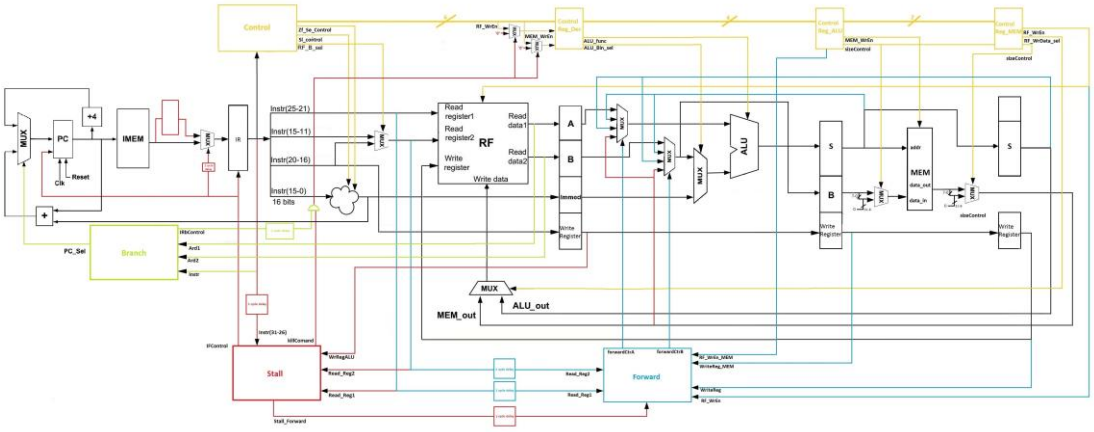
2/5/2017

Αναφορά Εργαστηρίου 5

LAB31231454

Μανώλης Πετράκος
Μιχαήλ Δακανάλης

Προεργασία



Περιγραφή Άσκησης

Το Pipeline είναι χωρισμένο σε 5 βαθμίδες. Το Instruction Fetch, το Decode, το ALU stage, το MEM stage και το Register Store. Ο διαχωρισμός των βαθμίδων γίνεται με την χρήση καταχωρητών για κάθε σήμα που θέλουμε να διατηρήσουμε στην επόμενη βαθμίδα. Επιτυγχάνεται παραλληλία και εκτελούνται 5 εντολές ταυτόχρονα. Όμως δημιουργούνται hazards που επιλύονται με την χρήση stalls και forwards.

Το **Control Unit** είναι ασύγχρονο και παράγει ,στο στάδιο του Decode, απευθείας όλα τα σήματα ελέγχου μέσω ενός πίνακα αληθείας. Εφόσον χρειάζονται τα σήματα, καθυστερούνται παράλληλα με το Datapath ώστε να χρησιμοποιηθούν στην κατάλληλη βαθμίδα. Τα σήματα μπορούν να χρησιμοποιηθούν σε πάνω από μία βαθμίδες.

Το **Forward Unit** είναι ένα ασύγχρονο κύκλωμα με σκοπό την επίλυση των περισσοτέρων RAW hazards. Όταν ένα πρόγραμμα θέλει να διαβάσει έναν καταχωρητή τον οποίο θέλει να γράψει σε μία από τις προηγούμενες δύο εντολές δεν έχει προλάβει το Pipeline να βάλει τη σωστή τιμή στο Register File. Τότε πρέπει η καινούργια τιμή να κάνει Forward, δηλαδή να προσπεράσει μία με δύο βαθμίδες. Για να αναγνωριστεί ένα τέτοιο hazard πρέπει πρώτα να κοιτάξουμε του καταχωρητές εγγραφής και τα RF_WrEn των βαθμίδων MEM stage και Register Store. Συγκρίνονται με του καταχωρητές ανάγνωσης της βαθμίδας ALU και αν οι καταχωρητές είναι ίδιοι και το RF_WrEn είναι 1 πρέπει να γίνει forward. Τέλος με την χρήση πολυπλεκτών η ALU παίρνει τις σωστές τιμές από την κατάλληλη βαθμίδα.

