5ο Εργαστήριο Αρχιτεκτονικής Η/Υ: Προσομοίωση επεξεργαστή

Α. Ευθυμίου

Παραδοτέο: Τρίτη 5 Απρίλη 2017, 23:00

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

Σας δίνεται ένας ολόκληρος επεξεργαστής σχεδόν ίδιος με αυτόν του συγγράμματος που παρουσιάστηκε στις διαλέξεις. Επιπλέον της jump, που περιγράφεται στο σύγγραμμα, έχουν προστεθεί οι εντολές lui, ori, addi, addiu[1].

Η πρώτη σας δουλειά (60% του βαθμού) είναι να τρέξετε ένα πρόγραμμα στον επεξεργαστή και, ελέγχοντας τα αποτελέσματα της εκτέλεσης, να βρείτε και να διορθώσετε τα δύο σφάλματα που έχει. Μετά θα προσθέσετε μια νέα εντολή, παραλλαγή της sw, στον επεξεργαστή και θα γράψετε ένα πρόγραμμα που την χρησιμοποιεί.

Θα πρέπει να έχετε μελετήσει τα μαθήματα για την υλοποίηση του MIPS σε ένα κύκλο ρολογιού που αντιστοιχούν μέχρι την ενότητα 4.4 του βιβλίου. 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 Η άσκηση

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

```
git remote add lab05_starter https://github.com/UoI-CSE-MYY402/lab05_starter.git
git fetch lab05_starter
git merge lab05_starter/master -m "Fetched lab05 starter files"
```

Ξεκινήστε το Quartus σύμφωνα με το σύστημά σας. Ανοίξτε το Quartus project με όνομα, mips.qpf, που υπάρχει έτοιμο. 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, γιατί θα τα χρειαστείτε για την μετέπειτα προσομοίωση.

Προσπάθησα να κρατήσω τα ονόματα των σημάτων ίδια με αυτά του συγγράμματος, αλλά σε κάποιες περιπτώσεις οι κανόνες ονομάτων του Quartus με υποχρέωσαν να αλλάξω λίγο κάποια ονόματα. Επίσης πρόσθεσα ονόματα σε διάφορα καλώδια γιατί αλλιώς παίρνουν αυτόματα κάποια ονόματα και μετά είναι δύσκολο να τα παρακολουθήσει κανείς.

2 Μετατροπή προγραμμάτων σε γλώσσα μηχανής

Για να προσομοιώσετε τον επεξεργαστή θα χρειαστείτε προγράμματα σε γλώσσα μηχανής και αρχικές τιμές στη μνήμη δεδομένων. Τα modules dmem και imem, οι μνήμες δεδομένων και εντολών του επεξεργαστή, έχουν τη δυνατότητα να φορτώσουν τα περιεχόμενα των αρχείων `data_memfile.dat` και `instr_memfile.dat` στις αντίστοιχες μνήμες. Αν ανοίξετε τα αρχεία αυτά θα δείτε δεκαεξαδικούς αριθμούς.

Τα αρχεία θα δημιουργηθούν με την βοήθεια του MARS. Ανοίξτε το αρχείο lab05.asm στον Mars. Παρατηρήστε ότι λείπει το συνηθισμένο τέλος προγραμμάτων assembly με την εντολή syscall. Αυτό γίνεται γιατί δεν είναι υλοποιημένη η syscall στον επεξεργαστή. Πρέπει να θυμάστε ότι και στα υπόλοιπα προγράμματά που θα τρέχουν στον επεξεργαστή δεν θα πρέπει να έχουν αυτή την εντολή.

[1] Αντίθετα με τις προδιαγραφές του MIPS, οι addi, addiu έχουν υλοποιηθεί ώστε να είναι ακριβώς όμοιες. Γενικά οι εντολές με απρόσημους αριθμούς δεν έχουν υλοποιηθεί. Η addiu είναι εξαίρεση γιατί χρησιμοποιείται μερικές φορές από τις ψευτοεντολές li, la που μας είναι χρήσιμες.

