Οι υπολογιστές στον πραγματικό κόσμο

Η Τεχνολογία των Πληροφοριών για τα 4 δισεκατομμύρια που δεν τη διαθέτουν

Σε όλο το βιβλίο θα δείτε ενότητες με τον τίτλο «Οι υπολογιστές στον πραγματικό κόσμο». Αυτές οι ενότητες περιγράφουν συναρπαστικές χρήσεις των υπολογιστών έξω από τις τυπικές τους λειτουργίες στον αυτοματισμό γραφείου και την επεξεργασία δεδομένων. Στόχος αυτών των ενοτήτων είναι να αναδείξουν την ποικιλομορφία των χρήσεων της τεχνολογίας των πληροφοριών (information technology).

Πρόβλημα προς λύση: Να γίνει η τεχνολογία των πληροφοριών διαθέσιμη και στην υπόλοιπη ανθρωπότητα, όπως οι αγρότες στα χωριά, πέρα από ένα πολυγλωσσικό σύνολο χαρακτήρων όπως το Unicode.

Λύση: Να αναπτυχθεί υπολογιστής, λογισμικό, και ένα σύστημα επικοινωνίας για ένα αγροτικό χωριό. Ωστόσο, δεν υπάρχει καθόλου ηλεκτρισμός, καθόλου τηλέφωνο, καθόλου τεχνική υποστήριξη, και οι χωρικοί δεν ξέρουν να διαβάσουν Αγγλικά.

Το Ίδρυμα Jhai αποδέχθηκε αυτή τη πρόκληση για πέντε χωριά της επαρχίας Hin Heup του Λάος. Αυτός ο αμερικανολαοτινός οργανισμός ιδρύθηκε με στόχο την ανύψωση του βιοτικού επιπέδου στην ύπαιθρο του Λάος μέσω της ανάπτυξης μιας εξαγωγικής οικονομίας. Επίσης έκτισε σπίτια, κατασκεύασε πηγάδια, και ίδρυσε έναν υφαντουργικό συνεταιρισμό. Όταν ρωτήθηκαν ποιο είναι το επόμενο πράγμα που θα ήθελαν, οι χωρικοί απάντησαν ότι ήθελαν πρόσβαση στο Διαδίκτυο! Πρώτον, ήθελαν να μαθαίνουν τις τιμές πριν μεταφέρουν τη σοδειά τους στην πλησιέστερη αγορά, που βρίσκεται 35 χιλιόμετρα μακριά. Θα μπορούσαν επίσης να μάθουν για την αγορά του εξωτερικού ώστε να πάρουν καλύτερες αποφάσεις σχετικά με το τι προϊόντα να καλλιεργήσουν, και για να αυξήσουν την διαπραγματευτική τους ικανότητα όταν είναι καιρός να τα πουλήσουν. Δεύτερον, ήθελαν να χρησιμοποιήσουν τηλεφωνία μέσω Διαδικτύου για να μιλούν με συγγενείς τους στο Λάος και στο εξωτερικό.

Ο στόχος ήταν «ένας ανθεκτικός υπολογιστής και εκτυπωτής, συναρμολογημένοι από εξαρτήματα του εμπορίου, που να καταναλώνει λιγότερα από 20 βατ σε κανονική χρήση — λιγότερα από 70 βατ όταν ο εκτυπωτής τυπώνει — και να αντέχει στη βρωμιά, τη ζέστη, και το νερό».

Το αποτέλεσμα ήταν η σχεδίαση του Jhai PC που χρησιμοποιεί μνήμη φλας (flash memory) αντί για μονάδα δίσκου, εξαλείφοντας έτσι τα κινούμενα μέρη από το PC ώστε να είναι πιο ανθεκτικό και ευκολότερο στη συντήρηση. Αντί για μια ενεργοβόρο οθόνη καθοδικού σωλήνα, έχει μια οθόνη υγρών κρυστάλλων. Για μειωμένο κόστος και κατανάλωση ρεύματος, χρησιμοποιεί μικροεπεξεργαστή 80486. Η ισχύς παρέχεται από μια μπαταρία αυτοκινήτου, η οποία

Ένας αγρότης τού Λάος που ήθελε πρόσβαση στο Διαδίκτυο.

μπορεί να φορτίζεται με ένα σύστημα πεταλιών και αλυσίδας ποδηλάτου. Ένας παλιός εκτυπωτής ακίδων (dot matrix printer) ολοκληρώνει το υλικό, φτάνοντας το κόστος περίπου στα 400 δολάρια. Το λειτουργικό σύστημα είναι Linux, και οι εφαρμογές είναι λογιστικά, ηλεκτρονικό ταχυδρομείο, και γραφή επιστολών, οι οποίες προσαρμόζονται στη γλώσσα του Λάος από μετανάστες.

Η επικοινωνιακή λύση είναι η υιοθέτηση ασύρματου δικτύου WiFi (IEEE 802.11b, δείτε το Κεφάλαιο 8). Το σχέδιο προβλέπει την ενίσχυση του σήματος με μεγαλύτερες κεραίες και την τοποθέτηση σταθμών επαναληπτών (repeater stations) στις κορυφές των λόφων ανάμεσα στο χωριό και την πόλη. Αυτοί οι επαναλήπτες τροφοδοτούνται από ηλιακές κυψέλες. Στο άλλο άκρο συνδέεται με το τοπικό τηλεφωνικό δίκτυο, κάτι που ολοκληρώνει τη σύνδεση με το Διαδίκτυο. Είκοσι πέντε εθελοντές στη Silicon Valley αναπτύσσουν αυτό το δίκτυο των Jhai PC.

Μια εναλλακτική προσπάθεια είναι ο *simputer*, που σημαίνει «απλός, φθηνός, πολυγλωσσικός υπολογιστής» (simple, inexpensive, multilingual computer). Ινδοί επιστήμονες υπολογιστών σχεδίασαν αυτόν τον ψηφιακό προσωπικό βοηθό (personal digital assistant — PDA), που είναι παρόμοιος με τον Palm Pilot, με στόχο να καλύπτει τις ανάγκες των χωρικών σε χώρες του τρίτου κόσμου. Η είσοδος δίνεται μέσω μιας οθόνης αφής και αναγνώρισης φωνής ώστε οι άνθρωποι να μη χρειάζεται να ξέρουν γραφή για να τον χρησιμοποιούν. Χρησιμοποιεί τρεις μπαταρίες AAA, που διαρκούν 3 έως 4 ώρες. Το κόστος είναι 250 δολάρια, και δεν υπάρχει καμία ειδική λύση για την επικοινωνία. Δεν είναι σαφές αν οι αγρότες στον αναπτυσσόμενο κόσμο θα ξόδευαν 250 δολάρια για ένα PDA, όταν ακόμη και οι μπαταρίες αποτελούν πολυτέλεια.

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

«Making the Web world-wide», *The Economist*, 26 Σεπτεμβρίου, 2002, www.jhai.org/economist

The Jhai Foundation, www.jhai.org/

«Computers for the Third World», *Scientific American*, Οκτώβριος 2002

Ινδός χωρικός ενώ χρησιμοποιεί τον simputer.

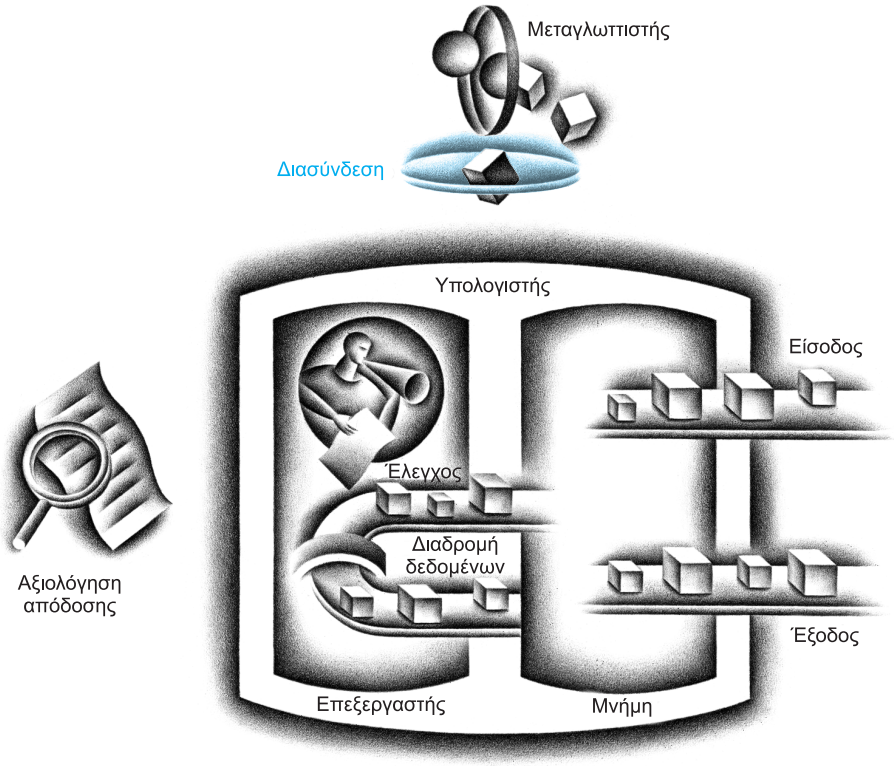
2

Εντολές: η γλώσσα του υπολογιστή

Μιλώ Ισπανικά στο Θεό,
Ιταλικά στις γυναίκες,
Γαλλικά στους άνδρες
και Γερμανικά στο άλογό μου.

Κάρολος 5ος, Βασιλιάς της Γαλλίας
1337–1380

Τα πέντε κλασικά συστατικά ενός υπολογιστή



2.1 Εισαγωγή

σύνολο εντολών (instruction set) Το λεξιλόγιο των διαταγών (commands) που είναι κατανοητές από μια συγκεκριμένη αρχιτεκτονική.

Για να διατάξεις το υλικό ενός υπολογιστή, πρέπει να μιλάς τη γλώσσα του. Οι λέξεις της γλώσσας ενός υπολογιστή ονομάζονται *εντολές (*instructions*)* και το λεξιλόγιό του ονομάζεται **σύνολο εντολών** (instruction set). Στο κεφάλαιο αυτό, θα δείτε το σύνολο εντολών ενός πραγματικού υπολογιστή, τόσο στη μορφή που γράφεται από τους ανθρώπους όσο και στη μορφή που διαβάζεται από τον υπολογιστή. Εισάγουμε τις εντολές με έναν αναλυτικό τρόπο («από επάνω προς τα κάτω» — top-down). Ξεκινώντας από μια σημειογραφία που μοιάζει σαν μια περιορισμένη γλώσσα προγραμματισμού, αποσαφηνίζουμε την περιγραφή βήμα προς βήμα μέχρι να δείτε την πραγματική γλώσσα ενός πραγματικού υπολογιστή. Το Κεφάλαιο 3 συνεχίζει την κατάβαση στα ενδότερα, αποκαλύπτοντας την αναπαράσταση των ακεραίων και των αριθμών κινητής υποδιαστολής και το υλικό που επενεργεί σε αυτούς τους αριθμούς.

Μπορεί να νομίσετε ότι οι γλώσσες των υπολογιστών ποικίλουν όσο αυτές των ανθρώπων, αλλά στην πραγματικότητα οι γλώσσες των υπολογιστών είναι αρκετά όμοιες, μοιάζοντας περισσότερο με τοπικές διαλέκτους παρά με ανεξάρτητες γλώσσες. Έτσι, όταν μάθεις μία είναι εύκολο να μάθεις και άλλες. Αυτή η ομοιότητα υπάρχει επειδή όλοι οι υπολογιστές κατασκευάζονται από τεχνολογίες υλικού βασισμένες σε παρόμοιες θεμελιώδεις αρχές και επειδή υπάρχουν μερικές βασικές λειτουργίες που όλοι οι υπολογιστές πρέπει να παρέχουν. Επιπλέον, οι σχεδιαστές υπολογιστών έχουν έναν κοινό στόχο: να βρουν μια γλώσσα που να κάνει εύκολη τη δόμηση του υλικού και του μεταγλωττιστή ενώ μεγιστοποιεί την απόδοση και ελαχιστοποιεί το κόστος. Ο στόχος αυτός έχει αποδείξει την αξία του στο χρόνο· το κείμενο που ακολουθεί γράφηκε πριν από την εποχή που μπορούσε κανείς να αγοράσει υπολογιστή, και ισχύει σήμερα όπως και το 1947:

> *Είναι εύκολο να δει κανείς με τυπικές-λογικές μεθόδους ότι υπάρχουν συγκεκριμένα [σύνολα εντολών] που σε αφαιρετικό επίπεδο είναι αρκετά για να ελέγξουν και να προκαλέσουν την εκτέλεση οποιασδήποτε ακολουθίας πράξεων. . . . Τα πραγματικά κρίσιμα ζητήματα, από τη σημερινή οπτική γωνία, στην επιλογή ενός [συνόλου εντολών] είναι περισσότερο πρακτικής φύσης: η απλότητα του εξοπλισμού που απαιτείται [από το σύνολο εντολών] και η καθαρότητα της εφαρμογής του στα πραγματικά σημαντικά προβλήματα, μαζί με την ταχύτητα που μπορεί να χειριστεί αυτά τα προβλήματα.*
>
> Burks, Goldstine, και von Neumann, 1947

Η «απλότητα του εξοπλισμού» είναι τόσο πολύτιμο θέμα για τους υπολογιστές της δεκαετίας του 2000 όσο ήταν και γι' αυτούς της δεκαετίας του 1950. Ο στόχος του κεφαλαίου αυτού είναι να διδάξει ένα σύνολο εντολών που ακολουθεί αυτή τη συμβουλή, δείχνοντας τόσο πώς αυτό αναπαρίσταται στο υλικό όσο και τη σχέση μεταξύ γλωσσών προγραμματισμού υψηλού επιπέδου και αυτής της πιο απλής γλώσσας. Τα παραδείγματά μας είναι στη γλώσσα προγραμματισμού C· η Ενότητα 2.14 δείχνει πώς θα άλλαζαν αυτά για μια αντικειμενοστρεφή (object-oriented) γλώσσα όπως η Java.

Μαθαίνοντας πώς να αναπαριστάτε τις εντολές, θα ανακαλύψετε επίσης το μυστικό της υπολογιστικής: την **έννοια του αποθηκευμένου προγράμματος** (stored-program concept). Επιπλέον, θα εξασκήσετε τις ικανότητές σας στις «ξένες γλώσσες» γράφοντας προγράμματα στη γλώσσα του υπολογιστή και εκτελώντας τα στον προσομοιωτή που συνοδεύει το βιβλίο. Θα δείτε επίσης την επίδραση των γλωσσών προγραμματισμού και της βελτιστοποίησης των μεταγλωττιστών στην απόδοση. Ολοκληρώνουμε το κεφάλαιο με μια ματιά στην ιστορική εξέλιξη των συνόλων εντολών, και μια παρουσίαση άλλων διαλέκτων υπολογιστών.

έννοια του αποθηκευμένου προγράμματος (stored-program concept) Η ιδέα ότι εντολές και δεδομένα πολλών τύπων μπορούν να αποθηκευτούν στη μνήμη ως αριθμοί, που οδήγησε στον υπολογιστή αποθηκευμένου προγράμματος (stored program computer).

Το σύνολο εντολών που επιλέξαμε προέρχεται από τον MIPS, που είναι τυπικό παράδειγμα των συνόλων εντολών που σχεδιάστηκαν από τη δεκαετία του 1980 μέχρι σήμερα. Σχεδόν 100 εκατομμύρια αυτών των δημοφιλών μικροεπεξεργαστών κατασκευάστηκαν το 2002 και βρίσκονται σε προϊόντα των εταιρειών ATI Technologies, Broadcom, Cisco, NEC, Nintendo, Silicon Graphics, Sony, Texas Instruments, και Toshiba, μεταξύ άλλων.

Αποκαλύπτουμε το σύνολο εντολών MIPS κομμάτι-κομμάτι, δίνοντας το σκεπτικό μαζί με τις δομές των υπολογιστών. Αυτή η αναλυτική (από επάνω προς τα κάτω), βήμα-βήμα διδασκαλία «πλέκει» τα συστατικά με τις επεξηγήσεις τους, κάνοντας τη συμβολική γλώσσα πιο προσιτή. Για να διατηρήσουμε τη συνολική εικόνα στο μυαλό μας, κάθε ενότητα τελειώνει με μια εικόνα που συνοψίζει το σύνολο εντολών τού MIPS που έχει αποκαλυφθεί μέχρι το σημείο εκείνο, τονίζοντας τα τμήματα που παρουσιάστηκαν στη συγκεκριμένη ενότητα.

2.2 Λειτουργίες του υλικού των υπολογιστών

Ασφαλώς πρέπει να υπάρχουν εντολές για την εκτέλεση των θεμελιωδών αριθμητικών πράξεων.

Burks, Goldstine, και von Neumann, 1947

Κάθε υπολογιστής πρέπει να είναι ικανός να εκτελεί αριθμητικές πράξεις. Η σημειογραφία της συμβολικής γλώσσας (assembly) του MIPS

```
add a, b, c
```

καθοδηγεί έναν υπολογιστή να προσθέσει τις δύο μεταβλητές `b` και `c` και να τοποθετήσει το άθροισμά τους στη μεταβλητή `a`.

Η σημειογραφία αυτή είναι αυστηρή, ως προς το ότι κάθε αριθμητική εντολή του MIPS εκτελεί μόνο μία πράξη (λειτουργία) και πρέπει πάντα να έχει ακριβώς τρεις μεταβλητές. Για παράδειγμα, υποθέστε ότι θέλουμε να βάλουμε το άθροισμα των μεταβλητών `b`, `c`, `d`, και `e` στη μεταβλητή `a`. (Στην ενότητα αυτή είμαστε σκόπιμα ασαφείς για το τι είναι μια «μεταβλητή»· στην επόμενη ενότητα θα το εξηγήσουμε αναλυτικά.)

Η παρακάτω ακολουθία εντολών προσθέτει τις τέσσερις μεταβλητές:

```
add a, b, c # το άθροισμα των b και c τοποθετείται στην a.
add a, a, d # το άθροισμα των b, c, και d είναι τώρα στην a.
add a, a, e # το άθροισμα των b, c, d, και e είναι τώρα στην a.
```

Έτσι, χρειάζονται τρεις εντολές για να υπολογιστεί το άθροισμα τεσσάρων μεταβλητών.

Οι λέξεις δεξιά από το σύμβολο της δίεσης (#) σε κάθε γραμμή είναι *σχόλια* (comments) για τον αναγνώστη, και ο υπολογιστής τα αγνοεί. Σημειώστε ότι, σε αντίθεση με άλλες γλώσσες προγραμματισμού, κάθε γραμμή στη γλώσσα αυτή μπορεί να περιέχει το πολύ μία εντολή. Μια άλλη διαφορά από τη C είναι ότι τα σχόλια τερματίζονται στο τέλος μιας γραμμής.

Ο φυσιολογικός αριθμός τελεστέων (operands) για μια πράξη όπως η πρόσθεση είναι τρεις: οι δύο αριθμοί που προστίθενται και μία θέση για την τοποθέτηση του αθροίσματος. Η απαίτηση να έχει κάθε εντολή ακριβώς τρεις τελεστέους, ούτε περισσότερους ούτε λιγότερους, συμμορφώνεται με τη φιλοσοφία της διατήρησης της απλότητας του υλικού: το υλικό για μεταβλητό αριθμό τελεστέων είναι πιο πολύπλοκο από το υλικό για σταθερό αριθμό. Η κατάσταση αυτή απεικονίζει την πρώτη από τέσσερις θεμελιώδεις αρχές της σχεδίασης του υλικού:

Σχεδιαστική Αρχή 1: Η απλότητα ευνοεί την κανονικότητα.

Μπορούμε τώρα να δείξουμε, στα δύο παραδείγματα που ακολουθούν, τη σχέση των προγραμμάτων που είναι γραμμένα σε γλώσσες προγραμματισμού υψηλού επιπέδου με τα προγράμματα σε αυτή την πιο πρωτόγονη σημειογραφία.

ΠΑΡΑΔΕΙΓΜΑ

Μεταγλώττιση δύο εντολών ανάθεσης τιμής C σε MIPS

Αυτό το τμήμα ενός προγράμματος C περιέχει τις πέντε μεταβλητές `a`, `b`, `c`, `d`, και `e`. Αφού και η Java αποτελεί εξέλιξη της C, το παράδειγμα αυτό και τα λίγα επόμενα δουλεύουν για οποιαδήποτε από τις δύο γλώσσες προγραμματισμού υψηλού επιπέδου:

```
a = b + c;
d = a - e;
```

Η μετάφραση από τη C σε εντολές της συμβολικής γλώσσας MIPS εκτελείται από το *μεταγλωττιστή* (compiler). Δείξτε τον κώδικα MIPS που παράγεται από ένα μεταγλωττιστή.

ΑΠΑΝΤΗΣΗ

Μια εντολή MIPS λειτουργεί σε δύο τελεστέους προέλευσης (source operands) και τοποθετεί το αποτέλεσμα σε έναν τελεστέο προορισμού (destination operand). Έτσι, οι δύο απλές παραπάνω εντολές μεταγλωττίζονται απευθείας σε αυτές τις δύο εντολές συμβολικής γλώσσας MIPS:

```
add a, b, c
sub d, a, e
```

Μεταγλώττιση μιας σύνθετης ανάθεσης τιμής της C σε MIPS

ΠΑΡΑΔΕΙΓΜΑ

Μια κάπως σύνθετη εντολή περιέχει τις πέντε μεταβλητές `f`, `g`, `h`, `i`, και `j`:

```
f = (g + h) - (i + j);
```

Τι θα μπορούσε να παραγάγει ένας μεταγλωττιστής τής C;

ΑΠΑΝΤΗΣΗ

Ο μεταγλωττιστής πρέπει να διαιρέσει την εντολή αυτή σε πολλές εντολές συμβολικής γλώσσας, αφού εκτελείται μόνο μία πράξη (λειτουργία) ανά εντολή του MIPS. Η πρώτη εντολή MIPS υπολογίζει το άθροισμα των `g` και `h`. Επειδή πρέπει να βάλουμε κάπου το αποτέλεσμα, ο μεταγλωττιστής δημιουργεί μια προσωρινή μεταβλητή που ονομάζεται `t0`:

```
add t0, g, h # η προσωρινή μεταβλητή t0 περιέχει το g + h
```

Παρόλο που η επόμενη πράξη είναι αφαίρεση, πρέπει να υπολογίσουμε πρώτα το άθροισμα των `i` και `j` για να μπορέσουμε να αφαιρέσουμε. Έτσι, η δεύτερη εντολή τοποθετεί το άθροισμα του `i` και του `j` σε μια άλλη προσωρινή μεταβλητή που δημιουργείται από το μεταγλωττιστή και ονομάζεται `t1`:

```
add t1, i, j # η προσωρινή μεταβλητή t1 περιέχει το i + j
```

Τέλος, η εντολή αφαίρεσης αφαιρεί το δεύτερο άθροισμα από το πρώτο και τοποθετεί τη διαφορά στη μεταβλητή `f`, ολοκληρώνοντας το μεταγλωττισμένο κώδικα:

```
sub f, t0, t1 # το f παίρνει το t0 - t1, που είναι το
              # (g + h) - (i + j)
```

Η Εικόνα 2.1 συνοψίζει τα τμήματα της συμβολικής γλώσσας MIPS που περιγράψαμε σε αυτή την ενότητα. Αυτές οι εντολές είναι συμβολικές αναπαραστάσεις αυτού που στην πραγματικότητα καταλαβαίνει ο επεξεργαστής MIPS. Στις λίγες επόμενες ενότητες, θα εξελίξουμε αυτή τη συμβολική αναπαράσταση στη πραγματική γλώσσα του MIPS, με το κάθε βήμα να κάνει τη συμβολική αναπαράσταση πιο συμπαγή.

Συμβολική γλώσσα MIPS

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Αριθμητικές πράξεις	add	`add a, b, c`	`a = b + c`	Πάντα τρεις τελεστέοι
	subtract	`sub a, b, c`	`a = b - c`	Πάντα τρεις τελεστέοι

ΕΙΚΟΝΑ 2.1 Αρχιτεκτονική του MIPS που έχουμε αναλύσει στην Ενότητα 2.2. Οι πραγματικοί τελεστέοι του υπολογιστή θα αποκαλυφθούν στην επόμενη ενότητα. Τα επισημασμένα μέρη σε τέτοιες συνόψεις δείχνουν τις δομές της συμβολικής γλώσσας του MIPS που παρουσιάσαμε στη συγκεκριμένη ενότητα· στην πρώτη αυτή εικόνα, όλα είναι καινούργια.

Αυτοεξέταση

Για μια δεδομένη συνάρτηση, ποια γλώσσα προγραμματισμού είναι πιθανό να χρειάζεται τις περισσότερες γραμμές κώδικα; Βάλτε στη σειρά τις εξής τρεις αναπαραστάσεις.

1. Java
2. C
3. Συμβολική γλώσσα MIPS

Επιπλέον ανάπτυξη: Για να αυξήσει τη φορητότητα, η Java αρχικά είχε στόχο να βασίζεται σε ένα διερμηνευτή λογισμικού (software interpreter). Το σύνολο εντολών αυτού του διερμηνευτή ονομάζεται *κώδικες byte της Java* (Java bytecodes) και είναι πολύ διαφορετικό από το σύνολο εντολών του MIPS. Για να φτάσει η απόδοση κοντά στο ισοδύναμο πρόγραμμα της C, τα συστήματα Java σήμερα τυπικά μεταγλωττίζουν τους κώδικες byte σε εγγενή (native) σύνολα εντολών όπως του MIPS. Επειδή αυτή η μεταγλώττιση γίνεται φυσιολογικά πολύ αργότερα από ό,τι στα προγράμματα της C, αυτοί οι μεταγλωττιστές Java συχνά λέγονται μεταγλωττιστές *«ακριβώς στην ώρα»* (Just-In-Time — JIT). Η Ενότητα 2.10 δείχνει πώς οι μεταγλωττιστές JIT χρησιμοποιούνται μετά από τους μεταγλωττιστές C στη διαδικασία εκκίνησης, και η ενότητα 2.13 δείχνει τις επιπτώσεις που έχει στην απόδοση η μεταγλώττιση και η διερμηνεία των προγραμμάτων Java. Τα παραδείγματα της Java στο κεφάλαιο αυτό παραλείπουν το βήμα των κωδίκων byte Java και δείχνουν μόνο τον κώδικα MIPS που παράγεται από ένα μεταγλωττιστή.

2.3 Τελεστέοι υλικού των υπολογιστών

Σε αντίθεση με τα προγράμματα γλωσσών υψηλού επιπέδου, οι τελεστέοι (operands) των αριθμητικών εντολών είναι περιορισμένοι· πρέπει να προέρχονται από έναν περιορισμένο αριθμό ειδικών θέσεων που κατασκευάζονται απευθείας στο υλικό και ονομάζονται *καταχωρητές* (registers). Οι καταχωρητές είναι τα δομικά στοιχεία της κατασκευής των υπολογιστών: είναι τα αρχέτυπα ή θεμελιώδη στοιχεία (primitives) που χρησιμοποιούνται στη σχεδίαση του υλικού και τα οποία είναι επίσης ορατά στον προγραμματιστή όταν ο υπολογιστής ολοκληρωθεί. Το μέγεθος ενός καταχωρητή στην αρχιτεκτονική MIPS είναι 32 bit· ομάδες των 32 bit εμφανίζονται τόσο συχνά ώστε τους έχει δοθεί το όνομα **λέξη** (word) στην αρχιτεκτονική MIPS.

λέξη (word) Η φυσική μονάδα προσπέλασης σε έναν υπολογιστή, συνήθως μια ομάδα από 32 bit· αντιστοιχεί στο μέγεθος ενός καταχωρητή στην αρχιτεκτονική MIPS.

Μια σημαντική διαφορά μεταξύ των μεταβλητών μιας γλώσσας προγραμματισμού και των καταχωρητών είναι ο περιορισμένος αριθμός των καταχωρητών, τυπικά 32 στους σημερινούς υπολογιστές. Ο MIPS έχει 32 καταχωρητές. (Σχετικά με την ιστορία του αριθμού των καταχωρητών δείτε την Ενότητα 2.19.) Έτσι, συνεχίζοντας την αναλυτική, βήμα προς βήμα ανάπτυξη της συμβολικής αναπαράστασης της γλώσσας του MIPS, στην ενότητα αυτή έχουμε προσθέσει τον περιορισμό ότι καθένας από τους τρεις τελεστέους των αριθμητικών εντολών του MIPS πρέπει να επιλέγεται από έναν από τους 32 καταχωρητές των 32 bit.

Η αιτία για το όριο των 32 καταχωρητών μπορεί να βρεθεί στη δεύτερη από τις τέσσερις θεμελιώδεις αρχές της τεχνολογίας υλικού:

Σχεδιαστική Αρχή 2: Το μικρότερο είναι ταχύτερο.

Ένας πολύ μεγάλος αριθμός καταχωρητών μπορεί να αυξήσει το χρόνο κύκλου ρολογιού απλώς επειδή τα ηλεκτρονικά σήματα αργούν περισσότερο όταν πρέπει να ταξιδέψουν μακρύτερα.

Γενικές οδηγίες όπως «το μικρότερο είναι ταχύτερο» δεν είναι θέσφατα· 31 καταχωρητές μπορεί να μην είναι ταχύτεροι από τους 32. Ωστόσο, η αλήθεια πίσω από τέτοιες παρατηρήσεις κάνει τους σχεδιαστές υπολογιστών να τις λαμβάνουν σοβαρά υπόψη τους. Στην περίπτωση αυτή, ο σχεδιαστής πρέπει να εξισορροπήσει τη σφοδρή επιθυμία των προγραμμάτων για περισσότερους καταχωρητές με την επιθυμία του σχεδιαστή να κρατήσει τον κύκλο ρολογιού σύντομο. Μια άλλη αιτία για τη χρήση όχι περισσότερων από 32 καταχωρητών είναι το πλήθος των bit που θα χρειαζόταν στη μορφή των εντολών (instruction format), όπως δείχνει η Ενότητα 2.4.

Τα Κεφάλαια 5 και 6 δείχνουν τον κεντρικό ρόλο που παίζουν οι καταχωρητές στην κατασκευή του υλικού· όπως θα δούμε στα κεφάλαια αυτά, η αποτελεσματική χρήση των καταχωρητών είναι κλειδί για την απόδοση των προγραμμάτων.

Παρόλο που θα μπορούσαμε να γράψουμε απλώς εντολές χρησιμοποιώντας για τους καταχωρητές αριθμούς από το 0 έως το 31, η σύμβαση στο MIPS για την αναπαράσταση ενός καταχωρητή είναι να χρησιμοποιούνται ονόματα δύο χαρακτήρων που ακολουθούν το σύμβολο του δολαρίου. Η Ενότητα 2.7 θα εξηγήσει του λόγους που κρύβονται πίσω από αυτά τα ονόματα. Προς το παρόν, θα χρησιμοποιήσουμε τα ονόματα `$s0`, `$s1`, ... για καταχωρητές που αντιστοιχούν σε μεταβλητές των προγραμμάτων C και Java, και τα `$t0`, `$t1`, ... για προσωρινούς καταχωρητές που χρειάζονται για τη μεταγλώττιση του προγράμματος σε εντολές MIPS.

Μεταγλώττιση ανάθεσης τιμής τής C με τη χρήση καταχωρητών

ΠΑΡΑΔΕΙΓΜΑ

Η συσχέτιση μεταβλητών του προγράμματος με καταχωρητές είναι καθήκον του μεταγλωττιστή. Πάρτε, για παράδειγμα, την εντολή ανάθεσης τιμής από το προηγούμενο παράδειγμά μας:

```
f = (g + h) - (i + j);
```

Οι μεταβλητές `f`, `g`, `h`, `i`, και `j` ανατίθενται στους καταχωρητές `$s0`, `$s1`, `$s2`, `$s3`, και `$s4`, αντίστοιχα. Ποιος είναι ο μεταγλωττισμένος κώδικας MIPS;

ΑΠΑΝΤΗΣΗ

Το μεταγλωττισμένο πρόγραμμα μοιάζει αρκετά με το προηγούμενο παράδειγμα, με τη διαφορά ότι αντικαθιστούμε τις μεταβλητές με τα ονόματα των καταχωρητών που αναφέρονται παραπάνω και δύο προσωρινούς καταχωρητές `$t0` και `$t1`, οι οποίοι αντιστοιχούν στις προσωρινές μεταβλητές παραπάνω:

```
add $t0, $s1, $s2 # ο καταχωρητής $t0 περιέχει το g + h
add $t1, $s3, $s4 # ο καταχωρητής $t1 περιέχει το i + j
sub $s0, $t0, $t1 # το f παίρνει το $t0 - $t1, που είναι το
                  # (g + h) - (i + j)
```

Τελεστέοι μνήμης

Οι γλώσσες προγραμματισμού έχουν απλές μεταβλητές που περιέχουν απλά στοιχεία δεδομένων όπως στα παραδείγματα αυτά, αλλά έχουν επίσης πιο πολύπλοκες δομές δεδομένων — πίνακες (arrays) και δομές (structures). Αυτές οι πολύπλοκες δομές δεδομένων μπορούν να περιέχουν πολύ περισσότερα στοιχεία δεδομένων από ό,τι οι καταχωρητές που υπάρχουν σε έναν υπολογιστή. Πώς μπορεί ένας υπολογιστής να αναπαραστήσει και να προσπελάσει τέτοιες μεγάλες δομές;

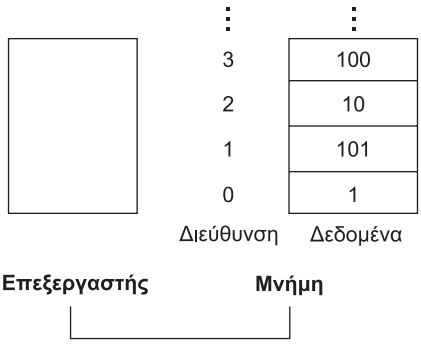
Θυμηθείτε τα πέντε συστατικά ενός υπολογιστή που παρουσιάσαμε στο Κεφάλαιο 1 και φαίνονται στη σελίδα 65. Ο επεξεργαστής μπορεί να διατηρήσει μόνον ένα μικρό μέρος των δεδομένων στους καταχωρητές, αλλά η μνήμη του περιέχει εκατομμύρια στοιχεία δεδομένων. Έτσι, οι δομές δεδομένων (οι πίνακες και οι δομές) διατηρούνται στη μνήμη.

