

# 5ο Εργαστήριο Αρχιτεκτονικής Η/Υ: Προγράμματα ελέγχου **διοχετευμένου επεξεργαστή**

Α. Ευθυμίου

Παραδοτέο: Παρασκευή 10 Δεκέμβρη 2021, 23:59

Ο σκοπός της άσκησης είναι η εμβάθυνση της κατανόησης λειτουργίας ενός διοχετευμένου επεξεργαστή, μέσω συγγραφής προγραμμάτων assembly που επιβεβαιώνουν αν ο έλεγχος διοχέτευσης έχει υλοποιηθεί σωστά.

Θα πρέπει να έχετε μελετήσει τα μαθήματα για την υλοποίηση του MIPS σε πέντε στάδια διοχέτευσης που αντιστοιχούν μέχρι την ενότητα 4.8 του συγγράμματος.

## 1 Η άσκηση

Σημαντικό μέρος της σχεδίασης ενός υπολογιστικού συστήματος είναι η επιβεβαίωση της ορθότητας σχεδίασής του (design verification). Σε έναν επεξεργαστή αυτό γίνεται (επιπλέον διαφόρων άλλων μεθόδων) με την εκτέλεση προγραμμάτων assembly που είναι γραμμένα έτσι ώστε να ελέγχουν αν οι εντολές εκτελούνται σωστά και ως προς τα αποτελέσματα αλλά και ως προς τον χρόνο εκτέλεσης.




Σε αυτή την άσκηση θα θεωρήσουμε ότι μνήμες, αθροιστές, αποκωδικοποιητές εντολών, κ.α. έχουν επιβεβαιωθεί ότι σχεδιάστηκαν σωστά, αλλά πρέπει να γίνει επιβεβαίωση της υλοποίησης του ελέγχου διοχέτευσης και συγκεκριμένα του ελέγχου προωθήσεων τιμών καταχωρητών. Θα θεωρήσουμε ότι ο επεξεργαστής που έχουμε είναι ένας διοχετευμένος MIPS πέντε σταδίων στην βελτιωμένη μορφή του (θα παρουσιαστεί στο μάθημα της Τετάρτης 1 Δεκ 2021) όπου οι διακλαδώσεις εκτελούνται στο στάδιο ID και υπάρχει υλικό για την προώθηση τιμών μεταξύ συνεχόμενων lw-sw.

Σε αυτή την υλοποίηση οι διακλαδώσεις εκτελούνται όσο γίνεται νωρίτερα, αλλά, αν εξαρτώνται από προηγούμενες εντολές (μέσω τιμών καταχωρητών που πρέπει να συγκρίνουν), υπάρχει το ενδεχόμενο να πρέπει να γίνουν προωθήσεις τιμών δεδομένων και/ή καθυστερήσεις. Για παράδειγμα στην παρακάτω ακολουθία εντολών:

add \$t0, \$t1, \$t2    # produce new \$t0 value  
beq \$t0, \$t3, label    # stall 1 cycle and then forward new \$t0 value to this instruction

θα πρέπει να καθυστερήσει η beq για έναν κύκλο και μετά να προωθηθεί η τιμή του t0 στην beq. Αν μεταξύ των add, beq παρεμβάλλοταν μια ανεξάρτητη εντολή, τότε θα γινόταν απευθείας προώθηση χωρίς να χρειάζεται η beq να καθυστερήσει για έναν κύκλο. Αν αντί της add η τιμή του t0 προερχόταν από μία lw, που βρίσκεται αμέσως πριν από την beq, θα χρειαζόνταν 2 κύκλοι καθυστέρησης για την beq.

