Instr).

Σε κάθε κύκλο ρολογιού, εφόσον είναι ενεργοποιημένο το PC\_LdEn, o PC διαβάζει τη νέα του τιμή (είσοδος) η οποία καθορίζεται στον πολυπλέκτη με βάση το PC\_sel. Η είσοδος του PC μπορεί να είναι είτε PC + 4 είτε PC + 4 + PC\_Immed (για εντολές Branch). Η έξοδος του PC είναι η διεύθυνση της επόμενης εντολής στην μνήμη ROM. Η μνήμη αυτή αρχικοποιείται στην αρχή με βάση το αρχείο *rom.data* και η έξοδος της είναι η εντολή που θα εκτελεστεί στον επόμενο κύκλο ρολογιού. Τέλος, ο καταχωρητής PC μηδενίζεται όταν ενεργοποιηθεί το σήμα Reset.

Οι εντολές που εξέρχονται έχουν τις εξής δύο δομές:



**Α2. DECSTAGE**

Το module αυτό είναι υπεύθυνο για την αποκωδικοποίηση της εντολής (Instr), το διάβασμα των αντίστοιχων καταχωρητών που εμπλέκονται στην εντολή και την καταχώρηση του αποτελέσματος των αντίστοιχων εντολών. Τα modules από τα οποία αποτελείται είναι ένα register file και δύο πολυπλέκτες. Δέχεται σαν είσοδο την εντολή προς εκτέλεση (Instr), τα σήματα ελέγχου RF\_WrEn, RF\_WrData\_sel και Clk καθώς και τις εξόδους ALU\_Out και MEM\_Out των αντίστοιχων βαθμίδων. Στην έξοδο του δίνει το τροποποιημένο Immed (16 to 32 bits) , τις τιμές των καταχωρητών που διαβάζονται από το register file και τέλος, τα δεδομένα προς εγγραφή στο register file (RF[rd]).

Αρχικά, η εντολή χωρίζεται στα κατάλληλα κομμάτια σύμφωνα με την δομή της και στην συνέχεια διαβάζονται οι κατάλληλοι καταχωρητές από το register file. Συγκεκριμένα, η έξοδος RF\_A έχει συνεχώς την τιμή του καταχωρητή rs ενώ η RF\_B την τιμή του καταχωρητή rt (χωρίς immediate) ή του rd (με immediate) σύμφωνα με το σήμα ελέγχου RF\_B\_sel. Στον επόμενο κύκλο ρολογιού, εφόσον είναι ενεργοποιημένο το σήμα RF\_WrEn και με βάση το σήμα RF\_WrData\_sel, γράφουμε στον κατάλληλο καταχωρητή (rd) τα δεδομένα του ALU\_Out ή του MEM\_Out.

Στις εντολές που τα πρώτα 16 bits περιέχουν το immediate, είναι αναγκαία η μετατροπή του σε 32 bits. Για αυτόν τον λόγο στην αρχή της βαθμίδας παίρνουμε τα δεδομένα Instr(15-0) και σύμφωνα με το opcode της εντολής (Instr(15-0)) το μετατρέπουμε σε 32 bits ως εξής:

|  |  |  |
| --- | --- | --- |
| **Εντολή** | **New Immediate** | **In Verilog** |
| addi, li, lb, lw, sb, sw | SignExtend(Immed) | SiEx [15:0] = {Instr[15:0]};  SiEx[31:16] = {16{Instr[15]}}; |
| andi, ori | ZeroFill(Immed) | ZeFi [15:0] = Instr[15:0];  ZeFi [31:16] = 16'b0; |
| b,beq,bne | SignExtend(Immed)<<2 | SiEx [15:0] = {Instr[15:0]};  SiEx[31:16] = {16{Instr[15]}};  shiftSiEx = SiEx<<2; |

**Α3. ALUSTAGE**

Το module αυτό είναι υπεύθυνο για την εκτέλεση αριθμητικών και λογικών πράξεων. Αποτελείται από μία αριθμητική και λογική μονάδα (ALU) και έναν πολυπλέκτη. Έχει σαν είσοδο τις τιμές RF\_A, RF\_B, Immed και τα σήματα ελέγχου ALU\_Bin\_sel και ALU\_func. Στην έξοδο του έχει το αποτέλεσμα της πράξης (ALU\_Out) και το σήμα Zero το οποίο ενεργοποιείται όταν το αποτέλεσμα της πράξης είναι μηδέν.

