7ο Εργαστήριο Αρχιτεκτονικής Η/Υ: Υλοποίηση λογικής προώθησης και καθυστέρησης διοχετευμένου επεξεργαστή

Α. Ευθυμίου

Παραδοτέο: Τρίτη 19 Απρίλη 2015, 23:00

Ο σκοπός αυτής της άσκησης είναι η εμβάθυνση της κατανόησης λειτουργίας ενός διοχετευμένου επεξεργαστή, χρησιμοποιώντας τα εργαλεία Quartus και Modelsim.

Σας δίνεται ένας διοχετευμένος επεξεργαστής, παρόμοιος με αυτόν του συγγράμματος, με την τεχνική μείωσης καθυστέρησης των διακλαδώσεων. Επιπλέον υπάρχουν οι επεκτάσεις εντολών που είχαν γίνει και στον επεξεργαστή της προηγούμενης άσκησης (j, lui, ori, addi, addiu.) Ο επεξεργαστής έχει έτοιμη την «υποδομή» για προώθηση δεδομένων προς το στάδιο εκτέλεσης (και προσπέλασης μνήμης), αλλά η λογική ελέγχου προώθησης (forwarding) και καθυστέρησης (stall) δεν έχει υλοποιηθεί. Η δουλειά σας είναι να γράψετε, σε Verilog, αυτή τη λογική ελέγχου.

Για να κάνετε την άσκηση **είναι εξαιρετικά σημαντικό να έχετε μελετήσει τα μαθήματα για την υλοποίηση του MIPS σε πέντε στάδια διοχέτευσης** που αντιστοιχούν μέχρι την ενότητα 4.8 του συγγράμματος. Παρακάτω γίνονται συχνές αναφορές σε τμήματα και εικόνες του συγγράμματος. Πρέπει επίσης να κάνετε μια γρήγορη επανάληψη στη γλώσσα Verilog και να θυμάστε βασικές αρχές ψηφιακής σχεδίασης.

Μή ξεχάσετε να επιστρέψετε τα παραδοτέα που αναφέρονται στο τέλος του κειμένου για να πάρετε βαθμό γι'αυτή την εργαστηριακή άσκηση!

1 Ο διοχετευμένος επεξεργαστής

Για να ξεκινήσετε θα χρειαστείτε όλα τα αρχεία του εργαστηρίου δίνοντας τις εξής εντολές:

```
git remote add lab07_starter https://github.com/UoI-CSE-MYY402/lab07_starter.git
git fetch lab07_starter
git merge lab07_starter/master -m "Fetched lab07 starter files"
```

Ξεκινήστε το Quartus σύμφωνα με το σύστημά σας. Ανοίξτε το Quartus project με όνομα, mips.qpf, που υπάρχει έτοιμο στα αρχεία του starter. Κάντε διπλό κλίκ στο mips για να δείτε το σχηματικό του επεξεργαστή. Εξερευνήστε το σχέδιο, δείτε τα περιεχόμενα των διαφόρων block/symbols και παρατηρήστε καλά τα ονόματα των σημάτων.

Οι καταχωρητές διοχέτευσης είναι χρωματισμένοι με κίτρινο χρώμα. Οι ακροδέκτες εισόδου-εξόδου τους είναι χωρισμένοι σε ομάδες: επάνω βρίσκονται τα σήματα που χρησιμοποιούνται στο αμέσως-επόμενο στάδιο και πιο κάτω τα σήματα που απλά μεταφέρονται σε πιο μακρινά στάδια. Χαρακτηριστικός είναι ο καταχωρητής διοχέτευσης μεταξύ των σταδίων ID και EX, όπου υπάρχουν 3 τέτοιες ομάδες σημάτων / ζευγών ακροδεκτών: επάνω βρίσκονται αυτά που καταλήγουν στο EX, στη μέση αυτά που μεταφέρονται στο στάδιο MEM και κάτω αυτά που προορίζονται για το στάδιο WB (μέσω του MEM βέβαια). Για να ξεχωρίζουν τα στάδια μεταξύ τους υπάρχουν σκούρες κόκκινες διακεκομμένες γραμμές επάνω και κάτω από κάθε καταχωρητή διοχέτευσης.

