Οργάνωση Υπολογιστών
Εργασία#2:Σχεδίαση επεξεργαστών πολλαπλών κύκλων και pipelined

Αναφορά

Γιάννης Περίδης

2018030069

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΑΡΙΝΌ ΕΞΆΜΗΝΟ 2021

Διδάσκων καθηγητής: Ιωαννίδης Σωτήριος

Υπεύθυνος εργαστηρίου: Παπαδημητρίου Κυπριανός

Σκοπός εργαστηριακής άσκησης

Στην εργαστηριακή άσκηση αυτή, κλήθηκα να κατασκευάσω έναν ολοκληρωμένο επεξεργαστή πολλαπλών κύκλων και έναν επεξεργαστή pipeline , οι οποίοι θα συνδεθούν μαζί με το ήδη υπάρχον module μνήμης RAM για να εξεταστεί η λειτουργία τους .Η σχεδίαση των πιο σύνθετων αυτών επεξεργαστών, βασίστηκε κυρίως στις μονάδες datapath και control του επεξεργαστή μονού κύκλου, πάντα βέβαια κάνοντας τις απαραίτητες αλλαγές και προσθήκες στις βαθμίδες του, αλλά και στην δημιουργία εξ ολοκλήρου νέων modules όπου κρίθηκε αναγκαίο. Η άσκηση χωρίστηκε σε 2 φάσεις ($4^{η}$ και $5^{η}$ φάση από τις συνολικά 5) που για την κάθε μια προτού ξεκινήσει η υλοποίηση της χρειάστηκε η κατανόηση σε βάθος της θεωρητικής λειτουργίας του κάθε επεξεργαστή.

Ο σκοπός της $4^{ης}$ φάσης , ήταν η προσθήκη καταχωρητών ανάμεσα στις βαθμίδες του datapath του επεξεργαστή μονού κύκλου, με σκοπό να μετατραπεί σε επεξεργαστή πολλαπλών κύκλων(η λειτουργία του θα αναλυθεί αργότερα).Ακόμη, ή σχεδίαση μιας μηχανής πεπερασμένων καταστάσεων (FSM), η οποία αναλαμβάνει τον ρόλο του control του επεξεργαστή, δηλαδή ελέγχει τη ροή εκτέλεσης της κάθε εντολής γεννώντας τα απαιτούμενα σήματα ελέγχου σε κάθε κύκλο ρολογιού. Τέλος, για την επικύρωση της ορθής λειτουργίας του κυκλώματος χρειάστηκε η δημιουργία διαφόρων αρχείων MIPS like εντολών. Για την ολοκλήρωση της παραπάνω διαδικασίας απαραίτητη ήταν η γνώση σχεδίασης σύγχρονης με το ρολόι FSM με σήμα reset και η κατανόηση των διαφορετικών format των εντολών MIPS.

Ο σκοπός της $5^{ης}$ και τελικής φάσης, ήταν η αλλαγή ορισμένων βαθμίδων του datapath του επεξεργαστή μονού κύκλου και η προσθήκη νέων σημάτων ελέγχου και νέων
λογικών κυκλωμάτων και καταχωρητών όχι μόνο για τα σήματα του datapath, αλλά αυτή τη φορά και για τα σήματα ελέγχου της μονάδας control.Με αποτέλεσμα την δημιουργία μιας αρκετά σύνθετης σχεδίασης για έναν pipeline επεξεργαστή. Για την ολοκλήρωση της παραπάνω διαδικασίας, απαραίτητη ήταν η αναλυτική και προσεκτική σχεδίαση των συνδέσεων μεταξύ των κυκλωμάτων και η προσπέλαση των απαραίτητων σημάτων κάθε φορά των σημάτων από την μια βαθμίδα στους καταχωρητές και μετά στην επόμενη και ξανά το ίδιο.

Προεργασία

Για την σχεδίαση ολόκληρης της εργασίας χρησιμοποιήθηκε το εργαλείο της γλώσσαςπεριγραφής υλικού VHDL, Xilinx ISE έκδοση 14.7,με το οποίο έχουμε ξαναδουλέψει και πλέον εξοικειωθεί. Για την σχεδίαση των block diagrams των datapath του επεξεργαστή πολλαπλών κύκλων και της FSM που βρίσκεται στην μονάδα control του και επίσης του ενωμένου σχήματος του control και datapath του επεξεργαστή pipeline, χρησιμοποιήθηκε το λογισμικό DIA και συγκεκριμένα του πακέτου σχημάτων Digital.

Περιγραφή

4η Φάση

Στην $4^{η}$ φάση, το κύκλωμα που κατασκευάστηκε είναι ένας επεξεργαστής πολλαπλών κύκλων. Η κυριότερη διαφορά μεταξύ ενός επεξεργαστή πολλαπλών κύκλων με έναν μονού κύκλου, είναι πως στον μονού κύκλου, το ρολόι πρέπει να είναι αρκετά μεγάλο όσο για να καλύπτει την πιο χρονοβόρα εντολή που μπορεί να εκτελεστεί δηλαδή παίρνει την τιμή του critical path.Επομένως περιορίζεται και δεν μπορεί η περίοδος του να πέσει σε λιγότερα δευτερόλεπτα, επομένως, οι μικρές και οι συνηθισμένες εντολές θα εκτελούνται σε ίδιο χρόνο με την μεγαλύτερη δυνατή εντολή. Αντιθέτως αν οι εντολές προσεγγιστούν αντί σαν ένα μεγάλο βήμα, σαν πολλά μικρότερα και χωριστούν κατάλληλα, τότε μπορεί ο χρόνος του ρολογιού να μειωθεί σημαντικά και οι εντολές να εκτελούνται σε πολλούς κύκλους ανάλογα το format τους.

