Οι υπολογιστές στον πραγματικό κόσμο

Βοήθεια στη διάσωση του περιβάλλοντος με δεδομένα

Πρόβλημα: Η παρακολούθηση φυτών και ζώων του περιβάλλοντός μας για τη συλλογή πληροφοριών που μπορεί να επηρεάσουν τις περιβαλλοντικές πολιτικές.

Λύση: Η ανάπτυξη ανθεκτικών, ενσωματωμένων υπολογιστών που λειτουργούν με μπαταρίες, με αισθητήρες, ασύρματη επικοινωνία, και κατάλληλο λογισμικό.

Η βιολόγος του Stanford Barbara Block μελετάει το γαλαζόπτερο τόνο (bluefin tuna). Ένα ζήτημα πολιτικής ήταν αν οι τόνοι στη μια πλευρά του Ατλαντικού είναι διαφορετικοί από αυτούς στην άλλη πλευρά. Αν είναι έτσι, τότε κάθε περιοχή θα μπορούσε να θέσει τις δικές της ποσοστώσεις (quotas). Αν όχι, τότε χρειαζόμαστε ποσοστώσεις για όλο τον ωκεανό.

Για να απαντήσει σε αυτή την ερώτηση, άρχισε να εμφυτεύει σε τόνους συσκευές που μπορούσαν να παρακολουθήσουν τα ταξίδια τους. Κάθε δύο λεπτά, μια αναδυόμενη ετικέτα αρχειοθέτησης μέσω δορυφόρου (pop-up satellite archival tag — PSAT) καταγράφει την πίεση του νερού, το φωτισμό του περιβάλλοντος, τη θερμοκρασία, την ώρα της ημέρας, και άλλες μετρήσεις. Τα δεδομένα αποθηκεύονται σε 1 MB μνήμης φλας (flash memory). Ο ενσωματωμένος μικροεπεξεργαστής των 8 bit εκτιμά το βάθος από την πίεση του νερού. Βρίσκει το γεωγραφικό μήκος χρησιμοποιώντας δεδομένα έντασης του φωτός και ώρας της ημέρας. Προσδιορίζει την ανατολή και τη δύση του ηλίου και, συνεπώς, τη μεσημβρία, και υπολογίζει τη διαφορά ώρας ανάμεσα στην τοπική μεσημβρία και τη μεσημβρία Μέσης Ώρας Γκρίνουιτς (Greenwich Mean Time), όπως ένας θαλασσοπόρος με έναν εξάντα και ένα χρονόμετρο. Η θερμοκρασία του νερού ταυτίζεται αργότερα με αρχεία από δορυφόρο για να προσδιοριστεί το γεωγραφικό πλάτος. Η Block δε βασίζεται σε ψαράδες για να πιάσει τους τόνους και να επιστρέψει τις ετικέτες PSAT. Οι PSAT προσκολλώνται στα ψάρια με μια καρφίτσα που διαλύεται μέσω ηλεκτρόλυσης αφού ο υπολογιστής ενεργοποιήσει μια μπαταρία. Η ετικέτα στη συνέχεια ανεβαίνει στην επιφάνεια και αρχίζει να εκπέμπει δεδομένα σε δορυφόρους. Η επιπλέουσα ετικέτα μπορεί να εκ-

Η Block και φοιτητές βάζουν μια ετικέτα σε ένα γαλαζόπτερο τόνο, που μπορεί να φτάσει σε βάρος μέχρι 900 κιλά και μήκος μέχρι 3 μέτρα.

Μια αναδυόμενη ετικέτα αρχειοθέτησης μέσω δορυφόρου (pop-up satellite archival tag — PSAT) και τα εσωτερικά ηλεκτρονικά της.

πέμπει για μέχρι και δύο εβδομάδες, μεταδίδοντας τα δεδομένα απευθείας στο εργαστήριο της Block.

Η Block ανακάλυψε ότι οι γαλαζόπτεροι τόνοι ταξιδεύουν περισσότερα από 16.000 χλμ. το χρόνο· τόνοι στους οποίους προσαρτώνται ετικέτες κοντά στην ανατολική ακτή των Ηνωμένων Πολιτειών διασχίζουν τον Ατλαντικό και αναπαράγονται τόσο στον Κόλπο του Μεξικού όσο και στην Ανατολική Μεσόγειο. Η ανακάλυψή της άλλαξε τους κανονισμούς ώστε οι τόνοι να μην αντιμετωπίζονται πλέον διαφορετικά στον Ανατολικό και το Δυτικό Ατλαντικό. Αναπτύσσει τώρα μια έρευνα για τη θαλάσσια ζωή του Ειρηνικού χρησιμοποιώντας μικρότερες ετικέτες για μικρότερα ζώα, και ετικέτες που εκπέμπουν κάθε φορά που ένα ψάρι βγαίνει στην επιφάνεια. Εικάζει ότι οι τόνοι με τις ετικέτες θα μπορούσαν να αποτελέσουν ιδανικά «οχήματα» για την παρακολούθηση των αλλαγών στους ωκεανούς.

Ο βιολόγος του Berkeley Todd Dawson μελετάει το οικοσύστημα της παράκτιας σεκόγιας του είδους *Sequoia sempervirens*, και συγκεκριμένα την αλληλεπίδραση της θαλάσσιας ομίχλης με τα δένδρα. Για χρόνια, η έρευνά του περιλαμβάνει την εγκατάσταση 50 κιλών εξοπλισμού και χιλιομέτρων καλωδίων για τη σύνδεση με αισθητήρες. Αυτή η δουλειά γίνεται συχνά σε ύψος μεγαλύτερο από 80 μέτρα από το έδαφος. Τα δεδομένα μπορούν να ανακτηθούν μόνο με την αναρρίχηση μέχρι έναν καταγραφέα δεδομένων (data logger) μεγέθους εκτυπωτή.

Ο καθηγητής Dawson και ένα φοιτητής ενώ αναρριχώνται σε μια σεκόγια για να εγκαταστήσουν συσκευές παρακολούθησης της ομίχλης.

Ο επιστήμονας υπολογιστών του Berkeley David Culler πρότεινε μια νέα προσέγγιση. Ο Dawson τοποθετεί τώρα μικροσκοπικούς ασύρματους αισθητήρες με μέγεθος κυλίνδρου φιλμ στα δένδρα. Κάθε αισθητήρας έχει όγκο περίπου 16 κυβικά εκατοστά, μπορεί να μεταδώσει μέχρι 40 KB/sec, και μπορεί να λειτουργεί για μήνες με μία μπαταρία τύπου C. Αφού οι αισθητήρες είναι μικροί και φθηνοί, μπορούν να τοποθετηθούν πολλοί από αυτούς σε ένα δένδρο. Τα δεδομένα συλλέγονται με ένα συμβατό φορητό υπολογιστή αφού ο χειριστής του φτάσει μέχρι τη βάση του δένδρου.

Ο Dawson διαπίστωσε ότι η θερινή ομίχλη είναι υπεύθυνη για το 25% έως 40% του νερού που προσλαμβάνουν οι σεκόγιες ολόκληρο το χρόνο. Τα δένδρα μπορεί επίσης να «πίνουν» νερό απευθείας από την ομίχλη μέσω μιας συμβιωτικής σχέσης με μύκητες που ζουν στα φύλλα τους.

Ο Dawson προβλέπει ότι τα δίκτυα των ασύρματων αισθητήρων (wireless sensor networks) θα αλλάξουν τον τρόπο που οι άνθρωποι κάνουν οικολογικές έρευνες.

Για περισσότερες πληροφορίες, δείτε αυτές τις αναφορές στη βιβλιοθήκη του CD

Block et al., «Migratory movements, depth preferences, and thermal biology of atlantic bluefin tuna» *Science* 293: 1310–14, 2001

«Redwoods», τοποθεσία Ιστού του εργαστηρίου τού καθηγητού Dawson

«Redwood's drinking water from fog», *The Forestry Source*, Νοέμβριος 2002

«Tagging of the Pacific Pelagics», www.toppcensus.org

Αισθητήρας Mica με μπαταρία τύπου C. Έχει μέγεθος περίπου όσο ένα φιλμ φωτογραφικής μηχανής.

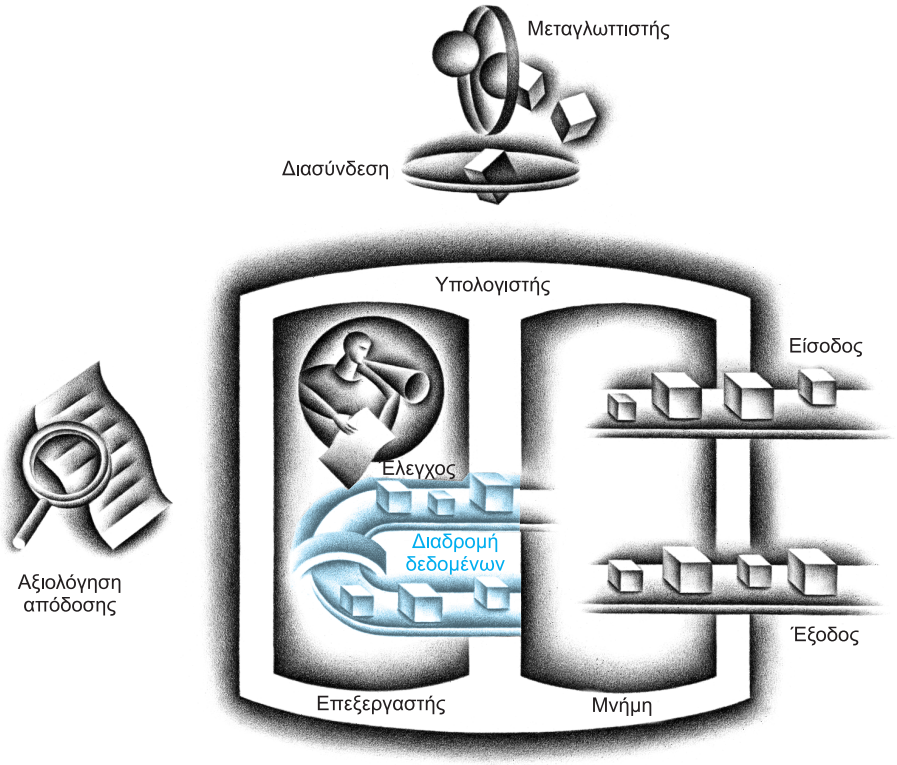
3

Αριθμητική για υπολογιστές

Η αριθμητική ακρίβεια
είναι η πραγματική ψυχή της επιστήμης

Sir D'arcy Wentworth Thompson
On Growth and Form, 1917

Τα πέντε κλασικά συστατικά ενός υπολογιστή



3.1 Εισαγωγή

Οι λέξεις των υπολογιστών αποτελούνται από δυαδικά ψηφία (bit)· επομένως, οι λέξεις μπορούν να αναπαρασταθούν ως δυαδικοί αριθμοί. Αν και οι φυσικοί αριθμοί 0, 1, 2, και ούτω καθεξής μπορούν να αναπαρασταθούν είτε σε δεκαδική είτε σε δυαδική μορφή, τι γίνεται με τους άλλους αριθμούς που εμφανίζονται συχνά; Για παράδειγμα:

- Με ποιο τρόπο αναπαρίστανται οι αρνητικοί αριθμοί;
- Ποιος είναι ο μεγαλύτερος αριθμός που μπορεί να αναπαρασταθεί από μία λέξη υπολογιστή;
- Τι συμβαίνει αν μια λειτουργία δημιουργεί έναν αριθμό μεγαλύτερο από αυτόν που μπορεί να αναπαρασταθεί;
- Τι γίνεται με τα κλάσματα και τους πραγματικούς αριθμούς (real numbers);

Και κάτω από όλες αυτές τις ερωτήσεις βρίσκεται ένα μυστήριο: Με ποιο τρόπο το υλικό πολλαπλασιάζει ή διαιρεί αριθμούς στην πραγματικότητα;

Ο στόχος αυτού του κεφαλαίου είναι να λύσει όχι μόνον αυτό το μυστήριο, αλλά και αυτό της αναπαράστασης των αριθμών, των αλγορίθμων αριθμητικής, του υλικού που ακολουθεί αυτούς τους αλγορίθμους, και των επιπτώσεων όλων αυτών στα σύνολα των εντολών. Αυτές οι γνώσεις μπορεί να εξηγήσουν παραξενιές που έχετε ήδη συναντήσει στους υπολογιστές. (Αν είστε εξοικειωμένοι με τους προσημασμένους δυαδικούς αριθμούς, μπορείτε να παρακάμψετε την επόμενη ενότητα και να συνεχίσετε με την Ενότητα 3.3 στη σελίδα 188.)

3.2 Προσημασμένοι και απρόσημοι αριθμοί

Οι αριθμοί μπορούν να αναπαρασταθούν σε οποιαδήποτε βάση· οι άνθρωποι προτιμούν τη βάση 10 και, όπως είδαμε στο Κεφάλαιο 2, η βάση 2 είναι καλύτερη για τους υπολογιστές. Για να αποφύγουμε τη σύγχυση, προσθέτουμε ως δείκτη (subscript) στους δεκαδικούς αριθμούς το *ten* (δέκα) και στους δυαδικούς αριθμούς το *two* (δύο).

Σε οποιαδήποτε αριθμητική βάση, η τιμή του *i*-οστού ψηφίου *d* είναι

$$d \times \text{βάση}^{i}$$

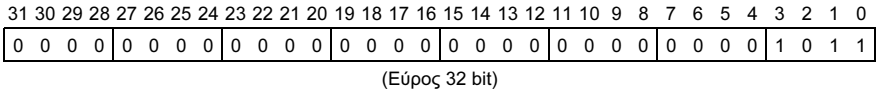
όπου το *i* ξεκινάει από 0 και αυξάνεται από δεξιά προς τα αριστερά. Αυτό οδηγεί σε έναν προφανή τρόπο για την αρίθμηση των bit στη λέξη: απλώς χρησιμοποιούμε τη δύναμη της βάσης για το συγκεκριμένο bit. Για παράδειγμα, ο αριθμός

$$1011_{two}$$

αναπαριστά τον αριθμό

$$(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)_{ten}$$
$$= (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1)_{ten}$$
$$= 8 + 0 + 2 + 1_{ten}$$
$$= 11_{ten}$$

Συνεπώς, τα bit μέσα σε μια λέξη αριθμούνται 0, 1, 2, 3, … από *δεξιά προς τα αριστερά*. Το παρακάτω σχήμα παρουσιάζει την αρίθμηση των bit μέσα σε μια λέξη του επεξεργαστή MIPS και την τοποθέτηση του αριθμού 1011_{ten}:



(Εύρος 32 bit)

Εφόσον οι λέξεις σχεδιάζονται και κατακόρυφα και οριζόντια, οι έννοιες του πιο αριστερού και του πιο δεξιού μπορεί να μην είναι σαφείς. Επομένως, η φράση **λιγότερο σημαντικό bit** (least significant bit) χρησιμοποιείται για αναφορά στο πιο δεξιό bit (το bit 0 στο παραπάνω σχήμα) και η φράση **περισσότερο σημαντικό bit** (most significant bit) για αναφορά στο πιο αριστερό bit (το bit 31).

λιγότερο σημαντικό bit (least significant bit) Το πιο δεξιό bit σε μια λέξη του MIPS.

περισσότερο σημαντικό bit (most significant bit) Το πιο αριστερό bit σε μια λέξη του MIPS.

Η λέξη του MIPS έχει μέγεθος 32 bit, και έτσι μπορούμε να αναπαραστήσουμε 2^{32} διαφορετικές σειρές 32 δυαδικών ψηφίων. Είναι φυσικό να επιτρέψουμε σε αυτούς τους συνδυασμούς να αναπαραστήσουν τους αριθμούς από το 0 έως το $2^{32} - 1$ ($4.294.967.295_{ten}$):

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$$
$$\ldots \qquad\qquad \ldots$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 4.294.967.293_{ten}$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 4.294.967.294_{ten}$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 4.294.967.295_{ten}$$

Αυτό σημαίνει ότι δυαδικοί αριθμοί των 32 bit μπορούν να αναπαρασταθούν με την τιμή του bit επί μια δύναμη του 2 (εδώ, το *xi* σημαίνει το *i*-οστό bit του *x*):

$$(x31 \times 2^{31}) + (x30 \times 2^{30}) + (x29 \times 2^{29}) + \ldots + (x1 \times 2^1) + (x0 \times 2^0)$$

Διασύνδεση υλικού και λογισμικού

Η βάση 2 δεν είναι φυσική για τους ανθρώπους· έχουμε 10 δάκτυλα και θεωρούμε φυσική τη βάση 10. Γιατί δε χρησιμοποίησαν οι υπολογιστές τη δεκαδική βάση; Στην πραγματικότητα, ο πρώτος εμπορικός υπολογιστής διέθετε δεκαδική αριθμητική. Το πρόβλημα ήταν ότι, παρόλα αυτά, ο υπολογιστής χρησιμοποιούσε σήματα ενεργού (on) και ανενεργού (off), και έτσι η αναπαράσταση κάθε δεκαδικού ψηφίου γινόταν και εκεί με πολλά δυαδικά ψηφία. Η δεκαδική αναπαράσταση αποδείχθηκε τόσο μη αποδοτική ώστε οι επόμενοι υπολογιστές επέστρεψαν σε πλήρη δυαδική αναπαράσταση, κάνοντας μετατροπή στη βάση 10 για τα σχετικά σπάνια συμβάντα εισόδου/εξόδου (input/output events).

ΠΑΡΑΔΕΙΓΜΑ

Κώδικας ASCII ή δυαδικοί αριθμοί;

Θα μπορούσαμε να αναπαραστήσουμε αριθμούς με συμβολοσειρές ψηφίων ASCII αντί με ακέραιους (δείτε την Εικόνα 2.21 στη σελίδα 109). Πόσο αυξάνει ο χώρος αποθήκευσης αν ο αριθμός 1 δισεκατομμύριο αναπαρίσταται σε ASCII αντί με έναν ακέραιο 32 bit;

ΑΠΑΝΤΗΣΗ

Ένα δισεκατομμύριο είναι 1 000 000 000, και έτσι θα απαιτούσε 10 ψηφία ASCII, με το καθένα μήκους 8 bit. Επομένως, η αύξηση του χώρου αποθήκευσης θα ήταν (10 × 8)/32, ή 2,5. Εκτός από την αύξηση του χώρου αποθήκευσης, το υλικό για την πρόσθεση, την αφαίρεση, τον πολλαπλασιασμό, και τη διαίρεση τέτοιων αριθμών είναι δύσκολο. Αυτές οι δυσκολίες εξηγούν γιατί οι επαγγελματίες της υπολογιστικής διδάσκονται να πιστεύουν ότι η δυαδική αναπαράσταση είναι φυσική και ότι ο περιστασιακός δεκαδικός υπολογιστής είναι μια γραφικότητα.

Κρατήστε στο μυαλό σας ότι οι παραπάνω σειρές δυαδικών ψηφίων είναι απλώς *αντιπρόσωποι* των αριθμών. Οι αριθμοί στην πραγματικότητα έχουν ένα άπειρο πλήθος από ψηφία, όλα σχεδόν 0 εκτός από λίγα από τα δεξιότερα. Συνήθως, απλώς δεν παρουσιάζουμε τα 0 που προηγούνται.

Για την πρόσθεση, την αφαίρεση, τον πολλαπλασιασμό, και τη διαίρεση αυτών των σειρών δυαδικών ψηφίων μπορεί να σχεδιαστεί το κατάλληλο υλικό. Αν ο αριθμός που είναι το σωστό αποτέλεσμα αυτών των πράξεων δεν μπορεί να αναπαρασταθεί από αυτά τα πιο δεξιά bit του υλικού, λέμε ότι συνέβη *υπερχείλιση* (overflow). Είναι στην ευθύνη του λειτουργικού συστήματος και του προγράμματος να αποφασίσουν τι θα γίνει αν συμβεί υπερχείλιση.

Τα προγράμματα υπολογιστών χειρίζονται τόσο θετικούς όσο και αρνητικούς αριθμούς, και έτσι χρειαζόμαστε μια δυαδική αναπαράσταση που να ξεχωρίζει το θετικό από τον αρνητικό. Η πιο προφανής λύση είναι η προσθήκη ενός ξεχωριστού προσήμου, το οποίο μπορεί να αναπαρασταθεί βολικά από ένα bit· το όνομα γι' αυτή την αναπαράσταση είναι αναπαράσταση *προσήμου και μεγέθους* (sign and magnitude).

Δυστυχώς, η αναπαράσταση προσήμου και μεγέθους έχει αρκετές αδυναμίες. Πρώτον, δεν είναι προφανές πού πρέπει να τοποθετηθεί το bit προσήμου. Στα δεξιά; Στα αριστερά; Οι πρώτοι υπολογιστές δοκίμασαν και τα δύο. Δεύτερον, οι αθροιστές (adders) προσήμου και μεγέθους μπορεί να χρειάζονται ένα επιπλέον βήμα για τον υπολογισμό του προσήμου, επειδή δεν μπορούμε να γνωρίζουμε εκ των προτέρων ποιο θα είναι το κατάλληλο πρόσημο. Τέλος, ένα ξεχωριστό bit προσήμου σημαίνει ότι η αναπαράσταση προσήμου και μεγέθους έχει και θετικό και αρνητικό μηδέν, γεγονός το οποίο μπορεί να οδηγήσει σε προβλήματα τους απρόσεκτους προγραμματιστές. Ως αποτέλεσμα αυτών των αδυναμιών, η αναπαράσταση προσήμου και μεγέθους εγκαταλείφθηκε σύντομα.

Στην έρευνα για μια πιο ελκυστική εναλλακτική επιλογή, προέκυψε το ερώτημα ποιο θα ήταν το αποτέλεσμα για απρόσημους (μη προσημασμένους) αριθμούς αν προσπαθούσαμε να αφαιρέσουμε ένα μεγάλο αριθμό από ένα μικρό. Η απάντηση είναι ότι θα δοκίμαζε να δανειστεί από μια σειρά αρχικών 0, ώστε το αποτέλεσμα να έχει μια σειρά από αρχικά 1.

Λόγω της έλλειψης καλύτερης προφανούς εναλλακτικής επιλογής, η τελική λύση ήταν η επιλογή της αναπαράστασης που έκανε το υλικό απλούστερο: αρχικά ψηφία 0 σημαίνουν θετικό αριθμό, και αρχικά ψηφία 1 σημαίνουν αρνητικό. Αυτή η σύμβαση για την αναπαράσταση προσημασμένων δυαδικών αριθμών ονομάζεται αναπαράσταση *συμπληρώματος ως προς δύο* (two's complement representation):

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$
...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 2.147.483.645_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 2.147.483.646_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2.147.483.647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -2.147.483.648_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -2.147.483.647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -2.147.483.646_{ten}$
...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = -3_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -1_{ten}$

Οι θετικοί μισοί αριθμοί, από το 0 έως το $2.147.483.647_{ten}$ ($2^{31} - 1$), χρησιμοποιούν την ίδια αναπαράσταση όπως πριν. Η σειρά bit $1000\ldots0000_{two}$ αναπαριστά τον πιο αρνητικό αριθμό $-2.147.483.648_{ten}$ (-2^{31}). Ακολουθείται από ένα σύνολο αρνητικών αριθμών που μειώνονται: $-2.147.483.647_{ten}$ ($1000\ldots0001_{two}$) έως -1_{ten} ($1111\ldots1111_{two}$).

Η αναπαράσταση συμπληρώματος ως προς δύο έχει έναν αρνητικό αριθμό, τον $-2.147.483.648_{ten}$, ο οποίος δεν έχει αντίστοιχο θετικό. Αυτή η έλλειψη ισορροπίας προκαλεί μια ανησυχία για τον απρόσεκτο προγραμματιστή, αλλά η αναπαράσταση προσήμου και μεγέθους είχε προβλήματα τόσο για τον προγραμματιστή όσο και για το σχεδιαστή του υλικού. Συνεπώς, κάθε υπολογιστής σήμερα χρησιμοποιεί δυαδική αναπαράσταση συμπληρώματος ως προς δύο για προσημασμένους αριθμούς.

Η αναπαράσταση συμπληρώματος ως προς δύο έχει το πλεονέκτημα ότι όλοι οι αρνητικοί αριθμοί έχουν τη μονάδα (1) στο πιο σημαντικό bit. Συνεπώς, το υλικό χρειάζεται να ελέγξει μόνον αυτό το bit για να διαπιστώσει αν ένας αριθμός είναι θετικός ή αρνητικός (ενώ το 0 θεωρείται θετικός). Αυτό το bit συχνά ονομάζεται *bit προσήμου* (sign bit). Αναγνωρίζοντας το ρόλο του bit προσήμου, μπορούμε να αναπαραστήσουμε θετικούς και αρνητικούς αριθμούς 32 bit χρησιμοποιώντας την τιμή κάθε bit πολλαπλασιασμένη με μια δύναμη του 2:

$$(x31 \times -2^{31}) + (x30 \times 2^{30}) + (x29 \times 2^{29}) + \ldots + (x1 \times 2^{1}) + (x0 \times 2^{0})$$

Το bit προσήμου πολλαπλασιάζεται με το -2^{31}, και τα υπόλοιπα bit πολλαπλασιάζονται στη συνέχεια με θετικές εκδόσεις των αντίστοιχων τιμών βάσης.

ΠΑΡΑΔΕΙΓΜΑ

ΑΠΑΝΤΗΣΗ

Μετατροπή δυαδικού σε δεκαδικό

Ποια είναι η δεκαδική τιμή του παρακάτω αριθμού 32 bit σε αναπαράσταση συμπληρώματος ως προς δύο;

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$$

Με αντικατάσταση των τιμών των bit του αριθμού στην παραπάνω εξίσωση παίρνουμε:

$$(1 \times -2^{31}) + (1 \times 2^{30}) + (1 \times 2^{29}) + \ldots + (1 \times 2^{2}) + (0 \times 2^{1}) + (0 \times 2^{0})$$
$$= -2^{31} + 2^{30} + 2^{29} + \ldots + 2^{2} + 0 + 0$$
$$= -2.147.483.648_{ten} + 2.147.483.644_{ten}$$
$$= -4_{ten}$$

Θα δούμε μια συντόμευση για την απλοποίηση της μετατροπής σύντομα.

Όπως ακριβώς μια πράξη με απρόσημους αριθμούς μπορεί να προκαλέσει υπερχείλιση της ικανότητας του υλικού να αναπαραστήσει το αποτέλεσμα, το ίδιο μπορεί να κάνει και μια πράξη με αριθμούς συμπληρώματος ως προς δύο. Υπερχείλιση (overflow) συμβαίνει όταν το αριστερότερο διατηρούμενο bit της δυαδικής σειράς δεν είναι ίδιο με τον άπειρο αριθμό ψηφίων στα αριστερά (το bit προσήμου είναι λάθος): 0 στα αριστερά της σειράς των bit όταν ο αριθμός είναι αρνητικός, ή 1 όταν ο αριθμός είναι θετικός.

Διασύνδεση υλικού και λογισμικού

Η διαφορά των προσημασμένων από τους απρόσημους αριθμούς ισχύει για τις φορτώσεις όπως και για τις αριθμητικές πράξεις. Η *λειτουργία* μιας προσημασμένης φόρτωσης είναι η αντιγραφή του προσήμου επαναληπτικά για τη συμπλήρωση του υπόλοιπου καταχωρητή — ονομάζεται *επέκταση προσήμου* (sign extension) — αλλά ο *σκοπός* της είναι η τοποθέτηση μιας σωστής αναπαράστασης του αριθμού μέσα σε εκείνον τον καταχωρητή. Οι απρόσημες φορτώσεις απλώς συμπληρώνουν με 0 τις θέσεις στα αριστερά των δεδομένων αφού ο αριθμός που αναπαρίσταται με τη σειρά των bit είναι απρόσημος.

Όταν φορτώνουμε μια λέξη 32 bit σε έναν καταχωρητή 32 bit, το σημείο αυτό δεν έχει νόημα· οι προσημασμένες και οι απρόσημες φορτώσεις είναι πανομοιότυπες. Ο MIPS παρέχει δύο παραλλαγές φόρτωσης byte: η εντολή *load byte* (`lb`) χειρίζεται το byte ως προσημασμένο αριθμό και έτσι επεκτείνει το πρόσημο για να γεμίσει τα 24 αριστερότερα bit του καταχωρητή, ενώ η εντολή *load byte unsigned* (`lbu`) δουλεύει με απρόσημους ακεραίους. Εφόσον τα προγράμματα της C σχεδόν πάντα χρησιμοποιούν byte για την αναπαράσταση χαρακτήρων αντί να θεωρούν τα byte προσημασμένους ακεραίους πολύ μικρού μήκους, η `lbu` στην πράξη χρησιμοποιείται αποκλειστικά για φόρτωση byte. Για παρόμοιους λόγους, η εντολή *load half* (`lh`) χειρίζεται την ημιλέξη (half word) ως προσημασμένο αριθμό και έτσι επεκτείνει το πρόσημο για να γεμίσει τα 16 πιο αριστερά bit του καταχωρητή, ενώ η εντολή *load half-word unsigned* (`lhu`) δουλεύει με απρόσημους ακεραίους.

Διασύνδεση υλικού και λογισμικού

Αντίθετα με τους αριθμούς που εξετάσαμε πιο πάνω, οι διευθύνσεις της μνήμης φυσιολογικά ξεκινούν από το 0 και συνεχίζουν μέχρι τη μεγαλύτερη δυνατή διεύθυνση. Για να το πούμε διαφορετικά, οι αρνητικές διευθύνσεις δεν έχουν νόημα. Έτσι, τα προγράμματα πρέπει κάποιες φορές να χειριστούν αριθμούς που μπορούν να είναι θετικοί ή αρνητικοί, και κάποιες φορές αριθμούς που μπορούν να είναι μόνο θετικοί. Μερικές γλώσσες προγραμματισμού αντικατοπτρίζουν αυτή τη διάκριση. Η C, για παράδειγμα, ονομάζει τους πρώτους από αυτούς *ακεραίους* (integers) — δηλώνονται ως `int` στο πρόγραμμα — και τους τελευταίους *απρόσημους ακεραίους* (unsigned integers) — δηλώνονται ως `unsigned int` στο πρόγραμμα. Κάποιες οδηγίες στυλ προγραμματισμού σε C συνιστούν ακόμη και να δηλώνονται οι αριθμοί της πρώτης περίπτωσης ως `signed int` για να διατηρήσουν τη διάκριση σαφή.

Οι εντολές σύγκρισης πρέπει να χειριστούν αυτή τη διάκριση. Μερικές φορές, μια σειρά bit με 1 στο πιο σημαντικό bit αναπαριστά έναν αρνητικό αριθμό, που φυσικά είναι μικρότερος από οποιονδήποτε θετικό αριθμό, ο οποίος πρέπει να έχει 0 στο πιο σημαντικό bit. Με τους απρόσημους ακεραίους από την άλλη, το 1 στο πιο σημαντικό bit αναπαριστά έναν αριθμό που είναι μεγαλύτερος από οποιονδήποτε αρχίζει με 0. (Θα εκμεταλλευθούμε αυτή τη διπλή σημασία του πιο σημαντικού bit για να μειώσουμε το κόστος των ελέγχων ορίων των πινάκων, λίγες σελίδες πιο κάτω.)

Ο MIPS παρέχει δύο εκδόσεις της σύγκρισης set on less than για να χειριστεί αυτές τις εναλλακτικές περιπτώσεις. Οι εντολές *set on less than* (`slt`) και *set on less than immediate* (`slti`) δουλεύουν με προσημασμένους ακεραίους. Οι απρόσημοι ακέραιοι συγκρίνονται με τις εντολές *set on less than unsigned* (`sltu`) και *set on less than immediate unsigned* (`sltiu`).

ΠΑΡΑΔΕΙΓΜΑ

Προσημασμένη και απρόσημη σύγκριση

Θεωρήστε ότι ο καταχωρητής `$s0` περιέχει το δυαδικό αριθμό

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two}$$

και ο καταχωρητής `$s1` περιέχει το δυαδικό αριθμό

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two}$$

Ποιες είναι οι τιμές των καταχωρητών `$t0` και `$t1` μετά από τις παρακάτω εντολές;

```
slt   $t0,$s0,$s1   # προσημασμένη σύγκριση
sltu  $t1,$s0,$s1   # απρόσημη σύγκριση
```

ΑΠΑΝΤΗΣΗ

Η τιμή στον καταχωρητή `$s0` αναπαριστά το –1 αν είναι ακέραια και το $4.294.967.295_{ten}$ αν είναι απρόσημη ακέραια. Η τιμή στον καταχωρητή `$s1` αναπαριστά το 1 σε οποιαδήποτε περίπτωση. Επομένως, ο καταχωρητής `$t0` περιέχει την τιμή 1, αφού $-1_{ten} < 1_{ten}$, και ο καταχωρητής `$t1` περιέχει την τιμή 0, αφού $4.294.967.295_{ten} > 1_{ten}$.

Πριν προχωρήσουμε στην πρόσθεση και την αφαίρεση, ας εξετάσουμε μερικές χρήσιμες συντομεύσεις όταν δουλεύουμε με αριθμούς σε συμπλήρωμα ως προς δύο.

Η πρώτη συντόμευση είναι ένας εύκολος τρόπος για να αλλάξουμε το πρόσημο ενός αριθμού σε συμπλήρωμα ως προς δύο. Απλώς αντιστρέφουμε κάθε 0 σε 1 και κάθε 1 σε 0, και στη συνέχεια προσθέτουμε 1 στο αποτέλεσμα. Αυτή η συντόμευση βασίζεται στην παρατήρηση ότι το άθροισμα ενός αριθμού και της αντεστραμμένης αναπαράστασής του πρέπει να είναι $111\ldots111_{two}$, που αναπαριστά το –1. Αφού $x + \bar{x} \equiv -1$, τότε $x + \bar{x} + 1 = 0$ ή $x + 1 = -x$.

ΠΑΡΑΔΕΙΓΜΑ

Συντόμευση υπολογισμού αντιθέτου

Βρείτε τον αντίθετο του 2_{ten} και μετά ελέγξτε το αποτέλεσμα υπολογίζοντας τον αντίθετο του -2_{ten}.

ΑΠΑΝΤΗΣΗ

2_{ten} = `0000 0000 0000 0000 0000 0000 0000 0010`$_{two}$

Ο αντίθετος αυτού του αριθμού με αντιστροφή των bit και πρόσθεση του ένα,

```
      1111 1111 1111 1111 1111 1111 1111 1101two
+                                          1two
-------------------------------------------------
=     1111 1111 1111 1111 1111 1111 1111 1110two
=     -2ten
```

Ακολουθώντας την άλλη κατεύθυνση, ο αριθμός

`1111 1111 1111 1111 1111 1111 1111 1110`$_{two}$

πρώτα αντιστρέφεται και στη συνέχεια αυξάνεται:

```
      0000 0000 0000 0000 0000 0000 0000 0001two
+                                          1two
-------------------------------------------------
=     0000 0000 0000 0000 0000 0000 0000 0010two
=     2ten
```

Η δεύτερη συντόμευση μας λέει με ποιο τρόπο να μετατρέψουμε ένα δυαδικό αριθμό, που αναπαρίσταται από *n* bit, σε έναν αριθμό που αναπαρίσταται με περισσότερα από *n* bit. Για παράδειγμα, το άμεσο πεδίο στις εντολές load, store, branch, add, και set on less than περιέχει έναν αριθμό 16 bit σε συμπλήρωμα ως προς δύο, που αναπαριστά ένα εύρος τιμών από το -32.768_{ten} (-2^{15}) έως το 32.767_{ten} ($2^{15} - 1$). Για την πρόσθεση ενός άμεσου πεδίου σε έναν καταχωρητή 32 bit, ο υπολογιστής πρέπει να μετατρέψει τον αριθμό 16 bit στον ισοδύναμό του των 32 bit. Η συντόμευση συνίσταται στη λήψη του πιο σημαντικού bit από τη μικρότερη ποσότητα — το bit προσήμου — και την αναπαραγωγή του για τη συμπλήρωση των νέων bit της μεγαλύτερης ποσότητας. Τα παλιά bit απλώς αντιγράφονται στο δεξιό τμήμα της νέας λέξης. Αυτή η συντόμευση ονομάζεται συνήθως *επέκταση προσήμου* (sign extension).

Η συντόμευση της επέκτασης προσήμου

ΠΑΡΑΔΕΙΓΜΑ

Μετατρέψτε τις δυαδικές εκδόσεις 16 bit των 2_{ten} και -2_{ten} σε δυαδικούς αριθμούς 32 bit.

ΑΠΑΝΤΗΣΗ

Η δυαδική έκδοση 16 bit του αριθμού 2 είναι

$$0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$$

Αυτή μετατρέπεται σε ένα δυαδικό αριθμό 32 bit, με τη δημιουργία 16 αντιγράφων της τιμής τού πιο σημαντικού bit (0) και τοποθέτησή τους στο αριστερό μισό της λέξης. Το δεξιό μισό παίρνει την παλιά τιμή:

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$$

Ας υπολογίσουμε τον αντίθετο της έκδοσης 16 bit του 2, χρησιμοποιώντας τη συντόμευση που παρουσιάσαμε νωρίτερα. Έτσι το,

$$0000\ 0000\ 0000\ 0010_{two}$$

γίνεται

$$\begin{array}{r} 1111\ 1111\ 1111\ 1101_{two} \\ +\qquad\qquad\qquad\quad 1_{two} \\ \hline = 1111\ 1111\ 1111\ 1110_{two} \end{array}$$

Η δημιουργία μιας έκδοσης 32 bit του αρνητικού αριθμού σημαίνει αντιγραφή του bit προσήμου 16 φορές και τοποθέτησή του στα αριστερά:

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2_{ten}$$

Αυτό το τέχνασμα δουλεύει επειδή οι θετικοί αριθμοί σε συμπλήρωμα ως προς δύο, στην πραγματικότητα, έχουν ένα άπειρο πλήθος 0 στα αριστερά, και

εκείνοι που είναι αρνητικοί σε συμπλήρωμα ως προς δύο έχουν ένα άπειρο πλήθος 1. Η σειρά bit που αναπαριστά έναν αριθμό κρύβει τα bit που προηγούνται για να χωρέσει στο εύρος του υλικού· η επέκταση προσήμου απλώς επαναφέρει κάποια από αυτά.

Η τρίτη συντόμευση μειώνει το κόστος για τον έλεγχο $0 \le x < y$, πράγμα που ταιριάζει με τον έλεγχο για αριθμοδείκτη εκτός ορίων (index out-of-bounds) για τους πίνακες. Το κλειδί είναι ότι οι αρνητικοί ακέραιοι σε σημειογραφία συμπληρώματος ως προς δύο μοιάζουν με μεγάλους αριθμούς σε απρόσημη σημειογραφία· αυτό σημαίνει ότι το πιο σημαντικό bit είναι ένα bit προσήμου στην πρώτη σημειογραφία, αλλά ένα μεγάλο μέρος του αριθμού στη δεύτερη. Έτσι, μια απρόσημη σύγκριση $x < y$ ελέγχει επίσης αν ο x είναι αρνητικός.