Μετατρέψτε το σε γλώσσα μηχανής πατώντας το κουμπί "Assemble" (F3). Τώρα μπορείτε να πάρετε το αντίστοιχο πρόγραμμα σε γλώσσα μηχανής, αλλά σε 2 χωριστά αρχεία για εντολές και δεδομένα. Από το μενού File, επιλέξτε Dump Memory. Θα εμφανιστεί ένα μικρό παράθυρο. Μπορείτε να επιλέξετε μεταξύ ".text" για το πρόγραμμα και ".data" για τα δεδομένα. Η μορφή που μπορεί να διαβαστεί από τη Verilog είναι "Hexadecimal Text" Τα ονόματα αρχείων πρέπει να είναι `data_memfile.dat` για δεδομένα και `instr_memfile.dat` για εντολές. Μη χρησιμοποιήσετε άλλα ονόματα. Τα imem.v, dmem.v περιμένουν τα συγκεκριμένα αρχεία για να δουλέψουν.

Για κάποιο λόγο (bug) το αρχείο δεδομένων που γράφει το Mars περιέχει 1024 γραμμές. Ο προσομοιωτής αργότερα διαμαρτύρεται, αλλά η προσομοίωση γίνεται κανονικά.

Προσοχή κάντε το Dump Memory της μνήμης δεδομένων πρίν τρέξετε το πρόγραμμα στον MARS. Διαφορετικά θα γράψει στο αρχείο τα δεδομένα όπως έχουν αλλαχθεί από την εκτέλεση του προγράμματος!

3 Προσομοίωση του επεξεργαστή

Αφού έχετε δημιουργήσει τα αρχεία που φορτώνονται στις μνήμες, μπορείτε να τρέξετε με προσομοίωση στον επεξεργαστή.

Ξεκινήστε τον προσομοιωτή με την εντολή vsim. Αν το Modelsim «θυμάται» το παλιό modelsim project του lab05, κλείστε το με File > Close. Το Modelsim μπορεί να θυμάται και τον προηγούμενο κατάλογο από όπου το τρέξατε, οπότε προσέξτε σε διάφορα παράθυρα/διαλόγους μήπως και δεν έχετε δώσει το σωστό κατάλογο.

Ξεκινήστε ένα νέο modelsim project με όνομα `mips_sim`. Προσθέστε όλα τα αρχεία Verilog (.v) **εκτός από** τα `lpm_constant4_bb.v lpm_constant4_syn.v`.

Κάντε Compile > Compile All

Ξεκινήστε την προσομοίωση: Simulate > Start Simulation. Διαλέξτε το top.v από τη βιβλιοθήκη Work.

Προσοχή, φαίνεται πως στα Windows το Modelsim τρέχει την προσομοίωση σε κατάλογο διαφορετικό από αυτό που βρίκονται τα αρχεία του lab05. Έτσι δεν βρίσκει το `instr_memfile.dat` και το `data_memfile.dat` που περιέχουν τα αρχικά δεδομένα της μνήμης εντολών (το πρόγραμμα σε γλώσσα μηχανής) και μνήμης δεδομένων (αρχικές τιμές). Δυστυχώς, **η μή εύρεση των αρχείων είναι απλά ένα warning για το Modelsim** και έτσι συνεχίζει την προσομοίωση διαβάζοντας x (η τιμή της verilog για μη αρχικοποιημένες τιμές) για εντολές και έτσι όλα τα σήματα γίνονται τελικά x. Αν δείτε το παράθυρο με τα μηνύματα του προσομοιωτή θα δείτε 2 warnings.

Η λύση είναι να αλλάξετε τις εντολές readmemh στα αρχεία imem.v dmem.v ώστε να έχουν το πλήρες μονοπάτι των αρχείων instr_memfile.dat και data_memfile.dat Εδώ προσέξτε να χρησιμοποιήσετε το / του Unix ως το διαχωριστικό ονομάτων καταλόγων και όχι το \ των Windows! Φυσικά θα χρειαστεί ξανά compile αφού κανατε αλλαγές.