Όπως προαναφέρθηκε , κάθε εντολή αποτελείται από διαφορετικό αριθμό βημάτων, δηλαδή κύκλων ρολογιού που χρειάζεται για να ολοκληρωθεί, ανάλογα με το είδος της.
Τα πρώτα 2 βήματα είναι ίδια για όλες τις εντολές ανεξαρτήτως ποιες είναι:

1. *Instruction Fetch*: Στο βήμα αυτό, φορτώνεται η εντολή από την μνήμη RAM από το instruction segment και περνιέται στον PC και αυξάνεται το PC+4 για να την επόμενη εντολή

2. *Decode*: Στο βήμα αυτό, διαβάζεται και αποκωδικοποιείται η εντολή που έρχεται από τον PC

3. *Execution*: Στο βήμα αυτό, αναλόγως την εντολή:
    a. *R-type*: Υπολογισμός αποτελέσματος πράξης της ALU
    b. *Branch/ Branch Equal/ Branch Not Equal*: Υπολογισμός διεύθυνσης της επόμενης εντολής μετά την διακλάδωση PC+4+4*Immediate
    c. *Memory Load/ Memory Store*=: Υπολογισμός της διεύθυνσης πρόσβασης στην μνήμη

4. *Memory Read:* Στο βήμα αυτό φορτώνονται δεδομένα από την μνήμη RAM από το data segment, προφανώς το βήμα αυτό θα γίνει μόνο σε εντολές Load

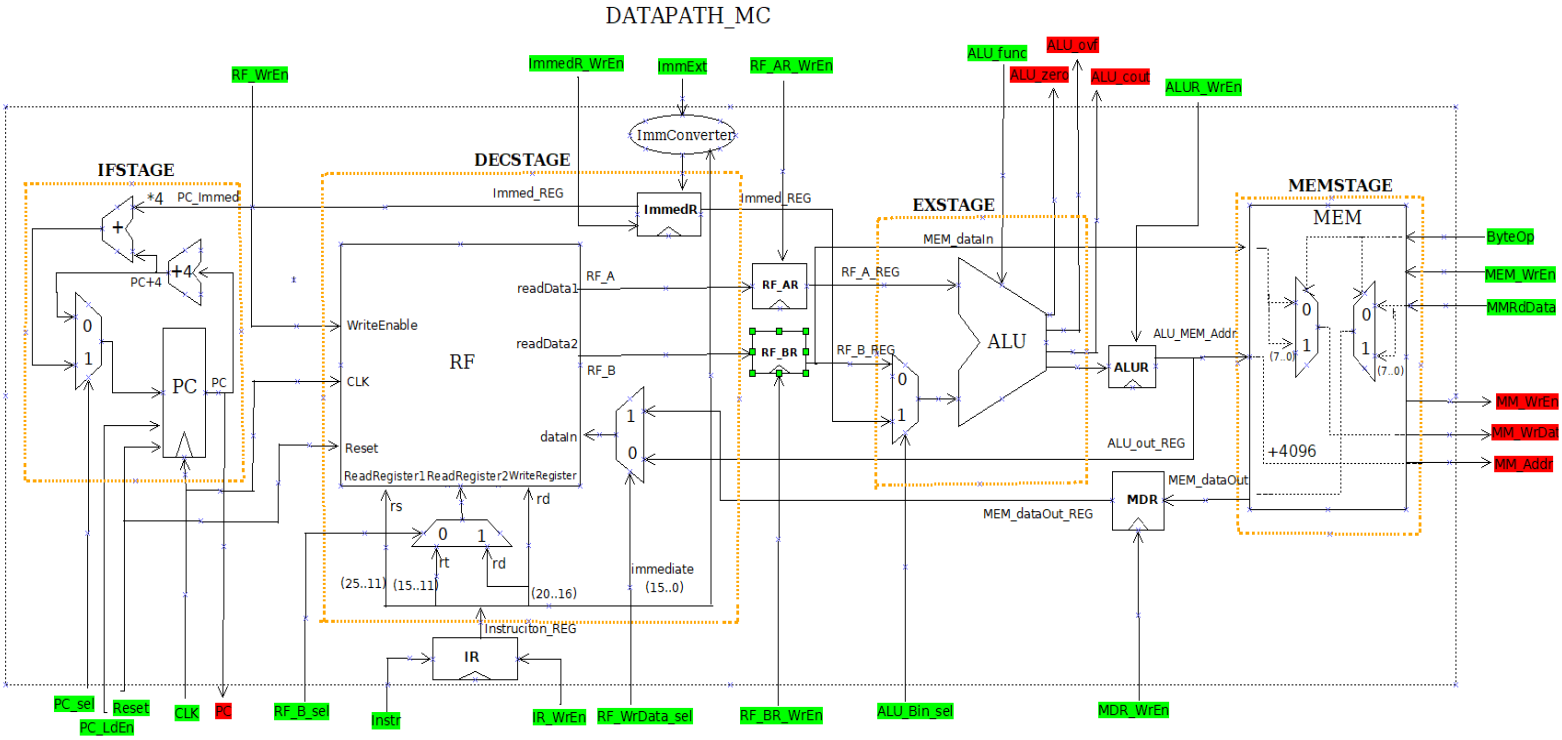
5. *Memory Write:* Στο βήμα αυτό γράφονται δεδομένα στην μνήμη RAM στο data segment, προφανώς το βήμα αυτό θα γίνει μόνο σε εντολές Store