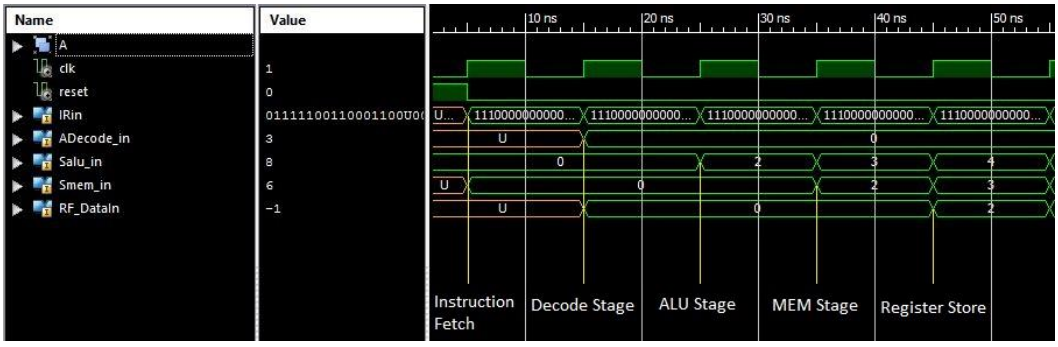
Το **Stall Unit** είναι ένα ασύγχρονο κύκλωμα με σκοπό την επίλυση των RAW hazards συνεχόμενων εντολών στην μνήμη δεδομένων. Όταν μία εντολή θέλει να διαβάσει μία

διεύθυνση μνήμης και η αμέσως επόμενη θέλει να χρησιμοποιήσει το αποτέλεσμα υπάρχει hazard. Σε αντίθεση με τα άλλα hazards δεν υπάρχει το αποτέλεσμα πριν την Register Store βαθμίδα και δεν γίνεται διπλό forward. Αναγκαστικά πρέπει να γίνει ένα Stall, δηλαδή να “σκοτώσουμε” μια εντολή και να επαναληφθεί στον επόμενο κύκλο. Αν η εντολή που βρίσκεται στο ALU stage είναι load συγκρίνεται ο καταχωρητής εγγραφής της με του καταχωρητές ανάγνωσης της επόμενης. Σε περίπτωση ταύτισης των καταχωρητών “σκοτώνεται” η δεύτερη εντολή μηδενίζοντας τα σήματα ελέγχου της RF_WrEn και MEM_WrEn. Έτσι συνεχίζει χωρίς το αποτέλεσμά της να αποθηκεύεται κάπου. Συγχρόνως σταματάμε τη λειτουργία ανάκλησης εντολών ώστε να επαναληφθεί η δεύτερη εντολή. Στον επόμενο κύκλο γίνεται προώθηση του αποτελέσματος της μνήμης στην ALU. Σταματώντας την λειτουργία ανάκλησης εντολών δημιουργείται ένα πρόβλημα όπου χάνεται η επόμενη εντολή από αυτές που δημιούργησαν το hazard. Αυτό γίνεται γιατί όταν σταματάει ο IR η μνήμη εντολών βγάζει την επόμενη. Η ενδιάμεση εντολή που χάνεται αποθηκεύεται προσωρινά σε ένα καταχωρητή και περνάει στον IR στον επόμενο κύκλο.

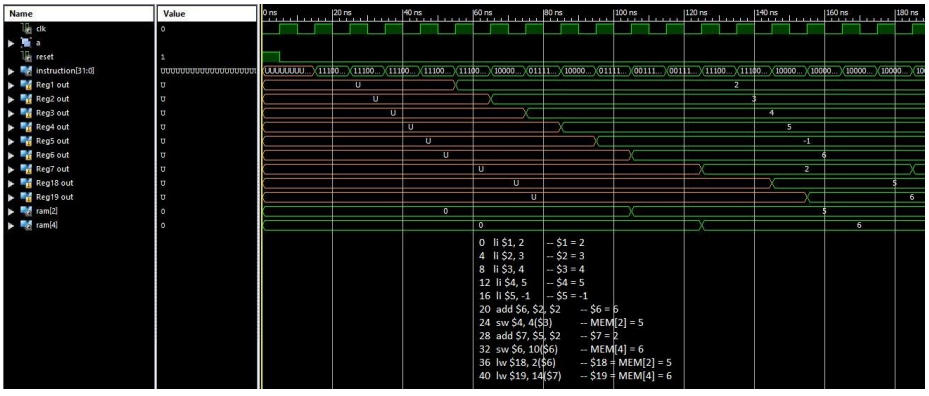
Το **Branch Unit** είναι και αυτό ένα ασύγχρονο κύκλωμα για την εξυπηρέτηση των εντολών branch. Αν η εντολή που βρίσκεται στο Decode Stage είναι b ή πετυχημένη beq, bne τότε προστίθεται στον PC ο immediate. Εκτελείται η αμέσως επόμενη από το branch εντολή, όπως στον MIPS, και “σκοτώνεται” η μεθεπόμενη. Δηλαδή δεν εκτελείται αυτή που έδειχνε ο PC όταν το branch ήταν στο decode stage. Τέλος το πρόγραμμα συνεχίζει κανονικά την εκτέλεση από την νέα διεύθυνση. Η σύγκριση δεν γίνεται στην ALU αλλά στο Branch Unit για να κερδίσουμε ένα κύκλο.