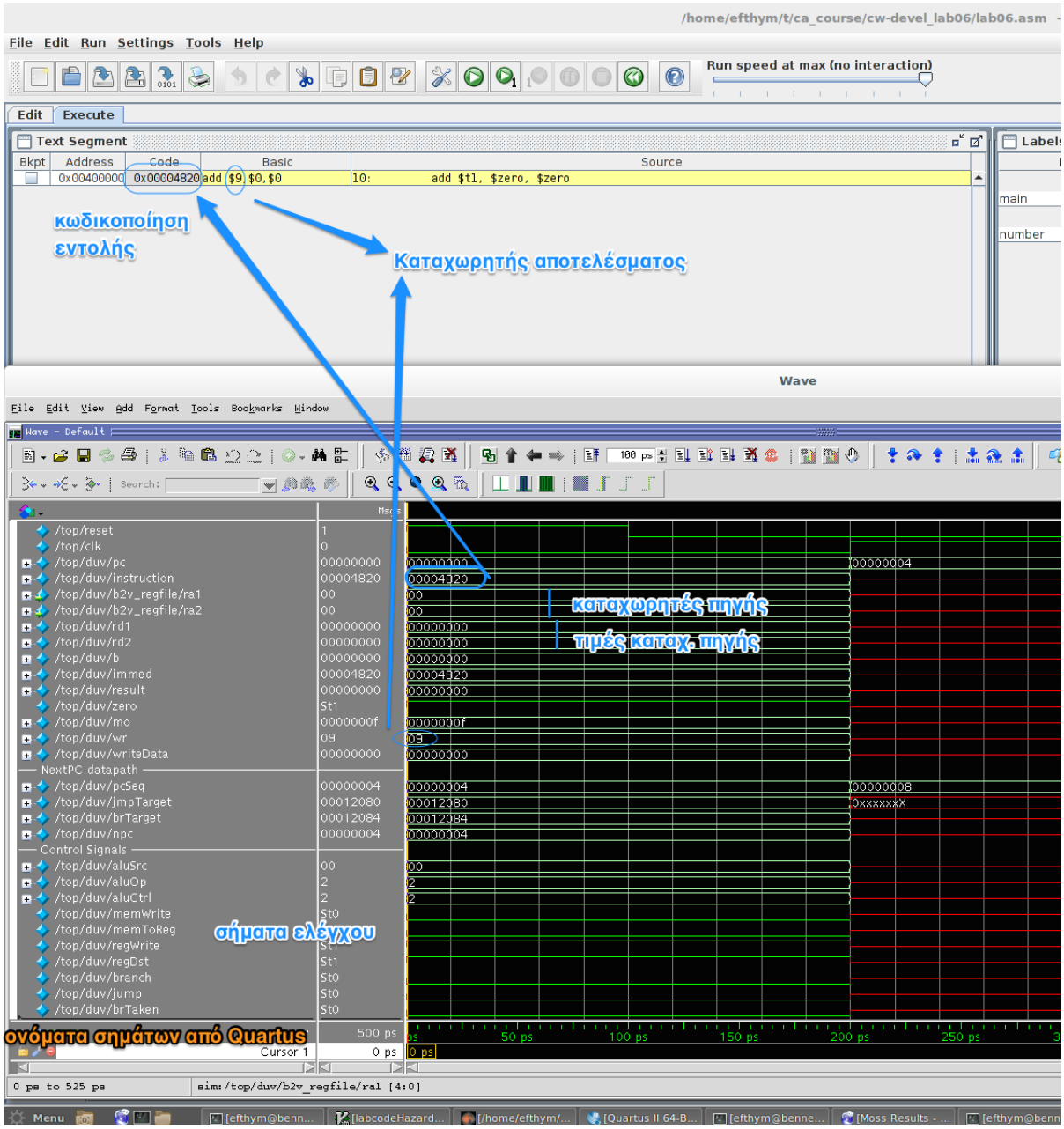
Αφού σιγουρευτείτε ότι η προσομοίωση γίνεται κανονικά, επιλέξτε τα σήματα που θέλετε να παρακολουθήσετε στις κυματομορφές. Θα πρέπει να κοιτάτε το σχηματικό στο Quartus για να βρίσκετε τα ονόματα των σημάτων που θέλετε. Για ευκολία, σας δίνεται ένα αρχείο με τα κύρια σήματα: wave.do. Για να το χρησιμοποιήσετε, επιλέξτε File > Load και στο παράθυρο που θα εμφανιστεί διαλέξτε το παραπάνω αρχείο. Μπορείτε να αλλάξετε θέση στα σήματα τραβώντας τα με το ποντίκι. Επίσης μπορείτε να τοποθετείτε διαχωριστικά (όπως στο wave.do) με Add > Divider