Για να φαίνονται καλύτερα τα μονοπάτια προώθησης, τα καλώδια που χρειάζονται σε προηγούμενα στάδια (πηγαίνουν προς τα αριστερά), έχουν διαφορετικό χρώμα ανάλογα με το στάδιο από το οποίο προέρχονται. Αυτά του σταδίου WB είναι κόκκινα, του σταδίου MEM είναι σκούρα πράσινα, του σταδίου EX ανοιχτό γαλάζιο και του σταδίου ID (διεύθυνση επόμενης εντολής σε περίπτωση διακλάδωσης ή άλματος) μαύρο.

Τα σήματα ελέγχου που είναι ενεργά σε κάποιο στάδιο έχουν γραμμές με ανοιχτό πράσινο χρώμα και το όνομα του σήματος είναι επίσης πράσινο. Σήματα ελέγχου που απλά μεταφέρονται σε άλλα στάδια

έχουν το συνηθισμένο σκούρο μωβ του Quartus. Εξαίρεση αποτελούν τα σήματα καθυστέρησης, stall, και εκκένωσης, flush, που έχουν ρόζ χρώμα επειδή έχουν επίδραση σε αρκετά στάδια ταυτόχρονα.

Πολλά σήματα χρειάζονται και μεταφέρονται σε περισσότερα από ένα στάδια, μέσω των καταχωρητών διοχέτευσης. Για να έχουν ξεχωριστά ονόματα και να φαίνεται καθαρά η χρήση τους, τα ονόματα τους ξεκινούν με ένα προθεμα που καθορίζει το στάδιο στο οποίο αντιστοιχούν ενώ το υπόλοιπο μέρος του ονόματος περιγράφει τη σκοπιμότητα του σήματος. Για παράδειγμα, το σήμα ελέγχου, που καθορίζει αν ο πολυπλέκτης του σταδίου WB που επιλέγει μεταξύ της εξόδου της μνήμης (για lw) ή της εξόδου της ALU (για αριθμητικές πράξεις), παράγεται στο στάδιο ID με το όνομα id_memToReg, συνεχίζει στο στάδιο EX με το όνομα ex_memToReg, μετά στο στάδιο MEM με όνομα mem_memToReg και, τέλος στο WB με όνομα wb_memToReg.

Με ροζ χρώμα έχει σημειωθεί και η μονάδα ελέγχου διοχέτευσης, pipe_ctrl, που παράγει τα σήματα προώθησης (forwardA, forwardB, ldstBypass), καθυστέρησης (stall) και εκκένωσης (flush). Αυτή είναι η μονάδα που θα πρέπει να αλλάξετε στην άσκηση αυτή.

Στον διοχετευμένο επεξεργαστή της άσκησης, η απόφαση για τη διακλάδωση παίρνεται στο στάδιο ID, όπως περιγράφεται στο σύγγραμμα στο τμήμα 4.8. Υπάρχουν όμως και μερικές ακόμη αλλαγές:

- Ενώ στο στάδιο EX μπορούν να γίνουν προωθήσεις δεδομένων, όπως περιγράφεται στο τμήμα 4.7 του συγγράμματος, **δεν γίνεται προώθηση δεδομένων προς το στάδιο ID**, εκτός από αυτήν που γίνεται από το στάδιο WB και υλοποιείται εσωτερικά στο αρχείο καταχωρητών όταν στον ίδιο κύκλο γίνεται εγγραφή και ανάγνωση του ίδιου καταχωρητή. Αν σε οποιοδήποτε από τα στάδια EX, MEM υπολογίζεται η τιμή ενός καταχωρητή τον οποίο διαβάζει μια εντολή beq που βρίσκεται στο στάδιο ID, θα χρειαστεί καθυστέρηση ενός ή δύο κύκλων.