ΠΑΡΑΔΕΙΓΜΑ

Συντόμευση ελέγχου ορίων

Χρησιμοποιήστε αυτή τη συντόμευση για να επιταχύνετε τον έλεγχο αριθμοδείκτη εκτός ορίων (index-out-of-bounds): άλμα στην ετικέτα `IndexOutOfBounds` αν `$a1` ≥ `$t2` ή αν ο `$a1` είναι αρνητικός.

ΑΠΑΝΤΗΣΗ

Ο κώδικας ελέγχου απλώς χρησιμοποιεί την εντολή `sltu` για να κάνει και τους δύο ελέγχους:

```
sltu $t0,$a1,$t2 # Προσωρ. καταχ. $t0=0 αν k>=length ή k<0
beq  $t0,$zero,IndexOutOfBounds # αν όχι, μετάβαση στην
                                # ετικέτα Error
```

Περίληψη

Το κύριο σημείο σε αυτή την ενότητα είναι το γεγονός ότι στον κόσμο των υπολογιστών χρειάζεται να αναπαραστήσουμε τόσο τους θετικούς όσο και τους αρνητικούς ακεραίους και, μολονότι υπάρχουν υπέρ και κατά για κάθε εναλλακτική λύση, η κυρίαρχη επιλογή από το 1965 είναι το συμπλήρωμα ως προς δύο. Η Εικόνα 3.1 παρουσιάζει τις προσθήκες στη συμβολική γλώσσα του MIPS που αποκαλύψαμε σε αυτή την ενότητα. (Η γλώσσα μηχανής του MIPS παρουσιάζεται επίσης στην πράσινη κάρτα στην αρχή του βιβλίου.)

Αυτοεξέταση

Ποιου τύπου μεταβλητή, που μπορεί να περιέχει τον αριθμό $1.000.000.000_{ten}$ απαιτεί τον περισσότερο χώρο μνήμης;

1. `int` στη C
2. συμβολοσειρά ASCII στη C
3. `string` στην Java (που χρησιμοποιεί Unicode)

Τελεστέοι του MIPS

Όνομα	Παράδειγμα	Σχόλια
32 καταχωρητές	`$s0-$s7, $t0-$t9, $gp, $fp, $zero, $sp, $ra, $at`	Γρήγορες θέσεις για δεδομένα. Στον MIPS, τα δεδομένα πρέπει να βρίσκονται σε καταχωρητές για να εκτελεστούν αριθμητικές πράξεις. Ο καταχωρητής `$zero` του MIPS έχει πάντα την τιμή 0. Ο καταχωρητής `$at` δεσμεύεται για το χειρισμό μεγάλων σταθερών από το συμβολομεταφραστή.
2^{30} λέξεις μνήμης	Memory[0], Memory[4], . . . , Memory[4294967292]	Προσπελάζονται μόνον από εντολές μεταφοράς δεδομένων. Ο MIPS χρησιμοποιεί διευθύνσεις byte, και έτσι οι διευθύνσεις διαδοχικών λέξεων διαφέρουν κατά 4. Η μνήμη κρατάει δομές δεδομένων, όπως πίνακες και διασκορπισμένους καταχωρητές, όπως αυτοί που αποθηκεύονται στις κλήσεις διαδικασιών.

Συμβολική γλώσσα του MIPS

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Αριθμητικές πράξεις	add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Τρεις τελεστέοι
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2 - $s3`	Τρεις τελεστέοι
	add immediate	`addi $s1,$s2,100`	`$s1 = $s2 + 100`	+ σταθερά
Μεταφορά δεδομένων	load word	`lw $s1,100($s2)`	`$s1 = Memory[$s2+100]`	Λέξη από τη μνήμη σε καταχωρητή
	store word	`sw $s1,100($s2)`	`Memory[$s2+100] = $s1`	Λέξη από καταχωρητή στη μνήμη
	load half unsigned	`lhu $s1,100($s2)`	`$s1 = Memory[$s2 + 100]`	Ημιλέξη από τη μνήμη σε καταχωρητή
	store half	`sh $s1,100($s2)`	`Memory[$s2 + 100] = $s1`	Ημιλέξη από καταχωρητή στη μνήμη
	load byte unsigned	`lbu s1,100($s2)`	`$s1 = Memory[$s2 + 100]`	Byte από τη μνήμη σε καταχωρητή
	store byte	`sb $s1,100($s2)`	`Memory[$s2 + 100] = $s1`	Byte από καταχωρητή στη μνήμη
	load upper immediate	`lui $s1,100`	`$s1 = 100 * 216`	Φορτώνει σταθερά στα υψηλότερα 16 bit
Λογικές πράξεις	and	`and $s1,$s2,$s3`	`$s1 = $s2 & $s3`	Τρεις τελεστέοι καταχωρητές· AND bit προς bit
	or	`or $s1,$s2,$s3`	`$s1 = $s2 | $s3`	Τρεις τελεστέοι καταχωρητές· OR bit προς bit
	nor	`nor $s1,$s2,$s3`	`$s1 = ~ ($s2 | $s3)`	Τρεις τελεστέοι καταχωρητές· NOR bit προς bit
	and immediate	`andi $s1,$s2,100`	`$s1 = $s2 & 100`	AND bit προς bit με σταθερά
	or immediate	`ori s1,$s2,100`	`$s1 = $s2 | 100`	OR bit προς bit με σταθερά
	shift left logical	`sll $s1,$s2,10`	`$s1 = $s2 << 10`	Αριστερή ολίσθηση με σταθερά
	shift right logical	`srl $s1,$s2,10`	`$s1 = $s2 >> 10`	Δεξιά ολίσθηση με σταθερά
Διακλάδωση υπό συνθήκη	branch on equal	`beq $s1,$s2,25`	αν `($s1 == $s2)` πήγαινε στο PC + 4 + 100	Έλεγχος ισότητας· διακλάδωση σχετική ως προς PC
	branch on not equal	`bne $s1,$s2,25`	αν `($s1 != $s2)` πήγαινε στο PC + 4 + 100	Έλεγχος ανισότητας· διακλάδωση σχετική ως προς PC
	set on less than	`Slt $s1,$s2,$s3`	αν `($s2 < $s3)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από· συμπλήρωμα ως προς δύο
	set less than immediate	`slti $s1,$s2,100`	αν `($s2 < 100)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από σταθερά· συμπλήρωμα ως προς δύο
	set less than unsigned	`sltu s1,$s2,$s3`	αν `($s2 < $s3)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από απρόσημοι αριθμοί
	set less than immediate unsigned	`sltiu s1,$s2,100`	αν `($s2 < 100)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από σταθερά απρόσημοι αριθμοί
Άλμα χωρίς συνθήκη	jump	`j 2500`	μετάβαση στο 10000	Άλμα στη διεύθυνση προορισμού
	jump register	`jr $ra`	μετάβαση στον `$ra`	Για εντολή switch και για επιστροφή διαδικασίας
	jump and link	`jal 2500`	`$ra` = PC + 4· μετάβαση στο 10000	Για κλήση διαδικασίας

ΕΙΚΟΝΑ 3.1 Η αρχιτεκτονική του MIPS που έχουμε αποκαλύψει έως τώρα. Το χρώμα επισημαίνει τα τμήματα από αυτό το κεφάλαιο, που προστέθηκαν στην αρχιτεκτονική του MIPS η οποία αποκαλύφθηκε στο Κεφάλαιο 2 (Εικόνα 2.27 στη σελίδα 123). Η γλώσσα μηχανής του MIPS παρουσιάζεται στην κάρτα συνοπτικής αναφοράς του MIPS στην αρχή του βιβλίου.

Επιπλέον ανάπτυξη: Η αναπαράσταση συμπληρώματος ως προς δύο παίρνει το όνομά της από τον κανόνα ότι το απρόσημο άθροισμα ενός αριθμού *n* bit και του αντιθέτου του είναι 2^n. Επομένως, το συμπλήρωμα ή η αλλαγή προσήμου ενός αριθμού *x* σε συμπλήρωμα ως προς δύο είναι $2^n - x$.

Μια τρίτη εναλλακτική αναπαράσταση ονομάζεται *συμπλήρωμα ως προς ένα* (one's complement). Ο αντίθετος ενός αριθμού σε συμπλήρωμα ως προς ένα βρίσκεται με την αντιστροφή κάθε bit από 0 σε 1 και από 1 σε 0, το οποίο βοηθάει στην εξήγηση του ονόματος, αφού το συμπλήρωμα του *x* είναι $2^n - x - 1$. Ήταν επίσης μια απόπειρα καλύτερης λύσης από την αναπαράσταση προσήμου και μεγέθους, και αρκετοί επιστημονικοί υπολογιστές χρησιμοποίησαν αυτή τη σημειογραφία. Αυτή η αναπαράσταση είναι παρόμοια με το συμπλήρωμα ως προς δύο εκτός από το γεγονός ότι έχει επίσης δύο μηδέν: το $00...00_{two}$ είναι το θετικό 0, και το $11...11_{two}$ είναι το αρνητικό 0. Ο πιο αρνητικός αριθμός $10...000_{two}$ αναπαριστά το $-2.147.483.647_{ten}$, και έτσι οι θετικοί και οι αρνητικοί είναι σε ισορροπία. Οι αθροιστές συμπληρώματος ως προς ένα χρειάζονται ένα επιπλέον βήμα για να αφαιρέσουν έναν αριθμό, και γι' αυτόν το λόγο σήμερα κυριαρχεί το συμπλήρωμα ως προς δύο.

Μια τελευταία σημειογραφία, την οποία θα δούμε όταν θα αναλύσουμε την κινητή υποδιαστολή (floating point), είναι η αναπαράσταση της πιο αρνητικής τιμής με $00...000_{two}$ και της πιο θετικής τιμής με $11...11_{two}$, όπου το 0 έχει τυπικά την τιμή $10...00_{two}$. Αυτή ονομάζεται **πολωμένη σημειογραφία** (biased notation), αφού πολώνει τον αριθμό ώστε ο αριθμός συν την πόλωση να έχει μη αρνητική αναπαράσταση.

πολωμένη σημειογραφία (biased notation) Σημειογραφία που αναπαριστά την πιο αρνητική τιμή με $00...000_{two}$ και την πιο θετική τιμή με $11...11_{two}$, και στην οποία το 0 έχει τυπικά την τιμή $10...00_{two}$, πολώνοντας με αυτόν τον τρόπο τον αριθμό ώστε ο αριθμός συν την πόλωση να έχει μια μη αρνητική αναπαράσταση.

Επιπλέον ανάπτυξη: Για προσημασμένους δεκαδικούς αριθμούς, χρησιμοποιήσαμε το «–» για να αναπαραστήσουμε τους αρνητικούς επειδή δεν υπάρχουν όρια στο μέγεθος ενός δεκαδικού αριθμού. Με δεδομένο το πεπερασμένο μέγεθος λέξης, οι δυαδικές και δεκαεξαδικές σειρές bit μπορούν να κωδικοποιήσουν το πρόσημο και, επομένως, δε χρησιμοποιούμε κανονικά το «+» ή το «–» με τη δυαδική ή τη δεκαεξαδική σημειογραφία.

Αφαίρεση: ο «κατεργάρης» φίλος της πρόσθεσης

No. 10, Top Ten Courses for Athletes at a Football Factory, David Letterman et al., *Book of Top Ten Lists*, 1990

3.3 Πρόσθεση και αφαίρεση

Η πρόσθεση στους υπολογιστές είναι ακριβώς αυτό που θα περιμένατε. Ψηφία προστίθενται bit με bit από δεξιά προς τα αριστερά, με κρατούμενα (carries) να μεταφέρονται στο επόμενο ψηφίο αριστερά, όπως θα κάνατε με το χέρι. Η αφαίρεση χρησιμοποιεί πρόσθεση: ο κατάλληλος τελεστέος απλώς αλλάζει πρόσημο πριν προστεθεί.

ΠΑΡΑΔΕΙΓΜΑ

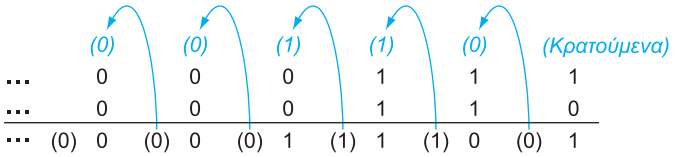
Δυαδική πρόσθεση και αφαίρεση

Ας προσπαθήσουμε να προσθέσουμε το 6_{ten} στο 7_{ten} και, μετά, να αφαιρέσουμε το 6_{ten} από το 7_{ten} επίσης σε δυαδική μορφή.

ΑΠΑΝΤΗΣΗ

$$
\begin{array}{ll}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\
+ & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\
\hline
= & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101_{two} = 13_{ten}
\end{array}
$$

Όλη η δράση συμβαίνει στα 4 bit στα δεξιά· η Εικόνα 3.2 παρουσιάζει τα αθροίσματα και τα κρατούμενα. Τα κρατούμενα παρουσιάζονται σε παρενθέσεις, ενώ τα βέλη δείχνουν πώς αυτά μεταβιβάζονται.



ΕΙΚΟΝΑ 3.2 Δυαδική πρόσθεση, με παρουσίαση των κρατουμένων από δεξιά προς τα αριστερά. Το δεξιότερο bit προσθέτει 1 στο 0, με αποτέλεσμα το άθροισμα αυτού του bit να είναι 1 και το κρατούμενο που παράγεται από αυτό το bit να είναι 0. Επομένως, η πράξη για το δεύτερο ψηφίο στα δεξιά είναι 0 + 1 + 1. Αυτή παράγει ένα 0 γι' αυτό το bit του αθροίσματος, και ένα κρατούμενο 1. Το τρίτο ψηφίο είναι το άθροισμα 1 + 1 + 1, που παράγει ένα κρατούμενο 1 και ένα bit αθροίσματος 1. Το τέταρτο bit είναι 1 + 0 + 0, παράγοντας ένα άθροισμα 1 και κανένα κρατούμενο.

Η αφαίρεση του 6_{ten} από το 7_{ten} μπορεί να γίνει απευθείας:

$$
\begin{array}{ll}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\
- & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\
\hline
= & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}
\end{array}
$$

ή μέσω πρόσθεσης με τη χρήση της αναπαράστασης ως προς δύο του –6:

$$
\begin{array}{ll}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\
+ & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010_{two} = -6_{ten} \\
\hline
= & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}
\end{array}
$$

Αναφέραμε νωρίτερα ότι συμβαίνει υπερχείλιση όταν το αποτέλεσμα μιας πράξης δεν μπορεί να αναπαρασταθεί στο διαθέσιμο υλικό — στην περίπτωση αυτή μια λέξη 32 bit. Πότε μπορεί να συμβεί υπερχείλιση στην πρόσθεση; Κατά την πρόσθεση τελεστέων με διαφορετικά πρόσημα, δεν μπορεί να συμβεί υπερχείλιση. Ο λόγος είναι ότι το άθροισμα δεν μπορεί να είναι μεγαλύτερο από έναν από τους τελεστέους. Για παράδειγμα, –10 + 4 = –6. Εφόσον οι τελεστέοι χωρούν σε 32 bit και το άθροισμα δεν είναι μεγαλύτερο από έναν από αυτούς, πρέπει να χωράει και αυτό σε 32 bit. Άρα, δεν μπορεί να συμβεί υπερχείλιση κατά την πρόσθεση θετικών με αρνητικούς τελεστέους.

Υπάρχουν παρόμοιοι περιορισμοί στην εμφάνιση υπερχείλισης κατά την αφαίρεση, αλλά ισχύει η αντίθετη ακριβώς αρχή: όταν τα πρόσημα των τελεστέων είναι *ίδια*, δεν μπορεί να συμβεί υπερχείλιση. Για να γίνει κατανοητό, θυμηθείτε ότι $x - y = x + (-y)$ επειδή αφαιρούμε αλλάζοντας το πρόσημο του δεύτερου τελεστέου και στη συνέχεια προσθέτουμε. Έτσι, όταν αφαιρούμε τελεστέους του ίδιου προσήμου, καταλήγουμε σε *πρόσθεση* τελεστέων *διαφορετικών* προσήμων. Από την προηγούμενη παράγραφο, γνωρίζουμε ότι δεν μπορεί να συμβεί υπερχείλιση ούτε σε αυτή την περίπτωση.

Έχοντας εξετάσει πότε δεν μπορεί να συμβεί υπερχείλιση στην πρόσθεση και την αφαίρεση, δεν έχουμε ακόμα βρει με ποιο τρόπο θα την ανακαλύψουμε όταν *συμβαίνει*. Υπερχείλιση συμβαίνει όταν κατά την πρόσθεση δύο θετικών αριθμών το αποτέλεσμα είναι αρνητικό, και αντίστροφα. Δεν υπάρχει αμφιβολία ότι η πρόσθεση ή η αφαίρεση δύο αριθμών 32 bit μπορούν να παραγάγουν ένα αποτέλεσμα που χρειάζεται 33 bit για να αναπαρασταθεί πλήρως. Η έλλειψη 33^{ου} bit σημαίνει ότι, όταν συμβαίνει υπερχείλιση, στο bit προσήμου ανατίθεται ένα bit της *τιμής* του αποτελέσματος αντί για το κατάλληλο πρόσημο του αποτελέσματος. Εφόσον χρειαζόμαστε μόνον ένα επιπλέον bit, μόνο το bit προσήμου μπορεί να είναι λάθος. Αυτό σημαίνει την παρουσία κρατουμένου στο bit προσήμου.

Υπερχείλιση συμβαίνει στην αφαίρεση όταν αφαιρείται ένας αρνητικός αριθμός από ένα θετικό και το αποτέλεσμα είναι αρνητικό, ή όταν αφαιρείται ένας θετικός αριθμός από έναν αρνητικό και το αποτέλεσμα είναι θετικό. Αυτό σημαίνει ότι παράγεται ένα δάνειο (borrow) από το bit προσήμου. Η Εικόνα 3.3 παρουσιάζει τους συνδυασμούς των πράξεων, τελεστέων, και αποτελεσμάτων που δείχνουν υπερχείλιση.

Μόλις είδαμε με ποιο τρόπο ανακαλύπτουμε την υπερχείλιση για αριθμούς σε συμπλήρωμα ως προς δύο σε έναν υπολογιστή. Τι γίνεται με τους απρόσημους ακεραίους; Οι απρόσημοι ακέραιοι χρησιμοποιούνται συνήθως για διευθύνσεις μνήμης όπου οι υπερχειλίσεις αγνοούνται.

Γι' αυτόν το λόγο, ο σχεδιαστής του υπολογιστή πρέπει να παρέχει ένα τρόπο για την παράβλεψη της υπερχείλισης σε μερικές περιπτώσεις και την αναγνώρισή της σε άλλες. Η λύση στον MIPS ήταν η ύπαρξη δύο ειδών αριθμητικών εντολών για την αναγνώριση των δύο επιλογών:

- Οι εντολές add (`add`), add immediate (`addi`), και subtract (`sub`) προκαλούν εξαιρέσεις (exceptions) κατά την υπερχείλιση.
- Οι εντολές add unsigned (`addu`), add immediate unsigned (`addiu`), και subtract unsigned (`subu`) *δεν* προκαλούν εξαιρέσεις κατά την υπερχείλιση.

Επειδή η C αγνοεί τις υπερχειλίσεις, οι μεταγλωττιστές του MIPS παράγουν πάντοτε τις απρόσημες εκδόσεις των αριθμητικών εντολών `addu`, `addiu`, και `subu`, ανεξάρτητα από τον τύπο των μεταβλητών. Ωστόσο, οι μεταγλωττιστές της γλώσσας Fortran του MIPS, επιλέγουν τις κατάλληλες αριθμητικές εντολές ανάλογα με τον τύπο των τελεστέων.

Πράξη	Τελεστέος A	Τελεστέος B	Αποτέλεσμα που δείχνει υπερχείλιση
$A + B$	> 0	> 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	> 0	< 0	< 0
$A - B$	< 0	> 0	≥ 0

ΕΙΚΟΝΑ 3.3 Συνθήκες υπερχείλισης για πρόσθεση και αφαίρεση.

Διασύνδεση υλικού και λογισμικού

Ο σχεδιαστής του υπολογιστή πρέπει να αποφασίσει με ποιο τρόπο θα χειριστεί τις αριθμητικές υπερχειλίσεις. Μολονότι μερικές γλώσσες όπως η C αγνοούν την υπερχείλιση ακεραίων, γλώσσες όπως η Ada και η Fortran απαιτούν να ειδοποιείται το πρόγραμμα. Ο προγραμματιστής ή το περιβάλλον προγραμματισμού πρέπει στη συνέχεια να αποφασίσουν τι θα κάνουν όταν προκύπτουν υπερχειλίσεις.

Ο MIPS εντοπίζει τις υπερχειλίσεις με μια **εξαίρεση** (exception), που σε πολλούς υπολογιστές ονομάζεται και **διακοπή** (interrupt). Μια εξαίρεση ή διακοπή είναι ουσιαστικά μια μη προγραμματισμένη κλήση διαδικασίας. Η διεύθυνση της εντολής που υπερχείλισε αποθηκεύεται σε έναν καταχωρητή, και ο υπολογιστής πραγματοποιεί άλμα σε μια προκαθορισμένη διεύθυνση ώστε να καλέσει την κατάλληλη ρουτίνα για την εξαίρεση. Η διεύθυνση που διακόπηκε αποθηκεύεται ώστε, σε κάποιες περιπτώσεις, να μπορεί το πρόγραμμα να συνεχίσει μετά την εκτέλεση του επανορθωτικού κώδικα. (Η Ενότητα 5.6 καλύπτει τις εξαιρέσεις με περισσότερες λεπτομέρειες· τα Κεφάλαια 7 και 8 περιγράφουν άλλες περιπτώσεις που συμβαίνουν εξαιρέσεις και διακοπές.)

Ο MIPS περιλαμβάνει έναν καταχωρητή που ονομάζεται *μετρητής προγράμματος εξαιρέσεων* (exception program counter — EPC) για να κρατάει τη διεύθυνση της εντολής που προκάλεσε την εξαίρεση. Η εντολή *move from system control* (`mfc0`) χρησιμοποιείται για να αντιγράψει τον EPC σε έναν καταχωρητή γενικού σκοπού ώστε το λογισμικό του MIPS να έχει τη δυνατότητα επιστροφής στη διακοπείσα εντολή μέσω μιας εντολής άλματος μέσω καταχωρητή (jump register).

εξαίρεση (exception) Ονομάζεται επίσης διακοπή (interrupt). Ένα μη προγραμματισμένο συμβάν που διακόπτει την κανονική εκτέλεση του προγράμματος· χρησιμοποιείται για την ανίχνευση υπερχείλισης.

διακοπή (interrupt) Εξαίρεση που προέρχεται έξω από τον επεξεργαστή. (Μερικές αρχιτεκτονικές χρησιμοποιούν τον όρο *διακοπή* για όλες τις εξαιρέσεις.)

Επιπλέον ανάπτυξη: Ο MIPS μπορεί να παγιδεύσει την υπερχείλιση αλλά, σε αντίθεση με πολλούς άλλους υπολογιστές, δεν υπάρχει διακλάδωση υπό συνθήκη για τον έλεγχο της υπερχείλισης. Μια ακολουθία εντολών του MIPS μπορεί να εντοπίσει την υπερχείλιση. Για την προσημασμένη πρόσθεση, η ακολουθία είναι η παρακάτω (δείτε την ενότητα «Σε μεγαλύτερο βάθος» του Κεφαλαίου 2 σχετικά με τις λογικές εντολές, που περιγράφει τον ορισμό των εντολών `xor`):

```
addu $t0, $t1, $t2 # $t0 = άθροισμα, αλλά όχι παγίδευση
xor  $t3, $t1, $t2 # Έλεγχος αν διαφέρουν τα πρόσημα
slt  $t3, $t3, $zero # $t3 = 1 αν διαφέρουν τα πρόσημα
bne  $t3, $zero, No_overflow # πρόσημα $t1,$t2 διάφορα,
                             # άρα όχι υπερχείλιση
xor  $t3, $t0, $t1 # πρόσημα ίσα· πρόσημο αθροίσματος
                   # ταιριάζει επίσης· $t3 αρνητικός αν
                   # πρόσημο αθροίσματος διαφέρει
slt  $t3, $t3, $zero # $t3 = 1 αν πρόσημο αθροίσματος
                     # διαφορετικό
bne  $t3, $zero, Overflow # Και τα τρία πρόσημα διάφορα·
                          # μετάβαση στην ετικέτα Overflow
```

Για την απρόσημη πρόσθεση (`$t0 = $t1 + $t2`), ο έλεγχος είναι

```
addu $t0, $t1, $t2      # $t0 = άθροισμα
nor  $t3, $t1, $zero      # $t3 = NOT $t1
         # (συμπλ. ως προς δύο - 1: 2^32 - $t1 - 1)
sltu $t3, $t3, $t2              # (2^32 - $t1 - 1)< $t2
        # ⇒2^32 - 1 << $t1 + $t2
bne $t3, $zero, Overflow # Αν(2^32 - 1 < $t1 + $t2) μετάβαση
                         # στην ετικέτα Overflow
```

Περίληψη

Το βασικό σημείο αυτής της ενότητας είναι το γεγονός ότι, ανεξάρτητα από την αναπαράσταση, το πεπερασμένο μέγεθος λέξης των υπολογιστών σημαίνει ότι οι αριθμητικές πράξεις (λειτουργίες) μπορούν να δημιουργήσουν αποτελέσματα τα οποία είναι πολύ μεγάλα για να χωρέσουν σε αυτό το σταθερό μέγεθος λέξης. Ο εντοπισμός της υπερχείλισης είναι εύκολος στους απρόσημους αριθμούς, παρά το ότι σχεδόν πάντοτε αγνοείται επειδή τα προγράμματα δε χρειάζεται να εντοπίζουν την υπερχείλιση για αριθμητικές πράξεις διευθύνσεων, την πιο κοινή χρήση των φυσικών αριθμών. Το συμπλήρωμα ως προς δύο παρουσιάζει μια μεγαλύτερη δυσκολία αλλά μερικά συστήματα λογισμικού απαιτούν ανίχνευση της υπερχείλισης, και έτσι σήμερα όλοι οι υπολογιστές πρέπει να την ανιχνεύουν. Η Εικόνα 3.4 παρουσιάζει τις προσθήκες στην αρχιτεκτονική του MIPS που έγιναν σε αυτή την ενότητα.

Αυτοεξέταση Μερικές γλώσσες προγραμματισμού επιτρέπουν αριθμητική ακεραίων συμπληρώματος ως προς δύο σε μεταβλητές που δηλώνονται ως byte και ως half. Ποιες εντολές του MIPS θα χρησιμοποιούνταν;

1. Φόρτωση με τις εντολές `lbu`, `lhu`· αριθμητικές πράξεις με τις εντολές `add`, `sub`, `mult`, `div`· και στη συνέχεια αποθήκευση με χρήση των εντολών `sb`, `sh`.
2. Φόρτωση με τις εντολές `lb`, `lh`· αριθμητικές πράξεις με τις εντολές `add`, `sub`, `mult`, `div`· και στη συνέχεια αποθήκευση με χρήση των `sb`, `sh`.
3. Φόρτωση με τις εντολές `lb`, `lh`· αριθμητικές πράξεις με τις εντολές `add`, `sub`, `mult`, `div`· και χρήση του `and` για εφαρμογή μάσκας στο αποτέλεσμα στα 8 ή 16 bit μετά από κάθε πράξη· στη συνέχεια, αποθήκευση με χρήση των εντολών `sb`, `sh`.

Επιπλέον ανάπτυξη: Είπαμε προηγουμένως ότι αντιγράφουμε τον EPC σε έναν καταχωρητή μέσω της εντολής `mfc0`, και στη συνέχεια επιστρέφουμε στον κώδικα που διακόπηκε μέσω μιας εντολής άλματος μέσω καταχωρητή. Αυτό οδηγεί σε ένα ενδιαφέρον ερώτημα: εφόσον πρέπει πρώτα να μεταφερθεί ο EPC σε έναν καταχωρητή για να χρησιμοποιηθεί στο άλμα μέσω καταχωρητή, με ποιο τρόπο μπορεί το άλμα στον καταχωρητή να επιστρέψει στον κώδικα που διακόπηκε *και* να επαναφέρει τις αρχικές τιμές σε *όλους* τους καταχωρητές; Είτε επαναφέρουμε πρώτα τους παλιούς καταχωρητές, καταστρέφοντας με αυτόν τον τρόπο τη διεύθυνση επιστροφής από τον EPC που τοποθετήσαμε σε έναν καταχωρητή για χρήση με το άλμα μέσω καταχωρητή, είτε επαναφέρουμε όλους τους καταχωρητές εκτός από αυτόν με τη διεύθυνση επιστροφής, ώστε να μπορούμε να εκτελέσουμε το άλμα — που σημαίνει ότι μια εξαίρεση θα οδηγούσε στην αλλαγή αυτού του καταχωρητή οποιαδήποτε στιγμή κατά την εκτέλεση του προγράμματος! Καμία εναλλακτική δεν είναι ικανοποιητική.

Για να απαλλάξουν το υλικό από αυτό το δίλημμα, οι προγραμματιστές του MIPS συμφώνησαν να δεσμεύσουν τους καταχωρητές `$k0` και `$k1` για το λειτουργικό

Συμβολική γλώσσα του MIPS

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Αριθμητικές πράξεις	add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Τρεις τελεστέοι· ανίχνευση υπερχείλισης
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2 - $s3`	Τρεις τελεστέοι· ανίχνευση υπερχείλισης
	add immediate	`addi $s1,$s2,100`	`$s1 = $s2 + 100`	+ σταθερά· ανίχνευση υπερχείλισης
	add unsigned	`addu $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Τρεις τελεστέοι· ανίχνευση υπερχείλισης
	subtract unsigned	`subu $s1,$s2,$s3`	`$s1 = $s2 - $s3`	Τρεις τελεστέοι· ανίχνευση υπερχείλισης
	add immediate unsigned	`addiu $s1,$s2,100`	`$s1 = $s2 + 100`	+ σταθερά· ανίχνευση υπερχείλισης
	move from coprocessor register	`mfc0 $s1,$epc`	`$s1 = $epc`	Χρησιμοποιείται για την αντιγραφή του PC εξαίρεσης και άλλων ειδικών καταχωρητών
Μεταφορά δεδομένων	load word	`lw $s1,100($s2)`	`$s1 = Memory[$s2+100]`	Λέξη από τη μνήμη σε καταχωρητή
	store word	`sw $s1,100($s2)`	`Memory[$s2+100] = $s1`	Λέξη από καταχωρητή στη μνήμη
	load half unsigned	`lhu $s1,100($s2)`	`$s1 = Memory[$s2 + 100]`	Ημιλέξη από τη μνήμη σε καταχωρητή
	store half	`sh $s1,100($s2)`	`Memory[$s2 + 100] = $s1`	Ημιλέξη από καταχωρητή στη μνήμη
	load byte unsigned	`lbu $s1,100($s2)`	`$s1 = Memory[$s2 + 100]`	Byte από τη μνήμη σε καταχωρητή
	store byte	`sb $s1,100($s2)`	`Memory[$s2 + 100] = $s1`	Byte από καταχωρητή στη μνήμη
	load upper immediate	`lui $s1,100`	`$s1 = 100 * 216`	Φορτώνει σταθερά στα υψηλότερα 16 bit
Λογικές πράξεις	and	`and $s1,$s2,$s3`	`$s1 = $s2 & $s3`	Τρεις τελεστέοι καταχωρητές· AND bit προς bit
	or	`or $s1,$s2,$s3`	`$s1 = $s2 | $s3`	Τρεις τελεστέοι καταχωρητές· OR bit προς bit
	nor	`nor $s1,$s2,$s3`	`$s1 = ~ ($s2 | $s3)`	Τρεις τελεστέοι καταχωρητές· NOR bit προς bit
	and immediate	`andi $s1,$s2,100`	`$s1 = $s2 & 100`	AND bit προς bit με σταθερά
	or immediate	`ori $s1,$s2,100`	`$s1 = $s2 | 100`	OR bit προς bit με σταθερά
	shift left logical	`sll $s1,$s2,10`	`$s1 = $s2 << 10`	Αριστερή ολίσθηση με σταθερά
	shift right logical	`srl $s1,$s2,10`	`$s1 = $s2 >> 10`	Δεξιά ολίσθηση με σταθερά
Διακλάδωση υπό συνθήκη	branch on equal	`beq $s1,$s2,25`	αν `($s1 == $s2)` πήγαινε στο PC + 4 + 100	Έλεγχος ισότητας διακλάδωση σχετική ως προς PC
	branch on not equal	`bne $s1,$s2,25`	αν `($s1 != $s2)` πήγαινε στο PC + 4 + 100	Έλεγχος ανισότητας διακλάδωση σχετική ως προς PC
	set on less than	`slt $s1,$s2,$s3`	αν `($s2 < $s3)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από· συμπλήρωμα ως προς δύο
	set less than immediate	`slti $s1,$s2,100`	αν `($s2 < 100)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από σταθερά· συμπλήρωμα ως προς δύο
	set less than unsigned	`sltu $s1,$s2,$s3`	αν `($s2 < $s3)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από· απρόσημοι αριθμοί
	set less than immediate unsigned	`sltiu $s1,$s2,100`	αν `($s2 < 100)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από σταθερά· απρόσημοι αριθμοί
Άλμα χωρίς συνθήκη	jump	`j 2500`	μετάβαση στο 10000	Άλμα στη διεύθυνση προορισμού
	jump register	`jr $ra`	μετάβαση στον `$ra`	Για εντολή switch και για επιστροφή διαδικασίας
	jump and link	`jal 2500`	`$ra` = PC + 4· μετάβαση στο 10000	Για κλήση διαδικασίας

ΕΙΚΟΝΑ 3.4 Η αρχιτεκτονική του MIPS που αποκαλύψαμε έως τώρα. Η μνήμη και οι καταχωρητές της αρχιτεκτονικής του MIPS της Εικόνας 3.1. της σελίδας 187 δεν περιλαμβάνονται για λόγους οικονομίας χώρου. Το χρώμα επισημαίνει τα τμήματα που αποκαλύψαμε από την Εικόνα 3.1. Η γλώσσα μηχανής του MIPS υπάρχει επίσης στην κάρτα συνοπτικής αναφοράς του MIPS.

σύστημα· αυτοί οι καταχωρητές *δεν* επαναφέρονται κατά τις εξαιρέσεις. Όπως οι μεταγλωττιστές του MIPS αποφεύγουν τη χρήση του καταχωρητή `$at` ώστε ο συμβολομεταφραστής να μπορεί να τον χρησιμοποιήσει ως προσωρινό καταχωρητή (δείτε την ενότητα «Διασύνδεση υλικού και λογισμικού» στη σελίδα 114 του Κεφαλαίου 2), αποφεύγουν και οι μεταγλωττιστές τη χρήση των καταχωρητών `$k0` και `$k1` ώστε να τους αφήσουν στη διάθεση του λειτουργικού συστήματος. Οι ρουτίνες χειρισμού εξαιρέσεων τοποθετούν τη διεύθυνση επιστροφής σε έναν από αυτούς τους καταχωρητές και, στη συνέχεια, χρησιμοποιούν το άλμα σε καταχωρητή για να επαναφέρουν τη διεύθυνση της εντολής.

Ο πολλαπλασιασμός είναι εκνευριστικός,
η διαίρεση είναι εξίσου απαίσια·
ο κανόνας των τριών με μπερδεύει,
και η εξάσκηση με τρελαίνει.
Από ανώνυμο χειρόγραφο της εποχής της Ελισάβετ, 1570

3.4 Πολλαπλασιασμός

Τώρα που έχουμε ολοκληρώσει την εξήγηση της πρόσθεσης και της αφαίρεσης, είμαστε έτοιμοι να δομήσουμε την πιο δύσκολη πράξη του πολλαπλασιασμού.

Πρώτα όμως, ας εξετάσουμε τον πολλαπλασιασμό δεκαδικών αριθμών σε κανονική γραφή ώστε να θυμηθούμε τα βήματα και τα ονόματα των τελεστέων. Για λόγους που θα γίνουν σαφείς σύντομα, περιορίζουμε αυτό το δεκαδικό παράδειγμα στη χρήση μόνο των ψηφίων 0 και 1. Ο πολλαπλασιασμός του 1000_{ten} με το 1001_{ten} έχει ως εξής:

Πολλαπλασιαστέος		1000_{ten}
Πολλαπλασιαστής	×	1001_{ten}
		1000
		0000
		0000
		1000
Γινόμενο		1001000_{ten}

Ο πρώτος τελεστέος ονομάζεται *πολλαπλασιαστέος* (multiplicand) και ο δεύτερος *πολλαπλασιαστής* (multiplier). Το τελικό αποτέλεσμα ονομάζεται *γινόμενο* (product). Όπως ίσως θυμάστε, ο αλγόριθμος που μάθατε στο γυμνάσιο είναι να παίρνετε τα ψηφία του πολλαπλασιαστέου ένα-ένα από δεξιά προς τα αριστερά, να πολλαπλασιάζετε τον πολλαπλασιαστέο με ένα ψηφίο του πολλαπλασιαστή, και να ολισθαίνετε τα ενδιάμεσα γινόμενα κατά ένα ψηφίο στα αριστερά των προηγούμενων ενδιάμεσων γινομένων.

Η πρώτη παρατήρηση είναι ότι ο αριθμός των ψηφίων του γινομένου είναι σημαντικά μεγαλύτερος από τον αριθμό είτε στον πολλαπλασιαστέο είτε στον πολλαπλασιαστή. Για την ακρίβεια, αν αγνοήσουμε τα bit προσήμου, το μήκος του πολλαπλασιασμού ενός πολλαπλασιαστέου n bit και ενός πολλαπλασιαστή m bit είναι ένα γινόμενο μήκους $n + m$. Αυτό σημαίνει ότι απαιτούνται $n + m$ bit για την αναπαράσταση όλων των πιθανών γινομένων. Έτσι, όπως η πρόσθεση, ο πολλαπλασιασμός πρέπει να αντιμετωπίσει την υπερχείλιση επειδή συχνά θέλουμε ένα γινόμενο 32 bit ως αποτέλεσμα του πολλαπλασιασμού δύο αριθμών 32 bit.

Στο παράδειγμα αυτό περιορίσαμε τα δεκαδικά ψηφία σε 0 και 1. Με δύο μόνον επιλογές, κάθε βήμα του πολλαπλασιασμού είναι απλό:

1. Απλώς τοποθετούμε ένα αντίγραφο του πολλαπλασιαστέου (1 × πολλαπλασιαστέος) στην κατάλληλη θέση αν το ψηφίο του πολλαπλασιαστή είναι 1, ή
2. Τοποθετούμε 0 (0 × πολλαπλασιαστέος) στην κατάλληλη θέση αν το ψηφίο είναι 0.