Τρέξτε την προσομοίωση, με το κατάλληλο εικονίδιο ή το F9. Για το πρόγραμμα που δίνεται αρκούν 6200ps. Κάντε Zoom-full (μεγεθυντικός φακός με σκούρο εσωτερικό ή πλήκτρο F), για να δείτε καλύτερα τις τιμές των σημάτων. Το παράθυρο κυματομορφών θα μοιάζει με το κάτω μέρος του σχήματος 1, αλλά θα έχει πολύ περισσότερες αλλαγές στις τιμές των σημάτων.

Μπορείτε να βρείτε την κωδικοποίηση μιας εντολής σε γλώσσα μηχανής από το πεδίο Code του Mars, καθώς επίσης και τους αριθμούς (όχι ονόματα) καταχωρητών, από το πεδίο Basic. (Ο κώδικας που γράψατε φαίνεται στο πεδίο Source.) Τα ονόματα των σημάτων στις κυματομορφές είναι τα ίδια με αυτά του σχηματικού του επεξεργαστή στο Quartus.

Παρατηρήστε ποιοι καταχωρητές διαβάζονται (ra1, ra2), τις τιμές τους (rd1, rd2), την πράξη που κάνει η ALU (aluCtrl), το αποτέλεσμα (result), τον αριθμό καταχωρητή στον οποίο γράφεται το αποτέλεσμα (wr), κλπ.

Στην τελευταία ακμή του ρολογιού, οι περισσότερες κυματομορφές κοκκινίζουν γιατί παίρνουν τιμές X, την ειδική τιμή της Verilog που δείχνει ότι δεν γνωρίζει αν ένα bit είναι 1 ή 0. Αυτό συμβαίνει γιατί η επόμενη εντολή που διαβάζεται από τη μνήμη (PC=0x...048) δεν είναι καθορισμένη και παριστάνεται με X. Ακολούθως τα X εξαπλώνονται σχεδόν σε όλα τα σήματα. Αυτό είναι φυσιολογικό να συμβαίνει στο τέλος του προγράμματος, αλλά αν το δείτε να συμβαίνει κατά τη διάρκεια της εκτέλεσης στα



Σχήμα 1: Κυματομορφές και Mars.

περισσότερα σήματα, κάποιο λάθος θα πρέπει να έχει συμβεί. Μερικές φορές, μεμονωμένα σήματα γίνονται X χωρίς να υπάρχει πρόβλημα. Αυτό συνήθως γίνεται σε σήματα που δεν είναι χρήσιμα για κάποιες εντολές, για παράδειγμα η τιμή του 2ου καταχωρητή, rt (σήμα rd2 στις κυματομορφές), όταν αυτός δεν χρησιμοποιείται για ανάγνωση αλλά είναι ο καταχωρητής προορισμού (π.χ. σε μια lw).

Στον κώδικα Verilog του επεξεργαστή έχουν προστεθεί καθυστερήσεις σε βασικά κυκλώματα (π.χ. ALU, αρχείο καταχωρητών, μνήμες). Έτσι σε κάθε ακμή του ρολογιού οι νέες τιμές εμφανίζονται με μια καθυστέρηση.[2] 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 εκτελείται σε έναν πλήρη κύκλο.

3.1 Διαφορές από τον MARS

Οι καταχωρητές, εκτός του 0 (zero) **δεν είναι αρχικοποιημένοι και περιέχουν X**. Μη στηρίζεστε στο γεγονός ότι έχουν 0 όπως στον MARS.