- Υπάρχει πρόβλεψη για προώθηση μεταξύ συνεχόμενων lw, sw που χρησιμοποιούν τον ίδιο καταχωρητή δεδομένων: συγκεκριμένα αν γίνεται αντιγραφή δεδομένων από τη μνήμη έτσι ώστε η lw να διαβάζει το δεδομένο και η sw να το γράφει ξανά στη μνήμη (σε άλλη θέση). Αυτό αναφέρεται ως «επιπλέον ανάπτυξη» στο 4.7, σελ. 433 του συγγράμματος. Για το σκοπό αυτό χρησιμοποιείται το σήμα ελέγχου ldStBypass και έχει προστεθεί ένας πολυπλέκτης στο στάδιο MEM, που δεν υπάρχει στις εικόνες του συγγράμματος. Αν το σήμα ελέγχου έχει την τιμή 1, τα δεδομένα που γράφονται στη μνήμη προέρχονται από το στάδιο WB όπου βρίσκεται η προηγούμενη lw.

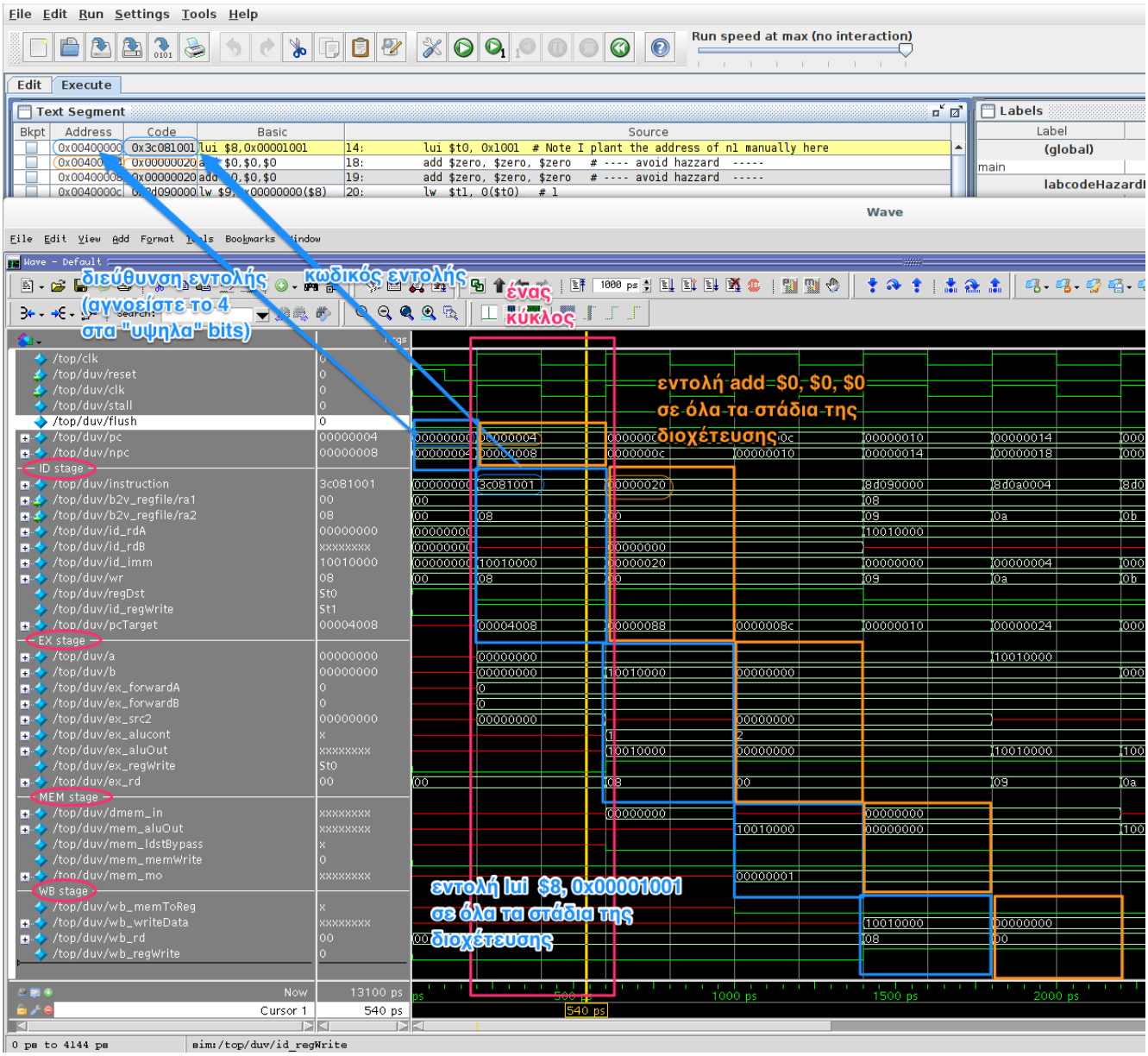
- Η «μονάδα προώθησης» αντί να βρίσκεται στο στάδιο EX (ή στο MEM για προώθηση μεταξύ lw, sw που αναφέρεται παραπάνω), βρίσκεται στο στάδιο ID και αποτελεί μέρος του ελέγχου διοχέτευσης, του ρόζ μπλοκ με όνομα pipe_ctrl. Έτσι όλες οι αποφάσεις παίρνονται στο στάδιο ID και τα σήματα ελέγχου μεταφέρονται μέχρι το κατάλληλο στάδιο. Αυτό είναι για λόγους ομοιομορφίας με ό,τι συμβαίνει στα υπόλοιπα σήματα ελέγχου, όπως, για παράδειγμα, στο memToReg που αναφέρθηκε παραπάνω.