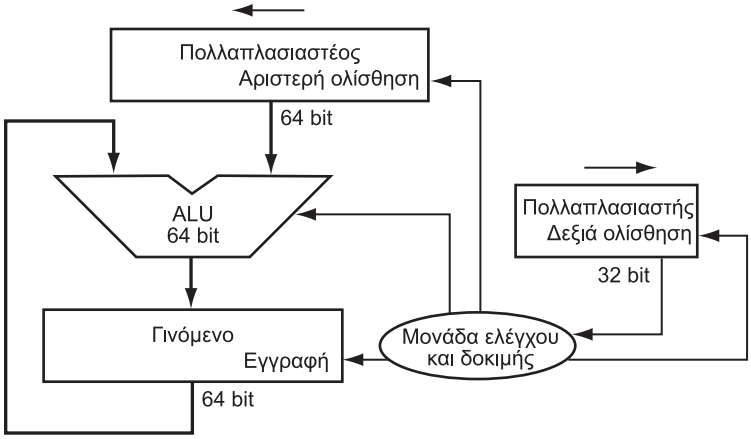
Παρόλο που το δεκαδικό παράδειγμα παραπάνω τυχαίνει να χρησιμοποιεί μόνο 0 και 1, ο πολλαπλασιασμός δυαδικών αριθμών πρέπει πάντοτε να χρησιμοποιεί 0 και 1, και με τον τρόπο αυτόν πάντοτε προσφέρει μόνον αυτές τις δύο επιλογές.

Τώρα που είδαμε τα βασικά του πολλαπλασιασμού, το παραδοσιακό επόμενο βήμα είναι να παράσχουμε σημαντικά βελτιστοποιημένο υλικό πολλαπλασιασμού. Σπάζουμε την παράδοση, με την πίστη ότι θα αποκτήσετε καλύτερη κατανόηση παρακολουθώντας την εξέλιξη του υλικού και των αλγορίθμων του πολλαπλασιασμού μέσα από πολλές γενιές. Προς το παρόν, ας υποθέσουμε ότι πολλαπλασιάζουμε μόνο θετικούς αριθμούς.

Σειριακή έκδοση αλγορίθμου και υλικού του πολλαπλασιασμού

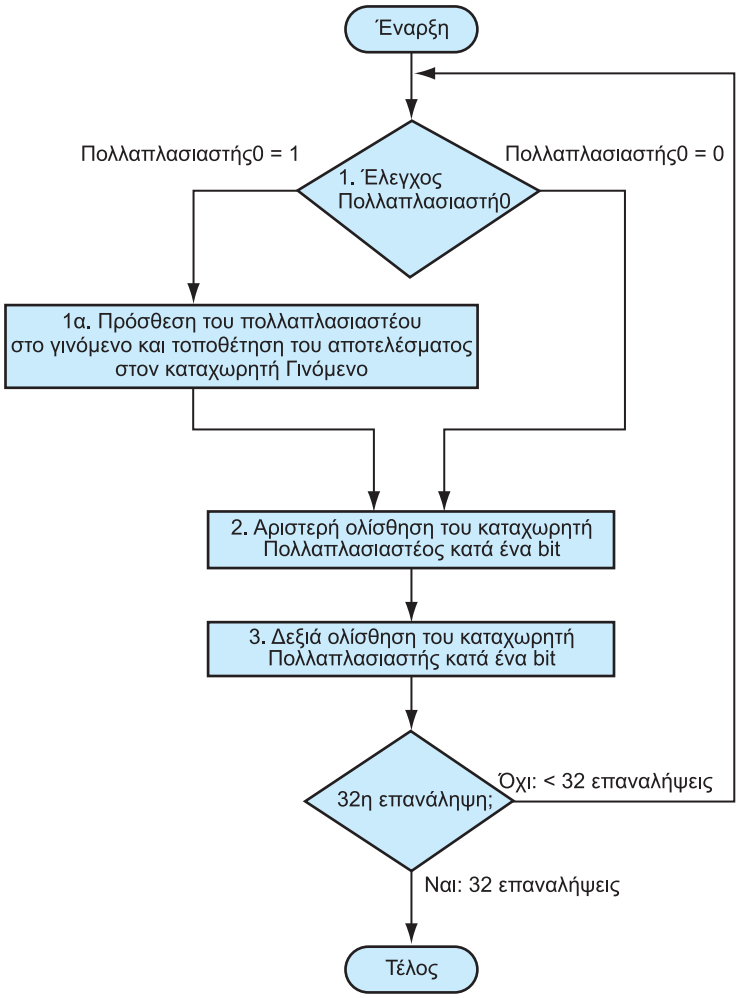
Αυτή η σχεδίαση μιμείται τον αλγόριθμο που μάθαμε στο σχολείο· το υλικό παρουσιάζεται στην Εικόνα 3.5. Έχουμε σχεδιάσει το υλικό με τρόπο που τα δεδομένα να ρέουν από πάνω προς τα κάτω ώστε να μοιάζουν περισσότερο στη μέθοδο με το χαρτί και το μολύβι.

Ας υποθέσουμε ότι ο πολλαπλασιαστής βρίσκεται στον καταχωρητή 32 bit *Πολλαπλασιαστής* και ότι ο καταχωρητής 64 bit *Γινόμενο* παίρνει ως αρχική τιμή 0. Από το παραπάνω παράδειγμα με χαρτί και μολύβι, γίνεται σαφές ότι πρέπει να μετακινούμε τον πολλαπλασιαστέο ένα ψηφίο αριστερά σε κάθε βήμα, αφού είναι πιθανό να προστίθεται στα ενδιάμεσα γινόμενα. Σε 32 βήματα, ένας πολλαπλασιαστέος 32 bit θα μετακινηθεί κατά 32 bit αριστερά. Έτσι χρειαζόμαστε έναν καταχωρητή των 64 bit *Πολλαπλασιαστέος*, με αρχική τιμή τα 32 bit του πολλαπλασιαστέου στο δεξιό μισό και 0 στο αριστερό μισό. Αυτός ο καταχωρητής ολισθαίνει αριστερά κατά 1 bit σε κάθε βήμα, ώστε να ευθυγραμμίσει τον πολλαπλασιαστέο με το άθροισμα που συσσωρεύεται στον καταχωρητή 64 bit *Γινόμενο*.



ΕΙΚΟΝΑ 3.5 Πρώτη έκδοση του υλικού πολλαπλασιασμού. Ο καταχωρητής Πολλαπλασιαστέος, η Αριθμητική και Λογική Μονάδα (Arithmetic and Logic Unit — ALU), και ο καταχωρητής Γινόμενο έχουν εύρος 64 bit, και μόνον ο καταχωρητής Πολλαπλασιαστής περιέχει 32 bit. Ο πολλαπλασιαστέος 32 bit αρχίζει στο δεξιό μισό του καταχωρητή Πολλαπλασιαστέος και ολισθαίνει αριστερά κατά 1 bit σε κάθε βήμα. Ο πολλαπλασιαστής ολισθαίνει στην αντίθετη κατεύθυνση σε κάθε βήμα. Ο αλγόριθμος ξεκινά με το γινόμενο να έχει αρχική τιμή 0. Η μονάδα ελέγχου και δοκιμής αποφασίζει πότε θα ολισθήσει τους καταχωρητές Πολλαπλασιαστέος και Πολλαπλασιαστής και πότε θα γράψει νέες τιμές στον καταχωρητή Γινόμενο.

Η Εικόνα 3.6 παρουσιάζει τα τρία βασικά βήματα που απαιτούνται για κάθε bit. Το λιγότερο σημαντικό bit του πολλαπλασιαστή (Πολλαπλασιαστής0) καθορίζει αν ο πολλαπλασιαστέος προστίθεται στον καταχωρητή *Γινόμενο*. Η αριστερή ολίσθηση στο βήμα 2 προκαλεί τη μετακίνηση των ενδιάμεσων τελεστέων αριστερά, όπως στον πολλαπλασιασμό με το χέρι. Η δεξιά ολίσθηση στο βήμα 3 μάς δίνει το επόμενο bit του πολλαπλασιαστή για να εξεταστεί στην επόμενη επανάληψη. Αυτά τα τρία βήματα επαναλαμβάνονται 32 φορές για τη λήψη του γινομένου. Αν κάθε βήμα απαιτούσε έναν κύκλο ρολογιού, αυτός ο αλγόριθμος θα απαιτούσε περίπου 100 κύκλους ρολογιού για τον πολλαπλασιασμό δύο αριθμών 32 bit. Η σχετική σημασία των αριθμητικών πράξεων όπως του πολλαπλασιασμού διαφέρει ανάλογα με το πρόγραμμα, αλλά η πρόσθεση και η αφαίρεση μπορεί να είναι περίπου από 5 έως 100 φορές πιο δημοφιλείς από τον πολλαπλασιασμό. Άρα, σε πολλές εφαρμογές ο πολλαπλασιασμός μπορεί να αναλώνει πολλούς κύκλους ρολογιού χωρίς να επηρεάζει σημαντικά την



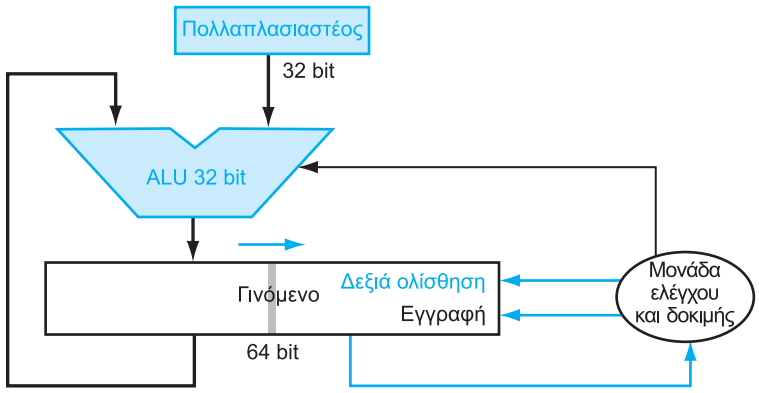
ΕΙΚΟΝΑ 3.6 Ο πρώτος αλγόριθμος πολλαπλασιασμού, με χρήση του υλικού που παρουσιάστηκε στην Εικόνα 3.5. Αν το λιγότερο σημαντικό bit του πολλαπλασιαστή είναι 1, ο πολλαπλασιαστέος προστίθεται στο γινόμενο. Αν όχι, ο αλγόριθμος προχωρεί στο επόμενο βήμα. Στα επόμενα δύο βήματα ο πολλαπλασιαστέος ολισθαίνει αριστερά, και ο πολλαπλασιαστής δεξιά. Αυτά τα τρία βήματα επαναλαμβάνονται 32 φορές.

απόδοση. Παρόλα αυτά, ο νόμος του Amdahl (δείτε το Κεφάλαιο 4 στη σελίδα 284) μας υπενθυμίζει ότι και μια μέτρια συχνότητα εκτέλεσης μιας αργής πράξης μπορεί να περιορίσει την απόδοση.

Ο αλγόριθμος αυτός και το υλικό εύκολα βελτιώνονται ώστε να απαιτούν 1 κύκλο ρολογιού ανά βήμα. Η επιτάχυνση προέρχεται από την εκτέλεση των πράξεων παράλληλα: ο πολλαπλασιαστής και ο πολλαπλασιαστέος ολισθαίνουν ενώ ο πολλαπλασιαστέος προστίθεται στο γινόμενο αν το bit του πολλαπλασιαστή είναι 1. Το υλικό απλώς πρέπει να εξασφαλίσει ότι ελέγχει το σωστό bit του πολλαπλασιαστή και ότι παίρνει την έκδοση του πολλαπλασιαστέου πριν από την ολίσθηση. Το υλικό συνήθως βελτιστοποιείται ακόμη περισσότερο για τον υποδιπλασιασμό του εύρους του αθροιστή και των καταχωρητών παρακολουθώντας τα σημεία στα οποία υπάρχουν μη χρησιμοποιούμενα τμήματα των καταχωρητών και των αθροιστών. Η Εικόνα 3.7 παρουσιάζει το αναθεωρημένο υλικό.

Διασύνδεση υλικού και λογισμικού

Η αντικατάσταση των αριθμητικών πράξεων με ολισθήσεις μπορεί επίσης να συμβεί κατά τον πολλαπλασιασμό με σταθερές. Κάποιοι μεταγλωττιστές αντικαθιστούν πολλαπλασιασμούς με μικρές σταθερές, με μια σειρά από ολισθήσεις και προσθέσεις. Εφόσον ένα bit αριστερά αναπαριστά έναν αριθμό δύο φορές μεγαλύτερο σε βάση 2, η αριστερή ολίσθηση των bit έχει το ίδιο αποτέλεσμα με τον πολλαπλασιασμό με μια δύναμη του 2. Όπως αναφέραμε στο Κεφάλαιο 2, σχεδόν κάθε μεταγλωττιστής θα πραγματοποιήσει τη βελτιστοποίηση μείωσης δύναμης (strength reduction) αντικαθιστώντας έναν πολλαπλασιασμό με δύναμη του 2 με μια αριστερή ολίσθηση.



ΕΙΚΟΝΑ 3.7 Βελτιωμένη έκδοση του υλικού του πολλαπλασιασμού. Συγκρίνετε αυτή με την πρώτη έκδοση στην Εικόνα 3.5. Ο καταχωρητής Πολλαπλασιαστέος, η Αριθμητική και Λογική Μονάδα (ALU), και ο καταχωρητής Πολλαπλασιαστής έχουν όλοι εύρος 32 bit, και μόνον ο καταχωρητής Γινόμενο παραμένει στα 64 bit. Τώρα το γινόμενο ολισθαίνει δεξιά. Ο ξεχωριστός καταχωρητής Πολλαπλασιαστής επίσης εξαφανίζεται. Αντί γι' αυτόν, ο πολλαπλασιαστής τοποθετείται στο δεξιό μισό του καταχωρητή Γινόμενο. Αυτές οι αλλαγές επισημαίνονται με χρώμα.

ΠΑΡΑΔΕΙΓΜΑ

ΑΠΑΝΤΗΣΗ

Ένας αλγόριθμος πολλαπλασιασμού

Χρησιμοποιώντας αριθμούς 4 bit για οικονομία χώρου, πολλαπλασιάστε $2_{ten} \times 3_{ten}$, ή $0010_{two} \times 0011_{two}$.

Η Εικόνα 3.8 παρουσιάζει την τιμή κάθε καταχωρητή σε καθένα από τα βήματα που ονομάζονται με βάση την Εικόνα 3.6, με τελική τιμή την $0000\ 0110_{two}$ ή 6_{ten}. Χρώμα χρησιμοποιείται για να επισημάνει τις τιμές των καταχωρητών που αλλάζουν σε αυτό το βήμα, και το bit στον κύκλο είναι αυτό που εξετάζεται για τον καθορισμό της πράξης στο επόμενο βήμα.

Προσημασμένος πολλαπλασιασμός

Μέχρι τώρα έχουμε χειριστεί θετικούς αριθμούς. Ο ευκολότερος τρόπος για να καταλάβουμε πώς να χειριστούμε προσημασμένους αριθμούς είναι να μετατρέψουμε πρώτα τον πολλαπλασιαστέο και τον πολλαπλασιαστή σε θετικούς αριθμούς, και στη συνέχεια να θυμηθούμε τα αρχικά τους πρόσημα. Οι αλγόριθμοι πρέπει στη συνέχεια να επαναληφθούν 31 φορές, αφήνοντας τα πρόσημα έξω από τον υπολογισμό. Όπως μάθαμε στο σχολείο, πρέπει να αλλάζουμε πρόσημο στο γινόμενο μόνον αν διαφέρουν τα αρχικά πρόσημα.

Αποδεικνύεται ότι ο τελευταίος αλγόριθμος δουλεύει για προσημασμένους αριθμούς, με την προϋπόθεση να θυμόμαστε ότι οι αριθμοί που χειριζόμαστε έχουν άπειρα ψηφία, και τους αναπαριστούμε μόνο με 32 bit. Έτσι, τα βήματα ολίσθησης θα έπρεπε να επεκτείνουν το πρόσημο του γινομένου για προσημασμένους αριθμούς. Όταν ο αλγόριθμος ολοκληρώνεται, το χαμηλότερο τμήμα της λέξης θα έχει το γινόμενο 32 bit.

Επανάληψη	Βήμα	Πολ/στής	Πολ/στέος	Γινόμενο
1	Αρχικές τιμές	0011①	0000 0010	0000 00100
2	1α: 1 ⇒ Γινόμενο = Γινόμενο + Πολλαπλασιαστέος	0011	0000 0010	0000 0010
	2: Αριστερή ολίσθηση Πολλαπλασιαστέου	0011	0000 0100	0000 0010
	3: Δεξιά ολίσθηση Πολλαπλασιαστή	001①	0000 0100	0000 0010
3	1α: 1 ⇒ Γινόμενο = Γινόμενο + Πολλαπλασιαστέος	0001	0000 0100	0000 0110
	2: Αριστερή ολίσθηση Πολλαπλασιαστέου	0001	0000 1000	0000 0110
	3: Δεξιά ολίσθηση Πολλαπλασιαστή	000①	0000 1000	0000 0110
4	1: 0 ⇒ καμία πράξη	0000	0000 1000	0000 0110
	2: Αριστερή ολίσθηση Πολλαπλασιαστέου	0000	0001 0000	0000 0110
	3: Δεξιά ολίσθηση Πολλαπλασιαστή	000①	0001 0000	0000 0110
5	1: 0 ⇒ καμία πράξη	0000	0001 0000	0000 0110
	2: Αριστερή ολίσθηση Πολλαπλασιαστέου	0000	0010 0000	0000 0110
	3: Δεξιά ολίσθηση Πολλαπλασιαστή	0000	0010 0000	0000 0110

ΕΙΚΟΝΑ 3.8 Παράδειγμα πολλαπλασιασμού με τη χρήση του αλγορίθμου της Εικόνας 3.6. Το bit που εξετάζεται για τον καθορισμό του επόμενου βήματος βρίσκεται μέσα σε χρωματιστό κύκλο.

Ταχύτερος πολλαπλασιασμός

Ο νόμος του Moore έχει προσφέρει τόσο περισσότερους πόρους, ώστε οι σχεδιαστές υλικού σήμερα να μπορούν να υλοποιήσουν πολύ γρηγορότερα τον πολλαπλασιασμό στο υλικό. Το αν ο πολλαπλασιαστέος θα προστεθεί ή όχι είναι γνωστό στην αρχή του πολλαπλασιασμού με την παρατήρηση καθενός από τα 32 bit του πολλαπλασιαστή. Γρηγορότεροι πολλαπλασιασμοί είναι εφικτοί με την παροχή ενός αθροιστή 32 bit για κάθε bit του πολλαπλασιαστή: η μια είσοδος είναι ο πολλαπλασιαστέος αφού έχει υποστεί AND με ένα bit του πολλαπλασιαστή, και η άλλη είναι η έξοδος ενός προηγούμενου αθροιστή. Η Εικόνα 3.9 παρουσιάζει τον τρόπο με τον οποίο πρέπει να συνδεθούν.

Για ποιο λόγο είναι αυτό το υλικό τόσο γρηγορότερο; Ο σειριακός πολλαπλασιαστής πληρώνει την επιβάρυνση ενός κύκλου ρολογιού για κάθε bit του γινομένου. Αυτός ο πολλαπλασιαστής με πίνακα αθροιστών (adders array) την αποφεύγει. Ένας δεύτερος λόγος είναι ότι αυτή η μεγάλη συλλογή αθροιστών προσφέρεται για πολλές βελτιστοποιήσεις ώστε να προκύψουν περαιτέρω βελτιώσεις. Ένα παράδειγμα είναι η χρήση *αθροιστών διατήρησης κρατουμένου* (carry save adders) για την άθροιση μιας τόσο μεγάλης στήλης αριθμών· δείτε τις Ασκήσεις 3.24 και 3.49. Ένας τρίτος λόγος είναι η ευκολία εισαγωγής διοχέτευσης (pipelining) σε μια τέτοια σχεδίαση ώστε να μπορεί να υποστηρίξει πολλούς πολλαπλασιασμούς ταυτόχρονα (δείτε το Κεφάλαιο 6).

Πολλαπλασιασμός στον MIPS

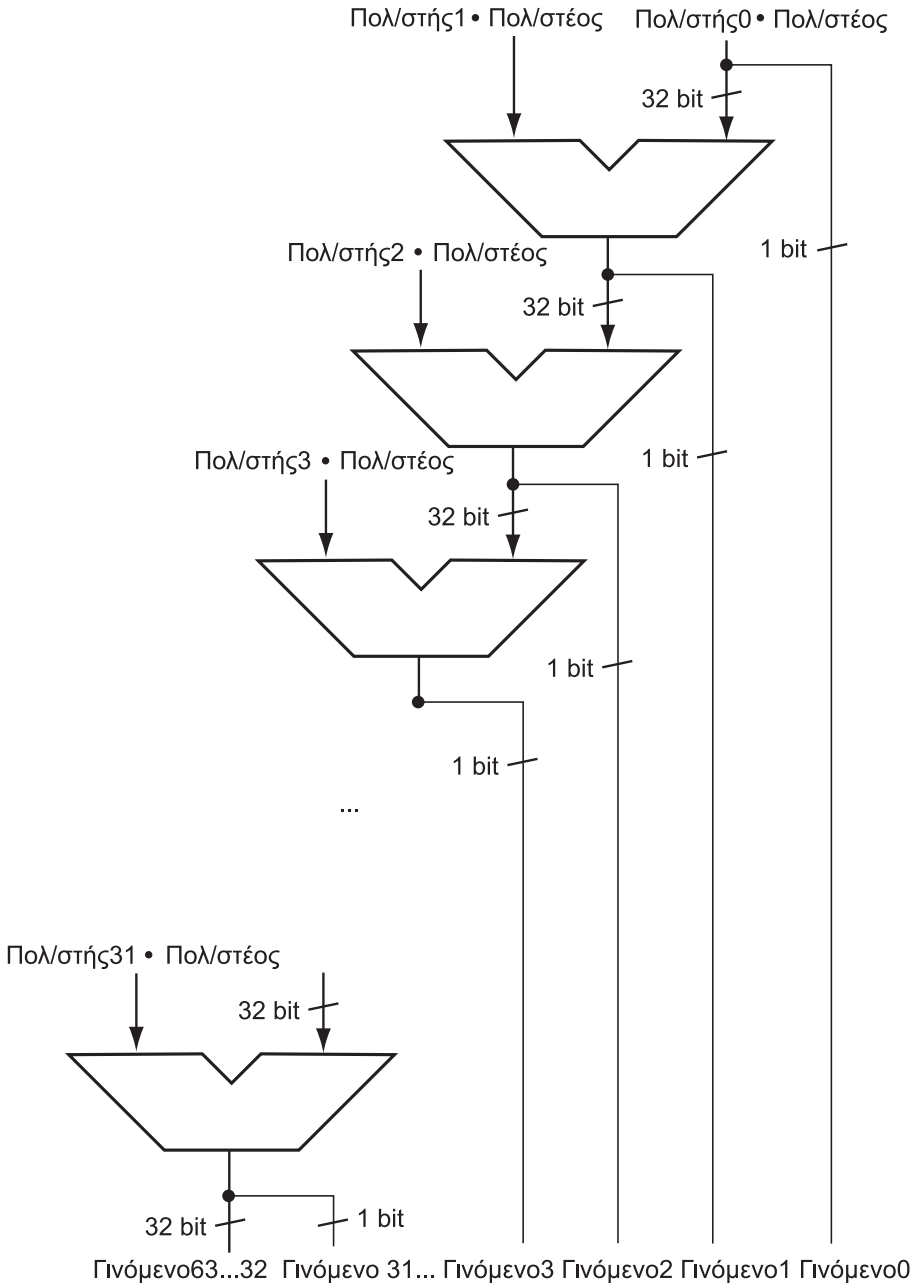
Ο MIPS παρέχει ένα ξεχωριστό ζεύγος καταχωρητών 32 bit που θα περιέχουν το γινόμενο 64 bit, οι οποίοι ονομάζονται *Hi* (υψηλός) και *Lo* (χαμηλός). Για την παραγωγή ενός κατάλληλα προσημασμένου ή απρόσημου γινομένου, ο MIPS έχει δύο εντολές: τη multiply (`mult`) και τη multiply unsigned (`multu`). Για την προσκόμιση (fetch) του ακέραιου γινομένου των 32 bit, ο προγραμματιστής χρησιμοποιεί την εντολή *move from lo* (`mflo`). Ο συμβολομεταφραστής του MIPS παράγει μια ψευδοεντολή για τον πολλαπλασιασμό, η οποία καθορίζει τρεις καταχωρητές γενικού σκοπού, παράγοντας εντολές `mflo` και `mfhi` για την τοποθέτηση του γινομένου στους καταχωρητές.

Περίληψη

Ο πολλαπλασιασμός υλοποιείται με απλό υλικό ολίσθησης και πρόσθεσης, που έχει προκύψει από τη μέθοδο με χαρτί και μολύβι που μάθαμε στο σχολείο. Οι μεταγλωττιστές χρησιμοποιούν ακόμη και εντολές ολίσθησης για πολλαπλασιασμούς με δυνάμεις του δύο.

Διασύνδεση υλικού και λογισμικού

Και οι δύο εντολές πολλαπλασιασμού του MIPS αγνοούν την υπερχείλιση, και επομένως είναι καθήκον του λογισμικού να ελέγξει αν το γινόμενο είναι πολύ μεγάλο για να χωρέσει σε 32 bit. Δεν υπάρχει υπερχείλιση αν ο Hi είναι 0 για την εντολή `multu` ή επανάληψη του προσήμου του Lo για την εντολή `mult`. Η εντολή *move from hi* (`mfhi`) μπορεί να χρησιμοποιηθεί για τη μεταφορά του Hi σε έναν καταχωρητή γενικού σκοπού με στόχο τον έλεγχο της υπερχείλισης.



ΕΙΚΟΝΑ 3.9 Υλικό γρήγορου πολλαπλασιασμού. Αντί για τη χρήση ενός μόνον αθροιστή 32 bit 32 φορές, αυτό το υλικό «ξετυλίγει το βρόχο» ώστε να χρησιμοποιήσει 31 αθροιστές. Κάθε αθροιστής παράγει ένα άθροισμα 32 bit και ένα κρατούμενο εξόδου. Το λιγότερο σημαντικό bit είναι ένα bit του γινομένου, και το κρατούμενο και τα υψηλότερα 31 bit του αθροίσματος μεταβιβάζονται στον επόμενο αθροιστή.

3.5 Διαίρεση

Divide et impera.

Λατινικά για τη φράση «Διαίρει και βασίλευε» αρχαίου πολιτικού αξιωματούχου, που αναφέρεται από τον Μακιαβέλλι, 1532

Η συμπληρωματική πράξη του πολλαπλασιασμού είναι η διαίρεση, μια πράξη ακόμα λιγότερο συχνή και ακόμη πιο παράξενη. Εμπεριέχει επίσης την πιθανότητα πραγματοποίησης μιας μαθηματικά ανεπίτρεπτης πράξης: της διαίρεσης με το 0.

Ας ξεκινήσουμε με ένα παράδειγμα διαίρεσης μεγάλων ακεραίων (long) χρησιμοποιώντας αριθμούς με βάση το δέκα για να θυμηθούμε τα ονόματα των τελεστέων και τον αλγόριθμο διαίρεσης του σχολείου. Για λόγους παρόμοιους με αυτούς της προηγούμενης ενότητας, περιορίζουμε τα δεκαδικά ψηφία μόνο σε 0 και 1. Το παράδειγμα είναι η διαίρεση του $1.001.010_{ten}$ με το 1000_{ten}:

```
                  1001ten          Πηλίκο
Διαιρέτης 1000ten ⌊1001010ten      Διαιρετέος
                 −1000
                    10
                   101
                  1010
                 -1000
                    10ten Υπόλοιπο
```

Οι δύο τελεστέοι — **διαιρετέος** (dividend) και **διαιρέτης** (divisor) και το αποτέλεσμα (**πηλίκο** — quotient) της διαίρεσης συνοδεύονται από ένα δεύτερο αποτέλεσμα που ονομάζεται **υπόλοιπο** (remainder). Ένας άλλος τρόπος έκφρασης της σχέσης μεταξύ των συστατικών αυτών μερών είναι ο εξής:

$$\text{Διαιρετέος} = \text{Πηλίκο} \times \text{Διαιρέτης} + \text{Υπόλοιπο}$$

διαιρετέος (dividend) Ο αριθμός που διαιρείται.

διαιρέτης (divisor) Ο αριθμός με τον οποίο διαιρείται ο διαιρετέος.

πηλίκο (quotient) Το κύριο αποτέλεσμα μιας διαίρεσης· ένας αριθμός ο οποίος, όταν πολλαπλασιάζεται με το διαιρέτη και προστίθεται στο υπόλοιπο, δίνει το διαιρετέο.

υπόλοιπο (remainder) Το δευτερεύον αποτέλεσμα της διαίρεσης· ένας αριθμός ο οποίος, όταν προστίθεται στο γινόμενο του πηλίκου με το διαιρέτη, δίνει το διαιρετέο.

όπου το υπόλοιπο είναι μικρότερο από το διαιρέτη. Σε μερικές σπάνιες περιπτώσεις, τα προγράμματα χρησιμοποιούν την εντολή της διαίρεσης απλώς για να πάρουν το υπόλοιπο, αγνοώντας το πηλίκο.

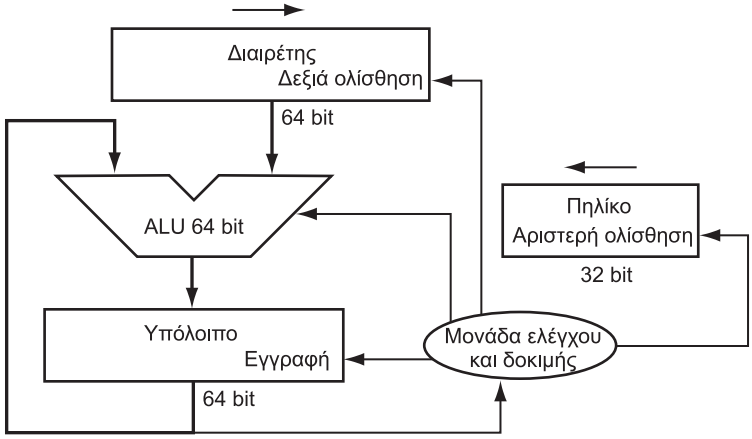
Ο αλγόριθμος του σχολείου προσπαθεί να βρει πόσο μεγάλος μπορεί να είναι ένας αριθμός που θα αφαιρεθεί, δημιουργώντας ένα ψηφίο του υπολοίπου σε κάθε προσπάθεια. Το προσεκτικά επιλεγμένο δεκαδικό παράδειγμά μας χρησιμοποιεί μόνο τους αριθμούς 0 και 1, και έτσι είναι εύκολο να δούμε πόσες φορές «χωράει» ο διαιρέτης στο διαιρετέο: είναι είτε 0 φορές είτε 1 φορά. Οι δυαδικοί αριθμοί περιέχουν μόνο 0 και 1, και έτσι η δυαδική διαίρεση είναι απλούστερη και περιορίζεται σε αυτές τις δύο επιλογές.

Ας υποθέσουμε ότι τόσο ο διαιρετέος όσο και ο διαιρέτης είναι θετικοί και, επομένως, το πηλίκο και το υπόλοιπο είναι μη αρνητικοί. Οι τελεστέοι της διαίρεσης και τα δύο αποτελέσματα είναι τιμές 32 bit, και θα αγνοήσουμε προς το παρόν το πρόσημο.

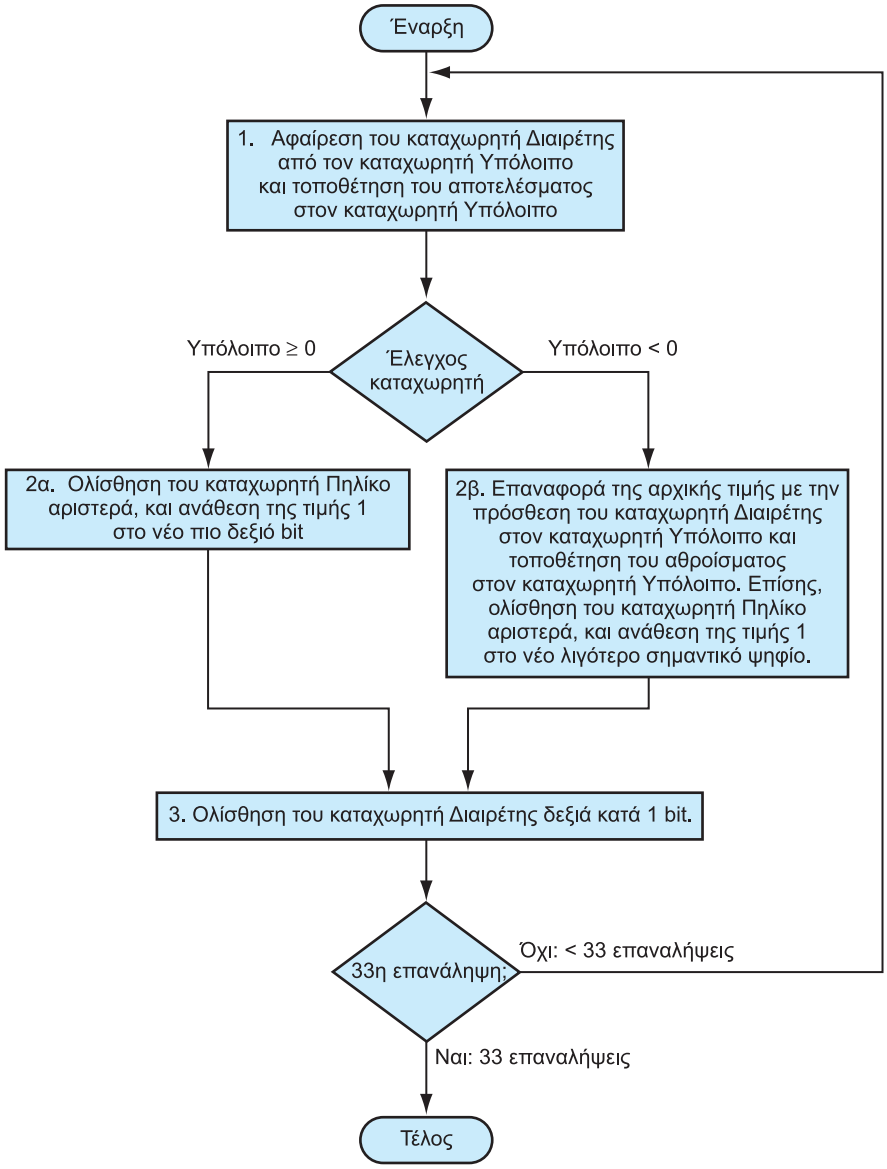
Αλγόριθμος και υλικό διαίρεσης

Η Εικόνα 3.10 παρουσιάζει υλικό που μιμείται τον αλγόριθμο του σχολείου. Ξεκινούμε με τον καταχωρητή 32 bit *Πηλίκο* με αρχική τιμή το 0. Κάθε επανάληψη του αλγορίθμου πρέπει να μετακινήσει το διαιρέτη δεξιά κατά ένα ψηφίο, και έτσι ξεκινούμε με το διαιρέτη τοποθετημένο στο αριστερό μισό του καταχωρητή 64 bit *Διαιρέτης* και τον ολισθαίνουμε δεξιά κατά 1 bit σε κάθε βήμα, ώστε να είναι ευθυγραμμισμένος με το διαιρετέο. Ο καταχωρητής *Υπόλοιπο* παίρνει ως αρχική τιμή του το διαιρετέο.

Η Εικόνα 3.11 παρουσιάζει τρία βήματα του πρώτου αλγορίθμου διαίρεσης. Σε αντίθεση με τον άνθρωπο, ο υπολογιστής δεν είναι αρκετά έξυπνος να γνωρίζει από την αρχή αν ο διαιρέτης είναι μικρότερος από το διαιρετέο. Πρέπει πρώτα να αφαιρέσει το διαιρέτη στο βήμα 1· θυμηθείτε ότι αυτός είναι ο τρόπος με τον οποίο πραγματοποιήσαμε τη σύγκριση στην εντολή set on less than. Αν το αποτέλεσμα είναι θετικό, ο διαιρέτης ήταν μικρότερος ή ίσος με το διαιρετέο, και επομένως παράγουμε ένα 1 στο πηλίκο (βήμα 2α). Αν το αποτέλεσμα είναι αρνητικό, το επόμενο βήμα είναι να επαναφέρουμε την αρχική τιμή προσθέτοντας πάλι το διαιρέτη στο υπόλοιπο και παράγοντας ένα 0 στο πηλίκο (βήμα 2β). Ο διαιρέτης ολισθαίνει δεξιά και ο αλγόριθμος επαναλαμβάνεται. Μετά την ολοκλήρωση των επαναλήψεων, το υπόλοιπο και το πηλίκο θα βρίσκονται στους ομώνυμους καταχωρητές.



ΕΙΚΟΝΑ 3.10 Πρώτη έκδοση του υλικού της διαίρεσης. Ο καταχωρητής Διαιρέτης, η Αριθμητική και Λογική Μονάδα (ALU), και ο καταχωρητής Υπόλοιπο έχουν εύρος 64 bit, και μόνον ο καταχωρητής Πηλίκο 32 bit. Ο διαιρέτης 32 bit ξεκινάει στο αριστερό μισό του καταχωρητή Διαιρέτης και ολισθαίνει δεξιά 1 bit σε κάθε επανάληψη. Το υπόλοιπο παίρνει ως αρχική τιμή του το διαιρετέο. Η μονάδα ελέγχου και δοκιμής αποφασίζει πότε να ολισθήσει τους καταχωρητές Διαιρέτης και Πηλίκο και πότε να γράψει νέα τιμή στον καταχωρητή Υπόλοιπο.



ΕΙΚΟΝΑ 3.11 Ένας αλγόριθμος διαίρεσης που χρησιμοποιεί το υλικό της Εικόνας 3.10. Αν το υπόλοιπο είναι θετικό, ο διαιρέτης χώρεσε στο διαιρετέο, και το βήμα 2α παράγει ένα 1 στο πηλίκο. Ένα αρνητικό υπόλοιπο μετά το βήμα 1 σημαίνει ότι ο διαιρέτης δε χώρεσε στο διαιρετέο, και το βήμα 2β παράγει ένα 0 στο πηλίκο και προσθέτει το διαιρέτη στο υπόλοιπο, αντιστρέφοντας με τον τρόπο αυτόν την αφαίρεση του βήματος 1. Η τελική ολίσθηση, στο βήμα 3, ευθυγραμμίζει το διαιρέτη κατάλληλα σε σχέση με το διαιρετέο, για την επόμενη επανάληψη. Αυτά τα βήματα επαναλαμβάνονται 33 φορές.

Ένας αλγόριθμος διαίρεσης

ΠΑΡΑΔΕΙΓΜΑ

Χρησιμοποιώντας μια έκδοση 4 bit του αλγορίθμου για οικονομία χώρου, ας προσπαθήσουμε να διαιρέσουμε το 7_{ten} με το 2_{ten}, ή το $0000\ 0111_{two}$ με το 0010_{two}.