Οι μνήμες έχουν χώρο για 128 εντολές και 128 λέξεις δεδομένων. Θα προσέξετε ότι μόνο τα 9 λιγότερο σημαντικά bit των διευθύνσεων χρησιμοποιούνται, έτσι οι τιμές διευθύνσεων που βλεπετε στον MARS είναι κάπως διαφορετικές από αυτές του Modelsim. Για παράδειγμα τα δεδομένα στον MARS ξεκινούν από τη διεύθυνση 0x10010000 ενώ στο Modelsim θα βλέπετε να ξεκινούν από τη διεύθυνση 0. Παρόμοια θα παρατηρήσετε ότι ενώ ο PC ξεκινάει με την τιμή 0, όταν εκτελεστεί η πρώτη j θα γίνει 0x004000030. Τα πιο σημαντικά bit των διευθύνσεων δεν συνδέονται πουθενά και έτσι το 0x004000030 θα αντιστοιχεί στη διεύθυνση 0x000000030. Το ίδιο φυσικά θα ισχύει και για τις υπόλοιπες διευθύνσεις που ξεκινούν από 0x004...

4 Μέρος 1: Προσομοίωση προγράμματος και έλεγχος σωστής λειτουργίας του επεξεργαστή

Τώρα που γνωρίζετε τη διαδικασία συγγραφής προγράμματος, μετατροπής του σε γλώσσα μηχανής και προσομοίωσης, ήρθε η στιγμή να επαληθεύσετε τη λειτουργία του επεξεργαστή χρησιμοποιώντας το lab05.asm.

Για την ώρα ο επεξεργαστής υλοποιεί το υποσύνολο των εντολών MIPS: add, sub, and or, slt, lw, sw, beq, j, lui, addi, addiu, ori. Υπάρχουν όμως 2 λάθη στην υλοποίηση: κάποιες εντολές δίνουν λάθος αποτελέσματα.

Θα πρέπει να βρείτε τα λάθη κοιτάζοντας τα αποτελέσματα του επεξεργαστή στις κυμματομορφές και συγκρίνοντάς τα με τα αποτελέσματα του MARS. Το αποτέλεσμα μιας εντολής εξαρτάται από το είδος της:

- οι αριθμητικές-λογικές εντολές (συμπεριλαμβανομένης και της slt) και η lw (ελέγξτε και τη διεύθυνση αν είναι σωστή), γράφουν αποτέλεσμα σε εναν καταχωρητή
- οι sw γράφουν στην κατάλληλη θέση μνήμης (ελέγξτε και τη διεύθυνση και την τιμή του καταχωρητή που αποθηκεύεται)
- οι beq, j έχουν ως αποτελεσμα την αλλαγή ροης προγράμματος, άρα θα πρέπει να ελέγξετε αν η επόμενη εντολη που εκτελείται είναι η σωστή.

Αφού βρείτε τις εντολές που δίνουν λάθος αποτέλεσμα, ψάξτε τον λόγο για τον οποίο γίνεται το λάθος. Ελέγξτε αν οι τιμές εισόδου είναι σωστές. Π.χ. για μια add ελέξτε τις τιμές των καταχωρητών πηγής,ενώ για μία addi αν η τιμή του καταχωρητή πηγής κσι της σταθεράς είναι σωστές. Ελέγξτε αν τα αποτελέσματα γράφονται στο σωστό καταχωρητή ή θέση μνήμης. Τέλος ελέγξτε αν κάποιο σήμα

[2] Αυτό **δεν φαίνεται** στο σχήμα 1 γιατί το screenshot πάρθηκε από προηγούμενη έκδοση του κώδικα.

ελέγχου είναι λάθος, π.χ. λάθος πράξη, ή δεν γίνεται εγγραφή σε καταχωρητή (το σήμα regWrite είναι 0 αντί για 1).

Αφού βρείτε ακριβώς τι φταίει, διορθώστε το. **Δεν θα χρειαστούν αλλαγές στο σχηματικό του επεξεργαστή**, παρά μόνο μικρές αλλαγές σε αρχεία verilog. Επιβεβαιώστε ότι πλέον ο επεξεργαστής δουλεύει σωστά, ελέγχοντας ότι τα αποτελέσματα είναι πλεόν ίδια με τον MARS.

5 Παραδοτέα

Ο κύριος σκοπός της άσκησης είναι η σε βάθος κατανόηση της λειτουργίας ενός επεξεργαστή. Αυτό επιτυγχάνεται με προσομοίωση προγραμμάτων, παρακολουθώντας πως αλλάζουν τα διάφορα σήματα του επεξεργαστή, ανάλογα με την εντολή που εκτελείται κάθε φορά και επιβεβαιώνοντας ότι τα αποτελέσματα είναι πάντα σωστά.

