Οργάνωση Υπολογιστών
Εργασία#1:Σχεδίαση επεξεργαστή μονού κύκλου

Αναφορά

Γιάννης Περίδης

2018030069

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΑΡΙΝΌ ΕΞΆΜΗΝΟ 2021

Διδάσκων καθηγητής: Ιωαννίδης Σωτήριος

Υπεύθυνος εργαστηρίου: Παπαδημητρίου Κυπριανός

(για ότι έχει σημειωθεί με * ,θα ακολουθήσει συνοπτικός κώδικας)

Σκοπός εργαστηριακής άσκησης

Στην εργαστηριακή άσκηση αυτή, κλήθηκα να κατασκευάσω από την αρχή έναν πλήρης λειτουργικό επεξεργαστή μονού κύκλου και και ένα module μνήμης RAM που θα συνδεθεί και θα εξεταστεί μαζί με τον επεξεργαστή. Η άσκηση χωρίστηκε σε 3ις φάσεις όπου η κάθε μια είχε διαφορετικές απαιτήσεις και αποσκοπούσε να με εξοικειώσει σε διαφορετικές τεχνικές υλοποίησης κώδικα περιγραφής hardware και σε θεωρητικές γνώσεις ,πάνω στην κατανόηση της λειτουργίας του επεξεργαστή.

Ο σκοπός της 1ης φάσης, ήταν η σχεδίαση μιας μονάδα αριθμητικών(ALU) και λογικών πράξεων και ενός αρχείου καταχωρητών(RF) και ύστερα η προσομοίωση της λειτουργίας τους .Για την ολοκλήρωση της παραπάνω διαδικασίας, χρειάστηκαν βασικές γνώσεις πάνω στην σχεδίαση λογικών κυκλωμάτων ,σε κύριο βαθμό στην μορφή data flow(πύλες κλπ) και behavioral(μορφή with-select κλπ) ,δηλαδή όχι τόσο στην ένωση τους αλλά στην δημιουργία τους από την αρχή.

Ο σκοπός της 2ης φάσης, ήταν ο ορισμός μιας αρχιτεκτονικής συνόλου εντολών(ISA) και η σχεδίαση τεσσάρων ξεχωριστών βαθμίδων, ανάκλασης εντολών(IFSTAGE),αποκωδικοποίησης εντολών(DECSTAGE),εκτέλεσης εντολών(EXSTAGE) και πρόσβασης μνήμης(MEMSTAGE) και η υλοποίηση μιας κύριας μνήμης(RAM).Για την εκπλήρωση της παραπάνω διαδικασίας χρειάστηκε μια μερική κατανόηση της κωδικοποίησης των εντολών του συνόλου αρχιτεκτονικής και η απόλυτη κατανόηση της λειτουργικότητας και της δομής της μνήμης .Επίσης, σημαντική ήταν η αντίληψη των εννοιών σύγχρονο και ασύγχρονο σήμα και κύκλωμα, με σκοπό την παραγωγή των εξόδων στις σωστές χρονικές στιγμές σε σχέση με τις εισόδους η το ρολόι.

Ο σκοπός της 3ης φάσης, ήταν η ολοκλήρωση και ο έλεγχος του DATAPATH του επεξεργαστή(δηλαδή η σύνδεση των τεσσάρων βαθμίδων) και η δημιουργία του CONTROL που παράγει τα απαραίτητα σήματα ελέγχου. Ύστερα γίνεται η σύνδεση αυτών των δύο και της μνήμης στο PROC_SC για να ολοκληρωθεί η δημιουργία του επεξεργαστή ενός κύκλου. Για την ολοκλήρωση της παραπάνω διαδικασίας ήταν απαραίτητη η απόλυτη κατανόηση του πως επικοινωνούν τα τέσσερα στάδια μεταξύ τους, αλλά και πως επικοινωνούν σε σχέση με την μνήμη, με σκοπό να συνδεθούν σωστά και να δημιουργήσουν τα datapth και proc_sc. Όσον αφορά τη δημιουργία του control ,αλλά και την επιβεβαίωση της λειτουργίας του επεξεργαστή, ήταν υποχρεωτική η κατανόηση πλέον σε βάθος της κωδικοποίησης των εντολών του συνόλου αρχιτεκτονικής και η γνώση βασικών εντολών τύπου MIPS.