  Για παράδειγμα, στο πρόγραμμα:

  ```
  add $t0, ... # no dependencies from above
  or  $t1, $t0, $zero
  ```

  θα πρέπει να γίνει προώθηση της τιμής του `$t0` από το στάδιο MEM, όπου θα έχει φτάσει η add, στο στάδιο EX όπου θα έχει φτάσει η or. Η απόφαση αυτή όμως θα ληφθεί όταν η or βρίσκεται ακόμα στο στάδιο ID και η add στο στάδιο EX, γιατί ο έλεγχος γίνεται στο στάδιο ID. Κοιτώντας το σχηματικό του mips στο Quartus, η μονάδα αυτή θα παράγει την τιμή `2'b10`[1] στο σήμα id_forwardA. Μετά από ένα κύκλο η εντολή or θα βρεθεί στο στάδιο EX και το παραπάνω σήμα θα περάσει στο ex_forwardA και θα επιλέξει την προωθημένη τιμή του `$t0` από το στάδιο MEM (σήμα mem_aluOut, σκούρο πράσινο).

[1] Ο αριθμός 2 χρησιμοποιώντας τον τρόπο αναπαράστασης της Verilog. Οι τιμές των σημάτων είναι ίδιες με του συγγράμματος και δίνονται στον πίνακα-εικόνα 4.55, σελ. 431.



Σχήμα 1: Κυματομορφή διοχετευμένου επεξεργαστή.

- Τέλος για απλοποίηση του σχηματικού και μεγαλύτερη ευελιξία, ο αποκωδικοποιητής εντολών της ALU (aludec στο lab06) έχει ενσωμματωθεί στον κύριο αποκωδικοποιητή (maindec).