Τρέχοντας ένα πρόγραμμα για να επιβεβαιώσετε ότι ο επεξεργαστής δουλεύει σωστά, θα αναγκαστείτε να εξετάσετε τα περισσότερα, αν όχι όλα, τα σήματα και θα διερευνήσετε ακραίες περιπτώσεις λειτουργίας, τις οποίες ο επεξεργαστής θα πρέπει να χειρίζεται σωστά. Θα χρησιμοποιήσετε τον MARS ως το λεγόμενο Golden model: το μοντέλο που θεωρούμε ότι είναι πάντα σωστό και με αυτό συγκρίνουμε τα αποτελέσματα της προσομοίωσης. Αν παίρνετε τις ίδιες τιμές με τον MARS, ο επεξεργαστής έχει υλοποιηθεί σωστά. Διαφορετικά θεωρείτε ότι ο MARS «έχει δίκιο» και το λάθος είναι στον επεξεργαστή.

Τα παραδοτέα του 1ου μέρους της άσκησης είναι τα αρχεία verilog του επεξεργαστή που θα αλλάξετε για να διορθώσετε το λάθος. Η παράδοση θα γίνει μέσω GitHub, όπως πάντα.

Όπως και στο προηγούμενο παραδοτέο μήν ανεβάσετε στο GitHub κανένα άλλο αρχείο, γιατί πιάνουν πολύ χώρο και υπάρχει κίνδυνος να πετύχετε τα όρια του GitHub για χώρο αποθήκευσης αποθετηρίου. Ειδικά αν χρησιμοποιείτε drag&drop για παράδοση στο GitHub, μήν στείλετε όλο τον κατάλογο lab05, αλλά τα επιμέρους αρχεία.

6 Μέρος 2: Προσθήκη εντολής store word with update

Έστω ότι αναλύοντας τον κώδικα πολλών εφαρμογών καταλήξαμε στο συμπέρασμα ότι χρειάζεται να προσθέσουμε στον επεξεργαστή μια εντολή που αποθηκεύει μια λέξη στη μνήμη και ταυτόχρονα αλλάζει τον καταχωρητή που χρησιμοποιείται για τον υπολογισμό της διεύθυνσης με την effective address. Ονομάζουμε αυτή την εντολή store word with update, και η σύνταξή της είναι `swupd $rt, imm($rs)`. Είναι ισοδύναμη με τις παρακάτω δύο εντολές:

```
sw   $rt, imm($rs)
addi $rs, $rs, imm
```

Η εντολή αυτή είναι τύπου I και θα χρησιμοποιεί την τιμή 0x2c (44) ως opcode. Η swupd μπορεί να υλοποιηθεί στον επεξεργαστή της άσκησης με πολύ μικρές αλλαγές στο σχηματικό και στον κυρίως αποκωδικοποιητή (maindec.v).

Για διευκόλυνση, επειδή η δημιουργία νέων συμβόλων στο quartus είναι κάπως προβληματική, υπάρχει ένας πολυπλέκτης 4 εισόδων πλάτους 5 bit, mux4_5. Δεν θα χρειαστείτε άλλα νέα κυκλώματα. Αν χρειαστεί να κάνετε μικρές αλλαγές σε αρχεία verilog, για παράδειγμα να αλλάξετε το πλάτος σε bits ενός σήματος ελέγχου, θα πρέπει να κάνετε τις αντίστοιχες αλλαγές και στα «σύμβολα» (αρχεία .bsf) του αντίστοιχου κυκλώματος (module), ώστε οι θύρες εισόδου-εξόδου να είναι σωστές. Ο ευκολότερος τρόπος είναι να ανοίξετε το αντίστοιχο αρχείο .bsf με έναν text editor και να αλλάξετε το όνομα του σήματος ελέγχου ώστε να έχει το νέο πλάτος. Για παράδειγμα αν ένα σήμα X που είχε πλάτος 1 bit, αλλάξει σε 3 bit, το όνομά του, από `X`, θα πρέπει να γίνει `X[2..0]`. Αυτή η αλλαγή πρέπει να γίνει σε κάθε σημείο στο αρχείο .bsf όπου υπάρχει το όνομα του σήματος. Εκτός από το σύμβολο, θα πρέπει να αλλάξετε στο σχηματικό τα labels των αντίστοιχων καλωδίων που συνδέουν τις αλλαγμένες θύρες, καθώς και να μετατραπούν από απλά καλώδια (node line) σε bus (bus line). Αυτό γίνεται με δεξί κλικ στο καλώδιο και την κατάλληλη επιλογή του τύπου καλωδίου.