Προεργασία

Για την σχεδίαση ολόκληρης της εργασίας χρησιμοποιήθηκε το εργαλείο της γλώσσας περιγραφής υλικού VHDL, Xilinx ISE έκδοση 14.7,με το οποίο ήταν αναγκαίο να γίνει περαιτέρω εξοικείωση και εμβάθυνση .Ακόμη, με σκοπό να προχωρήσει ομαλά η εργασία και να μην υπάρχουν αβεβαιότητες, έγινε εκτός της παρακολούθησης των φροντιστηρίων του μαθήματος, επανάληψη από το εργαστηριακό υλικό στις έννοιες και στον κώδικα της Προχωρημένης Λογικής Σχεδίασης και εντρύφηση στην λειτουργία των εντολών MIPS και στο format τους. Τέλος για την σχεδίαση των block diagrams του DATAPATH και του PROC_SC χρησιμοποιήθηκε το λογισμικό DIA.

Περιγραφή

Το κύκλωμα που κατασκευάστηκε είναι ένας επεξεργαστής μονού κύκλου .Αυτό σημαίνει πως κάθε εντολή απαιτεί ακριβώς έναν κύκλο μέχρι να εκτελεστεί, δηλαδή μέχρι να περάσει από όλα τα απαραίτητα στάδια επεξεργασίας της, ανάλογα το είδος της. Επίσης, γίνεται

αντιληπτό πως δεν θα υπάρχουν επικαλύψεις εκτέλεσης εντολών.Η σχεδίαση που ακολουθήθηκε είναι ιεραρχική top-down, επομένως θα αναλυθούν τα υποκυκλώματα που δημιουργήθηκαν στην κάθε φάση και τέλος το συνολικό top-module.

1η Φάση

Στην 1η φάση, αρχικά δημιουργήθηκε η μονάδα αριθμητικών και λογικών πράξεων ALU,η οποία παίρνει δύο 32 bit τελεστέους και μια 4 bit πράξη και ανάλογα με αυτήν βγάζει σαν έξοδο το αποτέλεσμα της πράξης και 3 διαφορετικά flags που επισημαίνουν αν το αποτέλεσμα είναι 0, αν το αποτέλεσμα είχε κρατούμενο εξόδου 1 ή αν έχει υπερχειλίσει .Τα σήματα ελέγχου υπερχείλισης και κρατούμενου εξόδου ενεργοποιούνται μόνο στην πρόσθεση και την αφαίρεση. Όσον αφορά την *υπερχείλιση** στην πρόσθεση έχουμε υπερχείλιση αν και οι δύο προσθετέοι είναι ομόσημοι μεταξύ τους και ετερόσημοι με το άθροισμα, ενώ στην αφαίρεση αν ο μειωτέος είναι θετικός και ο αφαιρετέος αρνητικός και η διαφορά αρνητική ή το ανάποδο.
Να σημειωθεί πως έγινε χρήση των βιβλιοθηκών ieee.numeric_std.all *και* ieee.std_logic_unsigned.all, οι οποίες απλοποίησαν την σχεδίαση, καθώς και δεν χρειάστηκε η δημιουργία από την αρχή κυκλωμάτων για να κάνουν τις αριθμητικές πράξεις (πρόσθεση , αφαίρεση) μεταξύ διανυσμάτων.