Όπως εξηγήσαμε νωρίτερα, οι αριθμητικές πράξεις γίνονται μόνο σε καταχωρητές στις εντολές MIPS· έτσι, ο MIPS πρέπει να περιλαμβάνει εντολές που μεταφέρουν δεδομένα μεταξύ μνήμης και καταχωρητών. Τέτοιες εντολές ονομάζονται **εντολές μεταφοράς δεδομένων** (data transfer instructions). Για να προσπελαστεί μια λέξη στη μνήμη, η εντολή πρέπει να δώσει τη **διεύθυνση** (address) της μνήμης. Η μνήμη είναι απλώς ένας μεγάλος μονοδιάστατος πίνακας, όπου η διεύθυνση ενεργεί ως αριθμοδείκτης (index) στον πίνακα αυτόν, ξεκινώντας από το 0. Για παράδειγμα, στην Εικόνα 2.2, η διεύθυνση του τρίτου στοιχείου δεδομένων είναι 2, και η τιμή του στοιχείου Memory[2] είναι 10.

εντολή μεταφοράς δεδομένων (data transfer instruction) Μια διαταγή που μεταφέρει δεδομένα μεταξύ μνήμης και καταχωρητών.

διεύθυνση (address) Μια τιμή που χρησιμοποιείται για να καθορίσει τη θέση ενός συγκεκριμένου στοιχείου δεδομένων μέσα σε έναν πίνακα μνήμης.

Η εντολή μεταφοράς δεδομένων που αντιγράφει δεδομένα από τη μνήμη σε έναν καταχωρητή ονομάζεται παραδοσιακά *φόρτωση* (load). Η μορφή της εντολής φόρτωσης είναι το όνομα της λειτουργίας (πράξης) ακολουθούμενο από τον καταχωρητή ο οποίος θα φορτωθεί, και μετά μια σταθερά και έναν καταχωρητή



ΕΙΚΟΝΑ 2.2 Διευθύνσεις μνήμης και περιεχόμενα της μνήμης σε αυτές τις θέσεις. Αυτή είναι μια απλούστευση της διευθυνσιοδότησης (addressing) του MIPS· η Εικόνα 2.3 παρουσιάζει την πραγματική διευθυνσιοδότηση του MIPS για διαδοχικές διευθύνσεις μνήμης.

που χρησιμοποιούνται για την προσπέλαση της μνήμης. Το άθροισμα του σταθερού τμήματος της εντολής και των περιεχομένων του δεύτερου καταχωρητή σχηματίζει τη διεύθυνση της μνήμης. Το πραγματικό όνομα στον MIPS για την εντολή αυτή είναι `lw`, που αντιστοιχεί στο *load word* (φόρτωση λέξης).

Μεταγλώττιση ανάθεσης τιμής όταν ένας τελεστέος είναι στη μνήμη

ΠΑΡΑΔΕΙΓΜΑ

Ας υποθέσουμε ότι ο `A` είναι ένας πίνακας 100 λέξεων και ότι ο μεταγλωττιστής έχει συσχετίσει τις μεταβλητές `g` και `h` με τους καταχωρητές `$s1` και `$s2` όπως πριν. Ας υποθέσουμε επίσης ότι η αρχική διεύθυνση, ή *διεύθυνση βάσης* (base address) του πίνακα βρίσκεται στον καταχωρητή `$s3`. Μεταγλωττίστε την επόμενη εντολή ανάθεσης τιμής της C:

```
g = h + A[8];
```

ΑΠΑΝΤΗΣΗ

Παρόλο που υπάρχει μόνο μία πράξη σε αυτή την εντολή ανάθεσης τιμής, ο ένας από τους τελεστέους είναι στη μνήμη, οπότε πρέπει πρώτα να μεταφέρουμε το `A[8]` σε έναν καταχωρητή. Η διεύθυνση αυτού του στοιχείου του πίνακα είναι το άθροισμα της βάσης του πίνακα `A`, που βρίσκεται στον καταχωρητή `$s3`, συν τον αριθμό για την επιλογή του στοιχείου 8. Τα δεδομένα πρέπει να τοποθετηθούν σε έναν προσωρινό καταχωρητή για να χρησιμοποιηθούν στην επόμενη εντολή. Με βάση την Εικόνα 2.2, η πρώτη μεταγλωττισμένη εντολή είναι

```
lw  $t0,8($s3) # ο προσωρινός καταχωρητής $t0 παίρνει το A[8]
```

(Στην επόμενη σελίδα θα κάνουμε μια μικρή προσαρμογή στην εντολή αυτή, αλλά προς το παρόν θα χρησιμοποιήσουμε αυτή την απλοποιημένη έκδοση.) Η επόμενη εντολή μπορεί να επενεργήσει στην τιμή του καταχωρητή `$t0` (που είναι ίση με το `A[8]`) αφού βρίσκεται σε έναν καταχωρητή. Η εντολή πρέπει να προσθέσει το `h` (που περιέχεται στον `$s2`) στο `A[8]` (`$t0`), και να τοποθετήσει το άθροισμα στον καταχωρητή ο οποίος αντιστοιχεί στο `g` (που έχει συνδεθεί με τον `$s1`):

```
add  $s1,$s2,$t0 # g = h + A[8]
```

Η σταθερή τιμή σε μια εντολή μεταφοράς δεδομένων ονομάζεται *σχετική απόσταση* (offset) και ο καταχωρητής που προστίθεται για να σχηματιστεί η διεύθυνση ονομάζεται *καταχωρητής βάσης* (base register).

Διασύνδεση υλικού και λογισμικού

Εκτός από τη συσχέτιση μεταβλητών και καταχωρητών, ο μεταγλωττιστής κατανέμει δομές δεδομένων όπως πίνακες και δομές σε θέσεις στη μνήμη. Ο μεταγλωττιστής μπορεί κατόπιν να τοποθετήσει την κατάλληλη αρχική διεύθυνση στις εντολές μεταφοράς δεδομένων.

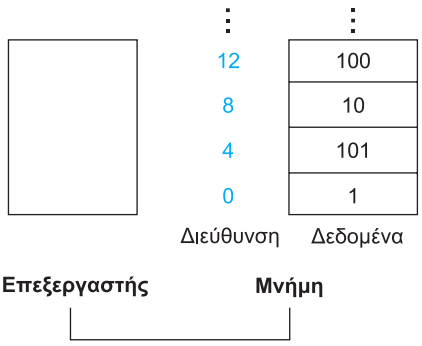
Εφόσον τα *byte* των 8 bit είναι χρήσιμα σε πολλά προγράμματα, οι περισσότερες αρχιτεκτονικές προσπελάζουν μεμονωμένα byte. Έτσι, η διεύθυνση μιας λέξης ταιριάζει με τη διεύθυνση ενός από τα 4 byte μέσα στη λέξη. Συνεπώς, οι διευθύνσεις διαδοχικών λέξεων διαφέρουν κατά 4. Για παράδειγμα, η Εικόνα 2.3 δείχνει τις πραγματικές διευθύνσεις MIPS για την Εικόνα 2.2· η διεύθυνση του byte της τρίτης λέξης είναι 8.

περιορισμός ευθυγράμμισης (alignment restriction) Η απαίτηση ώστε τα δεδομένα να ευθυγραμμίζονται στη μνήμη σε φυσικά όρια.

Στο MIPS, οι λέξεις πρέπει να ξεκινούν σε διευθύνσεις που είναι πολλαπλάσια του 4. Αυτή η απαίτηση ονομάζεται **περιορισμός ευθυγράμμισης** (alignment restriction), και τη διαθέτουν πολλές αρχιτεκτονικές. (Το Κεφάλαιο 5 εξηγεί γιατί η ευθυγράμμιση οδηγεί σε ταχύτερες μεταφορές δεδομένων.)

Οι υπολογιστές χωρίζονται σε αυτούς που χρησιμοποιούν τη διεύθυνση του αριστερότερου byte ή του byte στο «μεγάλο άκρο» (big end) ως διεύθυνση λέξης, και σε αυτούς που χρησιμοποιούν το δεξιότερο byte ή το byte στο «μικρό άκρο» (little end). Ο MIPS ανήκει στο στρατόπεδο του *Μεγάλου Άκρου* (Big Endian). (Το Παράρτημα Α, στη σελίδα 111, του δεύτερου τόμου δείχνει τις δύο επιλογές για την αρίθμηση των byte μιας λέξης.)

Η διευθυνσιοδότηση byte επηρεάζει επίσης τον αριθμοδείκτη (index) του πίνακα. Για να πάρουμε την κατάλληλη διεύθυνση byte στον παραπάνω κώδικα, *η σχετική απόσταση που θα προστεθεί στον καταχωρητή βάσης* `$s3` *πρέπει να είναι 4 × 8, ή 32*, ώστε η διεύθυνση φόρτωσης να επιλέξει το `A[8]` και όχι το `A[8/4]`. (Δείτε τη σχετική παγίδα στη σελίδα 162 της Ενότητας 2.17.)



ΕΙΚΟΝΑ 2.3 Πραγματικές διευθύνσεις μνήμης του MIPS και περιεχόμενα της μνήμης σε αυτές τις θέσεις. Οι διευθύνσεις που έχουν αλλάξει επισημαίνονται, ώστε να διακρίνονται σε σχέση με την Εικόνα 2.2. Επειδή ο MIPS προσπελάζει κάθε byte, οι διευθύνσεις λέξης είναι πολλαπλάσια του τέσσερα: υπάρχουν τέσσερα byte σε κάθε λέξη.

Η συμπληρωματική εντολή της φόρτωσης (load) ονομάζεται παραδοσιακά *αποθήκευση* (store)· αντιγράφει δεδομένα από έναν καταχωρητή στη μνήμη. Η μορφή της εντολής αποθήκευσης store είναι παρόμοια με αυτή της load: το όνομα της λειτουργίας, ακολουθούμενο από τον καταχωρητή ο οποίος θα αποθηκευτεί, τη σχετική απόσταση που θα επιλέξει ένα στοιχείο πίνακα, και τέλος τον καταχωρητή βάσης. Και πάλι, η διεύθυνση MIPS καθορίζεται κατά ένα μέρος από μια σταθερά και κατά ένα μέρος από τα περιεχόμενα ενός καταχωρητή. Το πραγματικό όνομα της εντολής στον MIPS είναι `sw`, που αντιστοιχεί στο *store word* (αποθήκευση λέξης).

Μεταγλώττιση με τη χρήση εντολών load και store

ΠΑΡΑΔΕΙΓΜΑ

Υποθέστε ότι η μεταβλητή `h` συσχετίζεται με τον καταχωρητή `$s2`, και ότι η διεύθυνση βάσης του πίνακα `A` περιέχεται στον καταχωρητή `$s3`. Ποιος είναι ο κώδικας της συμβολικής γλώσσας του MIPS για την παρακάτω εντολή ανάθεσης τιμής της C;

```
A[12] = h + A[8];
```

ΑΠΑΝΤΗΣΗ

Μολονότι υπάρχει μόνο μία πράξη σε αυτή την εντολή ανάθεσης τιμής της C, τώρα είναι στη μνήμη δύο από τους τελεστέους και, έτσι, χρειαζόμαστε ακόμη περισσότερες εντολές MIPS. Οι δύο πρώτες εντολές είναι οι ίδιες με το προηγούμενο παράδειγμα, εκτός του ότι αυτή τη φορά χρησιμοποιούμε την κατάλληλη σχετική απόσταση για τη διευθυνσιοδότηση του byte στην εντολή φόρτωσης λέξης load word ώστε να επιλέξουμε το `A[8]`, και η εντολή πρόσθεσης τοποθετεί το άθροισμα στον καταχωρητή `$t0`:

```
lw $t0,32($s3)   # ο προσωρινός καταχωρητής $t0 παίρνει
                 # το A[8]
add $t0,$s2,$t0  # ο προσωρινός καταχωρητής $t0 παίρνει
                 # το h + A[8]
```

Η τελευταία εντολή αποθηκεύει το άθροισμα στο στοιχείο `A[12]`, χρησιμοποιώντας το 48 ως σχετική απόσταση και τον καταχωρητή `$s3` ως καταχωρητή βάσης.

```
sw $t0,48($s3)   # αποθηκεύει το h + A[8] πίσω στο A[12]
```

Σταθεροί ή άμεσοι τελεστέοι

Πολλές φορές, ένα πρόγραμμα χρησιμοποιεί μια σταθερά σε μια πράξη (λειτουργία) — για παράδειγμα η αύξηση ενός αριθμοδείκτη ώστε να δείξει στο επόμενο στοιχείο ενός πίνακα. Για την ακρίβεια, όταν εκτελούμε τα μετροπρογράμματα SPEC2000 περισσότερες από τις μισές αριθμητικές εντολές του MIPS έχουν μια σταθερά ως τελεστέο.

Διασύνδεση υλικού και λογισμικού

Πολλά προγράμματα έχουν περισσότερες μεταβλητές από τους καταχωρητές που διαθέτουν οι υπολογιστές. Συνεπώς, ο μεταγλωττιστής προσπαθεί να κρατήσει τις πιο συχνά χρησιμοποιούμενες μεταβλητές σε καταχωρητές και τοποθετεί τις υπόλοιπες στη μνήμη, χρησιμοποιώντας φορτώσεις (loads) και αποθηκεύσεις (stores) για να μεταφέρει μεταβλητές μεταξύ καταχωρητών και μνήμης. Η διαδικασία της τοποθέτησης των λιγότερο συχνά χρησιμοποιούμενων μεταβλητών (ή αυτών που χρειάζονται αργότερα) στη μνήμη ονομάζεται *«σκόρπισμα»* (spilling) των καταχωρητών.

Η βασική αρχή του υλικού που συσχετίζει το μέγεθος με την ταχύτητα συνιστά η μνήμη να είναι πιο αργή από τους καταχωρητές αφού οι καταχωρητές είναι μικρότεροι. Και όντως έτσι συμβαίνει· οι προσπελάσεις δεδομένων είναι ταχύτερες αν τα δεδομένα βρίσκονται σε καταχωρητές αντί για τη μνήμη.

Επιπλέον, τα δεδομένα είναι πιο χρήσιμα όταν βρίσκονται σε έναν καταχωρητή. Μια αριθμητική εντολή τού MIPS μπορεί να διαβάσει δύο καταχωρητές, να επενεργήσει σε αυτούς, και να γράψει το αποτέλεσμα. Μια εντολή μεταφοράς δεδομένων του MIPS το μόνο που κάνει είναι να διαβάσει έναν τελεστέο ή να γράψει έναν τελεστέο, χωρίς να επενεργεί σε αυτόν.

Έτσι, οι καταχωρητές του MIPS και απαιτούν λιγότερο χρόνο για να προσπελαστούν *και* έχουν μεγαλύτερη ικανότητα διεκπεραίωσης (throughput) από τη μνήμη — ένας σπάνιος συνδυασμός — με αποτέλεσμα τα δεδομένα στους καταχωρητές να είναι ταχύτερα προσπελάσιμα και απλούστερα στη χρήση τους. Για να επιτευχθεί μέγιστη απόδοση, οι μεταγλωττιστές πρέπει να χρησιμοποιούν τους καταχωρητές αποδοτικά.

Χρησιμοποιώντας μόνο τις εντολές που έχουμε δει έως τώρα, πρέπει να φορτώσουμε μια σταθερά από τη μνήμη για να τη χρησιμοποιήσουμε. (Οι σταθερές θα έχουν τοποθετηθεί στη μνήμη κατά τη φόρτωση του προγράμματος.) Για παράδειγμα, για να προσθέσουμε τη σταθερά 4 στον καταχωρητή `$s3`, θα μπορούσαμε να χρησιμοποιήσουμε τον κώδικα

```
lw  $t0,AddrConstant4($s1)   # $t0 = σταθερά 4
add $s3,$s3,$t0              # $s3 = $s3 + $t0 ($t0 == 4)
```

με την παραδοχή ότι `AddrConstant4` είναι η διεύθυνση μνήμης της σταθεράς 4.

Μια εναλλακτική επιλογή που αποφεύγει την εντολή φόρτωσης είναι να διαθέσουμε εκδοχές των αριθμητικών εντολών στις οποίες ο ένας τελεστέος είναι σταθερά. Αυτή η γρήγορη εντολή πρόσθεσης με ένα σταθερό τελεστέο ονομάζεται *add immediate* (άμεση πρόσθεση) ή `addi`. Για να προσθέσουμε 4 στον καταχωρητή `$s3`, απλώς γράφουμε

```
addi $s3,$s3,4               # $s3 = $s3 + 4
```

Οι άμεσες εντολές δείχνουν την τρίτη σχεδιαστική αρχή του υλικού, που πρωτοαναφέραμε στην ενότητα «Πλάνες και παγίδες» του Κεφαλαίου 1:

Σχεδιαστική Αρχή 3: Κάνε τη συνηθισμένη περίπτωση γρήγορη.

Οι σταθεροί τελεστέοι εμφανίζονται συχνά, και η ενσωμάτωση των σταθερών μέσα στις αριθμητικές εντολές είναι πολύ ταχύτερη από τη φόρτωση των σταθερών από τη μνήμη.

Τελεστέοι του MIPS

Όνομα	Παράδειγμα	Σχόλια
32 καταχωρητές	`$s0,$s1,...`	Γρήγορες θέσεις για δεδομένα. Στο MIPS, τα δεδομένα πρέπει να βρίσκονται σε καταχωρητές για να εκτελεστούν αριθμητικές πράξεις
2^{30} λέξεις μνήμης	Memory[0], Memory[4], . . ., Memory[4294967292]	Προσπελάζονται μόνον από εντολές μεταφοράς δεδομένων στο MIPS. Ο MIPS χρησιμοποιεί διευθύνσεις byte, οπότε διαδοχικές διευθύνσεις λέξης διαφέρουν κατά 4. Η μνήμη κρατά δομές δεδομένων, πίνακες, και «διασκορπισμένους» (spilled) καταχωρητές.

Συμβολική γλώσσα MIPS

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Αριθμητικές πράξεις	add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Τρεις τελεστέοι· δεδομένα σε καταχωρητές
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2 - $s3`	Τρεις τελεστέοι· δεδομένα σε καταχωρητές
	add immediate	`addi $s1,$s2,100`	`$s1 = $s2 + 100`	Χρησιμοποιείται για να προσθέσει σταθερές
Μεταφορά δεδομένων	load word	`lw $s1,100($s2)`	`$s1 = Memory[$s2+100]`	Δεδομένα από τη μνήμη σε καταχωρητή
	store word	`sw $s1,100($s2)`	`Memory[$s2+100] = $s1`	Δεδομένα από καταχωρητή στη μνήμη

ΕΙΚΟΝΑ 2.4 Αρχιτεκτονική του MIPS που έχει αποκαλυφθεί μέχρι την Ενότητα 2.3. Τα επισημασμένα σημεία δείχνουν τις δομές της συμβολικής γλώσσας του MIPS που παρουσιάστηκαν για πρώτη φορά στην Ενότητα 2.3.

Η Εικόνα 2.4 συνοψίζει τα τμήματα της συμβολικής αναπαράστασης του συνόλου εντολών MIPS που περιγράψαμε στην ενότητα αυτή. Οι load word και store word είναι οι εντολές που αντιγράφουν λέξεις μεταξύ μνήμης και καταχωρητών στην αρχιτεκτονική MIPS. Άλλες μάρκες υπολογιστών χρησιμοποιούν και άλλες εντολές μαζί με τις load και store για να μεταφέρουν δεδομένα. Μια αρχιτεκτονική με τέτοιες εναλλακτικές περιπτώσεις είναι η Intel IA-32, που περιγράφεται στην Ενότητα 2.16.

Αυτοεξέταση

Με δεδομένη τη μεγάλη σημασία των καταχωρητών, ποιος είναι ο ρυθμός αύξησης του αριθμού των καταχωρητών σε ένα ολοκληρωμένο κύκλωμα στην πορεία του χρόνου;

1. Πολύ γρήγορος: Αυξάνονται τόσο γρήγορα όσο προβλέπει ο νόμος του Moore, ο οποίος αναμένει διπλασιασμό του αριθμού των τρανζίστορ ενός ολοκληρωμένου κυκλώματος κάθε 18 μήνες.
2. Πολύ αργός: Εφόσον τα προγράμματα συνήθως διανέμονται στη γλώσσα του υπολογιστή, υπάρχει αδράνεια στην αρχιτεκτονική του συνόλου εντολών και, συνεπώς, ο αριθμός των καταχωρητών αυξάνεται μόνο τόσο γρήγορα όσο γίνονται υλοποιήσιμα νέα σύνολα εντολών.

Επιπλέον ανάπτυξη: Μολονότι οι καταχωρητές του MIPS στο βιβλίο αυτό έχουν εύρος 32 bit, υπάρχει μια έκδοση 64 bit του συνόλου εντολών τού MIPS, με 32 καταχωρητές των 64 bit. Για χάρη της σαφήνειας, ονομάζονται επίσημα MIPS-32 και MIPS-64. Στο κεφάλαιο αυτό, χρησιμοποιούμε ένα υποσύνολο του MIPS-32. Το Παράρτημα Δ δείχνει τις διαφορές μεταξύ του MIPS-32 και του MIPS-64.

Η διευθυνσιοδότηση του MIPS με σχετική απόσταση (offset) και καταχωρητή βάσης ταιριάζει τέλεια τόσο στις δομές (structures) όσο και στους πίνακες (arrays), επειδή ο καταχωρητής μπορεί να δείξει στην αρχή της δομής, και η σχετική απόσταση μπορεί να επιλέξει το κατάλληλο στοιχείο. Θα δούμε ένα τέτοιο παράδειγμα στην Ενότητα 2.13.

Ο καταχωρητής στις εντολές μεταφοράς δεδομένων επινοήθηκε αρχικά για να κρατάει έναν αριθμοδείκτη πίνακα, με τη σχετική απόσταση να χρησιμοποιείται για την αρχική διεύθυνση του πίνακα. Έτσι, ο καταχωρητής βάσης ονομάζεται επίσης *καταχωρητής αριθμοδείκτη* (index register). Οι σημερινές μνήμες είναι πολύ μεγαλύτερες, και το μοντέλο λογισμικού για την κατανομή δεδομένων πιο εξελιγμένο, με αποτέλεσμα η διεύθυνση βάσης του πίνακα κανονικά να μεταβιβάζεται σε έναν καταχωρητή αφού, όπως θα δούμε, δεν μπορεί να χωρέσει στη σχετική απόσταση.

Η Ενότητα 2.4 εξηγεί ότι, εφόσον ο MIPS υποστηρίζει αρνητικές σταθερές, δεν υπάρχει ανάγκη για άμεση αφαίρεση (subtract immediate).

2.4 Αναπαράσταση εντολών στον υπολογιστή

Είμαστε τώρα έτοιμοι να εξηγήσουμε τη διαφορά ανάμεσα στον τρόπο που οι άνθρωποι δίνουν εντολές στους υπολογιστές και στον τρόπο που οι υπολογιστές βλέπουν τις εντολές. Πρώτα, ας εξετάσουμε γρήγορα τον τρόπο με το οποίο ένας υπολογιστής αναπαριστά τους αριθμούς.

Οι άνθρωποι διδάσκονται να σκέπτονται με βάση το 10, αλλά οι αριθμοί μπορούν να αναπαρασταθούν σε οποιαδήποτε βάση. Για παράδειγμα, 123 με βάση 10 = 1111011 με βάση 2.

Οι αριθμοί διατηρούνται στο υλικό των υπολογιστών ως μια σειρά ηλεκτρονικά σήματα υψηλής (high) και χαμηλής (low) τάσης και, έτσι, θεωρούνται αριθμοί με βάση το 2. (Όπως ακριβώς οι αριθμοί με βάση 10 λέγονται *δεκαδικοί* — decimal — αριθμοί, οι αριθμοί με βάση το 2 λέγονται *δυαδικοί* — binary — αριθμοί.) Ένα ψηφίο δυαδικού αριθμού είναι, επομένως, το «άτομο» (αυτό που δεν μπορεί να τμηθεί περισσότερο) της υπολογιστικής, αφού όλες οι πληροφορίες αποτελούνται από **δυαδικά ψηφία** (binary digits) ή *bit*. Αυτό το θεμελιώδες δομικό στοιχείο μπορεί να έχει μία από δύο τιμές, κάτι που μπορεί να θεωρηθεί με εναλλακτικούς τρόπους: υψηλό ή χαμηλό, ενεργό ή ανενεργό, αληθές ή ψευδές, 1 ή 0.

δυαδικό ψηφίο (binary digit) Ονομάζεται και bit. Ένας από τους δύο αριθμούς με βάση το 2, 0 ή 1, που είναι τα συστατικά των πληροφοριών.

Οι εντολές επίσης διατηρούνται στον υπολογιστή ως ακολουθίες από υψηλά και χαμηλά ηλεκτρονικά σήματα και μπορούν να αναπαρασταθούν με αριθμούς. Για την ακρίβεια, κάθε τμήμα μιας εντολής μπορεί να θεωρηθεί ως ένας ξεχωριστός αριθμός, και η τοποθέτηση αυτών των αριθμών δίπλα-δίπλα σχηματίζει την εντολή.

Εφόσον οι καταχωρητές είναι μέρος όλων σχεδόν των εντολών, πρέπει να υπάρχει μια σύμβαση για την αντιστοίχιση ονομάτων καταχωρητών σε αριθμούς. Στη συμβολική γλώσσα του MIPS, οι καταχωρητές `$s0` έως `$s7` αντιστοιχίζονται στους καταχωρητές 16 έως 23, και οι καταχωρητές `$t0` έως `$t7` αντιστοιχίζονται στους καταχωρητές 8 έως 15. Έτσι, `$s0` σημαίνει καταχωρητής 16, `$s1` σημαίνει καταχωρητής 17, `$s2` σημαίνει καταχωρητής 18, ..., `$t0` ση-

μαίνει καταχωρητής 8, `$t1` σημαίνει καταχωρητής 9, και ούτω καθεξής. Θα περιγράψουμε τη σύμβαση για τους υπόλοιπους από τους 32 καταχωρητές στις επόμενες ενότητες.

Μετάφραση μιας συμβολικής εντολής MIPS σε εντολή μηχανής

ΠΑΡΑΔΕΙΓΜΑ

Ας κάνουμε το επόμενο βήμα στην εξέλιξη της γλώσσας του MIPS με ένα παράδειγμα. Θα δείξουμε την πραγματική έκδοση γλώσσας MIPS για την εντολή που αναπαρίσταται συμβολικά ως

```
add $t0,$s1,$s2
```

πρώτα ως συνδυασμό δεκαδικών αριθμών και στη συνέχεια ως συνδυασμό δυαδικών αριθμών.

ΑΠΑΝΤΗΣΗ

Η δεκαδική αναπαράσταση είναι:

0	17	18	8	0	32

Καθένα από αυτά τα τμήματα μιας εντολής ονομάζεται *πεδίο* (field). Το πρώτο και το τελευταίο πεδίο (που περιέχουν 0 και 32 στην περίπτωση αυτή) μαζί λένε στον υπολογιστή MIPS ότι αυτή η εντολή εκτελεί πρόσθεση. Το δεύτερο πεδίο δίνει τον αριθμό του καταχωρητή, που είναι ο πρώτος τελεστέος προέλευσης (source operand) της πράξης της πρόσθεσης (17 = `$s1`), και το τρίτο πεδίο δίνει τον άλλο τελεστέο προέλευσης της πρόσθεσης (18 = `$s2`). Το τέταρτο πεδίο περιέχει τον αριθμό του καταχωρητή που πρόκειται να δεχθεί το άθροισμα (8 = `$t0`). Το πέμπτο πεδίο δε χρησιμοποιείται σε αυτή την εντολή, και έτσι τίθεται ίσο με 0. Συνεπώς, αυτή η εντολή προσθέτει τον καταχωρητή `$s1` στον καταχωρητή `$s2` και αποθηκεύει το αποτέλεσμα στον καταχωρητή `$t0`.

Αυτή η εντολή μπορεί επίσης να αναπαρασταθεί με πεδία δυαδικών αριθμών αντί για πεδία δεκαδικών:

000000	10001	10010	01000	00000	100000
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

Για να τη διακρίνουμε από τη συμβολική γλώσσα (assembly), ονομάζουμε την αριθμητική έκδοση των εντολών **γλώσσα μηχανής** (machine language), και μια ακολουθία τέτοιων εντολών *κώδικα μηχανής* (machine code).

Αυτή η διάταξη της εντολής ονομάζεται **μορφή εντολής** (instruction format). Όπως μπορείτε να διαπιστώσετε μετρώντας τον αριθμό των bit, αυτή η εντολή MIPS καταλαμβάνει ακριβώς 32 bit — το ίδιο μέγεθος με μία λέξη δεδομένων. Σε συμφωνία με τη σχεδιαστική αρχή μας ότι η απλότητα ευνοεί την κανονικότητα, όλες οι εντολές του MIPS έχουν μήκος 32 bit.

γλώσσα μηχανής (machine language) Δυαδική αναπαράσταση που χρησιμοποιείται για την επικοινωνία μέσα σε ένα υπολογιστικό σύστημα.

μορφή εντολής (instruction format) Μορφή αναπαράστασης μιας εντολής αποτελούμενη από πεδία δυαδικών αριθμών.

Δεκαεξαδικό	Δυαδικό	Δεκαεξαδικό	Δυαδικό	Δεκαεξαδικό	Δυαδικό	Δεκαεξαδικό	Δυαδικό
0_{hex}	0000_{two}	4_{hex}	0100_{two}	8_{hex}	1000_{two}	c_{hex}	1100_{two}
1_{hex}	0001_{two}	5_{hex}	0101_{two}	9_{hex}	1001_{two}	d_{hex}	1101_{two}
2_{hex}	0010_{two}	6_{hex}	0110_{two}	a_{hex}	1010_{two}	e_{hex}	1110_{two}
3_{hex}	0011_{two}	7_{hex}	0111_{two}	b_{hex}	1011_{two}	f_{hex}	1111_{two}

ΕΙΚΟΝΑ 2.5 Ο πίνακας μετατροπής δεκαεξαδικών-δυαδικών. Απλώς αντικαταστήστε ένα δεκαεξαδικό ψηφίο με τα αντίστοιχα τέσσερα δυαδικά, και αντίστροφα. Αν το μήκος του δυαδικού αριθμού δεν είναι πολλαπλάσιο του 4, προχωρήστε από δεξιά προς τα αριστερά.

δεκαεξαδικοί (hexadecimal) Αριθμοί με βάση το 16.

Ίσως νομίσετε ότι τώρα θα διαβάζετε και θα γράφετε μεγάλες και κοπιαστικές σειρές από δυαδικούς αριθμούς. Αποφεύγουμε αυτή την αγγαρεία χρησιμοποιώντας μια μεγαλύτερη βάση από τη δυαδική, η οποία μετατρέπεται εύκολα σε δυαδική. Εφόσον σχεδόν όλα τα μεγέθη δεδομένων των υπολογιστών είναι πολλαπλάσια του 4, είναι δημοφιλείς οι **δεκαεξαδικοί** (hexadecimal — με βάση το 16) αριθμοί. Επειδή η βάση 16 είναι δύναμη του 2, μπορούμε εύκολα να κάνουμε τη μετατροπή, αντικαθιστώντας κάθε ομάδα από τέσσερα δυαδικά ψηφία με ένα δεκαεξαδικό, και αντίστροφα. Η Εικόνα 2.5 μετατρέπει δεκαεξαδικούς αριθμούς σε δυαδικούς, και αντίστροφα.

Επειδή συχνά ασχολούμαστε με διαφορετικές βάσεις αριθμών, για να αποφύγουμε τη σύγχυση θα βάλουμε δείκτη *ten* στους δεκαδικούς αριθμούς, δείκτη *two* στους δυαδικούς αριθμούς, και δείκτη *hex* στους δεκαεξαδικούς. (Αν δεν υπάρχει δείκτης, η εξ ορισμού βάση είναι το 10.) Με την ευκαιρία, οι γλώσσες C και Java χρησιμοποιούν το συμβολισμό 0x*nnnn* για τους δεκαεξαδικούς αριθμούς.

ΠΑΡΑΔΕΙΓΜΑ

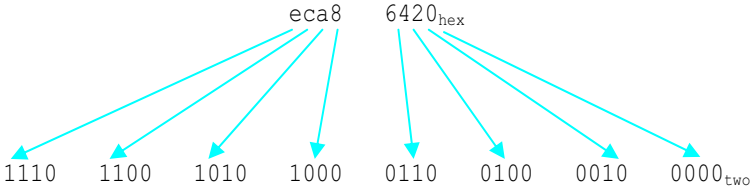
Δυαδικό σε δεκαεξαδικό και αντίστροφα

Μετατρέψτε τους επόμενους δεκαεξαδικούς και δυαδικούς αριθμούς στην άλλη βάση:

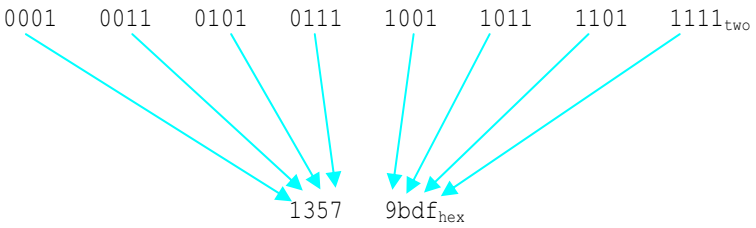
eca8 6420_{hex}
0001 0011 0101 0111 1001 1011 1101 1111_{two}

ΑΠΑΝΤΗΣΗ

Μια απλή αναζήτηση του πίνακα στη μια κατεύθυνση:



Και μετά το ίδιο στην άλλη κατεύθυνση:



Πεδία του MIPS

Στα πεδία του MIPS δίνουμε ονόματα για να μπορούμε να τα αναλύσουμε ευκολότερα:

op	rs	rt	rd	shamt	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

Η σημασία κάθε ονόματος των πεδίων των εντολών MIPS είναι η εξής:

- *op:* Βασική λειτουργία της εντολής, που παραδοσιακά ονομάζεται **opcode** (operation code — κωδικός λειτουργίας).
- *rs:* Ο τελεστέος προέλευσης του πρώτου καταχωρητή.
- *rt:* Ο τελεστέος προέλευσης του δεύτερου καταχωρητή.
- *rd:* Ο τελεστέος του καταχωρητή προορισμού. Αποθηκεύει το αποτέλεσμα της πράξης (λειτουργίας).
- *shamt:* Ποσότητα ολίσθησης (shift amount). (Η Ενότητα 2.5 εξηγεί τις εντολές ολίσθησης και τον όρο αυτόν· δε θα χρησιμοποιηθεί μέχρι τότε, και συνεπώς το πεδίο περιέχει το μηδέν.)
- *funct:* Συνάρτηση (function). Το πεδίο αυτό επιλέγει τη συγκεκριμένη παραλλαγή της πράξης (λειτουργίας) που περιέχεται στο πεδίο op και, μερικές φορές, ονομάζεται *κωδικός συνάρτησης* (function code).

opcode (κωδικός λειτουργίας) Το πεδίο που δείχνει τη λειτουργία και τη μορφή μιας εντολής.