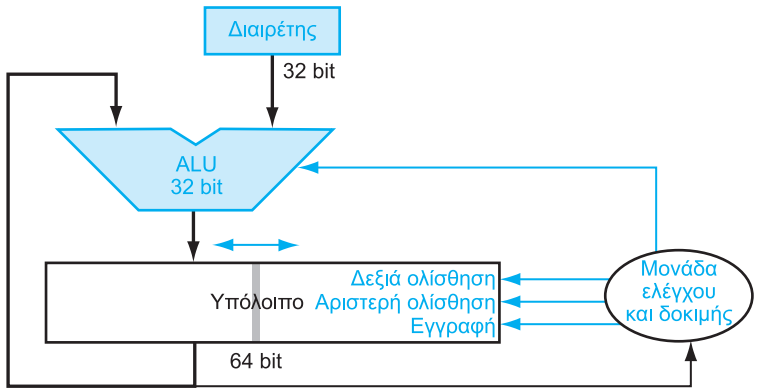
ΑΠΑΝΤΗΣΗ

Η Εικόνα 3.12 παρουσιάζει την τιμή κάθε καταχωρητή σε καθένα από τα βήματα, με το πηλίκο να είναι 3_{ten} και το υπόλοιπο 1_{ten}. Παρατηρήστε ότι ο έλεγχος στο βήμα 2, αν το υπόλοιπο είναι θετικό ή αρνητικό, απλώς διαπιστώνει αν το bit προσήμου του καταχωρητή *Υπόλοιπο* είναι 0 ή 1. Η απρόσμενη απαίτηση αυτού του αλγορίθμου είναι ότι χρειάζονται $n + 1$ βήματα για να δώσει το κατάλληλο πηλίκο και υπόλοιπο.

Αυτός ο αλγόριθμος και το υλικό μπορούν να βελτιωθούν ως προς την ταχύτητα και το κόστος. Η επιτάχυνση προέρχεται από την ολίσθηση των τελεστέων και του πηλίκου ταυτόχρονα με την αφαίρεση. Αυτή η βελτίωση μειώνει στο μισό το εύρος του αθροιστή και των καταχωρητών, παρακολουθώντας τα σημεία όπου υπάρχουν τμήματα των καταχωρητών και των αθροιστών που δε χρησιμοποιούνται. Η Εικόνα 3.13 παρουσιάζει το αναθεωρημένο υλικό.

Επανάληψη	Βήμα	Πηλίκο	Διαιρέτης	Υπόλοιπο
0	Αρχικές τιμές	0000	0010 000 0	0000 0111
1	1 Υπόλοιπο = Υπόλοιπο – Διαιρέτης	0000	0010 0000	①110 0111
	2β: Υπόλοιπο< 0 ⇒ +Διαιρέτης, Αριστερή ολίσθηση Πηλίκου, Πηλίκο0 = 0	0000	0010 0000	0000 0111
	3: Δεξιά ολίσθηση του καταχωρητή Διαιρέτης	0000	0001 0000	0000 0111
2	1 Υπόλοιπο = Υπόλοιπο – Διαιρέτης	0000	0001 0000	①111 0111
	2β: Υπόλοιπο< 0 ⇒ +Διαιρέτης, Αριστερή ολίσθηση Πηλίκου, Πηλίκο0 = 0	0000	0001 0000	0000 0111
	3: Δεξιά ολίσθηση του καταχωρητή Διαιρέτης	0000	0000 1000	0000 0111
3	1 Υπόλοιπο = Υπόλοιπο – Διαιρέτης	0000	0000 1000	①111 1111
	2β: Υπόλοιπο ≥ 0 ⇒ Αριστερή ολίσθηση Πηλίκου, Πηλίκο0 = 1	0000	0000 1000	0000 0111
	3: Δεξιά ολίσθηση του καταχωρητή Διαιρέτης	0000	0000 0100	0000 0111
4	1 Υπόλοιπο = Υπόλοιπο – Διαιρέτης	0000	0000 0100	⓪000 0011
	2α: Υπόλοιπο ≥ 0 ⇒ Αριστερή ολίσθηση Πηλίκου, Πηλίκο0 = 1	0001	0000 0100	0000 0011
	3: Δεξιά ολίσθηση του καταχωρητή Διαιρέτης	0001	0000 0010	0000 0011
5	1 Υπόλοιπο = Υπόλοιπο – Διαιρέτης	0001	0000 0010	⓪000 0001
	2α: Υπόλοιπο ≥ 0 ⇒ Αριστερή ολίσθηση Πηλίκου, Πηλίκο0 = 1	0011	0000 0010	0000 0001
	3: Δεξιά ολίσθηση του καταχωρητή Διαιρέτης	0011	0000 0001	0000 0001

ΕΙΚΟΝΑ 3.12 Παράδειγμα διαίρεσης με τη χρήση του αλγορίθμου της Εικόνας 3.11. Το bit που εξετάζεται για τον καθορισμό του επόμενου βήματος βρίσκεται μέσα σε χρωματιστό κύκλο.



ΕΙΚΟΝΑ 3.13 Μια βελτιωμένη έκδοση του υλικού διαίρεσης. Ο καταχωρητής Διαιρέτης, η Αριθμητική και Λογική Μονάδα, και ο καταχωρητής Πηλίκο έχουν εύρος 32 bit, και μόνον ο καταχωρητής Υπόλοιπο παραμένει στα 64 bit. Σε σύγκριση με την Εικόνα 3.10, η Αριθμητική και Λογική Μονάδα και ο καταχωρητής Διαιρέτης μειώνονται στο μισό και το υπόλοιπο ολισθαίνει αριστερά. Αυτή η έκδοση επίσης συνδυάζει τον καταχωρητή Πηλίκο με το δεξιό μισό του καταχωρητή Υπόλοιπο.

Προσημασμένη διαίρεση

Έως τώρα αγνοήσαμε τους προσημασμένους αριθμούς στη διαίρεση. Η απλούστερη λύση είναι να θυμόμαστε τα πρόσημα του διαιρέτη και του διαιρετέου και στη συνέχεια να κάνουμε αρνητικό το πρόσημο του πηλίκου αν τα πρόσημά τους διαφέρουν.

Επιπλέον ανάλυση: Η επιπλοκή της προσημασμένης διαίρεσης είναι ότι πρέπει να ορίσουμε και το πρόσημο του υπολοίπου. Μην ξεχνάτε ότι η παρακάτω εξίσωση πρέπει να ισχύει πάντα:

Διαιρετέος = Πηλίκο × Διαιρέτης + Υπόλοιπο

Για την κατανόηση του τρόπου ορισμού τού προσήμου τού υπολοίπου, ας δούμε το παράδειγμα της διαίρεσης όλων των συνδυασμών του $\pm 7_{ten}$ με το $\pm 2_{ten}$. Η πρώτη περίπτωση είναι εύκολη:

+7 ÷ +2: Πηλίκο = +3, Υπόλοιπο = +1

Έλεγχος των αποτελεσμάτων:

7 = 3 × 2 + (+1) = 6 + 1

Αν αλλάξουμε το πρόσημο του διαιρετέου, το πηλίκο πρέπει επίσης να αλλάξει:

–7 ÷ +2: Πηλίκο = –3

Αναδιατυπώνουμε τη βασική εξίσωση για τον υπολογισμό του υπολοίπου:

Υπόλοιπο = (Διαιρετέος – Πηλίκο × Διαιρέτης) = –7 – (–3 × +2) = –7 – (–6) = –1

Έτσι,

–7 ÷ +2: Πηλίκο = –3, Υπόλοιπο = –1

Ελέγχουμε και πάλι τα αποτελέσματα:

–7 = –3 × 2 + (–1) = –6 –1

Ο λόγος για τον οποίο το αποτέλεσμα δεν είναι Πηλίκο –4 και Υπόλοιπο +1, τα οποία θα ταίριαζαν επίσης στην εξίσωση, είναι ότι η απόλυτη τιμή του πηλίκου θα άλλαζε τότε ανάλογα με το πρόσημο του διαιρετέου και του διαιρέτη! Είναι σαφές ότι αν ίσχυε

$$-(x \div y) \# (-x) \div y$$

ο προγραμματισμός θα ήταν ακόμη μεγαλύτερη πρόκληση. Αυτή η ανώμαλη συμπεριφορά μπορεί να αποφευχθεί αν ακολουθήσουμε τον κανόνα ότι ο διαιρετέος και το υπόλοιπο πρέπει να έχουν ίδια πρόσημα, ανεξάρτητα από τα πρόσημα του διαιρέτη και του πηλίκου.

Υπολογίζουμε τους άλλους συνδυασμούς ακολουθώντας τον ίδιο κανόνα:

+7 ÷ –2: Πηλίκο = –3, Υπόλοιπο = +1

–7 ÷ –2: Πηλίκο = +3, Υπόλοιπο = –1

Έτσι, ο σωστά προσημασμένος αλγόριθμος διαίρεσης αλλάζει το πρόσημο του πηλίκου αν τα πρόσημα των τελεστέων είναι αντίθετα, και κάνει το πρόσημο του μη μηδενικού υπολοίπου να συμφωνεί με το διαιρετέο.

Ταχύτερη διαίρεση

Χρησιμοποιήσαμε 32 αθροιστές για την επιτάχυνση του πολλαπλασιασμού, αλλά δεν μπορούμε να εφαρμόσουμε το ίδιο τέχνασμα για τη διαίρεση. Ο λόγος είναι ότι πρέπει να γνωρίζουμε το πρόσημο της διαφοράς πριν εκτελέσουμε το επόμενο βήμα του αλγορίθμου ενώ, αντίθετα, στον πολλαπλασιασμό μπορούσαμε να υπολογίσουμε τα 32 μερικά γινόμενα απευθείας.

Υπάρχουν τεχνικές για την παραγωγή περισσότερων από ένα bit του πηλίκου ανά bit. Η τεχνική της *διαίρεσης SRT* προσπαθεί να μαντέψει αρκετά bit του πηλίκου ανά βήμα, χρησιμοποιώντας έναν πίνακα αναζήτησης (lookup table) βασισμένο στα υψηλότερα bit του διαιρετέου και του υπολοίπου. Η διόρθωση των λανθασμένων εκτιμήσεων γίνεται σε επόμενα βήματα. Μια τυπική τιμή σήμερα είναι 4 bit. Το κλειδί βρίσκεται στην εκτίμηση της τιμής που θα αφαιρεθεί. Στη δυαδική διαίρεση υπάρχει μόνο μία επιλογή. Αυτοί οι αλγόριθμοι χρησιμοποιούν 6 bit από το υπόλοιπο και 4 bit από το διαιρέτη για την αριθμοδεικτοδότηση ενός πίνακα που καθορίζει την εκτίμηση για κάθε βήμα.

Η ακρίβεια αυτής της γρήγορης μεθόδου εξαρτάται από την ύπαρξη κατάλληλων τιμών στον πίνακα αναζήτησης. Η πλάνη στη σελίδα 240 της Ενότητας 3.8 δείχνει τι μπορεί να συμβεί αν ο πίνακας είναι εσφαλμένος.

Η διαίρεση στον MIPS

Μπορεί να έχετε ήδη παρατηρήσει ότι μπορεί να χρησιμοποιηθεί το ίδιο σειριακό υλικό τόσο για πολλαπλασιασμό όσο και για διαίρεση, στις Εικόνες 3.7 και 3.13. Η μόνη απαίτηση είναι ένας καταχωρητής 64 bit που μπορεί να ολισθήσει αριστερά ή δεξιά και μια Αριθμητική και Λογική Μονάδα (ALU) 32 bit που προσθέτει ή αφαιρεί. Επομένως, ο MIPS χρησιμοποιεί τους καταχωρητές 32 bit Hi και Lo τόσο για τον πολλαπλασιασμό όσο και για τη διαίρεση. Όπως θα μπορούσαμε να αναμένουμε από τον παραπάνω αλγόριθμο, μετά την ολοκλήρωση της εντολής της διαίρεσης ο καταχωρητής Hi περιέχει το υπόλοιπο και ο Lo περιέχει το πηλίκο.

Για το χειρισμό τόσο προσημασμένων όσο και απρόσημων ακεραίων, ο MIPS έχει δύο εντολές: τις *divide* (`div`) και *divide unsigned* (`divu`). Ο συμβολομεταφραστής του MIPS επιτρέπει στις εντολές διαίρεσης να καθορίζουν τρεις καταχωρητές, παράγοντας τις εντολές `mflo` ή `mfhi` για την τοποθέτηση του κατάλληλου αποτελέσματος σε έναν καταχωρητή γενικού σκοπού.

Περίληψη

Το κοινό υλικό που υποστηρίζει τον πολλαπλασιασμό και τη διαίρεση επιτρέπει στον MIPS να διαθέτει ένα ζεύγος καταχωρητών 32 bit που χρησιμοποιούνται τόσο για τον πολλαπλασιασμό όσο και για τη διαίρεση. Η Εικόνα 3.14 συνοψίζει τις προσθήκες στην αρχιτεκτονική του MIPS για τις δύο τελευταίες ενότητες.

Διασύνδεση υλικού και λογισμικού

Οι εντολές διαίρεσης του MIPS αγνοούν την υπερχείλιση και, έτσι, το λογισμικό πρέπει να αποφασίσει αν το πηλίκο είναι πολύ μεγάλο. Εκτός από την υπερχείλιση, η διαίρεση μπορεί επίσης να οδηγήσει σε ένα μη επιτρεπτό υπολογισμό: τη διαίρεση με το 0. Μερικοί υπολογιστές ξεχωρίζουν αυτά τα δύο ανώμαλα συμβάντα. Το λογισμικό του MIPS πρέπει να ελέγχει το διαιρέτη για να εντοπίζει και τη διαίρεση με το 0 και την υπερχείλιση.

Επιπλέον ανάπτυξη: Ένας ακόμη ταχύτερος αλγόριθμος δεν προσθέτει αμέσως πάλι το διαιρέτη αν το υπόλοιπο είναι αρνητικό. Απλώς προσθέτει το διαιρετέο στο ολισθημένο υπόλοιπο στο επόμενο βήμα αφού $(r + d) \times 2 - d = r \times 2 + d \times 2 - d = r \times 2 + d$. Αυτός ο *αλγόριθμος διαίρεσης χωρίς επαναφορά* (nonrestoring division algorithm), ο οποίος απαιτεί έναν κύκλο ρολογιού ανά βήμα, διερευνάται περισσότερο στην Άσκηση 3.29· ο αλγόριθμος που παρουσιάστηκε εδώ ονομάζεται *διαίρεση με επαναφορά* (restoring division). ×

3.6 Κινητή υποδιαστολή

Η ταχύτητα δε σε οδηγεί πουθενά αν έχεις λάθος κατεύθυνση.
Αμερικανική παροιμία

Πέρα από τους προσημασμένους και τους απρόσημους ακεραίους, οι γλώσσες προγραμματισμού υποστηρίζουν αριθμούς με κλασματικά μέρη, που ονομάζονται *πραγματικοί* (reals) στα μαθηματικά. Ακολουθούν μερικά παραδείγματα πραγματικών αριθμών:

$3{,}14159265..._{ten}$ (π)

$2{,}71828..._{ten}$ (e)

$0{,}000000001_{ten}$ ή $1{,}0_{ten} \times 10^{-9}$ (δευτερόλεπτα σε ένα νανοδευτερόλεπτο — nanosecond)

$3.155.760.000_{ten}$ ή $3{,}15576_{ten} \times 10^{9}$ (δευτερόλεπτα σε έναν τυπικό αιώνα)

Παρατηρήστε ότι, στην τελευταία περίπτωση, ο αριθμός δεν αναπαριστά ένα μικρό κλάσμα αλλά είναι μεγαλύτερος από ό,τι θα μπορούσαμε να αναπαραστήσουμε με έναν προσημασμένο ακέραιο 32 bit. Η εναλλακτική σημειογραφία για τους δύο τελευταίους αριθμούς ονομάζεται **επιστημονική σημειογραφία** (scientific notation), και έχει ένα μόνο ψηφίο στα αριστερά της υποδιαστολής. Ένας αριθμός στην επιστημονική σημειογραφία, ο οποίος δεν έχει αρχικά 0, ονομάζεται **κανονικοποιημένος** (normalized) αριθμός, και αυτός είναι ο συνηθισμένος τρόπος γραφής του. Για παράδειγμα, ο αριθμός $1{,}0_{ten} \times 10^{-9}$ είναι σε κανονικοποιημένη επιστημονική σημειογραφία, αλλά οι $0{,}1_{ten} \times 10^{-8}$ και $10{,}0_{ten} \times 10^{-10}$ δεν είναι.

επιστημονική σημειογραφία (scientific notation) Μια σημειογραφία που δίνει αριθμούς με ένα μόνο ψηφίο στα αριστερά της υποδιαστολής.

κανονικοποιημένος (normalized) Ένας αριθμός σε σημειογραφία κινητής υποδιαστολής χωρίς αρχικά 0.

Συμβολική γλώσσα του MIPS

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Αριθμητικές πράξεις	add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Τρεις τελεστέοι· ανίχνευση υπερχείλισης
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2 - $s3`	Τρεις τελεστέοι· ανίχνευση υπερχείλισης
	add immediate	`addi $s1,$s2,100`	`$s1 = $s2 + 100`	+ σταθερά· ανίχνευση υπερχείλισης
	add unsigned	`addu $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Τρεις τελεστέοι· ανίχνευση υπερχείλισης
	subtract unsigned	`subu $s1,$s2,$s3`	`$s1 = $s2 - $s3`	Τρεις τελεστέοι· ανίχνευση υπερχείλισης
	add immediate unsigned	`addiu $s1,$s2,100`	`$s1 = $s2 + 100`	+ σταθερά· ανίχνευση υπερχείλισης
	move from coprocessor register	`mfc0 $s1,$epc`	`$s1 = $epc`	Χρησιμοποιείται για την αντιγραφή του PC εξαίρεσης και άλλων ειδικών καταχωρητών
	multiply	`mult $s2,$s3`	Hi, Lo = `$s2 × $s3`	προσημασμένο γινόμενο 64 bit στους Hi, Lo
	multiply unsigned	`multu $s2,$s3`	Hi, Lo = `$s2 × $s3`	απρόσημο γινόμενο 64 bit στους Hi, Lo
	divide	`div $s2,$s3`	Lo = `$s2 / $s3`, Hi = `$s2 mod s3`	Lo = πηλίκο, Hi = υπόλοιπο
	divide unsigned	`divu $s2,$s3`	Lo = `$s2 / $s3`, Hi = `$s2 mod s3`	Απρόσημο πηλίκο και υπόλοιπο
	move from Hi	`mfhi $s1`	`$s1` = Hi	Χρησιμοποιείται για να πάρει αντίγραφο του Hi
	move from Lo	`mflo $s1`	`$s1` = Lo	Χρησιμοποιείται για να πάρει αντίγραφο του Lo
Μεταφορά δεδομένων	load word	`lw $s1,100($s2)`	`$s1 = Memory[$s2+100]`	Λέξη από τη μνήμη σε καταχωρητή
	store word	`sw $s1,100($s2)`	`Memory[$s2+100] = $s1`	Λέξη από καταχωρητή στη μνήμη
	load half unsigned	`lhu $s1,100($s2)`	`$s1 = Memory[$s2 + 100]`	Ημιλέξη από τη μνήμη σε καταχωρητή
	store half	`sh $s1,100($s2)`	`Memory[$s2 + 100] = $s1`	Ημιλέξη από καταχωρητή στη μνήμη
	load byte unsigned	`lbu $s1,100($s2)`	`$s1 = Memory[$s2 + 100]`	Byte από τη μνήμη σε καταχωρητή
	store byte	`sb $s1,100($s2)`	`Memory[$s2 + 100] = $s1`	Byte από καταχωρητή στη μνήμη
	load upper immediate	`lui $s1,100`	`$s1 = 100 * 216`	Φορτώνει σταθερά στα υψηλότερα 16 bit
Λογικές πράξεις	and	`and $s1,$s2,$s3`	`$s1 = $s2 & $s3`	Τρεις τελεστέοι καταχωρητές· AND bit προς bit
	or	`or $s1,$s2,$s3`	`$s1 = $s2 | $s3`	Τρεις τελεστέοι καταχωρητές· OR bit προς bit
	nor	`nor $s1,$s2,$s3`	`$s1 = ~ ($s2 | $s3)`	Τρεις τελεστέοι καταχωρητές· NOR bit προς bit
	and immediate	`andi $s1,$s2,100`	`$s1 = $s2 & 100`	AND bit προς bit με σταθερά
	or immediate	`ori s1,$s2,100`	`$s1 = $s2 | 100`	OR bit προς bit με σταθερά
	shift left logical	`sll $s1,$s2,10`	`$s1 = $s2 << 10`	Αριστερή ολίσθηση με σταθερά
	shift right logical	`srl $s1,$s2,10`	`$s1 = $s2 >> 10`	Δεξιά ολίσθηση με σταθερά
Διακλάδωση υπό συνθήκη	branch on equal	`beq $s1,$s2,25`	αν `($s1 == $s2)` πήγαινε στο PC + 4 + 100	Έλεγχος ισότητας· διακλάδωση σχετική ως προς PC
	branch on not equal	`bne $s1,$s2,25`	αν `($s1 != $s2)` πήγαινε στο PC + 4 + 100	Έλεγχος ανισότητας· διακλάδωση σχετική ως προς PC
	set on less than	`slt $s1,$s2,$s3`	αν `($s2 < $s3)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από· συμπλήρωμα ως προς δύο
	set less than immediate	`slti $s1,$s2,100`	αν `($s2 < 100)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από στα-θερά· συμπλήρωμα ως προς δύο
	set less than unsigned	`sltu $s1,$s2,$s3`	αν `($s2 < $s3)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από· απρόσημοι αριθμοί
	set less than immediate unsigned	`sltiu $s1,$s2,100`	αν `($s2 < 100)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από σταθερά· απρόσημοι αριθμοί

ΕΙΚΟΝΑ 3.14 Η αρχιτεκτονική του MIPS που αποκαλύψαμε έως τώρα. Η μνήμη και οι καταχωρητές της αρχιτεκτονικής του MIPS δεν περιλαμβάνονται για λόγους οικονομίας χώρου, αλλά αυτή η ενότητα πρόσθεσε τους καταχωρητές hi και lo για την υποστήριξη του πολλαπλασιασμού και της διαίρεσης. Το χρώμα επισημαίνει τα τμήματα που αποκαλύψαμε μετά την Εικόνα 3.4 της σελίδας 193. Η γλώσσα μηχανής του MIPS υπάρχει επίσης στην κάρτα συνοπτικής αναφοράς του MIPS στην αρχή του βιβλίου.

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Άλμα χωρίς συνθήκη	jump	`j 2500`	μετάβαση στο 10000	Άλμα στη διεύθυνση προορισμού
	jump register	`jr $ra`	μετάβαση στον `$ra`	Για εντολή switch και για επιστροφή διαδικασίας
	jump and link	`jal 2500`	`$ra` = PC + 4· μετάβαση στο 10000	Για κλήση διαδικασίας

ΕΙΚΟΝΑ 3.14 Η αρχιτεκτονική του MIPS που αποκαλύψαμε έως τώρα *(συνέχεια)*

Όπως ακριβώς μπορούμε να παρουσιάσουμε δεκαδικούς αριθμούς (με βάση το 10) σε επιστημονική σημειογραφία, μπορούμε να παρουσιάσουμε και δυαδικούς αριθμούς σε επιστημονική σημειογραφία:

$$1{,}0_{\text{two}} \times 2^{-1}$$

Για τη διατήρηση ενός δυαδικού αριθμού σε κανονικοποιημένη μορφή, χρειαζόμαστε μια βάση που να μπορούμε να αυξήσουμε ή να μειώσουμε ακριβώς κατά τον αριθμό των bit που ο αριθμός πρέπει να ολισθήσει ώστε να έχει ένα μη μηδενικό ψηφίο στα αριστερά της υποδιαστολής. Μόνον η βάση 2 καλύπτει την ανάγκη αυτή.

Η αριθμητική υπολογιστών η οποία υποστηρίζει τέτοιους αριθμούς ονομάζεται **κινητής υποδιαστολής** (floating point) επειδή αναπαριστά αριθμούς στους οποίους το δυαδικό σημείο δεν είναι σε σταθερή θέση, όπως είναι για τους ακέραιους. Η γλώσσα προγραμματισμού C χρησιμοποιεί το όνομα float για τέτοιους αριθμούς. Όπως στην επιστημονική σημειογραφία, οι αριθμοί αναπαρίστανται με ένα μοναδικό μη μηδενικό ψηφίο στα αριστερά της υποδιαστολής. Σε δυαδική αναπαράσταση, η μορφή είναι

$$1{,}xxxxxxxxx_{\text{two}} \times 2^{yyyy}$$

(Αν και ο υπολογιστής αναπαριστά τον εκθέτη σε βάση 2 όπως και το υπόλοιπο του αριθμού, για την απλοποίηση της σημειογραφίας παρουσιάζουμε τον εκθέτη σε δεκαδική μορφή.)

κινητή υποδιαστολή (floating point) Αριθμητική υπολογιστών που αναπαριστά αριθμούς στους οποίους η υποδιαστολή δεν είναι σε σταθερή θέση.

Μια τυποποιημένη επιστημονική σημειογραφία για πραγματικούς αριθμούς σε κανονικοποιημένη μορφή προσφέρει τρία πλεονεκτήματα. Απλοποιεί την ανταλλαγή δεδομένων που περιλαμβάνουν αριθμούς κινητής υποδιαστολής· απλοποιεί τους αλγορίθμους αριθμητικής κινητής υποδιαστολής λόγω της βεβαιότητας ότι οι αριθμοί θα είναι πάντοτε σε αυτή τη μορφή· και αυξάνει την ακρίβεια των αριθμών που μπορούν να αποθηκευτούν σε μια λέξη, εφόσον τα όχι απαραίτητα προπορευόμενα 0 αντικαθίστανται από πραγματικά ψηφία στα δεξιά τής υποδιαστολής.

Αναπαράσταση κινητής υποδιαστολής

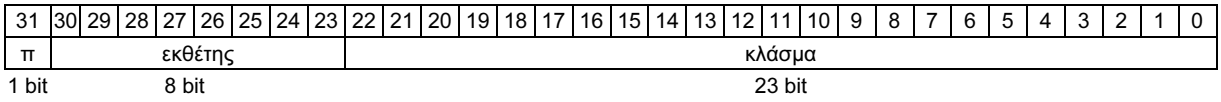
Ένας σχεδιαστής μιας αναπαράστασης κινητής υποδιαστολής πρέπει να βρει ένα συμβιβασμό μεταξύ του μεγέθους του **κλάσματος** (fraction) και του μεγέθους του **εκθέτη** (exponent) εφόσον μια λέξη σταθερού μεγέθους σημαίνει ότι πρέπει να πάρετε ένα bit από το ένα για να προσθέσετε ένα bit στο άλλο. Υπάρχει ένας συμβιβασμός μεταξύ ακρίβειας (precision) και εύρους (range). Η αύξηση του μεγέθους του κλάσματος ενισχύει την ακρίβειά του, ενώ η αύξηση του μεγέθους του εκθέτη αυξάνει το εύρος των αριθμών που μπορούν να αναπαρασταθούν. Όπως μας υπενθυμίζει η οδηγία σχεδιασμού από το Κεφάλαιο 2, η καλή σχεδίαση απαιτεί καλό συμβιβασμό.

κλάσμα (fraction) Η τιμή, γενικά μεταξύ 0 και 1, που τοποθετείται στο κλασματικό πεδίο.

εκθέτης (exponent) Στο σύστημα αναπαράστασης αριθμών της αριθμητικής κινητής υποδιαστολής, η τιμή που τοποθετείται στο πεδίο του εκθέτη.

Οι αριθμοί κινητής υποδιαστολής είναι συνήθως πολλαπλάσια του μεγέθους μίας λέξης. Η αναπαράσταση ενός αριθμού κινητής υποδιαστολής στον MIPS παρουσιάζεται παρακάτω, όπου π είναι το πρόσημο του αριθμού κινητής υποδιαστολής (1 σημαίνει αρνητικός), εκθέτης είναι η τιμή του πεδίου 8 bit του

εκθέτη (μαζί με το πρόσημο του εκθέτη), και κλάσμα είναι ο αριθμός των 23 bit. Αυτή η αναπαράσταση ονομάζεται *προσήμου και μεγέθους* (sign and magnitude), αφού το πρόσημο έχει ένα δικό του bit ξεχωριστά από τον υπόλοιπο αριθμό.



Γενικά, οι αριθμοί κινητής υποδιαστολής έχουν τη μορφή

$$(-1)^{\text{πρόσημο}} \times \text{κλάσμα} \times 2^{\text{εκθέτης}}$$

το *κλάσμα* εμπεριέχει την τιμή στο πεδίο του κλάσματος και το *εκθέτης* εμπεριέχει την τιμή στο πεδίο του εκθέτη· η ακριβής σχέση με αυτά τα πεδία θα εξηγηθεί σύντομα. (Θα δούμε σύντομα ότι ο MIPS κάνει κάτι λίγο πιο πολύπλοκο.)

Αυτά τα επιλεγμένα μεγέθη εκθέτη και κλάσματος δίνουν στην υπολογιστική αριθμητική του MIPS ένα εκπληκτικό εύρος. Σε έναν υπολογιστή μπορούν να αναπαρασταθούν κλάσματα τόσο μικρά όσο το $2{,}0_{\text{ten}} \times 10^{-38}$ και αριθμοί τόσο μεγάλοι όσο το $2{,}0_{\text{ten}} \times 10^{38}$. Δυστυχώς, το τεράστιο διαφέρει από το άπειρο, και έτσι είναι και πάλι πιθανό κάποιοι αριθμοί να είναι πολύ μεγάλοι. Έτσι, διακοπές λόγω υπερχείλισης μπορούν να συμβούν και σε αριθμητική κινητής υποδιαστολής όπως και σε ακέραια αριθμητική. Σημειώστε ότι **υπερχείλιση κινητής υποδιαστολής** (floating point overflow) εδώ σημαίνει ότι ο εκθέτης είναι πολύ μεγάλος για να αναπαρασταθεί από το πεδίο του εκθέτη.

υπερχείλιση κινητής υποδιαστολής (floating point overflow) Κατάσταση στην οποία ένας θετικός εκθέτης γίνεται πολύ μεγάλος για να χωρέσει στο πεδίο του εκθέτη (exponent).

Η κινητή υποδιαστολή παρέχει επίσης ένα νέο είδος εξαιρετικού συμβάντος. Όπως ακριβώς οι προγραμματιστές πρέπει να ξέρουν πότε έχουν υπολογίσει έναν αριθμό ο οποίος είναι πολύ μεγάλος για να αναπαρασταθεί, πρέπει να γνωρίζουν και αν το μη μηδενικό κλάσμα που υπολογίζουν έχει γίνει τόσο μικρό που δεν μπορεί να αναπαρασταθεί· οποιοδήποτε από τα δύο θα μπορούσε να οδηγήσει σε ένα πρόγραμμα που δίνει λάθος αποτελέσματα. Για να το διακρίνουμε από την υπερχείλιση, ονομάζουμε αυτό το συμβάν **ανεπάρκεια κινητής υποδιαστολής** (floating point underflow). Η κατάσταση αυτή συμβαίνει όταν ο αρνητικός εκθέτης είναι υπερβολικά μεγάλος για να χωρέσει στο εκθετικό πεδίο.

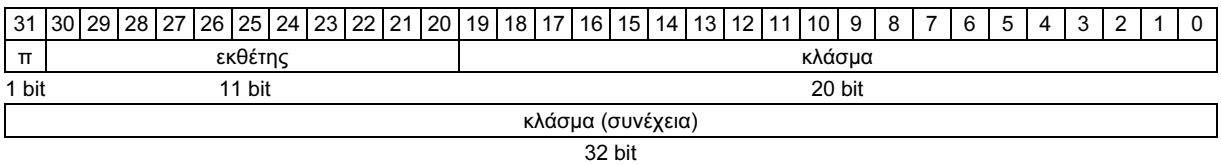
ανεπάρκεια κινητής υποδιαστολής (floating point underflow) Κατάσταση στην οποία ένας αρνητικός εκθέτης γίνεται πολύ μεγάλος για να χωρέσει στο πεδίο του εκθέτη (exponent).

Ένας τρόπος μείωσης των πιθανοτήτων εμφάνισης ανεπάρκειας και υπερχείλισης είναι η παροχή μιας άλλης μορφής που έχει μεγαλύτερο εκθέτη. Στη C ο αριθμός αυτός ονομάζεται double, και οι πράξεις με αριθμούς τύπου double ονομάζονται αριθμητική κινητής υποδιαστολής **διπλής ακρίβειας** (double precision)· αριθμητική κινητής υποδιαστολής **απλής ακρίβειας** (single precision) είναι το όνομα της προηγούμενης μορφής.

διπλή ακρίβεια (double precision) Μια τιμή κινητής υποδιαστολής που αναπαρίσταται με δύο λέξεις 32 bit.

απλή ακρίβεια (single precision) Μια τιμή κινητής υποδιαστολής που αναπαρίσταται με μία λέξη 32 bit.

Η αναπαράσταση ενός αριθμού κινητής υποδιαστολής διπλής ακρίβειας απαιτεί δύο λέξεις του MIPS, όπως φαίνεται στο επόμενο σχήμα, όπου *π* είναι πάλι το πρόσημο του αριθμού, *εκθέτης* είναι η τιμή του εκθετικού πεδίου 11 bit, και *κλάσμα* είναι αριθμός 52 bit στο κλάσμα.



Η διπλή ακρίβεια του MIPS επιτρέπει αριθμούς σχεδόν τόσο μικρούς όσο το $2{,}0_{ten} \times 10^{-308}$ και σχεδόν τόσο μεγάλους όσο το $2{,}0_{ten} \times 10^{308}$. Αν και η διπλή ακρίβεια αυξάνει το εύρος του εκθέτη, το μεγαλύτερο πλεονέκτημα είναι η μεγαλύτερη ακρίβειά της λόγω του μεγαλύτερου σημαντικού (significand).

Αυτές οι μορφές προχωρούν πέρα από τον MIPS. Αποτελούν μέρος του *προτύπου IEEE 754 floating-point*, που συναντάται σχεδόν σε κάθε υπολογιστή που σχεδιάστηκε από το 1980. Αυτό το πρότυπο βελτίωσε σημαντικά τόσο την ευκολία μεταφοράς προγραμμάτων κινητής υποδιαστολής όσο και την ποιότητα της αριθμητικής υπολογιστών.

Για να συμπτύξει ακόμη περισσότερα bit μέσα στο σημαντικό, το IEEE 754 κάνει το αρχικό 1 bit των κανονικοποιημένων δυαδικών αριθμών υπονοούμενο (implicit). Έτσι, ο αριθμός έχει στην πραγματικότητα μήκος 24 bit σε απλή ακρίβεια (το υπονοούμενο 1 και ένα κλάσμα 23 bit), και μήκος 53 bit σε διπλή ακρίβεια (1 + 52). Για να είμαστε ακριβείς, χρησιμοποιούμε τον όρο *σημαντικό* (significand) για τον αριθμό 24 bit ή 53 bit ο οποίος είναι 1 συν το κλάσμα, και *κλάσμα* (fraction) όταν εννοούμε τον αριθμό 23 bit ή 52 bit. Εφόσον το 0 δεν έχει αρχικό 1, του δίνεται η δεσμευμένη τιμή εκθέτη 0 ώστε το υλικό να μην προσαρτήσει σε αυτό ένα αρχικό 1.

Έτσι, το $00...00_{two}$ αναπαριστά το 0· η αναπαράσταση των υπόλοιπων αριθμών χρησιμοποιεί την προηγούμενη μορφή με την πρόσθεση του κρυμμένου 1:

$$(-1)^{\text{πρόσημο}} \times (1 + \text{κλάσμα}) \times 2^{\text{εκθέτης}}$$

όπου τα bit του κλάσματος αναπαριστούν έναν αριθμό μεταξύ 0 και 1 και ο *εκθέτης* καθορίζει την τιμή στο εκθετικό πεδίο, που θα εξηγηθεί αναλυτικά σύντομα. Αν μετρήσουμε τα bit του κλάσματος *από αριστερά προς τα δεξιά* s1, s2, s3, . . . , τότε η τιμή είναι

$$(-1)^{\text{πρόσημο}} \times (1 + (s1 \times 2^{-1}) + (s2 \times 2^{-2}) + (s3 \times 2^{-3}) + (s4 \times 2^{-4}) + ...) \times 2^{\text{εκθέτης}}$$

Η Εικόνα 3.15 παρουσιάζει τις κωδικοποιήσεις των αριθμών κινητής υποδιαστολής κατά IEEE 754. Άλλα χαρακτηριστικά του IEEE 754 είναι ειδικά σύμβολα αναπαράστασης ασυνήθιστων συμβάντων. Για παράδειγμα, αντί για διακοπή (interrupt) σε μια διαίρεση με το 0, το λογισμικό μπορεί να αναθέσει στο αποτέλεσμα μια σειρά bit που αναπαριστούν το $+\infty$ ή το $-\infty$· ο μεγαλύτερος εκθέτης δεσμεύεται γι' αυτά τα ειδικά σύμβολα. Όταν ο προγραμματιστής τυπώσει τα αποτελέσματα, το πρόγραμμα θα τυπώσει ένα σύμβολο του απείρου. (Για τους εκπαιδευμένους στα μαθηματικά, σκοπός του απείρου είναι η δημιουργία μιας τοπολογικής σύγκλισης για τους πραγματικούς αριθμούς.)

Το πρότυπο IEEE 754 έχει ακόμη ένα σύμβολο για το αποτέλεσμα μη επιτρεπτών πράξεων, όπως η 0/0 ή η αφαίρεση του απείρου από το άπειρο. Αυτό το σύμβολο είναι το *NaN*, από τη φράση *Not a Numbe*r (όχι αριθμός). Σκοπός των συμβόλων NaN είναι να επιτρέψουν στους προγραμματιστές να αναβάλλουν μερικούς ελέγχους και αποφάσεις για αργότερα στο πρόγραμμα, όταν αυτό θα είναι πιο βολικό.

Οι σχεδιαστές του προτύπου IEEE 754 χρειάζονταν επίσης μια αναπαράσταση κινητής υποδιαστολής με την οποία θα μπορούσαν εύκολα να επεξεργαστούν ακέραιες συγκρίσεις κυρίως για ταξινόμηση. Αυτή η ανάγκη εξηγεί γιατί το πρόσημο βρίσκεται στο πιο σημαντικό bit, επιτρέποντας ένα γρήγορο έλεγχο μικρότερου από, μεγαλύτερου από, ή ίσου με το 0. (Είναι λίγο πιο πολύπλοκο από μια απλή ταξινόμηση ακεραίων, αφού η σημειογραφία είναι ουσιαστικά προσήμου και μεγέθους παρά συμπληρώματος ως προς δύο.)