Στην συνέχεια δημιουργήθηκαν τα απαραίτητα components έτσι ώστε να συνδεθούν και να σχηματίσουν το αρχείο καταχωρητών RF.Χρειάστηκαν ένας σύγχρονος καταχωρητής των 32 bit με reset (όταν είναι ενεργό στο 1,μηδενίζει τις εξόδους) και είσοδο ενεργοποίησης εγγραφής. Ένας απλός αποκωδικοποιητής που δέχεται σαν είσοδο ένα σήμα 5 bits και παράγει σαν έξοδο ένα σήμα 32 bits το οποίο είναι ανάλογα τις 32 πιθανές εισόδους ‘1’ μόνο σε ένα bit την φορά και τα υπόλοιπα 31 bit είναι ‘0’.Τέλος δημιουργήθηκε ένας πολυπλκέκτης που δέχονται σαν είσοδο 32 αριθμούς των 32 bit, σαν είσοδο επιλογής ένα σήμα 5 bit και βγάζει σαν έξοδο έναν 32 bit αριθμό .
Το RF, περιέχει 2 εισόδους ασύγχρονης ανάγνωσης και τις αντίστοιχες εξόδους που διαβάστηκαν, 1 είσοδο σύγχρονης εγγραφής και την αντίστοιχη είσοδο εισαγωγής δεδομένων, σήμα ενεργοποίησης εγγραφής, reset και ρολόι. Για την δημιουργία του, χρειάστηκαν 32 καταχωρητές των οποίων τα instances δημιουργήθηκαν με *for-generate**,οι οποίοι δέχονται όλοι τα ίδια δεδομένα εγγραφής, αλλά κάθε φορά μόνο ένας από αυτούς έχει ενεργοποιημένη την άδεια εγγραφής του. Προσοχή χρειάζεται να επισημανθεί στο γεγονός ότι ο καταχωρητής μηδέν (R0) πρέπει να έχει πάντα την τιμή 0. Τέλος, οι 32 bit έξοδοι των καταχωρητών αποθηκεύονται σε έναν πίνακα που αποτελείται από 32 θέσεις που η κάθε μια περιέχει μια από τις εξόδους.Έπειτα επιλέγεται από τις 2 δεδομένες διευθύνσεις, ποιές από αυτές θα διαβαστούν και θα είναι οι έξοδοι ανάγνωσης.

2η Φάση

Η 2η φάση αποτελείται από 4 στάδια και την μνήμη RAM.
Το 1ο στάδιο είναι η βαθμίδα ανάκλασης εντολών(IFSTAGE).
Σε αυτό το στάδιο διαβάζουμε μια εντολή από την μνήμη και συγκεκριμένα από το instruction segment, που βρίσκεται μεταξύ των θέσεων 0 έως 1024 και την φορτώνουμε σε έναν καταχωρητή , τον Program Counter, ο οποίος σε κάθε κύκλο περιέχει την διεύθυνση της εντολής που εκτελείται την συγκεκριμένη χρονική στιγμή. Η διεύθυνση αυτή χρησιμοποιείται στον υπολογισμό της επόμενης εντολής που θα φορτωθεί στον PC.Αν η επόμενη εντολή που θα έρθει είναι brunch,brunch equal ή brunch not equal με συνθήκη που ικανοποιείται , τότε ο PC παίρνει την τιμή **{ PC+4+4*immediate** }και πηγαίνει στην διεύθυνση αυτή, αλλιώς αν είναι μια άλλη οποιαδήποτε εντολή ο PC προχωράει στην επόμενη εντολή και παίρνει την

τιμή {**PC+4**}.Αν ο PC ξεπεράσει το instruction segment της RAM δηλαδή την τιμή 1024, τότε μηδενίζεται και πλέον δεν μπορούμε να φορτώσουμε άλλες εντολές .Πρέπει να σημειωθεί πως η μνήμη(RAM) δέχεται σαν είσοδο τα 12 έως 2 bits του PC, ώστε να λάβει είσοδο με διεύθυνση λέξης και όχι byte.

Το 2ο στάδιο είναι η βαθμίδα αποκωδικοποίησης εντολών(DECSTAGE).

Σε αυτό το στάδιο η εντολή που φορτώνεται από την μνήμη αποκωδικοποιείται, με σκοπό την εγγραφή και την προσπέλαση στο αρχείο καταχωρητών RF,όπως και την επέκταση της τιμής immediate.Ο καταχωρητής ανάγνωσης 1 είναι πάντα ο καταχωρητής **rs** ενώ ανάλογα με το format της εντολής ο καταχωρητής ανάγνωσης 2 είναι είτε ο **rt** είτε ο **rd** ,ενώ ο **rd** είναι πάντα ο καταχωρητής εγγραφής .Ακόμη, το **immediate** επεκτείνεται από 16 bits σε 32 bits μέσω ενός converter, με 4 διαφορετικούς τρόπους

- 00/ Γέμισμα με μηδενικά
- 01/ Επέκταση πρόσημου
- 10/ Γέμισμα με μηδενικά και λογική ολίσθηση αριστερά κατά 16
- 11/ Επέκταση πρόσημου και λογική ολίσθηση αριστερά κατά 2

Τέλος, επιλέγεται η εισαγωγή δεδομένων προς εγγραφή είτε από την μνήμη RAM σε περίπτωση εντολών load word ή load byte, είτε στις υπόλοιπες εντολές από την έξοδο της ALU.