Ένα πρόβλημα παρουσιάζεται όταν μια εντολή χρειάζεται μεγαλύτερα πεδία από αυτά που φαίνονται παραπάνω. Για παράδειγμα, η εντολή load word πρέπει να ορίζει δύο καταχωρητές και μία σταθερά. Αν η διεύθυνση έπρεπε να χρησιμοποιήσει ένα από τα πεδία των 5 bit στην παραπάνω μορφή, η σταθερά μέσα στην εντολή load word θα περιοριζόταν σε 2^5, ή 32. Αυτή η σταθερά χρησιμοποιείται για να επιλέξει στοιχεία από πίνακες ή δομές δεδομένων και, συχνά, χρειάζεται να είναι πολύ μεγαλύτερη από 32. Αυτό το πεδίο των 5 bit είναι πολύ μικρό για να είναι χρήσιμο.

Έτσι, έχουμε μια διένεξη ανάμεσα στην επιθυμία να κρατήσουμε όλες τις εντολές στο ίδιο μήκος και στην επιθυμία να έχουμε μία μόνο μορφή εντολών. Αυτό μας οδηγεί στην τελευταία σχεδιαστική αρχή του υλικού:

Σχεδιαστική Αρχή 4: Η καλή σχεδίαση απαιτεί καλούς συμβιβασμούς.

Ο συμβιβασμός που έχει επιλεγεί από τους σχεδιαστές του MIPS είναι να έχουν όλες οι εντολές το ίδιο μήκος, απαιτώντας έτσι διαφορετικές μορφές για διαφορετικά είδη εντολών. Για παράδειγμα, η πιο πάνω μορφή ονομάζεται *τύπος R* (R-type) ή *μορφή R* (R-format) — από τον καταχωρητή (register). Ένας δεύτερος τύπος μορφής εντολής ονομάζεται *τύπος I* (I-type) ή *μορφή I* (I-format) — από τη λέξη immediate (άμεσο) και χρησιμοποιείται από τις άμεσες (immediate) εντολές και τις εντολές μεταφοράς δεδομένων (data transfer). Τα πεδία της μορφής I είναι

op	rs	rt	σταθερά ή διεύθυνση
6 bit	5 bit	5 bit	16 bit

Η διεύθυνση (address) των 16 bit σημαίνει ότι μια εντολή φόρτωσης λέξης (load word) μπορεί να φορτώσει οποιαδήποτε λέξη μέσα σε ένα εύρος $\pm 2^{15}$ ή 32.768 byte ($\pm 2^{13}$ ή 8192 λέξεις) από τη διεύθυνση που βρίσκεται στον καταχωρητή βάσης `rs`. Παρόμοια, η εντολή add immediate περιορίζεται σε σταθερές όχι μεγαλύτερες από $\pm 2^{15}$. (Το Κεφάλαιο 3 εξηγεί πώς αναπαρίστανται οι αρνητικοί αριθμοί.) Βλέπουμε ότι σε αυτή τη μορφή θα ήταν δύσκολο να υπάρχουν περισσότεροι από 32 καταχωρητές, αφού τα πεδία `rs` και `rt` θα χρειαζόταν το καθένα άλλο ένα bit, πράγμα που θα έκανε δυσκολότερο το να χωρέσουν όλα σε μία λέξη.

Ας δούμε την εντολή load word της σελίδας 75:

```
lw $t0,32($s3) # ο προσωρινός καταχωρητής $t0 παίρνει το A[8]
```

Εδώ, το 19 (για τον καταχωρητή `$s3`) τοποθετείται στο πεδίο `rs`, το 8 (για τον `$t0`) τοποθετείται στο πεδίο `rt`, και το 32 τοποθετείται στο πεδίο διεύθυνσης. Σημειώστε ότι η έννοια του πεδίου `rt` έχει αλλάξει για την εντολή αυτή: σε μια εντολή load word, το πεδίο `rt` προσδιορίζει τον καταχωρητή *προορισμού* (destination register), ο οποίος δέχεται το αποτέλεσμα της φόρτωσης.

Παρόλο που οι πολλές εναλλακτικές μορφές των εντολών κάνουν το υλικό πιο πολύπλοκο, μπορούμε να μειώσουμε την πολυπλοκότητα κρατώντας τις μορφές παρόμοιες. Για παράδειγμα, τα τρία πρώτα πεδία των μορφών τύπου R και τύπου I είναι του ίδιου μεγέθους και έχουν τα ίδια ονόματα· το τέταρτο πεδίο στον τύπο I είναι ίσο με το μέγεθος των τριών τελευταίων πεδίων του τύπου R.

Σε περίπτωση που αναρωτιέστε, οι μορφές ξεχωρίζουν από την τιμή του πρώτου πεδίου: σε κάθε μορφή ανατίθεται ένα διακριτό σύνολο από τιμές του πρώτου πεδίου (`op`) ώστε το υλικό να γνωρίζει αν θα χειριστεί το τελευταίο μισό της εντολής σαν τρία πεδία (τύπος R) ή σαν ένα πεδίο (τύπος I). Η Εικόνα 2.6 δείχνει τους αριθμούς που χρησιμοποιούνται σε κάθε πεδίο για τις εντολές του MIPS που καλύψαμε μέχρι την Ενότητα 2.3.

Εντολή	Μορφή	op	rs	rt	rd	shamt	funct	address
`add`	R	0	reg	reg	reg	0	32_{ten}	δ.ε.
`sub` (subtract)	R	0	reg	reg	reg	0	34_{ten}	δ.ε.
`add immediate`	I	8_{ten}	reg	reg	δ.ε.	δ.ε.	δ.ε.	σταθερά
`lw` (load word)	I	35_{ten}	reg	reg	δ.ε.	δ.ε.	δ.ε.	διεύθυνση
`sw` (store word)	I	43_{ten}	reg	reg	δ.ε.	δ.ε.	δ.ε.	διεύθυνση

ΕΙΚΟΝΑ 2.6 Κωδικοποίηση εντολών του MIPS. Στον παραπάνω πίνακα, «reg» σημαίνει αριθμός καταχωρητή (register) μεταξύ 0 και 31, «διεύθυνση» σημαίνει μια διεύθυνση των 16 bit και «δ.ε.» («δεν εφαρμόζεται») σημαίνει ότι αυτό το πεδίο δεν εμφανίζεται στην μορφή αυτή. Σημειώστε ότι οι εντολές `add` και `sub` έχουν την ίδια τιμή στο πεδίο `op`· το υλικό χρησιμοποιεί το πεδίο funct για να προσδιορίσει την παραλλαγή της λειτουργίας: add (32) ή subtract (34).

Μετάφραση συμβολικής γλώσσας του MIPS σε γλώσσα μηχανής

ΠΑΡΑΔΕΙΓΜΑ

Μπορούμε τώρα να δούμε ένα παράδειγμα που καλύπτει όλη τη διαδρομή, από αυτό που γράφει ο προγραμματιστής, μέχρι αυτό που εκτελεί ο υπολογιστής. Αν ο καταχωρητής `$t1` περιέχει τη διεύθυνση βάσης του πίνακα `A` και ο `$s2` αντιστοιχεί στο `h`, η εντολή ανάθεσης τιμής

```
A[300] = h + A[300];
```

μεταγλωττίζεται σε

```
lw  $t0,1200($t1) # Ο προσωρινός καταχωρητής $t0 παίρνει
                  # το A[300]
add $t0,$s2,$t0   # Ο προσωρινός καταχωρητής $t0 παίρνει
                  # το h + A[300]
sw  $t0,1200($t1) # αποθηκεύει ξανά το h + A[300] στο A[300]
```

Ποιος είναι ο κώδικας γλώσσας μηχανής του MIPS γι' αυτές τις τρεις εντολές;

ΑΠΑΝΤΗΣΗ

Για χάρη της ευκολίας, ας αναπαραστήσουμε τις εντολές γλώσσας μηχανής πρώτα με δεκαδικούς αριθμούς. Από την Εικόνα 2.6, μπορούμε να προσδιορίσουμε τις τρεις εντολές γλώσσας μηχανής:

op	rs	rt	rd	address/ shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

Η εντολή `lw` προσδιορίζεται με το 35 (δείτε την Εικόνα 2.6) στο πρώτο πεδίο (op). Ο καταχωρητής βάσης 9 (`$t1`) καθορίζεται στο δεύτερο πεδίο (rs), και ο καταχωρητής προορισμού 8 (`$t0`) καθορίζεται στο τρίτο πεδίο (rt). Η σχετική απόσταση (offset) για την επιλογή του `A[300]` (1200 = 300 × 4) βρίσκεται στο τελευταίο πεδίο (της διεύθυνσης — address).

Η εντολή `add` που ακολουθεί προσδιορίζεται με 0 στο πρώτο πεδίο (op) και 32 στο τελευταίο πεδίο (funct). Οι τρεις τελεστέοι καταχωρητών (18, 8, και 8) βρίσκονται στο δεύτερο, το τρίτο, και το τέταρτο πεδίο και αντιστοιχούν στους `$s2`, `$t0`, και `$t0`.

Η εντολή `sw` προσδιορίζεται με το 43 στο πρώτο πεδίο. Το υπόλοιπο αυτής της τελευταίας εντολής είναι πανομοιότυπο με την εντολή `lw`.

Το δυαδικό ισοδύναμο της δεκαδικής μορφής είναι το εξής (το 1200 σε βάση 10 είναι ίσο με 0000 0100 1011 0000 σε βάση 2):

100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Παρατηρήστε την ομοιότητα στις δυαδικές αναπαραστάσεις της πρώτης και της τελευταίας εντολής. Η μόνη διαφορά είναι το τρίτο bit από αριστερά.

Η Εικόνα 2.7 συνοψίζει τα μέρη της συμβολικής γλώσσας του MIPS που περιγράψαμε σε αυτή την ενότητα. Όπως θα δούμε στα Κεφάλαια 5 και 6, η ομοιότητα των δυαδικών αναπαραστάσεων σχετικών μεταξύ τους εντολών απλοποιεί τη σχεδίαση του υλικού. Αυτές οι εντολές είναι άλλο ένα παράδειγμα της κανονικότητας της αρχιτεκτονικής MIPS.

Αυτοεξέταση

Γιατί δεν έχει ο MIPS εντολή άμεσης αφαίρεσης (subtract immediate);

1. Οι αρνητικές σταθερές εμφανίζονται πολύ λιγότερο συχνά στη C και την Java, και επομένως δεν είναι η συνήθης περίπτωση και δεν αξίζουν ειδική υποστήριξη.
2. Εφόσον το άμεσο (immediate) πεδίο διατηρεί τόσο αρνητικές όσο και θετικές σταθερές, η άμεση πρόσθεση (add immediate) με έναν αρνητικό αριθμό είναι ισοδύναμη με την άμεση αφαίρεση με ένα θετικό αριθμό, συνεπώς η άμεση αφαίρεση είναι πλεονασμός.

Η ΣΥΝΟΛΙΚΗ εικόνα

Οι σημερινοί υπολογιστές κτίζονται με βάση δύο αρχές-κλειδιά:

1. Οι εντολές αναπαρίστανται με αριθμούς.
2. Τα προγράμματα αποθηκεύονται στη μνήμη για να διαβαστούν ή να γραφούν, ακριβώς όπως οι αριθμοί.

Αυτές οι αρχές οδηγούν στην έννοια του *αποθηκευμένου προγράμματος* (stored program concept)· η επινόησή της απελευθέρωσε το τζίνι της υπολογιστικής από το μπουκάλι του. Η Εικόνα 2.8 παρουσιάζει την ισχύ της έννοιας· συγκεκριμένα, η μνήμη μπορεί να περιέχει τον πηγαίο κώδικα ενός προγράμματος διορθωτή (editor), τον αντίστοιχο μεταγλωττισμένο κώδικα μηχανής, το κείμενο που χρησιμοποιεί το μεταγλωττισμένο πρόγραμμα, ακόμη και το μεταγλωττιστή που παρήγαγε τον κώδικα μηχανής.

Μια συνέπεια της αναπαράστασης των εντολών με αριθμούς είναι ότι τα προγράμματα συχνά κυκλοφορούν ως αρχεία δυαδικών αριθμών. Η εμπορική συνέπεια είναι ότι οι υπολογιστές μπορούν να κληρονομήσουν έτοιμο λογισμικό με την προϋπόθεση ότι είναι συμβατοί με ένα υπάρχον σύνολο εντολών. Τέτοια «δυαδική συμβατότητα» (binary compatibility) συχνά οδηγεί τη βιομηχανία να ευθυγραμμίζεται γύρω από ένα μικρό αριθμό αρχιτεκτονικών συνόλου εντολών.

Τελεστέοι του MIPS

Όνομα	Παράδειγμα	Σχόλια
32 καταχωρητές	`$s0, $s1,....$s7` `$t0, $t1,....$t7`	Γρήγορες θέσεις για δεδομένα. Στο MIPS, τα δεδομένα πρέπει να βρίσκονται σε καταχωρητές για να εκτελεστούν αριθμητικές πράξεις. Οι καταχωρητές `$s0-$s7` αντιστοιχούν στους αριθμούς 16-23, και οι `$t0-$t7` στους 8-15.
2^{30} λέξεις μνήμης	Memory[0], Memory[4], . . ., Memory[4294967292]	Προσπελάζονται μόνον από εντολές μεταφοράς δεδομένων στο MIPS. Ο MIPS χρησιμοποιεί διευθύνσεις byte, και έτσι διαδοχικές διευθύνσεις λέξης διαφέρουν κατά 4. Η μνήμη κρατά δομές δεδομένων, πίνακες και «διασκορπισμένους» (spilled) καταχωρητές.

Συμβολική γλώσσα του MIPS

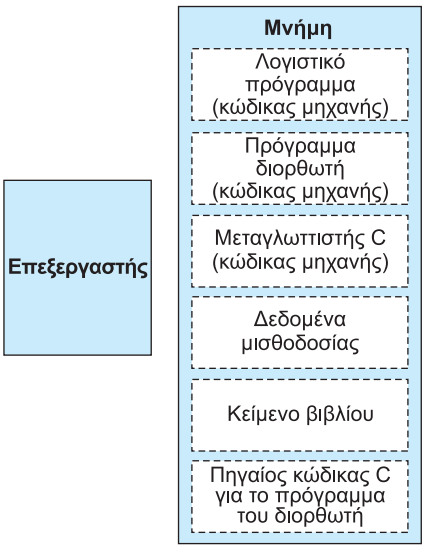
Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Αριθμητικές πράξεις	add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Τρεις τελεστέοι· δεδομένα σε καταχωρητές
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2 - $s3`	Τρεις τελεστέοι· δεδομένα σε καταχωρητές
Μεταφορά δεδομένων	load word	`lw $s1,100($s2)`	`$s1 = Memory[$s2+100]`	Δεδομένα από τη μνήμη σε καταχωρητή
	store word	`sw $s1,100($s2)`	`Memory[$s2+100] = $s1`	Δεδομένα από καταχωρητή στη μνήμη

Γλώσσα μηχανής του MIPS

Όνομα	Μορφή	Παράδειγμα						Σχόλια
add	R	0	18	19	17	0	32	`add $s1,$s2,$s3`
sub	R	0	18	19	17	0	34	`sub $s1,$s2,$s3`
addi	I	8	18	17	100			`addi $s1,$s2,100`
lw	I	35	18	17	100			`lw $s1,100($s2)`
sw	I	43	18	17	100			`sw $s1,100($s2)`
Μέγεθος πεδίου		6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	Όλες οι εντολές του MIPS 32 bit
Μορφή R	R	op	rs	rt	rd	shamt	funct	Μορφή αριθμητική εντολής
Μορφή I	I	op	rs	rt	address			Μορφή μεταφοράς δεδομένων

ΕΙΚΟΝΑ 2.7 Αρχιτεκτονική του MIPS που αποκαλύψαμε μέχρι την Ενότητα 2.4. Τα επισημασμένα σημεία δείχνουν τις δομές της συμβολικής γλώσσας του MIPS που παρουσιάσαμε για πρώτη φορά στην Ενότητα 2.4. Οι δύο μορφές εντολών μέχρι τώρα είναι οι R και I. Τα πρώτα 16 bit είναι τα ίδια: και τα δύο περιέχουν ένα πεδίο *op*, που δίνει τη βασική λειτουργία· ένα πεδίο *rs* που δίνει έναν από τους τελεστέους προέλευσης· και το πεδίο *rt* που προσδιορίζει τον άλλο τελεστέο προέλευσης, εκτός από τη load word που προσδιορίζει τον καταχωρητή προορισμού. Η μορφή R διαιρεί τα τελευταία 16 bit σε ένα πεδίο *rd* που προσδιορίζει τον καταχωρητή προορισμού· ένα πεδίο *shamt*, το οποίο εξηγείται στην Ενότητα 2.5, και το πεδίο *funct* που προσδιορίζει τη συγκεκριμένη λειτουργία των εντολών μορφής R. Η μορφή I κρατάει τα τελευταία 16 bit ως ένα μόνο πεδίο *address* (διεύθυνσης).

Επιπλέον ανάπτυξη: Η αναπαράσταση δεκαδικών αριθμών σε βάση 2 παρέχει έναν εύκολο τρόπο να αναπαρασταθούν θετικοί ακέραιοι με λέξεις του υπολογιστή. Το Κεφάλαιο 3 εξηγεί πώς αναπαριστούμε τους αρνητικούς αριθμούς, αλλά για την ώρα πιστέψτε ότι μια λέξη των 32 bit μπορεί να αναπαραστήσει ακεραίους μεταξύ -2^{31} και $+2^{31} - 1$ ή –2.147.483.648 έως +2.147.483.647, και το πεδίο σταθεράς των 16 bit κρατάει πραγματικά από -2^{15} έως $+2^{15} - 1$ ή –32.768 έως +32.767. Τέτοιοι ακέραιοι ονομάζονται αριθμοί *συμπληρώματος ως προς δύο* (two's complement). Το Κεφάλαιο 3 δείχνει πώς θα κωδικοποιούσαμε την εντολή `addi $t0,$t0,-1` ή την `lw $t0,-4($s0)`, οι οποίες απαιτούν αρνητικούς αριθμούς στο πεδίο σταθεράς της άμεσης μορφής (μορφή I).



ΕΙΚΟΝΑ 2.8 Η έννοια του αποθηκευμένου προγράμματος (stored program concept). Τα αποθηκευμένα προγράμματα επιτρέπουν σε έναν υπολογιστή που τηρεί λογιστικά βιβλία να γίνει, σε χρόνο όσο το άνοιγμα και το κλείσιμο των ματιών, ένας υπολογιστής που βοηθάει ένα συγγραφέα να γράψει ένα βιβλίο. Η αλλαγή συμβαίνει με την απλή φόρτωση προγραμμάτων και δεδομένων στη μνήμη και την εντολή στον υπολογιστή να ξεκινήσει την εκτέλεση από μια συγκεκριμένη θέση της μνήμης. Ο χειρισμός των εντολών με τον ίδιο τρόπο όπως τα δεδομένα απλοποιεί σημαντικά τόσο το υλικό της μνήμης όσο και το λογισμικό των υπολογιστικών συστημάτων. Πιο συγκεκριμένα, η τεχνολογία μνήμης που απαιτείται για τα δεδομένα μπορεί επίσης να χρησιμοποιηθεί για τα προγράμματα, και προγράμματα όπως οι μεταγλωττιστές, για παράδειγμα, μπορούν να μεταφράσουν κώδικα, γραμμένο σε μια σημειογραφία πολύ πιο βολική για τους ανθρώπους, σε κώδικα που μπορεί να καταλάβει ο υπολογιστής.

«Contrariwise», continued Tweedledee, «if it was so, it might be· and if it were so, it would be· but as it isn't, it ain't. That's logic.»

Lewis Carroll, *Οι Περιπέτειες της Αλίκης στη Χώρα των Θαυμάτων*, 1865

2.5 Λογικές λειτουργίες (πράξεις)

Αν και οι πρώτοι υπολογιστές ήταν επικεντρωμένοι σε ολόκληρες λέξεις, έγινε σύντομα σαφές ότι ήταν χρήσιμο να γίνονται πράξεις σε πεδία bit μέσα σε μια λέξη, ή ακόμη και σε μεμονωμένα bit. Η εξέταση των χαρακτήρων μέσα σε μια λέξη, καθένας από τους οποίους αποθηκεύεται σε 8 bit, είναι ένα παράδειγμα τέτοιας λειτουργίας. Στη συνέχεια προστέθηκαν εντολές για την απλοποίηση, μεταξύ άλλων, της σύμπτυξης (packing) και της ανάπτυξης (unpacking) των bit σε λέξεις. Αυτές οι εντολές ονομάζονται λογικές πράξεις — ή λειτουργίες — (logical operations). Η Εικόνα 2.9 παρουσιάζει τις λογικές λειτουργίες στη C και την Java.

Λογικές λειτουργίες	Τελεστές C	Τελεστές Java	Εντολές MIPS
Αριστερή ολίσθηση (Shift left)	<<	<<	sll
Δεξιά ολίσθηση (Shift right)	>>	>>>	srl
AND bit προς bit	&	&	and, andi
OR bit προς bit	|	|	or, ori
NOT bit προς bit	~	~	nor

ΕΙΚΟΝΑ 2.9 Λογικοί τελεστές (operators) στη C και την Java, και οι αντίστοιχες εντολές του MIPS. Ο MIPS υλοποιεί τη NOT με μια NOR της οποίας ο ένας τελεστέος είναι μηδέν.

Η πρώτη κατηγορία τέτοιων λειτουργιών ονομάζεται *ολίσθηση* (shift). Οι αντίστοιχες πράξεις μεταφέρουν όλα τα bit μιας λέξης αριστερά ή δεξιά, συμπληρώνοντας τις θέσεις των bit που αδειάζουν με μηδενικά. Για παράδειγμα, αν ο καταχωρητής `$s0` περιείχε

$$0000\ 0000\ 0000\ 00000\ 000\ 0000\ 0000\ 0000\ 1001_{two} = 9_{ten}$$

και εκτελούνταν η εντολή αριστερής ολίσθησης κατά 4 θέσεις, η νέα τιμή θα έμοιαζε με την:

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001\ 0000_{two} = 144_{ten}$$

Η δίδυμη της αριστερής ολίσθησης είναι η δεξιά ολίσθηση. Το πραγματικό όνομα των δύο εντολών ολίσθησης του MIPS είναι *shift left logical* (`sll` — αριστερή λογική ολίσθηση) και *shift right logical* (`srl` — δεξιά λογική ολίσθηση). Η επόμενη εντολή εφαρμόζει την παραπάνω λειτουργία, με την προϋπόθεση ότι το αποτέλεσμα πρέπει να μεταφερθεί στον καταχωρητή `$t2`.

```
sll  $t2,$s0,4           # reg $t2 = reg $s0 << 4 bit
```

Καθυστερήσαμε την εξήγηση του πεδίου *shamt* στη μορφή R. Σημαίνει *shift amount* (ποσότητα ολίσθησης) και χρησιμοποιείται στις εντολές ολίσθησης. Έτσι, η έκδοση γλώσσας μηχανής της παραπάνω εντολής είναι

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0

Η κωδικοποίηση της `sll` είναι 0 τόσο στο πεδίο op όσο και στο πεδίο funct, το πεδίο rd περιέχει τον καταχωρητή `$t2`, το rt περιέχει τον `$s0`, και το shamt περιέχει 4. Το πεδίο rs δε χρησιμοποιείται και, έτσι, τίθεται ίσο με 0.

Η αριστερή λογική ολίσθηση έχει ένα επιπλέον πλεονέκτημα. Η αριστερή ολίσθηση κατά i bit δίνει το ίδιο αποτέλεσμα όπως ο πολλαπλασιασμός με 2^i (το

Κεφάλαιο 3 εξηγεί το γιατί). Για παράδειγμα, η παραπάνω εντολή `sll` ολισθαίνει κατά 4, κάτι που δίνει το ίδιο αποτέλεσμα όπως ο πολλαπλασιασμός με το 2^4 ή το 16. Η πρώτη σειρά bit πιο πάνω αναπαριστά το 9, και $9 \times 16 = 144$, η τιμή της δεύτερης σειράς bit.

Μια άλλη χρήσιμη πράξη (λειτουργία) που απομονώνει πεδία είναι η *AND*. Η AND είναι πράξη bit που αφήνει 1 στο αποτέλεσμα μόνον αν και τα δύο bit των τελεστέων είναι 1. Για παράδειγμα, αν ο καταχωρητής `$t2` περιέχει ακόμη

$$0000\ 0000\ 0000\ 0000\ 0000\ 1101\ 0000\ 0000_{two}$$

και ο καταχωρητής `$t1` περιέχει

$$0000\ 0000\ 0000\ 0000\ 0011\ 1100\ 0000\ 0000_{two}$$

τότε, μετά την εκτέλεση της επόμενης εντολής του MIPS

```
and $t0,$t1,$t2   # reg $t0 = reg $t1 & reg $t2
```

η τιμή του καταχωρητή `$t0` θα ήταν

$$0000\ 0000\ 0000\ 0000\ 0000\ 1100\ 0000\ 0000_{two}$$

Όπως μπορείτε να δείτε, η AND μπορεί να εφαρμόσει μια σειρά bit σε ένα σύνολο bit για να τοποθετήσει μηδενικά εκεί όπου υπάρχει 0 στη σειρά των bit. Μια τέτοια σειρά bit, σε συνδυασμό με μια εντολή AND ονομάζεται παραδοσιακά *μάσκα* (mask), αφού η μάσκα «καλύπτει» κάποια bit.

Για να τοποθετήσουμε μια τιμή σε μια από αυτές τις «θάλασσες» των 0, υπάρχει η δίδυμη της AND που ονομάζεται *OR*. Είναι μια πράξη bit που τοποθετεί 1 στο αποτέλεσμα αν *οποιοδήποτε* από τα bit των δύο τελεστέων είναι 1. Για να το αναλύσουμε περισσότερο, αν οι καταχωρητές `$t1` και `$t2` είναι αμετάβλητοι από το προηγούμενο παράδειγμα, το αποτέλεσμα της επόμενης εντολής του MIPS

```
or $t0,$t1,$t2    # reg $t0 = reg $t1 | reg $t2
```

είναι η εξής τιμή στον καταχωρητή `$t0`:

$$0000\ 0000\ 0000\ 0000\ 0011\ 1101\ 0000\ 0000_{two}$$

NOT Μια λογική πράξη (λειτουργία) bit με έναν τελεστέο που αντιστρέφει τα bit· αυτό σημαίνει ότι αντικαθιστά κάθε 1 με 0 και κάθε 0 με 1.

NOR Μια λογική πράξη (λειτουργία) bit με δύο τελεστέους, που υπολογίζει το NOT του OR των δύο τελεστέων.

Η τελευταία λογική πράξη (λειτουργία) είναι μια πράξη άρνησης. Η **NOT** παίρνει έναν τελεστέο και τοποθετεί την τιμή 1 στο αποτέλεσμα αν ένα bit του τελεστέου είναι 0, και αντίστροφα. Για τη διατήρηση της μορφής με τους δύο τελεστέους, οι σχεδιαστές του MIPS αποφάσισαν να συμπεριλάβουν την εντολή **NOR** (NOT OR) αντί για τη NOT. Αν ένας τελεστέος είναι μηδέν, τότε είναι ισοδύναμη με τη NOT. Για παράδειγμα, A NOR 0 = NOT(A OR 0) = NOT (A).

Αν ο καταχωρητής `$t1` είναι αμετάβλητος από το προηγούμενο παράδειγμα και ο καταχωρητής `$t3` έχει την τιμή 0, το αποτέλεσμα της επόμενης εντολής του MIPS

```
nor $t0,$t1,$t3   # reg $t0 = ~ (reg $t1 | reg $t3)
```

είναι η επόμενη τιμή στον καταχωρητή `$t0`:

`1111 1111 1111 1111 1100 0011 1111 1111`$_{two}$

Η Εικόνα 2.9 πιο πάνω δείχνει τη σχέση ανάμεσα στους τελεστές της C και της Java και τις εντολές του MIPS. Οι σταθερές είναι χρήσιμες τόσο στις λογικές λειτουργίες AND και OR όσο και στις αριθμητικές πράξεις, οπότε ο MIPS παρέχει επίσης τις εντολές *and immediate* (άμεσο and — `andi`) και *or immediate* (άμεσο or — `ori`). Οι σταθερές είναι σπάνιες για τη NOR, αφού η κύρια χρήση της είναι να αντιστρέφει τα bit ενός τελεστέου· έτσι, το υλικό δεν έχει άμεση (immediate) έκδοση. Η Εικόνα 2.10, που συνοψίζει τις εντολές του MIPS τις οποίες έχουμε δει μέχρι τώρα, επισημαίνει τις λογικές εντολές.

Τελεστέοι του MIPS

Όνομα	Παράδειγμα	Σχόλια
32 καταχωρητές	`$s0, $s1,....$s7` `$t0, $t1,....$t7`	Γρήγορες θέσεις για δεδομένα. Στο MIPS, τα δεδομένα πρέπει να βρίσκονται σε καταχωρητές για να εκτελεστούν αριθμητικές πράξεις. Οι καταχωρητές `$s0-$s7` αντιστοιχούν στους αριθμούς 16-23, και οι `$t0-$t7` στους 8-15.
2^{30} λέξεις μνήμης	Memory[0], Memory[4], . . ., Memory[4294967292]	Προσπελάζονται μόνον από εντολές μεταφοράς δεδομένων στο MIPS. Ο MIPS χρησιμοποιεί διευθύνσεις byte, και έτσι διαδοχικές διευθύνσεις λέξης διαφέρουν κατά 4. Η μνήμη κρατά δομές δεδομένων, πίνακες και «διασκορπισμένους» (spilled) καταχωρητές.

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
Λογικές πράξεις	and	`and $s1,$s2,$s3`	`$s1 = $s2 & $s3`	Τρεις τελεστέοι καταχωρητή· AND bit προς bit
	or	`or $s1,$s2,$s3`	`$s1 = $s2 | $s3`	Τρεις τελεστέοι καταχωρητή· OR bit προς bit
	nor	`nor $s1,$s2,$s3`	`$s1 = ~ ($s2 | $s3)`	Τρεις τελεστέοι καταχωρητή· NOR bit προς bit
	and immediate	`andi $s1,$s2,100`	`$s1 = $s2 & 100`	AND bit προς bit καταχωρητή με σταθερά
	or immediate	`ori $s1,$s2,100`	`$s1 = $s2 | 100`	OR bit προς bit καταχωρητή 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
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

ΕΙΚΟΝΑ 2.10 Η αρχιτεκτονική του MIPS που έχουμε αποκαλύψει μέχρι εδώ. Το χρώμα δείχνει τα μέρη που προστέθηκαν από την Εικόνα 2.7 της σελίδας 85. Στην αρχή τού βιβλίου θα βρείτε επίσης κατάλογο εντολών της γλώσσας μηχανής του MIPS.

Η χρησιμότητα ενός αυτόματου υπολογιστή βρίσκεται στη δυνατότητα χρήσης μιας δεδομένης ακολουθίας εντολών επαναληπτικά, όπου ο αριθμός των επαναλήψεων εξαρτάται από τα αποτελέσματα του υπολογισμού. Όταν ολοκληρωθεί η επανάληψη, εφαρμόζεται μια διαφορετική ακολουθία [εντολών] και, έτσι, στις περισσότερες περιπτώσεις πρέπει να δώσουμε δύο παράλληλες σειρές [εντολών] των οποίων θα προηγείται μια εντολή η οποία καθορίζει ποια ρουτίνα θα ακολουθηθεί. Αυτή η επιλογή μπορεί να εξαρτάται από το πρόσημο ενός αριθμού (με το μηδέν να θεωρείται συν για τους σκοπούς της μηχανής). Κατά συνέπεια, εισάγουμε [μια εντολή] (την [εντολή] μετάβασης υπό συνθήκη) η οποία, με βάση το πρόσημο ενός δεδομένου αριθμού, προκαλεί την εκτέλεση της κατάλληλης από τις δύο ρουτίνες.

Burks, Goldstine, και von Neumann, 1947

2.6 Εντολές λήψης αποφάσεων

Αυτό που διακρίνει έναν υπολογιστή από μια απλή αριθμομηχανή είναι η ικανότητά του να παίρνει αποφάσεις. Με βάση τα δεδομένα εισόδου και τις τιμές που παράγονται κατά τον υπολογισμό, εκτελούνται διαφορετικές εντολές. Η λήψη αποφάσεων αντιπροσωπεύεται συνήθως στις γλώσσες προγραμματισμού με τη χρήση της εντολής *if*, μερικές φορές συνδυασμένης με εντολές *go to* και ετικέτες (labels). Η συμβολική γλώσσα του MIPS περιλαμβάνει δύο εντολές λήψης αποφάσεων, ανάλογες με μία εντολή *if* με ένα *go to*. Η πρώτη εντολή είναι η

```
beq register1, register2, L1
```

Αυτή η εντολή σημαίνει «πήγαινε στην εντολή με την ετικέτα `L1` αν η τιμή στον καταχωρητή `register1` είναι ίση με την τιμή στον καταχωρητή `register2`». Το μνημονικό όνομα `beq` σημαίνει *branch if equal* (διακλάδωση σε περίπτωση ισότητας). Η δεύτερη εντολή είναι η

```
bne register1, register2, L1
```

Αυτή η εντολή σημαίνει «πήγαινε στην εντολή με ετικέτα `L1` αν η τιμή στον καταχωρητή `register1` *δεν* είναι ίση με την τιμή του καταχωρητή `register2`». Το μνημονικό όνομα `bne` σημαίνει *branch if not equal* (διακλάδωση σε περίπτωση μη ισότητας). Αυτές οι δύο εντολές ονομάζονται παραδοσιακά **διακλαδώσεις υπό συνθήκη** (conditional branches).