2 Κατανόηση προσομοίωσης διοχετευμένου επεξεργαστή

Για να επαληθεύσετε ότι υλοποιήσατε σωστά τη μονάδα ελέγχου διοχέτευσης, θα χρειαστεί να προσομοιώσετε τον επεξεργαστή που θα τρέχει ένα προγράμμα που ενεργοποιεί όλους τους πιθανούς κινδύνους δεδομένων. Για να προσομοιώσετε προγράμματα που τρέχουν στον επεξεργαστή θα πρέπει να χρησιμοποιήσετε τον Mars όπως και στην προηγούμενη άσκηση (με dump memory). Επίσης πρέπει να ξεκινάτε την προσομοίωση με το Modelsim και να παρατηρείτε κυματομορφές, πάλι όπως στην προηγούμενη άσκηση. **Προσοχή όταν ξεκινήσετε το Modelsim, κλείστε το παλιό project και όταν**

ανοίξετε καινούριο σιγουρευτείτε ότι έχετε βάλει τον σωστό κατάλογο, lab07, στο παράθυρο νέου project. Επειδή πολλά ονόματα αρχείων είναι ίδια με το lab06 είναι εύκολο να μπερδευτείτε και να δουλέυετε σε λάθος κατάλογο.

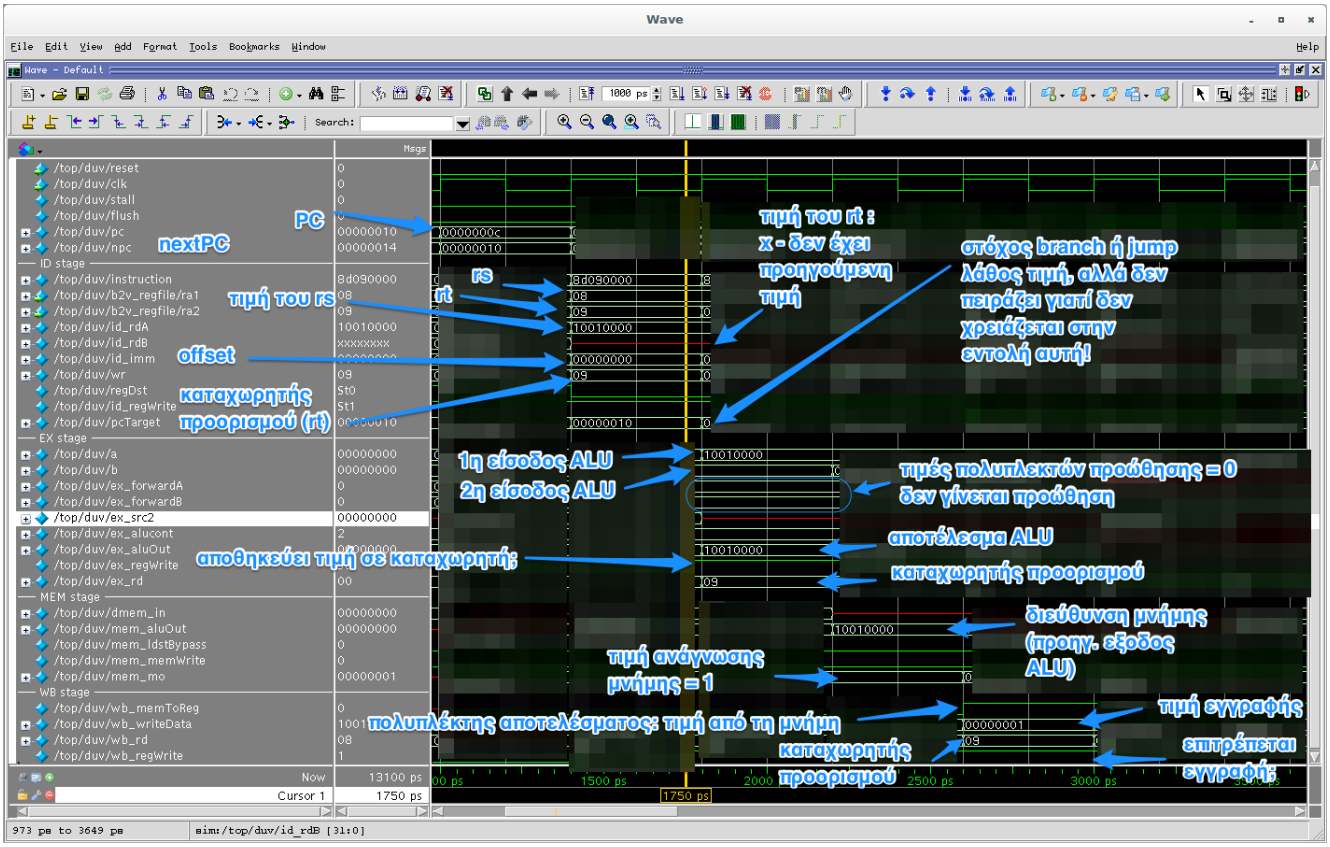
Το τελευταίο είναι κάπως δυσκολότερο γιατί η εκτέλεση μιας εντολής γίνεται είναι πλέον σε 5 κύκλους, υπάρχουν περισσότερα σήματα για να εξεταστούν και συμβαίνουν καθυστερήσεις (stall), εκκενώσεις (flush), κλπ.