Η πρώτη είσοδος της ALU είναι η τιμή RF\_A ενώ η δεύτερη είναι RF\_B ή Immed σύμφωνα με το σήμα ελέγχου ALU\_Bin\_sel. Τέλος, η πράξη που εκτελείται στην μονάδα καθορίζεται από το σήμα ALU\_func.

**Α4. MEMSTAGE**

Το module αυτό αποτελείται από μία μνήμη RAM και είναι υπεύθυνο για την εγγραφή και την ανάγνωση δεδομένων της μνήμης. Έχει ως είσοδο το αποτέλεσμα ALU\_Out της προηγούμενης βαθμίδας την τιμή του καταχωρητή RF\_B και ένα σήμα ελέγχου MEM\_WrEn.

Όταν το MEM\_WrEn είναι ενεργοποιημένο γράφουμε στην διεύθυνση μνήμης [ALU\_Out] (ΜΕΜ[ALU\_Out]) τα δεδομένα RF\_B, ενώ όταν είναι απενεργοποιημένο διαβάζουμε τα δεδομένα MEM[ALU\_Out].

Επιπλέον modules που χρειάστηκαν για την υλοποίηση των παραπάνω βαθμίδων:

**Mux modules**

Σε διάφορα σημεία της δημιουργίας του datapath ήταν απαραίτητη η χρήση διάφορων πολυπλεκτών. Για αυτό φτιάξαμε τα εξής modules:

1. mux2to1 (δύο είσοδοι των 32 bit, μία έξοδος των 32 bit)
2. mux2to1\_5bit (δύο είσοδοι των 5 bit, μία έξοδος των 5 bit)
3. mux32to1 (τριάντα δύο είσοδοι των 32 bit, μία έξοδος των 32 bit)

**Register Module**

Το module αυτό αποτελεί έναν 32 bit καταχωρητή. Τα δεδομένα στην είσοδο του καταχωρητή εγγράφονται σε κάθε νέο κύκλο ρολογιού (posedge Clk) εφόσον το σήμα WrEn είναι ενεργοποιημένο.

**Register File module**

Το αρχείο καταχωρητών αποτελείται από ένα decoder module, 32 register modules και δύο mux32to1 modules. Σκοπός του είναι η εγγραφή και η ανάγνωση των δεδομένων καταχωρητών.

**Alu module**

Το module αυτό είναι υπεύθυνο για την εκτέλεση αριθμητικών και λογικών πράξεων.

**Memory modules**

Για την δημιουργία του επεξεργαστή ήταν αναγκαία η χρήση μίας μνήμης ROM και μίας μνήμης RAM. Η πρώτη αρχικοποιείται με τις εντολές προς εκτέλεση από το αρχείο *rom.data,* ενώ η δεύτερη χρησιμοποιείται για την αποθήκευση των αποτελεσμάτων συγκεκριμένων εντολών.

1. **CONTROL**

Το συγκεκριμένο module αποτελεί την μονάδα ελέγχου του *datapath* και είναι υπεύθυνο για την αναγνώριση της εντολής και την ρύθμιση των σημάτων ελέγχου. Δέχεται σας είσοδο την εντολή Instr που βγαίνει από την βαθμίδα IFSTAGE του *datapath* και το σήμα εξόδου Zero της βαθμίδας ALUSTAGE*.* Ανάλογα με την εντολή, το control δίνει στην έξοδο τα κατάλληλα σήματα για να εκτελεστεί σωστά η εντολή στο *datapath*.

**Ανάλυση του τρόπου λειτουργείας του control**

Η αναγνώριση της κάθε εντολής γίνεται με βάση το opcode της που ισούται με το Instr[31:26] και αποθηκεύεται σε έναν ενδιάμεσο καταχωρητή. Οι βασικές κατηγορίες εντολών είναι:

1. Εντολές αριθμητικών και λογικών πράξεων
   1. Πράξεις μόνο με καταχωρητή/ες: *add, sub, and, not, or, shr, shl, sla,rol, ror*
   2. Πράξεις με καταχωρητή και Immediate: *li, addi, andi, ori*
2. Εντολές διακλάδωσης: *b, beq, bne*
3. Εντολές μνήμης
   1. Εντολές ανάγνωσης: *lb, lw*
   2. Εντολές εγγραφής: *sb, sw*

Καθορισμός των σημάτων ελέγχου ανάλογα με το είδος τις εντολής:

**1a. Πράξεις μόνο με καταχωρητή/ες**

|  |  |  |  |
| --- | --- | --- | --- |
| **Βαθμίδα** | **Σήμα Ελέγχου** | **Τιμή** | **Σχόλια** |
| IFSTAGE | PC\_sel | 0 | Διάβασμα αμέσως επόμενης εντολής ,  PC = PC + 4 |
| PC\_LdEn | 1 |
| DECSTAGE | RF\_B\_sel | 0 | Επιλογή καταχωρητή rt |
| RF\_WrEn | 1 | Εγγραφή στο regfile |
| RF\_WrData\_sel | 1 | RF[rd] = ALU\_Out |
| ALUSTAGE | ALU\_Bin\_sel | 0 | Πράξη στην ALU με RF\_B |
| ALU\_func | Instr[3:0] | Επιλογή πράξης |
| MEMSTAGE | Mem\_WrEn | 0 | Δεν γράφουμε στην μνήμη |

**1b. Πράξεις με καταχωρητή και Immediate**

|  |  |  |  |
| --- | --- | --- | --- |
| **Βαθμίδα** | **Σήμα Ελέγχου** | **Τιμή** | **Σχόλια** |
| IFSTAGE | PC\_sel | 0 | Διάβασμα αμέσως επόμενης εντολής,  PC = PC + 4 |
| PC\_LdEn | 1 |
| DECSTAGE | RF\_B\_sel | 1 | Επιλογή καταχωρητή rd |
| RF\_WrEn | 1 | Εγγραφή στο regfile |
| RF\_WrData\_sel | 1 | RF[rd] = ALU\_Out |
| ALUSTAGE | ALU\_Bin\_sel | 1 | Πράξη στην ALU με Immed |
| ALU\_func | 0 | Για την *li* \* |
|  | opcode[3:0] | Για τις *addi, andi, ori* |
| MEMSTAGE | Mem\_WrEn | 0 | Δεν γράφουμε στην μνήμη |

\*στην εντολή <*li>:* ALU\_Out = RF[rs] + Immed = RF[0] + Immed = Immed. Στην συγκεκριμένη περίπτωση ο καταχωρητής rs (Instr(25-21)) είναι πάντα ο $0 του register file ο οποίος έχει συνεχώς την έξοδο μηδέν.

**2. Εντολές διακλάδωσης**

Εντολή branch <*b>*

|  |  |  |  |
| --- | --- | --- | --- |
| **Βαθμίδα** | **Σήμα Ελέγχου** | **Τιμή** | **Σχόλια** |
| IFSTAGE | PC\_sel | 1 | Διάβασμα επόμενης εντολής ,  PC = PC + 4 + Immed |
| PC\_LdEn | 1 |
| DECSTAGE | RF\_B\_sel | 0 | Δεν μας ενδιαφέρει |
| RF\_WrEn | 0 | Δεν γράφουμε στο regfile |
| RF\_WrData\_sel | 0 | Δεν μας ενδιαφέρει |
| ALUSTAGE | ALU\_Bin\_sel | 0 | Δεν μας ενδιαφέρει |
| ALU\_func | 0 | Δεν μας ενδιαφέρει |
| MEMSTAGE | Mem\_WrEn | 0 | Δεν γράφουμε στην μνήμη |

Εντολές branch <*beq> και <bne>*

|  |  |  |  |
| --- | --- | --- | --- |
| **Βαθμίδα** | **Σήμα Ελέγχου** | **Τιμή** | **Σχόλια** |
| IFSTAGE | PC\_sel | 0 ή 1 | *<beq>:*  if(zero)->PC\_sel=1 --PC=PC+4 +Immed  else -> PC\_sel = 0 --PC=PC+4  *<bne>:*  if(!zero)->PC\_sel=1 --PC=PC+4 +Immed  else -> PC\_sel = 0 --PC=PC+4 |
| PC\_LdEn | 1 |
| DECSTAGE | RF\_B\_sel | 1 | Επιλογή καταχωρητή rd για σύγκριση |
| RF\_WrEn | 0 | Δεν μας ενδιαφέρει |
| RF\_WrData\_sel | 0 | Δεν μας ενδιαφέρει |
| ALUSTAGE | ALU\_Bin\_sel | 0 | Πράξη στην ALU με RF\_B |
| ALU\_func | 1 | Επιλογή πράξης (αφαίρεση για σύγκριση) |
| MEMSTAGE | Mem\_WrEn | 0 | Δεν γράφουμε στην μνήμη |