Μεταγλώττιση *if-then-else* σε διακλαδώσεις υπό συνθήκη

ΠΑΡΑΔΕΙΓΜΑ

Στο επόμενο τμήμα κώδικα, οι `f`, `g`, `h`, `i`, και `j` είναι μεταβλητές. Αν οι πέντε μεταβλητές `f` έως `j` αντιστοιχούν στους πέντε καταχωρητές `$s0` έως `$s4`, ποιος είναι ο μεταγλωττισμένος κώδικας MIPS γι' αυτή την εντολή *if* της C;

```
if (i == j) f = g + h; else f = g - h;
```

ΑΠΑΝΤΗΣΗ

Η Εικόνα 2.11 είναι ένα διάγραμμα ροής (flowchart) που δείχνει τι πρέπει να κάνει ο κώδικας MIPS. Η πρώτη παράσταση συγκρίνει για ισότητα, οπότε φαίνεται ότι θα θέλαμε την εντολή `beq`. Γενικά, ο κώδικας θα είναι πιο αποδοτικός αν ελέγξουμε για την αντίθετη συνθήκη ώστε η διακλάδωση να παρακάμψει τον κώδικα που εκτελεί το επόμενο τμήμα *then* της *if* (η ετικέτα `Else` ορίζεται παρακάτω):

```
bne $s3,$s4,Else        # πήγαινε στο Else αν i ≠ j
```

Η επόμενη εντολή ανάθεσης τιμής εκτελεί μία μόνο λειτουργία, και αν όλοι οι τελεστέοι ανατεθούν σε καταχωρητές, είναι μόνο μία εντολή:

```
add $s0,$s1,$s2  # f = g + h (παραλείπεται αν i ≠ j)
```

Τώρα χρειάζεται να πάμε στο τέλος της εντολής *if*. Το παράδειγμα αυτό εισάγει άλλον έναν τύπο διακλάδωσης, που συχνά ονομάζεται *διακλάδωση χωρίς συνθήκη* (unconditional branch). Αυτή η εντολή λέει ότι ο επεξεργαστής ακολουθεί πάντοτε τη διακλάδωση. Για να διακρίνει τις διακλαδώσεις με συνθήκη από τις χωρίς συνθήκη, το όνομα του MIPS γι' αυτόν τον τύπο εντολής είναι *jump* (άλμα), που συντομεύεται σε `j` (η ετικέτα `Exit` ορίζεται παρακάτω).

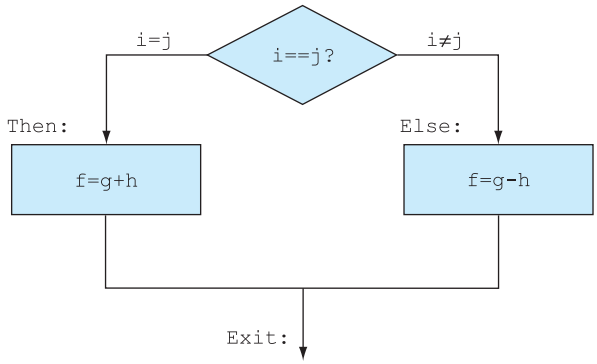
```
j Exit    # μετάβαση στην Exit
```

Η εντολή ανάθεσης τιμής στο τμήμα *else* της εντολής *if* μπορεί και πάλι να μεταγλωττιστεί σε μία μόνον εντολή. Απλώς χρειάζεται να προσαρτήσουμε την ετικέτα `Else` στην εντολή αυτή. Δείχνουμε επίσης την ετικέτα `Exit` η οποία βρίσκεται μετά από την εντολή αυτή, ορίζοντας το τέλος του μεταγλωττισμένου κώδικα της δομής *if-then-else*:

```
Else:sub $s0,$s1,$s2 # f = g - h (παρακάμπτεται αν i = j)
Exit:
```

διακλάδωση υπό συνθήκη (conditional branch) Μια εντολή που απαιτεί τη σύγκριση δύο τιμών και επιτρέπει τη μεταφορά του ελέγχου σε μια νέα διεύθυνση στο πρόγραμμα με βάση το αποτέλεσμα της σύγκρισης.

Παρατηρήστε ότι ο συμβολομεταφραστής (assembler) απαλλάσσει το μεταγλωττιστή και τον προγραμματιστή της συμβολικής γλώσσας από το ανιαρό έργο του υπολογισμού διευθύνσεων για τις διακλαδώσεις, όπως ακριβώς κάνει και στον υπολογισμό διευθύνσεων δεδομένων για τις εντολές load και store (δείτε την Ενότητα 2.10).



ΕΙΚΟΝΑ 2.11 Απεικόνιση των επιλογών στην παραπάνω εντολή *if*. Το αριστερό ορθογώνιο αντιστοιχεί στο τμήμα *then* της εντολής *if* και το δεξιό αντιστοιχεί στο τμήμα *else*.

Διασύνδεση υλικού και λογισμικού

Οι μεταγλωττιστές συχνά δημιουργούν διακλαδώσεις και ετικέτες εκεί που δεν εμφανίζονται στη γλώσσα προγραμματισμού. Η αποφυγή του φορτίου της γραφής ρητών ετικετών και διακλαδώσεων είναι ένα από τα πλεονεκτήματα της γραφής προγραμμάτων σε γλώσσες προγραμματισμού υψηλού επιπέδου, και ένας λόγος για τον οποίο η γραφή κώδικα είναι ταχύτερη σε αυτό το επίπεδο.

Βρόχοι

Οι αποφάσεις είναι σημαντικές τόσο για την επιλογή μεταξύ δύο εναλλακτικών — όπως στις εντολές *if* — όσο και για την επανάληψη ενός υπολογισμού — όπως στους βρόχους (loops). Οι ίδιες συμβολικές εντολές είναι τα δομικά στοιχεία και για τις δύο περιπτώσεις.

Μεταγλώττιση ενός βρόχου *while* της C

ΠΑΡΑΔΕΙΓΜΑ

Ένας παραδοσιακός βρόχος της C είναι ο εξής:

```
while (save[i] == k)
       i += 1;
```

Υποθέστε ότι το `i` και το `k` αντιστοιχούν στους καταχωρητές `$s3` και `$s5`, και η βάση του πίνακα `save` είναι στον καταχωρητή `$s6`. Ποιος είναι ο συμβολικός κώδικας MIPS που αντιστοιχεί στο τμήμα αυτό της C;

ΑΠΑΝΤΗΣΗ

Το πρώτο βήμα είναι να φορτώσουμε το στοιχείο `save[i]` σε έναν προσωρινό καταχωρητή. Για να μπορέσουμε να φορτώσουμε το `save[i]` σε έναν προσωρινό καταχωρητή, χρειαζόμαστε τη διεύθυνσή του. Για να μπορέσουμε να προσθέσουμε το `i` στη βάση του πίνακα `save` ώστε να σχηματίσουμε τη διεύθυνση, πρέπει να πολλαπλασιάσουμε τον αριθμοδείκτη `i` με 4 εξαιτίας του προβλήματος της διευθυνσιοδότησης κατά byte. Ευτυχώς, μπορούμε να χρησιμοποιήσουμε την αριστερή λογική ολίσθηση (shift left logical) αφού η αριστερή ολίσθηση κατά 2 bit πολλαπλασιάζει επί 4 (δείτε τη σελίδα 87 στην Ενότητα 2.5). Στην εντολή πρέπει να προσθέσουμε την ετικέτα `Loop`, ώστε να μπορούμε να επιστρέψουμε σε αυτή στο τέλος του βρόχου:

```
Loop: sll $t1,$s3,2  # προσωρινός καταχωρητής $t1 = 4 * i
```

Για να πάρουμε τη διεύθυνση του στοιχείου `save[i]`, πρέπει να προσθέσουμε τον καταχωρητή `$t1` και τη βάση του πίνακα `save` στον καταχωρητή `$s6`:

```
add $t1,$t1,$s6  # $t1 = διεύθυνση του save[i]
```

Τώρα μπορούμε να χρησιμοποιήσουμε αυτή τη διεύθυνση για να φορτώσουμε το στοιχείο `save[i]` σε έναν προσωρινό καταχωρητή:

```
lw $t0,0($t1)  # προσωρινός καταχωρητής # $t0 = save[i]
```

Η επόμενη εντολή εκτελεί τον έλεγχο του βρόχου, προκαλώντας έξοδο αν `save[i] ≠ k`:

```
bne $t0,$s5, Exit    # μετάβαση στο Exit αν save[i] ≠ k
```

Η επόμενη εντολή προσθέτει `1` στον αριθμοδείκτη `i`:

```
addi $s3,$s3,1       # i = i + 1
```

Στο τέλος του βρόχου, η ροή επιστρέφει στον έλεγχο *while* στην κορυφή του βρόχου. Απλώς προσθέτουμε την ετικέτα `Exit` μετά από αυτό, και τελειώσαμε:

```
    j Loop           # πήγαινε στο Loop
Exit:
```

(Δείτε την Άσκηση 2.33 για μια βελτιστοποίηση αυτής της ακολουθίας.)

Διασύνδεση υλικού και λογισμικού

Τέτοιες ακολουθίες εντολών που τελειώνουν σε διακλάδωση είναι τόσο θεμελιώδεις στη μεταγλώττιση που τους δίνεται η δική τους κωδική λέξη: **βασικό μπλοκ** (basic block) είναι μια ακολουθία εντολών χωρίς διακλαδώσεις, εκτός πιθανόν από το τέλος της, και χωρίς προορισμούς διακλάδωσης (branch targets) ή ετικέτες διακλάδωσης, εκτός πιθανόν από την αρχή. Σε κάποια από τις πρώτες φάσεις της μεταγλώττισης, το πρόγραμμα διαιρείται σε βασικά μπλοκ.

βασικό μπλοκ (basic block) Μια ακολουθία εντολών χωρίς διακλαδώσεις (εκτός ίσως από το τέλος) και χωρίς προορισμούς διακλάδωσης ή ετικέτες διακλάδωσης (εκτός ίσως από την αρχή).

Ο έλεγχος για ισότητα ή μη ισότητα είναι πιθανόν ο πιο δημοφιλής έλεγχος, αλλά μερικές φορές είναι χρήσιμο να δούμε αν μια μεταβλητή είναι μικρότερη από μια άλλη μεταβλητή. Για παράδειγμα, ένας βρόχος *for* μπορεί να πρέπει να ελέγξει αν μια μεταβλητή αριθμοδείκτη είναι μικρότερη από το 0. Τέτοιες συγκρίσεις πραγματοποιούνται στη συμβολική γλώσσα του MIPS με μια εντολή που συγκρίνει δύο καταχωρητές και θέτει έναν τρίτο καταχωρητή ίσο με 1 αν ο πρώτος είναι μικρότερος από το δεύτερο· διαφορετικά τον θέτει ίσο με 0. Η εντολή του MIPS ονομάζεται *set on less than* ή `slt`. Για παράδειγμα, η εντολή

```
slt $t0, $s3, $s4
```

σημαίνει ότι ο καταχωρητής `$t0` τίθεται ίσος με 1 αν η τιμή στον καταχωρητή `$s3` είναι μικρότερη από την τιμή στον καταχωρητή `$s4`, αλλιώς ο καταχωρητής `$t0` τίθεται ίσος με 0.

Οι τελεστέοι σταθερών είναι δημοφιλείς στις συγκρίσεις. Αφού ο καταχωρητής `$zero` είναι πάντα 0, μπορούμε ήδη να συγκρίνουμε με το 0. Για να συγκρίνουμε με άλλες τιμές, υπάρχει μια άμεση έκδοση της εντολής set on less than. Για να ελέγξουμε αν ο καταχωρητής `$s2` είναι μικρότερος από τη σταθερά 10, μπορούμε απλώς να γράψουμε:

```
slti $t0,$s2,10    # $t0 = 1 αν $s2 < 10
```

Σεβόμενη την προειδοποίηση του von Neumann σχετικά με την απλότητα του «εξοπλισμού», η αρχιτεκτονική του MIPS δεν περιλαμβάνει διακλάδωση σε περίπτωση μικρότερου (branch on less than) επειδή είναι υπερβολικά πολύπλοκη· είτε θα αύξανε το χρόνο του κύκλου ρολογιού είτε θα έπαιρνε επιπλέον κύκλους ρολογιού ανά εντολή. Δύο ταχύτερες εντολές είναι πιο χρήσιμες.

Διασύνδεση υλικού και λογισμικού

Οι μεταγλωττιστές τού MIPS χρησιμοποιούν τις εντολές `slt`, `slti`, `beq`, `bne`, και τη σταθερή τιμή 0 (πάντα διαθέσιμη με την ανάγνωση του καταχωρητή `$zero`) για να δημιουργήσουν όλες τις σχετικές συνθήκες: ισότητα, μη ισότητα, μικρότερο από, μικρότερο από ή ίσο, μεγαλύτερο από, μεγαλύτερο από ή ίσο. (Όπως ίσως αναμένατε, ο καταχωρητής `$zero` αντιστοιχεί στον καταχωρητή 0.)

Η εντολή case/switch

Οι περισσότερες γλώσσες προγραμματισμού έχουν μια εντολή *case* ή *switch* που επιτρέπει στον προγραμματιστή να επιλέξει μία από πολλές εναλλακτικές επιλογές με βάση μία μόνο τιμή. Ο απλούστερος τρόπος να υλοποιηθεί η *switch* είναι μέσω μιας σειράς ελέγχων συνθήκης, όπου η εντολή *switch* μετατρέπεται σε μια αλυσίδα από εντολές *if-then-else*.

πίνακας διευθύνσεων άλματος (jump address table) Ονομάζεται επίσης πίνακας αλμάτων (jump table). Ένας πίνακας διευθύνσεων εναλλακτικών ακολουθιών εντολών.

Μερικές φορές οι εναλλακτικές επιλογές μπορούν να κωδικοποιηθούν πιο αποδοτικά ως πίνακας διευθύνσεων εναλλακτικών ακολουθιών εντολών, που ονομάζεται **πίνακας διευθύνσεων άλματος** (jump address table) και το πρόγραμμα χρειάζεται μόνο τον αριθμοδείκτη του πίνακα και μετά πραγματοποιεί άλμα στην κατάλληλη ακολουθία. Ο πίνακας αλμάτων είναι τότε απλώς ένας πίνακας λέξεων οι οποίες περιέχουν διευθύνσεις που αντιστοιχούν σε ετικέτες στον κώδικα. Δείτε τις ασκήσεις Σε μεγαλύτερο βάθος της Ενότητας 2.20 για περισσότερες λεπτομέρειες σχετικά με τους πίνακες διευθύνσεων αλμάτων.

Για την υποστήριξη τέτοιων καταστάσεων, οι υπολογιστές όπως ο MIPS περιλαμβάνουν μια εντολή *jump register* (`jr`), που σημαίνει ένα άλμα χωρίς συνθήκη σε μια διεύθυνση που καθορίζεται σε έναν καταχωρητή. Το πρόγραμμα φορτώνει την κατάλληλη καταχώριση, από τον πίνακα αλμάτων, σε έναν καταχωρητή και μεταπηδά στην κατάλληλη διεύθυνση χρησιμοποιώντας μια εντολή jump register. Αυτή η εντολή περιγράφεται στην Ενότητα 2.7.

Διασύνδεση υλικού και λογισμικού

Αν και υπάρχουν πολλές εντολές αποφάσεων και βρόχων στις γλώσσες προγραμματισμού όπως οι C και Java, η θεμελιώδης εντολή που τις υλοποιεί στο επόμενο χαμηλότερο επίπεδο είναι η διακλάδωση υπό συνθήκη.

Η Εικόνα 2.12 συνοψίζει τα μέρη της συμβολικής γλώσσας του MIPS που περιγράψαμε στην ενότητα αυτή, και η Εικόνα 2.13 την αντίστοιχη γλώσσα μηχανής του MIPS. Το βήμα αυτό κατά την εξέλιξη της γλώσσας του MIPS έχει προσθέσει διακλαδώσεις και άλματα στη συμβολική μας αναπαράσταση, και έχει σταθεροποιήσει τη χρήσιμη τιμή 0 σε έναν καταχωρητή.

Επιπλέον ανάπτυξη: Αν έχετε ακούσει για τις *καθυστερημένες* (ή *όψιμες*) *διακλαδώσεις* (delayed branches), οι οποίες καλύπτονται στο Κεφάλαιο 6, μην ανησυχείτε: ο συμβολομεταφραστής τού MIPS τις κάνει αόρατες στον προγραμματιστή συμβολικής γλώσσας.

Τελεστέοι του MIPS

Όνομα	Παράδειγμα	Σχόλια
32 καταχωρητές	`$s0,$s1,...,$s7` `$t0,$t1,,$t7,` `$zero`	Γρήγορες θέσεις για δεδομένα. Στο MIPS, τα δεδομένα πρέπει να βρίσκονται σε καταχωρητές για να εκτελεστούν αριθμητικές πράξεις. Οι καταχωρητές `$s0-$s7` αντιστοιχούν στους αριθμούς 16-23, και οι `$t0-$t7` στους 8-15. Ο καταχωρητής `$zero` του MIPS είναι πάντα ίσος με 0.
2^{30} λέξεις μνήμης	Memory[0], Memory[4], . . ., Memory[4294967292]	Προσπελάζονται μόνον από εντολές μεταφοράς δεδομένων στο MIPS. Ο MIPS χρησιμοποιεί διευθύνσεις byte, οπότε διαδοχικές διευθύνσεις λέξεων διαφέρουν κατά 4. Η μνήμη κρατά δομές δεδομένων, πίνακες και «διασκορπισμένους» (spilled) καταχωρητές.

Συμβολική γλώσσα του MIPS

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Αριθμητικές πράξεις	add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Τρεις τελεστέοι· δεδομένα σε καταχωρητές
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2 - $s3`	Τρεις τελεστέοι· δεδομένα σε καταχωρητές
Μεταφορά δεδομένων	load word	`lw $s1,100($s2)`	`$s1 = Memory[$s2+100]`	Δεδομένα από τη μνήμη σε καταχωρητή
	store word	`sw $s1,100($s2)`	`Memory[$s2+100] = $s1`	Δεδομένα από καταχωρητή στη μνήμη
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 καταχωρητή με σταθερά
	or immediate	`ori $s1,$s2,100`	`$s1 = $s2 | 100`	OR bit προς bit καταχωρητή 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 με συνθήκη	branch on equal	`beq $s1,$s2,L`	αν `($s1 == $s2)` πήγαινε στο L	Έλεγχος ισότητας και διακλάδωση
	branch on not equal	`bne $s1,$s2,L`	αν `($s1 != $s2)` πήγαινε στο L	Έλεγχος μη ισότητας και διακλάδωση
	set on less than	`slt $s1,$s2,$s3`	αν `($s2 < $s3)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από· χρήση με τις `beq, bne`
	set on less than immediate	`slti $s1,$s2,100`	αν `($s2 < 100)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Άμεση σύγκριση μικρότερο από· χρήση με τις `beq, bne`
Άλμα χωρίς συνθήκη	jump	`j L`	πήγαινε στο L	Άλμα στη διεύθυνση προορισμού

ΕΙΚΟΝΑ 2.12 Αρχιτεκτονική του MIPS που έχουμε αποκαλύψει μέχρι την Ενότητα 2.6. Τα επισημασμένα σημεία δείχνουν τις δομές του MIPS που παρουσιάσαμε για πρώτη φορά στην Ενότητα 2.6.

Αυτοεξέταση

Η C έχει πολλές εντολές για αποφάσεις και βρόχους, ενώ ο MIPS λίγες. Ποια από τα επόμενα εξηγούν ή όχι αυτή την ανισορροπία; Γιατί;

1. Περισσότερες εντολές απόφασης κάνουν τον κώδικα πιο ευανάγνωστο και εύληπτο.

Γλώσσα μηχανής του MIPS

Όνομα	Μορφή	Παράδειγμα						Σχόλια
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
and	R	0	18	19	17	0	36	and $s1,$s2,$s3
or	R	0	18	19	17	0	37	or $s1,$s2,$s3
nor	R	0	18	19	17	0	39	nor $s1,$s2,$s3
andi	I	12	18	17	100			andi $s1,$s2,100
ori	I	13	18	17	100			ori $s1,$s2,100
sll	R	0	0	18	17	10	0	sll $s1,$s2,10
srl	R	0	0	18	17	10	2	srl $s1,$s2,10
beq	I	4	17	18	25			beq $s1,$s2,100
bne	I	5	17	18	25			bne $s1,$s2,100
slt	R	0	18	19	17	0	42	slt $s1,$s2,$s3
j	J	2	2500					j 10000 (δείτε την Ενότητα 2.9)
Μέγεθος πεδίου		6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	Όλες οι εντολές του MIPS 32 bit
Μορφή R	R	op	rs	rt	rd	shamt	funct	Μορφή αριθμητική εντολής
Μορφή I	I	op	rs	rt	address			Μορφή μεταφοράς δεδομένων και διακλάδωσης

ΕΙΚΟΝΑ 2.13 Γλώσσα μηχανής του MIPS που έχει αποκαλυφθεί στην Ενότητα 2.6. Τα επισημασμένα σημεία δείχνουν τις δομές του MIPS που πρωτοπαρουσιάσαμε στην Ενότητα 2.6. Η μορφή J (J-format), που χρησιμοποιείται για τις εντολές άλματος (jump), εξηγείται στην Ενότητα 2.9. Η Ενότητα 2.9 εξηγεί επίσης τις κατάλληλες τιμές στα πεδία διευθύνσεων των εντολών διακλάδωσης.

2. Λιγότερες εντολές αποφάσεων απλοποιούν την εργασία του υποκείμενου επιπέδου που είναι υπεύθυνο για την εκτέλεση.
3. Περισσότερες εντολές αποφάσεων σημαίνουν λιγότερες γραμμές κώδικα, κάτι που γενικά μειώνει το χρόνο γραφής κώδικα.
4. Περισσότερες εντολές αποφάσεων σημαίνουν λιγότερες γραμμές κώδικα, πράγμα που γενικά έχει ως αποτέλεσμα την εκτέλεση λιγότερων πράξεων (λειτουργιών).

Γιατί η C παρέχει δύο σύνολα τελεστών για το AND (& και &&) και δύο σύνολα τελεστών για το OR (| και ||) ενώ ο MIPS δεν το κάνει;

1. Οι λογικές πράξεις (λειτουργίες) AND και OR υλοποιούν τα & και | ενώ οι διακλαδώσεις υπό συνθήκη υλοποιούν τα && και ||.
2. Η προηγούμενη πρόταση έχει και την αντίστροφή της: οι && και || αντιστοιχούν σε λογικές πράξεις ενώ οι & και | αντιστοιχίζονται σε διακλαδώσεις υπό συνθήκη.
3. Είναι πλεονασμός και σημαίνουν το ίδιο πράγμα: οι && και || απλώς έχουν κληρονομηθεί από τη γλώσσα προγραμματισμού B, τον πρόγονο της C.

2.7 Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Μια **διαδικασία** (procedure) ή συνάρτηση (function) είναι ένα εργαλείο που οι προγραμματιστές C ή Java χρησιμοποιούν στα προγράμματά τους, τόσο για να τα κάνουν πιο εύκολα κατανοητά όσο και για να δώσουν τη δυνατότητα επαναχρησιμοποίησης του κώδικα. Οι διαδικασίες επιτρέπουν στον προγραμματιστή να επικεντρώνεται σε ένα μόνο τμήμα του έργου κάθε φορά, με παραμέτρους που λειτουργούν σαν φράγμα ανάμεσα στη διαδικασία και το υπόλοιπο του προγράμματος και των δεδομένων, δεχόμενες τιμές και επιστρέφοντας αποτελέσματα. Περιγράφουμε το ισοδύναμο σε Java στο τέλος αυτής της ενότητας, αλλά η Java χρειάζεται όλα όσα χρειάζεται από έναν υπολογιστή και η C.

διαδικασία (procedure) Μια αποθηκευμένη υπορουτίνα που εκτελεί ένα συγκεκριμένο έργο με βάση τις παραμέτρους τις οποίες δέχεται ως είσοδο.

Μπορείτε να θεωρήσετε μια διαδικασία ως έναν κατάσκοπο ο οποίος φεύγει με ένα μυστικό σχέδιο, βρίσκει πόρους, κάνει το έργο του, καλύπτει τα ίχνη του, και μετά επιστρέφει στην αφετηρία του με το επιθυμητό αποτέλεσμα. Τίποτε άλλο δεν πρέπει να αλλάξει από τη στιγμή που η αποστολή έχει ολοκληρωθεί. Επιπλέον, ένας κατάσκοπος λειτουργεί μόνο σε βάση «πρέπει να ξέρω», και έτσι δεν μπορεί να κάνει υποθέσεις για τον εργοδότη του.

Παρόμοια, στην εκτέλεση μιας διαδικασίας, το πρόγραμμα πρέπει να ακολουθήσει τα επόμενα έξι βήματα:

1. Τοποθέτηση των παραμέτρων σε ένα μέρος όπου η διαδικασία μπορεί να τις προσπελάσει.
2. Μεταβίβαση του ελέγχου στη διαδικασία.
3. Λήψη των πόρων αποθήκευσης που χρειάζεται η διαδικασία.
4. Εκτέλεση της επιθυμητής εργασίας.
5. Τοποθέτηση της τιμής του αποτελέσματος σε ένα μέρος όπου το καλούν πρόγραμμα (calling program) μπορεί να την προσπελάσει.
6. Επιστροφή του ελέγχου στο σημείο εκκίνησης, αφού η διαδικασία μπορεί να κληθεί από διάφορα σημεία σε ένα πρόγραμμα.

Όπως είπαμε πιο πάνω, οι καταχωρητές είναι το ταχύτερο μέρος διατήρησης δεδομένων σε έναν υπολογιστή, οπότε πρέπει να τους χρησιμοποιούμε όσο το δυνατόν περισσότερο. Το λογισμικό του MIPS ακολουθεί την παρακάτω σύμβαση στην κατανομή των 32 καταχωρητών του για την κλήση διαδικασιών:

- `$a0-$a3`: τέσσερις καταχωρητές ορίσματος (argument registers) στους οποίους μεταβιβάζονται οι παράμετροι
- `$v0-$v1`: δύο καταχωρητές τιμής (value registers) στους οποίους επιστρέφονται τιμές
- `$ra`: ένας καταχωρητής διεύθυνσης επιστροφής (return address register) για την επιστροφή στην αφετηρία

εντολή άλματος και σύνδεσης (jump-and-link instruction) Μια εντολή που μεταφέρεται σε μια διεύθυνση και ταυτόχρονα αποθηκεύει τη διεύθυνση της επόμενης εντολής σε έναν καταχωρητή (τον `$ra` στον MIPS).

Εκτός από την κατανομή αυτών των καταχωρητών, η συμβολική γλώσσα του MIPS περιέχει μια εντολή μόνο για τις διαδικασίες: μεταφέρεται σε μια διεύθυνση και ταυτόχρονα αποθηκεύει τη διεύθυνση της επόμενης εντολής στον καταχωρητή `$ra`. Η **εντολή άλματος και σύνδεσης** (jump-and-link — `jal`) γράφεται απλώς

```
jal ΔιεύθυνσηΔιαδικασίας
```

Το τμήμα *link* (σύνδεση) του ονόματος σημαίνει ότι σχηματίζεται μια διεύθυνση ή σύνδεση που δείχνει στη θέση κλήσης, ώστε να δώσει τη δυνατότητα στη διαδικασία να επιστρέψει στην κατάλληλη διεύθυνση. Αυτή η «σύνδεση», που αποθηκεύεται στον καταχωρητή `$ra`, ονομάζεται **διεύθυνση επιστροφής** (return address). Η διεύθυνση επιστροφής χρειάζεται, επειδή η ίδια διαδικασία μπορεί να κληθεί από διάφορα μέρη του προγράμματος.

διεύθυνση επιστροφής (return address) Ένας σύνδεσμος με το σημείο κλήσης, που επιτρέπει σε μια διαδικασία να επιστρέψει στην κατάλληλη διεύθυνση· στον MIPS αποθηκεύεται στον καταχωρητή `$ra`.

μετρητής προγράμματος (program counter — PC) Ο καταχωρητής που περιέχει τη διεύθυνση της εκτελούμενης εντολής του προγράμματος.

Στην έννοια του αποθηκευμένου προγράμματος εξυπακούεται η ανάγκη ύπαρξης ενός καταχωρητή ο οποίος να κρατάει τη διεύθυνση της εντολής που εκτελείται. Για ιστορικούς λόγους, ο καταχωρητής αυτός σχεδόν πάντα λέγεται **μετρητής προγράμματος** (program counter), και συντομεύεται σε *PC* στην αρχιτεκτονική MIPS, παρόλο που ένα πιο λογικό όνομα θα ήταν *καταχωρητής διεύθυνσης εντολής* (instruction address register). Η εντολή `jal` αποθηκεύει την τιμή PC+4 στον καταχωρητή `$ra` για να δημιουργήσει ένα σύνδεσμο προς την επόμενη εντολή και να ρυθμίσει την επιστροφή της διαδικασίας.

Για να υποστηρίξουν τέτοιες καταστάσεις, οι υπολογιστές όπως ο MIPS χρησιμοποιούν μια εντολή *jump register* (`jr`), που σημαίνει άλμα χωρίς συνθήκη στη διεύθυνση που καθορίζεται σε έναν καταχωρητή:

```
jr $ra
```

Η εντολή jump register μεταφέρει τον έλεγχο στη διεύθυνση που περιέχει ο καταχωρητής `$ra` – πράγμα που είναι αυτό ακριβώς που θέλουμε. Έτσι, το καλούν πρόγραμμα, ή **καλών** (caller), τοποθετεί τις τιμές των παραμέτρων στους καταχωρητές `$a0-$a3` και χρησιμοποιεί την εντολή `jal X` για να μεταπηδήσει στη διαδικασία `X` (που μερικές φορές λέγεται **καλούμενος** — callee). Ο καλούμενος στη συνέχεια εκτελεί τους υπολογισμούς, τοποθετεί τα αποτελέσματα στους καταχωρητές `$v0-$v1`, και επιστρέφει τον έλεγχο στον καλούντα μέσω της εντολής `jr $ra`.

καλών (caller) Το πρόγραμμα που ενεργοποιεί μια διαδικασία και παρέχει τις αναγκαίες τιμές παραμέτρων.

καλούμενος (callee) Διαδικασία που εκτελεί μια σειρά αποθηκευμένων εντολών με βάση παραμέτρους που παρέχονται από τον καλούντα, και μετά επιστρέφει τον έλεγχο σε αυτόν.

Χρήση περισσότερων καταχωρητών

Ας υποθέσουμε ότι ένας μεταγλωττιστής χρειάζεται περισσότερους καταχωρητές για μια διαδικασία, από τους τέσσερις καταχωρητές ορισμάτων και τους δύο καταχωρητές επιστροφής τιμής. Αφού πρέπει να καλύψουμε τα ίχνη μας μετά την ολοκλήρωση της αποστολής, οποιοιδήποτε καταχωρητές χρειάζεται ο καλών πρέπει να επαναφέρονται στις τιμές που περιείχαν *πριν* κληθεί η διαδικασία. Αυτή η περίπτωση είναι ένα παράδειγμα όπου πρέπει να «σκορπίσουμε» καταχωρητές στη μνήμη, όπως αναφέραμε στην ενότητα «Διασύνδεση υλικού και λογισμικού» της σελίδας 76.

στοίβα (stack) Μια δομή δεδομένων για το διασκορπισμό καταχωρητών, οργανωμένη σε μια ουρά «τελευταίο μέσα πρώτο έξω» (last-in-first-out queue).

δείκτης στοίβας (stack pointer) Μια τιμή που δείχνει την πιο πρόσφατα κατανεμημένη διεύθυνση σε μια στοίβα η οποία δείχνει πού πρέπει να διασκορπιστούν οι καταχωρητές ή πού μπορούν να βρεθούν προηγούμενες τιμές καταχωρητών.

Η ιδανική δομή δεδομένων για το διασκορπισμό καταχωρητών είναι μια **στοίβα** (stack) — μια ουρά «τελευταίο μέσα πρώτο έξω» (last-in-first-out). Η στοίβα χρειάζεται ένα δείκτη προς την πιο πρόσφατα κατανεμημένη διεύθυνση στη στοίβα για να δείξει το σημείο στο οποίο η επόμενη διαδικασία πρέπει να τοποθετήσει τους καταχωρητές οι οποίοι θα διασκορπιστούν ή το σημείο που βρίσκονται οι παλιές τιμές των καταχωρητών. Ο **δείκτης στοίβας** (stack pointer) προσαρμόζεται ανά μία λέξη για κάθε καταχωρητή που αποθηκεύεται ή επαναφέρεται. Οι στοίβες είναι τόσο δημοφιλείς που έχουν τις δικές τους συνθηματικές λέξεις για τη μεταφορά δεδομένων από και προς αυτές: η προσθήκη δεδομένων στη στοίβα ονομάζεται *τοποθέτηση* (push) και η αφαίρεση δεδομένων από τη στοίβα ονομάζεται *εξαγωγή* (pop).

Το λογισμικό του MIPS εκχωρεί άλλον έναν καταχωρητή μόνο για τη στοίβα: το δείκτη στοίβας (stack pointer — `$sp`), που χρησιμοποιείται για την αποθήκευση των καταχωρητών τους οποίους χρειάζεται ο καλούμενος. Λόγω ιστορικού προηγουμένου, οι στοίβες «μεγαλώνουν» από τις υψηλότερες προς τις χαμηλότερες διευθύνσεις. Αυτή η σύμβαση σημαίνει ότι τοποθετεί (push) κανείς τιμές στη στοίβα αφαιρώντας από το δείκτη στοίβας (stack pointer). Η πρόσθεση στο δείκτη στοίβας συρρικνώνει τη στοίβα, εξάγοντας (pop) έτσι τιμές από τη στοίβα.