6. *Write Back:* Στο βήμα αυτό, αναλόγως την εντολή:
    a. *Memory Load:* Γράφονται τα δεδομένα της μνήμης στο αρχείο καταχωρητών
    b. *R-type*: Γράφεται το αποτέλεσμα της πράξης της ALU στο αρχείο καταχωρητών

- *Branch / Beq / Bne* =>IF-DEC-EX **3** clock cycles
- *R-type* =>IF-DEC-EX-WB **4** clock cycles
- *Memory Store (Byte or Word)* =>IF-DEC-EX-MW **4** clock cycles
- *Memory Load (Byte or Word)* =>IF-DEC-EX-MR-WB **5** clock cycles

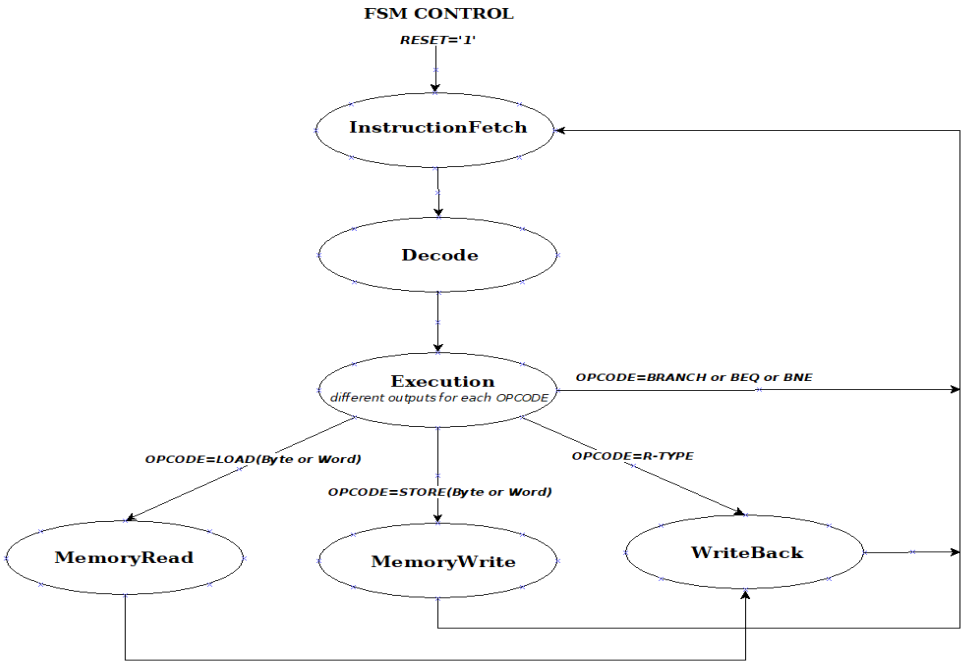
Είναι φανερό, πως για να γίνει εφικτός ο διαχωρισμός των βημάτων αυτών πρέπει σε κάθε κύκλο ρολογιού να είναι ενεργό μόνο ένα από τα στάδια επεξεργασίας της εντολής. Για να επιτευχθεί το παραπάνω, θα πρέπει να εισάγουμε καταχωρητές μεταξύ των σταδίων έτσι ώστε να αποθηκεύονται οι τιμές που χρειαζόμαστε να χρησιμοποιήσουμε σε κάποιο επόμενο κύκλο ρολογιού κάθε φορά. Παρακάτω φαίνεται το datapath του πολλαπλών κύκλων επεξεργαστή και φαίνονται καθαρά οι καταχωρητές που πρέπει να προστεθούν και πως συνδέονται με τα στάδια.

- ***Instruction Register***: αποθηκεύει την εντολή που θα πρέπει να τρέξουμε για 3 έως 5 κύκλους και δεν αφήνει αφού αποθηκεύσει την εντολή, να προχωρήσει η επόμενη μέχρι να ολοκληρωθεί η τρέχουσα.
- ***RF_A /RF_B registers***: αποθηκεύουν τα δεδομένα των read επεξεργαστών A και B αντίστοιχα
- ***Immediate register****:* αποθηκεύει την τιμή του εκτεταμένου immediate
- ***ALU register***: αποθηκεύει το αποτέλεσμα της ALU
- ***Memory Data register***: αποθηκεύει τα δεδομένα της μνήμης



Block diagram of DATAPATH_MC

Στην συνέχεια φαίνεται παρακάτω , ένα κομμάτι της μονάδας ελέγχου, το οποίο είναι μια μηχανή πεπερασμένων καταστάσεων που είναι υπεύθυνη για να κάνει τις μεταβάσεις μεταξύ των σταδίων αναλόγως το format της εντολής .Η μηχανή αλλάζει σύγχρονα με το ρολόι στην θετική ακμή του ,έχει σήμα Reset και παίρνει σαν είσοδο το OPCODE της εντολής, που σύμφωνα με αυτό θα ακολουθηθούν και οι αντίστοιχες μεταβάσεις στα στάδια. Παράγει ένα μέρος από τα απαιτούμενα σήματα ελέγχου, συγκεκριμένα παράγει τα Write Enables,σήματα εγγραφής όλων των νέων καταχωρητών, του RF και της μνήμης και φυσικά του PC.Τα υπόλοιπα σήματα παράγονται με τον ίδιο τρόπο με τον μονού κύκλου καταχωρητή.