Κυμματομορφές

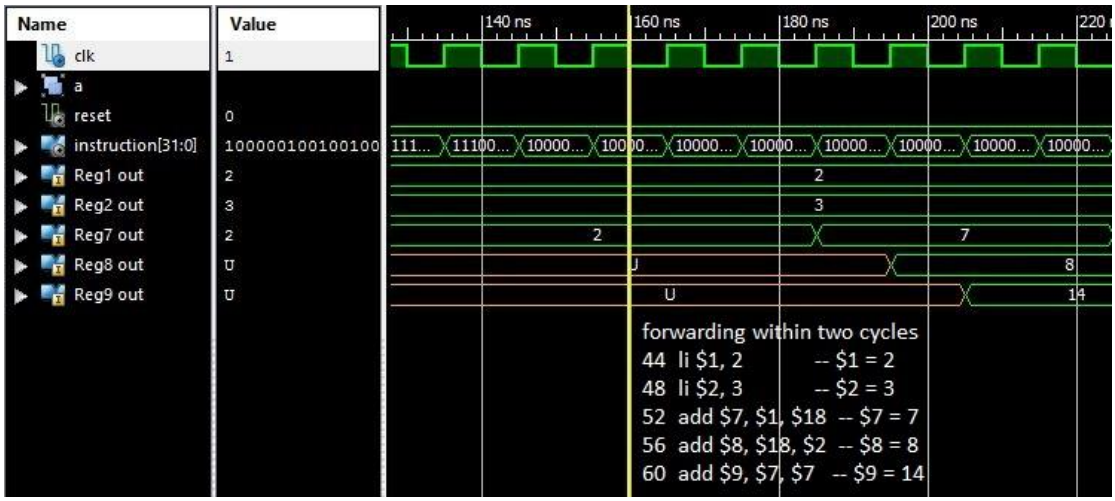
Pipeline Fill(Πορεία Πρώτης εντολής)



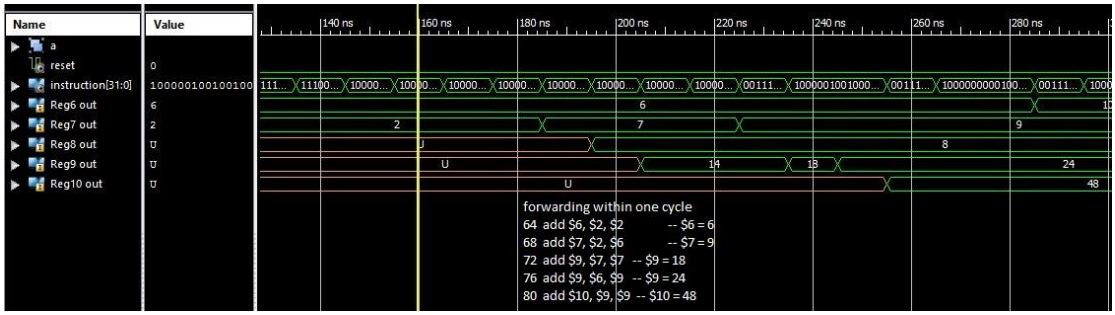
Λειτουργία των εντολών li,add,lw,sw :