**3a. Εντολές ανάγνωσης από την μνήμη**

|  |  |  |  |
| --- | --- | --- | --- |
| **Βαθμίδα** | **Σήμα Ελέγχου** | **Τιμή** | **Σχόλια** |
| IFSTAGE | PC\_sel | 0 | Διάβασμα αμέσως επόμενης εντολής ,  PC = PC + 4 |
| PC\_LdEn | 1 |
| DECSTAGE | RF\_B\_sel | 1 | Επιλογή καταχωρητή rd |
| RF\_WrEn | 1 | Εγγραφή στο regfile |
| RF\_WrData\_sel | 0 | RF[rd] = MEM\_Out\* |
| ALUSTAGE | ALU\_Bin\_sel | 1 | Πράξη στην ALU με Immed |
| ALU\_func | 0 | Πρόσθεση RF[rs] + Immed |
| MEMSTAGE | Mem\_WrEn | 0 | Δεν γράφουμε στην μνήμη |

**\***Η έξοδος της μνήμης στις εντολές ανάγνωσης είναι MEM\_Out = M[ALU\_Out] = M[RF[rs] + Immed]. Στην εντολή *<lb>* όμως το MEM\_Out τροποποιείται και η νέα του δομή είναι η εξής:

*ZeroFill(31 downto 8) & MEM[RF[rs] + SignExtend(Imm)](7 downto 0)*

και σε Verilog: *{24'd0,MEM\_out[7:0] }*

**Η παραπάνω μετατροπή γίνεται στο DECSTAGE σύμφωνα με το opcode.**

**3b. Εντολές εγγραφής στην μνήμη**

|  |  |  |  |
| --- | --- | --- | --- |
| **Βαθμίδα** | **Σήμα Ελέγχου** | **Τιμή** | **Σχόλια** |
| IFSTAGE | PC\_sel | 0 | Διάβασμα αμέσως επόμενης εντολής ,  PC = PC + 4 |
| PC\_LdEn | 1 |
| DECSTAGE | RF\_B\_sel | 1 | Επιλογή καταχωρητή rd |
| RF\_WrEn | 0 | Δεν μας ενδιαφέρει |
| RF\_WrData\_sel | 0 | Δεν μας ενδιαφέρει |
| ALUSTAGE | ALU\_Bin\_sel | 1 | Πράξη στην ALU με Immed |
| ALU\_func | 0 | Πρόσθεση RF[rs] + Immed |
| MEMSTAGE | Mem\_WrEn | 1 | Γράφουμε στην μνήμη\* |

**\***Τα δεδομένα εγγραφής στην είσοδο της μνήμης είναι η έξοδος RF\_B του register file. Στην συγκεκριμένη περίπτωση έχουμε MEM\_DataIn = RF[rd]. Στην εντολή *<sb>* όμως το ΜΕΜ\_DataIn τροποποιείται και η νέα του δομή είναι η εξής:

*ZeroFill(31 downto 8) & RF[rd] (7 downto 0)*

και σε Verilog: {24'd0,wRF\_B[7:0]}

**Η παραπάνω μετατροπή γίνεται στο DECSTAGE σύμφωνα με το opcode.**

**Processor**

O processor αποτελεί το top module της σύνθεσης μας. Σκοπός του είναι η σύνδεση του control με το datapath και η παραγωγή κύκλων ρολογιού (Clk). Έχει ως είσοδο ένα σήμα reset και στην έξοδο του τα ALU\_Out, MEM\_Out, και την τιμή προς αποθήκευση στον RF[rd].

Reset: Το σήμα αυτό όταν είναι high ο PC register μηδενίζεται και ξεκινάει η ανάγνωση της μνήμης ROM από την αρχή. Όσο παραμένει low, ο PC αυξάνεται σε κάθε κύκλο ρολογιού.

Clk: Με το σήμα αυτό επιτυγχάνεται ο συγχρονισμός όλων των βαθμίδων. Πιο συγκεκριμένα, είναι απαραίτητο για τα modules PCregister, ROM, Register File και RAM.