Block diagram of FSM in control

5η Φάση

Στην 5η φάση το κύκλωμα που κατασκευάστηκε είναι ένας pipelined επεξεργαστής. Η διαφορά που καθιστά αυτόν τον επεξεργαστή καλύτερο, αλλά και πιο περίπλοκο σε σχέση με τους 2 άλλους που έχουν σχεδιαστεί είναι πως ο pipelined επεξεργαστής μπορεί να εκτελεί πολλές εντολές μαζί σε έναν κύκλο ρολογιού, σε αντίθεση με τους μονού και πολλαπλών κύκλων στους οποίους για να ξεκινήσει να εκτελείται μια νέα εντολή πρέπει να έχει ολοκληρωθεί πρώτα η τρέχουσα. Ουσιαστικά θα γίνει επικάλυψη εντολών. Ο επεξεργαστής pipeline, μοιάζει αρκετά με τον multicycle,με την διαφορά ότι τώρα τρέχουμε την επόμενη εντολή με το που γίνει εφικτό , χωρίς να περιμένουμε να τελειώσει η τρέχουσα.
Η αρχή λειτουργίας του pipeline επεξεργαστή, βασίζεται επίσης στον διαχωρισμό του σε στάδια .Η προϋπόθεση για να λειτουργήσουν πολλαπλές εντολές παράλληλα στον ίδιο κύκλο, είναι απλά να μην βρίσκονται στο ίδιο στάδιο την ίδια χρονική στιγμή.
Παρακάτω φαίνονται τα στάδια του και η λειτουργία τους:

1. *IF_stage:* Σε αυτό το στάδιο, φορτώνεται η εντολή από την μνήμη RAM από το instruction segment και περνιέται στον PC και αυξάνεται το PC+4 για να την επόμενη εντολή_

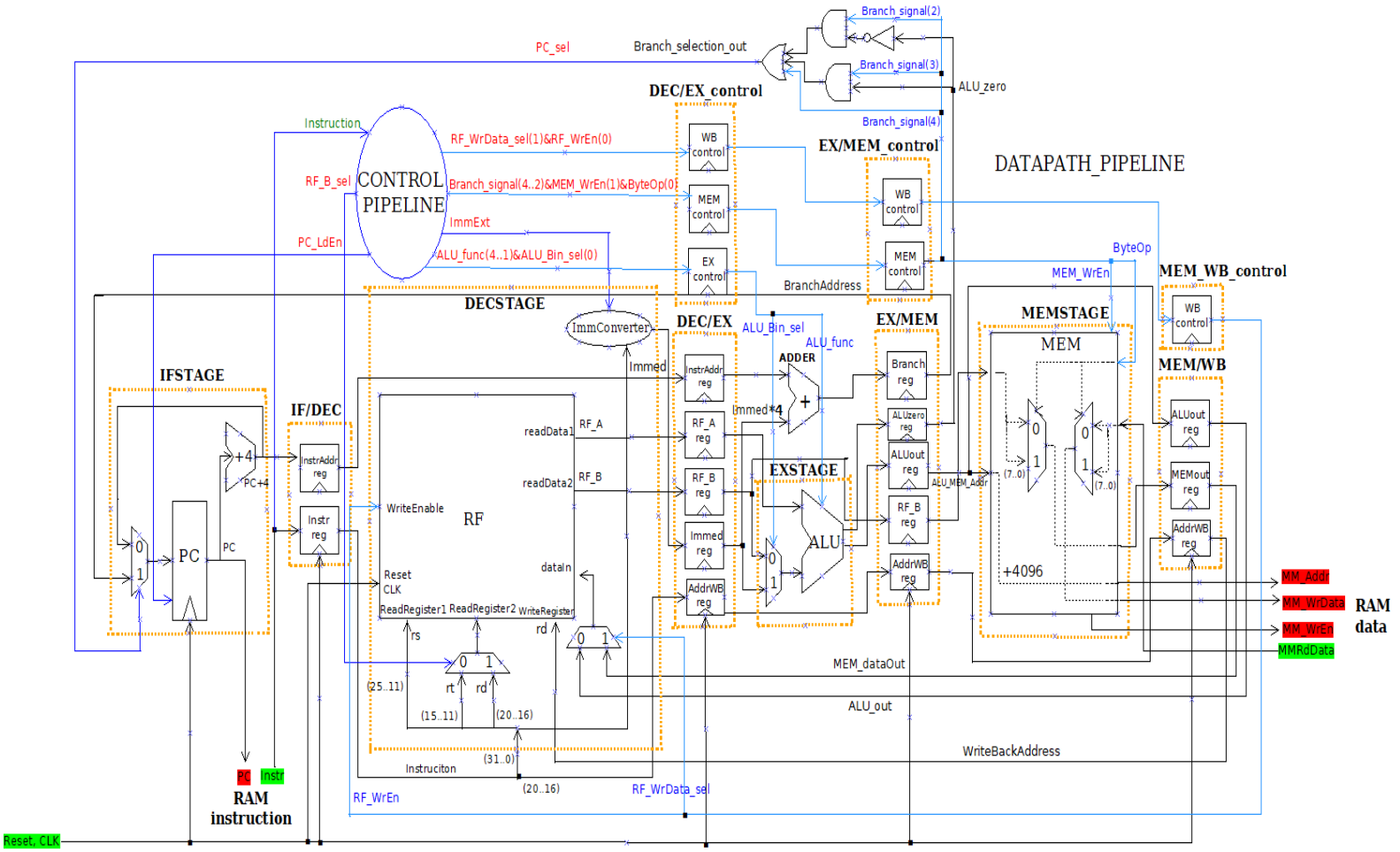
2. *DEC_stage:* Σε αυτό το στάδιο, αποκωδικοποιείται η εντολή, διαβάζονται οι καταχωρητές από το RF, επεκτείνεται το immediate, υπολογίζεται η διεύθυνση του καταχωρητή εγγραφής

3. *EX_stage:* Σε αυτό το στάδιο, υπολογίζεται η διεύθυνση σε περίπτωση που γίνει branch, υπολογίζεται η διεύθυνση προσπέλασης της μνήμης και γίνεται η πράξη στην ALU είτε μεταξύ 2 καταχωρητών, είτε μεταξύ καταχωρητή και immediate

4. *MEM_stage:* Σε αυτό το στάδιο, φορτώνουμε δεδομένα από ή γράφουμε δεδομένα στο data segment της RAM

5. *WB_stage:* Σε αυτό το στάδιο ανανεώνεται, γράφεται ο RF με ένα αποτέλεσμα της ALU ή από ένα load από δεδομένα της μνήμης.