Το 3ο στάδιο είναι η βαθμίδα εκτέλεσης εντολών(EXSTAGE).

Στο στάδιο αυτό γίνεται η εκτέλεση της εντολής και ο υπολογισμός του αποτελέσματος της ALU.Το 1ο όρισμα της , θα είναι σίγουρα τα δεδομένα ανάγνωσης του καταχωρητή 1, ενώ το δεύτερο όρισμα της θα είναι είτε τα δεδομένα ανάγνωσης του καταχωρητή 2, είτε το επεκταμένο immediate.

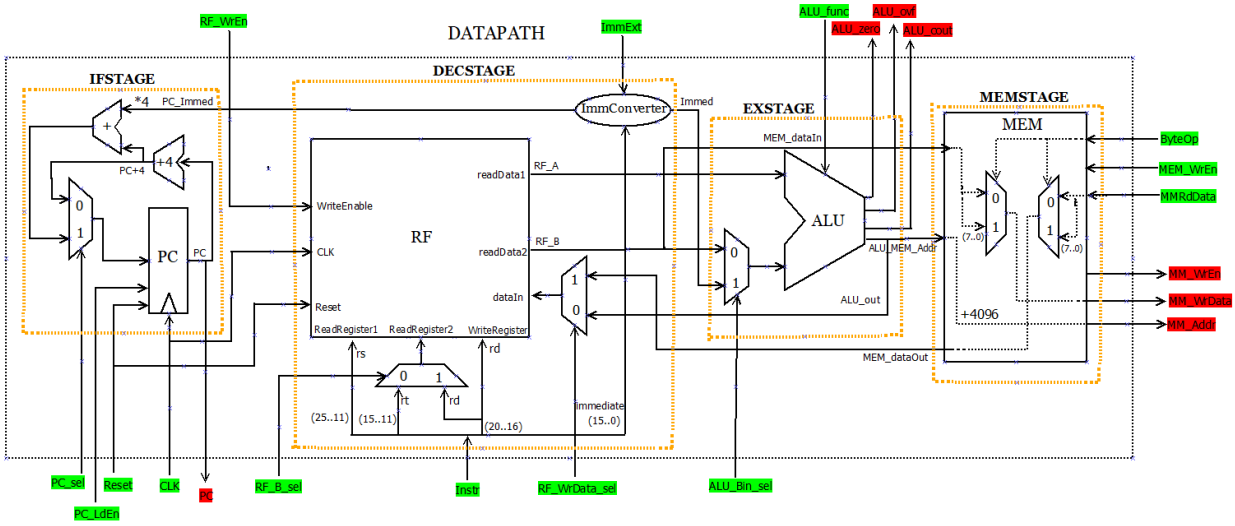
Το 4ο και τελευταίο στάδιο είναι η βαθμίδα προσπέλασης της μνήμης(MEMSTAGE).

Το στάδιο αυτό είναι υπεύθυνο για την αλληλεπίδραση του αρχείου καταχωρητών και της μνήμης RAM. Χρησιμοποιείται για φόρτωση δεδομένων από την μνήμη στο RF και για εγγραφή δεδομένων από το RF προς την μνήμη, δηλαδή στις εντολές store και load.Το στάδιο αυτό, είναι επίσης υπεύθυνο να φορτώσει ότι ζητάτε από την εντολή word(4 bytes) ή byte(το τελευταίο byte της λέξης).