Απλή ακρίβεια		Διπλή ακρίβεια		Αναπαριστώμενο αντικείμενο
Εκθέτης	Κλάσμα	Εκθέτης	Κλάσμα	
0	0	0	0	0
0	μη μηδενικό	0	μη μηδενικό	± μη κανονικοποιημένος αριθμός
1-254	οτιδήποτε	1-2046	οτιδήποτε	± αριθμός κινητής υποδιαστολής
255	0	2047	0	± άπειρο
255	μη μηδενικό	2047	μη μηδενικό	NaN (όχι αριθμός)

ΕΙΚΟΝΑ 3.15 Κωδικοποίηση IEEE 754 αριθμών κινητής υποδιαστολής. Ένα ξεχωριστό bit προσήμου καθορίζει το πρόσημο. Οι μη κανονικοποιημένοι αριθμοί περιγράφονται στην ενότητα «Επιπλέον ανάπτυξη» της σελίδας 235.

Η τοποθέτηση του εκθέτη πριν από το σημαντικό απλοποιεί επίσης την ταξινόμηση αριθμών κινητής υποδιαστολής με τη χρήση εντολών σύγκρισης ακεραίων, εφόσον οι αριθμοί με μεγαλύτερους εκθέτες φαίνονται μεγαλύτεροι από αριθμούς με μικρότερους εκθέτες μια και οι δύο εκθέτες έχουν το ίδιο πρόσημο.

Οι αρνητικοί εκθέτες δημιουργούν μια πρόκληση για την απλοποιημένη ταξινόμηση. Αν χρησιμοποιήσουμε συμπλήρωμα ως προς δύο, ή οποιαδήποτε άλλη σημειογραφία στην οποία οι αρνητικοί εκθέτες έχουν 1 στο πιο σημαντικό bit του πεδίου εκθέτη, ο αρνητικός εκθέτης θα μοιάζει με ένα μεγάλο αριθμό. Για παράδειγμα, η αναπαράσταση του $1{,}0_{two} \times 2^{-1}$ θα ήταν

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.	.	.

(Θυμηθείτε ότι το αρχικό 1 υπονοείται στο σημαντικό.) Η τιμή $1{,}0_{two} \times 2^{+1}$ θα έμοιαζε ως ο μικρότερος δυαδικός αριθμός

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.	.	.

Η κατάλληλη σημειογραφία πρέπει επομένως να αναπαριστά τον πιο αρνητικό εκθέτη ως $00\ldots00_{two}$, και τον πιο θετικό ως $11\ldots11_{two}$. Αυτή η σύμβαση ονομάζεται *πολωμένη σημειογραφία* (biased notation), όπου η πόλωση (bias) είναι ο αριθμός που αφαιρείται από την κανονική, απρόσημη αναπαράσταση για τον καθορισμό της πραγματικής τιμής.

Το πρότυπο IEEE 754 χρησιμοποιεί μια πόλωση 127 για απλή ακρίβεια, και έτσι το –1 αναπαρίσταται από τη σειρά bit της τιμής $-1 + 127_{ten}$, ή $126_{ten} = 0111\ 1110_{two}$, και το +1 αναπαρίσταται από το $1 + 127$, ή $128_{ten} = 1000\ 0000_{two}$. Ο πολωμένος εκθέτης σημαίνει ότι η τιμή που αναπαρίσταται από έναν αριθμό κινητής υποδιαστολής είναι στην πραγματικότητα

$$(-1)^{\text{πρόσημο}} \times (1 + \text{κλάσμα}) \times 2^{(\text{εκθέτης} - \text{πόλωση})}$$

Η εκθετική πόλωση για τη διπλή ακρίβεια είναι 1023.

Με τον τρόπο αυτόν, η σημειογραφία κατά IEEE 754 μπορεί να υποστεί επεξεργασία από ακέραιες συγκρίσεις για την επιτάχυνση της ταξινόμησης αριθμών κινητής υποδιαστολής. Ας παρουσιάσουμε την αναπαράσταση.

Αναπαράσταση κινητής υποδιαστολής

ΠΑΡΑΔΕΙΓΜΑ

Δείξτε τη δυαδική αναπαράσταση του αριθμού $-0{,}75_{ten}$ κατά IEEE 754 σε απλή και διπλή ακρίβεια.

ΑΠΑΝΤΗΣΗ

Ο αριθμός $-0{,}75_{ten}$ είναι επίσης

$$-3/4_{ten} \text{ ή } -3/2^2_{ten}$$

Επίσης, αναπαρίσταται από το δυαδικό κλάσμα

$$-11_{two}/2^2_{ten} \text{ ή } -0{,}11_{two}$$

Στην επιστημονική σημειογραφία, η τιμή είναι

$$-0{,}11_{two} \times 2^0$$

και σε κανονικοποιημένη επιστημονική σημειογραφία, είναι

$$-1{,}1_{two} \times 2^{-1}$$

Η γενική αναπαράσταση για έναν αριθμό απλής ακρίβειας είναι

$$(-1)^{\text{πρόσημο}} \times (1 + \text{κλάσμα}) \times 2^{(\text{εκθέτης} - 127)}$$

Όταν αφαιρούμε την πόλωση 127 από τον εκθέτη τού $-1{,}1_{two} \times 2^{-1}$, το αποτέλεσμα είναι

$$(-1)^1 \times (1 + ,1000\ 0000\ 0000\ 0000\ 0000\ 0000_{two}) \times 2^{(126 - 127)}$$

Η απλής ακρίβειας δυαδική αναπαράσταση του $-0{,}75_{ten}$ είναι τότε

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 bit 8bit 23 bit

Η διπλής ακρίβεια αναπαράσταση είναι

$$(-1)^1 \times (1 + ,1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two}) \times 2^{(1022-1023)}$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 bit — 11 bit — 20 bit

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

32 bit

Τώρα ας προσπαθήσουμε να πάμε στην άλλη κατεύθυνση.

Μετατροπή δυαδικού σε δεκαδικό κινητής υποδιαστολής

ΠΑΡΑΔΕΙΓΜΑ

Ποιος δεκαδικός αριθμός αναπαρίσταται από τον παρακάτω αριθμό κινητής υποδιαστολής απλής ακρίβειας;

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.	.	.

ΑΠΑΝΤΗΣΗ

Το bit προσήμου είναι 1, το εκθετικό πεδίο περιέχει το 129, και το κλασματικό πεδίο περιέχει το $1 \times 2^{-2} = 1/4$, ή 0,25. Με τη χρήση της βασικής εξίσωσης,

$$(-1)^{\text{πρόσημο}} \times (1 + \text{κλάσμα}) \times 2^{(\text{εκθέτης} - \text{πόλωση})}$$

$$= (-1)^1 \times (1+0,25) \times 2^{(129-127)}$$
$$= -1 \times 1,25 \times 2^2$$
$$= -1,25 \times 4$$
$$= -5,0$$

Στις επόμενες ενότητες θα παρουσιάσουμε τους αλγορίθμους για πρόσθεση και πολλαπλασιασμό κινητής υποδιαστολής. Στον πυρήνα τους χρησιμοποιούν τις αντίστοιχες ακέραιες πράξεις, αλλά είναι απαραίτητη επιπλέον πρόνοια για το χειρισμό των εκθετών και την κανονικοποίηση του αποτελέσματος. Παρουσιάζουμε πρώτα μια εξαγωγή των αλγορίθμων για δεκαδικούς βασισμένη στη διαίσθηση, και στη συνέχεια πιο λεπτομερώς τη δυαδική έκδοση στις εικόνες.

Επιπλέον ανάπτυξη: Σε μια προσπάθεια αύξησης του εύρους χωρίς αφαίρεση bit από το σημαντικό, κάποιοι υπολογιστές πριν από το πρότυπο IEEE 754 χρησιμοποίησαν βάση διαφορετική από το 2. Για παράδειγμα, τα μεγάλα υπολογιστικά συστήματα IBM 360 και 370 χρησιμοποιούν βάση 16. Εφόσον η αλλαγή του εκθέτη της IBM κατά ένα σημαίνει ολίσθηση του σημαντικού κατά 4 bit, αριθμοί κανονικοποιημένοι ως προς βάση 16 μπορούν να έχουν έως και τρία αρχικά bit 0! Επομένως, τα δεκαεξαδικά ψηφία σημαίνουν ότι μπορούν να παραλειφθούν μέχρι τρία bit από το σημαντικό, κάτι που οδηγεί σε απροσδόκητα προβλήματα στην ακρίβεια της αριθμη-

τικής κινητής υποδιαστολής, όπως σημειώνεται στην Ενότητα 3.6. Πρόσφατα μεγάλα υπολογιστικά συστήματα της IBM υποστηρίζουν τόσο το πρότυπο IEEE 754 όσο και τη δεκαεξαδική μορφή.

Πρόσθεση κινητής υποδιαστολής

Ας προσθέσουμε αριθμούς σε επιστημονική σημειογραφία «με το χέρι», για να δείξουμε τα προβλήματα της πρόσθεσης αριθμών κινητής υποδιαστολής: $9{,}999_{ten} \times 10^{1} + 1{,}610_{ten} \times 10^{-1}$. Υποθέστε ότι μπορούμε να αποθηκεύσουμε μόνο τέσσερα δεκαδικά ψηφία του σημαντικού και δύο δεκαδικά ψηφία του εκθέτη.

Βήμα 1. Για να μπορούμε να προσθέσουμε σωστά αυτούς τους αριθμούς, πρέπει να ευθυγραμμίσουμε την υποδιαστολή του αριθμού που έχει το μικρότερο εκθέτη. Έτσι, χρειαζόμαστε μια μορφή του μικρότερου αριθμού, $1{,}610_{ten} \times 10^{-1}$, που ταιριάζει με το μεγαλύτερο εκθέτη. Παίρνουμε αυτή τη μορφή παρατηρώντας ότι υπάρχουν πολλές αναπαραστάσεις ενός μη κανονικοποιημένου αριθμού κινητής υποδιαστολής σε επιστημονική σημειογραφία:

$$1{,}610_{ten} \times 10^{-1} = 0{,}1610_{ten} \times 10^{0} = 0{,}01610_{ten} \times 10^{1}$$

Ο αριθμός στα δεξιά είναι η έκδοση που θέλουμε, επειδή ο εκθέτης του ταιριάζει με τον εκθέτη του μεγάλου αριθμού $9{,}999_{ten} \times 10^{1}$. Έτσι, το πρώτο βήμα ολισθαίνει το σημαντικό του μικρότερου αριθμού δεξιά, μέχρι ο διορθωμένος εκθέτης του να ταιριάζει με αυτόν του μεγαλύτερου αριθμού. Αλλά μπορούμε να αναπαραστήσουμε μόνο τέσσερα δεκαδικά ψηφία, οπότε, μετά την ολίσθηση, ο αριθμός είναι στην πραγματικότητα:

$$0{,}016_{ten} \times 10^{1}$$

Βήμα 2. Στη συνέχεια έρχεται η πρόσθεση των σημαντικών:

$$\begin{array}{r} 9{,}999_{ten} \\ +\ 0{,}016_{ten} \\ \hline 10{,}015_{ten} \end{array}$$

Το άθροισμα είναι $10{,}015_{ten} \times 10^{1}$.

Βήμα 3. Αυτό το άθροισμα δεν είναι σε κανονικοποιημένη επιστημονική σημειογραφία, οπότε πρέπει να το προσαρμόσουμε:

$$10{,}015_{ten} \times 10^{1} = 1{,}0015_{ten} \times 10^{2}$$

Έτσι, μετά την πρόσθεση ίσως χρειάζεται να ολισθήσουμε το άθροισμα για να το φέρουμε σε κανονικοποιημένη μορφή, προσαρμόζοντας κατάλληλα τον εκθέτη. Αυτό το παράδειγμα παρουσιάζει την

ολίσθηση προς τα δεξιά αλλά, αν ο ένας αριθμός ήταν θετικός και ο άλλος αρνητικός, θα ήταν πιθανό το άθροισμα να έχει πολλά αρχικά μηδενικά και να απαιτεί αριστερές ολισθήσεις. Όποτε ο εκθέτης αυξάνεται ή μειώνεται, πρέπει να ελέγξουμε για υπερχείλιση (overflow) ή ανεπάρκεια (underflow) — που σημαίνει ότι πρέπει να βεβαιωθούμε ότι ο εκθέτης εξακολουθεί να χωράει στο πεδίο του.

Βήμα 4. Αφού υποθέσαμε ότι το σημαντικό μπορεί να έχει μήκος μόνο τέσσερα ψηφία (εξαιρώντας το πρόσημο), πρέπει να στρογγυλοποιήσουμε τον αριθμό. Στο αλγόριθμό μας του σχολείου, οι κανόνες αποκόπτουν τον αριθμό αν το ψηφίο στα δεξιά της κατάλληλης θέσης είναι μεταξύ 0 και 4, και προσθέτουν 1 στο ψηφίο αν ο αριθμός στα δεξιά είναι μεταξύ 5 και 9. Ο αριθμός

$$1{,}0015_{ten} \times 10^2$$

στρογγυλοποιείται στα τέσσερα ψηφία του σημαντικού στον

$$1{,}002_{ten} \times 10^2$$

αφού το τέταρτο ψηφίο στα δεξιά της υποδιαστολής είναι μεταξύ 5 και 9. Σημειώστε ότι αν ήμαστε άτυχοι στη στρογγυλοποίηση, όπως με την πρόσθεση του 1 σε μια σειρά από 9, το άθροισμα μπορεί να μην ήταν πια κανονικοποιημένο και θα χρειαζόταν να εκτελέσουμε πάλι το βήμα 3.

Η Εικόνα 3.16 δείχνει τον αλγόριθμο για τη δυαδική πρόσθεση κινητής υποδιαστολής που είναι σύμφωνη με το δεκαδικό παράδειγμα. Τα βήματα 1 και 2 είναι παρόμοια με το παράδειγμα που μόλις εξετάσαμε: προσαρμόζουν το σημαντικό του αριθμού με το μικρότερο εκθέτη και μετά προσθέτουν τα δύο σημαντικά. Το βήμα 3 κανονικοποιεί τα αποτελέσματα, επιβάλλοντας έναν έλεγχο για υπερχείλιση ή ανεπάρκεια. Ο έλεγχος για υπερχείλιση και ανεπάρκεια στο βήμα 3 εξαρτάται από την ακρίβεια των τελεστέων. Θυμηθείτε ότι η σειρά με όλα τα bit μηδέν στον εκθέτη δεσμεύεται και χρησιμοποιείται για την αναπαράσταση του μηδενός σε κινητή υποδιαστολή. Επίσης, η σειρά με όλα τα bit ένα στον εκθέτη δεσμεύεται για να δείξει τιμές και καταστάσεις έξω από την εμβέλεια των αριθμών κινητής υποδιαστολής (δείτε την ενότητα «Επιπλέον ανάπτυξη» στη σελίδα 235). Έτσι, για απλή ακρίβεια ο μέγιστος εκθέτης είναι 127 και ο ελάχιστος εκθέτης –126. Τα όρια για διπλή ακρίβεια είναι 1023 και –1022, αντίστοιχα.

ΠΑΡΑΔΕΙΓΜΑ

Δεκαδική πρόσθεση κινητής υποδιαστολής

Προσπαθήστε να προσθέσετε τους αριθμούς $0{,}5_{ten}$ και $-0{,}4375_{ten}$ σε δυαδική μορφή χρησιμοποιώντας τον αλγόριθμο της Εικόνας 3.16.

ΑΠΑΝΤΗΣΗ

Ας δούμε πρώτα τη δυαδική έκδοση των δύο αριθμών σε κανονικοποιημένη επιστημονική σημειογραφία, υποθέτοντας ότι κρατούμε 4 bit ακρίβειας.

$$\begin{aligned} 0{,}5_{ten} &= 1/2_{ten} &&= 1/2^1{}_{ten} \\ &= 0{,}1_{two} &&= 0{,}1_{two} \times 2^0 &&= 1{,}000_{two} \times 2^{-1} \\ -0{,}4375_{ten} &= -7/16_{ten} &&= -7/2^4{}_{ten} \\ &= -0{,}0111_{two} &&= -0{,}0111_{two} \times 2^0 &&= -1{,}110_{two} \times 2^{-2} \end{aligned}$$

Τώρα ακολουθούμε τον αλγόριθμο:

Βήμα 1. Το σημαντικό του αριθμού με το μικρότερο εκθέτη ($-1{,}11_{two} \times 2^{-2}$) ολισθαίνει δεξιά μέχρι ο εκθέτης του να ταιριάζει με αυτόν του μεγαλύτερου αριθμού:

$$-1{,}110_{two} \times 2^{-2} = 0{,}111_{two} \times 2^{-1}$$

Βήμα 2. Προσθέτουμε τα σημαντικά:

$$1{,}000_{two} \times 2^{-1} + (-0{,}111_{two} \times 2^{-1}) = 0{,}001_{two} \times 2^{-1}$$

Βήμα 3. Κανονικοποιούμε το άθροισμα ελέγχοντας για υπερχείλιση ή ανεπάρκεια:

$$\begin{aligned} 0{,}001_{two} \times 2^{-1} &= 0{,}010_{two} \times 2^{-2} = 0{,}100_{two} \times 2^{-3} \\ &= 1{,}000_{two} \times 2^{-4} \end{aligned}$$

Εφόσον $127 \geq -4 \geq -126$, δεν υπάρχει υπερχείλιση ή ανεπάρκεια. (Ο πολωμένος εκθέτης θα ήταν $-4 + 127$, ή 123, το οποίο βρίσκεται μεταξύ του 1 και του 254, το μικρότερο και το μεγαλύτερο μη δεσμευμένους πολωμένους εκθέτες.)

Βήμα 4. Στρογγυλοποιούμε το άθροισμα:

$$1{,}000_{two} \times 2^{-4}$$

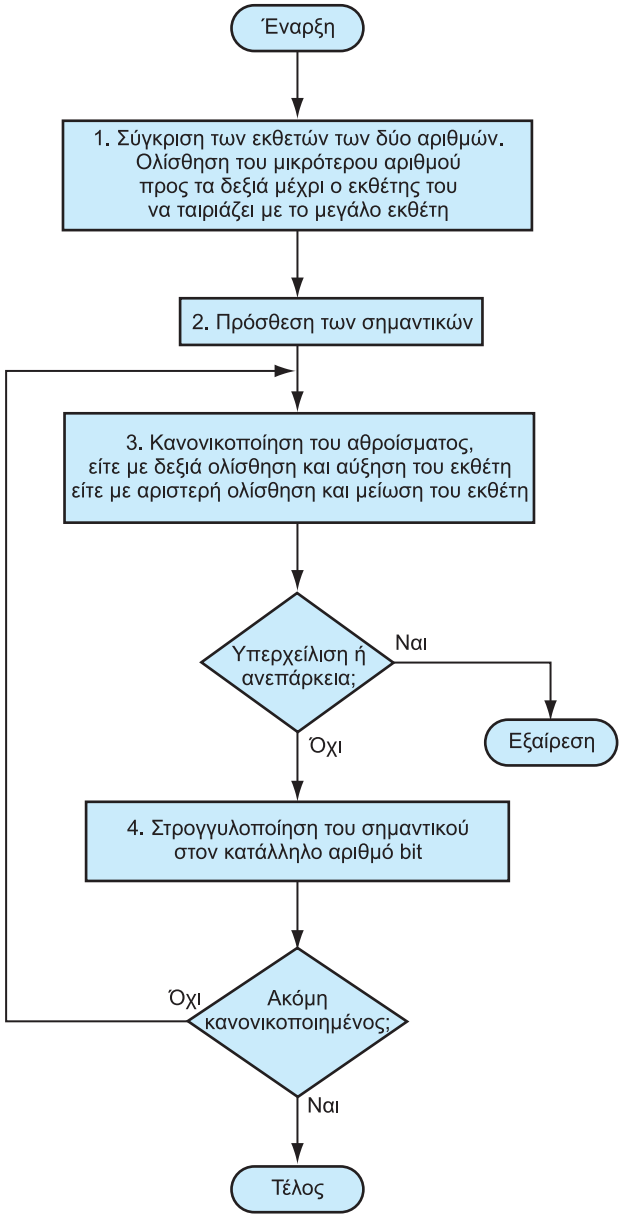
Το άθροισμα ήδη χωράει ακριβώς στα 4 bit, άρα δεν υπάρχει αλλαγή στα bit λόγω στρογγυλοποίησης.

Το άθροισμα είναι λοιπόν

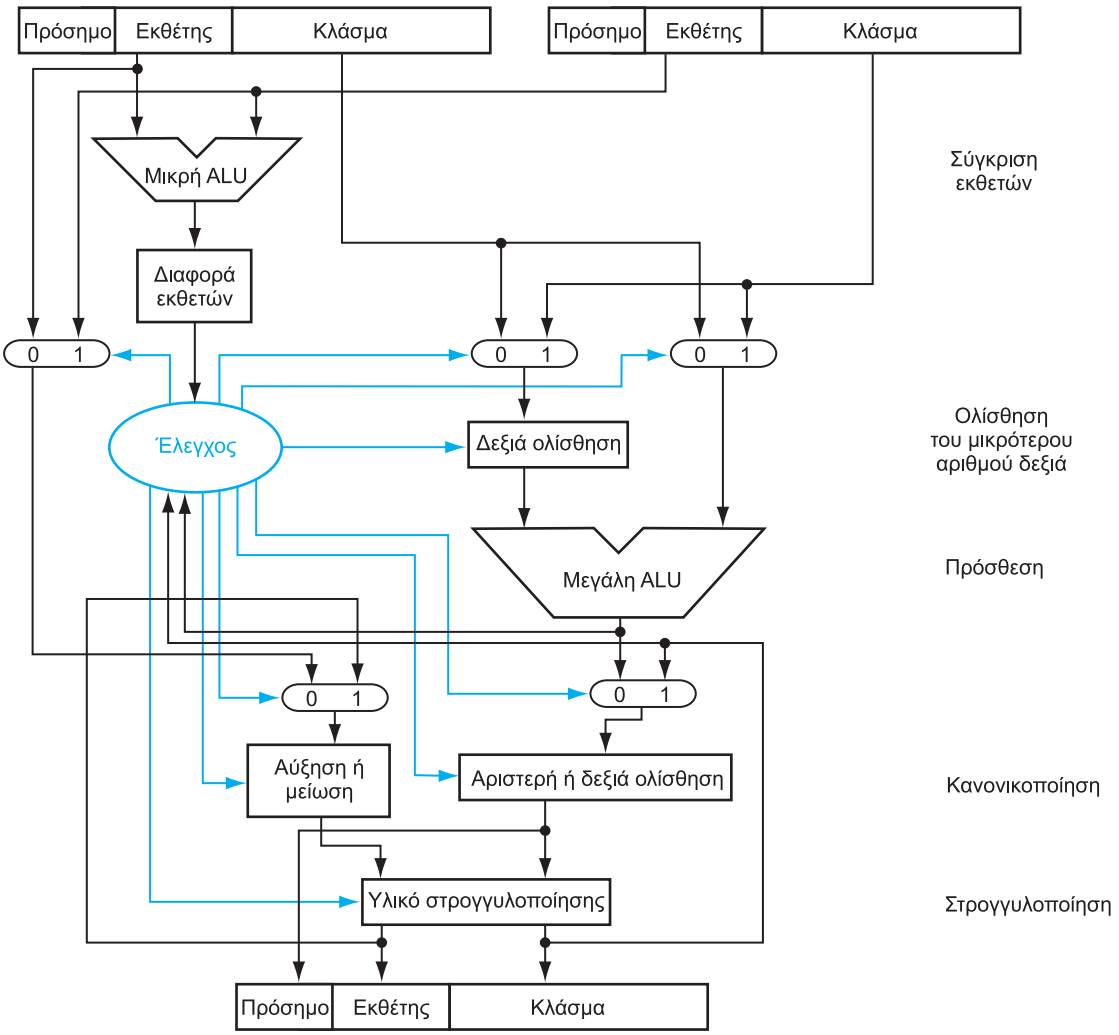
$$\begin{aligned} 1{,}000_{two} \times 2^{-4} &= 0{,}0001000_{two} &&= 0{,}0001_{two} \\ &= 1/2^4{}_{ten} &&= 1/16_{ten} &&= 0{,}0625_{ten} \end{aligned}$$

Το άθροισμα είναι αυτό που θα αναμέναμε από την πρόσθεση του $0{,}5_{ten}$ στο $-0{,}4375_{ten}$.

Πολλοί υπολογιστές αφιερώνουν υλικό στην εκτέλεση των πράξεων κινητής υποδιαστολής όσο πιο γρήγορα είναι δυνατόν. Η Εικόνα 3.17 δίνει το περίγραμμα της βασικής οργάνωσης του υλικού για την πρόσθεση κινητής υποδιαστολής.



ΕΙΚΟΝΑ 3.16 Πρόσθεση κινητής υποδιαστολής. Η κανονική διαδρομή είναι να εκτελέσουμε τα βήματα 3 και 4 μία φορά, αλλά αν η στρογγυλοποίηση προκαλέσει μη κανονικοποιημένο άθροισμα, πρέπει να επαναλάβουμε το βήμα 3.



ΕΙΚΟΝΑ 3.17 Διάγραμμα δομής μιας αριθμητικής μονάδας αφιερωμένης στη πρόσθεση κινητής υποδιαστολής. Τα βήματα της Εικόνας 3.16 αντιστοιχούν σε κάθε μπλοκ, από επάνω προς τα κάτω. Πρώτα αφαιρείται ο εκθέτης ενός τελεστέου από τον άλλο με τη χρήση μιας μικρής ALU για να προσδιοριστεί ποιος είναι μεγαλύτερος και πόσο. Αυτή η διαφορά ελέγχει τους τρεις πολυπλέκτες (multiplexers)· από αριστερά προς τα δεξιά, επιλέγουν το μεγαλύτερο εκθέτη, το σημαντικό του μικρότερου αριθμού, και το σημαντικό του μεγαλύτερου αριθμού. Το μικρότερο σημαντικό ολισθαίνει δεξιά και, μετά, τα σημαντικά προστίθενται με τη χρήση της μεγάλης ALU. Το βήμα κανονικοποίησης στη συνέχεια ολισθαίνει το άθροισμα αριστερά ή δεξιά και αυξάνει ή μειώνει τον εκθέτη. Η στρογγυλοποίηση δημιουργεί το τελικό αποτέλεσμα, το οποίο μπορεί να απαιτεί πάλι κανονικοποίηση.

Πολλαπλασιασμός κινητής υποδιαστολής

Τώρα που έχουμε εξηγήσει την πρόσθεση κινητής υποδιαστολής, ας δοκιμάσουμε τον πολλαπλασιασμό κινητής υποδιαστολής. Καταρχήν θα πολλαπλασιάσουμε δεκαδικούς αριθμούς σε επιστημονική σημειογραφία «με το χέρι»: $1{,}110_{ten} \times 10^{10} \times 9{,}200_{ten} \times 10^{-5}$. Υποθέστε ότι μπορούμε να αποθηκεύσουμε μόνο τέσσερα ψηφία του σημαντικού και δύο ψηφία του εκθέτη.

Βήμα 1. Αντίθετα με την πρόσθεση, υπολογίζουμε τον εκθέτη του γινομένου προσθέτοντας απλώς τους εκθέτες των τελεστέων:

$$\text{Νέος εκθέτης} = 10 + (-5) = 5$$

Ας κάνουμε το ίδιο και με τους πολωμένους εκθέτες για να βεβαιωθούμε ότι παίρνουμε το ίδιο αποτέλεσμα: 10 + 127 = 137 και –5 + 127 = 122, οπότε

$$\text{Νέος εκθέτης} = 137 + 122 = 259$$

Αυτό το αποτέλεσμα είναι πολύ μεγάλο για το πεδίο 8 bit του εκθέτη, άρα κάτι μας διέφυγε! Το πρόβλημα είναι με την πόλωση επειδή προσθέτουμε και τις πολώσεις και τους εκθέτες:

$$\text{Νέος εκθέτης} = (10 + 127) + (-5 + 127) = (5 + 2 \times 127) = 259$$

Επομένως, για να πάρουμε το σωστό πολωμένο άθροισμα όταν προσθέτουμε πολωμένους αριθμούς, πρέπει να αφαιρέσουμε την πόλωση από το άθροισμα:

$$\text{Νέος εκθέτης} = 137 + 122 - 127 = 259 - 127 = 132 = (5 + 127)$$

και το 5 είναι πράγματι ο εκθέτης που υπολογίσαμε αρχικά.

Βήμα 2. Ακολουθεί ο πολλαπλασιασμός των σημαντικών:

$$\begin{array}{r} 1{,}110_{ten} \\ \times\ 9{,}200_{ten} \\ \hline 0000 \\ 0000 \\ 2220 \\ 9990 \\ \hline 10212000_{ten} \end{array}$$

Υπάρχουν τρία ψηφία στα δεξιά της υποδιαστολής για κάθε τελεστέο, άρα η υποδιαστολή τοποθετείται έξι ψηφία από τα δεξιά στο σημαντικό του γινομένου:

$$10{,}212000_{ten}$$

Υποθέτοντας ότι μπορούμε να κρατήσουμε μόνο τρία ψηφία στα δεξιά της υποδιαστολής, το γινόμενο είναι $10{,}212_{ten} \times 10^{5}$.

Βήμα 3. Το γινόμενο αυτό δεν είναι κανονικοποιημένο, άρα χρειάζεται να το κανονικοποιήσουμε:

$$10{,}212_{ten} \times 10^5 = 1{,}0212_{ten} \times 10^6$$

Έτσι, μετά τον πολλαπλασιασμό μπορούμε να ολισθήσουμε δεξιά το γινόμενο κατά ένα ψηφίο ώστε να έλθει σε κανονικοποιημένη μορφή, προσθέτοντας 1 στον εκθέτη. Στο σημείο αυτό, μπορούμε να ελέγξουμε για υπερχείλιση και ανεπάρκεια. Υπερχείλιση μπορεί να συμβεί αν και οι δύο τελεστέοι είναι μικροί — δηλαδή, αν και οι δύο έχουν μεγάλους αρνητικούς εκθέτες.

Βήμα 4. Υποθέσαμε ότι το σημαντικό έχει μήκος μόνο τέσσερα ψηφία (εκτός από το πρόσημο), άρα πρέπει να στρογγυλοποιήσουμε τον αριθμό. Ο αριθμός

$$1{,}0212_{ten} \times 10^6$$

στρογγυλοποιείται στα τέσσερα ψηφία του σημαντικού στον

$$1{,}021_{ten} \times 10^6$$

Βήμα 5. Το πρόσημο του γινομένου εξαρτάται από τα πρόσημα των αρχικών τελεστέων. Αν είναι ίδια, το πρόσημο είναι θετικό· αλλιώς είναι αρνητικό. Έτσι το γινόμενο είναι

$$+\ 1{,}021_{ten} \times 10^6$$

Το πρόσημο του αθροίσματος στον αλγόριθμο της πρόσθεσης προσδιοριζόταν από την πρόσθεση των σημαντικών, αλλά στον πολλαπλασιασμό το πρόσημο του γινομένου καθορίζεται από τα πρόσημα των τελεστέων.

Και πάλι, όπως δείχνει η Εικόνα 3.18 ο πολλαπλασιασμός των δυαδικών αριθμών κινητής υποδιαστολής είναι αρκετά όμοιος με τα βήματα που μόλις ολοκληρώσαμε. Καταρχήν υπολογίζουμε το νέο εκθέτη του γινομένου προσθέτοντας τους πολωμένους εκθέτες, φροντίζοντας να αφαιρέσουμε μία πόλωση ώστε να πάρουμε το σωστό αποτέλεσμα. Το επόμενο βήμα είναι ο πολλαπλασιασμός των σημαντικών, ακολουθούμενος από ένα προαιρετικό βήμα κανονικοποίησης. Το μέγεθος του εκθέτη ελέγχεται για υπερχείλιση ή ανεπάρκεια και, μετά, το γινόμενο στρογγυλοποιείται. Αν η στρογγυλοποίηση οδηγήσει σε επιπλέον κανονικοποίηση, ελέγχουμε άλλη μία φορά το μέγεθος του εκθέτη. Τέλος, ορίζουμε το bit προσήμου ίσο με 1 αν τα πρόσημα των τελεστέων ήταν διαφορετικά (αρνητικό γινόμενο) ή ίσο με 0 αν ήταν ίδια (θετικό γινόμενο).

Δεκαδικός πολλαπλασιασμός κινητής υποδιαστολής

ΠΑΡΑΔΕΙΓΜΑ

Ας δοκιμάσουμε να πολλαπλασιάσουμε τους αριθμούς $0{,}5_{ten}$ και $-0{,}4375_{ten}$, χρησιμοποιώντας τα βήματα της Εικόνας 3.18.

ΑΠΑΝΤΗΣΗ

Στο δυαδικό σύστημα, αυτό που πρέπει να γίνει είναι να πολλαπλασιαστεί ο $1{,}000_{two} \times 2^{-1}$ με τον $-1{,}110_{two} \times 2^{-2}$.

Βήμα 1. Πρόσθεση των εκθετών χωρίς πόλωση:

$$-1 + (-2) = -3$$

ή, χρησιμοποιώντας την πολωμένη αναπαράσταση:

$$(-1 + 127) + (-2 + 127) - 127 = (-1 - 2) + (127 + 127 - 127)$$
$$= -3 + 127 = 124$$

Βήμα 2. Πολλαπλασιασμός των σημαντικών:

$$\begin{array}{rr} & 1{,}000_{two} \\ \times & 1{,}110_{two} \\ \hline & 0000 \\ & 1000 \\ & 1000 \\ & 1000 \\ \hline & 1110000_{two} \end{array}$$

Το γινόμενο είναι $1{,}110000_{two} \times 2^{-3}$, αλλά πρέπει να το κρατήσουμε στα 4 bit οπότε είναι $1{,}110_{two} \times 2^{-3}$.

Βήμα 3. Τώρα ελέγχουμε το γινόμενο για να βεβαιωθούμε ότι είναι κανονικοποιημένο, και μετά ελέγχουμε τον εκθέτη για υπερχείλιση ή ανεπάρκεια. Το γινόμενο είναι ήδη κανονικοποιημένο και, αφού $127 \geq -3 \geq -126$, δεν υπάρχει υπερχείλιση ή ανεπάρκεια. (Με πολωμένη αναπαράσταση, $254 \geq 124 \geq 1$, άρα ο εκθέτης χωράει.)

Βήμα 4. Η στρογγυλοποίηση του γινομένου δεν επιφέρει καμία αλλαγή:

$$1{,}110_{two} \times 2^{-3}$$

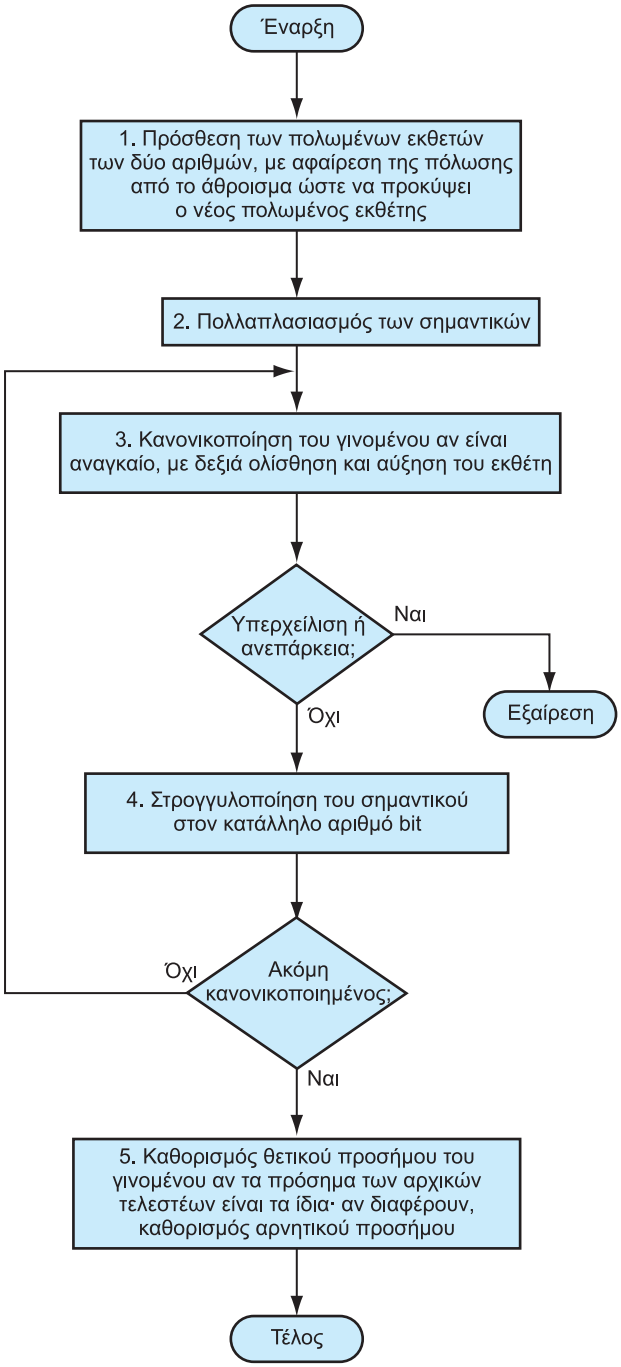
Βήμα 5. Αφού τα πρόσημα των αρχικών τελεστέων διαφέρουν, κάνουμε το πρόσημο του γινομένου αρνητικό. Έτσι, το γινόμενο είναι

$$-1{,}110_{two} \times 2^{-3}$$

Μετατρέπουμε στο δεκαδικό για να ελέγξουμε τα αποτελέσματά μας:

$$-1{,}110_{two} \times 2^{-3} = -0{,}001110_{two} = -0{,}00111_{two}$$
$$= -7/2^{5}{}_{ten} = -7/32_{ten} = -0{,}21875_{ten}$$

Το γινόμενο του $0{,}5_{ten}$ και του $-0{,}4375_{ten}$ είναι πράγματι $-0{,}21875_{ten}$.



ΕΙΚΟΝΑ 3.18 Πολλαπλασιασμός κινητής υποδιαστολής. Η κανονική διαδρομή είναι να εκτελέσουμε τα βήματα 3 και 4 μία φορά, αλλά αν η στρογγυλοποίηση έχει ως αποτέλεσμα μη κανονικοποίηση του γινομένου, πρέπει να επαναλάβουμε το βήμα 3.

Εντολές κινητής υποδιαστολής στον MIPS

Ο MIPS υποστηρίζει τις μορφές απλής ακρίβειας και διπλής ακρίβειας του προτύπου IEEE 754 με τις παρακάτω εντολές:

- *Πρόσθεση* κινητής υποδιαστολής *απλής ακρίβειας* (`add.s`) και *πρόσθεση διπλής ακρίβειας* (`add.d`)
- *Αφαίρεση* κινητής υποδιαστολής *απλής ακρίβειας* (`sub.s`) και *αφαίρεση διπλής ακρίβειας* (`sub.d`)
- *Πολλαπλασιασμός* κινητής υποδιαστολής *απλής ακρίβειας* (`mul.s`) και *πολλαπλασιασμός διπλής ακρίβειας* (`mul.d`)
- *Διαίρεση* κινητής υποδιαστολής *απλής ακρίβειας* (`div.s`) και *διαίρεση διπλής ακρίβειας* (`div.d`)
- *Σύγκριση* κινητής υποδιαστολής *απλής ακρίβειας* (`c.x.s`) και *σύγκριση διπλής ακρίβειας* (`c.x.d`), όπου το `x` μπορεί να είναι *equal* (ίσο — `eq`), *not equal* (διάφορο — `ne`), *less than* (μικρότερο — `lt`), *less than or equal* (μικρότερο ή ίσο — `le`), *greater than* (μεγαλύτερο — `gt`), ή *greater than or equal* (μεγαλύτερο ή ίσο — `ge`)
- *Διακλάδωση* (branch) κινητής υποδιαστολής *αληθής* (`bc1t`) και *διακλάδωση ψευδής* (`bc1f`)

Η σύγκριση κινητής υποδιαστολής δίνει σε ένα bit τιμή αληθές (true) ή ψευδές (false), ανάλογα με τη συνθήκη σύγκρισης, και στη συνέχεια μια διακλάδωση κινητής υποδιαστολής αποφασίζει αν θα διακλαδώσει ή όχι με βάση τη συνθήκη.