Είναι φανερό πως για να υλοποιηθεί το παραπάνω, θα χρειαστούν καταχωρητές στο datapath μεταξύ των 5 σταδίων , οι οποίοι θα αποθηκεύουν όλες τις εξόδους που βγαίνουν από ένα στάδιο και θα τις περνάνε σαν εισόδους στο επόμενο στάδιο. Ακόμη, τώρα σε αντίθεση με πριν θα χρειαστούν και καταχωρητές για τα σήματα ελέγχου του control, καθώς και αυτά είναι επίσης απαραίτητα για να μην χάνεται η αναγκαία πληροφορία που χρειάζονται οι εντολές για να λειτουργήσουν όταν περνάνε από το ένα στάδιο στο επόμενο. Για κάθε ομάδα registers datapath και control ,μεταξύ των σταδίων , δημιουργήθηκαν νέα modules τα οποία ύστερα συνδέθηκαν μαζί με το datapath και το control σε ένα ανώτερο module, με σκοπό την ιεραρχική top-down σχεδίαση του κυκλώματος. Τέλος, σχεδιάστηκε άλλη μια μονάδα με λογικές πύλες δύο επιπέδων, με σκοπό την σωστή επιλογή του πότε ο PC θα πηγαίνει στην διεύθυνση του branch. Παρακάτω φαίνεται το αναλυτικό block diagram , ενωμένων λεπτομερώς όλων των μονάδων (control-datapath-regisetrs) εκτός της μνήμης RAM .



Block diagram of CONTROL_PIPELINE +DATAPATH_PIPELINE

Εδώ θα πρέπει να αναφερθεί πως η σχεδίαση του κυκλώματος είναι ακόμα πιο σύνθετη από αυτή που φαίνεται, καθώς και σε ένα τέτοιου είδους κύκλωμα με επικαλύψεις εντολών , είναι πολύ συχνό φαινόμενα να υπάρξουν κίνδυνοι δεδομένων (data hazards) , οι οποίοι πρέπει να αντιμετωπιστούν με νέα κατάλληλα λογικά κυκλώματα που τα διαχειρίζονται, πράγμα που δεν έγινε στην συγκεκριμένη σχεδίαση λόγω έλλειψης χρονικών περιθωρίων παράδοσης.