Στην εργασία αυτή συμβαίνει η σύμβαση πως οι διευθύνσεις της μνήμης RAM χωρίζονται σε δύο μέρη το **instruction segment** από την θέση **0 έως 1023** words, το οποίο περιέχει τις εντολές και το **data segment** από την θέση **1024 έως 2047** words.Επομένως, ορίζοντας κατά αυτόν τον τρόπο τα στάδια πρέπει να προσθέσω **4096 offset** έτσι ώστε να είναι στην μέση χωρισμένα τα στάδια της μνήμης, αν πρόσθετα 1024 , το στάδιο των εντολών θα τελείωνε αντί στην 1023, στην 255 θέση.

3η Φάση

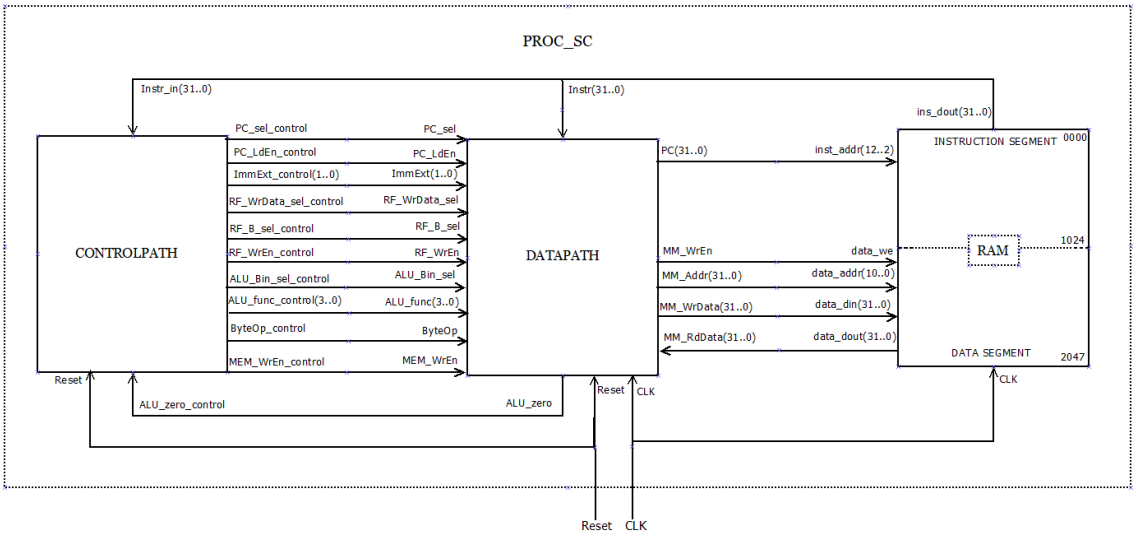
Στην 3η φάση ενώθηκαν όπως φαίνεται στο παρακάτω block diagram, τα 4 στάδια αυτά και δημιούργησαν το DATAPATH,το οποίο αποτελεί το κομμάτι του επεξεργαστή που είναι υπεύθυνο να πραγματοποιήσει όλες τις λειτουργίες του επεξεργαστή, ανάλογα με την κάθε διαφορετική εντολή που δίνεται.



Block Diagram of DATAPATH

Στην συνέχεια δημιουργήθηκε το CONTROL το οποίο είναι υπεύθυνο να δημιουργήσει και να στείλει τα κατάλληλα σήματα στο DATAPATH έτσι ώστε αυτό να λειτουργήσει όπως το κατευθύνει αυτό .Παίρνει σαν εισόδους την εντολή, το reset και το *σήμα ελέγχου μηδενικής εξόδου** από την ALU ,έτσι ώστε να μπορεί να διαχειριστεί τις εντολές διακλαδώσεων brunch equal και brunch not equal, ενώ παράγει σαν εξόδους όλες τις εισόδου του datapath.

Τέλος, συνενώθηκαν τα παραπάνω modules και η μνήμη RAM(το instruction και το data segment μαζί) στο υψηλότερο top-level αρχείο της εργασίας το PROC_SC, το οποίο θα τρέξουμε για να επιβεβαιώσουμε την συνολική λειτουργία του επεξεργαστή μονού κύκλου. Φαίνεται παρακάτω στο block diagram, ο τρόπος που συνδέονται τα modules.Δεν έχει εξόδους και ούτε εισόδους παρά μόνο το ρολόι και το reset.