Το σχήμα 1 δείχνει τις 2 πρώτες εντολές που εκτελούνται στον επεξεργαστή καθώς και τις αντιστοιχίσεις κώδικα μηχανής και διεύθυνσης εντολής μεταξύ MARS και κυματομορφής. Με μπλέ πλαίσιο φαίνεται η πρώτη εντολή (lui) καθώς προχωράει από στάδιο σε στάδιο σε κάθε κύκλο ρολογιού. Με πορτοκαλί φαίνεται η επόμενη εντολή (add). Στα αριστερά φαίνονται τα όνομα σημαντικών σημάτων ομαδοποιημένα ανά στάδιο διοχέτευσης.

Ενώ στην προηγούμενη άσκηση όλα τα σήματα μιας εντολής φαινόταν στον ίδιο κύκλο, αυτό δεν συμβαίνει τώρα. Έτσι, όπως δείχνει το δεξί βελάκι του σχήματος 1, για να δεί κανείς την κωδικοποίηση της εντολής (σήμα instruction) πρέπει να περιμένει μέχρι αυτή να φτάσει στο 2ο κύκλο εκτέλεσής της, στο στάδιο ID.

Χρειάζεται προσοχή για να μη χαθεί κανείς σε μια τέτοια κυματομορφή. Η διεύθυνση της εντολής και η κωδικοποίησή της βοηθούν ώστε να αναγνωρίσει κανείς την εντολή που εκτελείται κάθε φορά.

Το σχήμα 2 δείχνει την κυματομορφή της εντολής `lw $t2, 4($t0)` με τον καταχωρητή t0 να έχει την τιμή 0x1001000. Τα σήματα πριν και μετά την εντολή έχουν σβηστεί για να φαίνεται καθαρότερα η εκτέλεσή της. Στο σχήμα επισημαίνονται τα σημαντικότερα σήματα. Θα πρέπει να δείτε το σχηματικό



Σχήμα 2: Κυματομορφή εντολής lw $10, 0x04($8) - lw $t2, 4($t0).

στο Quartus για τα υπόλοιπα σήματα.

Στον κώδικα Verilog του επεξεργαστή έχουν προστεθεί καθυστερήσεις σε βασικά κυκλώματα (π.χ. ALU, αρχείο καταχωρητών, μνήμες), όπως και στην προηγούμενη άσκηση. Έτσι σε κάθε ακμή του ρολογιού οι νέες τιμές εμφανίζονται με μια καθυστέρηση. Αυτό **δεν φαίνεται** στα σχήματα γιατί τα screenshots πάρθηκαν από προηγούμενη έκδοση του κώδικα. Παρατηρήστε τις τιμές λίγο αργότερα, περίπου στην κατερχόμενη ακμή του ρολογιού, όταν οι τιμές θα έχουν σταθεροποιηθεί. Προσοχή όμως ο πρώτος κύκλος είναι «μισός»: όταν πέσει το σήμα reset στο 0, ο PC παίρνει την τιμή 0 και ξεκινάει την εκτέλεση της πρώτης εντολής, που ολοκληρώνεται πρίν την πρώτη ανοδική ακμή του ρολογιού. Από εκεί και έπειτα, κάθε εντολή ξεκινάει σε κάθε κύκλο.