**ΠΑΡΑΤΗΡΗΣΕΙΣ:**

1. Κάθε εντολή χρειάζεται έναν κύκλο ρολογιού για να εκτελεστεί. Για να αναλύσουμε σωστά τον χρόνο που χρειάζονται οι εντολές διακρίνουμε τις εξής κατηγορίες:
   1. Εντολές που δεν χρησιμοποιούν την μνήμη RAM
   2. Εντολές που χρησιμοποιούν την μνήμη RAM

Στην (a) περίπτωση το Instruction Fetch πραγματοποιείται με την άνοδο του ρολογιού και μέχρι την επόμενη άνοδο, γίνεται η πράξη που δηλώνει η εντολή και τα δεδομένα του αποτελέσματος περιμένουν να γραφτούν στον Register File. Ένας καταχωρητής γράφει τα δεδομένα στην άνοδο του ρολογιού όπως φαίνεται και στον κώδικα της Verilog:

*module register(input signed [31:0] Data, input CLK, input WE,*

*output signed [31:0] Dout);*

*reg [31:0] res;*

*assign Dout = res;*

*always @(posedge CLK)*

*begin*

*if (WE)*

*res <= Data;*

*end*

*endmodule*

Όταν έρθει το επόμενο posedge Clk ταυτόχρονα αλλάζει η κατάσταση του WE (RF\_WrEn) σύμφωνα με την επόμενη εντολή. Έτσι, για να μην υπάρξει λάθος στα δεδομένα εγγραφής στο always block του καταχωρητή έχει γίνει non-blocking assignement.

Στην (b) περίπτωση το Instruction Fetch πραγματοποιείται πάλι με την άνοδο του ρολογιού, γίνεται η πράξη που δηλώνει η εντολή και τα δεδομένα του αποτελέσματος περιμένουν είτε να γραφτούν στην μνήμη είτε να χρησιμοποιηθούν για ανάγνωση της μνήμης. Προκειμένου να υπάρξει συγχρονισμός ορίσαμε την μνήμη RAM να λειτουργεί στην κάθοδο του ρολογιού. Αυτό βέβαια, στις εντολές που γράφουνε στην μνήμη (*sb, sw*) δεν ήταν απαραίτητο λόγω του non-blocking assignement. Παρατηρήσαμε ότι στις εντολές ανάγνωσης (*lb, lw*) τα δεδομένα της εξόδου της μνήμης έπρεπε να φτάσουν στην είσοδο του RF στον ίδιο κύκλο ρολογιού ώστε να σήματα ελέγχου να μην έχουν αλλάξει με βάση την επόμενη εντολή, ειδικά στην εντολή <*lb*> όπου το MEM\_Out τροποποιείται σύμφωνα με το opcode. Οπότε στον μισό κύκλο ρολογιού εκτελείται η πράξη της εντολής και στον άλλον μισό χρησιμοποιείται η μνήμη και ο RF.

1. Παρατηρήσαμε ότι υπήρχε καθυστέρηση ίση με έναν κύκλο ρολογιού μεταξύ της εξόδου του PC register και της εξόδου της μνήμης ROM. Αυτό συνέβαινε, λόγω της λειτουργίας και των δύο module στην άνοδο και μόνο του ρολογιού. Έτσι, συγχρονίσαμε την μνήμη με βάση την έξοδο του PC. Όταν η νέα διεύθυνση βγει από τον PC (με βάση το Clk) η μνήμη δεν καθυστερεί και βγάζει την επόμενη εντολή. Για να το πετύχουμε αυτό ορίσαμε την μνήμη να λειτουργεί με always (\*).

*module rom(input clk, input [9:0] addr,*

*output reg [31:0] dout);*

*reg [31:0] ROM [1023:0];*

*initial*

*begin*

*$readmemb("rom3.data", ROM);*

*end*

*always @(\*)*

*begin*

*dout <= ROM[addr];*

*end*

*endmodule*

1. **ΠΡΟΣΟΧΗ!** Επειδή είχαμε δύο αρχεία *rom.data,* ένα για το part2 και ένα για το part3, τα μετονομάσαμε σε *rom2.data* και *rom3.data*. Στον τελικό κώδικα που υποβάλαμε, στην μνήμη rom διαβάζουμε το *rom3.data* (όπως φαίνεται πάνω στον κώδικα)*.* Οπότε, είτε το σημείο αυτό στον κώδικα, είτε το όνομα του αρχείου πρέπει να αλλαχτεί για να τρέξει σε άλλο μηχάνημα η προσομοίωση.

**Simulation**

Παρακάτω φαίνονται οι κυματομορφές των αποτελεσμάτων της εκτέλεσης των εντολών του αρχείου *rom.data.* Η προσομοίωση έγινε με το αρχείο test bench του processor module *(processor\_tb.v).*