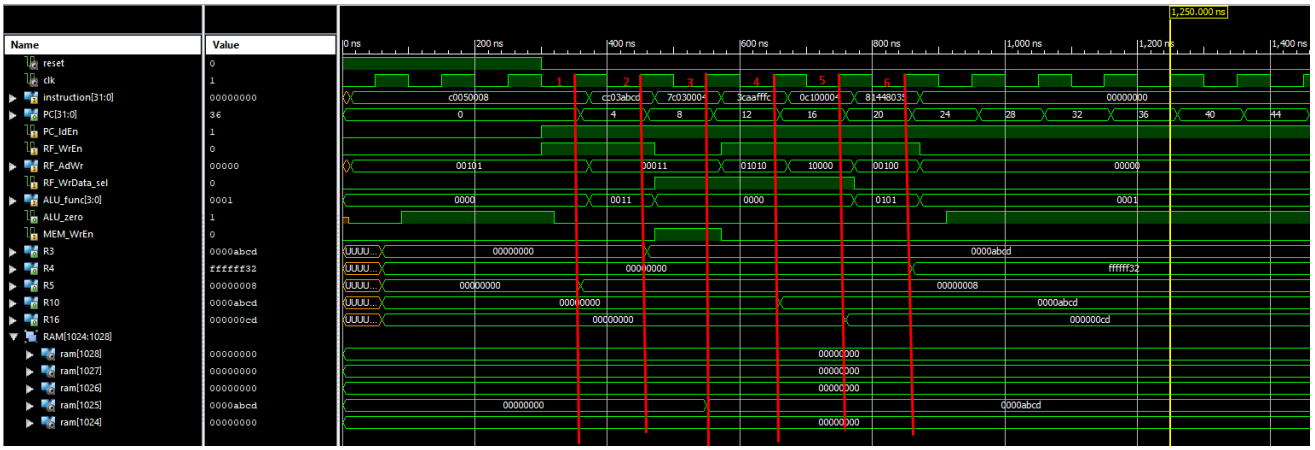
Block Diagram PROC_SC

Κυματομορφές-Προσομοίωση

Παρακάτω παρουσιάζονται οι κυματομορφές του του επεξεργαστή μονού κύκλου συνολικά PROC_SC.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 των προγραμμάτων 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
```

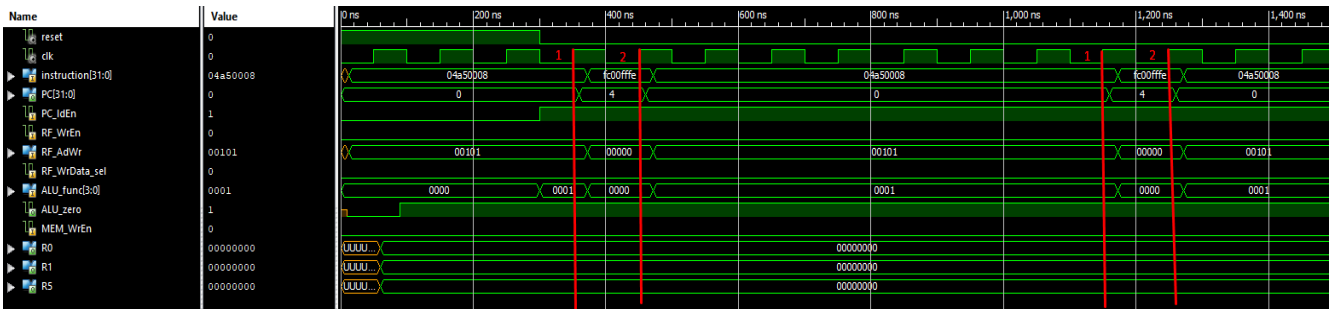


Αρχικά για τους πρώτους 3 κύκλους το σήμα reset είναι ενεργό,οποότε όλες οι έξοδοι μηδενίζουν. Ύστερα, εκτελείται 1 εντολή σε κάθε κύκλο ρολογιού και έτσι ο PC αυξάνεται κατά 4 σε κάθε κύκλο. Παρατηρώ ακόμα πως λόγο των καθυστερήσεων που έχουν προστεθεί στα submodules του κυκλώματος, η έξοδος καθυστερεί 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 στην διεύθυνση λέξης μνήμης 1025(η $2^{η}$ διεύθυνση λέξης του data segment) 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), δηλαδή το CD(hex) στον R10
6. Γίνεται πράξη NAND μεταξύ R10 και R16=>0000ABCD nand 000000CD=FFFFFF32

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 1^{η} εντολή .Επομένως μπαίνουμε σε έναν ατέρμον βρόγχο μεταξύ των 2 αυτών εντολών και η εντολή 3 δεν εκτελείται ποτέ

Συμπεράσματα

Μετά από την διεξαγωγή της εργασίας αυτής, έμαθα να ψάχνω και να διαχειρίζομαι βιβλιοθήκες και πακέτα της VHDL,με σκοπό την διευκόλυνσή μου και την ελαχιστοποίηση των modules μου. Έγινε κατανοητή η ανάγκη για εξαντλητική δοκιμή στα testbenches όλων και των ακραίων περιπτώσεων στα κυκλώματα που υλοποιήθηκαν, με σκοπό την ορθή λειτουργία τους σε όλες τις πιθανές συνθήκες. Ενώ παράλληλα κατάλαβα πως για την αναλυτική επιβεβαίωση λειτουργίας ενός ολοκληρωμένου κυκλώματος, χρειάζεται να χειρίζεσαι με άνεση το εργαλείο προσομοίωσης κυματομορφών εισόδων, εξόδων και εσωτερικών σημάτων, και ταυτόχρονα απαιτείται εξίσου πολύς χρόνος, αν όχι παραπάνω για την αποσφαλμάτωσή του κυκλώματος παρά για την γραφή του κώδικα που το υλοποιεί. Τέλος , το κυριότερο πράγμα που αποκόμισα ήταν το πώς να κάνω top-down ιεραρχική σχεδίαση, σπάζοντας ένα σύνθετο πρόβλημα σε μικρότερα,ενώνοντας τα κομμάτια του κυκλώματος sub-modules που είχα ήδη δημιουργήσει στα ζητούμενα συνολικά στάδια top-modules.

Κώδικας

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