Οι σχεδιαστές του MIPS αποφάσισαν να προσθέσουν ξεχωριστούς καταχωρητές κινητής υποδιαστολής — που ονομάζονται `$f0`, `$f1`, `$f2`, … — οι οποίοι χρησιμοποιούνται είτε για απλή ακρίβεια είτε για διπλή ακρίβεια. Έτσι, συμπεριέλαβαν ξεχωριστές εντολές φόρτωσης (load) και αποθήκευσης (store) για τους καταχωρητές κινητής υποδιαστολής: τις `lwc1` και `swc1` αντίστοιχα. Οι καταχωρητές βάσης για τις μεταφορές δεδομένων κινητής υποδιαστολής παραμένουν ακέραιοι καταχωρητές. Ο κώδικας MIPS για τη φόρτωση δύο αριθμών απλής ακρίβειας από τη μνήμη, τη πρόσθεσή τους, και την αποθήκευση του αθροίσματος μπορεί να μοιάζει με τον παρακάτω:

```
lwc1   $f4,x($sp)  # Φόρτωση του αριθμού 32 bit K.Y. στον F4
lwc1   $f6,y($sp)  # Φόρτωση του αριθμού 32 bit K.Y. στον F6
add.s  $f2,$f4,$f6 # F2 = F4 + F6 απλή ακρίβεια
swc1   $f2,z($sp)  # αποθήκευση του αριθμού 32 bit K.Y. από
                   # τον F2
```

Ένας καταχωρητής διπλής ακρίβειας είναι στη πραγματικότητα ένα ζεύγος καταχωρητών απλής ακρίβειας άρτιου-περιττού, και χρησιμοποιεί τον αριθμό του άρτιου καταχωρητή ως όνομά του.

Η Εικόνα 3.19 συνοψίζει το τμήμα κινητής υποδιαστολής της αρχιτεκτονικής του MIPS που αποκαλύπτεται στο κεφάλαιο αυτό, δείχνοντας με χρώμα τις προσθήκες για την υποστήριξη κινητής υποδιαστολής. Ανάλογα με την Εικόνα 2.25 στη σελίδα 121 του Κεφαλαίου 2, παρουσιάζουμε την κωδικοποίηση αυτών των εντολών στην Εικόνα 3.20.

Τελεστέοι κινητής υποδιαστολής του MIPS

Όνομα	Παράδειγμα	Σχόλια
32 καταχωρητές κινητής υποδιαστολής	`$f0, $f1, $f2, . . . , $f31`	Οι καταχωρητές κινητής υποδιαστολής του MIPS χρησιμοποιούνται ανά ζεύγη για αριθμούς διπλής ακρίβειας.
2^{30} λέξεις μνήμης	Memory[0], Memory[4], . . . , Memory[4294967292]	Προσπελάζονται μόνον από εντολές μεταφοράς δεδομένων. Ο MIPS χρησιμοποιεί διευθύνσεις byte, και έτσι οι διευθύνσεις διαδοχικών λέξεων διαφέρουν κατά 4. Η μνήμη κρατάει δομές δεδομένων, όπως πίνακες και διασκορπισμένους καταχωρητές, όπως αυτοί που αποθηκεύονται στις κλήσεις διαδικασιών.

Συμβολική γλώσσα κινητής υποδιαστολής του MIPS

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Αριθμητικές πράξεις	FP add single	`add.s f2,$f4,$f6`	`$f2 = $f4 + $f6`	Πρόσθεση K.Y (απλή ακρίβεια)
	FP subtract single	`sub.s $f2,$f4,$f6`	`$f2 = $f4 - $f6`	Αφαίρεση K.Y (απλή ακρίβεια)
	FP multiply single	`mul.s $f2,$f4,$f6`	`$f2 = $f4 × $f6`	Πολλαπλασιασμός K.Y (απλή ακρίβεια)
	FP divide single	`div.s $f2,$f4,$f6`	`$f2 = $f4 / $f6`	Διαίρεση K.Y (απλή ακρίβεια)
	FP add double	`add.d $f2,$f4,$f6`	`$f2 = $f4 + $f6`	Πρόσθεση K.Y (διπλή ακρίβεια)
	FP subtract double	`sub.d $f2,$f4,$f6`	`$f2 = $f4 - $f6`	Αφαίρεση K.Y (διπλή ακρίβεια)
	FP multiply double	`mul.d $f2,$f4,$f6`	`$f2 = $f4 × $f6`	Πολλαπλασιασμός K.Y (διπλή ακρίβεια)
	FP divide double	`div.d $f2,$f4,$f6`	`$f2 = $f4 / $f6`	Διαίρεση K.Y (διπλή ακρίβεια)
Μεταφορά δεδομένων	load word copr. 1	`lwc1 $f1,100($s2)`	`$f1 = Memory[$s2 + 100]`	Δεδομένα 32 bit σε καταχωρητή K.Y
	store word copr. 1	`swc1 $f1,100($s2)`	`Memory[$s2 + 100] = $f1`	Δεδομένα 32 bit στη μνήμη
Διακλάδωση υπό συνθήκη	branch on FP true	`bc1t 25`	αν `(cond == 1)` μετάβαση στο PC + 4 + 100	Σχετική ως προς PC διακλάδωση αν ισχύει συνθήκη K.Y
	branch on FP false	`bc1f 25`	αν `(cond == 0)` μετάβαση στο PC + 4 + 100	Σχετική ως προς PC διακλάδωση αν δεν ισχύει συνθήκη K.Y
	FP compare single (eq,ne,lt,le,gt,ge)	`c.lt.s $f2,$f4`	αν `($f2 < $f4)` τότε `cond = 1`· αλλιώς `cond = 0`	Σύγκριση K.Y «μικρότερο από», απλής ακρίβειας
	FP compare double (eq,ne,lt,le,gt,ge)	`c.lt.d $f2,$f4`	αν `($f2 < $f4)` τότε `cond = 1`· αλλιώς `cond = 0`	Σύγκριση K.Y «μικρότερο από», διπλής ακρίβειας

Γλώσσα μηχανής κινητής υποδιαστολής του MIPS

Όνομα	Μορφή	Παράδειγμα						Σχόλια
`add.s`	R	17	16	6	4	2	0	`add.s $f2,$f4,$f6`
`sub.s`	R	17	16	6	4	2	1	`sub.s $f2,$f4,$f6`
`mul.s`	R	17	16	6	4	2	2	`mul.s $f2,$f4,$f6`
`div.s`	R	17	16	6	4	2	3	`div.s $f2,$f4,$f6`
`add.d`	R	17	17	6	4	2	0	`add.d $f2,$f4,$f6`
`sub.d`	R	17	17	6	4	2	1	`sub.d $f2,$f4,$f6`
`mul.d`	R	17	17	6	4	2	2	`mul.d $f2,$f4,$f6`
`div.d`	R	17	17	6	4	2	3	`div.d $f2,$f4,$f6`
`lwc1`	I	49	20	2	100			`lwc1 $f2,100($s4)`
`swc1`	I	57	20	2	100			`swc1 $f2,100($s4)`
`bc1t`	I	17	8	1	25			`bc1t 25`
`bc1f`	I	17	8	0	25			`bc1f 25`
`c.lt.s`	R	17	16	4	2	0	60	`c.lt.s $f2,$f4`
`c.lt.d`	R	17	17	4	2	0	60	`c.lt.d $f2,$f4`
Μέγεθος πεδίου		6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	Όλες οι εντολές του MIPS είναι 32 bit

ΕΙΚΟΝΑ 3.19 Αρχιτεκτονική κινητής υποδιαστολής του MIPS που αποκαλύψαμε μέχρι εδώ. Δείτε την Ενότητα A.10 στο Παράρτημα A, στη σελίδα 117, για περισσότερες λεπτομέρειες.

op(31:26):								
28-26 31-29	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	Rfmt	Bltz/gez	j	jal	beq	bne	blez	bgtz
1(001)	addi	addiu	slti	sltiu	andi	ori	xori	lui
2(010)	TLB	FlPt						
3(011)								
4(100)	lb	lh	lwl	lw	lbu	lhu	lwr	
5(101)	sb	sh	swl	sw			swr	
6(110)	lwc0	lwc1						
7(111)	swc0	swc1						

op(31:26) = 010001 (FlPt), (rt(16:16) = 0 => *c* = f, rt(16:16) = 1 => *c* = t), rs(25:21):								
23-21 25-24	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(00)	mfc1		cfc1		mtc1		ctc1	
1(01)	bc1.*c*							
2(10)	*f* = single	*f* = double						
3(11)								

op(31:26) = 010001 (FlPt), (*f* επάνω: 10000 => *f* = s, 10001 => *f* = d), funct(5:0):								
2-0 5-3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	add.*f*	sub.*f*	mul.*f*	div.*f*		abs.*f*	mov.*f*	neg.*f*
1(001)								
2(010)								
3(011)								
4(100)	cvt.s.*f*	cvt.d.*f*			cvt.w.*f*			
5(101)								
6(110)	c.f.*f*	c.un.*f*	c.eq.*f*	c.ueq.*f*	c.olt.*f*	c.ult.*f*	c.ole.*f*	c.ule.*f*
7(111)	c.sf.*f*	c.ngle.*f*	c.seq.*f*	c.ngl.*f*	c.lt.*f*	c.nge.*f*	c.le.*f*	c.ngt.*f*

ΕΙΚΟΝΑ 3.20 Κωδικοποίηση εντολών κινητής υποδιαστολής MIPS. Αυτή η σημειογραφία δίνει την τιμή ενός πεδίου σε μία γραμμή και μία στήλη. Για παράδειγμα, στο επάνω μέρος της εικόνας η εντολή `lw` βρίσκεται στη γραμμή με αριθμό 4 (100_{two} για τα bit 31-29 της εντολής) και στη στήλη με αριθμό 3 (011_{two} για τα bit 28-26 της εντολής), οπότε η αντίστοιχη τιμή του πεδίου op (bit 31-26) είναι 100011_{two}. Η υπογράμμιση σημαίνει ότι το πεδίο χρησιμοποιείται κάπου αλλού. Για παράδειγμα, η FlPt στη γραμμή 2 στήλη 1 (op = 010001_{two}) ορίζεται στο κάτω μέρος της εικόνας. Έτσι `sub.f` στη γραμμή 0 στήλη 1 του κάτω τμήματος σημαίνει ότι το πεδίο funct (bit 5-0 της εντολής) είναι 000001_{two} και το πεδίο op (bit 31-26) είναι 010001_{two}. Σημειώστε το πεδίο 5 bit rs, που προσδιορίζεται στο μεσαίο τμήμα της εικόνας, καθορίζει αν η λειτουργία είναι απλής ακρίβειας (f = s άρα rs = 10000) ή διπλής ακρίβειας (f = d άρα rs = 10001). Παρόμοια, το bit 16 της εντολής καθορίζει αν η εντολή `bc1.c` ελέγχει για αληθές (bit 16 = 1 => `bc1.t`) ή ψευδές (bit 16 = 0 => `bc1.f`). Οι εντολές με χρώμα περιγράφονται στα Κεφάλαια 2 ή 3, ενώ το Παράρτημα Α καλύπτει όλες τις εντολές.

Διασύνδεση υλικού και λογισμικού

Ένα ζήτημα που αντιμετωπίζουν οι σχεδιαστές υπολογιστών για την υποστήριξη αριθμητικής κινητής υποδιαστολής είναι αν θα χρησιμοποιήσουν τους ίδιους καταχωρητές που χρησιμοποιούνται για τις εντολές ακεραίων ή αν θα προσθέσουν ένα ειδικό σύνολο για την κινητή υποδιαστολή. Επειδή τα προγράμματα φυσιολογικά εκτελούν πράξεις ακεραίων και πράξεις κινητής υποδιαστολής σε διαφορετικά δεδομένα, ο διαχωρισμός των καταχωρητών θα αύξανε μόνο κατά λίγο τον αριθμό των εντολών που χρειάζονται για να εκτελεστεί ένα πρόγραμμα. Η κύρια επίπτωση είναι η δημιουργία ενός ξεχωριστού συνόλου εντολών μεταφοράς δεδομένων, για τη μετακίνηση δεδομένων μεταξύ καταχωρητών κινητής υποδιαστολής και μνήμης.

Τα πλεονεκτήματα των ξεχωριστών καταχωρητών κινητής υποδιαστολής είναι η ύπαρξη διπλάσιων καταχωρητών χωρίς να χρησιμοποιούνται περισσότερα bit στη μορφή της εντολής, η ύπαρξη διπλάσιου εύρους ζώνης με ξεχωριστά σύνολα καταχωρητών ακεραίων και καταχωρητών κινητής υποδιαστολής, και η δυνατότητα προσαρμογής των καταχωρητών στην κινητή υποδιαστολή· για παράδειγμα, κάποιοι υπολογιστές μετατρέπουν τους τελεστέους όλων των μεγεθών, που βρίσκονται σε καταχωρητές, σε μία μόνον εσωτερική μορφή.

Μεταγλώττιση ενός προγράμματος C κινητής υποδιαστολής σε κώδικα συμβολικής γλώσσας του MIPS

ΠΑΡΑΔΕΙΓΜΑ

Ας μετατρέψουμε μια θερμοκρασία από Φαρενάιτ (Fahrenheit) σε βαθμούς Κελσίου (Celsius):

```
float f2c (float fahr)
        {
                return ((5.0/9.0) * (fahr - 32.0));
        }
```

Υποθέστε ότι το όρισμα κινητής υποδιαστολής `fahr` μεταβιβάζεται στον καταχωρητή `$f12`, και το αποτέλεσμα πρέπει να πάει στον `$f0`. (Αντίθετα με τους ακέραιους καταχωρητές, ο καταχωρητής κινητής υποδιαστολής 0 μπορεί να περιέχει έναν αριθμό.) Ποιος είναι ο κώδικας συμβολικής γλώσσας του MIPS;

ΑΠΑΝΤΗΣΗ

Υποθέτουμε ότι ο μεταγλωττιστής τοποθετεί τις τρεις σταθερές κινητής υποδιαστολής στη μνήμη ώστε να είναι προσπελάσιμες από τον καθολικό δείκτη (global pointer) `$gp`. Οι δύο πρώτες εντολές φορτώνουν τις σταθερές 5.0 και 9.0 σε καταχωρητές κινητής υποδιαστολής:

```
f2c:
     lwc1 $f16,const5($gp)   # $f16 = 5.0 (5.0 στη μνήμη)
     lwc1 $f18,const9($gp)   # $f18 = 9.0 (9.0 στη μνήμη)
```

Μετά διαιρούνται για να πάρουμε το κλάσμα 5.0/9.0:

```
div.s $f16, $f16, $f18      # $f16 = 5.0 / 9.0
```

(Πολλοί μεταγλωττιστές θα διαιρούσαν το 5.0 με το 9.0 κατά το χρόνο μεταγλώττισης — compile time — και θα αποθήκευαν μόνο τη σταθερά 5.0/9.0 στη μνήμη, αποφεύγοντας έτσι τη διαίρεση κατά το χρόνο εκτέλεσης — run time.) Στη συνέχεια, φορτώνουμε τη σταθερά 32.0 και μετά την αφαιρούμε από το `fahr` (`$f12`):

```
lwc1 $f18, const32($gp)     # $f18 = 32.0
sub.s $f18, $f12, $f18      # $f18 = fahr - 32.0
```

Τέλος, πολλαπλασιάζουμε τα δύο ενδιάμεσα αποτελέσματα και τοποθετούμε το αποτέλεσμα στον καταχωρητή `$f0` ως επιστρεφόμενο αποτέλεσμα, και μετά επιστρέφουμε:

```
mul.s $f0, $f16, $f18       # $f0 = (5/9)*(fahr - 32.0)
jr $ra        # επιστροφή
```

Τώρα, ας εκτελέσουμε πράξεις κινητής υποδιαστολής σε μήτρες (matrices), με έναν κώδικα που συναντάται συχνά σε επιστημονικά προγράμματα.

ΠΑΡΑΔΕΙΓΜΑ

Μεταγλώττιση διαδικασίας κινητής υποδιαστολής της C με δισδιάστατες μήτρες, σε MIPS

Οι περισσότεροι υπολογισμοί κινητής υποδιαστολής εκτελούνται με διπλή ακρίβεια. Ας εκτελέσουμε πολλαπλασιασμό μητρών X = X + Y * Z. Ας υποθέσουμε ότι οι X, Y, και Z είναι όλες τετραγωνικές μήτρες με 32 στοιχεία σε κάθε διάσταση.

```
void mm (double x[][], double y[][], double z[][])
{
        int i, j, k;

        for (i = 0; i! = 32; i = i + 1)
        for (j = 0; j! = 32; j = j + 1)
        for (k = 0; k! = 32; k = k + 1)
          x[i][j] = x[i][j] + y[i][k] * z[k][j];
}
```

Οι αρχικές διευθύνσεις των πινάκων είναι παράμετροι, άρα βρίσκονται στους καταχωρητές `$a0`, `$a1`, και `$a2`. Υποθέστε ότι οι ακέραιες μεταβλητές βρίσκονται στους `$s0`, `$s1`, και `$s2`, αντίστοιχα. Ποιος είναι ο κώδικας συμβολικής γλώσσας MIPS για το σώμα της διαδικασίας;

ΑΠΑΝΤΗΣΗ

Παρατηρήστε ότι το x[i][j] χρησιμοποιείται στον εσωτερικότερο βρόχο επάνω. Αφού ο αριθμοδείκτης του βρόχου είναι το k, δεν επηρεάζει το x[i][j] και άρα μπορούμε να αποφύγουμε τη φόρτωση και την αποθήκευση του x[i][j] σε κάθε επανάληψη. Αντίθετα, ο μεταγλωττιστής φορτώνει το στοιχείο x[i][j] σε έναν καταχωρητή έξω από το βρόχο, συσσωρεύει το άθροισμα των γινομένων των y[i][k] και z[k][j] στον ίδιο καταχωρητή, και μετά αποθηκεύει το άθροισμα στο x[i][j] κατά τον τερματισμό του εσωτερικότερου βρόχου.

Διατηρούμε τον κώδικα απλούστερο χρησιμοποιώντας τις ψευδοεντολές της συμβολικής γλώσσας li (που φορτώνει μια σταθερά σε έναν καταχωρητή) και l.d και s.d (που ο συμβολομεταφραστής μετατρέπει σε ένα ζεύγος εντολών μεταφοράς δεδομένων, lwc1 ή swc1, σε ένα ζεύγος καταχωρητών κινητής υποδιαστολής).

Το σώμα της διαδικασίας ξεκινά αποθηκεύοντας την τιμή τερματισμού 32 του βρόχου σε έναν προσωρινό καταχωρητή, και μετά δίνοντας αρχικές τιμές στις μεταβλητές των τριών βρόχων *for*:

```
mm:...
      li  $t1, 32 # $t1 = 32 (τέλος μεγέθους γραμμής/βρόχου)
      li  $s0, 0  # i = 0· αρχικές τιμές πρώτου βρόχου for
L1:   li  $s1, 0  # j = 0· αρχικές τιμές δεύτερου βρόχου for
L2:   li  $s2, 0  # k = 0· αρχικές τιμές τρίτου βρόχου for
```

Για να υπολογίσουμε τη διεύθυνση του στοιχείου x[i][j], πρέπει να γνωρίζουμε πώς αποθηκεύεται ένας δισδιάστατος πίνακας 32 × 32 στη μνήμη. Όπως θα αναμένατε, η διάταξή του είναι ίδια σαν να ήταν 32 μονοδιάστατοι πίνακες, καθένας με 32 στοιχεία. Έτσι, το πρώτο βήμα είναι να παρακάμψουμε τους i «μονοδιάστατους πίνακες» ή γραμμές, για να πάμε σε αυτόν που θέλουμε. Επομένως, πολλαπλασιάζουμε τον αριθμοδείκτη της πρώτης διάστασης με το μέγεθος της γραμμής, το 32. Αφού το 32 είναι δύναμη του 2, μπορούμε αντί γι' αυτό να χρησιμοποιήσουμε μια ολίσθηση:

```
sll $t2, $s0, 5 # $t2 = i * 2^5 (μέγεθος γραμμής του x)
```

Τώρα προσθέτουμε το δεύτερο αριθμοδείκτη ώστε να επιλέξουμε το j-οστό στοιχείο της κατάλληλης γραμμής:

```
addu $t2, $t2, $s1      # $t2 = i * μέγεθος(γραμμής) + j
```

Για να μετατρέψουμε αυτό το άθροισμα σε αριθμοδείκτη byte, το πολλαπλασιάζουμε με το μέγεθος ενός στοιχείου της μήτρας σε byte. Αφού κάθε στοιχείο είναι 8 byte για διπλή ακρίβεια, μπορούμε αντί γι' αυτό να εκτελέσουμε αριστερή ολίσθηση κατά 3:

```
sll $t2, $t2, 3 # $t2 = σχετική απόσταση byte του [i][j]
```

Στη συνέχεια προσθέτουμε αυτό το άθροισμα στη διεύθυνση βάσης του x, που δίνει τη διεύθυνση του στοιχείου x[i][j], και μετά φορτώνουμε τον αριθμό διπλής ακρίβειας x[i][j] στον καταχωρητή $f4:

```
addu $t2, $a0, $t2      # $t2 = διεύθυνση byte του x[i][j]
l.d  $f4, 0($t2)        # $f4 = 8 byte του x[i][j]
```

Οι επόμενες πέντε εντολές είναι ουσιαστικά πανομοιότυπες με τις τελευταίες πέντε: υπολογίζουν τη διεύθυνση και μετά φορτώνουν τον αριθμό διπλής ακρίβειας `z[k][j]`.

```
L3: sll  $t0, $s2, 5 # $t0 = k * 2^5 (μέγεθος γραμμής του z)
    addu $t0, $t0, $s1 # $t0 = k * μέγεθος(γραμμής) + j
    sll $t0, $t0, 3 # $t0 = σχετική απόσταση byte του [k][j]
    addu $t0, $a2, $t0 # $t0 = διεύθυνση byte του z[k][j]
    l.d $f16, 0($t0) # $f16 = 8 byte του z[k][j]
```

Παρόμοια, οι επόμενες πέντε εντολές μοιάζουν με τις πέντε τελευταίες: υπολογίζουν τη διεύθυνση και μετά φορτώνουν τον αριθμό διπλής ακρίβειας `y[i][k]`.

```
sll   $t2, $s0, 5 # $t0 = i * 2^5 (μέγεθος γραμμής του y)
addu  $t0, $t0, $s2 # $t0 = i * μέγεθος(γραμμής) + k
sll   $t0, $t0, 3 # $t0 = σχετική διεύθυνση byte του [i][k]
addu  $t0, $a1, $t0 # $t0 = διεύθυνση byte του y[i][k]
l.d   $f18, 0($t0) # $f18 = 8 byte του y[i][k]
```

Τώρα που έχουμε φορτώσει όλα τα δεδομένα, είμαστε έτοιμοι να κάνουμε μερικές πράξεις κινητής υποδιαστολής! Πολλαπλασιάζουμε τα στοιχεία του `y` και του `z` που βρίσκονται στους καταχωρητές `$f18` και `$f16`, και μετά συσσωρεύουμε το άθροισμα στον `$f4`.

```
mul.d $f16, $f18, $f16 # $f16 = y[i][k] * z[k][j]
add.d $f4, $f4, $f16 # f4 = x[i][j]+ y[i][k] * z[k][j]
```

Το τελευταίο τμήμα αυξάνει το δείκτη `k` και επιστρέφει στην αρχή του βρόχου αν ο δείκτης δεν είναι 32. Αν είναι 32, και συνεπώς το τέλος του εσωτερικότερου βρόχου, πρέπει να αποθηκεύσουμε στο στοιχείο `x[i][j]` το άθροισμα που συσσωρεύθηκε στον `$f4`.

```
addiu  $s2, $s2, 1 # $k k + 1
bne    $s2, $t1, L3 # αν (k != 32) μετάβαση στην L3
s.d    $f4, 0($t2) # x[i][j] = $f4
```

Παρόμοια, αυτές οι τελευταίες τέσσερις εντολές αυξάνουν τη μεταβλητή αριθμοδείκτη του μεσαίου και του εξωτερικότερου βρόχου, και επιστρέφουν στην αρχή του βρόχου αν ο δείκτης δεν είναι 32 ενώ προκαλούν έξοδο από το βρόχο αν ο δείκτης είναι 32.

```
addiu   $s1,  $s1, 1 # j = j + 1
bne     $s1,  $t1, L2 # αν (j != 32) μετάβαση στην L2
addiu   $s0,  $s0, 1 # i = i + 1
bne     $s0,  $t1, L1 # αν (i != 32) μετάβαση στην L1
...
```

Επιπλέον ανάπτυξη: Η διάταξη του πίνακα που εξετάσαμε στο παράδειγμα, η οποία ονομάζεται *διάταξη κατά γραμμές* (row major order), χρησιμοποιείται από τη C και πολλές άλλες γλώσσες προγραμματισμού. Η Fortran αντίθετα χρησιμοποιεί *διάταξη κατά στήλες* (column major order), όπου ο πίνακας αποθηκεύεται στήλη προς στήλη.

Στην αρχή μπορούσαν να χρησιμοποιηθούν μόνο 16 από τους 32 καταχωρητές κινητής υποδιαστολής του MIPS για πράξεις απλής ακρίβειας: οι `$f0`, `$f2`, `$f4`, ..., `$f30`. Η διπλή ακρίβεια υπολογίζεται με τη χρήση ζευγών αυτών των καταχωρητών. Οι καταχωρητές κινητής υποδιαστολής με περιττό αριθμό χρησιμοποιούνταν μόνο για τη φόρτωση και την αποθήκευση του δεξιού μισού των αριθμών κινητής υποδιαστολής 64 bit. Η αρχιτεκτονική MIPS-32 πρόσθεσε τις εντολές `l.d` και `s.d` στο σύνολο

εντολών. Η MIPS-32 πρόσθεσε επίσης εκδόσεις όλων των εντολών κινητής υποδιαστολής «σε ζεύγη απλής ακρίβειας» (paired single versions) όπου μια εντολή έχει ως αποτέλεσμα δύο παράλληλες πράξεις κινητής υποδιαστολής σε δύο τελεστέους των 32 bit μέσα σε καταχωρητές των 64 bit. Για παράδειγμα, η `add.ps F0, F2, F4` είναι ισοδύναμη με την `add.s F0, F2, F4` ακολουθούμενη από την `add.s F1, F3, F5`.

Ένας άλλος λόγος για τους ξεχωριστούς καταχωρητές ακεραίων και κινητής υποδιαστολής είναι ότι οι μικροεπεξεργαστές τη δεκαετία του 1980 δεν είχαν αρκετά τρανζίστορ για να βάλουν τη μονάδα αριθμών κινητής υποδιαστολής στο ίδιο ολοκληρωμένο κύκλωμα με τη μονάδα ακεραίων. Έτσι, η μονάδα κινητής υποδιαστολής, μαζί με τους καταχωρητές κινητής υποδιαστολής, ήταν διαθέσιμη προαιρετικά ως ένα δεύτερο τσιπ. Τέτοια προαιρετικά τσιπ επιτάχυνσης (accelerator chips) ονομάζονται *συνεπεξεργαστές* (coprocessors), και εξηγούν το ακρωνύμιο των εντολών φόρτωσης κινητής υποδιαστολής του MIPS: `lwc1` σημαίνει load word to coprocessor 1 (φόρτωσε τη λέξη στο συνεπεξεργαστή 1, τη μονάδα κινητής υποδιαστολής). (Ο συνεπεξεργαστής 0 ασχολείται με την εικονική μνήμη — virtual memory, περιγράφεται στο Κεφάλαιο 7.) Από τις αρχές της δεκαετίας του 1990, οι μικροεπεξεργαστές έχουν ολοκληρωμένες μονάδες κινητής υποδιαστολής (και σχεδόν οτιδήποτε άλλο) επάνω στο ολοκληρωμένο κύκλωμα, και έτσι ο όρος «συνεπεξεργαστής» πήγε να συναντήσει τους όρους «συσσωρευτής» (accumulator) και «μνήμη πυρήνων» (core memory) ως απαρχαιωμένος όρος που προδίδει την ηλικία του ομιλητή.

Επιπλέον ανάπτυξη: Παρόλο που υπάρχουν πολλοί τρόποι να «ρίξουμε» υλικό στον πολλαπλασιασμό κινητής υποδιαστολής ώστε να τον κάνουμε πιο γρήγορο, η διαίρεση κινητής υποδιαστολής παρουσιάζει σημαντικά περισσότερες προκλήσεις για να γίνει γρήγορη και ακριβής. Οι αργές διαιρέσεις στους πρώτους υπολογιστές οδήγησαν στην αφαίρεση των διαιρέσεων από πολλούς αλγορίθμους, αλλά οι παράλληλοι υπολογιστές έχουν εμπνεύσει τη νέα ανακάλυψη αλγορίθμων με πολλές διαιρέσεις που δουλεύουν καλύτερα σε αυτούς. Έτσι, ίσως χρειαστούμε ταχύτερες διαιρέσεις.

Μια τεχνική για την αξιοποίηση ενός γρήγορου πολλαπλασιαστή είναι η *επανάληψη του Newton* (Newton's iteration), όπου η διαίρεση επανορίζεται ως η εύρεση του μηδενός μιας συνάρτησης για τον υπολογισμό της αντίστροφης $1/x$, η οποία μετά πολλαπλασιάζεται με τον άλλο τελεστέο. Τεχνικές επανάληψης *δεν μπορούν* να στρογγυλοποιηθούν κατάλληλα χωρίς τον υπολογισμό πολλών επιπλέον bit. Ένα ολοκληρωμένο κύκλωμα της Texas Instruments λύνει αυτό το πρόβλημα υπολογίζοντας έναν αντίστροφο με πρόσθετη ακρίβεια.

Επιπλέον ανάπτυξη: Η Java υιοθετεί το πρότυπο IEEE 754 με το όνομά του στον ορισμό της για τους τύπους δεδομένων και τις πράξεις κινητής υποδιαστολής. Έτσι, ο κώδικας στο πρώτο παράδειγμα θα μπορούσε να έχει παραχθεί για μια μέθοδο κλάσης (τάξης) που μετατρέπει βαθμούς Φαρενάιτ σε Κελσίου.

Το δεύτερο παράδειγμα χρησιμοποιεί πίνακες πολλών διαστάσεων, οι οποίοι δεν υποστηρίζονται απευθείας από την Java. Η Java επιτρέπει πίνακες πινάκων (arrays of arrays), αλλά κάθε πίνακας πρέπει να έχει το δικό του μήκος, αντίθετα με τους πίνακες πολλών διαστάσεων της C. Όπως στα παραδείγματα του Κεφαλαίου 2, μια έκδοση Java αυτού του δεύτερου παραδείγματος θα χρειαζόταν εκτεταμένο έλεγχο ορίων των πινάκων, μεταξύ των οποίων και ένας νέος υπολογισμός μήκους στο τέλος της γραμμής. Θα έπρεπε επίσης να ελεγχθεί ότι η αναφορά αντικειμένου δεν είναι μηδενική (null).

Ακριβής αριθμητική

Αντίθετα με τους ακεραίους, οι οποίοι μπορούν να αναπαραστήσουν ακριβώς κάθε αριθμό ανάμεσα στο μέγιστο και στον ελάχιστο, οι αριθμοί κινητής υποδιαστολής είναι φυσιολογικά προσεγγίσεις ενός αριθμού που δεν μπορούν να αναπαραστήσουν πραγματικά. Ο λόγος είναι ότι υπάρχει μια άπειρη ποικιλία πραγματικών αριθμών μεταξύ, ας πούμε, του 0 και του 1, αλλά δεν μπορούν να

αναπαρασταθούν περισσότεροι από 2^{53} ακριβώς, σε διπλή ακρίβεια κινητής υποδιαστολής. Το καλύτερο που μπορούμε να κάνουμε είναι να πάρουμε μια αναπαράσταση κινητής υποδιαστολής κοντά στον πραγματικό αριθμό. Έτσι, το πρότυπο IEEE 754 παρέχει πολλούς τρόπους στρογγυλοποίησης για να επιτρέψει στον προγραμματιστή να διαλέξει την κατάλληλη προσέγγιση.

Η στρογγυλοποίηση ακούγεται αρκετά απλή, αλλά η στρογγυλοποίηση με ακρίβεια απαιτεί το υλικό να περιλαμβάνει επιπλέον bit στον υπολογισμό. Στα προηγούμενα παραδείγματα, ήμαστε ασαφείς για τον αριθμό των bit που μπορεί να καταλαμβάνει μια ενδιάμεση αναπαράσταση αλλά είναι σαφές πως, αν κάθε ενδιάμεσο αποτέλεσμα έπρεπε να περικοπεί στον ακριβή αριθμό ψηφίων, δε θα υπήρχε ευκαιρία για στρογγυλοποίηση. Το πρότυπο IEEE 754, συνεπώς, κρατάει πάντα 2 επιπλέον bit στα δεξιά κατά τη διάρκεια ενδιαμέσων προσθέσεων, που ονομάζονται **φρουρός** (guard) και **στρογγύλευση** (round), αντίστοιχα. Ας δούμε ένα δεκαδικό παράδειγμα για να δείξουμε την αξία αυτών των επιπλέον ψηφίων.

φρουρός (guard) Το πρώτο από δύο επιπλέον bit που κρατούνται στα δεξιά κατά τη διάρκεια ενδιαμέσων υπολογισμών αριθμών κινητής υποδιαστολής· χρησιμοποιείται για τη βελτίωση της ακρίβειας της στρογγυλοποίησης.

στρογγύλευση (round) Μέθοδος για να κάνουμε ένα ενδιάμεσο αποτέλεσμα κινητής υποδιαστολής να χωρέσει στη μορφή κινητής υποδιαστολής· ο τυπικός στόχος είναι να βρούμε τον πλησιέστερο αριθμό που μπορεί να αναπαρασταθεί από τη μορφή.

Στρογγυλοποίηση με ψηφία-φρουρούς (guard digits)

ΠΑΡΑΔΕΙΓΜΑ

Προσθέστε το $2{,}56_{ten} \times 10^0$ στο $2{,}34_{ten} \times 10^2$, υποθέτοντας ότι έχουμε τρία σημαντικά δεκαδικά ψηφία. Στρογγυλοποιήστε στον πλησιέστερο δεκαδικό αριθμό με τρία σημαντικά δεκαδικά ψηφία, πρώτα με φρουρό (guard) και στρογγύλευση (round) και μετά χωρίς αυτά.

ΑΠΑΝΤΗΣΗ

Πρώτα πρέπει να ολισθήσουμε το μικρότερο αριθμό προς τα δεξιά για να ευθυγραμμίσουμε τους εκθέτες, άρα ο $2{,}56_{ten} \times 10^0$ γίνεται $0{,}0256_{ten} \times 10^2$. Αφού έχουμε ψηφία guard και round, έχουμε τη δυνατότητα να αναπαραστήσουμε τα δύο λιγότερο σημαντικά ψηφία όταν ευθυγραμμίζουμε τους εκθέτες. Το ψηφίο guard κρατάει το 5 και το ψηφίο round το 6. Το άθροισμα είναι

$$\begin{array}{r} 2{,}3400_{ten} \\ +\ \underline{0{,}0256_{ten}} \\ 2{,}3656_{ten} \end{array}$$

Έτσι το άθροισμα είναι $2{,}3656_{ten} \times 10^2$. Επειδή έχουμε δύο ψηφία για να στρογγυλοποιήσουμε, θέλουμε οι τιμές 0 μέχρι 49 να στρογγυλοποιούνται προς τα κάτω, και οι τιμές 51 μέχρι 99 να στρογγυλοποιούνται προς τα άνω, με το 50 να είναι η λύση της ισοπαλίας. Η στρογγυλοποίηση του αθροίσματος προς τα άνω με τρία σημαντικά ψηφία δίνει $2{,}37_{ten} \times 10^2$.

Η μέθοδος αυτή *χωρίς* τα ψηφία guard και round «πετάει» δύο ψηφία από τον υπολογισμό. Το νέο άθροισμα είναι τότε

$$\begin{array}{r} 2{,}34_{ten} \\ +\ \underline{0{,}02_{ten}} \\ 2{,}36_{ten} \end{array}$$

Η λύση είναι $2{,}36_{ten} \times 10^2$, που διαφέρει κατά 1 στο τελευταίο ψηφίο από το παραπάνω άθροισμα.

Επειδή η χειρότερη περίπτωση στη στρογγυλοποίηση είναι όταν ο πραγματικός αριθμός είναι ακριβώς στη μέση της απόστασης ανάμεσα σε δύο αναπαραστάσεις κινητής υποδιαστολής, η ακρίβεια κινητής υποδιαστολής φυσιολογικά μετράται με βάση τον αριθμό των λανθασμένων bit στα λιγότερο σημαντικά bit του σημαντικού (significand)· η μέτρηση ονομάζεται αριθμός **μονάδων στην τελευταία θέση** (units in the last place, ή **ulp**). Αν ένας αριθμός διαφέρει σε 2 από τα λιγότερο σημαντικά bit, λέγεται ότι διαφέρει κατά 2 ulp. Δεδομένου ότι δεν υπάρχει υπερχείλιση, ανεπάρκεια, ή εξαιρέσεις μη επιτρεπτών πράξεων, το IEEE 754 εγγυάται ότι ο υπολογιστής χρησιμοποιεί τον αριθμό που βρίσκεται μέσα στο εύρος μισού ulp.

μονάδες στην τελευταία θέση (units in the last place — ulp) Το πλήθος των λανθασμένων bit στα λιγότερο σημαντικά bit του σημαντικού, ανάμεσα στον πραγματικό αριθμό και τον αριθμό που μπορεί να αναπαρασταθεί.

Επιπλέον ανάπτυξη: Παρόλο που το παραπάνω παράδειγμα χρειαζόταν μόνον ένα επιπλέον ψηφίο, ο πολλαπλασιασμός μπορεί να χρειάζεται δύο. Ένα δυαδικό γινόμενο μπορεί να έχει ένα αρχικό bit 0· έτσι, το βήμα στρογγυλοποίησης πρέπει να ολισθήσει το γινόμενο 1 bit αριστερά. Αυτό ολισθαίνει το ψηφίο-φρουρό (guard digit) στο λιγότερο σημαντικό bit του γινομένου, αφήνοντας το ψηφίο στρογγύλευσης (round digit) να βοηθήσει στην ακριβή στρογγυλοποίηση του γινομένου.