Μεταγλώττιση μιας διαδικασίας της C που δεν καλεί άλλη διαδικασία

ΠΑΡΑΔΕΙΓΜΑ

Ας μετατρέψουμε το παράδειγμα της σελίδας 69 σε μια διαδικασία της C:

```
int leaf_example (int g, int h, int i, int j)
{
    int f;

    f = (g + h) - (i + j);
    return f;
}
```

Ποιος είναι ο μεταγλωττισμένος κώδικας συμβολικής γλώσσας του MIPS;

ΑΠΑΝΤΗΣΗ

Οι μεταβλητές παράμετροι `g`, `h`, `i`, και `j` αντιστοιχούν στους καταχωρητές ορίσματος `$a0`, `$a1`, `$a2`, και `$a3`, και η `f` αντιστοιχεί στον `$s0`. Το μεταγλωττισμένο πρόγραμμα αρχίζει με την ετικέτα της διαδικασίας:

```
leaf_example:
```

Το επόμενο βήμα είναι να αποθηκεύσουμε τους καταχωρητές που χρησιμοποιούνται από τη διαδικασία. Η εντολή ανάθεσης τιμής της C στο σώμα της διαδικασίας είναι πανομοιότυπη με το παράδειγμα της σελίδας 69, που χρησιμοποιεί δύο προσωρινούς καταχωρητές. Έτσι, χρειάζεται να αποθηκεύσουμε τρεις καταχωρητές: `$s0`, `$t0`, και `$t1`. Τοποθετούμε τις παλιές τιμές στη στοίβα, δημιουργώντας σε αυτή χώρο για τρεις λέξεις, και μετά τις αποθηκεύουμε:

```
addi $sp,$sp,-12 # δημιουργία χώρου για 3 αντικείμενα στη
                 # στοίβα
sw $t1, 8($sp)   # αποθήκευση καταχωρητή $t1 για χρήση έπειτα
sw $t0, 4($sp)   # αποθήκευση καταχωρητή $t0 για χρήση έπειτα
sw $s0, 0($sp)   # αποθήκευση καταχωρητή $s0 για χρήση έπειτα
```

Η Εικόνα 2.14 δείχνει τη στοίβα πριν, κατά τη διάρκεια, και μετά την κλήση της διαδικασίας. Οι επόμενες τρεις εντολές αντιστοιχούν στο σώμα της διαδικασίας, που ακολουθεί το παράδειγμα της σελίδας 69:

```
add $t0, $a0, $a1 # ο καταχωρητής $t0 περιέχει το g+h
add $t1, $a2, $a3 # ο καταχωρητής $t1 περιέχει το i+j
sub $s0, $t0, $t1 # f = $t0-$t1, που είναι (g+h) - (i+j)
```

Για να επιστρέψουμε την τιμή της `f`, την αντιγράφουμε σε έναν καταχωρητή τιμής επιστροφής:

```
add $v0,$s0,$zero # επιστρέφει το f ($v0 = $s0 + 0)
```

Πριν επιστρέψουμε, επαναφέρουμε τις τρεις παλιές τιμές των καταχωρητών που αποθηκεύσαμε, εξάγοντάς τες από τη στοίβα:

```
lw $s0, 0($sp)  # επαναφορά καταχωρητή $s0 για τον καλούντα
lw $t0, 4($sp)  # επαναφορά καταχωρητή $t0 για τον καλούντα
lw $t1, 8($sp)  # επαναφορά καταχωρητή $t1 για τον καλούντα
addi $sp,$sp,12 # ρύθμιση στοίβας να διαγράψει 3 αντικείμενα
```

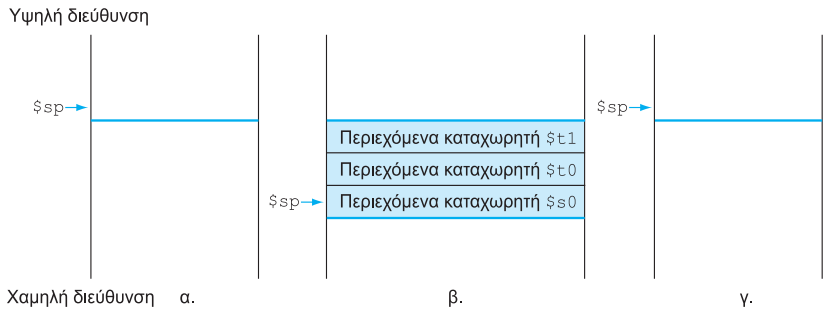
Η διαδικασία ολοκληρώνεται με μια εντολή jump register, με χρήση της διεύθυνσης επιστροφής:

```
jr $ra          # επιστροφή στην καλούσα ρουτίνα
```

Στο παραπάνω παράδειγμα χρησιμοποιήσαμε προσωρινούς καταχωρητές και υποθέσαμε ότι οι παλιές τους τιμές πρέπει να αποθηκευτούν και να επαναφερθούν. Για να αποφύγει την αποθήκευση και την επαναφορά ενός καταχωρητή του οποίου η τιμή δε χρησιμοποιείται ποτέ, κάτι που μπορεί να συμβεί με έναν προσωρινό καταχωρητή, το λογισμικό του MIPS διαχωρίζει 18 από τους καταχωρητές σε δύο ομάδες:

- `$t0-$t9`: 10 προσωρινοί (temporary) καταχωρητές που *δεν* διατηρούνται από την καλούμενη διαδικασία σε μια κλήση διαδικασίας
- `$s0-$s7`: 8 αποθηκευμένοι (saved) καταχωρητές που πρέπει να διατηρηθούν σε μια κλήση διαδικασίας (αν χρησιμοποιούνται, η καλούμενη διαδικασία τους αποθηκεύει και τους επαναφέρει)

Αυτή η απλή σύμβαση περιορίζει το διασκορπισμό των καταχωρητών. Στο παραπάνω παράδειγμα, εφόσον ο καλών (η καλούσα διαδικασία) δεν αναμένει ότι οι καταχωρητές `$t0` και `$t1` θα διατηρηθούν σε μια κλήση διαδικασίας, μπορούμε να γλιτώσουμε δύο αποθηκεύσεις (stores) και δύο φορτώσεις (loads) από τον κώδικα. Πρέπει να διατηρήσουμε την αποθήκευση και την επαναφορά του `$s0`, αφού ο καλούμενος πρέπει να υποθέσει ότι ο καλών χρειάζεται την τιμή του.



ΕΙΚΟΝΑ 2.14 Οι τιμές του δείκτη στοίβας (stack pointer) και της στοίβας (α) πριν, (β) κατά τη διάρκεια και (γ) μετά την κλήση της διαδικασίας. Ο δείκτης στοίβας πάντα δείχνει στην «κορυφή» της στοίβας, ή στην τελευταία λέξη της στοίβας σε αυτό το διάγραμμα.

Ένθετες διαδικασίες

Οι διαδικασίες που δεν καλούν άλλες λέγονται *διαδικασίες-φύλλα* (leaf procedures). Η ζωή θα ήταν απλή αν όλες οι διαδικασίες ήταν διαδικασίες-φύλλα, αλλά δεν είναι. Όπως ένας κατάσκοπος μπορεί να προσλάβει άλλους κατασκόπους ως μέρος μιας αποστολής, οι οποίοι με τη σειρά τους μπορεί να χρησιμοποιήσουν ακόμη περισσότερους κατασκόπους, έτσι και οι διαδικασίες καλούν άλλες διαδικασίες. Επιπλέον, οι αναδρομικές (recursive) διαδικασίες καλούν ακόμη και «κλώνους» του εαυτού τους. Όπως πρέπει να είμαστε προσεκτικοί όταν χρησιμοποιούμε καταχωρητές σε διαδικασίες, περισσότερη προσοχή πρέπει να δοθεί όταν καλούμε διαδικασίες μη φύλλα (nonleaf procedures).

Για παράδειγμα, υποθέστε ότι το κύριο πρόγραμμα καλεί τη διαδικασία A με ένα όρισμα 3, τοποθετώντας την τιμή 3 στον καταχωρητή `$a0` και μετά χρησιμοποιώντας την εντολή `jal A`. Υποθέστε κατόπιν ότι η διαδικασία A καλεί τη διαδικασία B μέσω της εντολής `jal B` με ένα όρισμα 7, που επίσης τοποθετείται στον `$a0`. Εφόσον η A δεν έχει ακόμη ολοκληρώσει το έργο της, υπάρχει μια διένεξη σχετικά με τη χρήση του καταχωρητή `$a0`. Παρόμοια, υπάρχει μια διένεξη σχετικά με τη διεύθυνση επιστροφής στον καταχωρητή `$ra`, αφού αυτός τώρα περιέχει τη διεύθυνση επιστροφής για τη B. Αν δεν πάρουμε μέτρα για να αποτρέψουμε αυτό το πρόβλημα, η διένεξη θα αφαιρέσει από τη διαδικασία A τη δυνατότητα να επιστρέψει στην καλούσα διαδικασία της.

Μια λύση είναι να τοποθετούμε όλους τους άλλους καταχωρητές που πρέπει να διατηρηθούν στη στοίβα, ακριβώς όπως κάναμε με τους αποθηκευμένους (saved) καταχωρητές. Ο καλών τοποθετεί (push) όποιους καταχωρητές ορίσματος (`$a0-$a3`) ή προσωρινούς καταχωρητές (`$t0-$t9`) χρειάζονται μετά την κλήση. Ο καλούμενος τοποθετεί τον καταχωρητή διεύθυνσης επιστροφής `$ra` και όποιους αποθηκευμένους καταχωρητές (`$s0-$s7`) χρησιμοποιούνται από τον καλούμενο. Ο δείκτης στοίβας `$sp` ρυθμίζεται να παρακολουθεί τον αριθμό των καταχωρητών που τοποθετούνται στη στοίβα. Στην επιστροφή, οι καταχωρητές επαναφέρονται από τη μνήμη και ο δείκτης στοίβας επαναρυθμίζεται.

Μεταγλώττιση μιας αναδρομικής διαδικασίας της C, που δείχνει τη σύνδεση ένθετων διαδικασιών

ΠΑΡΑΔΕΙΓΜΑ

Ας καταπιαστούμε με μια αναδρομική διαδικασία που υπολογίζει το παραγοντικό:

```
int fact (int n)
{
    if (n < 1) return (1);
          else return (n * fact(n-1));
}
```

Ποιος είναι ο κώδικας συμβολικής γλώσσας του MIPS;

ΑΠΑΝΤΗΣΗ

Η μεταβλητή παράμετρος n αντιστοιχεί στον καταχωρητή ορίσματος $a0. Το μεταγλωττισμένο πρόγραμμα ξεκινάει με μια ετικέτα της διαδικασίας και μετά αποθηκεύει δύο καταχωρητές στη στοίβα, τον καταχωρητή διεύθυνσης επιστροφής ($ra) και τον $a0:

```
fact:
    addi $sp,$sp,-8   # ρύθμιση της στοίβας για 2 αντικείμενα
    sw   $ra, 4($sp)  # αποθήκευση διεύθυνσης επιστροφής
    sw   $a0, 0($sp)  # αποθήκευση του ορίσματος n
```

Την πρώτη φορά που καλείται η fact, η sw αποθηκεύει μια διεύθυνση στο πρόγραμμα που κάλεσε τη fact. Οι επόμενες δύο εντολές ελέγχουν αν το n είναι μικρότερο από 1, και διακλαδίζονται στην ετικέτα L1 αν n ≥ 1.

```
    slti $t0,$a0,1     # έλεγχος αν n < 1
    beq  $t0,$zero,L1  # αν n >= 1, μετάβαση στην L1
```

Αν το n είναι μικρότερο από 1, η fact επιστρέφει 1 τοποθετώντας 1 σε έναν καταχωρητή τιμής: προσθέτει 1 στο 0 και βάζει το άθροισμα στον $v0. Στη συνέχεια, εξάγει τις δύο αποθηκευμένες τιμές στη στοίβα και μεταπηδά στη διεύθυνση επιστροφής:

```
    addi $v0,$zero,1   # επιστρέφει 1
    addi $sp,$sp,8     # εξάγει 2 αντικείμενα από τη στοίβα
    jr   $ra           # επιστροφή στην εντολή μετά την jal
```

Πριν από την εξαγωγή (pop) των δύο αντικειμένων από τη στοίβα, θα μπορούσαμε να έχουμε φορτώσει τους καταχωρητές $a0 και $ra. Επειδή οι $a0 και $ra δεν αλλάζουν όταν n είναι μικρότερο από 1, παραλείπουμε αυτές τις εντολές.

Αν το n δεν είναι μικρότερο από 1, το όρισμα n μειώνεται και καλείται πάλι η συνάρτηση fact με τη μειωμένη τιμή:

```
L1: addi $a0,$a0,-1   # n >= 1: το όρισμα παίρνει το (n – 1)
    jal fact          # κλήση της fact με (n – 1)
```

Η επόμενη εντολή είναι εκεί που επιστρέφει η fact. Τώρα επαναφέρονται η παλιά διεύθυνση επιστροφής και το παλιό όρισμα, μαζί με το δείκτη στοίβας:

```
lw   $a0, 0($sp)   # επιστροφή από την jal: επαναφορά
                   # του ορίσματος n
lw   $ra, 4($sp)   # επαναφορά της διεύθυνσης επιστροφής
addi $sp, $sp,8    # ρύθμιση δείκτη στοίβας για εξαγωγή 2
                   # αντικειμένων
```

Στη συνέχεια, ο καταχωρητής τιμής $v0 παίρνει το γινόμενο του παλιού ορίσματος $a0 και της τρέχουσας τιμής του καταχωρητή τιμής. Υποθέτουμε ότι είναι διαθέσιμη μια εντολή πολλαπλασιασμού, παρόλο που δεν καλύπτεται μέχρι το Κεφάλαιο 3:

```
    mul $v0,$a0,$v0     # επιστροφή n * fact (n - 1)
```

Τέλος, η `fact` διακλαδίζεται ξανά στη διεύθυνση επιστροφής:

```
    jr $ra              # επιστροφή στον καλούντα
```

Διασύνδεση υλικού και λογισμικού

Μια μεταβλητή της C είναι μια θέση αποθήκευσης, και η ερμηνεία της εξαρτάται τόσο από τον *τύπο* (type) της όσο και από την *τάξη αποθήκευσής* της (storage class). Οι τύποι αναλύονται με λεπτομέρειες στο Κεφάλαιο 3, αλλά τα παραδείγματα περιλαμβάνουν ακεραίους και χαρακτήρες. Η C έχει δύο τάξεις αποθήκευσης: την *αυτόματη* (automatic) και τη *στατική* (static). Οι αυτόματες μεταβλητές είναι τοπικές σε μια διαδικασία και απορρίπτονται κατά την έξοδο (exit) από τη διαδικασία. Οι στατικές μεταβλητές υπάρχουν μεταξύ εξόδων και εισόδων σε διαδικασίες. Οι μεταβλητές της C που δηλώνονται έξω από όλες τις διαδικασίες θεωρούνται στατικές, όπως και όποιες μεταβλητές δηλώνονται με τη λέξη-κλειδί (keyword) `static`. Οι υπόλοιπες είναι αυτόματες. Για να απλοποιήσει την πρόσβαση σε στατικά δεδομένα, το λογισμικό του MIPS δεσμεύει άλλον έναν καταχωρητή, που ονομάζεται **καθολικός δείκτης** (global pointer) ή `$gp`.

καθολικός δείκτης (global pointer) Ο καταχωρητής που δεσμεύεται για να δείχνει σε στατικά δεδομένα.

Η Εικόνα 2.15 συνοψίζει τις πληροφορίες που διατηρούνται κατά την κλήση μιας διαδικασίας. Σημειώστε ότι υπάρχουν διάφορες μέθοδοι για τη διατήρηση της στοίβας. Η στοίβα επάνω από το δείκτη `$sp` διατηρείται απλώς με την εξασφάλιση ότι ο καλούμενος δε γράφει επάνω από τον `$sp`· ο ίδιος ο `$sp` διατηρείται από τον καλούμενο, που προσθέτει ακριβώς την ίδια ποσότητα που αφαιρέθηκε, και οι άλλοι καταχωρητές διατηρούνται με τοποθέτηση στη στοίβα (αν χρησιμοποιούνται) και εξαγωγή από εκεί. Αυτές οι ενέργειες εγγυώνται επίσης ότι ο καλών θα ξαναπάρει με μια φόρτωση (load) από τη στοίβα τα ίδια δεδομένα όπως αυτά που τοποθέτησε εκεί με τη φόρτωση (store), επειδή ο καλούμενος υπόσχεται να διατηρήσει τον `$sp` και επειδή ο καλούμενος επίσης υπόσχεται να μην τροποποιήσει το μέρος της στοίβας του καλούντος, δηλαδή την περιοχή επάνω από τον `$sp` τη στιγμή της κλήσης.

Διατηρούνται	Δε διατηρούνται
Αποθηκευμένοι (saved) καταχωρητές: `$s0-$s7`	Προσωρινοί (temporary) καταχωρητές: `$t0-$t9`
Καταχωρητής δείκτη στοίβας (stack pointer): `$sp`	Καταχωρητές ορίσματος: `$a0-$a3`
Καταχωρητής διεύθυνσης επιστροφής (return address): `$ra`	Καταχωρητές τιμής επιστροφής (return value): `$v0-$v1`
Στοίβα επάνω από το δείκτη στοίβας	Στοίβα κάτω από το δείκτη στοίβας

ΕΙΚΟΝΑ 2.15 Τι διατηρείται και τι όχι κατά την κλήση μιας διαδικασίας. Αν το λογισμικό βασίζεται στον καταχωρητή δείκτη πλαισίου (frame pointer register) ή στον καταχωρητή καθολικού δείκτη (global pointer register), που αναλύονται στις επόμενες ενότητες, αυτοί διατηρούνται επίσης.

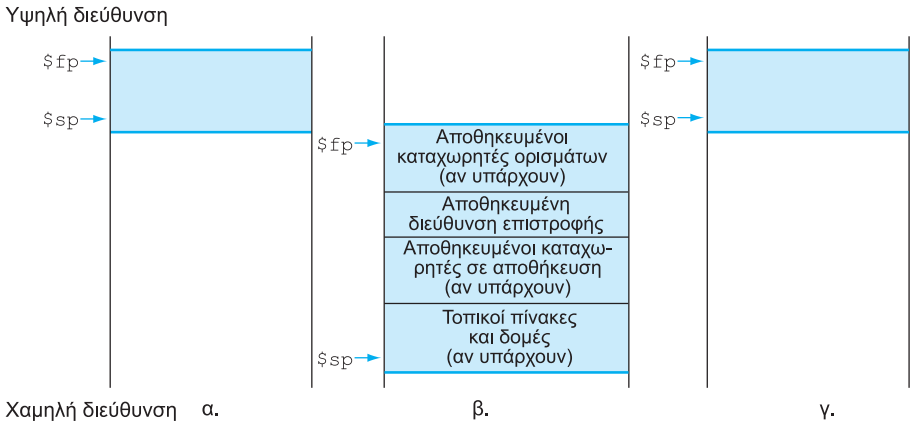
Κατανομή χώρου για νέα δεδομένα στη στοίβα (stack)

πλαίσιο διαδικασίας (procedure frame) Ονομάζεται επίσης **εγγραφή ενεργοποίησης** (activation record). Το τμήμα της στοίβας που περιέχει τους αποθηκευμένους καταχωρητές και τις τοπικές μεταβλητές μιας διαδικασίας.

δείκτης πλαισίου (frame pointer) Μια τιμή που δείχνει τη θέση των αποθηκευμένων καταχωρητών και των τοπικών μεταβλητών για μια δεδομένη διαδικασία.

Η τελευταία περιπλοκή είναι ότι η στοίβα χρησιμοποιείται επίσης για την αποθήκευση μεταβλητών που είναι τοπικές στη διαδικασία και που δε χωρούν σε καταχωρητές, όπως τοπικοί πίνακες (arrays) και δομές (structures). Το τμήμα της στοίβας που περιέχει τους αποθηκευμένους καταχωρητές μιας διαδικασίας και τις τοπικές μεταβλητές ονομάζεται **πλαίσιο διαδικασίας** (procedure frame) ή **εγγραφή ενεργοποίησης** (activation record). Η Εικόνα 2.16 δείχνει την κατάσταση της στοίβας πριν, κατά τη διάρκεια, και μετά την κλήση της διαδικασίας.

Κάποιο λογισμικό του MIPS χρησιμοποιεί ένα **δείκτη πλαισίου** (frame pointer — `$fp`) για να δείχνει στην πρώτη λέξη του πλαισίου μιας διαδικασίας. Ένας δείκτης στοίβας μπορεί να αλλάξει στη διάρκεια μιας διαδικασίας και, έτσι, οι αναφορές σε μια τοπική μεταβλητή στη μνήμη μπορεί να έχουν διαφορετικές σχετικές αποστάσεις (offsets) ανάλογα με το σημείο που βρίσκονται στη διαδικασία, πράγμα που κάνει τη διαδικασία πιο δυσνόητη. Εναλλακτικά, ένας δείκτης πλαισίου παρέχει ένα σταθερό καταχωρητή βάσης μέσα σε μια διαδικασία για τοπικές αναφορές στη μνήμη. Σημειώστε ότι μια εγγραφή ενεργοποίησης εμφανίζεται στη στοίβα ανεξάρτητα από τη ρητή χρήση ενός δείκτη πλαισίου. Αποφεύγαμε τον `$fp` αποφεύγοντας αλλαγές στον `$sp` μέσα σε μια διαδικασία: στα παραδείγματά μας, η στοίβα ρυθμίζεται μόνο στη είσοδο και την έξοδο από τη διαδικασία.



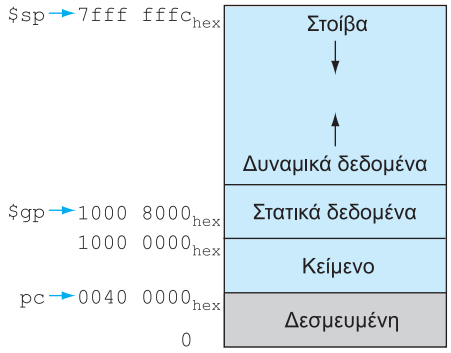
ΕΙΚΟΝΑ 2.16 Αναπαράσταση κατανομής στοίβας (α) πριν, (β) κατά τη διάρκεια, και (γ) μετά την κλήση διαδικασίας. Ο δείκτης πλαισίου (`$fp`) δείχνει στην πρώτη λέξη του πλαισίου, συχνά έναν αποθηκευμένο καταχωρητή ορίσματος, και ο δείκτης στοίβας (`$sp`) δείχνει στην κορυφή της στοίβας. Η στοίβα προσαρμόζεται ώστε να δημιουργήσει χώρο για όλους τους αποθηκευμένους καταχωρητές και όποιες τοπικές μεταβλητές βρίσκονται στη μνήμη. Εφόσον ο δείκτης στοίβας μπορεί να αλλάξει κατά τη διάρκεια της εκτέλεσης του προγράμματος, είναι ευκολότερο για τους προγραμματιστές να αναφέρονται στις μεταβλητές μέσω του σταθερού δείκτη πλαισίου, παρόλο που αυτό θα μπορούσε να γίνει απλώς με το δείκτη στοίβας και λίγες αριθμητικές πράξεις με τις διευθύνσεις. Αν δεν υπάρχουν τοπικές μεταβλητές στη στοίβα μέσα σε μια διαδικασία, ο μεταγλωττιστής θα κάνει οικονομία χρόνου *μην* ορίζοντας και *μην* επαναφέροντας το δείκτη πλαισίου. Όταν ένας δείκτης πλαισίου χρησιμοποιείται, παίρνει αρχικές τιμές με τη βοήθεια της διεύθυνσης στον `$sp` σε μια κλήση, και ο `$sp` επαναφέρεται με τη χρήση του `$fp`.

Κατανομή χώρου για νέα δεδομένα στο σωρό (heap)

Εκτός από τις αυτόματες μεταβλητές που είναι τοπικές στις διαδικασίες, οι προγραμματιστές της C χρειάζονται χώρο στη μνήμη για τις στατικές μεταβλητές και τις δυναμικές δομές δεδομένων. Η Εικόνα 2.17 δείχνει τη σύμβαση του MIPS για την κατανομή μνήμης. Η στοίβα ξεκινάει στο υψηλό άκρο της μνήμης και αυξάνεται προς τα κάτω. Το πρώτο μέρος του χαμηλού άκρου της μνήμης δεσμεύεται, ακολουθούμενο από το «καταφύγιο» του κώδικα μηχανής του MIPS, που παραδοσιακά ονομάζεται **τμήμα κειμένου** (text segment). Επάνω από τον κώδικα είναι το *τμήμα στατικών δεδομένων* (static data segment), η θέση για τις σταθερές (constants) και άλλες στατικές μεταβλητές. Παρόλο που οι πίνακες συνήθως έχουν σταθερό μήκος και, συνεπώς, ταιριάζουν καλά στο τμήμα στατικών δεδομένων, οι δομές δεδομένων όπως οι συνδεδεμένες λίστες συνήθως αυξομειώνονται στη διάρκεια της ζωής τους. Το τμήμα για τέτοιες δομές δεδομένων παραδοσιακά ονομάζεται *σωρός* (heap) και τοποθετείται αμέσως μετά στη μνήμη. Σημειώστε ότι αυτή η κατανομή δίνει τη δυνατότητα στη στοίβα και το σωρό να αυξάνουν σε μέγεθος το ένα προς το άλλο, επιτρέποντας έτσι την αποδοτική χρήση της μνήμης καθώς τα δύο τμήματα επεκτείνονται και συρρικνώνονται.

τμήμα κειμένου (text segment) Το τμήμα ενός αντικειμενικού αρχείου (object file) του Unix που περιέχει τον κώδικα γλώσσας μηχανής για τις ρουτίνες του πηγαίου αρχείου.

Η C κατανέμει και ελευθερώνει χώρο στο σωρό με ρητές (explicit) συναρτήσεις. Η συνάρτηση `malloc()` κατανέμει χώρο στο σωρό και επιστρέφει ένα δείκτη προς αυτόν, και η συνάρτηση `free()` ελευθερώνει χώρο στο σωρό στον οποίο δείχνει ο δείκτης. Η κατανομή μνήμης στη C ελέγχεται από τα προγράμματα, και είναι η πηγή πολλών συνηθισμένων και δύσκολων σφαλμάτων (bugs). Η παράλειψη της απελευθέρωσης χώρου μνήμης οδηγεί σε «διαρροή μνήμης» (memory leak) που τελικά χρησιμοποιεί τόση πολλή μνήμη ώστε το λειτουργικό σύστημα μπορεί να καταρρεύσει. Η απελευθέρωση χώρου πολύ νωρίς οδηγεί σε



ΕΙΚΟΝΑ 2.17 Η κατανομή μνήμης του MIPS για πρόγραμμα και δεδομένα. Αυτές οι διευθύνσεις είναι μόνο μια σύμβαση λογισμικού και όχι μέρος της αρχιτεκτονικής του MIPS. Από επάνω προς τα κάτω, ο δείκτης στοίβας παίρνει αρχική τιμή $7fff\ fffc_{hex}$ και αυξάνει προς τα κάτω προς το τμήμα δεδομένων. Στο άλλο άκρο, ο κώδικας του προγράμματος (το «κείμενο») αρχίζει στη διεύθυνση $0040\ 0000_{hex}$. Τα στατικά δεδομένα αρχίζουν στο $1000\ 0000_{hex}$. Ακολουθούν τα δυναμικά δεδομένα, που κατανέμονται με την εντολή `malloc` στη C και την εντολή `new` στην Java και αυξάνονται προς τη στοίβα σε μια περιοχή που λέγεται σωρός. Ο καθολικός δείκτης (global pointer — `$gp`) παίρνει ως τιμή μια διεύθυνση για να γίνει ευκολότερα η προσπέλαση δεδομένων. Παίρνει ως αρχική τιμή το $1000\ 8000_{hex}$ ώστε να έχει πρόσβαση στις διευθύνσεις από $1000\ 0000_{hex}$ έως $1000\ ffff_{hex}$ χρησιμοποιώντας θετική και αρνητική σχετική απόσταση 16 bit από τον `$gp` (δείτε τα σχετικά με τη διευθυνσιοδότηση συμπληρώματος ως προς δύο στο Κεφάλαιο 3).

«αιωρούμενους δείκτες» (dangling pointers) που μπορεί να έχουν ως αποτέλεσμα οι δείκτες να δείχνουν σε πράγματα στα οποία το πρόγραμμα δε σκόπευε να δείξει ποτέ.

Η Εικόνα 2.18 συνοψίζει τις συμβάσεις των καταχωρητών για τη συμβολική γλώσσα του MIPS. Οι Εικόνες 2.19 και 2.20 συνοψίζουν τα μέρη των συμβολικών εντολών MIPS που περιγράψαμε μέχρι εδώ, και τις αντίστοιχες εντολές μηχανής του MIPS.

Επιπλέον ανάπτυξη: Και αν υπάρχουν περισσότερες από τέσσερις παράμετροι; Η σύμβαση του MIPS είναι να τοποθετεί τις επιπλέον παραμέτρους στη στοίβα ακριβώς επάνω από το δείκτη πλαισίου. Η διαδικασία έπειτα αναμένει ότι οι τέσσερις πρώτες παράμετροι θα είναι στους καταχωρητές `$a0` έως `$a3` και οι υπόλοιπες στη μνήμη, προσπελάσιμες μέσω του δείκτη πλαισίου.

Όπως είπαμε στη λεζάντα της Εικόνας 2.16, ο δείκτης πλαισίου είναι βολικός επειδή όλες οι αναφορές στις μεταβλητές της στοίβας μέσα στη διαδικασία θα έχουν την ίδια σχετική απόσταση. Ωστόσο, ο δείκτης πλαισίου δεν είναι απαραίτητος. Ο μεταγλωττιστής της C του GNU για τον MIPS χρησιμοποιεί ένα δείκτη πλαισίου, αλλά ο μεταγλωττιστής της C της MIPS/Silicon Graphics όχι. Χρησιμοποιεί τον καταχωρητή 30 ως άλλον έναν αποθηκευμένο (saved) καταχωρητή (`$s8`).

Η εντολή `jal` στη πραγματικότητα αποθηκεύει τη διεύθυνση της εντολής που *ακολουθεί* την εντολή `jal` στον καταχωρητή `$ra`, επιτρέποντας έτσι η επιστροφή μιας διαδικασίας να γίνεται απλώς με την εντολή `jr $ra`.

Αυτοεξέταση

Ποιες από τις παρακάτω προτάσεις για τη C και την Java είναι γενικά αληθείς;

1. Οι κλήσεις διαδικασιών στη C είναι ταχύτερες από την κλήση μεθόδων στην Java.
2. Οι προγραμματιστές C χειρίζονται τα δεδομένα ρητά ενώ αυτό γίνεται αυτόματα στην Java.
3. Η C οδηγεί σε περισσότερα σφάλματα δεικτών και διαρροής μνήμης από ό,τι η Java.
4. Η C μεταβιβάζει παραμέτρους σε καταχωρητές ενώ η Java τις μεταβιβάζει στη στοίβα.

Όνομα	Αριθμός καταχωρητή	Χρήση	Διατηρείται κατά την κλήση;
`$zero`	0	η σταθερή τιμή 0	δ.ε.
`$v0-$v1`	2-3	τιμές αποτελεσμάτων και υπολογισμού εκφράσεων	όχι
`$a0-$a3`	4-7	ορίσματα	όχι
`$t0-$t7`	8-15	προσωρινοί	όχι
`$s0-$s7`	16-23	αποθηκευμένοι	ναι
`$t8-$t9`	24-25	περισσότεροι προσωρινοί	όχι
`$gp`	28	καθολικός δείκτης	ναι
`$sp`	29	δείκτης στοίβας	ναι
`$fp`	30	δείκτης πλαισίου	ναι
`$ra`	31	διεύθυνση επιστροφής	ναι

ΕΙΚΟΝΑ 2.18 Συμβάσεις των καταχωρητών του MIPS. Ο καταχωρητής 1, που ονομάζεται `$at`, είναι δεσμευμένος για το συμβολομεταφραστή (δείτε την Ενότητα 2.10), και οι καταχωρητές 26-27, που ονομάζονται `$k0-$k1`, είναι δεσμευμένοι για το λειτουργικό σύστημα (δ.ε. = δεν εφαρμόζεται).

Τελεστέοι του MIPS

Όνομα	Παράδειγμα	Σχόλια
32 καταχωρητές	`$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra`	Γρήγορες θέσεις για δεδομένα. Στον MIPS, τα δεδομένα πρέπει να βρίσκονται σε καταχωρητές για να εκτελεστούν αριθμητικές πράξεις. Ο καταχωρητής `$zero` του MIPS είναι πάντα ίσος με 0. Ο `$gp (28)` είναι ο καθολικός δείκτης, ο `$sp (29)` είναι ο δείκτης στοίβας, ο `$fp (30)` είναι ο δείκτης πλαισίου, και ο `$ra (31)` είναι η διεύθυνση επιστροφής.
2^{30} λέξεις μνήμης	Memory[0], Memory[4], . . ., Memory[4294967292]	Προσπελάζονται μόνον από εντολές μεταφοράς δεδομένων στο MIPS. Ο MIPS χρησιμοποιεί διευθύνσεις byte, και έτσι διαδοχικές διευθύνσεις λέξης διαφέρουν κατά 4. Η μνήμη κρατά δομές δεδομένων, πίνακες και «διασκορπισμένους» (spilled) καταχωρητές, όπως αυτά αποθηκεύονται κατά τις κλήσεις διαδικασιών.

Συμβολική γλώσσα του MIPS

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Αριθμητικές πράξεις	add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Τρεις τελεστέοι καταχωρητές
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2 - $s3`	Τρεις τελεστέοι καταχωρητές
Μεταφορά δεδομένων	load word	`lw $s1,100($s2)`	`$s1 = Memory[$s2+100]`	Δεδομένα από τη μνήμη σε καταχωρητή
	store word	`sw $s1,100($s2)`	`Memory[$s2+100] = $s1`	Δεδομένα από καταχωρητή στη μνήμη
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 καταχωρητή με σταθερά
	or immediate	`ori $s1,$s2,100`	`$s1 = $s2 | 100`	OR bit προς bit καταχωρητή 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 με συνθήκη	branch on equal	`beq $s1,$s2,L`	αν `($s1 == $s2)` πήγαινε στο L	Έλεγχος ισότητας και διακλάδωση
	branch on not equal	`bne $s1,$s2,L`	αν `($s1 != $s2)` πήγαινε στο L	Έλεγχος μη ισότητας και διακλάδωση
	set on less than	`slt $s1,$s2,$s3`	αν `($s2 < $s3)` τότε **$s1 = 1**· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από· χρήση με τις `beq, bne`
	set on less than immediate	`slti $s1,$s2,100`	αν `($s2 < 100)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Άμεση σύγκριση μικρότερο από· χρήση με τις `beq, bne`
Άλμα χωρίς συνθήκη	jump	`j L`	πήγαινε στο L	Άλμα στη διεύθυνση προορισμού
	jump register	`jr $ra`	πήγαινε στο `$ra`	Για επιστροφή διαδικασίας
	jump and link	`jal L`	`$ra = PC + 4`· πήγαινε στο L	Για κλήση διαδικασίας

ΕΙΚΟΝΑ 2.19 Αρχιτεκτονική του MIPS που έχουμε αποκαλύψει μέχρι την Ενότητα 2.7. Τα επισημασμένα σημεία δείχνουν τις δομές συμβολικής γλώσσας MIPS που παρουσιάστηκαν στην Ενότητα 2.7. Η μορφή J (J-format), που χρησιμοποιείται για τις εντολές jump και jump-and-link, εξηγείται στην Ενότητα 2.9.