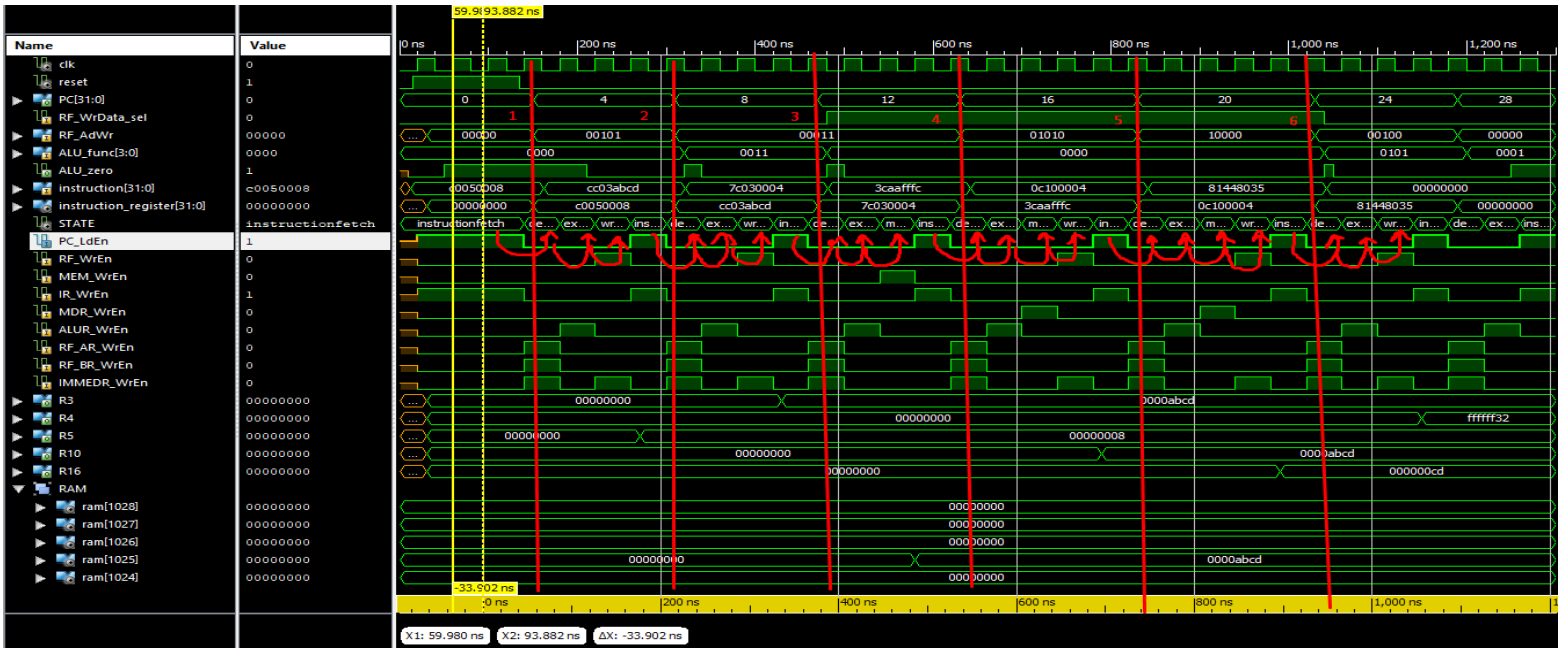
Κυματομορφές-Προσομοίωση

Παρακάτω παρουσιάζονται οι κυματομορφές του επεξεργαστή πολλαπλών κύκλων συνολικά, δηλαδή το test bench του top-module PROCESSOR_MC. Με σκοπό την προσομοίωση της λειτουργίας του , πρέπει να φορτωθούν στην μνήμη δύο αρχεία, στα οποία έχουν γραφτεί οι εντολές του πρώτου και του δεύτερου σετ εντολών προγράμματος αναφοράς.

Αρχικά στο πρώτο σετ στο αρχείο **anafora1.data** εκτελούνται οι παρακάτω εντολές

```
1 00: addi r5, r0, 8
2 04: ori  r3, r0, 0xABCD
3 08: sw   r3, 4(r0)
4 0C: lw   r10, -4(r5)
5 10: lb   r16, 4(r0)
6 14: nand r4, r10, r16
```

Αρχικά για τους πρώτους 3 κύκλους το σήμα reset είναι ενεργό ,οπότε όλες οι έξοδοι μηδενίζουν. Ύστερα, εκτελείται 1 εντολή για πολλαπλούς κύκλους ρολογιού ανάλογο το είδος της και ο PC αυξάνεται κατά 4 σε κάθε instruction fetch. Παρατηρώ ακόμα πως λόγο των καθυστερήσεων που έχουν προστεθείστα submodules του κυκλώματος, η έξοδος καθυστερεί κάποια ns να πάρει τιμή σε σχέση με την θετική ακμή του ρολογιού.



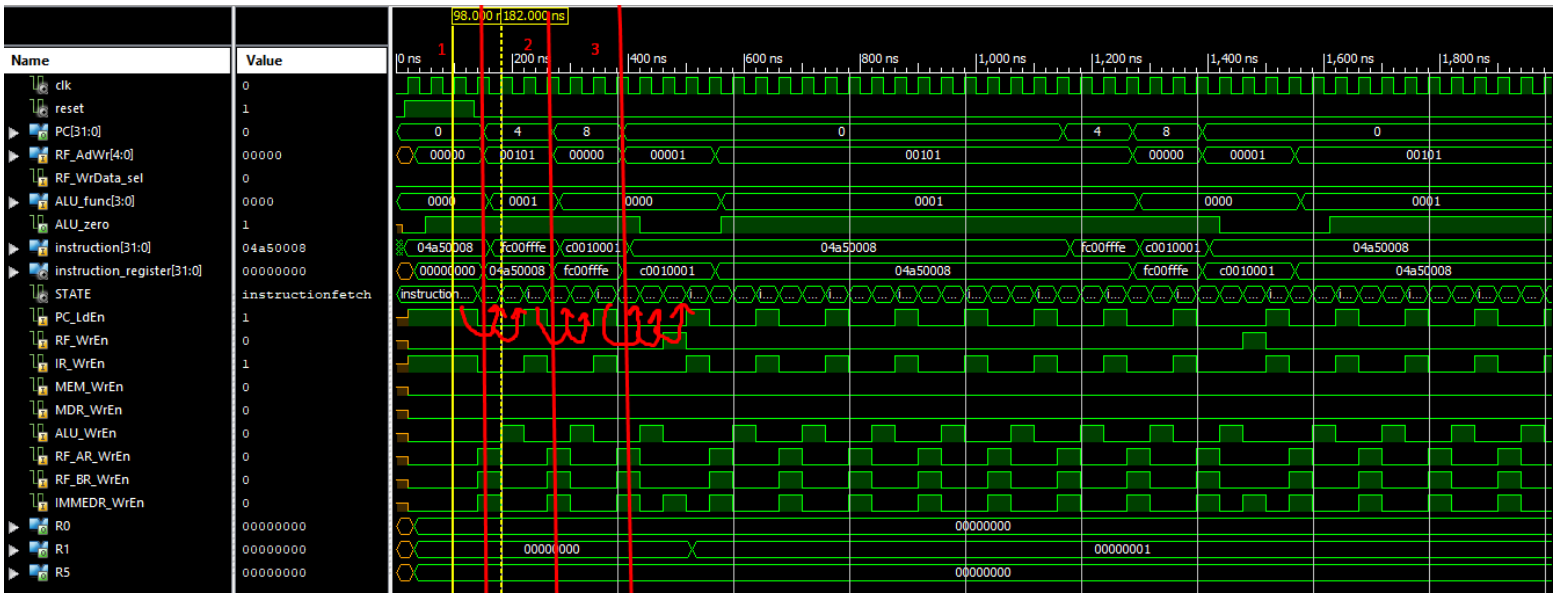
Στα σημαδεμένα σημεία έχω:

1. Καταχωρείται στον R5 το 8(hex)
2. Καταχωρείται στον R3 το ABCD(hex)
3. Καταχωρείται στη διεύθυνση λέξης μνήμης 1025(2η διεύθυνση λέξης του datasegment) το ABCD(hex) από τον R3
4. Φορτώνεται από την διεύθυνση μνήμης 1025 η τιμή ABCD(hex) στον R10
5. Φορτώνεται από την διεύθυνση μνήμης 1025 το τελευταίο Byte της τιμής ABCD(hex), το CD(hex) στον R10
6. Γίνεται πράξη NAND μεταξύ R10 και R16=>0000ABCD nand 000000CD=FFFFFF32

Αυτό που πρέπει να παρατηρήσουμε από την κυματομορφή είναι ο καταχωρητής instruction ο οποίος φορτώνει μια μια τις εντολές από την μνήμη, και είναι πάντα μια πίσω από το κανονικό Instruction που έρχεται. Ακόμη, παρατηρούμε πως στις R-type εντολές addi,ori και nand εκτελούνται σε 4 κύκλους ρολογίου όπως και ήταν αναμενόμενο από την θεωρία και ακολουθούν τα states :IF-DEC-EX-WB.Αντίστοιχα η εντολή store εκτελείται σε 4 κύκλους και ακολουθεί τις καταστάσεις :IF-DEC-EX-MW. Τέλος, οι δύο εντολές load word/byte εκτελούνται σε 5 κύκλους και ισχύει για τις καταστάσεις :IF-DEC-EX-MR-WB.Επίσης, πρέπει να παρατηρήσουμε τα write enables που ισχύουν σε κάθε state, τα κυριότερα από αυτά είναι πως το PC_LdEn και το IR_WrEn είναι ενεργά στο instruction fetch, ενώ τα RF_WrEn και MEM_WrEn είναι ενεργά στο write back και στο memory write αντίστοιχα.

Στην συνέχεια στο δεύτερο σετ ,στο αρχείο **anafora2.data** εκτελούνται οι παρακάτω εντολές:

```
1 00: bne r5, r5, 8
2 04: b -2
3 08: addi r1, r0, 1
```



Οι αρχικές παρατηρήσεις ισχύουν και για αυτό το σετ. Στα σημαδεμένα σημεία έχω:

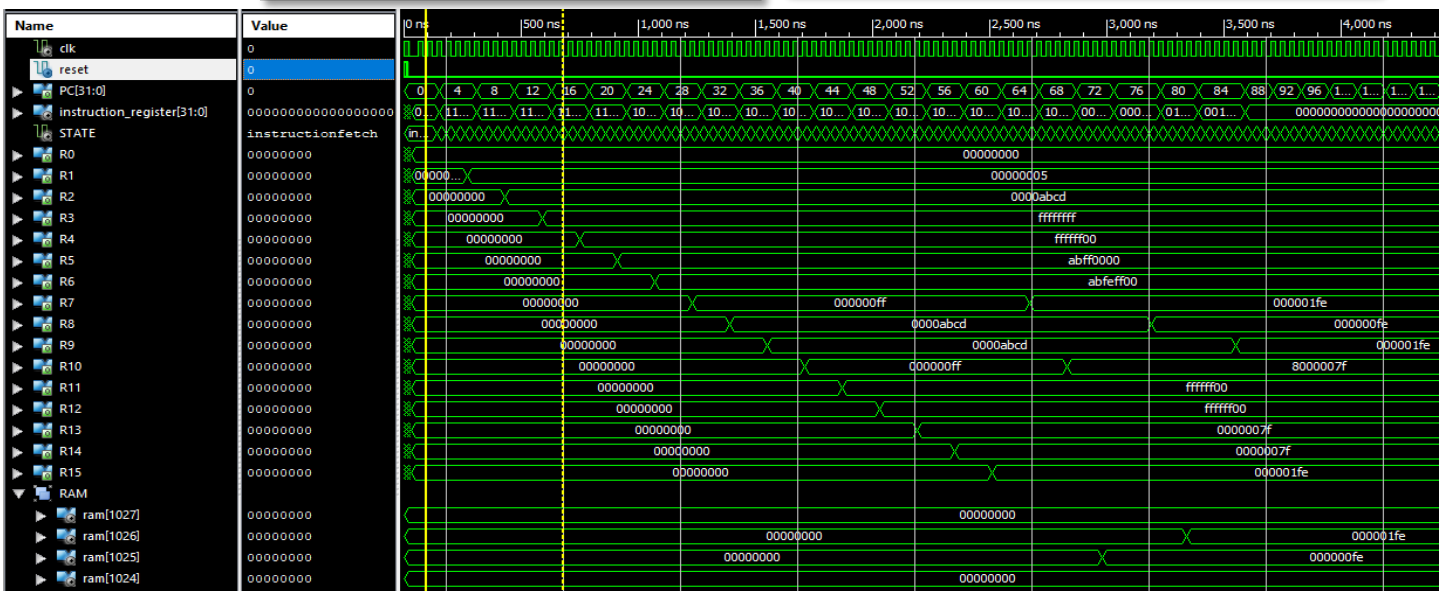
1. Εντολή διακλάδωσης στην θέση 8, αν ο R5≠R5,συνθήκη που δεν θα συμβεί ποτέ στο πρόγραμμά μας. Η εκτέλεση συνεχίζεται κανονικά στην επόμενη εντολή PC+4
2. Εντολή διακλάδωσης στην θέση 4+4+4*(-2)=0, δηλαδή γίνεται επιστροφής την 1η εντολή .Επομένως μπαίνουμε σε έναν ατέρμον βρόγχο μεταξύ των 2 αυτών εντολών καιη εντολή 3 δεν εκτελείται ποτέ