Υπάρχουν τέσσερις τρόποι στρογγυλοποίησης: πάντα στρογγυλοποίηση προς τα άνω (round up, προς το $+\infty$), πάντα στρογγυλοποίηση προς τα κάτω (round down, προς το $-\infty$), αποκοπή (truncate), και στρογγυλοποίηση προς τον πλησιέστερο άρτιο (round to nearest even). Ο τελευταίος τρόπος καθορίζει τι θα γίνει αν ο αριθμός βρίσκεται ακριβώς στη μέση της απόστασης ανάμεσα σε δύο αναπαραστάσεις. Η Εφορία πάντα στρογγυλοποιεί το 0,50 ευρώ προς τα άνω, πιθανόν για όφελός της. Ένας πιο δίκαιος τρόπος θα ήταν να το στρογγυλοποιεί στις μισές περιπτώσεις προς τα άνω και στις μισές προς τα κάτω. Το IEEE 754 λέει ότι, αν το λιγότερο σημαντικό ψηφίο που διατηρείται σε μια περίπτωση όπου το αποτέλεσμα βρίσκεται στο μέσο της απόστασης είναι περιττό, προστίθεται ένα· αν είναι άρτιο, γίνεται αποκοπή. Αυτή η μέθοδος δημιουργεί πάντα ένα 0 στο λιγότερο σημαντικό bit στην ισόπαλη περίπτωση, δίνοντας γι' αυτόν το λόγο το όνομα του τρόπου στρογγυλοποίησης. Αυτός ο τρόπος είναι αυτός που χρησιμοποιείται πιο συχνά και ο μόνος που υποστηρίζει η Java.

Ο στόχος των επιπλέον bit στρογγυλοποίησης είναι να επιτρέψουν στον υπολογιστή να πάρει τα ίδια αποτελέσματα σαν να υπολογίζονταν τα ενδιάμεσα αποτελέσματα με άπειρη ακρίβεια και μετά να γινόταν στρογγυλοποίηση.

Για να υποστηρίξει αυτόν το στόχο και τη στρογγυλοποίηση στον πλησιέστερο άρτιο, το πρότυπο διαθέτει ένα τρίτο bit επιπλέον του φρουρού (guard) και της στρογγύλευσης (round)· αυτό το bit παίρνει τιμή 1 όποτε υπάρχουν μη μηδενικά bit στα δεξιά του bit στρογγύλευσης. Αυτό το **επίμονο bit** (sticky bit) επιτρέπει στον υπολογιστή να βλέπει τη διαφορά ανάμεσα στο $0{,}50\ldots00_{ten}$ και το $0{,}50\ldots01_{ten}$ όταν στρογγυλοποιεί.

επίμονο bit (sticky bit) Ένα bit που χρησιμοποιείται στη στρογγυλοποίηση επιπλέον του φρουρού (guard) και της στρογγύλευσης (round) και παίρνει τιμή 1 όποτε υπάρχουν μη μηδενικά bit στα δεξιά του bit στρογγύλευσης.

Το επίμονο bit μπορεί να πάρει τιμή 1, για παράδειγμα, κατά τη πρόσθεση, όταν ο μικρότερος αριθμός ολισθαίνει προς τα δεξιά. Υποθέστε ότι στο παραπάνω παράδειγμα προσθέσαμε το $5{,}01_{ten} \times 10^{-1}$ στο $2{,}34_{ten} \times 10^{2}$. Ακόμη και με φρουρό και στρογγύλευση, θα προσθέταμε το 0,0050 στο 2,34, με άθροισμα 2,3450. Το επίμονο bit θα είχε πάρει τιμή 1 αφού υπάρχουν μη μηδενικά ψηφία στα δεξιά. Χωρίς το επίμονο bit, για να θυμόμαστε αν κάποια ψηφία 1 ολίσθησαν εκτός, θα υποθέταμε ότι ο αριθμός είναι ίσος με 2,345000...00 και θα στρογγυλοποιούσαμε στον πλησιέστερο άρτιο 2,34. Αντίθετα, με το επίμονο bit να θυμάται ότι ο αριθμός είναι μεγαλύτερος από 2,345000...00, θα στρογγυλοποιούσαμε στο 2,35.

Περίληψη

Η ενότητα «Συνολική εικόνα» παρακάτω ενισχύει τη έννοια του αποθηκευμένου προγράμματος που γνωρίσαμε στο Κεφάλαιο 2· δεν μπορούμε να προσδιορίσουμε την έννοια των πληροφοριών βλέποντας τα bit, επειδή τα ίδια bit μπορεί να αναπαριστούν μια ποικιλία αντικειμένων. Αυτή η ενότητα δείχνει ότι η

αριθμητική υπολογιστών είναι πεπερασμένη και, συνεπώς, μπορεί να μη συμφωνεί με τη φυσική αριθμητική. Για παράδειγμα, η αναπαράσταση κινητής υποδιαστολής του προτύπου IEEE 754

$$(-1)^{\text{πρόσημο}} \times (1 + \text{κλάσμα}) \times 2^{(\text{εκθέτης} - \text{πόλωση})}$$

είναι σχεδόν πάντα μια προσέγγιση του πραγματικού αριθμού. Τα υπολογιστικά συστήματα πρέπει να είναι προσεκτικά ώστε να ελαχιστοποιούν το χάσμα ανάμεσα στην αριθμητική υπολογιστών και την αριθμητική του πραγματικού κόσμου, και οι προγραμματιστές μερικές φορές πρέπει να γνωρίζουν τις συνέπειες αυτής της προσέγγισης.

Τύπος C	Τύπος Java	Μεταφορές δεδομένων	Λειτουργίες
int	int	lw, sw, lui	addu, addiu, subu, mult, div, and, andi, or, ori, nor, slt, slti
unsigned int	——	lw, sw, lui	addu, addiu, subu, multu, divu, and, andi, or, ori, nor, sltu, sltiu
char	——	lb, sb, lui	addu, addiu, subu, multu, divu, and, andi, or, ori, nor, sltu, sltiu
——	char	lh, sh, lui	addu, addiu, subu, multu, divu, and, andi, or, ori, nor, sltu, sltiu
float	float	lwc1, swc1	add.s, sub.s, mult.s, div.s, c.eq.s, c.lt.s, c.le.s
double	double	l.d, s.d	add.d, sub.d, mult.d, div.d, c.eq.d, c.lt.d, c.le.d

Η ΣΥΝΟΛΙΚΗ εικόνα

Οι σειρές των bit δεν έχουν κανένα εσωτερικό νόημα. Μπορεί να αντιπροσωπεύουν προσημασμένους αριθμούς, απρόσημους αριθμούς, αριθμούς κινητής υποδιαστολής, εντολές, και ούτω καθεξής. Αυτό που αναπαρίσταται εξαρτάται από την εντολή που επενεργεί στα bit της λέξης.

Η βασική διαφορά ανάμεσα στους αριθμούς των υπολογιστών και τους αριθμούς στον πραγματικό κόσμο είναι ότι οι αριθμοί των υπολογιστών έχουν περιορισμένο μέγεθος και, συνεπώς, περιορισμένη ακρίβεια· είναι δυνατόν να υπολογιστεί ένας αριθμός υπερβολικά μεγάλος ή υπερβολικά μικρός για να αναπαρασταθεί σε μία λέξη. Οι προγραμματιστές πρέπει να θυμούνται αυτά τα όρια και να γράφουν προγράμματα κατάλληλα.

Διασύνδεση υλικού και λογισμικού

Στο προηγούμενο κεφάλαιο παρουσιάσαμε τις τάξεις (κλάσεις) αποθήκευσης (storage classes) της γλώσσας προγραμματισμού C (δείτε την ενότητα «Διασύνδεση υλικού και λογισμικού» στη σελίδα 103). Ο παραπάνω πίνακας δείχνει μερικούς από τους τύπους δεδομένων της C και της Java, μαζί με τις εντολές μεταφοράς δεδομένων του MIPS και τις εντολές που επενεργούν σε αυτούς τους τύπους, οι οποίες εμφανίζονται στα Κεφάλαια 2 και 3. Σημειώστε ότι η Java δεν περιλαμβάνει απρόσημους αριθμούς.

Αυτοεξέταση

Υποθέστε ότι υπάρχει μια μορφή κινητής υποδιαστολής 16 bit του IEEE 754 με εκθέτη των 5 bit. Ποιο θα ήταν το πιθανό εύρος αριθμών που θα μπορούσε να αναπαραστήσει;

1.	$1{,}000\ 000\ 00 \times 2^{0}$	έως	$1{,}111\ 111\ 11 \times 2^{31}$,	0
2.	$\pm 1{,}000\ 000\ 0 \times 2^{-14}$	έως	$\pm 1{,}111\ 111\ 1 \times 2^{15}$,	$\pm 0, \pm\infty$, NaN
3.	$\pm 1{,}000\ 000\ 00 \times 2^{-14}$	έως	$\pm 1{,}111\ 111\ 11 \times 2^{15}$,	$\pm 0, \pm\infty$, NaN
4.	$\pm 1{,}000\ 000\ 00 \times 2^{-15}$	έως	$\pm 1{,}111\ 111\ 11 \times 2^{14}$,	$\pm 0, \pm\infty$, NaN

Επιπλέον ανάπτυξη: Για να εξυπηρετήσει συγκρίσεις που μπορεί να περιλαμβάνουν NaN, το πρότυπο περιλαμβάνει τη *διατεταγμένη* (ordered) και τη *μη διατεταγμένη* (unordered) ως επιλογές για τις συγκρίσεις. Έτσι, το πλήρες σύνολο εντολών του MIPS έχει πολλές παραλλαγές συγκρίσεων για να υποστηρίξει τους «μη αριθμούς» NaN. (Η Java δεν υποστηρίζει μη διατεταγμένες συγκρίσεις.)

Σε μια προσπάθεια να «στραγγίξει» και το τελευταίο bit ακρίβειας από μια πράξη κινητής υποδιαστολής, το πρότυπο επιτρέπει σε κάποιους αριθμούς να αναπαρίστανται σε μη κανονικοποιημένη μορφή. Αντί να έχει ένα κενό ανάμεσα στο 0 και στο μικρότερο κανονικοποιημένο αριθμό, το IEEE επιτρέπει *μη κανονικοποιημένους* (denormalized, denorm) *αριθμούς* (επίσης γνωστούς ως *υποκανονικούς* — subnormals). Έχουν τον ίδιο εκθέτη όπως το μηδέν αλλά μη μηδενικό σημαντικό. Επιτρέπουν σε έναν αριθμό να εκφυλίζεται σε σημαντικότητα μέχρι να γίνει 0, πράγμα που ονομάζεται *βαθμιαία ανεπάρκεια* (gradual underflow). Για παράδειγμα, ο μικρότερος θετικός κανονικοποιημένος αριθμός απλής ακρίβειας είναι

$$1{,}0000\ 0000\ 0000\ 0000\ 0000\ 000_{two} \times 2^{-126}$$

αλλά ο μικρότερος μη κανονικοποιημένος αριθμός απλής ακρίβειας είναι

$$0{,}0000\ 0000\ 0000\ 0000\ 0000\ 001_{two} \times 2^{-126}, \text{ ή } 1{,}0_{two} \times 2^{-149}$$

Για διπλή ακρίβεια, το κενό των μη κανονικοποιημένων αριθμών αλλάζει από $1{,}0 \times 2^{-1022}$ σε $1{,}0 \times 2^{-1074}$.

Η πιθανότητα ενός περιστασιακού μη κανονικοποιημένου τελεστέου έχει φέρει πονοκεφάλους στους σχεδιαστές κινητής υποδιαστολής οι οποίοι προσπαθούν να κατασκευάσουν γρήγορες μονάδες κινητής υποδιαστολής. Έτσι, πολλοί υπολογιστές προκαλούν μια εξαίρεση αν ένας τελεστέος είναι μη κανονικοποιημένος, αφήνοντας το λογισμικό να ολοκληρώσει τη λειτουργία. Παρόλο που οι υλοποιήσεις με λογισμικό είναι απολύτως έγκυρες, η χαμηλότερη απόδοσή τους έχει μειώσει τη δημοτικότητα των μη κανονικοποιημένων αριθμών στο φορητό λογισμικό κινητής υποδιαστολής. Επίσης, αν οι προγραμματιστές δεν αναμένουν μη κανονικοποιημένους αριθμούς, τα προγράμματά τους μπορεί να αιφνιδιαστούν.

3.7 Πραγματικότητα: κινητή υποδιαστολή στην αρχιτεκτονική IA-32

Η αρχιτεκτονική IA-32 διαθέτει κανονικές εντολές πολλαπλασιασμού και διαίρεσης που λειτουργούν πλήρως σε καταχωρητές, αντίθετα με τον MIPS που βασίζεται στους Hi και Lo. (Για την ακρίβεια, σε επόμενες εκδόσεις του συνόλου εντολών MIPS προστέθηκαν παρόμοιες εντολές.)

Οι βασικές διαφορές συναντώνται στις εντολές κινητής υποδιαστολής. Η αρχιτεκτονική κινητής υποδιαστολής IA-32 είναι διαφορετική από όλους τους άλλους υπολογιστές στον κόσμο.

Η αρχιτεκτονική κινητής υποδιαστολής IA-32

Ο συνεπεξεργαστής κινητής υποδιαστολής Intel 8087 ανακοινώθηκε το 1980. Αυτή η αρχιτεκτονική επέκτεινε τον 8086 με περίπου 60 εντολές κινητής υποδιαστολής.

Με τις εντολές κινητής υποδιαστολής της, η Intel παρείχε μια αρχιτεκτονική στοίβας (stack architecture): οι φορτώσεις (loads) τοποθετούν αριθμούς στη στοίβα, οι πράξεις (λειτουργίες) βρίσκουν τους τελεστέους στα δύο κορυφαία στοιχεία των στοιβών, και οι αποθηκεύσεις (stores) μπορούν να εξαγάγουν στοιχεία από τη στοίβα. Η Intel συμπλήρωσε αυτή την αρχιτεκτονική στοίβας με εντολές και τρόπους διευθυνσιοδότησης που επιτρέπουν στην αρχιτεκτονική να έχει μερικά από τα οφέλη ενός μοντέλου καταχωρητή-μνήμης (register-memory model). Εκτός από την εύρεση τελεστέων στα δύο κορυφαία στοιχεία της στοίβας, ένας τελεστέος μπορεί να βρίσκεται στη μνήμη ή σε έναν από τους επτά καταχωρητές του ολοκληρωμένου κυκλώματος κάτω από την κορυφή της στοίβας. Έτσι, ένα πλήρες σύνολο εντολών στοίβας συμπληρώνεται από ένα περιορισμένο σύνολο εντολών καταχωρητή-μνήμης.

Ωστόσο, αυτό το υβρίδιο είναι ακόμη ένα περιορισμένο μοντέλο καταχωρητή-μνήμης, επειδή οι φορτώσεις μεταφέρουν πάντα δεδομένα στην κορυφή της στοίβας ενώ αυξάνουν το δείκτη κορυφής της στοίβας, και οι αποθηκεύσεις μπορούν μόνο να μεταφέρουν την κορυφή της στοίβας στη μνήμη. Η Intel χρησιμοποιεί τη σημειογραφία `ST` για να επισημάνει την κορυφή της στοίβας και την `ST(i)` για να αναπαραστήσει τον *i*-οστό καταχωρητή κάτω από την κορυφή της στοίβας.

Ένα άλλο πρωτότυπο χαρακτηριστικό αυτής της αρχιτεκτονικής είναι ότι οι τελεστέοι έχουν μεγαλύτερο εύρος στη στοίβα καταχωρητών από ό,τι όταν αποθηκεύονται στη μνήμη, και όλες οι λειτουργίες εκτελούνται με αυτή την ευρύτερη εσωτερική ακρίβεια. Αντίθετα με το μέγιστο των 64 bit του MIPS, οι τελεστέοι κινητής υποδιαστολής της IA-32 στη στοίβα έχουν εύρος 80 bit. Οι αριθμοί μετατρέπονται αυτόματα στην εσωτερική μορφή των 80 bit σε μια φόρτωση, και μετατρέπονται ξανά στο κατάλληλο μέγεθος σε μια αποθήκευση. Αυτή η *επεκτεταμένη διπλή ακρίβεια* (double extended precision) δεν υποστηρίζεται από τις γλώσσες προγραμματισμού, παρόλο που είναι χρήσιμη στους προγραμματιστές μαθηματικού λογισμικού.

Τα δεδομένα της μνήμης μπορούν να είναι αριθμοί κινητής υποδιαστολής των 32 bit (απλή ακρίβεια) ή των 64 bit (διπλή ακρίβεια). Η έκδοση καταχωρητή-μνήμης αυτών των εντολών μετατρέπει τον τελεστέο μνήμης σε αυτή τη μορφή των 80 bit της Intel πριν εκτελέσει την πράξη (λειτουργία). Οι εντολές μεταφοράς δεδομένων μετατρέπουν επίσης αυτόματα τους ακεραίους των 16 και 32 bit σε αριθμούς κινητής υποδιαστολής και αντίστροφα, για φορτώσεις και αποθηκεύσεις ακεραίων.

Οι πράξεις κινητής υποδιαστολής της IA-32 μπορούν να διαιρεθούν σε τέσσερις κύριες κατηγορίες:

1. Εντολές μετακίνησης δεδομένων, μεταξύ των οποίων φόρτωση (load), φόρτωση σταθεράς (load constant), και αποθήκευση (store)
2. Αριθμητικές εντολές, μεταξύ των οποίων πρόσθεση (add), αφαίρεση (subtract), πολλαπλασιασμός (multiply), διαίρεση (divide), τετραγωνική ρίζα (square root), και απόλυτη τιμή (absolute value)
3. Σύγκρισης, μεταξύ των οποίων εντολές που στέλνουν το αποτέλεσμα στον επεξεργαστή ακεραίων ώστε να μπορεί να εκτελέσει διακλάδωση

Μεταφορά δεδομένων	Αριθμητικές πράξεις	Σύγκριση	Τριγωνομετρικές
F{I}LD mem/ST(i)	F{I}ADD{P} mem/ST(i)	F{I}COM{P}	FPATAN
F{I}ST{P} mem/ST(i)	F{I}SUB{R}{P} mem/ST(i)	F{I}UCOM{P}{P}	F2XM1
FLDPI	F{I}MUL{P} mem/ST(i)	FSTSW AX/mem	FCOS
FLD1	F{I}DIV{R}{P} mem/ST(i)		FPTAN
FLDZ	FSQRT		FPREM
	FABS		FSIN
	FRNDINT		FYL2X

ΕΙΚΟΝΑ 3.21 Οι εντολές κινητής υποδιαστολής της αρχιτεκτονικής IA-32. Χρησιμοποιούμε τα άγκιστρα {} για να δείξουμε προαιρετικές παραλλαγές των βασικών πράξεων (λειτουργιών): {I} σημαίνει ότι υπάρχει ακέραια έκδοση της εντολής, {P} σημαίνει ότι αυτή η παραλλαγή εξάγει έναν τελεστέο από τη στοίβα μετά τη λειτουργία, και {R} σημαίνει αντιστροφή της σειράς των τελεστέων σε αυτή τη λειτουργία. Η πρώτη στήλη δείχνει τις εντολές μεταφοράς δεδομένων, οι οποίες μετακινούν δεδομένα στη μνήμη ή σε έναν από τους καταχωρητές κάτω από την κορυφή της στοίβας. Οι τρεις τελευταίες λειτουργίες στην πρώτη στήλη τοποθετούν σταθερές στη στοίβα: το π, το 1,0, και το 0,0. Η δεύτερη στήλη περιέχει τις αριθμητικές πράξεις που περιγράψαμε παραπάνω. Σημειώστε ότι οι τελευταίες τρεις λειτουργούν μόνο στην κορυφή της στοίβας. Η τρίτη στήλη είναι οι εντολές σύγκρισης. Επειδή δεν υπάρχουν ειδικές εντολές διακλάδωσης κινητής υποδιαστολής, το αποτέλεσμα της σύγκρισης πρέπει να μεταφερθεί στη CPU χειρισμού ακεραίων μέσω της εντολής FSTSW, είτε στον καταχωρητή AX είτε στη μνήμη, και να ακολουθήσει μια εντολή SAHF για να οριστούν οι κωδικοί συνθήκης (condition codes). Η σύγκριση κινητής υποδιαστολής μπορεί τότε να ελεγχθεί με τη χρήση εντολών διακλάδωσης ακεραίων. Η τελευταία στήλη δίνει τις πράξεις κινητής υποδιαστολής υψηλότερου επιπέδου. Δεν παρέχονται όλοι οι συνδυασμοί που υπονοούνται από τη σημειογραφία. Έτσι, οι λειτουργίες F{I}SUB{R}{P} αναπαριστούν τις εντολές που συναντώνται στην αρχιτεκτονική IA-32: FSUB, FISUB, FSUBR, FISUBR, FSUBP, FSUBRP. Για τις εντολές αφαίρεσης ακεραίων δεν υπάρχει εξαγωγή από τη στοίβα (FISUBP) ή αντίστροφη εξαγωγή από τη στοίβα (FISUBRP).

Εντολή	Τελεστέοι	Σχόλιο
FADD		Και οι δύο τελεστέοι στη στοίβα· το αποτέλεσμα αντικαθιστά την κορυφή της στοίβας.
FADD	ST(i)	Ένας τελεστέος προέλευσης είναι ο *i*-οστός καταχωρητής κάτω από την κορυφή της στοίβας· το αποτέλεσμα αντικαθιστά την κορυφή της στοίβας.
FADD	ST(i), ST	Ένας τελεστέος προέλευσης είναι η κορυφή της στοίβας· το αποτέλεσμα αντικαθιστά το *i*-οστό καταχωρητή κάτω από την κορυφή της στοίβας.
FADD	mem32	Ένας τελεστέος προέλευσης είναι μια θέση 32 bit της μνήμης· το αποτέλεσμα αντικαθιστά την κορυφή της στοίβας.
FADD	mem64	Ένας τελεστέος προέλευσης είναι μια θέση 64 bit της μνήμης· το αποτέλεσμα αντικαθιστά την κορυφή της στοίβας.

ΕΙΚΟΝΑ 3.22 Οι παραλλαγές των τελεστέων για την πρόσθεση κινητής υποδιαστολής στην IA-32.

4. Τριγωνομετρικές εντολές, μεταξύ των οποίων υπολογισμός ημιτόνου (sine), συνημιτόνου (cosine), λογαρίθμου (log), και ύψωσης σε δύναμη (exponentiation)

Η Εικόνα 3.21 δείχνει μερικές από τις 60 πράξεις (λειτουργίες) κινητής υποδιαστολής. Σημειώστε ότι παίρνουμε ακόμη περισσότερους συνδυασμούς όταν συμπεριλάβουμε τους τρόπους λειτουργίας των τελεστέων γι' αυτές τις πράξεις. Η Εικόνα 3.22 παρουσιάζει τις πολλές επιλογές για την πρόσθεση αριθμών κινητής υποδιαστολής.

Οι εντολές κινητής υποδιαστολής κωδικοποιούνται με τη χρήση του κωδικού λειτουργίας (opcode) ESC του 8086 και του προσδιοριστή διεύθυνσης επιθεματικού byte — postbyte address specifier (δείτε την Εικόνα 2.46 στη σελίδα 161). Οι λειτουργίες μνήμης δεσμεύουν 2 bit για να προσδιορίσουν αν ο τελεστέος είναι αριθμός κινητής υποδιαστολής 32 ή 64 bit ή ακέραιος 16 ή 32 bit. Αυτά τα ίδια 2 bit χρησιμοποιούνται σε εκδόσεις που δεν προσπελάζουν τη μνήμη για να προσδιοριστεί αν πρέπει να γίνει εξαγωγή από τη στοίβα μετά τη λειτουργία και αν το αποτέλεσμα πρέπει να καταλήξει στην κορυφή της στοίβας ή σε ένα χαμηλότερο καταχωρητή.

Η απόδοση των πράξεων κινητής υποδιαστολής στην οικογένεια IA-32 παραδοσιακά βρισκόταν πολύ πίσω από άλλους υπολογιστές. Αν είναι απλώς έλλειψη προσοχής από τους μηχανικούς της Intel ή κάποιο ψεγάδι της αρχιτεκτονικής είναι δύσκολο να το γνωρίζουμε. Μπορούμε να πούμε ότι από το 1980 έχουν ανακοινωθεί πολλές νέες αρχιτεκτονικές, και καμία δεν έχει ακολουθήσει τα βήματα της Intel. Επιπλέον, η Intel δημιούργησε μια πιο παραδοσιακή αρχιτεκτονική κινητής υποδιαστολής ως μέρος του SSE2.

Η αρχιτεκτονική κινητής υποδιαστολής επέκτασης συνεχούς ροής SIMD 2 (Streaming SIMD Extension 2 — SSE2) της Intel

Το Κεφάλαιο 2 επισημαίνει ότι το 2001 η Intel πρόσθεσε 144 εντολές στην αρχιτεκτονική της, συμπεριλαμβάνοντας καταχωρητές και πράξεις κινητής υποδιαστολής διπλής ακρίβειας. Η προσθήκη αυτή περιλαμβάνει οκτώ καταχωρητές που μπορούν να χρησιμοποιηθούν για τελεστέους κινητής υποδιαστολής, δίνοντας στο μεταγλωττιστή ένα διαφορετικό στόχο για πράξεις κινητής υποδιαστολής από ό,τι η ξεχωριστή αρχιτεκτονική στοίβας. Οι μεταγλωττιστές μπορούν να προτιμήσουν να χρησιμοποιήσουν τους οκτώ καταχωρητές SSE2 ως καταχωρητές κινητής υποδιαστολής σαν αυτούς που συναντώνται σε άλλους υπολογιστές. Η AMD επέκτεινε τον αριθμό τους σε 16 ως μέρος της αρχιτεκτονικής AMD64, την οποία η Intel ονόμασε EM64T για δική της χρήση.

Εκτός από τη διατήρηση ενός αριθμού απλής ή διπλής ακρίβειας σε έναν καταχωρητή, η Intel επιτρέπει τη στοίβαξη πολλών τελεστέων κινητής υποδιαστολής σε ένα μόνο καταχωρητή SSE2 των 128 bit: τέσσερις απλής ακρίβειας ή δύο διπλής ακρίβειας. Αν οι τελεστέοι μπορούν να τακτοποιηθούν στη μνήμη ως ευθυγραμμισμένα δεδομένα των 128 bit, τότε μεταφορές δεδομένων των 128 bit μπορούν να φορτώσουν και να αποθηκεύσουν πολλούς τελεστέους ανά εντολή. Αυτή η συμπυκνωμένη μορφή κινητής υποδιαστολής υποστηρίζεται από αριθμητικές πράξεις που μπορούν να λειτουργήσουν ταυτόχρονα σε τέσσερις αριθμούς απλής ακρίβειας ή δύο διπλής ακρίβειας. Η νέα αρχιτεκτονική μπορεί να υπερδιπλασιάσει την απόδοση σε σχέση με την αρχιτεκτονική στοίβας.

Έτσι, τα μαθηματικά μπορούν να οριστούν ως το αντικείμενο στο οποίο ποτέ δε γνωρίζουμε για ποιο θέμα μιλούμε, ούτε αν αυτό που λέμε είναι αλήθεια.

Bertrand Russell, *Recent Words on the Principles of Mathematics*, 1901

3.8 Πλάνες και παγίδες

Οι αριθμητικές πλάνες και παγίδες γενικά πηγάζουν από τη διαφορά ανάμεσα στην περιορισμένη ακρίβεια της αριθμητικής υπολογιστών και την άπειρη ακρίβεια της φυσικής αριθμητικής.

Πλάνη: Η πρόσθεση κινητής υποδιαστολής είναι προσεταιριστική· δηλαδή, $x + (y + z) = (x + y) + z$.

Με δεδομένο το μεγάλο εύρος των αριθμών που μπορούν να αναπαρασταθούν στην κινητή υποδιαστολή, προκύπτουν προβλήματα όταν προσθέτουμε δύο μεγάλους αριθμούς με αντίθετα πρόσημα με ένα μικρό αριθμό.

Για παράδειγμα, υποθέστε ότι $x = -1{,}5_{ten} \times 10^{38}$, $y = 1{,}5_{ten} \times 10^{38}$, και $z = 1{,}0$, και ότι όλοι αυτοί είναι αριθμοί απλής ακρίβειας. Τότε

$$\begin{aligned} x + (y + z) &= -1{,}5_{ten} \times 10^{38} + (1{,}5_{ten} \times 10^{38} + 1{,}0) \\ &= -1{,}5_{ten} \times 10^{38} + (1{,}5_{ten} \times 10^{38}) = 0{,}0 \\ (x + y) + z &= (-1{,}5_{ten} \times 10^{38} + 1{,}5_{ten} \times 10^{38}) + 1{,}0 \\ &= (0{,}0_{ten}) + 1{,}0 \\ &= 1{,}0 \end{aligned}$$

Συνεπώς, $x + (y + z) \neq (x + y) + z$.

Επειδή οι αριθμοί κινητής υποδιαστολής έχουν περιορισμένη ακρίβεια και οδηγούν σε προσεγγίσεις των πραγματικών αποτελεσμάτων, το $1{,}5_{ten} \times 10^{38}$ είναι τόσο μεγαλύτερο από το $1{,}0_{ten}$ ώστε το $1{,}5_{ten} \times 10^{38} + 1{,}0$ παραμένει ίσο με $1{,}5_{ten} \times 10^{38}$. Γι' αυτόν το λόγο, το άθροισμα των x, y, και z είναι ίσο με 0,0 ή 1,0, ανάλογα με τη σειρά των προσθέσεων κινητής υποδιαστολής και, έτσι, η πρόσθεση κινητής υποδιαστολής *δεν* είναι προσεταιριστική.

> *Πλάνη: Ακριβώς όπως μια εντολή αριστερής ολίσθησης μπορεί να αντικαταστήσει έναν ακέραιο πολλαπλασιασμό με μια δύναμη του 2, μια δεξιά ολίσθηση είναι το ίδιο με μια ακέραια διαίρεση με μια δύναμη του 2.*

Θυμηθείτε ότι ένας δυαδικός αριθμός x, όπου xi σημαίνει το i-οστό bit, αναπαριστά τον αριθμό

$$\ldots + (x3 \times 2^3) + (x2 \times 2^2) + (x1 \times 2^1) + (x0 \times 2^0)$$

Η ολίσθηση των bit τού x δεξιά κατά n bit φαίνεται να είναι το ίδιο με τη διαίρεση με το 2^n. Και αυτό *είναι* αλήθεια για απρόσημους αριθμούς. Το πρόβλημα είναι με τους προσημασμένους αριθμούς. Για παράδειγμα, υποθέστε ότι θέλουμε να διαιρέσουμε το -5_{ten} με το 4_{ten}· το πηλίκο θα έπρεπε να είναι -1_{ten}. Η αναπαράσταση συμπληρώματος ως προς δύο του -5_{ten} είναι

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011_{two}$$

Σύμφωνα με αυτή την πλάνη, η δεξιά ολίσθηση κατά δύο θα διαιρούσε με το 4_{ten} (2^2):

$$0011\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two}$$

Με 0 στο bit προσήμου, αυτό το αποτέλεσμα είναι σαφώς λάθος. Η τιμή που δημιουργήθηκε με τη δεξιά ολίσθηση είναι στη πραγματικότητα $1.073.741.822_{ten}$ αντί για -1_{ten}.

Μια λύση θα ήταν να έχουμε μια αριθμητική δεξιά ολίσθηση — arithmetic right shift (δείτε την ενότητα Σε μεγαλύτερο βάθος: αλγόριθμος Booth) που επεκτείνει το bit προσήμου αντί να εισάγει μηδενικά. Μια αριθμητική δεξιά ολίσθηση του -5_{ten} κατά 2 bit δίνει

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two}$$

Το αποτέλεσμα είναι -2_{ten} αντί για -1_{ten}· κοντά, αλλά όχι ακριβώς.

Ο επεξεργαστής PowerPC, ωστόσο, έχει μια γρήγορη εντολή ολίσθησης (*δεξιά αλγεβρική ολίσθηση* — shift right algebraic) η οποία, συνδυασμένη με μια ειδική πρόσθεση (add with carry — πρόσθεση με κρατούμενο), δίνει το ίδιο αποτέλεσμα όπως η διαίρεση με μια δύναμη του 2.

Παγίδα: Η εντολή add immediate unsigned (`addiu`) *του MIPS κάνει επέκταση προσήμου του άμεσου (immediate) πεδίου της των 16 bit.*

Παρά το όνομά της, η add immediate unsigned (`addiu`) χρησιμοποιείται για την πρόσθεση σταθερών σε προσημασμένους ακεραίους όταν δε μας απασχολεί η υπερχείλιση. Ο MIPS δεν έχει εντολή άμεσης αφαίρεσης (subtract immediate) και οι αρνητικοί αριθμοί χρειάζονται επέκταση προσήμου, οπότε οι σχεδιαστές του MIPS αποφάσισαν να εφαρμόσουν επέκταση προσήμου στο άμεσο πεδίο.

Πλάνη: Μόνον οι θεωρητικοί μαθηματικοί νοιάζονται για την ακρίβεια της κινητής υποδιαστολής.

Οι τίτλοι των εφημερίδων του Νοεμβρίου 1994 αποδεικνύουν ότι αυτή η πρόταση είναι πλάνη (δείτε την Εικόνα 3.23). Τα επόμενα είναι η ιστορία πίσω από τους τίτλους.

Ο Pentium χρησιμοποιεί έναν τυποποιημένο αλγόριθμο διαίρεσης κινητής υποδιαστολής που παράγει πολλά bit του πηλίκου σε κάθε βήμα, χρησιμοποιώντας τα περισσότερο σημαντικά bit του διαιρέτη και του διαιρετέου για να προβλέψει τα επόμενα 2 bit του πηλίκου. Η πρόβλεψη λαμβάνεται από έναν πίνακα αναζήτησης (lookup table) που περιέχει –2, –1, 0, +1, ή +2. Η πρόβλεψη πολλαπλασιάζεται με το διαιρέτη και αφαιρείται από το υπόλοιπο για να δώσει ένα νέο υπόλοιπο. Όπως στη διαίρεση χωρίς επαναφορά (nonrestoring division — δείτε την Άσκηση 3.29), αν μια προηγούμενη πρόβλεψη δώσει πολύ μεγάλο υπόλοιπο, το μερικό υπόλοιπο προσαρμόζεται σε μια επόμενη διέλευση.

ΕΙΚΟΝΑ 3.23 Δείγματα άρθρων σε εφημερίδες και περιοδικά του Νοεμβρίου 1994, όπως οι *New York Times*, *San Jose Mercury News*, *San Francisco Chronicle*, και *Infoworld*. Το σφάλμα διαίρεσης κινητής υποδιαστολής του Pentium μπήκε μέχρι και στον κατάλογο «Top 10» του *David Letterman Late Show* στην τηλεόραση. Η Intel τελικά υπέστη μια ανεπανόρθωτη ζημιά 300 εκατομμυρίων δολαρίων για να αντικαταστήσει τα προβληματικά τσιπ.

Προφανώς υπήρχαν πέντε στοιχεία του πίνακα (αναζήτησης) από τον 80486, που η Intel νόμιζε ότι δε θα μπορούσαν ποτέ να προσπελαστούν, οπότε βελτιστοποίησε τον προγραμματίσιμο λογικό πίνακα (programmable logic array — PLA) να επιστρέφει 0 αντί για 2 σε αυτές τις περιπτώσεις στον Pentium. Η Intel είχε άδικο: ενώ τα πρώτα 11 bit ήταν πάντοτε σωστά, θα εμφανιζόταν περιστασιακά λάθη στα bit 12 έως 52, ή στα δεκαδικά ψηφία από το 4ο έως το 15ο.

Η υπόθεση του ηθικοπλαστικού έργου του σφάλματος του Pentium είναι η εξής:

- *Ιούλιος 1994:* Η Intel ανακαλύπτει το σφάλμα στον Pentium. Το πραγματικό κόστος για τη διόρθωση του σφάλματος ήταν πολλές εκατοντάδες χιλιάδες δολάρια. Με κανονικές διαδικασίες διόρθωσης σφάλματος, η αλλαγή, ο επανέλεγχος, και η επαναφορά των διορθωμένων τσιπ στην παραγωγή θα έπαιρνε μήνες. Η Intel σχεδίαζε να βάλει σωστά τσιπ στην παραγωγή τον Ιανουάριο 1995, εκτιμώντας ότι 3 με 5 εκατομμύρια Pentium θα παράγονταν με το σφάλμα.
- *Σεπτέμβριος 1994:* Ένας καθηγητής μαθηματικών του Lynchburg College στη Virginia, ο Thomas Nicely, ανακαλύπτει το σφάλμα. Αφού κάλεσε την τεχνική εξυπηρέτηση της Intel και δεν πήρε καμία επίσημη απάντηση, δημοσίευσε την ανακάλυψή του στο Διαδίκτυο. Γρήγορα κέρδισε οπαδούς, και μερικοί επισήμαναν ότι ακόμη και μικρά λάθη γίνονται μεγάλα όταν πολλαπλασιάζονται με μεγάλους αριθμούς: για παράδειγμα, το κλάσμα των ανθρώπων με μια σπάνια ασθένεια πολλαπλασιασμένο με τον πληθυσμό της Ευρώπης μπορεί να οδηγήσει σε λάθος εκτίμηση του αριθμού των ασθενών.
- *7 Νοεμβρίου, 1994:* Η *Electronic Engineering Times* βάζει την ιστορία στην πρώτη σελίδα, και σύντομα ακολουθούν και άλλες εφημερίδες.
- *22 Νοεμβρίου, 1994:* Η Intel βγάζει μια ανακοίνωση τύπου, αποκαλώντας (το σφάλμα) απλώς «στιγμιαία ανωμαλία» (glitch). Ο Pentium «μπορεί να κάνει λάθη στο ένατο ψηφίο. . . . Ακόμη και οι περισσότεροι μηχανικοί και οικονομικοί αναλυτές απαιτούν ακρίβεια μόνο μέχρι το τέταρτο ή το πέμπτο δεκαδικό ψηφίο. Οι χρήστες λογιστικών φύλλων (spreadsheets) και επεξεργαστών κειμένου (word processors) δε χρειάζεται να ανησυχούν. . . . Υπάρχουν ίσως μερικές δεκάδες άνθρωποι που θα επηρεαστούν από αυτό. Μέχρι τώρα, έχουμε επικοινωνήσει μόνο με έναν. . . . [Μόνο] θεωρητικοί μαθηματικοί (με υπολογιστές Pentium που αγοράστηκαν πριν από το καλοκαίρι) πρέπει να ανησυχούν.» Αυτό που ενόχλησε πολλούς ήταν ότι ζητήθηκε από τους πελάτες να περιγράψουν την εφαρμογή τους στην Intel, και στη συνέχεια η *Intel* θα αποφάσιζε αν η εφαρμογή τους άξιζε ένα νέο Pentium χωρίς το σφάλμα της διαίρεσης.
- *5 Δεκεμβρίου, 1994:* Η Intel ισχυρίζεται ότι το σφάλμα συμβαίνει μία φορά σε 27.000 χρόνια για τον τυπικό χρήστη λογιστικού φύλλου. Η Intel υποθέτει ότι ένας χρήστης κάνει 1000 διαιρέσεις την ημέρα και πολλαπλασιάζει το ρυθμό εμφάνισης του λάθους υποθέτοντας ότι οι αριθμοί κινητής υποδιαστολής είναι τυχαίοι, που σημαίνει ένα στα 9 δισεκατομμύρια, και μετά παίρνει 9 εκατομμύρια ημέρες ή 27.000 χρόνια. Τα πράγματα αρχίζουν να ηρεμούν, παρόλο που η Intel αμέλησε να εξηγήσει γιατί ένας τυπικός πελάτης χρησιμοποιεί αριθμούς κινητής υποδιαστολής με τυχαίο τρόπο.