Ως βοηθήματα δίνονται ένα αρχείο wave.do που όταν φορτωθεί στο Modelsim παρακολουθεί τα σήματα που φαίνονται στα προηγούμενα σχήματα, και ένα πρόγραμμα assembly, labcodeHazardFree.asm, για πειραματισμό. Στο labcodeHazardFree.asm θα δείτε κάποια σχόλια δίπλα σε εντολές που γράφουν τον καταχωρητή zero, με τη φράση: "avoid hazard". Αυτές οι εντολές έχουν προστεθεί ώστε να **μην** χρειάζονται προωθήσεις ή καθυστερήσεις μεταξύ εντολών, αφού η μονάδα ελέγχου διοχέτευσης δεν είναι πλήρης ακόμη. Αφού την συπληρώσετε μπορείτε να σβήσετε τις εντολές αυτές και να κάνετε ένα βασικό έλεγχο των καθυστερήσεων και προωθήσεων.

Πριν προχωρήσετε παρακάτω, τρέξτε με προσομοίωση αυτό το πρόγραμμα ως το τέλος και δείτε τι συμβαίνει σε κάθε κύκλο. Θα χρειαστεί να είστε εξοικιωμένοι με τα σήματα και τον διοχετευμένο τρόπο εκτέλεσης πριν να μπορέσετε να ολοκληρώσετε την μονάδα ελέγχου διοχέτευσης.

Για βαθμολόγηση θα χρησιμοποιηθεί ένα πιο αναλυτικό πρόγραμμα που εξετάζει περισσότερες περιπτώσεις. Αυτό δεν σας παρέχεται, γιατί θα μπορούσατε διαβάζοντάς το να βρείτε τις περιπτώσεις που πρέπει να εξετάσετε στον έλεγχο διοχέτευσης.

3 Μονάδα ελέγχου διοχέτευσης

Η μονάδα ελέγχου διοχέτευσης παράγει τα σήματα ελέγχου που αφορούν τη διοχέτευση. Από αυτά, τα σήματα εκκένωσης (flush), που χρησιμοποιείται για να αδειάσει το προηγούμενο στάδιο της διοχέτευσης (Instruction Fetch), και αλλαγής ροής (flowChange), που χρησιμοποιείται για να επιλέξει την επόμενη τιμή του PC, είναι ήδη έτοιμα. Το flush εξαρτάται και από το σήμα stall που πρέπει να υλοποιήσετε γιατί αν μια εντολή διακλάδωσης είναι σταματημένη (stalled) το δεύτερο σήμα, που δείχνει αν η διακλάδωση ακολουθείται, δεν είναι έγκυρο.

Τα σήματα που πρέπει να υλοποιήσετε είναι τα id_forwardA, id_forwardB, id_ldstBypass, και stall. Τα δύο πρώτα ελέγχουν τους πολυπλέκτες στις εισόδους της ALU που επιλέγουν μεταξύ της τιμής που προέρχεται από το αρχείο καταχωρητών, δηλαδή χωρίς να γίνει προώθηση, και τιμών που προέρχονται από τα στάδια MEM ή WB. Το id_ldstBypass ελέγχει τον πολυπλέκτη στην είσοδο δεδομένων προς εγγραφή στη μνήμη και επιλέγει μεταξύ της τιμής που προέρχεται από το αρχείο καταχωρητών ή την τιμή από την προηγούμενη lw (που βρίσκεται στο στάδιο WB). Τέλος το σήμα stall, όταν είναι ενεργό (τιμή 1), εμποδίζει την τρέχουσα εντολή του σταδίου ID να προχωρήσει στο στάδιο EX και, φυσικά, την εντολή του σταδίου IF να προχωρήσει στο ID. Αυτό μπορεί να συμβεί για δύο πιθανούς λόγους στον επεξεργαστή αυτόν: είτε κάποια εντολή διακλάδωσης δεν έχει κάποια τιμή καταχωρητή που χρειάζεται για τον έλεγχο της συνθήκης, ή μια εντολή δεν έχει την τιμή που προέρχεται από μια προηγούμενη lw.