```
--different cases that we have overflow in addision and substraction
--addition ovf: when A>0,B>0 and A+B<0 OR when A<0,B<0 and A+B>0
--subtraction ovf: when A>0,B<0 and A-B<0 OR when A<0,B>0 and A-B>0
Ovf <='1' after 10ns when ((A(31)='0' AND B(31)='0' AND Op="0000" AND (Output_top(31)='1' )) OR
                           (A(31)='1' AND B(31)='1'  AND Op="0000" AND (Output_top(31)='0' )) OR
                           (A(31)='0' AND B(31)='1'  AND Op="0001" AND (Output_top(31)='1' )) OR
                           (A(31)='1' AND B(31)='0'  AND Op="0001" AND (Output_top(31)='0' )))else '0' after 10ns;
```

** 1) Τρόπος εύρεσης σήματος υπερχείλισης στην ALU **

```
--port mapping of 32 registers , using for generate
registers32x32:
for i in 0 to 31 generate
registers:REG
port map(CLK=>CLK,
         RST=>RST,
         --here we store every write enable bit in a vector
         WE=>WE_top(i),
         Datain=>Din,
         --here we store every registers output in the array
         Dataout=>Dataout_top(i));
end generate registers32x32;
```

** 2) Τρόπος δημιουργίας των instances των 32 καταχωρητών με for-generate.To write enable_top είναι εσωτερικό σήμα 32 bit vector, ενώ το Dataout_top είναι εσωτερικό σήμα, πίνακας 32χ32 στο RF **

```
ALU_func_control<="0001";

--if the result of the substraction is 0 then the alu_zero will be 1
if(ALU_zero_control='1') then
PC_sel_control<='1';
--else if the result of the substarction is not 0,alu_zero will be 0
else
PC_sel_control<='0';
end if;
PC_LdEn_control<='1';
```

3)Τρόπος δημιουργίας της εντολής brunch equal, αφαίρεση των καταχωρητών RF[rd]-RD[rd] , αν η διαφορά είναι 0 (δηλαδή το zero flag ενεργό)τότε κάνε brunch, αλλιώς όχι. Αντίστοιχα ισχύει και για την bne στο CONTROL