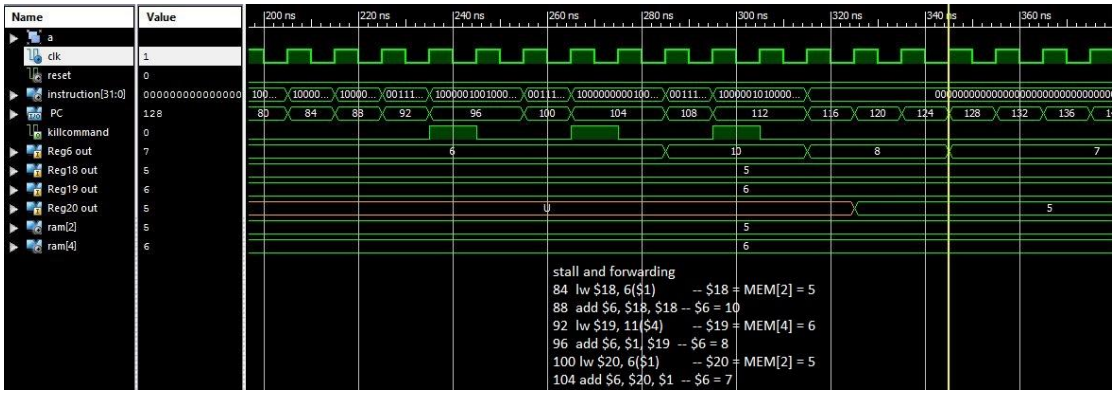
Forwarding μεταξύ δύο κύκλων. Οι καταχωρητές 1 και 2 είχαν γραφτεί από πριν με τις ίδιες τιμές γι’ αυτό δεν φαίνεται αλλαγή στις κυμματομορφές τους.



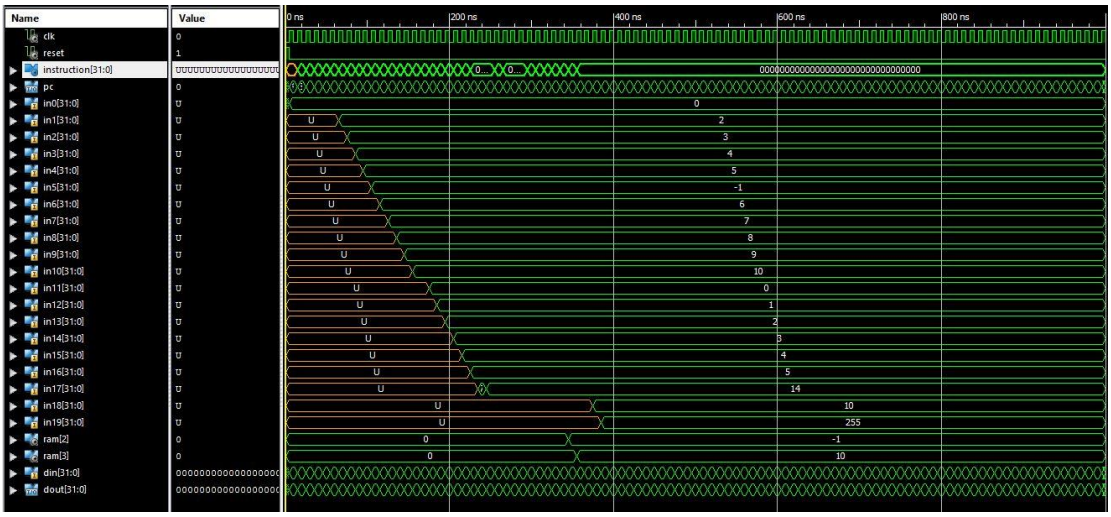
Forwarding ενός κύκλου:



Stall και Forward. Φαίνονται τα stall στο PC και στον IR καθώς κρατάνε δύο κύκλους αντί ένα. Επίσης φαίνεται και το σήμα killCommand που “σκοτώνει” τις εντολές.



Παρακάτω φαίνονται και όλες οι εντολές του Bonus εκτός τις branch που δεν δουλεύουν σωστά.



Συμπεράσματα

Μάθαμε πως να κατασκευάζουμε Pipeline επεξεργαστή και να αντιμετωπίζουμε hazards.

Επειδή οι branch δεν δούλευαν, αφαιρέσαμε τον κώδικα και απλά περιγράψαμε τι σχεδιάσαμε.

Μερικές εντολές που δημιουργούσαν hazards είχαν ήδη γίνει πριν με αποτέλεσμα να μην μπορούμε να τις τεστάρουμε σωστά. Έπρεπε να αλλάξουμε τον κώδικα του rom.data για να δούμε ότι τρέχει σωστά.

Π.χ. η εντολή με pc 44 είχε ξαναγίνει με pc 0 και δεν φαινόνταν το αποτέλεσμα του hazard αν είχαμε λάθος.