ΗΥ-225: Οργάνωση Υπολογιστών
Άνοιξη 2012

Τμ. Επ. Υπολογιστών

Σειρά Ασκήσεων 5: Εντολές Συγκρίσεων και Διακλαδώσεων

Παράδοση: Δευτέρα 12 Μαρτίου 2012 (βδ. 5.1) σε χαρτί, στο μάθημα (από βδ. 3.2)

5.1 Περίληψη Συγκρίσεων, Διακλαδώσεων, και Αλμάτων στον MIPS

Οι εντολές μεταφοράς ελέγχου (CTI, control transfer instructions) καθορίζουν να εκτελεστεί σαν επόμενη εντολή --πάντοτε ή υπό ορισμένες συνθήκες μόνο-- μια έντολη άλλη από την "επόμενη από κάτω" τους εντολή. Όταν η μεταφορά ελέγχου γίνεται υπό συνθήκη, οι εντολές συνήθως ονομάζονται **διακλαδώσεις** (branch). Όταν η μεταφορά γίνεται πάντοτε, οι εντολές συνήθως λέγονται **άλματα** (jump). Επίσης υπάρχουν καλέσματα διαδικασιών, λειτουργικού συστήματος, και επιστροφές από αυτά. Στον MIPS, η συνθήκη διακλάδωσης μπορεί να έχει περιορισμένες μόνο μορφές, γιά λόγους ταχύτητας. Οι μόνες συνθήκες διακλαδώσεων που υπάρχουν αφορούν συγκρίσεις ενός καταχωρητή με το μηδέν (δεν θα ασχοληθούμε με αυτές στο "δικό μας" υποσύνολο του MIPS), και συγκρίσεις δύο καταχωρητών *μόνο γιά ισότητα ή ανισότητα* και όχι γιά άλλων μορφών σχέσεις (μικρότερος, μεγαλύτερος, κλπ). Οι εντολές αυτές είναι:

- **`beq $rs, $rt, label`** # διακλάδωση εάν: `$rs == $rt`
- **`bne $rs, $rt, label`** # διακλάδωση εάν: `$rs != $rt`

Στη γλώσσα Assembly, η διεύθυνση προορισμού της διακλάδωσης δηλώνεται απλά με μια ετικέτα (label), και αναλαμβάνει ο Assembler να υπολογίσει και να βάλει τη σωστή δυαδική τιμή. Στη γλώσσα μηχανής, οι εντολές διακλάδωσης ακολουθούν το I-format, και η διεύθυνση προορισμού προκύπτει ως εξής:

```
PC_new := (PC_br + 4) + 4 * ImmOffset
```

όπου `PC_br` είναι η διεύθυνση της ίδιας της εντολής διακλάδωσης, `ImmOffset` είναι η σταθερή ποσότητα των 16 bits του I-format θεωρούμενη ως προσημασμένος αριθμός σε συμπλήρωμα ως προς 2 (δηλαδή sign-extended), και `PC_new` είναι η διεύθυνση της εντολής προορισμού σε περίπτωση επιτυχίας της διακλάδωσης. Η αύξηση `(PC_br+4)` γίνεται γιά λόγους ευκολίας του hardware (όλες οι εντολές αυξάνουν τον PC κατά 4). Ο πολλαπλασιασμός του ImmOffset επί 4 γίνεται γιά να εκμεταλλευτούμε το γεγονός ότι η διεύθυνση όλων των εντολών του MIPS είναι ακέραιο πολλαπλάσιο του 4, κι έτσι να τετραπλασιάσουμε το "βεληνεκές" των διακλαδώσεων --με άλλα λόγια, ο αριθμός ImmOffset μετράει πλήθος εντολών μπροστά (θετικός) ή πίσω (αρνητικός), αντί να μετρά πλήθος bytes μπροστά ή πίσω. (στην πραγματικότητα, ο MIPS έχει "καθυστερημένες διακλαδώσεις" (delayed branches), γιά λόγους καλύτερης εκμετάλλευσης της ομοχειρίας (pipelining) του hardware, αλλά εμείς θα το αγνοήσουμε σε αυτό το μάθημα --το θέμα αυτό συζητείται εν εκτάσει στο ΗΥ-425).