Γλώσσα μηχανής του MIPS

Όνομα	Μορφή	Παράδειγμα						Σχόλια
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
and	R	0	18	19	17	0	36	and $s1,$s2,$s3
or	R	0	18	19	17	0	37	or $s1,$s2,$s3
nor	R	0	18	19	17	0	39	nor $s1,$s2,$s3
andi	I	12	18	17	100			andi $s1,$s2,100
ori	I	13	18	17	100			ori $s1,$s2,100
sll	R	0	0	18	17	10	0	sll $s1,$s2,10
srl	R	0	0	18	17	10	2	srl $s1,$s2,10
beq	I	4	17	18	25			beq $s1,$s2,100
bne	I	5	17	18	25			bne $s1,$s2,100
slt	R	0	18	19	17	0	42	slt $s1,$s2,$s3
j	J	2	2500					j 10000 (δείτε την Ενότητα 2.9)
jr	R	0	31	0	0	0	8	jr $ra
jal	J	3	2500					jal 10000 (δείτε την Ενότητα 2.9)
Μέγεθος πεδίου		6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	Όλες οι εντολές του MIPS 32 bit
Μορφή R	R	op	rs	rt	rd	shamt	funct	Μορφή αριθμητική εντολής
Μορφή I	I	op	rs	rt	address			Μορφή μεταφοράς δεδομένων και διακλάδωσης

ΕΙΚΟΝΑ 2.20 Γλώσσα μηχανής του MIPS που έχουμε αποκαλύψει μέχρι την Ενότητα 2.7. Τα επισημασμένα σημεία δείχνουν τις δομές συμβολικής γλώσσας MIPS που παρουσιάσαμε στην Ενότητα 2.7. Η μορφή J (J-format), που χρησιμοποιείται για τις εντολές jump και jump-and-link, εξηγείται στην Ενότητα 2.9. Η ενότητα αυτή εξηγεί επίσης γιατί η τοποθέτηση του 25 στο πεδίο διεύθυνσης των εντολών γλώσσας μηχανής `beq` και `bne` είναι ισοδύναμο με 100 στη συμβολική γλώσσα.

!(@ | = >
(wow open tab at bar is great)
Τέταρτος στίχος του ποιήματος πληκτρολογίου «Hatless Atlas,» 1991 (μερικοί δίνουν ονόματα στους χαρακτήρες ASCII: «!» είναι το «wow», «(» είναι «open», «|» είναι «bar», και ούτω καθεξής)

2.8 Η επικοινωνία με τους ανθρώπους

Οι υπολογιστές εφευρέθηκαν για να «καταβροχθίζουν» αριθμούς αλλά, μόλις έγιναν εμπορικά βιώσιμοι, χρησιμοποιήθηκαν για την επεξεργασία κειμένου. Οι περισσότεροι υπολογιστές σήμερα χρησιμοποιούν byte των 8 bit για να αναπαραστήσουν χαρακτήρες, ενώ ο Αμερικανικός Πρότυπος Κώδικας Ανταλλαγής Πληροφοριών (American Standard Code for Information Interchange — ASCII) είναι η αναπαράσταση που σχεδόν όλοι ακολουθούν. Η Εικόνα 2.21 παρουσιάζει τον κώδικα ASCII.

Μια σειρά εντολών μπορεί να εξαγάγει ένα byte από μια λέξη, οπότε οι load word και store word είναι αρκετές για τη μεταφορά τόσο byte όσο και λέξεων.

Τιμή ASCII	Χαρακτήρας	Τιμή ASCII	Χαρακτήρας	Τιμή ASCII	Χαρακτήρας	Τιμή ASCII	Χαρακτήρας	Τιμή ASCII	Χαρακτήρας	Τιμή ASCII	Χαρακτήρας
32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

ΕΙΚΟΝΑ 2.21 Η αναπαράσταση των χαρακτήρων κατά ASCII. Παρατηρήστε ότι τα κεφαλαία και τα πεζά γράμματα διαφέρουν ακριβώς κατά 32· αυτή η παρατήρηση μπορεί να οδηγήσει σε συντομεύσεις στον έλεγχο ή στην εναλλαγή κεφαλαίων και πεζών. Οι τιμές που δεν εμφανίζονται περιλαμβάνουν χαρακτήρες μορφοποίησης. Για παράδειγμα, το 8 αναπαριστά το backspace, το 9 αναπαριστά ένα χαρακτήρα στηλοθέτη (tab), και το 13 την επαναφορά κεφαλής (carriage return). Μια άλλη χρήσιμη τιμή είναι το 0 για τον κενό (ή μηδενικό) χαρακτήρα (null), την τιμή που η γλώσσα προγραμματισμού C χρησιμοποιεί για να σημειώσει το τέλος μιας συμβολοσειράς.

Επειδή όμως το κείμενο είναι δημοφιλές σε μερικά προγράμματα, ο MIPS παρέχει εντολές για μεταφορά byte. Η εντολή load byte (`lb`) φορτώνει ένα byte από τη μνήμη, και το τοποθετεί στα δεξιότερα 8 bit ενός καταχωρητή. Η εντολή store byte (`sb`) παίρνει ένα byte από τα δεξιότερα 8 bit ενός καταχωρητή και το γράφει στη μνήμη. Έτσι, με την επόμενη ακολουθία εντολών αντιγράφουμε ένα byte:

```
lb $t0,0($sp)          # Ανάγνωση byte από την προέλευση
sb $t0,0($gp)          # Εγγραφή byte στον προορισμό
```

Οι χαρακτήρες κανονικά συνδυάζονται σε συμβολοσειρές ή αλφαριθμητικά (strings) που έχουν μεταβλητό αριθμό χαρακτήρων. Υπάρχουν τρεις επιλογές για την αναπαράσταση μιας συμβολοσειράς: (1) η πρώτη θέση τής συμβολοσειράς είναι δεσμευμένη για να δίνει το μήκος της, (2) μια συνοδευτική μεταβλητή περιέχει το μήκος τής συμβολοσειράς (όπως σε μια δομή), ή (3) ένας χαρακτήρας, που χρησιμοποιείται για να σημειώσει το τέλος τής συμβολοσειράς, δείχνει στην τελευταία θέση τής συμβολοσειράς. Η C χρησιμοποιεί την τρίτη επιλογή, τερματίζοντας μια συμβολοσειρά με ένα byte που η τιμή του είναι 0 (ονομάζεται null στον κώδικα ASCII). Έτσι, η συμβολοσειρά «Cal» αναπαρίσταται στη C από τα παρακάτω 4 byte, σε δεκαδική σημειογραφία: 67, 97, 108, 0.

ΠΑΡΑΔΕΙΓΜΑ

Μεταγλώττιση μιας διαδικασίας αντιγραφής συμβολοσειράς, που δείχνει τον τρόπο χρήσης των συμβολοσειρών της C

Η διαδικασία `strcpy` αντιγράφει τη συμβολοσειρά `y` στη συμβολοσειρά `x`, χρησιμοποιώντας τη σύμβαση της C για τερματισμό με το μηδενικό (null) byte:

```
void strcpy (char x[], char y[])
{
    int i;
    i = 0;
    while ((x[i] = y[i]) != '\0') /* αντιγραφή και έλεγχος
                                     του byte */
     i += 1;
}
```

Ποιος είναι ο συμβολικός κώδικας του MIPS;

ΑΠΑΝΤΗΣΗ

Ακολουθεί το βασικό τμήμα συμβολικού κώδικα του MIPS. Υποθέστε ότι οι διευθύνσεις βάσης των πινάκων `x` και `y` βρίσκονται στους καταχωρητές `$a0` και `$a1`, ενώ το `i` βρίσκεται στον `$s0`. Η εντολή `strcpy` ρυθμίζει το δείκτη στοίβας και μετά αποθηκεύει τον αποθηκευμένο (saved) καταχωρητή `$s0` στη στοίβα:

```
strcpy:
    addi  $sp,$sp,-4     # ρύθμιση της στοίβας για 1 ακόμη
                         # αντικείμενο
    sw    $s0, 0($sp)    # αποθήκευση του $s0
```

Για να δώσουμε στο `i` αρχική τιμή 0, η επόμενη εντολή θέτει τον `$s0` ίσο με 0 προσθέτοντας 0 και 0 και βάζοντας το άθροισμα στον `$s0`:

```
    add   $s0,$zero,$zero   # i = 0 + 0
```

Αυτή είναι η αρχή του βρόχου. Η διεύθυνση του στοιχείου `y[i]` σχηματίζεται καταρχήν με την πρόσθεση του `i` στο `y[]`:

```
L1: add   $t1,$s0,$a1   # η διεύθυνση του y[i] στον $t1
```

Σημειώστε ότι δε χρειάζεται να πολλαπλασιάσουμε το `i` με 4, αφού το `y` είναι πίνακας *byte* και όχι λέξεων, όπως σε προηγούμενα παραδείγματα.

Για να φορτώσουμε το χαρακτήρα στο στοιχείο `y[i]`, χρησιμοποιούμε την εντολή load byte, που τοποθετεί το χαρακτήρα στον καταχωρητή `$t2`:

```
    lb    $t2,0($t1)    # $t2 = y[i]
```

Ένας αντίστοιχος υπολογισμός διεύθυνσης τοποθετεί τη διεύθυνση του `x[i]` στον `$t3`, και μετά ο χαρακτήρας που περιέχει ο `$t2` αποθηκεύεται στη διεύθυνση αυτή.

```
add   $t3,$s0,$a0   # η διεύθυνση του x[i] στον $t3
sb    $t2, 0($t3)   # x[i] = y[i]
```

Στη συνέχεια βγαίνουμε από το βρόχο αν ο χαρακτήρας ήταν 0· δηλαδή, αν είναι ο τελευταίος χαρακτήρας τής συμβολοσειράς:

```
beq   $t2,$zero,L2  # αν y[i] == 0, μετάβαση στην L2
```

Αν όχι, αυξάνουμε το `i` και ο βρόχος επαναλαμβάνεται:

```
addi  $s0, $s0,1    # i = i + 1
j     L1            # μετάβαση στην L1
```

Αν ο βρόχος δεν επαναληφθεί, ήταν ο τελευταίος χαρακτήρας τής συμβολοσειράς· επαναφέρουμε τον καταχωρητή `$s0` και το δείκτη στοίβας και επιστρέφουμε.

```
L2: lw    $s0, 0($sp)   # y[i] == 0: τέλος τής συμβολοσειράς
                        # επαναφορά του παλιού $s0
    addi  $sp,$sp,4     # εξαγωγή 1 λέξης από τη στοίβα
    jr    $ra           # επιστροφή
```

Οι αντιγραφές συμβολοσειρών στη C συνήθως χρησιμοποιούν δείκτες (pointers) αντί για πίνακες, ώστε να αποφύγουν τις πράξεις με το `i` στον παραπάνω κώδικα. Δείτε την Ενότητα 2.15 για μια εξήγηση των υπέρ και των κατά πινάκων και δεικτών.

Αφού η παραπάνω διαδικασία `strcpy` είναι μια διαδικασία-φύλλο, ο μεταγλωττιστής θα μπορούσε να κατανείμει το `i` σε έναν προσωρινό καταχωρητή και να αποφύγει την αποθήκευση και την επαναφορά του `$s0`. Έτσι, αντί να θεωρούμε τους καταχωρητές `$t` μόνον ως προσωρινούς, μπορούμε να τους θεωρούμε ως καταχωρητές που πρέπει να χρησιμοποιεί ο καλούμενος όποτε είναι βολικό. Όταν ένας μεταγλωττιστής βρίσκει μια διαδικασία-φύλλο, εξαντλεί όλους τους προσωρινούς καταχωρητές πριν χρησιμοποιήσει καταχωρητές που πρέπει να αποθηκεύσει.

Χαρακτήρες και συμβολοσειρές στην Java

Το πρότυπο *Unicode* είναι μια παγκόσμια κωδικοποίηση των αλφαβήτων των περισσότερων ανθρώπινων γλωσσών. Η Εικόνα 2.22 είναι μια λίστα των αλφαβήτων Unicode· υπάρχουν περίπου τόσα *αλφάβητα* Unicode όσα είναι τα χρήσιμα *σύμβολα* στον κώδικα ASCII. Για μεγαλύτερη κάλυψη, η Java χρησιμοποιεί την κωδικοποίηση Unicode για τους χαρακτήρες. Εξ ορισμού, χρησιμοποιεί 16 bit για να αναπαραστήσει ένα χαρακτήρα.

Λατινικό	Malayalam	Tagbanwa	Γενικά σύμβολα στίξης
Ελληνικό	Sinhala	Χμερ	Γράμματα τροποποίησης αποστάσεων
Κυριλλικό	Thai	Μογγολικό	Νομισματικά σύμβολα
Αρμενικό	Lao	Limbu	Συνδυασμός διακριτικών
Εβραϊκό	Θιβετιανό	Tai Le	Συνδυασμός σημείων για σύμβολα
Αραβικό	Myanmar	Kangxi Radicals	Εκθέτες και δείκτες
Συριακό	Γεωργιανό	Hiragana	Αριθμητικές μορφές
Thaana	Hangul Jamo	Katakana	Μαθηματικοί τελεστές
Devanagari	Αιθιοπικό	Bopomofo	Μαθηματικά αλφαριθμητικά σύμβολα
Bengali	Cherokee	Kanbun	Αλφάβητο Μπράϊγ
Gurmukhi	Ενιαίο Καναδικό Συλλαβικό Αβορίγινων	Shavian	Οπτική αναγνώριση χαρακτήρων
Gujarati	Ogham	Osmanya	Βυζαντινά μουσικά σύμβολα
Oriya	Ρουνικό	Κυπριακό συλλαβικό	Μουσικά σύμβολα
Tamil	Tagalog	Σύμβολα Tai Xuan Jing	Βέλη
Telugu	Hanunoo	Σύμβολα Yijing Hexagram	Σχεδίαση πλαισίων
Kannada	Buhid	Αιγαιακοί αριθμοί	Γεωμετρικά σχήματα

ΕΙΚΟΝΑ 2.22 Παραδείγματα αλφαβήτων Unicode. Η έκδοση 4.0 του Unicode έχει περισσότερα από 160 «μπλοκ», που είναι το όνομα για ένα σύνολο συμβόλων. Κάθε μπλοκ είναι πολλαπλάσιο του 16. Για παράδειγμα, τα Ελληνικά αρχίζουν στο 0370_{hex} και τα Κυριλλικά στο 0400_{hex}. Οι τρεις πρώτες στήλες παρουσιάζουν 48 μπλοκ που αντιστοιχούν σε ανθρώπινες γλώσσες σε κατά προσέγγιση αριθμητική σειρά Unicode. Η τελευταία στήλη έχει 16 μπλοκ που είναι πολυγλωσσικά (multilingual) και δεν είναι σε σειρά. Μια κωδικοποίηση 16 bit, που ονομάζεται UTF-16, είναι η προεπιλεγμένη. Μια κωδικοποίηση μεταβλητού μήκους που ονομάζεται UTF-8 κρατάει το υποσύνολο ASCII ως 8 bit και χρησιμοποιεί 16 έως 32 bit για τους υπόλοιπους χαρακτήρες. Η UTF-32 χρησιμοποιεί 32 bit ανά χαρακτήρα. Για περισσότερες πληροφορίες σχετικά με το θέμα, δείτε στη διεύθυνση *www.unicode.org*.

Το σύνολο εντολών του MIPS διαθέτει συγκεκριμένες εντολές για τη φόρτωση και την αποθήκευση τέτοιων ποσοτήτων 16 bit, που ονομάζονται *ημιλέξεις* (halfwords). Η εντολή load half (`lh`) φορτώνει μια ημιλέξη από τη μνήμη, τοποθετώντας τη στα δεξιότερα 16 bit ενός καταχωρητή. Η store half (`sh`) παίρνει μια ημιλέξη από τα δεξιότερα 16 bit ενός καταχωρητή και τη γράφει στη μνήμη. Αντιγράφουμε μια ημιλέξη με την ακολουθία εντολών

```
lh $t0,0($sp)    # ανάγνωση ημιλέξης (16 bit) από
                 # την προέλευση
sh $t0,0($gp)    # εγγραφή ημιλέξης (16 bit) στον προορισμό
```

Οι συμβολοσειρές είναι μια πρότυπη (standard) τάξη (κλάση) της Java με ειδική εγγενή υποστήριξη και προκαθορισμένες μεθόδους για τη συνένωση (concatenation), τη σύγκριση, και τη μετατροπή. Σε αντίθεση με τη C, η Java περιλαμβάνει μια λέξη που δίνει το μήκος τής συμβολοσειράς, παρόμοια με τους πίνακες της Java.

Επιπλέον ανάπτυξη: Το λογισμικό του MIPS προσπαθεί να κρατήσει τη στοίβα ευθυγραμμισμένη σε διευθύνσεις λέξεων, επιτρέποντας στο πρόγραμμα να χρησιμοποιεί πάντα τις εντολές `lw` και `sw` (που πρέπει να είναι ευθυγραμμισμένες) για την

προσπέλαση της στοίβας. Αυτή η σύμβαση σημαίνει ότι μια μεταβλητή τύπου `char` κατανεμημένη στη στοίβα καταλαμβάνει 4 byte, παρόλο που χρειάζεται λιγότερα. Παρόλα αυτά, μια μεταβλητή συμβολοσειράς της C ή ένας πίνακας byte *όντως* θα τοποθετήσει 4 byte ανά λέξη και μια μεταβλητή συμβολοσειράς της Java ή ένας πίνακας από μικρούς ακεραίους (shorts) θα τοποθετήσει 2 ημιλέξεις ανά λέξη.

Αυτοεξέταση

Ποιες από τις επόμενες προτάσεις σχετικά με χαρακτήρες και συμβολοσειρές στη C και την Java είναι αληθείς;

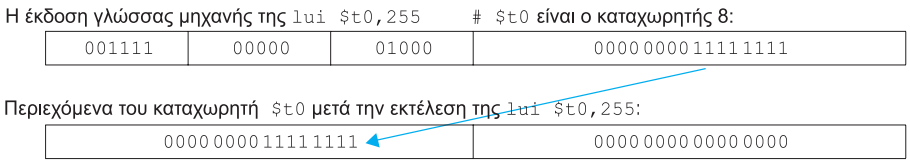
1. Μια συμβολοσειρά στη C καταλαμβάνει περίπου τη μισή μνήμη από ό,τι η ίδια συμβολοσειρά στην Java.
2. Οι συμβολοσειρές είναι απλώς ένα ανεπίσημο όνομα για μονοδιάστατους πίνακες χαρακτήρων στη C και την Java.
3. Οι συμβολοσειρές της C και της Java χρησιμοποιούν το χαρακτήρα null (0) για να σημειώσουν το τέλος μιας συμβολοσειράς.
4. Οι λειτουργίες σε συμβολοσειρές, όπως η εύρεση του μήκους, είναι ταχύτερες στη C από ό,τι στην Java.

2.9 Διευθυνσιοδότηση του MIPS για άμεσους τελεστέους και διευθύνσεις 32 bit

Παρόλο που η διατήρηση όλων των εντολών του MIPS σε μήκος 32 bit απλοποιεί το υλικό, υπάρχουν φορές που θα ήταν βολικό να έχουμε μια σταθερά 32 bit ή μια διεύθυνση 32 bit. Η ενότητα αυτή αρχίζει με τη γενική λύση για μεγάλες σταθερές και μετά δείχνει τις βελτιστοποιήσεις για διευθύνσεις εντολών που χρησιμοποιούνται στις διακλαδώσεις και τα άλματα.

Άμεσοι τελεστέοι των 32 bit

Αν και οι σταθερές είναι συχνά μικρές και χωρούν στο πεδίο 16 bit, μερικές φορές είναι μεγαλύτερες. Το σύνολο εντολών του MIPS περιλαμβάνει την εντολή *load upper immediate* (`lui`) ειδικά για να ορίζουμε τα ανώτερα 16 bit μιας σταθεράς σε έναν καταχωρητή, επιτρέποντας σε μια επόμενη εντολή να ορίσει τα κατώτερα 16 bit της σταθεράς. Η Εικόνα 2.23 παρουσιάζει τη λειτουργία τής εντολής `lui`.



ΕΙΚΟΝΑ 2.23 Το αποτέλεσμα της εντολής `lui`. Η εντολή `lui` μεταφέρει την τιμή του πεδίου άμεσης σταθεράς των 16 bit στα αριστερότερα 16 bit του καταχωρητή, συμπληρώνοντας τα κατώτερα 16 bit με μηδενικά.

Διασύνδεση υλικού και λογισμικού

Είτε ο μεταγλωττιστής (compiler) είτε ο συμβολομεταφραστής (assembler) πρέπει να διαιρέσει τις μεγάλες σταθερές σε τμήματα και μετά να τις επανασυναρμολογήσει σε έναν καταχωρητή. Όπως θα αναμένατε, ο περιορισμός μεγέθους του άμεσου πεδίου μπορεί να είναι ένα πρόβλημα για τις διευθύνσεις μνήμης στις φορτώσεις και τις αποθηκεύσεις όπως και για τις σταθερές στις άμεσες εντολές. Αν αυτή η δουλειά ανατεθεί στο συμβολομεταφραστή, όπως συμβαίνει στο λογισμικό του MIPS, ο συμβολομεταφραστής πρέπει να έχει διαθέσιμο έναν προσωρινό καταχωρητή στον οποίο να δημιουργεί τις μεγάλες τιμές. Αυτός είναι ο λόγος για τον καταχωρητή `$at`, ο οποίος είναι δεσμευμένος για το συμβολομεταφραστή.

Έτσι, η συμβολική αναπαράσταση της γλώσσας μηχανής του MIPS δεν περιορίζεται πια από το υλικό, αλλά από οτιδήποτε επιλέξει να συμπεριλάβει ο δημιουργός ενός συμβολομεταφραστή (δείτε την Ενότητα 2.10). Θα μείνουμε κοντά στο υλικό για να εξηγήσουμε την αρχιτεκτονική του υπολογιστή, επισημαίνοντας πότε χρησιμοποιούμε την εμπλουτισμένη γλώσσα του συμβολομεταφραστή που δεν υπάρχει στον επεξεργαστή.

Φόρτωση μιας σταθεράς 32 bit

ΠΑΡΑΔΕΙΓΜΑ

Ποιος είναι ο συμβολικός κώδικας MIPS για τη φόρτωση της επόμενης σταθεράς 32 bit στον καταχωρητή `$s0`;

```
0000 0000 0011 1101 0000 1001 0000 0000
```

ΑΠΑΝΤΗΣΗ

Πρώτα φορτώνουμε τα ανώτερα 16 bit, που είναι ίσα με 61 στο δεκαδικό σύστημα, χρησιμοποιώντας την εντολή `lui`:

```
lui $s0,61   # 61 δεκαδικό = 0000 0000 0011 1101 δυαδικό
```

Μετά από αυτό, η τιμή του καταχωρητή `$s0` είναι

```
0000 0000 0011 1101 0000 0000 0000 0000
```

Το επόμενο βήμα είναι να προσθέσουμε τα κατώτερα 16 bit, που η δεκαδική τους τιμή είναι 2304:

```
ori $s0, $s0, 2304   # 2304 δεκαδικό = 0000 1001 0000 0000
```

Η τελική τιμή στον καταχωρητή `$s0` είναι αυτή που θέλουμε:

```
0000 0000 0011 1101 0000 1001 0000 0000
```

Επιπλέον ανάπτυξη: Η δημιουργία σταθερών των 32 bit χρειάζεται προσοχή. Η εντολή `addi` αντιγράφει το αριστερότερο bit του άμεσου πεδίου 16 bit της εντολής στα ανώτερα 16 bit μιας λέξης. Η εντολή *logical or immediate* (`ori`) της Ενότητας 2.5 φορτώνει μηδενικά στα ανώτερα 16 bit και έτσι χρησιμοποιείται από το συμβολομεταφραστή σε συνδυασμό με τη `lui` για τη δημιουργία σταθερών των 32 bit.

Διευθυνσιοδότηση σε διακλαδώσεις και άλματα

Οι εντολές άλματος του MIPS έχουν την απλούστερη διευθυνσιοδότηση. Χρησιμοποιούν την τελευταία μορφή εντολής τού MIPS, που λέγεται *τύπος J* (J-type), που αποτελείται από 6 bit για το πεδίο λειτουργίας (`op`) και τα υπόλοιπα bit για το πεδίο διεύθυνσης. Έτσι, η εντολή

```
j 10000   # μετάβαση στη θέση 10000
```

θα μπορούσε να μεταφραστεί στην παρακάτω μορφή (στην πραγματικότητα, είναι λίγο πιο πολύπλοκη, όπως θα δούμε στην επόμενη σελίδα):

2	10000
6 bit	26 bit

όπου η τιμή του κωδικού λειτουργίας (opcode) της jump είναι 2 και η διεύθυνση άλματος είναι 10000.

Σε αντίθεση με την εντολή άλματος jump, η εντολή διακλάδωσης υπό συνθήκη πρέπει να καθορίσει δύο τελεστέους επιπλέον της διεύθυνσης διακλάδωσης. Έτσι, η εντολή

```
bne $s0,$s1,Exit   # μετάβαση στην Exit αν $s0 ≠ $s1
```

συμβολομεταφράζεται στην παρακάτω εντολή, αφήνοντας μόνο 16 bit για τη διεύθυνση διακλάδωσης:

5	16	17	Exit
6 bit	5 bit	5 bit	16 bit

Αν οι διευθύνσεις του προγράμματος έπρεπε να χωρέσουν σε αυτό το πεδίο 16 bit, θα σήμαινε ότι κανένα πρόγραμμα δε θα μπορούσε να είναι μεγαλύτερο από 2^{16}, μέγεθος υπερβολικά μικρό για να αποτελεί ρεαλιστική επιλογή σήμερα. Μια εναλλακτική επιλογή θα ήταν να καθοριστεί ένας καταχωρητής που θα προστίθεται πάντα στη διεύθυνση διακλάδωσης, ώστε η εντολή διακλάδωσης να υπολογίζει το εξής:

Μετρητής προγράμματος (program counter) =
Καταχωρητής + Διεύθυνση διακλάδωσης

Αυτό το άθροισμα επιτρέπει στο πρόγραμμα να έχει μέγεθος έως 2^{32}, ενώ είναι επίσης ικανό να χρησιμοποιήσει διακλαδώσεις υπό συνθήκη, λύνοντας το πρόβλημα του μεγέθους της διεύθυνσης διακλάδωσης. Η ερώτηση είναι, τότε, ποιος καταχωρητής;

Η απάντηση προκύπτει αν δούμε πώς χρησιμοποιούνται οι διακλαδώσεις υπό συνθήκη. Οι διακλαδώσεις υπό συνθήκη συναντώνται σε βρόχους και εντολές *if*, και έτσι συνήθως διακλαδώνονται σε μια κοντινή εντολή. Για παράδειγμα, περίπου οι μισές από όλες τις διακλαδώσεις υπό συνθήκη στα μετροπρογράμματα (benchmarks) SPEC2000 πηγαίνουν σε θέσεις που απέχουν λιγότερο από 16 εντολές. Αφού ο μετρητής προγράμματος (program counter — PC) περιέχει τη διεύθυνση της τρέχουσας εντολής, μπορούμε να διακλαδώσουμε μέσα

σε ένα εύρος $\pm 2^{15}$ λέξεων από την τρέχουσα εντολή αν χρησιμοποιήσουμε τον PC ως τον καταχωρητή που θα προστεθεί στη διεύθυνση. Σχεδόν όλοι οι βρόχοι και οι εντολές *if* είναι πολύ μικρότεροι από 2^{16} λέξεις, οπότε ο PC είναι η ιδανική επιλογή.

διευθυνσιοδότηση σχετική ως προς PC (PC-relative addressing) Ένας τρόπος διευθυνσιοδότησης στον οποίο η διεύθυνση είναι το άθροισμα του μετρητή προγράμματος (PC) και μιας σταθεράς στην εντολή.

Αυτή η μορφή διευθυνσιοδότησης διακλάδωσης ονομάζεται **διευθυνσιοδότηση σχετική ως προς PC** (PC-relative addressing). Όπως θα δούμε στο Κεφάλαιο 5, είναι βολικό για το υλικό να αυξάνει νωρίς το μετρητή προγράμματος (PC) ώστε να δείχνει στην επόμενη εντολή. Έτσι, η διεύθυνση του MIPS είναι στην πραγματικότητα σχετική ως προς τη διεύθυνση της επόμενης εντολής (PC + 4), σε αντιδιαστολή με την τρέχουσα εντολή (PC).

Όπως όλοι οι σύγχρονοι υπολογιστές, ο MIPS χρησιμοποιεί διευθυνσιοδότηση σχετική ως προς PC για όλες τις διακλαδώσεις υπό συνθήκη, επειδή ο προορισμός αυτών των εντολών είναι πιθανό να βρίσκεται κοντά στη διακλάδωση. Από την άλλη, οι εντολές άλματος και σύνδεσης (jump-and-link) καλούν διαδικασίες που δεν έχουν λόγο να βρίσκονται κοντά στην κλήση και, έτσι, φυσιολογικά χρησιμοποιούν άλλες μορφές διευθυνσιοδότησης. Έτσι, η αρχιτεκτονική MIPS προσφέρει μεγάλες διευθύνσεις για κλήσεις διαδικασιών χρησιμοποιώντας τη μορφή τύπου J τόσο για την εντολή jump όσο και για την εντολή jump-and-link.

Επειδή όλες οι εντολές τού MIPS έχουν μήκος 4 byte, ο MIPS επεκτείνει την απόσταση της διακλάδωσης κάνοντας τη σχετική ως προς PC διευθυνσιοδότηση να αναφέρεται στον αριθμό των *λέξεων* μέχρι την επόμενη εντολή αντί για τον αριθμό των byte. Έτσι, το πεδίο 16 bit μπορεί να διακλαδώσει τη ροή τέσσερις φορές πιο μακριά ερμηνεύοντας το πεδίο ως σχετική διεύθυνση λέξης αντί για σχετική διεύθυνση byte. Παρόμοια, το πεδίο 26 bit στις εντολές άλματος jump είναι επίσης διεύθυνση λέξης, που σημαίνει ότι αντιπροσωπεύει μια διεύθυνση byte των 28 bit.

Επιπλέον ανάπτυξη: Αφού ο μετρητής προγράμματος είναι 32 bit, 4 bit πρέπει να προέλθουν από κάπου αλλού. Η εντολή jump του MIPS αντικαθιστά μόνο τα 28 κατώτερα bit του μετρητή προγράμματος, αφήνοντας τα 4 υψηλότερα bit τού PC αμετάβλητα. Ο φορτωτής (loader) και το πρόγραμμα σύνδεσης (linker — Ενότητα 2.9) πρέπει να προσέχουν να αποφεύγουν την τοποθέτηση ενός προγράμματος σε ένα όριο διευθύνσεων των 256 MB (64 εκατομμύρια εντολές)· διαφορετικά, μια εντολή jump πρέπει να αντικατασταθεί από μια εντολή jump register της οποίας να προηγείται μια άλλη εντολή για τη φόρτωση της πλήρους διεύθυνσης των 32 bit σε έναν καταχωρητή.

Παρουσίαση της σχετικής απόστασης διακλάδωσης στη γλώσσα μηχανής

ΠΑΡΑΔΕΙΓΜΑ

Ο βρόχος *while* στη σελίδα 92 μεταγλωττίστηκε στον εξής συμβολικό κώδικα του MIPS:

```
Loop: sll   $t1,$s3,2  # προσωρινός καταχωρητής $t1 = 4 * i
    add $t1,$t1,$s6    # $t1 = διεύθυνση του save[i]
    lw  $t0,0($t1)     # προσωρινός καταχωρητής $t0 = save[i]
    bne $t0,$s5, Exit  # μετάβαση στην Exit αν save[i] ≠ k
    addi $s3,$s3,1     # i = i + 1
    j   Loop           # μετάβαση στη Loop
Exit:
```

Αν υποθέσουμε ότι τοποθετούμε το βρόχο να ξεκινάει στη θέση 80000 της μνήμης, ποιος είναι ο κώδικας μηχανής MIPS για το βρόχο αυτόν;

ΑΠΑΝΤΗΣΗ

Οι συμβολομεταφρασμένες εντολές και οι διευθύνσεις τους θα έμοιαζαν ως εξής:

80000	0	0	19	9	2	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024	…					

Θυμηθείτε ότι οι εντολές τού MIPS έχουν διευθύνσεις byte, οπότε οι διευθύνσεις διαδοχικών λέξεων διαφέρουν κατά 4, τον αριθμό των byte που περιέχονται σε μία λέξη. Η εντολή `bne` στην τέταρτη γραμμή προσθέτει 2 λέξεις ή 8 byte στη διεύθυνση της *επόμενης* εντολής (80016), καθορίζοντας τον προορισμό της διακλάδωσης ως προς την επόμενη εντολή (8 + 80016) αντί ως προς την εντολή διακλάδωσης (12 + 80012) ή χρησιμοποιώντας την πλήρη διεύθυνση προορισμού (80024). Η εντολή άλματος στην τελευταία γραμμή χρησιμοποιεί την πλήρη διεύθυνση (20000 × 4 = 80000), που αντιστοιχεί στην ετικέτα `Loop`.

Διασύνδεση υλικού και λογισμικού

Σχεδόν κάθε διακλάδωση υπό συνθήκη έχει προορισμό μια κοντινή θέση, αλλά σε μερικές περιπτώσεις διακλαδώνεται πολύ μακριά, μακρύτερα από ό,τι μπορεί να αναπαρασταθεί με τα 16 bit της εντολής διακλάδωσης υπό συνθήκη. Ο συμβολομεταφραστής σπεύδει να σώσει την κατάσταση όπως ακριβώς και με τις μεγάλες διευθύνσεις και σταθερές: προσθέτει ένα άλμα χωρίς συνθήκη προς τον προορισμό της διακλάδωσης, και αντιστρέφει τη συνθήκη ώστε η διακλάδωση να αποφασίσει αν θα παραλείψει το άλμα.