Στην παραπάνω διοχετευμένη υλοποίηση, υπάρχουν τρεις μονάδες στις οποίες μπορούν να προωθηθούν αποτελέσματα - τιμές:

1. Ο συγκριτής τιμών για τις διακλαδώσεις, που βρίσκεται στο στάδιο ID → 
2. Η ALU, που βρίσκεται στο στάδιο EXE → 
3. Η είσοδος δεδομένων προς εγγραφή της μνήμης δεδομένων, που βρίσκεται στο στάδιο MEM → 

Οι δύο πρώτες (συγκριτής, ALU) έχουν 2 ανεξάρτητες εισόδους, επομένως θα πρέπει να ελεγχθούν οι προωθήσεις και προς τις δύο εισόδους. Συνολικά υπάρχουν 2+2+1, 5 πιθανοί προορισμοί για τιμές που προωθούνται.

Οι προωθημένες τιμές προέρχονται από την ALU, για αριθμητικές - λογικές εντολές (π.χ. add, or, andi, κ.α.), είτε από την μνήμη για την εντολή lw. Στην περίπτωση που η ALU είναι αυτή που δίνει την τιμή προς προώθηση, υπάρχουν 2 (υπο)περιπτώσεις ανάλογα με το αν η εντολή που περιμένει την τιμή είναι

αμέσως μετά ή παρεμβάλεται μια ανεξάρτητη εντολή μεταξύ τους. Θα πρέπει να ελεγχθούν και οι δύο. Οι δύο (υπο)περιπτώσεις φαίνονται στα παρακάτω παραδείγματα.

1 {  
add \$t0, \$t1, \$t2 # produce new \$t0 value  
add \$t3, \$t0, \$t4 # forward new \$t0 value to this instruction → x5

3 {  
add \$t0, \$t1, \$t2 # produce new \$t0 value  
add \$s0, \$s1, \$s2 # independent instruction w.r.t. \$t0  
add \$t3, \$t0, \$t4 # forward new \$t0 value to this instruction → x5

Επομένως υπάρχουν 3 πιθανές πηγές τιμών προς προώθηση που με τους πιθανούς προορισμούς σχηματίζουν 15 περιπτώσεις που πρέπει να ελεγχθούν. Υπάρχουν και κάποιες επιπλέον ειδικές περιπτώσεις, όπως για παράδειγμα ο έλεγχος των προωθήσεων του καταχωρητή zero: στην περίπτωση αυτή δεν θα πρέπει να προωθούνται τιμές καθώς ο καταχωρητής αυτός έχει πάντα τιμή μηδέν και εγγραφές σε αυτόν δεν αλλάζουν την τιμή του.

Το αντικείμενο της άσκησης είναι ένα πρόγραμμα που θα ελέγχει τις 15 αυτές περιπτώσεις και όποιες άλλες μπορείτε να σκεφτείτε. Για να πάρετε τον αρχικό κώδικα, δημιουργήστε το αποθετήριο σας στο GitHub, ακολουθώντας το σύνδεσμο <https://classroom.github.com/a/>

### 1.1 Παράδειγμα προγράμματος επιβεβαίωσης

Για να έχετε ένα παράδειγμα ενός προγράμματος επιβεβαίωσης, στο αρχείο `testZero.asm` θα βρείτε το πρόγραμμα που ελέγχει τις προωθήσεις του καταχωρητή zero. Θα πρέπει να το μελετήσετε προσεκτικά και να καταλάβετε πως λειτουργεί γιατί θα σας βοηθήσει να γράψετε το δικό σας πρόγραμμα επιβεβαίωσης. Το παράδειγμα ελέγχει με τη σειρά πολλές περιπτώσεις. Αν βρεθεί λάθος καταλήγει στο label exit με τον καταχωρητή v1 να έχει έναν αριθμό που αντιστοιχεί στο πρώτο λάθος που βρέθηκε. Αν όλα τα τεστ περάσουν με επιτυχία, τότε ο v1 θα έχει την τιμή 0. Από την τιμή του v1, ο σχεδιαστής θα μπορεί δει ποιά περίπτωση δεν έχει υλοποιηθεί σωστά, να την διορθώσει στον επεξεργαστή και να ξανατρέξει το πρόγραμμα για να δει αν υπάρχουν άλλα λάθη.