- *12 Δεκεμβρίου, 1994:* Η IBM Research Division διαφοροποιείται από τον υπολογισμό της Intel για το ρυθμό των λαθών (μπορείτε να διαβάσετε αυτό το άρθρο στη διεύθυνση *www.mkp.com/books_catalog/cod/links.htm*). Η IBM ισχυρίζεται ότι τα κοινά προγράμματα λογιστικών φύλλων, εκτελώντας επανυπολογισμούς για 15 λεπτά την ημέρα, θα μπορούσαν να παραγάγουν λάθη σχετικά με τον Pentium τόσο συχνά όσο μία φορά κάθε 24 ημέρες. Η IBM υποθέτει 5000 διαιρέσεις το δευτερόλεπτο για 15 λεπτά, που οδηγεί σε 4,2 εκατομμύρια διαιρέσεις την ημέρα, και δεν υποθέτει τυχαία κατανομή των αριθμών αλλά, αντίθετα, υπολογίζει την πιθανότητα σε μία στα 100 εκατομμύρια. Ως αποτέλεσμα, η IBM διακόπτει αμέσως την προώθηση όλων των προσωπικών υπολογιστών της που βασίζονται σε Pentium. Η κατάσταση παίρνει και πάλι φωτιά για την Intel.
- *21 Δεκεμβρίου, 1994:* Η Intel δημοσιεύει την παρακάτω ανακοίνωση, υπογεγραμμένη από τον πρόεδρό της, το διευθύνοντα σύμβουλο και εκτελεστικό διευθυντή (chief executive officer), το λειτουργικό διευθυντή (chief operating officer), και τον πρόεδρο του διοικητικού συμβουλίου: «Εμείς στην Intel επιθυμούμε να απολογηθούμε ειλικρινά για τους χειρισμούς μας στο σφάλμα του επεξεργαστή Pentium που πρόσφατα δημοσιοποιήθηκε. Το σύμβολο Intel Inside σημαίνει ότι ο υπολογιστής σας έχει μικροεπεξεργαστή που δεν υστερεί σε σχέση με κανέναν άλλο σε ποιότητα και απόδοση. Χιλιάδες εργαζομένων της Intel δουλεύουν πολύ σκληρά για να το εξασφαλίσουν αυτό. Αλλά κανένας μικροεπεξεργαστής δεν είναι ποτέ τέλειος. Αυτό που η Intel εξακολουθεί να πιστεύει είναι ότι ένα εξαιρετικά μικρό πρόβλημα απέκτησε τεχνηέντως δική του ζωή. Παρόλο που η Intel σταθερά υποστηρίζει την ποιότητα της τρέχουσας έκδοσης του επεξεργαστή Pentium, αναγνωρίζουμε ότι πολλοί χρήστες έχουν ανησυχίες. Θέλουμε να ανακουφίσουμε αυτές τις ανησυχίες. Η Intel θα ανταλλάξει την τρέχουσα έκδοση του επεξεργαστή Pentium με μια ενημερωμένη έκδοση, στην οποία αυτό το σφάλμα διαίρεσης κινητής υποδιαστολής έχει διορθωθεί, για κάθε ιδιοκτήτη που θα το ζητήσει, δωρεάν κατά τη διάρκεια της ζωής του υπολογιστή του.» Αναλυτές εκτιμούν ότι αυτή η ανάκληση κόστισε στην Intel 500 εκατομμύρια δολάρια, ενώ οι εργαζόμενοι της Intel δεν πήραν Χριστουγεννιάτικο δώρο εκείνη τη χρονιά.

Η ιστορία αυτή φέρνει στην επιφάνεια μερικά σημεία για να συλλογιστεί ο καθένας. Πόσο φθηνότερο θα ήταν να διορθωθεί το σφάλμα τον Ιούλιο του 1994; Ποιο ήταν το κόστος της διόρθωσης της ζημιάς στη φήμη της Intel; Και ποια είναι η εταιρική ευθύνη στην αποκάλυψη σφαλμάτων σε ένα προϊόν τόσο ευρύτατα χρησιμοποιούμενο και στο οποίο βασίζονται οι χρήστες, όπως είναι ένας μικροεπεξεργαστής;

Τον Απρίλιο του 1997, στους μικροεπεξεργαστές Pentium Pro και Pentium II αποκαλύφθηκε άλλο ένα σφάλμα κινητής υποδιαστολής. Όταν οι εντολές αποθήκευσης αριθμού κινητής υποδιαστολής σε ακέραιο (`fist`, `fistp`) συναντούν έναν αρνητικό αριθμό κινητής υποδιαστολής που είναι πολύ μεγάλος για να χωρέσει σε μια λέξη των 16 ή 32 bit αφού μετατραπεί σε ακέραιο, ενεργοποιούν το λάθος bit στη λέξη κατάστασης FPO — εξαίρεση ακρίβειας (precision exception) αντί για εξαίρεση μη επιτρεπτής λειτουργίας (invalid operation

exception). Πιστώνεται στην Intel ότι αυτή τη φορά παραδέχθηκαν δημόσια το σφάλμα και πρόσφεραν μια επιδιόρθωση λογισμικού (software patch) για να το αντιμετωπίσει — μια τελείως διαφορετική αντίδραση από αυτή του 1994.

3.9 Συμπερασματικές παρατηρήσεις

Η αριθμητική υπολογιστών, σε σχέση με την αριθμητική με χαρτί και μολύβι, πάσχει από περιορισμένη ακρίβεια. Αυτός ο περιορισμός μπορεί να έχει ως αποτέλεσμα μη επιτρεπτές πράξεις υπολογισμού αριθμών μεγαλύτερων ή μικρότερων από τα προκαθορισμένα όρια. Τέτοιες ανωμαλίες, που ονομάζονται «υπερχείλιση» ή «ανεπάρκεια», μπορεί να οδηγήσουν σε εξαιρέσεις (exceptions) ή σε διακοπές (interrupts), συμβάντα επείγουσας αντιμετώπισης παρόμοια με τις μη αναμενόμενες κλήσεις υπορουτινών. Το Κεφάλαιο 5 καλύπτει τις εξαιρέσεις με περισσότερες λεπτομέρειες.

Η αριθμητική κινητής υποδιαστολής έχει την επιπλέον δυσκολία ότι αποτελεί προσέγγιση των πραγματικών αριθμών, και χρειάζεται προσοχή να εξασφαλίσουμε ότι ο αριθμός που επιλέγει ο υπολογιστής είναι η πλησιέστερη αναπαράσταση του πραγματικού αριθμού. Οι προκλήσεις της ανακρίβειας και της περιορισμένης αναπαράστασης ευθύνονται κατά ένα μέρος για την ανάπτυξη του πεδίου της αριθμητικής ανάλυσης (numerical analysis).

Με την πάροδο των χρόνων, η αριθμητική υπολογιστών έχει προτυποποιηθεί σε μεγάλο βαθμό, βελτιώνοντας σημαντικά τη φορητότητα των προγραμμάτων. Η ακέραια δυαδική αριθμητική συμπληρώματος ως προς δύο και η δυαδική αριθμητική κινητής υποδιαστολής με βάση το πρότυπο IEEE 754 συναντώνται στην τεράστια πλειοψηφία των υπολογιστών που πωλούνται σήμερα. Για παράδειγμα, κάθε επιτραπέζιος υπολογιστής που πωλήθηκε από τότε που αυτό το βιβλίο τυπώθηκε για πρώτη φορά ακολουθεί αυτές τις συμβάσεις.

Μια παρενέργεια του υπολογιστή αποθηκευμένου προγράμματος είναι ότι οι σειρές των bit δεν έχουν κάποιο εσωτερικό νόημα. Η ίδια σειρά bit μπορεί να αναπαριστά έναν προσημασμένο ακέραιο, έναν απρόσημο ακέραιο, έναν αριθμό κινητής υποδιαστολής, μια εντολή, και ούτω καθεξής. Η εντολή που επενεργεί στη λέξη είναι αυτή η οποία καθορίζει το νόημά της.

Την εξήγηση της αριθμητικής υπολογιστών στο κεφάλαιο αυτό ακολουθεί μια περιγραφή που καλύπτει πολύ περισσότερα από το σύνολο εντολών του MIPS. Ένα σημείο σύγχυσης είναι οι εντολές που καλύψαμε στα κεφάλαια αυτά σε σχέση με τις εντολές που εκτελούνται από τα ολοκληρωμένα κυκλώματα του MIPS και σε σχέση με τις εντολές που γίνονται δεκτές από τους συμβολομεταφραστές του MIPS. Οι δύο επόμενες εικόνες προσπαθούν να το ξεκαθαρίσουν.

Η Εικόνα 3.24 παρουσιάζει τον κατάλογο των εντολών MIPS που καλύψαμε στο κεφάλαιο αυτό και στο Κεφάλαιο 2. Ονομάζουμε το σύνολο των εντολών στο αριστερό μέρος της εικόνας *πυρήνα του MIPS* (MIPS core). Τις εντολές δεξιά τις ονομάζουμε *αριθμητικό πυρήνα του MIPS* (MIPS arithmetic core). Στα αριστερά της Εικόνας 3.25 είναι οι εντολές που εκτελεί ο επεξεργαστής του MIPS και δεν υπάρχουν στην Εικόνα 3.24. Ονομάζουμε το πλήρες σύνολο των εντολών υλικού *MIPS-32*. Στα δεξιά της Εικόνας 3.25 είναι οι εντολές που γίνονται δεκτές από το συμβολομεταφραστή και δεν αποτελούν μέρος του MIPS-32. Ονομάζουμε το σύνολο αυτών των εντολών *ψευδο-MIPS* (Pseudo MIPS).

Υποσύνολο εντολών	Ακέραιοι	Κινητής υποδιαστολής
Πυρήνας MIPS (MIPS core)	95%	57%
Αριθμητικός πυρήνας MIPS (MIPS arithmetic core)	0%	41%
Υπόλοιπες MIPS-32	5%	2%

Εντολές πυρήνα MIPS	Όνομα	Μορφή	Αριθμητικός πυρήνας MIPS	Όνομα	Μορφή
add	add	R	multiply	mult	R
add immediate	addi	I	multiply unsigned	multu	R
add unsigned	addu	R	divide	div	R
add immediate unsigned	addiu	I	divide unsigned	divu	R
subtract	sub	R	move from Hi	mfhi	R
subtract unsigned	subu	R	move from Lo	mflo	R
and	and	R	move from system control (EPC)	mfc0	R
and immediate	andi	I	floating-point add single	add.s	R
or	or	R	floating-point add double	add.d	R
or immediate	ori	I	floating-point subtract single	sub.s	R
nor	nor	R	floating-point subtract double	sub.d	R
shift left logical	sll	R	floating-point multiply single	mul.s	R
shift right logical	srl	R	floating-point multiply double	mul.d	R
load upper immediate	lui	I	floating-point divide single	div.s	R
load word	lw	I	floating-point divide double	div.d	R
store word	sw	I	load word to floating-point single	lwc1	I
load halfword unsigned	lhu	I	store word to floating-point single	swc1	I
store halfword	sh	I	load word to floating-point double	ldc1	I
load byte unsigned	lbu	I	store word to floating-point double	sdc1	I
store byte	sb	I	branch on floating-point true	bc1t	I
branch on equal	beq	I	branch on floating-point false	bc1f	I
branch on not equal	bne	I	floating-point compare single	c.x.s	R
jump	j	J	(x = eq, neq, lt, le, gt, ge)		
jump and link	jal	J	floating-point compare double	c.x.d	R
jump register	jr	R	(x = eq, neq, lt, le, gt, ge)		
set less than	slt	R			
set less than immediate	slti	I			
set less than unsigned	sltu	R			
set less than immediate unsigned	sltiu	I			

ΕΙΚΟΝΑ 3.24 Το σύνολο εντολών του MIPS που καλύψαμε μέχρι εδώ. Το βιβλίο αυτό επικεντρώνεται στις εντολές της αριστερής στήλης.

Υπόλοιπες MIPS-32	Όνομα	Μορφή	Ψευδο-MIPS	Όνομα	Μορφή
exclusive or (*rs* ⊕ *rt*)	xor	R	move	move	rd,rs
exclusive or immediate	xori	I	absolute value	abs	rd,rs
shift right arithmetic	sra	R	not (¬*rs*)	not	rd,rs
shift left logical variable	sllv	R	negate *(signed ή unsigned)*	neg*s*	rd,rs
shift right logical variable	srlv	R	rotate left	rol	rd,rs,rt
shift right arithmetic variable	srav	R	rotate right	ror	rd,rs,rt
move to Hi	mthi	R	multiply and don't check oflw *(signed ή uns.)*	mul*s*	rd,rs,rt
move to Lo	mtlo	R	multiply and check oflw *(signed ή uns.)*	mulo*s*	rd,rs,rt
load halfword	lh	I	divide and check overflow	div	rd,rs,rt
load byte	lb	I	divide and don't check overflow	divu	rd,rs,rt
load word left (*unaligned*)	lwl	I	remainder *(signed ή unsigned)*	rem*s*	rd,rs,rt
load word right (*unaligned*)	lwr	I	load immediate	li	rd,imm
store word left (*unaligned*)	swl	I	load address	la	rd,addr
store word right (*unaligned*)	swr	I	load double	ld	rd,addr
load linked (*atomic update*)	ll	I	store double	sd	rd,addr
store cond. (*atomic update*)	sc	I	unaligned load word	ulw	rd,addr
move if zero	movz	R			
move if not zero	movn	R	unaligned store word	usw	rd,addr
multiply and add (S ή uns.)	madd*s*	R			
multiply and subtract (S ή uns.)	msub*s*	I	unaligned load halfword *(signed ή uns.)*	ulh*s*	rd,addr
branch on > zero and link	bgezal	I	unaligned store halfword	ush	rd,addr
branch on < zero and link	bltzal	I	branch	b	Επικέτα
jump and link register	jalr	R	branch on equal zero	beqz	rs,L
branch compare to zero	bxz	I	branch on compare *(signed ή unsigned)*	bxs	rs,rt,L
branch compare to zero likely	bxzl	I	(x = lt, le, gt, ge)		
(x = lt, le, gt, ge)			set equal	seq	rd,rs,rt
branch compare reg likely	bxl	I	set not equal	sne	rd,rs,rt
trap if compare reg	tx	R	set on compare *(signed ή unsigned)*	sx*s*	rd,rs,rt
trap if compare immediate	txi	I	(x = lt, le, gt, ge)		
(x = eq, neq, lt, le, gt, ge)			load to floating point (*s ή d*)	l.*f*	rd,addr
return from exception	rfe	R	store from floating point (*s ή d*)	s.*f*	rd,addr
system call	syscall	I			
break *(cause exception)*	break	I			
move from FP to integer	mfc1	R			
move to FP from integer	mtc1	R			
FP move (*s ή d*)	mov.*f*	R			
FP move if zero (*s ή d*)	movz.*f*	R			
FP move if not zero (*s ή d*)	movn.*f*	R			
FP square root (*s ή d*)	sqrt.*f*	R			
FP absolute value (*s ή d*)	abs.*f*	R			
FP negate (*s ή d*)	neg.*f*	R			
FP convert (*w, s, ή d*)	cvt.*f*.*f*	R			
FP compare un (*s ή d*)	c.xn.*f*	R			

ΕΙΚΟΝΑ 3.25 Υπόλοιπα συνόλου εντολών MIPS-32 και «ψευδο-MIPS». *f* σημαίνει εντολές κινητής υποδιαστολής απλής (s) και διπλής (d) ακρίβειας, και *s* σημαίνει προσημασμένες και απρόσημες (u) εκδόσεις. Ο MIPS-32 διαθέτει επίσης εντολές κινητής υποδιαστολής για πολλαπλασιασμό και πρόσθεση ή αφαίρεση (madd.*f*/msub.*f*), οροφής (ceiling — ceil.*f*), αποκοπής (truncate — trunc.*f*), στρογγυλοποίησης (round — round.*f*), και αντιστρόφου (reciprocal —recip.*f*),

Η Εικόνα 3.26 παρουσιάζει τη δημοτικότητα των εντολών MIPS στα μετροπρογράμματα SPEC2000 για ακεραίους και αριθμούς κινητής υποδιαστολής. Παρουσιάζονται όλες οι εντολές οι υπεύθυνες τουλάχιστον για το 1% των εντολών που εκτελούνται. Ο πίνακας που ακολουθεί συνοψίζει αυτές τις πληροφορίες.

Σημειώστε ότι, αν και οι προγραμματιστές και οι συγγραφείς μεταγλωττιστών μπορούν να χρησιμοποιήσουν τον MIPS-32 για να έχουν έναν πλουσιότερο κατάλογο επιλογών, οι εντολές του πυρήνα του MIPS κυριαρχούν στην εκτέλεση των μετροπρογραμμάτων SPEC2000 για ακέραιους, και ο ακέραιος πυρήνας συν τον αριθμητικό πυρήνα κυριαρχούν στα SPEC2000 κινητής υποδιαστολής.

Στο υπόλοιπο του βιβλίου, επικεντρωνόμαστε στις εντολές του πυρήνα του MIPS — το σύνολο εντολών ακεραίων εκτός του πολλαπλασιασμού και της διαίρεσης — για να κάνουμε ευκολότερη την εξήγηση του σχεδιασμού του υπολογιστή. Όπως μπορούμε να δούμε, ο πυρήνας του MIPS περιλαμβάνει τις πιο δημοφιλείς εντολές τού MIPS, και να είστε σίγουροι ότι η κατανόηση ενός υπολογιστή που εκτελεί τον πυρήνα τού MIPS θα σας δώσει αρκετό υπόβαθρο για να κατανοήσετε ακόμη πιο φιλόδοξους υπολογιστές.

Πυρήνας MIPS	Όνομα	Ακέραιοι	Κινητής υποδιαστολής	Αριθμητικός πυρήνας + MIPS-32	Όνομα	Ακέραιοι	Κινητής υποδιαστολής
add	add	0%	0%	FP add double	add.d	0%	8%
add immediate	addi	0%	0%	FP subtract double	sub.d	0%	3%
add unsigned	addu	7%	21%	FP multiply double	mul.d	0%	8%
add immediate unsigned	addiu	12%	2%	FP divide double	div.d	0%	0%
subtract unsigned	subu	3%	2%	load word to FP double	l.d	0%	15%
and	and	1%	0%	store word to FP double	s.d	0%	7%
and immediate	andi	3%	0%	shift right arithmetic	sra	1%	0%
or	or	7%	2%	load half	lhu	1%	0%
or immediate	ori	2%	0%	branch less than zero	bltz	1%	0%
nor	nor	3%	1%	branch greater or equal	bgez	1%	0%
shift left logical	sll	1%	1%	branch less or equal zero	blez	0%	1%
shift right logical	srl	0%	0%	multiply	mul	0%	1%
load upper immediate	lui	2%	5%				
load word	lw	24%	15%				
store word	sw	9%	2%				
load byte	lbu	1%	0%				
store byte	sb	1%	0%				
branch on equal (zero)	beq	6%	2%				
branch on not equal (zero)	bne	5%	1%				
jump and link	jal	1%	0%				
jump register	jr	1%	0%				
set less than	slt	2%	0%				
set less than immediate	slti	1%	0%				
set less than unsigned	sltu	1%	0%				
set less than imm. uns.	sltiu	1%	0%				

ΕΙΚΟΝΑ 3.26 Η συχνότητα των εντολών MIPS για τα μετροπρογράμματα SPEC2000 ακεραίων και κινητής υποδιαστολής. Στον πίνακα περιλαμβάνονται όλες οι εντολές που συμμετέχουν τουλάχιστον στο 1% των εκτελούμενων εντολών. Οι ψευδοεντολές μετατρέπονται σε MIPS-32 πριν από την εκτέλεση και, έτσι, δεν εμφανίζονται εδώ. Αυτά τα δεδομένα προέρχονται από το Κεφάλαιο 2 του βιβλίου *Computer Architecture: A Quantitative Approach,* τρίτη έκδοση.

Β ΤΟΜΟΣ 3.10 Ιστορική προοπτική και πρόσθετες πηγές

Ο νόμος του Gresham («Τα κακά χρήματα διώχνουν τα καλά») για τους υπολογιστές θα έλεγε, «Ο γρήγορος νικά τον αργό ακόμη κι αν ο γρήγορος είναι λάθος.»
W. Kahan, 1992

Αυτή η ενότητα εξετάζει την ιστορία της κινητής υποδιαστολής από τον Von Neumann ακόμη, μαζί με την εκπληκτικά αμφιλεγόμενη προσπάθεια των προτύπων τού IEEE και την αιτιολογία σχετικά με την αρχιτεκτονική στοίβας 80 bit για αριθμούς κινητής υποδιαστολής στην αρχιτεκτονική IA-32. Δείτε την Ενότητα 3.1.

3.11 Ασκήσεις

Ποτέ μην υποτάσσεσαι, ποτέ μην υποτάσσεσαι, ποτέ, ποτέ, ποτέ — σε τίποτε, σπουδαίο ή ασήμαντο, μεγάλο ή μικρό — ποτέ μην υποτάσσεσαι.
Winston Churchill, ομιλία στο Harrow School, 1941, *Abroad*, 1869

3.1 [3] <§3.2> Μετατρέψτε το 4096_{ten} σε ένα δυαδικό αριθμό των 32 bit σε συμπλήρωμα ως προς δύο.

3.2 [3] <§3.2> Μετατρέψτε το -2047_{ten} σε ένα δυαδικό αριθμό των 32 bit σε συμπλήρωμα ως προς δύο.

3.3 [5] <§3.2> Μετατρέψτε το $-2.000.000_{ten}$ σε ένα δυαδικό αριθμό των 32 bit σε συμπλήρωμα ως προς δύο.

3.4 [5] <§3.2> Ποιο δεκαδικό αριθμό αντιπροσωπεύει ο επόμενος δυαδικός σε συμπλήρωμα ως προς δύο: `1111 1111 1111 1111 1111 1111 0000 0110`$_{two}$;

3.5 [5] <§3.2> Ποιο δεκαδικό αριθμό αντιπροσωπεύει ο επόμενος δυαδικός αριθμός σε συμπλήρωμα ως προς δύο:
`1111 1111 1111 1111 1111 1111 1110 1111`$_{two}$;

3.6 [5] <§3.2> Ποιο δεκαδικό αριθμό αντιπροσωπεύει ο επόμενος δυαδικός αριθμός σε συμπλήρωμα ως προς δύο:
`0111 1111 1111 1111 1111 1111 1110 1111`$_{two}$;

3.7 [10] <§3.2> Βρείτε τη συντομότερη ακολουθία εντολών MIPS που προσδιορίζει την απόλυτη τιμή ενός ακεραίου σε συμπλήρωμα ως προς δύο. Μετατρέψτε την παρακάτω εντολή (που είναι αποδεκτή από το συμβολομεταφραστή του MIPS):

```
abs $t2, $t3
```

Αυτή η εντολή σημαίνει ότι ο καταχωρητής `$t2` περιέχει ένα αντίγραφο του καταχωρητή `$t3` αν ο καταχωρητής `$t3` είναι θετικός, και το συμπλήρωμα ως προς δύο του καταχωρητή `$t3` αν ο `$t3` είναι αρνητικός. (Υπόδειξη: μπορεί να γίνει με τρεις εντολές.)

3.8 [10] <§3.2> Για περισσότερη εξάσκηση: Αναπαραστάσεις αριθμών

3.9 [10] <§3.2> Αν `A` είναι μια διεύθυνση 32 bit, τυπικά μια ακολουθία εντολών σαν την

```
lui $t0, A_upper
ori $t0, $t0, A_lower
lw  $s0, 0($t0)
```

μπορεί να χρησιμοποιηθεί για τη φόρτωση της λέξης, που βρίσκεται στη διεύθυνση `A`, σε έναν καταχωρητή (στην περίπτωση αυτή τον `$s0`). Θεωρήστε την εξής εναλλακτική λύση, η οποία είναι πιο αποδοτική:

```
lui $t0, A_upper_adjusted
lw  $s0, A_lower($t0)
```

Περιγράψτε πώς ρυθμίζεται το `A_upper` ώστε να επιτρέψει να δουλέψει αυτός ο απλούστερος κώδικας. (Υπόδειξη: το `A_upper` χρειάζεται να ρυθμιστεί επειδή το `A_lower` θα έχει υποστεί επέκταση προσήμου.)

3.10 [10] <§3.3> Βρείτε τη συντομότερη ακολουθία εντολών MIPS που προσδιορίζει αν υπάρχει κρατούμενο εξόδου από την πρόσθεση δύο καταχωρητών, ας πούμε, των `$t3` και `$t4`. Τοποθετήστε 0 ή 1 στον καταχωρητή `$t2` αν το κρατούμενο εξόδου είναι 0 ή 1, αντίστοιχα. (Υπόδειξη: Μπορεί να γίνει με δύο εντολές.)

3.11 [15] <§3.3> Για περισσότερη εξάσκηση: Γραφή κώδικα MIPS για αριθμητικές πράξεις

3.12 [15] <§3.3> Υποθέστε ότι όλες οι εντολές διακλάδωσης υπό συνθήκη εκτός της `beq` και της `bne` αφαιρούνται από το σύνολο εντολών του MIPS μαζί με την `slt` και όλες τις παραλλαγές της (`slti`, `sltu`, `sltui`). Δείξτε πώς να εκτελέσουμε την εντολή

```
slt $t0, $s0, $s1
```

χρησιμοποιώντας το τροποποιημένο σύνολο εντολών στο οποίο η `slt` δεν είναι διαθέσιμη. (Υπόδειξη: Απαιτεί περισσότερες από δύο εντολές.)

3.13 [10] <§3.3> Σχεδιάστε τις πύλες (gates) του bit αθροίσματος (Sum) ενός αθροιστή, αν δίνεται η εξίσωση της σελίδας B-28.

3.14 [5] <§3.4> Για περισσότερη εξάσκηση: Γραφή κώδικα MIPS για αριθμητικές πράξεις

3.15 [20] <§3.4> Για περισσότερη εξάσκηση: Γραφή κώδικα MIPS για αριθμητικές πράξεις

3.16 [2 εβδομάδες] <§3.4> Για περισσότερη εξάσκηση: Προσομοίωση μηχανών MIPS

3.17 [1 εβδομάδα] <§3.4> Για περισσότερη εξάσκηση: Προσομοίωση μηχανών MIPS

3.18 [5] <§3.4> Για περισσότερη εξάσκηση: Αθροιστές πρόβλεψης κρατουμένου

3.19 [15] <§3.4> Για περισσότερη εξάσκηση: Αθροιστές πρόβλεψης κρατουμένου

3.20 [10] <§3.4> Για περισσότερη εξάσκηση: Σχετική απόδοση αθροιστών

3.21 [15] <§3.4> Για περισσότερη εξάσκηση: Σχετική απόδοση αθροιστών

3.22 [15] <§3.4> Για περισσότερη εξάσκηση: Σχετική απόδοση αθροιστών

3.23 [30] <§3.4> Σε μεγαλύτερο βάθος: Αλγόριθμος Booth

3.24 [15][30] <§3.4> Για περισσότερη εξάσκηση: Ειδικοί καταχωρητές του MIPS

3.25 [10] <§§3.5, 3.4> Σε μεγαλύτερο βάθος: Η εντολή πολλαπλασιασμού-πρόσθεσης του Power PC

3.26 [20] <§3.5> Σε μεγαλύτερο βάθος: Η εντολή πολλαπλασιασμού-πρόσθεσης του Power PC

3.27 <§§3.3, 3.4, 3.5> Αν $x = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0101\ 1011_{two}$ και $y = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101_{two}$ αναπαριστούν προσημασμένους ακεραίους σε συμπλήρωμα ως προς δύο, εκτελέστε τις παρακάτω πράξεις αναλύοντας τα βήματα:

α. $x + y$
β. $x - y$
γ. $x * y$
δ. x/y

3.28 [20] <§§3.3, 3.4, 3.5> Εκτελέστε τις ίδιες πράξεις (λειτουργίες) όπως στην Άσκηση 3.27, αλλά με $x = 1111\ 1111\ 1111\ 1111\ 1011\ 0011\ 0101\ 0011_{two}$ και $y = 0000\ 0000\ 0000\ 0000\ 0000\ 0010\ 1101\ 0111_{two}$.

3.29 [30] <§3.5> Ο αλγόριθμος διαίρεσης στην Εικόνα 3.11 της σελίδας 203 ονομάζεται *διαίρεση με επαναφορά* (restoring division) επειδή, κάθε φορά που το αποτέλεσμα της αφαίρεσης του διαιρέτη από το διαιρετέο είναι αρνητικό, πρέπει να προσθέσετε πάλι το διαιρέτη στο διαιρετέο ώστε να επαναφέρετε την αρχική τιμή. Θυμηθείτε ότι η αριστερή ολίσθηση είναι ίδια με τον πολλαπλασιασμό με το 2. Ας δούμε την τιμή του αριστερού μισού του υπολοίπου ξανά, αρχίζοντας από το βήμα 3β του αλγορίθμου διαίρεσης και μετά πηγαίνοντας στο βήμα 2:

$$(\text{Υπόλοιπο} + \text{Διαιρέτης}) \times 2 - \text{Διαιρέτης}$$

Αυτή η τιμή δημιουργείται από την επαναφορά του υπολοίπου μετά από την πρόσθεση του διαιρέτη, την αριστερή ολίσθηση του αποτελέσματος, και την αφαίρεση του διαιρέτη. Απλοποιώντας το αποτέλεσμα παίρνουμε

$$\text{Υπόλοιπο} \times 2 + \text{Διαιρέτης} \times 2 - \text{Διαιρέτης} = \text{Υπόλοιπο} \times 2 + \text{Διαιρέτης}$$

Με βάση αυτή την παρατήρηση, γράψτε έναν αλγόριθμο *διαίρεσης χωρίς επαναφορά* (nonrestoring division) χρησιμοποιώντας τη σημειογραφία της Εικόνας 3.11 που δεν προσθέτει το διαιρέτη στο υπόλοιπο στο βήμα 3β. Δείξτε ότι ο αλγόριθμός σας δουλεύει, διαιρώντας το $0000\ 1011_{two}$ με το 0011_{two}.

3.30 [15] <§§3.2, 3.6> Η ενότητα «Συνολική εικόνα» στη σελίδα 234 αναφέρει ότι τα bit δεν έχουν εσωτερική σημασία. Με δεδομένη την παρακάτω σειρά των bit:

```
1010 1101 0001 0000 0000 0000 0000 0010
```

τι αναπαριστά αυτή, αν υποθέσουμε ότι είναι

α. ένας ακέραιος σε συμπλήρωμα ως προς δύο;
β. ένας απρόσημος ακέραιος;
γ. ένας αριθμός κινητής υποδιαστολής απλής ακρίβειας;
δ. μια εντολή MIPS;

Μπορεί να σας φανούν χρήσιμες οι Εικόνες 3.20 (σελίδα 226) και B A.10.2 (σελίδα 118).

3.31 <§§3.2, 3.6> Αυτή η άσκηση είναι παρόμοια με την Άσκηση 3.30, αλλά αυτή τη φορά χρησιμοποιήστε την εξής σειρά bit:

```
0010 0100 1001 0010 0100 1001 0010 0100
```

3.32 [10] <§3.6> B Για περισσότερη εξάσκηση: Αναπαραστάσεις αριθμών κινητής υποδιαστολής

3.33 [10] <§3.6> B Για περισσότερη εξάσκηση: Αναπαραστάσεις αριθμών κινητής υποδιαστολής

3.34 [10] <§3.6> B Για περισσότερη εξάσκηση: Γραφή κώδικα MIPS για αριθμητικές πράξεις κινητής υποδιαστολής

3.35 [5] <§3.6> Προσθέστε το $2{,}85_{ten} \times 10^3$ στο $9{,}84_{ten} \times 10^4$, υποθέτοντας ότι έχετε μόνο τρία σημαντικά ψηφία, πρώτα με φρουρό (guard) και στρογγύλευση (round) και μετά χωρίς.

3.36 [5] <§3.6> Αυτή η άσκηση είναι παρόμοια με την Άσκηση 3.35, αλλά αυτή τη φορά χρησιμοποιήστε τους αριθμούς $3{,}63_{ten} \times 10^4$ και $6{,}87_{ten} \times 10^3$.

3.37 [5] <§3.6> Δείξτε τη δυαδική αναπαράσταση κατά IEEE 754 για τον αριθμό κινητής υποδιαστολής 20_{ten} σε απλή και διπλή ακρίβεια.

3.38 [5] <§3.6> Αυτή η άσκηση είναι παρόμοια με την Άσκηση 3.37, αλλά αυτή τη φορά αντικαταστήστε τον αριθμό 20_{ten} με τον $20{,}5_{ten}$.

3.39 [10] <§3.6> Αυτή η άσκηση είναι παρόμοια με την Άσκηση 3.37, αλλά αυτή τη φορά αντικαταστήστε τον αριθμό 20_{ten} με τον $0{,}1_{ten}$.

3.40 [10] <§3.6> Αυτή η άσκηση είναι παρόμοια με την Άσκηση 3.37, αλλά αυτή τη φορά αντικαταστήστε τον αριθμό 20_{ten} με το δεκαδικό κλάσμα –5/6.

3.41 [10] <§3.6> Υποθέστε ότι εισάγουμε μια νέα εντολή που προσθέτει τρεις αριθμούς κινητής υποδιαστολής. Υποθέτοντας ότι τους προσθέτουμε μαζί με έναν τριπλό αθροιστή με φρουρό (guard), στρογγύλευση (round), και επίμονο bit (sticky bit), θα έχουμε εγγυημένα αποτελέσματα μέσα σε 1 ulp από τα αποτελέσματα με τη χρήση δύο ξεχωριστών εντολών πρόσθεσης;

3.42 [15] <§3.6> Αν x = 0100 0110 1101 1000 0000 0000 0000 0000_{two} και y = 1011 1110 1110 0000 0000 0000 0000 0000_{two} αναπαριστούν αριθμούς κινητής υποδιαστολής απλής ακρίβειας κατά IEEE 754, εκτελέστε τις παρακάτω πράξεις παρουσιάζοντας αναλυτικά όλα τα βήματα:

α. $x + y$
β. $x * y$

3.43 [15] <§3.6> Αν x = 0101 1111 1011 1110 0100 0000 0000 0000_{two}, y = 0011 1111 1111 1000 0000 0000 0000 0000_{two}, και z = 1101 1111 1011 1110 0100 0000 0000 0000_{two} αναπαριστούν αριθμούς κινητής υποδιαστολής απλής ακρίβειας κατά IEEE 754, εκτελέστε τις παρακάτω πράξεις παρουσιάζοντας αναλυτικά όλα τα βήματα:

α. $x + y$
β. (αποτέλεσμα του α) + z
γ. Γιατί αυτό το αποτέλεσμα είναι αντίθετο με τη διαίσθηση;

3.44 [20] <§§3.6, 3.7> Το πρότυπο κινητής υποδιαστολής IEEE 754 προσδιορίζει τη διπλή ακρίβεια των 64 bit με ένα σημαντικό 53 bit (μαζί με ένα υπονοούμενο 1) και έναν εκθέτη 11 bit. Η αρχιτεκτονική IA-32 προσφέρει μια επιλογή επεκτεταμένης ακρίβειας (extended precision) με ένα σημαντικό 64 bit και έναν εκθέτη 16 bit.

α. Υποθέτοντας ότι η επεκτεταμένη ακρίβεια είναι παρόμοια με την απλή και τη διπλή ακρίβεια, ποια είναι η πόλωση (bias) στον εκθέτη;
β. Ποιο είναι το εύρος αριθμών που μπορούν να αναπαρασταθούν με την επιλογή επεκτεταμένης ακρίβειας;
γ. Πόσο μεγαλύτερη είναι αυτή η ακρίβεια συγκρινόμενη με τη διπλή ακρίβεια;

3.45 [5] <§§3.6, 3.7> Η εσωτερική αναπαράσταση των αριθμών κινητής υποδιαστολής στην αρχιτεκτονική IA-32 έχει εύρος 80 bit. Αυτό περιλαμβάνει έναν εκθέτη των 16 bit. Ωστόσο, διαφημίζει επίσης και ένα σημαντικό των 64 bit. Πώς είναι αυτό δυνατόν;

3.46 [10] <§3.7> Ενώ η αρχιτεκτονική IA-32 επιτρέπει εσωτερικά αριθμούς κινητής υποδιαστολής των 80 bit, μόνον αριθμοί κινητής υποδιαστολής των 64 bit μπορούν να φορτωθούν ή να αποθηκευτούν. Ξεκινώντας μόνο με αριθμούς των 64 bit, πόσες πράξεις (λειτουργίες) απαιτούνται πριν χρησιμοποιηθεί το πλήρες εύρος των 80 bit; Δώστε ένα παράδειγμα.

3.47 [25] <§3.8> Για περισσότερη εξάσκηση: Κινητή υποδιαστολή σε αλγορίθμους

3.48 [30] <§3.8> Για περισσότερη εξάσκηση: Τρόποι στρογγυλοποίησης κινητής υποδιαστολής

3.49 [30] <§3.8> Για περισσότερη εξάσκηση: Μη κανονικοποιημένοι αριθμοί

3.50 [10] <§3.9> Για περισσότερη εξάσκηση: Αξιολόγηση συχνοτήτων εντολών

3.51 [10] <§3.9> Για περισσότερη εξάσκηση: Αξιολόγηση συχνοτήτων εντολών

3.52 [10] <§3.9> Για περισσότερη εξάσκηση: Αξιολόγηση συχνοτήτων εντολών

3.53 [10] <§3.9> Για περισσότερη εξάσκηση: Αξιολόγηση συχνοτήτων εντολών

3.54 [15] <§3.9> Για περισσότερη εξάσκηση: Αξιολόγηση της απόδοσης

3.55 [15] <§3.9> Για περισσότερη εξάσκηση: Αξιολόγηση της απόδοσης

Απαντήσεις στην αυτοεξέταση

§3.2, σελίδα 186: 3, επειδή κάθε χαρακτήρας σε μια συμβολοσειρά της Java καταλαμβάνει 16 bit συν μία λέξη για το μήκος.
§3.3, σελίδα 192: 2.
§3.6, σελίδα 235: 3.