Διακλάδωση πολύ μακριά

ΠΑΡΑΔΕΙΓΜΑ

Με δεδομένη μια διακλάδωση αν ο καταχωρητής `$s0` είναι ίσος με τον καταχωρητή `$s1`,

```
beq $s0,$s1, L1
```

αντικαταστήστε τη με ένα ζεύγος εντολών που παρέχει μια πολύ μεγαλύτερη απόσταση διακλάδωσης.

ΑΠΑΝΤΗΣΗ

Οι επόμενες εντολές αντικαθιστούν την κοντινή διακλάδωση υπό συνθήκη:

```
    bne   $s0,$s1, L2
    j     L1
L2:
```

Σύνοψη τρόπων διευθυνσιοδότησης του MIPS

τρόπος διευθυνσιοδότησης (addressing mode) Ένας από πολλούς κανόνες ορισμού διευθύνσεων που καθορίζονται από τη διαφορετική χρήση των τελεστέων και των διευθύνσεων.

Οι διάφορες μορφές διευθυνσιοδότησης γενικά ονομάζονται **τρόποι διευθυνσιοδότησης** (addressing modes). Οι τρόποι διευθυνσιοδότησης του MIPS είναι οι εξής:

1. **Διευθυνσιοδότηση μέσω καταχωρητή** (register addressing), όπου ο τελεστέος είναι ένας καταχωρητής.
2. **Διευθυνσιοδότηση βάσης** ή **μετατόπισης** (base ή displacement addressing), όπου ο τελεστέος βρίσκεται σε μια θέση μνήμης της οποίας η διεύθυνση είναι το άθροισμα ενός καταχωρητή και μιας σταθεράς μέσα στην εντολή.
3. **Άμεση διευθυνσιοδότηση** (immediate addressing), όπου ο τελεστέος είναι μια σταθερά μέσα στην ίδια την εντολή.
4. **Σχετική ως προς PC διευθυνσιοδότηση** (PC-relative addressing), όπου η διεύθυνση είναι το άθροισμα του μετρητή προγράμματος και μιας σταθεράς μέσα στην εντολή.
5. **Ψευδο-απευθείας διευθυνσιοδότηση** (pseudodirect addressing), όπου η διεύθυνση άλματος είναι τα 26 bit της εντολής συνενωμένα με τα ανώτερα bit του PC.

Διασύνδεση υλικού και λογισμικού

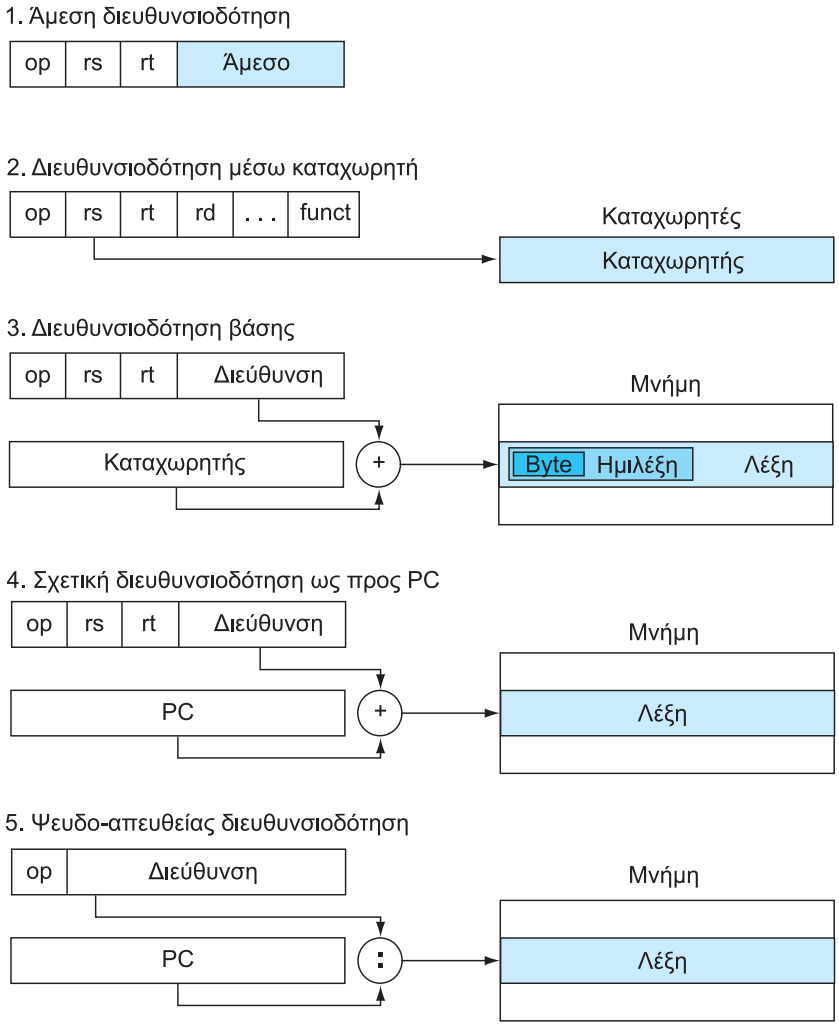
Αν και παρουσιάζουμε την αρχιτεκτονική του MIPS να έχει διευθύνσεις 32 bit, σχεδόν όλοι οι μικροεπεξεργαστές (μεταξύ των οποίων και ο MIPS) έχουν επεκτάσεις διευθύνσεων των 64 bit (δείτε το Παράρτημα Δ). Αυτές οι επεκτάσεις έγιναν σε ανταπόκριση στις ανάγκες του λογισμικού για μεγαλύτερα προγράμματα. Η διαδικασία της επέκτασης του συνόλου εντολών επιτρέπει στις αρχιτεκτονικές να επεκταθούν με έναν τρόπο που αφήνει το λογισμικό να μεταφέρεται στην επόμενη γενιά της αρχιτεκτονικής διατηρώντας τη συμβατότητα με τις προηγούμενες.

Σημειώστε ότι μια πράξη (λειτουργία) μπορεί να χρησιμοποιεί περισσότερους από έναν τρόπους διευθυνσιοδότησης. Η πρόσθεση, για παράδειγμα, χρησιμοποιεί τόσο την άμεση διευθυνσιοδότηση (`addi`) όσο και τη διευθυνσιοδότηση μέσω καταχωρητή (`add`). Η Εικόνα 2.24 δείχνει με ποιο τρόπο προσδιορίζονται οι τελεστέοι για κάθε τρόπο διευθυνσιοδότησης. Η ενότητα Σε μεγαλύτερο βάθος δείχνει άλλους τρόπους διευθυνσιοδότησης που συναντώνται στον επεξεργαστή IBM PowerPC.

Αποκωδικοποίηση της γλώσσας μηχανής

Μερικές φορές, αναγκάζεστε να κάνετε αντίστροφη ανάλυση της γλώσσας μηχανής για να δημιουργήσετε την αρχική συμβολική γλώσσα. Ένα παράδειγμα

είναι όταν βλέπουμε ένα αντίγραφο πυρήνα (core dump). Η Εικόνα 2.25 δείχνει την κωδικοποίηση του MIPS των πεδίων για τη γλώσσα μηχανής MIPS. Αυτή η εικόνα βοηθάει όταν μεταφράζουμε μόνοι μας από τη συμβολική γλώσσα σε γλώσσα μηχανής και αντίστροφα.



ΕΙΚΟΝΑ 2.24 Αναπαράσταση των πέντε τρόπων διευθυνσιοδότησης του MIPS. Οι τελεστέοι είναι σκιασμένοι με χρώμα. Ο τελεστέος του τρόπου 3 είναι στη μνήμη, ενώ ο τελεστέος του τρόπου 2 είναι ένας καταχωρητής. Σημειώστε ότι διάφορες εκδόσεις των εντολών load και store προσπελάζουν byte, ημιλέξεις, ή λέξεις. Για τον τρόπο 1, ο τελεστέος είναι 16 bit της ίδιας της εντολής. Οι τρόποι 4 και 5 διευθυνσιοδοτούν εντολές στη μνήμη, όπου ο τρόπος 4 προσθέτει μια διεύθυνση 16 bit που έχει υποστεί αριστερή ολίσθηση κατά 2 bit στο μετρητή προγράμματος, και ο τρόπος 5 συνενώνει μια διεύθυνση 26 bit, που έχει υποστεί αριστερή ολίσθηση κατά 2 bit, με τα 4 υψηλότερα bit του PC.

ΠΑΡΑΔΕΙΓΜΑ

ΑΠΑΝΤΗΣΗ

Αποκωδικοποίηση κώδικα μηχανής

Ποια είναι η εντολή συμβολικής γλώσσας που αντιστοιχεί στην επόμενη εντολή μηχανής;

```
00af8020hex
```

Το πρώτο βήμα για τη μετατροπή του δεκαεξαδικού σε δυαδικό είναι να βρούμε τα πεδία op:

```
(Bit      3128  26                              5   2   0)
          0000 0000 1010 1111 1000 0000 0010 0000
```

Κοιτάζουμε στο πεδίο op για να προσδιορίσουμε την πράξη (λειτουργία). Με τη βοήθεια της Εικόνας 2.25, όταν τα bit 31-29 είναι 000 και τα bit 28-26 είναι 000, πρόκειται για μια εντολή μορφής R. Ας μετασχηματίσουμε τη δυαδική εντολή σε πεδία της μορφής R, που παρατίθενται στην Εικόνα 2.26:

op	rs	rt	rd	shamt	funct
000000	00101	01111	10000	00000	100000

Το κάτω τμήμα της Εικόνας 2.25 προσδιορίζει τη λειτουργία μιας εντολής μορφής R. Στην περίπτωση αυτή, τα bit 5-3 είναι 100 και τα bit 2-0 είναι 000, που σημαίνει ότι αυτή η δυαδική σειρά αναπαριστά μια εντολή πρόσθεσης `add`.

Αποκωδικοποιούμε το υπόλοιπο της εντολής κοιτάζοντας τις τιμές των πεδίων. Οι δεκαδικές τιμές είναι 5 για το πεδίο rs, 15 για το rt, 16 για το rd (το shamt δε χρησιμοποιείται). Η Εικόνα 2.18 λέει ότι αυτοί οι αριθμοί αναπαριστούν τους καταχωρητές `$a1`, `$t7`, και `$s0`. Τώρα μπορούμε να δείξουμε τη συμβολική εντολή:

```
add $s0,$a1,$t7
```

Η Εικόνα 2.26 δείχνει όλες τις μορφές εντολών του MIPS. Η Εικόνα 2.27 δείχνει τη συμβολική γλώσσα που αποκαλύψαμε στο Κεφάλαιο 2· το υπόλοιπο κρυμμένο τμήμα των εντολών MIPS ασχολείται κυρίως με αριθμητικές πράξεις και καλύπτεται στο επόμενο κεφάλαιο.

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)	μορφή R	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
1(001)	add immediate	addiu	set less than imm.	sltiu	andi	ori	xori	load upper imm.
2(010)	TLB	FlPt						
3(011)								
4(100)	load byte	load half	lwl	load word	lbu	lhu	lwr	
5(101)	store byte	store half	swl	store word			swr	
6(110)	lwc0	lwc1						
7(111)	swc0	swc1						

op(31:26)=010000(TLB), rs(25:21)								
23-21 25-24	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	mfc0		cfc0		mtc0		ctc0	
1(001)								
2(010)								
3(011)								

op(31:26)=000000 (μορφή R), 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)	shift left logical		shift right logical	sra	sllv		srlv	srav
1(001)	jump reg.	jalr			syscall	break		
2(010)	mfhi	mthi	mflo	mtlo				
3(011)	mult	multu	div	divu				
4(100)	add	addu	subtract	subu	and	or	xor	not ή (nor)
5(101)			set l.t.	sltu				
6(110)								
7(111)								

ΕΙΚΟΝΑ 2.25 Κωδικοποίηση εντολών MIPS. Αυτός ο συμβολισμός δίνει την τιμή ενός πεδίου ανά γραμμή και στήλη. Για παράδειγμα, το επάνω μέρος της εικόνας δείχνει την εντολή load word στη γραμμή αριθμός 4 (100_{two} για τα bit 31-29 της εντολής) και στη στήλη αριθμός 3 (011_{two} για τα bit 28-26 της εντολής), και έτσι η αντίστοιχη τιμή του πεδίου op (bit 31-26) είναι 100011_{two}. Η υπογράμμιση σημαίνει ότι το πεδίο χρησιμοποιείται κάπου αλλού. Για παράδειγμα, η μορφή R στη γραμμή 0 και στη στήλη 0 (op = 000000_{two}) ορίζεται στο κάτω μέρος της εικόνας. Έτσι, η εντολή αφαίρεσης subtract στη γραμμή 4 στήλη 2 στο κάτω τμήμα σημαίνει ότι το πεδίο funct (bit 5-0) της εντολής είναι 100010_{two} και το πεδίο op (bit 31-26) είναι 000000_{two}. Η τιμή FlPt στη γραμμή 2 στήλη 1 ορίζεται στην Εικόνα 3.20 στο Κεφάλαιο 3. Bltz/gez είναι ο κωδικός λειτουργίας τεσσάρων εντολών που βρίσκονται στο Παράρτημα Α: bltz, bgez, bltal, και bgezal. Το Κεφάλαιο 2 περιγράφει εντολές που δίνονται με το πλήρες όνομά τους και με χρώμα, ενώ το Κεφάλαιο 3 περιγράφει εντολές που δίνονται με μνημονικά ονόματα και με χρώμα. Το Παράρτημα Α καλύπτει όλες τις εντολές.

Όνομα	Παράδειγμα						Σχόλια
Μέγεθος πεδίου	6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	Όλες οι εντολές του MIPS 32 bit
Μορφή R	op	rs	Rt	rd	shamt	funct	Μορφή αριθμητικής εντολής
Μορφή I	op	rs	rt	address/immediate			Μορφή μεταφοράς, διακλάδωσης, και άμεση
Μορφή J	op	διεύθυνση προορισμού					Μορφή εντολής άλματος

ΕΙΚΟΝΑ 2.26 Μορφές εντολών τού MIPS στο Κεφάλαιο 2. Τα επισημασμένα σημεία δείχνουν μορφές εντολών που παρουσιάστηκαν στην ενότητα αυτή.

Αυτοεξέταση

Ποιο είναι το εύρος διευθύνσεων για διακλαδώσεις υπό συνθήκη στο MIPS (K = 1024);

1. Διευθύνσεις μεταξύ 0 και 64K – 1
2. Διευθύνσεις μεταξύ 0 και 256K – 1
3. Διευθύνσεις μέχρι περίπου 32K πριν από τη διακλάδωση έως 32K μετά
4. Διευθύνσεις μέχρι περίπου 128K πριν από τη διακλάδωση έως 128K μετά

Ποιο είναι το εύρος διευθύνσεων για τις εντολές jump και jump and link στο MIPS (M = 1024K);

1. Διευθύνσεις μεταξύ 0 και 64M – 1
2. Διευθύνσεις μεταξύ 0 και 256M – 1
3. Διευθύνσεις μέχρι περίπου 32M πριν από τη διακλάδωση έως 32M μετά
4. Διευθύνσεις μέχρι περίπου 128M πριν από τη διακλάδωση έως 128M μετά
5. Οπουδήποτε μέσα σε ένα μπλοκ 64M διευθύνσεων, όπου ο μετρητής προγράμματος παρέχει τα υψηλότερα 6 bit
6. Οπουδήποτε μέσα σε ένα μπλοκ 256M διευθύνσεων, όπου ο μετρητής προγράμματος παρέχει τα 4 υψηλότερα bit

Ποια είναι η εντολή συμβολικής γλώσσας τού MIPS που αντιστοιχεί στην εντολή μηχανής με την τιμή $0000\ 0000_{hex}$;

1. `j`
2. `μορφή R`
3. `addi`
4. `sll`
5. `mfc0`
6. Μη ορισμένος κωδικός λειτουργίας (opcode): δεν υπάρχει έγκυρη εντολή που να αντιστοιχεί στο 0.

Τελεστέοι του MIPS

Όνομα	Παράδειγμα	Σχόλια
32 καταχωρητές	`$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra, $at`	Γρήγορες θέσεις για δεδομένα. Στο MIPS, τα δεδομένα πρέπει να βρίσκονται σε καταχωρητές για να εκτελεστούν αριθμητικές πράξεις. Ο καταχωρητής `$zero` του MIPS είναι πάντα ίσος με 0. Ο καταχωρητής `$at` δεσμεύεται για το συμβολομεταφραστή για να χειριστεί τις μεγάλες σταθερές.
2^{30} λέξεις μνήμης	Memory[0], Memory[4], . . ., Memory[4294967292]	Προσπελάζονται μόνον από εντολές μεταφοράς δεδομένων στο MIPS. Ο MIPS χρησιμοποιεί διευθύνσεις byte, οπότε διαδοχικές διευθύνσεις λέξεων διαφέρουν κατά 4. Η μνήμη κρατά δομές δεδομένων, πίνακες, και «διασκορπισμένους» (spilled) καταχωρητές, όπως αυτά που αποθηκεύονται κατά τις κλήσεις διαδικασιών.

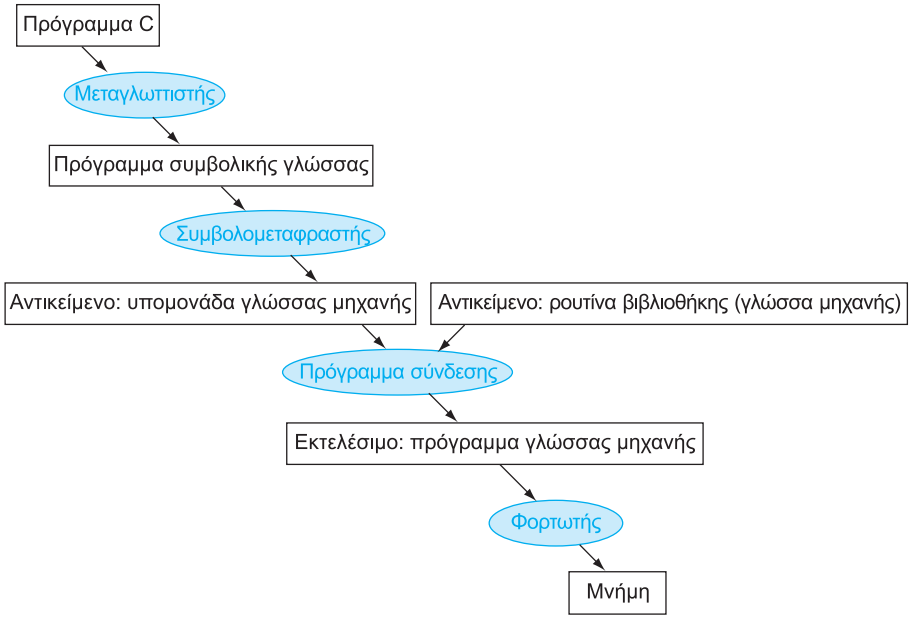
Συμβολική γλώσσα του MIPS

Κατηγορία	Εντολή	Παράδειγμα	Σημασία	Σχόλια
Αριθμητικές πράξεις	add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Τρεις τελεστέοι καταχωρητές
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2 - $s3`	Τρεις τελεστέοι καταχωρητές
	add immediate	`addi $s1,$s2,100`	`$s1 = $s2 + 100`	Χρησιμοποιείται για την πρόσθεση σταθερών
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	`lh $s1,100($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	`lb $s1,100($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 immed.	`lui $s1,100`	`$s1 = 100 * 216`	Φορτώνει σταθερά στα ανώτερα 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 καταχωρητή με σταθερά
	or immediate	`ori $s1,$s2,100`	`$s1 = $s2 | 100`	OR bit προς bit καταχωρητή 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 με συνθήκη	branch on equal	`beq $s1,$s2,25`	αν `($s1 == $s2)` πήγαινε στο PC + 4 + 100	Έλεγχος ισότητας· σχετική ως προς PC διακλάδωση
	branch on not equal	`bne $s1,$s2,25`	αν `($s1 != $s2)` πήγαινε στο PC + 4 + 100	Έλεγχος μη ισότητας· σχετική ως προς PC διακλάδωση
	set on less than	`slt $s1,$s2,$s3`	αν `($s2 < $s3)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από· χρήση με τις `beq, bne`
	set on less than immediate	`slti $s1,$s2,100`	αν `($s2 < 100)` τότε `$s1 = 1`· αλλιώς `$s1 = 0`	Σύγκριση μικρότερο από σταθερά
Άλμα χωρίς συνθήκη	jump	`j 2500`	μετάβαση στο 10000	Άλμα στη διεύθυνση προορισμού
	jump register	`jr $ra`	μετάβαση στο `$ra`	Για εναλλαγή και επιστροφή διαδικασίας
	jump and link	`jal 2500`	`$ra` = PC + 4· μετάβαση στο 10000	Για κλήση διαδικασίας

ΕΙΚΟΝΑ 2.27 Αρχιτεκτονική του MIPS που έχουμε αποκαλύψει στο Κεφάλαιο 2. Τα επισημασμένα σημεία δείχνουν μέρη από τις Ενότητες 2.8 και 2.9.

2.10 Μετάφραση και εκκίνηση προγράμματος

Αυτή η ενότητα περιγράφει τα τέσσερα βήματα για το μετασχηματισμό ενός προγράμματος C, που βρίσκεται σε ένα αρχείο στο δίσκο, σε πρόγραμμα που εκτελείται σε έναν υπολογιστή. Η Εικόνα 2.28 παρουσιάζει την ιεραρχία της μετάφρασης. Μερικά συστήματα συνδυάζουν αυτά τα βήματα για να μειώσουν το χρόνο της μετάφρασης, αλλά αυτές είναι οι τέσσερις λογικές φάσεις από τις οποίες πρέπει να περάσουν τα προγράμματα. Αυτή η ενότητα ακολουθεί τη συγκεκριμένη ιεραρχία μετάφρασης.



ΕΙΚΟΝΑ 2.28 Ιεραρχία μετάφρασης για τη γλώσσα C. Μια γλώσσα προγραμματισμού υψηλού επιπέδου μεταγλωττίζεται πρώτα σε ένα πρόγραμμα συμβολικής γλώσσας, και στη συνέχεια συμβολομεταφράζεται (assembled) σε μια αντικειμενική υπομονάδα (object module) γλώσσας μηχανής. Το πρόγραμμα σύνδεσης (linker) συνδυάζει πολλές υπομονάδες με ρουτίνες βιβλιοθήκης για τον προσδιορισμό όλων των αναφορών. Ο φορτωτής (loader), κατόπιν, τοποθετεί τον κώδικα μηχανής σε κατάλληλες θέσεις της μνήμης ώστε να εκτελεστούν από τον επεξεργαστή. Για να επιταχυνθεί η διαδικασία της μετάφρασης, κάποια βήματα παραλείπονται ή συγχωνεύονται. Μερικοί μεταγλωττιστές παράγουν αντικειμενικές υπομονάδες απευθείας, και μερικά συστήματα χρησιμοποιούν φορτωτές σύνδεσης που υλοποιούν τα δύο τελευταία βήματα. Για την αναγνώριση του τύπου του αρχείου, το UNIX ακολουθεί έναν κανόνα κατάληξης (προέκτασης) για τα αρχεία: τα πηγαία αρχεία της C ονομάζονται `x.c`, τα αρχεία συμβολικής γλώσσας `x.s`, τα αντικειμενικά αρχεία ονομάζονται `x.o`, οι στατικά συνδεδεμένες ρουτίνες βιβλιοθήκης `x.a`, οι δυναμικά συνδεδεμένες ρουτίνες βιβλιοθήκης `x.so`, και τα εκτελέσιμα αρχεία εξ ορισμού αποκαλούνται `a.out`. Το MS-DOS χρησιμοποιεί τις προεκτάσεις `.C`, `.ASM`, `.OBJ`, `.LIB`, `.DLL`, και `.EXE` για τον ίδιο σκοπό.

Μεταγλωττιστής

Ο μεταγλωττιστής (compiler) μετασχηματίζει ένα πρόγραμμα C σε πρόγραμμα συμβολικής γλώσσας (assembly language), μια συμβολική μορφή αυτού το οποίο καταλαβαίνει η μηχανή. Τα προγράμματα γλωσσών υψηλού επιπέδου απαιτούν πολύ λιγότερες γραμμές κώδικα από τη συμβολική γλώσσα και, έτσι, η παραγωγικότητα του προγραμματιστή είναι πολύ υψηλότερη.

Το 1975, πολλά λειτουργικά συστήματα (operating systems) και συμβολομεταφραστές (assemblers) γράφονταν σε **συμβολική γλώσσα** (assembly language) επειδή οι μνήμες ήταν μικρές και οι μεταγλωττιστές χαμηλής απόδοσης. Η κατά 128.000 φορές αύξηση της χωρητικότητας της μνήμης σε κάθε ολοκληρωμένο κύκλωμα DRAM μείωσε τις ανησυχίες σχετικά με το μέγεθος των προγραμμάτων, και οι μεταγλωττιστές βελτιστοποίησης (optimizing compilers) σήμερα μπορούν να παράγουν προγράμματα συμβολικής γλώσσας σχεδόν τόσο καλά όσο και ένας ειδικός της συμβολικής γλώσσας, και μερικές φορές ακόμη καλύτερα για μεγάλα προγράμματα.

συμβολική γλώσσα (assembly language) Μια γλώσσα συμβόλων που μπορεί να μεταφραστεί σε δυαδική μορφή.

Συμβολομεταφραστής

Όπως είπαμε στη σελίδα 114, αφού η συμβολική γλώσσα είναι η διασύνδεση με το λογισμικό υψηλού επιπέδου, ο συμβολομεταφραστής μπορεί επίσης να χειριστεί συνήθεις παραλλαγές εντολών γλώσσας μηχανής σαν να ήταν και αυτές εντολές. Το υλικό δε χρειάζεται να υλοποιεί αυτές τις εντολές· ωστόσο, η παρουσία τους στη συμβολική γλώσσα απλοποιεί τη μετάφραση και τον προγραμματισμό. Τέτοιες εντολές ονομάζονται **ψευδοεντολές** (pseudoinstructions).

ψευδοεντολή (pseudoinstruction) Μια συνήθης παραλλαγή των εντολών της συμβολικής γλώσσας που συχνά αντιμετωπίζεται σαν να ήταν και αυτή εντολή.

Όπως είπαμε και προηγουμένως, το υλικό του MIPS εξασφαλίζει ότι ο καταχωρητής `$zero` θα έχει πάντα την τιμή 0. Αυτό σημαίνει ότι, οποτεδήποτε χρησιμοποιείται, ο καταχωρητής `$zero` δίνει την τιμή 0, και ο προγραμματιστής δεν μπορεί να αλλάξει την τιμή του. Ο καταχωρητής `$zero` χρησιμοποιείται για να δημιουργήσει την εντολή της συμβολικής γλώσσας `move` (μετακίνηση) η οποία αντιγράφει τα περιεχόμενα ενός καταχωρητή σε έναν άλλο. Κατά συνέπεια, ο συμβολομεταφραστής τού MIPS δέχεται αυτή την εντολή παρά το γεγονός ότι δεν υπάρχει στην αρχιτεκτονική τού MIPS:

```
move $t0,$t1        # ο καταχωρητής $t0 παίρνει
                    # τον καταχωρητή $t1
```

Ο συμβολομεταφραστής μετατρέπει αυτή την εντολή συμβολικής γλώσσας σε γλώσσα μηχανής ισοδύναμη με την παρακάτω εντολή:

```
add $t0,$zero,$t1   # ο καταχωρητής $t0 παίρνει
                    # 0 + καταχωρητή $t1
```

Ο συμβολομεταφραστής τού MIPS μετατρέπει επίσης την εντολή `blt` (branch on less than — διακλάδωση αν μικρότερο από) στις δύο εντολές `slt` και `bne` που αναφέρονται στην ενότητα Διασύνδεση υλικού λογισμικού της σελίδας 94. Άλλα παραδείγματα περιλαμβάνουν τις εντολές `bgt`, `bge`, και `ble`. Επίσης, μετατρέπει διακλαδώσεις σε μακρινές θέσεις σε μια διακλάδωση και ένα άλμα (jump). Όπως αναφέραμε πιο πάνω, ο συμβολομεταφραστής τού MIPS επιτρέπει σε σταθερές 32 bit να φορτώνονται σε έναν καταχωρητή παρά το όριο 16 bit των άμεσων εντολών.

Για να συνοψίσουμε, οι ψευδοεντολές δίνουν στο MIPS ένα πλουσιότερο σύνολο εντολών συμβολικής γλώσσας από αυτές που υλοποιούνται από το υλικό. Το μόνο κόστος είναι η δέσμευση ενός καταχωρητή, του `$at`, για χρήση από το συμβολομεταφραστή. Αν πρόκειται να γράψετε προγράμματα σε συμβολική

γλώσσα, η χρήση των ψευδοεντολών θα απλοποιήσει το έργο σας. Ωστόσο, για να καταλάβετε την αρχιτεκτονική του MIPS και να είστε σίγουροι ότι θα πάρετε την καλύτερη απόδοση, μελετήστε τις πραγματικές εντολές του MIPS που παρουσιάζονται στις Εικόνες 2.25 και 2.27.

Οι συμβολομεταφραστές δέχονται επίσης αριθμούς σε μια ποικιλία βάσεων. Εκτός από τη δυαδική και τη δεκαδική, συνήθως δέχονται μια βάση η οποία είναι πιο συνοπτική από τη δυαδική ενώ μετατρέπεται εύκολα σε ένα δυαδικό μοτίβο (σειρά). Οι συμβολομεταφραστές του MIPS χρησιμοποιούν δεκαεξαδική (hexadecimal) βάση.

Τέτοια χαρακτηριστικά είναι βολικά, αλλά το βασικό καθήκον ενός συμβολομεταφραστή είναι η μετάφραση της συμβολικής γλώσσας σε κώδικα μηχανής. Ο συμβολομεταφραστής μετατρέπει ένα πρόγραμμα συμβολικής γλώσσας σε ένα *αντικειμενικό αρχείο* (object file), το οποίο είναι ένας συνδυασμός εντολών **γλώσσας μηχανής**, δεδομένων, και πληροφοριών που είναι απαραίτητες για τη σωστή τοποθέτηση των εντολών στη μνήμη.

γλώσσα μηχανής (machine language) Δυαδική αναπαράσταση που χρησιμοποιείται για την επικοινωνία μέσα σε ένα υπολογιστικό σύστημα.

Για να παραγάγει τη δυαδική έκδοση κάθε εντολής στο πρόγραμμα της συμβολικής γλώσσας, ο συμβολομεταφραστής πρέπει να προσδιορίσει τις διευθύνσεις που αντιστοιχούν σε όλες τις ετικέτες (labels). Οι συμβολομεταφραστές παρακολουθούν τις ετικέτες που χρησιμοποιούνται στις διακλαδώσεις (branches) και στις εντολές μεταφοράς δεδομένων καταγράφοντάς τες σε έναν **πίνακα συμβόλων** (symbol table). Όπως ίσως θα αναμένατε, ο πίνακας περιέχει ζεύγη συμβόλων και διευθύνσεων.

πίνακας συμβόλων (symbol table) Ένας πίνακας που συνδυάζει ονόματα ετικετών με τις διευθύνσεις των λέξεων της μνήμης που καταλαμβάνουν οι εντολές.

Το αντικειμενικό αρχείο για συστήματα UNIX τυπικά περιέχει 6 διακριτά μέρη:

- Η *επικεφαλίδα αντικειμενικού αρχείου* (object file header) περιγράφει το μέγεθος και τη θέση των υπολοίπων τμημάτων του αντικειμενικού αρχείου.
- Το *τμήμα κειμένου* (text segment) περιέχει τον κώδικα γλώσσας μηχανής.
- Το *τμήμα στατικών δεδομένων* (static data segment) περιέχει δεδομένα που κατανέμονται για όλη τη διάρκεια ζωής του προγράμματος. (Το UNIX επιτρέπει στα προγράμματα να χρησιμοποιούν είτε στατικά δεδομένα, τα οποία κατανέμονται για όλη τη διάρκεια εκτέλεσης του προγράμματος, είτε *δυναμικά δεδομένα* — dynamic data — τα οποία μπορούν να αυξάνονται ή να μειώνονται σε μέγεθος ανάλογα με τις ανάγκες του προγράμματος.)
- Οι *πληροφορίες επανατοποθέτησης* (relocation information) προσδιορίζουν λέξεις εντολών και δεδομένων που εξαρτώνται από απόλυτες διευθύνσεις (absolute addresses) όταν το πρόγραμμα φορτώνεται στη μνήμη.
- Ο *πίνακας συμβόλων* (symbol table) περιέχει τις υπόλοιπες ετικέτες που δεν είναι καθορισμένες, όπως εξωτερικές αναφορές.
- Οι *πληροφορίες αποσφαλμάτωσης* (debugging information) περιέχουν μια συνοπτική περιγραφή του τρόπου με τον οποίο μεταγλωττίζονται οι υπομονάδες, ώστε ένας αποσφαλματωτής (debugger) να μπορεί να συσχετίσει τις εντολές της μηχανής με τα πηγαία αρχεία της C και να κάνει αναγνώσιμες τις δομές δεδομένων.

Η επόμενη υποενότητα δείχνει πώς προσαρτώνται τέτοιες ρουτίνες οι οποίες έχουν ήδη συμβολομεταφραστεί, όπως οι ρουτίνες βιβλιοθήκης.

Πρόγραμμα σύνδεσης

Ό,τι έχουμε παρουσιάσει μέχρι τώρα προϋποθέτει ότι μια αλλαγή σε μια από τις γραμμές μιας διαδικασίας (procedure) απαιτεί μεταγλώττιση και συμβολομετάφραση ολόκληρου του προγράμματος. Η πλήρης επαναμετάφραση είναι μια φοβερή σπατάλη υπολογιστικών πόρων. Αυτή η επανάληψη είναι ιδιαίτερα δαπανηρή για τυπικές ρουτίνες βιβλιοθήκης επειδή οι προγραμματιστές θα μεταγλώττιζαν και θα συμβολομετέφραζαν ρουτίνες οι οποίες εξ ορισμού δεν αλλάζουν ποτέ. Μια εναλλακτική λύση είναι η μεταγλώττιση και η συμβολομετάφραση κάθε διαδικασίας ανεξάρτητα, ώστε κάποια αλλαγή σε μια γραμμή να απαιτεί μεταγλώττιση και συμβολομετάφραση μίας μόνο διαδικασίας. Αυτή η εναλλακτική λύση απαιτεί ένα νέο πρόγραμμα συστήματος, που λέγεται **συνδετικός διορθωτής** (link editor) ή **πρόγραμμα σύνδεσης** (linker), το οποίο παίρνει όλα τα προγράμματα γλώσσας μηχανής τα οποία έχουν συμβολομεταφραστεί ανεξάρτητα και τα συνενώνει.