Στις πρώτες γραμμές του κώδικα, τίθενται τιμές σε καταχωρητές που χρησιμοποιούνται παρακάτω, ενώ στο τμήμα .data υπάρχουν και μερικές αρχικές τιμές στη μνήμη για να μπορούν να γίνουν έλεγχοι και με εντολές μεταφοράς δεδομένων.

### 1.2 Σημαντικές παρατηρήσεις

Το πρόγραμμά σας θα πρέπει να παραδοθεί στο αρχείο `testForwarding.asm` και να ακολουθεί την σύμβαση του `testZero.asm` όπου η τιμή στον v1 δείχνει το είδος του λάθους, με την τιμή 0 να σημαίνει ότι όλα τα τεστ πέρασαν με επιτυχία.

Θα πρέπει να περνάει από τον assembler και να τρέχει στον MARS αλλά οι εντολές που θα χρησιμοποιούνται περιορίζονται σε αυτές που αναφέρονται στην εργαστηριακή άσκηση 4: add, sub, and, or, slt, lw, sw, beq, j, lui, addi, ori. Αν δεν είστε σίγουροι, μπορείτε να ακολουθήσετε την μεθοδολογία της άσκησης 4 για να τρέξετε το πρόγραμμά σας στο modelsim project της 4ης άσκησης. Θα πρέπει να τρέχει σωστά εκεί (εκτός του syscall) καθώς δεν γίνεται διοχέτευση. Φυσικά και στον Mars θα τρέχει με επιτυχία (τελική τιμή v1 = 0) καθώς ο Mars είναι προσομοιωτής που εκτελεί τις εντολές με τη σειρά και δεν ασχολείται με τον λεπτομερή χρονισμό της κάθε εντολής. Ο έλεγχος -βαθμολόγηση του προγράμματος θα πραγματοποιηθεί με άλλους προσομοιωτές από τον διδάσκοντα. Οι προσομοιωτές αυτοί δεν θα σας δοθούν καθώς απαιτούν άδεια χρήσης και μόνο ένας χρήστης μπορεί να προσομοιώσει κάθε φορά.

Επειδή είναι δύσκολο να ελεγχθούν τότε συμβαίνουν καθυστερήσεις στην εκτέλεση εντολών (stall), θα ελέγξετε μόνο τις προωθήσεις και όχι τις περιπτώσεις όπου χρειάζεται καθυστέρηση. Στο `testZero.asm`, σε μερικά σημεία ειδικά πριν από εντολές διακλαδώσεων, έχουν προστεθεί ανεξάρτητες εντολές (σημειώνονται με σχόλιο dummy) ώστε να μην χρειάζεται ο επεξεργαστής να καθυστερήσει την διακλάδωση για να προωθήσει τις τιμές. Ακολουθήστε την ίδια μέθοδο όπου χρειάζεται.

κινδυνος δεδομενων

Θα πρέπει να έχετε σχόλια που να περιγράφουν την περίπτωση που θέλετε να ελέγξετε. Σε μερικές περιπτώσεις μπορεί να χρειαστεί να είναι αναλυτικά, αλλά επειδή οι περιπτώσεις έχουν μεγάλες ομοιότητες, για παράδειγμα όταν ελέγχεται η ίδια περίπτωση αλλά σε διαφορετική είσοδο, μπορείτε να αναφέρεστε σε προηγούμενες περιπτώσεις και να περιορίζετε τα σχόλια στις επόμενες περιπτώσεις. Δείτε τα σχόλια στο testZero.asm για να έχετε μια εικόνα της λεπτομέρειας που χρειάζεται.

Η βαθμολόγηση θα γίνει με βάση τις περιπτώσεις που καλύπτει σωστά το πρόγραμμά σας και την εξήγηση των περιπτώσεων με σχόλια. Αν σκεφτείτε άλλες περιπτώσεις για έλεγχο, εξηγείστε τις αναλυτικά. Θα υπάρχουν λίγοι επιπλέον βαθμοί ως επιβράβευση γι'αυτές.