Υπάρχουν ήδη τέσσερα ξεχωριστά always blocks για τα σήματα αυτά, αλλά για την ώρα παίρνουν προεπιλεγμένες τιμές που δεν κάνουν καμία προώθηση και δεν προκαλούν καμία καθυστέρηση. Θα πρέπει να προσθέσετε κώδικα σε Verilog που να εξετάζει σε ποιές περιπτώσεις θα πρέπει να γίνει προώθηση και από πού καθώς και σε ποιές περιπτώσεις θα γίνει καθυστέρηση. Θα χρειαστείτε μερικές if-else, πιθανότατα με σύνθετες συνθήκες (`&&`, `||`, ...), αλλά σίγουρα δεν θα χρειαστεί να κάνετε επαναλήψεις ή κάτι ιδιαίτερα περίπλοκο. Χρησιμοποιείστε απλές αναθέσεις (με το =, **όχι** <=) γιατί αυτά είναι συνδιαστικά κυκλώματα.

Η μονάδα ελέγχου διοχέτευσης δέχεται ως εισόδους πολλά σήματα. Κάποια προέρχονται από άλλα στάδια και μεταφέρουν τον καταχωρητή προορισμού του σταδίου και πληροφορίες για το αν η εντολή στο

στάδιο είναι φόρτωση από μνήμη ή αν η εντολή πράγματι γράφει αποτελέσματα σε καταχωρητή. Πολλά σήματα προέρχονται από τον κύριο αποκωδικοποιητή που βρίσκεται στο στάδιο ID και προσδιορίζουν («εξηγούν») την τρέχουσα εντολή. Δεν θα πρέπει να χρειαστείτε άλλα σήματα (θύρες) εισόδου-εξόδου. Επειδή στα σήματα ελέγχου οι λεπτομέρειες είναι εξαιρετικά σημαντικές, μερικές φορές, αντί να στηριχθείτε στα ονόματα σημάτων για να καταλάβαιτε τί κάνουν, ίσως να χρειαστεί να δείτε τον κώδικα άλλων modules, και ιδιαίτερα του κύριου αποκωδικοποιητή (maindec).

4 Παραδοτέα

Το παραδοτέο της άσκησης είναι το αλλαγμένο αρχείο `pipe_ctrl.v`. Αν ελέγχοντας την ορθότητα της υλοποίησης προωθήσεων γράψατε κάποιο πρόγραμα assembly που εξετάζει ενδιαφέρουσες περιπτώσεις, μπορείτε να το προσθέσετε και αυτό, με όνομα αρχείου lab07.asm.

Η παράδοση θα γίνει μέσω GitHub, όπως πάντα. **Όπως και στο προηγούμενο παραδοτέο μήν ανεβάσετε στο GitHub κανένα άλλο αρχείο, εκτός αυτών που υπάρχουν στο starter, γιατί πιάνουν πολύ χώρο.**

5 Καθαρισμός αρχείων

Στους υπολογιστές των εργαστηρίων, επειδή ο διαθέσιμος χώρος σας στο δίσκο είναι περιορισμένος (quota), και τα εργαλεία που χρησιμοποιήσατε δημιουργούν πολλά και μεγάλα αρχεία, όταν τελειώσετε με την άσκηση, σβήστε όλα τα περιτά αρχεία. Κρατήστε μόνο ότι αρχείο υπάρχει ήδη στο αποθετήριο του GitHub και τυχόν προγράμματα assembly που γράψατε για έλεγχο.