Οι υπόλοιπες μορφές συγκρίσεων, που δεν γίνονται μέσα στις εντολές διακλάδωσης, υλοποιούνται με ειδικές, ξεχωριστές, αριθμητικές εντολές σύγκρισης. Πρόκειται γιά εντολές ανάλογες προς την πρόσθεση ή την αφαίρεση, μόνο που το αποτέλεσμά τους είναι τύπου Boolean αντί τύπου ακέραιος. Τέτοια αποτελέσματα τύπου Boolean έχουν δύο μόνο δυνατές τιμές: 0 γιά ψευδές, και 1 γιά αληθές. Τα αποτελέσματα αυτά γράφονται στους γνωστούς μας, κανονικούς (32μπιτους) καταχωρητές, σαν οι ακέραιοι 0 ή 1, δηλαδή στο LS bit του καταχωρητή, με όλα τα υπόλοιπα bits του καταχωρητή μηδενικά. Οι εντολές σύγκρισης που εμείς θα έχουμε στο δικό μας υποσύνολο του MIPS είναι οι:

- **`slt $rd, $rs, $rt`** # set less than --αριθμητική σύγκριση των καταχωρητών: `$rs < $rt`. Το αποτέλεσμα, τύπου Boolean, γράφεται στον καταχωρητή $rd.
- **`slti $rd, $rs, imm`** # set less than immediate --αριθμητική σύγκριση καταχωρητή και προσημασμένης (sign-extended) σταθερής ποσότητας imm: `$rs < imm`. Το αποτέλεσμα (Boolean) γράφεται στον καταχωρητή $rd.

Εκτός από τις εντολές διακλάδωσης υπό συνθήκη, το υποσύνολο εντολών του MIPS που χρησιμοποιούμε στο μάθημα περιλαμβάνει και τις παρακάτω άλλες εντολές μεταφοράς ελέγχου:

`j target`
: Αλμα (jump) χωρίς συνθήκη: επόμενη προς εκτέλεση εντολή είναι η εντολή στη διεύθυνση `target`. Χρησιμοποιεί το J-format, το οποίο φαίνεται στη σελίδα 171 του βιβλίου (επάνω -

Α' τόμος, 4η έκδοση). Η τελική διεύθυνση προορισμού (32 bits) προκύπτει από τα 4 παλαιά MS bits του PC, τα 26 bits του πεδίου προορισμού της εντολής, και από 2 μηδενικά LS bits, όπως εξηγείται στη σελίδα 172 του βιβλίου (κάτω). Στην σελίδα 174 φαίνεται και η χρήση της εντολής jump, μαζί με μια branch, γιά την έμμεση σύνθεση διακλάδωσης υπό συνθήκη σε απόσταση μεγαλύτερη από το βελησνεκές των απλών διακλαδώσεων.

`jr $rs`

Αλμα σε προορισμό που καθορίζεται από καταχωρητή (jump register): επόμενη προς εκτέλεση εντολή είναι η εντολή στη διεύθυνση που περιέχεται στον καταχωρητή rs (με άλλα λόγια, ο $rs περιέχει τη διεύθυνση προορισμού, δηλαδή έναν pointer στην επόμενη προς εκτέλεση εντολή). Η εντολή αυτή μας επιτρέπει να μεταφέρουμε τον έλεγχο (την εκτέλεση του προγράμματος) σε αυθαίρετη θέση μνήμης, η οποία μπορεί και να ποικίλει κατά την εκτέλεση του προγράμματος (run-time variable) και πιθανόν να εξαρτάται και από τα δεδομένα (data dependent). Χρησιμοποιείται γιά μετάφραση του **`switch`** statement, όπως περιγράφεται στη σελίδα 150 του βιβλίου (Α' τόμος, 4η έκδοση), γιά μεταφορά του ελέγχου οσοδήποτε μακριά (σελίδα 172 κάτω του βιβλίου), και γιά επιστροφή από διαδικασία όπως περιγράφεται αμέσως παρακάνω.