Παρατηρούμε πως ισχύουν τα ίδια με προηγουμένως όσον αφορά τον instruction register και τα write enables.Επίσης φαίνεται όπως και είναι το σωστό, ότι οι εντολές διακλάδωσης bne και branch, χρειάζονται μονάχα 3 κύκλους ρολογιού για να ολοκληρωθούν και ακολουθούν τα στάδια: IF-DEC-EX.Τέλος γίνεται αντιληπτό ένα μικρό λάθος στην κυματομορφή , το οποίο πιθανότατα θα οφείλεται στον τρόπο σχεδίασης της μονάδας ελέγχου control.Ενώ η εντολή branch εκτελείται κανονικά και μας πηγαίνει στην επιθυμητή διεύθυνση , η επόμενη εντολή από την branch προλαβαίνει να εκτελεστεί αφού έχει ήδη γίνει το instruction fetch της (στην περίπτωσή μας εκτελείται η εντολή addi κάτω από το branch) και ύστερα εκτελείται η διακλάδωση στην επιθυμητή διεύθυνση(δηλαδή στην διεύθυνση 0)

Τέλος, δημιουργήθηκε άλλο ένα σετ περισσότερων εντολών, το **anafora3.data**, **το οποίο αποτελείται από όλες τις δυνατές εντολές του MIPS που δεν δοκιμάστηκαν στα 2 προηγούμενα αρχεία**. Εκτελούνται οι παρακάτω εντολές:

```
00:addi r1,r0,5         ->r1=00000005(HEX)
04:ori r2,r0,0000ABCD ->r2=0000ABCD(HEX)
08:li r3,FFFF           ->r3=FFFFFFFF(HEX)
12:nandi r4,r3,00FF     ->r4=FFFFFF00(HEX)
16:lui r5,ABFF          ->r5=ABFF0000(HEX)
20:add r6,r4,r5         ->r6=ABFEFF00(HEX)
24:sub r7,r3,r4         ->r7=000000FF(HEX)
28:and r8,r3,r2         ->r8=0000ABCD(HEX)
32:or r9,ro,r2          ->r9=0000ABCD(HEX)
36:not r10,r4           ->r10=00FFFFFF(HEX)
40:nand r11,r3,r10      ->r11=FFFFFF00(HEX)
```

```
44:nor r12,r0,r10       ->r12=FFFFFF00(HEX)
48:sra r13,r10          ->r13=0000007F(HEX)
52:srl r14,r10          ->r14=0000007F(HEX)
56:sll r15,r10          ->r15=000001FE(HEX)
60:ror r7,r7            ->r7=0000001FE(HEX)
64:rol r10,r10          ->r10=8000007F(HEX)
68:sb r7,4(r0)          ->RAM[1025]=000000FE(HEX)
72:lb r8,4(r0)          ->r8=000000FE(HEX)
76:sw r7,8(r0)          ->RAM[1025]=000001FE(HEX)
80:lw r9,8(r0)          ->r9=000001FE(HEX)
```



Τέλος, λόγω περιορισμένου χρονικού ορίου που υπήρξε δεν πρόλαβα να δημιουργήσω test benches και να κάνω έλεγχο ορθότητας για τον επεξεργαστή pipeline, παρά μόνο ολοκλήρωσα την σχεδίαση του.

Συμπεράσματα

Μετά από την διεξαγωγή της εργασίας αυτής, έμαθα πως ακόμα και ένα ολοκληρωμένο κύκλωμα, χρησιμοποιώντας πολλές διαφορετικές προσεγγίσεις επίλυσης ,διαχωρίζοντας το πρόβλημα σε περισσότερα στάδια και κρατώντας τα βασικά του σημεία και αφαιρώντας ή προσθέτοντας νέα κομμάτια, πάντα έχει περιθώρια βελτιστοποίησης. Βέβαια, κάθε μια από τις παραλλαγές των επεξεργαστών, είναι καλύτερη ή χειρότερη ανάλογα με τις απαιτήσεις του σχεδιαστή στον τομέα της ελαχιστοποίησης του χρόνου επεξεργασίας των εντολών, στον παράγοντα του κόστους του hardware δημιουργίας μιας σχεδίασης ,αλλά φυσικά και στον βαθμό απλότητας ή περιπλοκότητας της δημιουργίας τους.

Κώδικας : *Σε αυτό το σημείο δίνονται κάποια σημαντικά και λεπτά σημεία του κώδικα της άσκησης.*

```
decode_execution_control_registers:DEC_EX_control_regs
port map(--we concrete the signals needed to pass to the next stages in vectors:
         --2 bit vector going to control WRITE BACK registers from stage ot stage
         --RF_WrData_sel(1) & RF_WrEn(0)
         DEC_EX_WB_in=>RF_WrData_sel & RF_WrEn,
         --5 bit vector going to control MEMORY registers from stage to stage
         --Branch_sig(4..2) & MEM_WrEn(1) & ByteOp(0)
         DEC_EX_MEM_in=>(Branch_sig & MEM_WrEn & ByteOp),
         --5 bit vector going to control EXECUTION registers from stage to stage
         --ALU_func(4..1) & ALU_Bin_sel(0)
         DEC_EX_EX_in=>ALU_func & ALU_Bin_sel,
```

Τρόπος με τον οποίο εισάγονται οι έξοδοι της μονάδας ελέγχου control_pipeline στους 3 control register, συνενώνονται τα σήματα ελέγχου και φτιάχνουν 1 διάνυσμα 2bits και 2 διανύσματα 5bits.