Αν δημιουργήσετε αυτόματα το σύμβολο από το αρχείο verilog, όπως στο προηγούμενο εργαστήριο, το

μέγεθός του θα αλλάξει σημαντικά όπως και η θέση των θυρών και έτσι θα αναγκαστείτε να κάνετε μεγάλες αλλαγές στο σχηματικό του επεξεργαστή.

Οι αλλαγές στο σχηματικό θα πρέπει να ελεγχθούν, να σωθούν και να ξανα-δημιουργήσετε το ανίστοιχο αρχείο verilog που χρειάζεται για την προσομοίωση. Οι αλλαγές στα αρχεία verilog χρειάζονται ξανά compile στο modelsim πριν την προσομοίωση.

7 Επαλήθευση υλοποίησης

Για επαλήθευση της υλοποίησης θα μετατρέψετε το πρόγραμμα του αρχείου lab02_v2.asm, που φαίνεται παρακάτω, ώστε αντί για sw και addi να χρησιμοποιεί μία swupd στο βρόγχο. Το πρόγραμμα αρχικοποιεί έναν πίνακα 128 ακεραίων με τιμές -1. Θα χρειαστεί και μια μικρή αλλαγή στον κώδικα εκτός του βρόγχου ώστε να προσπελαύνει τις ίδιες ακριβώς θέσεις με το αρχικό, αλλά οι επιπλέον εντολές, επειδή είναι εκτός της επανάληψης δεν θα προσθέσουν σημαντική καθυστέρηση. Η τελευταία εντολή έχει προστεθεί για να φαίνεται το τέλος του βρόγχου στις κυμματομορφές - δεν έχει καμία άλλη χρησιμότητα.

```
        .data
table:  .word 0:0x20
        .globl main
        .text
main:
        la   $a0, table
        addi $t0, $zero, -1
        addi $t1, $a0,   0x20
loop:
        beq  $a0, $t1,  endLoop
        sw   $t0, 0($a0)
        addi $a0, $a0, 4
        j    loop
endLoop:
        add  $t0, $zero, $zero
```

Επειδή ο MARS δεν γνωρίζει την εντολή swupd, για να περάσει από τον assembler του MARS, χρησιμοποιείστε την απλή sw στη θέση της και, αφού κάνετε dump memory, βρείτε και αλλάξτε την κωδικοποιημένη sw σε swupd, με το χέρι, στο αρχείο `instr_memfile.dat`. Αυτό γίνεται αλλάζοντας το τμήμα opcode της sw ώστε να έχει τον κωδικό της swupd. Μετά προσομοιώστε τον επεξεργαστή με το αλλαγμένο πρόγραμμα για να βεβαιωθείτε ότι έχετε υλοποιήσει σωστά την swupd.

8 Παραδοτέα

Τα παραδοτέα του 2ου μέρους της άσκησης είναι τα αρχεία verilog του επεξεργαστή που θα αλλάξετε για να υλοποιήσετε την swupd, το σχηματικό του επεξεργαστή mips.bdf και το lab05_v2.asm που χρησιμοποιεί την swupd. Η παράδοση θα γίνει μέσω GitHub, όπως πάντα.

Δεν πειράζει που κάποια αρχεία θα έχουν ήδη αλλαχθεί από το 1ο μέρος της άσκησης. Έχοντας λύσει το 1ο μέρος, συνεχίστε αλλάζοντας τα αρχεία verilog και το σχηματικό και παραδώστε τη λύση του 2ου μέρους που περιλαμβάνει και τις διορθώσεις που χρειάστηκε να κάνετε στο 1ο.