`jal target`

Αλμα και Σύνδεση (jump and link) -- κάλεσμα διαδικασίας: έχει το ίδιο format με την εντολή jump, και κάνει τα ίδια με εκείνην (ίδια διεύθυνση προορισμού), συν, επιπλέον, αποθηκεύει τη διεύθυνση της επόμενής της εντολής (παλαιό PC συν 4) στον καταχωρητή 31 ($ra, ή $31). Χρησιμοποιείται γιά κάλεσμα διαδικασίας, όπως περιγράφεται στις σελίδες 152-153 του βιβλίου (Α' τόμος, 4η έκδοση). Η διεύθυνση που αποθηκεύεται μπορεί στη συνέχεια να χρησιμοποιηθεί γιά επιστροφή από τη διαδικασία, μέσω της εντολής **`jr $ra`**. Η διεύθυνση επιστροφής αποθηκεύεται πάντα στον $ra ($31) --το "31" δεν φαίνεται πουθενά μέσα στην εντολή **`jal`**, απλώς το βάζει το hardware αυτόματα.

Άσκηση 5.2: Διακλάδωση με Σύγκριση Καταχωρητή-Σταθεράς

Είπαμε ότι, γιά λόγους ταχύτητας, οι μόνες συνθήκες διακλάδωσης του MIPS που αφορούν δύο αυθαίρετους αριθμούς (όχι έναν αριθμό και το μηδέν) είναι συγκρίσεις ισότητας/ανισότητας και όχι συγκρίσεις μεγαλύτερος/μικρότερος. Ομως, επίσης, οι εντολές αυτές (`beq, bne`) υπάρχουν μόνο στη μορφή που δέχεται δύο καταχωρητές σαν τελεστέους, και δεν μπορούν να συγκρίνουν καταχωρητή με σταθερή ποσότητα (immediate). Ο λόγος δεν μπορεί να έχει να κάνει με ταχύτητα, αφού η σύγκριση ισότητας/ανισότητας καταχωρητή-σταθεράς είναι τουλάχιστο το ίδιο γρήγορη με την αντίστοιχη σύγκριση δύο καταχωρητών. Γιατί λοιπόν πιστεύετε ότι δεν υπάρχουν τέτοιες εντολές "`beqi`" και "`bnei`" στον MIPS; Δώστε την απάντησή σας σε χαρτί και εξηγήστε.

Άσκηση 5.3: Εντολές Σύγκρισης και Διακλάδωσης

Έστω ότι η μεταβλητή **`i`** βρίσκεται στον καταχωρητή $16, η μεταβλητή **`j`** στον καταχωρητή $17, και ότι **`CONST`** σημαίνει μια αυθαίρετη προσημασμένη σταθερή ποσότητα μέχρι και 16 bits. Συνθέστε τις παρακάτω περιπτώσεις κώδικα χρησιμοποιώντας αποκλειστικά και μόνο εντολές μεταξύ των:

`beq, bne, slt, slti, addi`

και καμια άλλη. Όπου χρειάζεστε προσωρινό καταχωρητή, χρησιμοποιήστε τον $at (Assembler temporary) ο οποίος είναι ο $1 (αυτόν χρησιμοποιεί ο Assembler γιά να συνθέτει τις ψευδοεντολές του). Δώστε τις απαντήσεις σας σε χαρτί.

i. `if ( i == j ) goto L1;` (ίσο)
ii. `if ( i != j ) goto L1;` (διάφορο)
iii. `if ( i < j ) goto L1;` (μικρότερο)
iv. `if ( i <= j ) goto L1;` (μικρότερο ή ίσο)
v. `if ( i > j ) goto L1;` (μεγαλύτερο)
vi. `if ( i >= j ) goto L1;` (μεγαλύτερο ή ίσο)
vii. `if ( i == CONST ) goto L1;` (ίσο)
viii. `if ( i != CONST ) goto L1;` (διάφορο)
ix. `if ( i < CONST ) goto L1;` (μικρότερο)
x. `if ( i <= CONST ) goto L1;` (μικρότερο ή ίσο)
xi. `if ( i > CONST ) goto L1;` (μεγαλύτερο)
xii. `if ( i >= CONST ) goto L1;` (μεγαλύτερο ή ίσο)

Υπόδειξη: παίξτε με τη σειρά που βάζετε τους 2 τελεστέους πηγής στην εντολή σύγκρισης, με τον καταχωρητή $0 που περιέχει πάντα μηδέν (ή "ψευδές"), με το είδος της διακλάδωσης (ίσο/άνισο ή

ψευδές/αληθές), με τις σταθερές `CONST, CONST+1, CONST-1, -CONST`, και με τους (πολλούς) συνδυασμούς όλων αυτών. Θα εκτιμήστε το πόσα πολλά μπορεί να κάνει το software, εκμεταλλευόμενο λίγους, προσεκτικά επιλεγμένους δομικούς λίθους hardware, χωρίς απώλεια ταχύτητας!

Άσκηση 5.4: Μετάφραση Βρόχου "While"

Στην παράγραφο 2.7 του βιβλίου, σελ. 147 (4η έκδοση), υπάρχει ο παρακάτω βρόχος μεταφρασμένος σε Assembly κατά τέτοιο τρόπο ώστε σε κάθε ανακύκλωση να εκτελείται τόσο μια διακλάδωση υπό συνθήκη (branch), όσο και ένα άλμα χωρίς συνθήκη (jump) (η εντολή `sll` είναι η "shift left logical", δηλαδή αριστερή ολίσθηση, εδώ κατά 2 θέσεις (bits)):

```
while (save[i] == k) {     Loop: sll  $t1, $s3, 2    # t1 = 4 * i
  i = i+1;                       add  $t1, $t1, $s6  # t1 = διεύθυνση του save[i]
}                                lw   $t0, 0($t1)    # t0 = save[i]
                                 bne  $t0, $s5, Exit # if save[i] != k goto Exit
                                 addi $s3, $s3, 1    # i = i+1
                                 j    Loop           # επανάληψη: goto Loop
                           Exit: ....                # υπόλοιπο πρόγραμμα
```

Όμως, μόνον οι εξαιρετικά απλοϊκοί μεταφραστές θα έφτιαχναν τέτοιο κώδικα --οι συνηθισμένοι μεταφραστές παράγουν κώδικα που τρέχει πιο γρήγορα στη συνηθισμένη περίπτωση, όταν δηλαδή το πλήθος των ανακυκλώσεων είναι σημαντικά μεγαλύτερο του 1. Ξαναγράψτε το βρόχο σε Assembly, ούτως ώστε **μόνο ένα** branch ή jump να εκτελείται σε κάθε ανακύκλωση. (Κατά την είσοδο ή την έξοδο από το βρόχο επιτρέπεται να εκτελούνται δύο εντολές μεταφοράς ελέγχου -- αυτό που μας ενδιαφέρει είναι να γλυτώνουμε τη μια από αυτές κατά τις υπόλοιπες επαναλήψεις του βρόχου, που αποτελούν και την πλειοψηφία των φορών που αυτός εκτελείται). Έστω ότι ο βρόχος εκτελείται 10 φορές. Πόσες εντολές συνολικά εκτελούνταν με τον παλαιό κώδικα, και πόσες συνολικά με τον νέο;