πρόγραμμα σύνδεσης (linker) Λέγεται επίσης **συνδετικός διορθωτής** (link editor). Ένα πρόγραμμα του συστήματος το οποίο συνδυάζει προγράμματα γλώσσας μηχανής που έχουν συμβολομεταφραστεί ανεξάρτητα, και προσδιορίζει όλες τις μη ορισμένες ετικέτες σε ένα εκτελέσιμο αρχείο.

Υπάρχουν τρία βήματα για το πρόγραμμα σύνδεσης:

1. Τοποθέτηση υπομονάδων κώδικα και δεδομένων συμβολικά στη μνήμη.
2. Προσδιορισμός των διευθύνσεων των ετικετών δεδομένων και εντολών.
3. Επιδιόρθωση (patch) τόσο των εσωτερικών αναφορών (internal references) όσο και των εξωτερικών αναφορών (external references).

Το πρόγραμμα σύνδεσης χρησιμοποιεί τις πληροφορίες επανατοποθέτησης και τον πίνακα συμβόλων κάθε αντικειμενικής υπομονάδας για τον προσδιορισμό όλων των μη ορισμένων ετικετών. Τέτοιες αναφορές εμφανίζονται σε εντολές διακλάδωσης, εντολές άλματος, και διευθύνσεις δεδομένων, και έτσι το καθήκον αυτού του προγράμματος είναι λίγο-πολύ αυτό ενός διορθωτή: βρίσκει τις παλιές διευθύνσεις και τις αντικαθιστά με τις νέες. Η διόρθωση είναι η προέλευση του ονόματος «συνδετικός διορθωτής» ή «πρόγραμμα σύνδεσης». Ο λόγος για τον οποίο ένα πρόγραμμα διόρθωσης έχει νόημα είναι ότι είναι πολύ ταχύτερη η σύνδεση του κώδικα σε σχέση με την επαναλαμβανόμενη μεταγλώττιση και συμβολομετάφραση.

Αφού προσδιοριστούν όλες οι εξωτερικές αναφορές, το πρόγραμμα σύνδεσης κατόπιν προσδιορίζει τις θέσεις της μνήμης που θα καταλάβει κάθε υπομονάδα. Θυμηθείτε ότι η Εικόνα 2.17 στη σελίδα 105 παρουσιάζει τη σύμβαση του MIPS για την κατανομή του προγράμματος και των δεδομένων στη μνήμη. Εφόσον τα αρχεία έχουν συμβολομεταφραστεί ανεξάρτητα, ο συμβολομεταφραστής δε θα μπορούσε να γνωρίζει πού θα τοποθετηθούν οι εντολές και τα δεδομένα μιας υπομονάδας σε σχέση με τις υπόλοιπες υπομονάδες. Όταν το πρόγραμμα σύνδεσης τοποθετεί μια υπομονάδα στη μνήμη, όλες οι απόλυτες αναφορές (absolute references), δηλαδή διευθύνσεις μνήμης οι οποίες δεν είναι σχετικές ως προς έναν καταχωρητή, πρέπει να επανατοποθετηθούν ώστε να αντιστοιχούν στην πραγματική θέση της υπομονάδας.

Το πρόγραμμα σύνδεσης παράγει ένα **εκτελέσιμο αρχείο** (executable file) το οποίο μπορεί να εκτελεστεί σε υπολογιστή. Τυπικά, αυτό το αρχείο έχει την ίδια μορφή όπως ένα αντικειμενικό αρχείο, εκτός από το γεγονός ότι δεν περιέχει μη προσδιορισμένες αναφορές. Είναι δυνατόν να υπάρχουν μερικώς συνδεδεμένα αρχεία, όπως ρουτίνες βιβλιοθήκης, τα οποία εξακολουθούν να περιέχουν μη προσδιορισμένες διευθύνσεις και, επομένως, έχουν ως αποτέλεσμα αντικειμενικά αρχεία.

εκτελέσιμο αρχείο (executable file) Ένα λειτουργικό πρόγραμμα σε μορφή αντικειμενικού αρχείου, το οποίο δεν περιέχει μη προσδιορισμένες αναφορές, πληροφορίες επανατοποθέτησης, πίνακα συμβόλων, ή πληροφορίες αποσφαλμάτωσης.

ΠΑΡΑΔΕΙΓΜΑ

Σύνδεση αντικειμενικών αρχείων

Συνδέστε τα δύο επόμενα αντικειμενικά αρχεία. Δείξτε τις ενημερωμένες διευθύνσεις μερικών από τις πρώτες εντολές του ολοκληρωμένου εκτελέσιμου αρχείου. Δείχνουμε τις εντολές σε συμβολική γλώσσα για να κάνουμε το παράδειγμα κατανοητό· στην πραγματικότητα, οι εντολές θα ήταν αριθμοί.

Παρατηρήστε ότι στα αντικειμενικά αρχεία έχουμε επισημάνει τις διευθύνσεις και τα σύμβολα τα οποία πρέπει να ενημερωθούν κατά τη διαδικασία της σύνδεσης: τις εντολές οι οποίες αναφέρονται στις διευθύνσεις των διαδικασιών `A` και `B` και τις εντολές οι οποίες αναφέρονται στις διευθύνσεις των λέξεων δεδομένων `X` και `Y`.

Επικεφαλίδα αντικειμενικού αρχείου			
	Όνομα	Διαδικασία `A`	
	Μέγεθος κειμένου	100_{hex}	
	Μέγεθος δεδομένων	20_{hex}	
Τμήμα κειμένου	Διεύθυνση	Εντολή	
	0	`lw $a0, 0($gp)`	
	4	`jal 0`	
	...	...	
Τμήμα δεδομένων	0	(X)	
	...	...	
Πληροφορίες επανατοποθέτησης	Διεύθυνση	Τύπος εντολής	Εξάρτηση
	0	`lw`	X
	4	`jal`	B
Πίνακας συμβόλων	Ετικέτα	Διεύθυνση	
	X	—	
	B	—	
Επικεφαλίδα αντικειμενικού αρχείου			
	Όνομα	Διαδικασία `B`	
	Μέγεθος κειμένου	300_{hex}	
	Μέγεθος δεδομένων	30_{hex}	
	Διεύθυνση	Εντολή	
	0	`sw $a1, 0($gp)`	
	4	`jal 0`	
	...	...	
Τμήμα δεδομένων	0	(Y)	
	...	...	
Πληροφορίες επανατοποθέτησης	Διεύθυνση	Τύπος εντολής	Εξάρτηση
	0	`sw`	Y
	4	`jal`	A
Πίνακας συμβόλων	Ετικέτα	Διεύθυνση	
	Y	—	
	A	—	

ΑΠΑΝΤΗΣΗ

Η διαδικασία A πρέπει να βρει τη διεύθυνση της μεταβλητής με ετικέτα `X` ώστε να την τοποθετήσει στην εντολή load, και τη διεύθυνση της διαδικασί-

ας B ώστε να την τοποθετήσει στην εντολή jal. Η διαδικασία B χρειάζεται τη διεύθυνση της μεταβλητής με ετικέτα Y για την εντολή store, και τη διεύθυνση της διαδικασίας A για τη δική της εντολή jal.

Από την Εικόνα 2.17 της σελίδας 105, γνωρίζουμε ότι το τμήμα κειμένου ξεκινάει στη διεύθυνση $40\ 0000_{hex}$ και το τμήμα δεδομένων στη διεύθυνση $1000\ 0000_{hex}$. Το κείμενο της διαδικασίας A είναι τοποθετημένο στην πρώτη διεύθυνση, και τα δεδομένα της στη δεύτερη. Η επικεφαλίδα του αντικειμενικού αρχείου για τη διαδικασία A λέει ότι το κείμενό της έχει μήκος 100_{hex} byte και τα δεδομένα της 20_{hex} byte, οπότε η αρχική διεύθυνση για το κείμενο της διαδικασίας B είναι $40\ 0100_{hex}$, και τα δεδομένα της ξεκινούν στη διεύθυνση $1000\ 0020_{hex}$.

Επικεφαλίδα εκτελέσιμου αρχείου		
	Μέγεθος κειμένου	300_{hex}
	Μέγεθος δεδομένων	50_{hex}
Τμήμα κειμένου	Διεύθυνση	Εντολή
	$0040\ 0000_{hex}$	lw $a0, 8000_{hex} ($gp)
	$0040\ 0004_{hex}$	jal $40\ 0100_{hex}$
	...	...
	$0040\ 0100_{hex}$	sw $a1, 8020_{hex} ($gp)
	$0040\ 0104_{hex}$	jal $40\ 0000_{hex}$
	...	...
Τμήμα δεδομένων	Διεύθυνση	
	$1000\ 0000_{hex}$	(X)
	...	...
	$1000\ 0020_{hex}$	(Y)
	...	...

Τώρα το πρόγραμμα σύνδεσης ενημερώνει τα πεδία διευθύνσεων των εντολών. Χρησιμοποιεί το πεδίο τύπου εντολής για να μάθει τη μορφή της εντολής που πρέπει να διορθωθεί. Εδώ έχουμε δύο τύπους:

1. Οι εντολές jal είναι εύκολες επειδή χρησιμοποιούν ψευδο-απευθείας διευθυνσιοδότηση (pseudodirect addressing). Η jal στη διεύθυνση $40\ 0004_{hex}$ παίρνει την τιμή $40\ 0100_{hex}$ (τη διεύθυνση της διαδικασίας B) στο πεδίο διεύθυνσής της, και η εντολή jal στη διεύθυνση $40\ 0104_{hex}$ παίρνει την τιμή $40\ 0000_{hex}$ (τη διεύθυνση της διαδικασίας A) στο πεδίο διεύθυνσής της.
2. Οι διευθύνσεις των εντολών load και store είναι δυσκολότερες επειδή είναι σχετικές ως προς έναν καταχωρητή βάσης. Η Εικόνα 2.17 δείχνει ότι ο $gp παίρνει ως αρχική τιμή την $1000\ 8000_{hex}$. Για να πάρουμε τη διεύθυνση $1000\ 0000_{hex}$ (τη διεύθυνση της λέξης X), τοποθετούμε την τιμή 8000_{hex} στο πεδίο διεύθυνσης της lw στη διεύθυνση $40\ 0000_{hex}$. Το Κεφάλαιο 3 εξηγεί την αριθμητική των υπολογιστών συμπληρώματος ως προς δύο (two's complement) των 16 bit, που αποτελεί το λόγο για τον οποίο η τιμή 8000_{hex} στο πεδίο διεύθυνσης δίνει την τιμή $1000\ 0000_{hex}$ ως διεύθυνση. Όμοια, τοποθετούμε την τιμή 8020_{hex} στο πεδίο διεύθυνσης της εντολής sw στη διεύθυνση $40\ 0100_{hex}$ για να πάρουμε τη διεύθυνση $1000\ 0020_{hex}$ (τη διεύθυνση της λέξης Y).

Φορτωτής

Τώρα που το εκτελέσιμο αρχείο βρίσκεται στο δίσκο, το λειτουργικό σύστημα το διαβάζει και το τοποθετεί στη μνήμη και το ξεκινάει. Στα συστήματα UNIX ακολουθεί τα εξής βήματα:

1. Διαβάζει την επικεφαλίδα του εκτελέσιμου αρχείου για να προσδιορίσει το μέγεθος των τμημάτων κειμένου και δεδομένων.
2. Δημιουργεί ένα χώρο διευθύνσεων αρκετά μεγάλο για το κείμενο και τα δεδομένα.
3. Αντιγράφει τις εντολές και τα δεδομένα από το εκτελέσιμο αρχείο στη μνήμη.
4. Αντιγράφει τις παραμέτρους (αν υπάρχουν) για το κύριο πρόγραμμα (main program) στη στοίβα (stack).
5. Δίνει αρχικές τιμές στους καταχωρητές της μηχανής και τοποθετεί το δείκτη στοίβας (stack pointer) στην πρώτη ελεύθερη θέση.
6. Μεταπηδά σε μια ρουτίνα εκκίνησης που αντιγράφει τις παραμέτρους στους καταχωρητές ορισμάτων (argument registers) και καλεί την κύρια ρουτίνα του προγράμματος. Όταν η κύρια ρουτίνα επιστρέφει, η ρουτίνα εκκίνησης τερματίζει το πρόγραμμα με μια κλήση του συστήματος για έξοδο (`exit`).

φορτωτής (loader) Ένα πρόγραμμα συστήματος το οποίο τοποθετεί ένα αντικειμενικό πρόγραμμα στην κύρια μνήμη ώστε να είναι έτοιμο να εκτελεστεί.

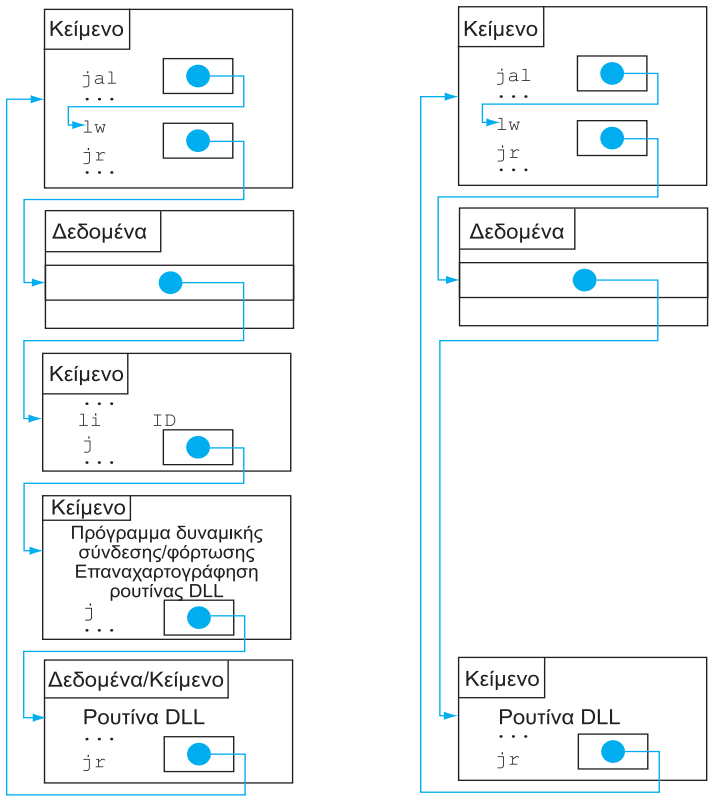
Οι Ενότητες A.3 και A.4 στο Παράρτημα Α περιγράφουν τα προγράμματα σύνδεσης και τους **φορτωτές** (loaders) πιο αναλυτικά.

Βιβλιοθήκες δυναμικής σύνδεσης

Το πρώτο τμήμα αυτής της ενότητας περιγράφει την παραδοσιακή προσέγγιση για τη σύνδεση βιβλιοθηκών πριν από την εκτέλεση του προγράμματος. Μολονότι αυτή η στατική προσέγγιση είναι ο γρηγορότερος τρόπος κλήσης ρουτινών βιβλιοθήκης, έχει μερικά μειονεκτήματα:

- Οι ρουτίνες βιβλιοθήκης γίνονται τμήμα του εκτελέσιμου κώδικα. Αν κυκλοφορήσει μια νέα έκδοση της βιβλιοθήκης που διορθώνει σφάλματα ή υποστηρίζει νέες συσκευές υλικού, το στατικά συνδεδεμένο πρόγραμμα εξακολουθεί να χρησιμοποιεί την παλιά έκδοση.
- Φορτώνει ολόκληρη τη βιβλιοθήκη ακόμη και αν αυτή δε χρησιμοποιείται ολόκληρη όταν εκτελείται το πρόγραμμα. Η βιβλιοθήκη μπορεί να είναι μεγάλη σε σχέση με το πρόγραμμα· για παράδειγμα, η πρότυπη βιβλιοθήκη της C έχει μέγεθος 2,5 MB.

Αυτά τα μειονεκτήματα οδηγούν σε δυναμικά συνδεδεμένες βιβλιοθήκες (dynamically linked libraries — DLL), ή βιβλιοθήκες δυναμικής σύνδεσης, όπου οι ρουτίνες της βιβλιοθήκης δε συνδέονται και δε φορτώνονται μέχρι να εκτελεστεί το πρόγραμμα. Τόσο το πρόγραμμα όσο και οι ρουτίνες της βιβλιοθήκης διατηρούν επιπλέον πληροφορίες για τη θέση μη τοπικών διαδικασιών και για τα ονόματά τους. Στην αρχική έκδοση των δυναμικά συνδεδεμένων βιβλιοθηκών, ο φορτωτής εκτελούσε ένα πρόγραμμα δυναμικής σύνδεσης, χρησιμοποιώντας τις επιπλέον πληροφορίες στο αρχείο για να βρει τις κατάλληλες βιβλιοθήκες και να ενημερώσει όλες τις εξωτερικές αναφορές.



α. Πρώτη κλήση ρουτίνας DLL β. Επόμενες κλήσεις ρουτίνας DLL

ΕΙΚΟΝΑ 2.29 Δυναμική σύνδεση βιβλιοθήκης μέσω «ράθυμης» (lazy) σύνδεσης διαδικασιών. (α) Τα βήματα για την πρώτη φορά που γίνεται κλήση στη ρουτίνα DLL. (β) Τα βήματα για την εύρεση της ρουτίνας, την επαναχαρτογράφησή της, και τη σύνδεσή της παραλείπονται σε διαδοχικές κλήσεις. Όπως θα δούμε στο Κεφάλαιο 7, το λειτουργικό σύστημα μπορεί να αποφύγει την αντιγραφή της κατάλληλης ρουτίνας επαναχαρτογραφώντας τη με τη χρήση διαχείρισης εικονικής μνήμης (virtual memory management).

Το μειονέκτημα των πρώτων εκδόσεων των βιβλιοθηκών δυναμικής σύνδεσης ήταν το γεγονός ότι εξακολουθούσαν να συνδέουν όλες τις ρουτίνες της βιβλιοθήκης που μπορεί να καλούνταν αντί εκείνες που καλούνταν κατά την εκτέλεση του προγράμματος. Αυτή η παρατήρηση οδήγησε στη «ράθυμη» (lazy) έκδοση σύνδεσης διαδικασιών των δυναμικά συνδεδεμένων βιβλιοθηκών, όπου κάθε ρουτίνα συνδέεται μόνο *μετά* την κλήση της.

Όπως σε πολλές περιπτώσεις στον επιστημονικό τομέα μας, αυτό το τέχνασμα βασίζεται σε ένα επίπεδο εμμεσότητας. Η Εικόνα 2.29 παρουσιάζει την τεχνική. Στην αρχή, οι μη τοπικές ρουτίνες καλούν ένα σύνολο ψευδο-ρουτινών (dummy routines) στο τέλος του προγράμματος, με μία καταχώριση για κάθε μία μη τοπική ρουτίνα. Αυτές οι ψευδο-καταχωρίσεις περιέχουν από ένα έμμεσο άλμα η κάθε μία.

Την πρώτη φορά που καλείται η ρουτίνα της βιβλιοθήκης, το πρόγραμμα καλεί την εικονική καταχώριση και ακολουθεί το έμμεσο άλμα. Δείχνει σε κώδικα που τοποθετεί έναν αριθμό σε έναν καταχωρητή για να προσδιορίσει την κατάλληλη ρουτίνα βιβλιοθήκης, και στη συνέχεια μεταπηδά στο πρόγραμμα δυναμικής σύνδεσης-φόρτωσης. Το πρόγραμμα σύνδεσης-φόρτωσης βρίσκει την κατάλληλη ρουτίνα, την επαναπεικονίζει (remap), και αλλάζει τη διεύθυνση στη θέση του έμμεσου άλματος ώστε να δείχνει σε εκείνη τη ρουτίνα.

Κατόπιν, μεταφέρεται σε αυτή. Όταν η ρουτίνα ολοκληρώνεται, το πρόγραμμα επιστρέφει στο αρχικό σημείο κλήσης. Έπειτα, εκτελεί έμμεσο άλμα στη ρουτίνα χωρίς τα επιπλέον άλματα.

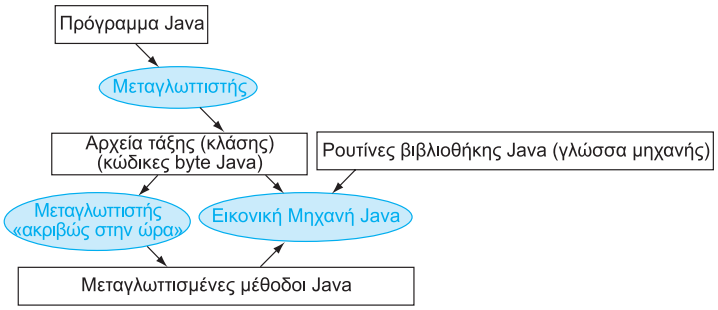
Για να συνοψίσουμε, οι DLL απαιτούν πρόσθετο χώρο για τις πληροφορίες που χρειάζονται για τη δυναμική σύνδεση, αλλά δεν απαιτούν την αντιγραφή ή τη σύνδεση ολόκληρων βιβλιοθηκών. Πληρώνουν μια σημαντική επιβάρυνση την πρώτη φορά που καλείται μια ρουτίνα, αλλά ένα μόνον έμμεσο άλμα από εκεί και έπειτα. Σημειώστε ότι η επιστροφή από τη βιβλιοθήκη δεν έχει καμία επιβάρυνση. Τα Windows της Microsoft βασίζονται εκτεταμένα σε «ράθυμες» δυναμικά συνδεδεμένες βιβλιοθήκες, και αυτός είναι και ο κανονικός τρόπος εκτέλεσης προγραμμάτων στα συστήματα UNIX σήμερα.

Εκκίνηση ενός προγράμματος Java

Η παραπάνω ανάλυση παρουσιάζει το παραδοσιακό μοντέλο εκτέλεσης ενός προγράμματος, όπου δίνεται έμφαση στο μικρό χρόνο εκτέλεσης για ένα πρόγραμμα που προορίζεται για συγκεκριμένη αρχιτεκτονική συνόλου εντολών, ή ακόμη και για μια συγκεκριμένη υλοποίηση αυτής της αρχιτεκτονικής. Πράγματι, είναι δυνατή η εκτέλεση προγραμμάτων της Java όπως ακριβώς εκείνων της C. Ωστόσο, η Java δημιουργήθηκε με ένα διαφορετικό σύνολο στόχων. Ένας στόχος ήταν η γρήγορη και ασφαλής εκτέλεση σε οποιονδήποτε υπολογιστή, ακόμη και αν αυτό αύξανε το χρόνο εκτέλεσης.

Η Εικόνα 2.30 δείχνει τα τυπικά βήματα μετάφρασης και εκτέλεσης για προγράμματα Java. Αντί να μεταγλωττίζεται στη συμβολική γλώσσα του υπολογιστή προορισμού, η Java μεταγλωττίζεται πρώτα σε εντολές που διερμηνεύονται εύκολα: το σύνολο εντολών **κώδικα byte Java** (Java bytecode). Αυτό το σύνολο εντολών είναι σχεδιασμένο με στόχο να παραμένει κοντά στην Java ώστε το συγκεκριμένο βήμα μεταγλώττισης να είναι στοιχειώδες. Στην πράξη δε γίνονται καθόλου βελτιστοποιήσεις. Όπως ο μεταγλωττιστής της C, ο μεταγλωττιστής της Java ελέγχει τους τύπους των δεδομένων και παράγει τις κατάλληλες

κώδικας byte Java (Java bytecode) Εντολή από ένα σύνολο εντολών που έχει σχεδιαστεί για να διερμηνεύει προγράμματα της γλώσσας Java.



ΕΙΚΟΝΑ 2.30 Μια ιεραρχία μετάφρασης για τη γλώσσα Java. Ένα πρόγραμμα Java μεταγλωττίζεται πρώτα σε δυαδική έκδοση κώδικα byte Java, όπου όλες οι διευθύνσεις καθορίζονται από το μεταγλωττιστή. Το πρόγραμμα της Java είναι τώρα έτοιμο να εκτελεστεί στο διερμηνευτή (interpreter), που ονομάζεται Εικονική Μηχανή Java (Java Virtual Machine — JVM). Η Εικονική Μηχανή Java συνδέει τις κατάλληλες μεθόδους της βιβλιοθήκης Java ενώ το πρόγραμμα εκτελείται. Για να επιτύχει καλύτερη απόδοση, η Εικονική Μηχανή Java μπορεί να καλέσει το μεταγλωττιστή «ακριβώς στην ώρα» (Just In Time, JIT, compiler), ο οποίος μεταγλωττίζει επιλεκτικά μεθόδους στην εγγενή γλώσσα μηχανής της μηχανής στην οποία εκτελείται.

λειτουργίες για τον κάθε τύπο. Τα προγράμματα της Java διανέμονται στη δυαδική έκδοση αυτού του κώδικα byte.

Ένας διερμηνευτής λογισμικού (software interpreter), που ονομάζεται **Εικονική Μηχανή Java** (Java Virtual Machine — JVM), μπορεί να εκτελέσει κώδικες byte Java. Διερμηνευτής είναι ένα πρόγραμμα που προσομοιώνει μια αρχιτεκτονική συνόλου εντολών. Για παράδειγμα, ο προσομοιωτής του MIPS που χρησιμοποιείται σε αυτό το βιβλίο είναι ένας διερμηνευτής. Δεν υπάρχει ανάγκη για ξεχωριστό βήμα συμβολομετάφρασης, εφόσον είτε η μετάφραση είναι τόσο απλή που ο μεταγλωττιστής συμπληρώνει τις διευθύνσεις, είτε η Εικονική Μηχανή Java τις βρίσκει κατά το χρόνο εκτέλεσης.

Εικονική Μηχανή Java (Java Virtual Machine — JVM) Το πρόγραμμα το οποίο διερμηνεύει κώδικες byte Java.

Το θετικό της διερμηνείας είναι η φορητότητα. Η διαθεσιμότητα εικονικών μηχανών Java σε λογισμικό σήμαινε ότι οι περισσότεροι θα μπορούσαν να γράψουν και να εκτελέσουν προγράμματα Java σύντομα μετά την αναγγελία της Java. Σήμερα, οι εικονικές μηχανές Java υπάρχουν σε εκατομμύρια συσκευές, από κινητά τηλέφωνα μέχρι φυλλομετρητές Διαδικτύου (Internet browsers).

Το μειονέκτημα της διερμηνείας είναι η χαμηλή απόδοση. Οι απίστευτες πρόοδοι στην απόδοση τις δεκαετίες του 1980 και του 1990 έκαναν τη διερμηνεία εφικτή για πολλές σημαντικές εφαρμογές, αλλά ο συντελεστής επιβράδυνσης 10 σε σύγκριση με παραδοσιακά μεταγλωττισμένα προγράμματα C έκανε την Java μη ελκυστική για μερικές εφαρμογές.

Για να διατηρηθεί η φορητότητα και να βελτιωθεί η ταχύτητα εκτέλεσης, η επόμενη φάση της ανάπτυξης της Java ήταν οι μεταγλωττιστές που μετέφραζαν κατά την εκτέλεση του προγράμματος. Τέτοιου είδους **μεταγλωττιστές «ακριβώς στην ώρα»** (Just In Time, JIT, compilers) τυπικά αναλύουν το προφίλ του εκτελούμενου προγράμματος για να βρουν τις κρίσιμες μεθόδους, και στη συνέχεια τις μεταγλωττίζουν στο εγγενές σύνολο εντολών στο οποίο εκτελείται η εικονική μηχανή. Το μεταγλωττισμένο τμήμα διατηρείται για την επόμενη φορά που θα εκτελεστεί το πρόγραμμα, ώστε να αυξάνεται η ταχύτητά του κάθε φορά που εκτελείται. Αυτή η ισορροπία διερμηνείας και μεταγλώττισης εξελίσσεται με το χρόνο, ώστε τα συχνά εκτελούμενα προγράμματα της Java να υποφέρουν λίγο από την επιβάρυνση της διερμηνείας.

μεταγλωττιστής «ακριβώς στην ώρα» (Just In Time, JIT, compiler) Το όνομα που δίνεται συνήθως σε ένα μεταγλωττιστή ο οποίος λειτουργεί κατά το χρόνο εκτέλεσης, μεταφράζοντας τα διερμηνευμένα τμήματα του κώδικα σε εγγενή κώδικα του υπολογιστή.

Καθώς οι υπολογιστές γίνονται ταχύτεροι ώστε οι μεταγλωττιστές να μπορούν να κάνουν περισσότερα, και καθώς οι ερευνητές εφευρίσκουν τρόπους για την επιτόπου (on the fly) μεταγλώττιση της Java, το χάσμα απόδοσης μεταξύ της Java και της C ή της C++ κλείνει. Η Ενότητα 2.14 προχωρεί σε πολύ μεγαλύτερο βάθος στην υλοποίηση της Java, των κωδίκων byte Java (Java bytecodes), της Εικονικής Μηχανής Java (Java Virtual Machine), και των μεταγλωττιστών «ακριβώς στην ώρα».

Αυτοεξέταση

Ποια από τα πλεονεκτήματα ενός διερμηνευτή ως προς ένα μεταφραστή νομίζετε ότι ήταν τα πιο σημαντικά για τους σχεδιαστές της Java;

1. Ευκολία γραφής ενός διερμηνευτή
2. Καλύτερα μηνύματα σφαλμάτων
3. Μικρότερος αντικειμενικός κώδικας
4. Ανεξαρτησία από τη μηχανή

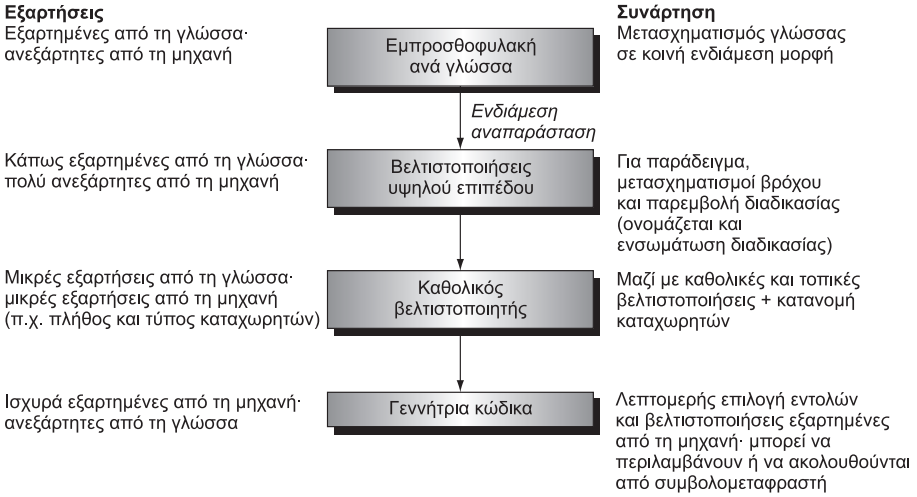
2.11 Πώς βελτιστοποιούν οι μεταγλωττιστές

Επειδή ο μεταγλωττιστής επηρεάζει σημαντικά την απόδοση ενός υπολογιστή, η κατανόηση της τεχνολογίας των μεταγλωττιστών είναι σήμερα απαραίτητη για την κατανόηση της απόδοσης. Ο σκοπός αυτής της ενότητας είναι να δώσει μια σύντομη γενική μορφή των βελτιστοποιήσεων που χρησιμοποιεί ένας μεταγλωττιστής για να επιτύχει καλύτερη απόδοση. Η επόμενη ενότητα παρουσιάζει την εσωτερική ανατομία ενός μεταγλωττιστή. Θα ξεκινήσουμε με την Εικόνα 2.31, η οποία δείχνει τη δομή πρόσφατων μεταγλωττιστών, και θα περιγράψουμε τις βελτιστοποιήσεις με τη σειρά των διελεύσεων μέσω αυτής της δομής.

Βελτιστοποιήσεις υψηλού επιπέδου

Οι βελτιστοποιήσεις υψηλού επιπέδου (high level optimizations) είναι μετασχηματισμοί οι οποίοι πραγματοποιούνται κοντά στο επίπεδο του πηγαίου κώδικα.

Ο πιο συνηθισμένος μετασχηματισμός υψηλού επιπέδου είναι πιθανόν η *παρεμβολή διαδικασίας* (procedure inlining), η οποία αντικαθιστά την κλήση σε μια συνάρτηση με το σώμα της συνάρτησης, αντικαθιστώντας τις παραμέτρους της διαδικασίας με τα ορίσματα της κλήσης. Άλλοι μετασχηματισμοί υψηλού επιπέδου περιλαμβάνουν μετασχηματισμούς βρόχου που μπορούν να μειώσουν την επιβάρυνση από το βρόχο, να βελτιώσουν την προσπέλαση της μνήμης, και να αξιοποιήσουν το υλικό αποτελεσματικότερα. Για παράδειγμα, στους βρόχους



ΕΙΚΟΝΑ 2.31 Η δομή ενός σύγχρονου μεταγλωττιστή βελτιστοποίησης αποτελείται από έναν αριθμό διελεύσεων ή φάσεων. Λογικά, κάθε διέλευση μπορεί να θεωρηθεί ότι ολοκληρώνεται πριν από την επόμενη. Στην πράξη, κάποιες διελεύσεις μπορούν να χειριστούν μία διαδικασία κάθε φορά, οπότε ουσιαστικά διαπλέκονται με κάποια άλλη διέλευση.

που εκτελούν πολλές επαναλήψεις, όπως αυτοί που παραδοσιακά ελέγχονται από μια εντολή *for*, συχνά είναι χρήσιμη η βελτιστοποίηση **«ξετυλίγματος» βρόχου** (loop unrolling). Το ξετύλιγμα βρόχου περιλαμβάνει την επανάληψη του σώματος του βρόχου πολλές φορές, και την εκτέλεση του μετασχηματισμένου βρόχου λιγότερες φορές. Το ξετύλιγμα βρόχου μειώνει την επιβάρυνση λόγω του βρόχου και παρέχει ευκαιρίες για πολλές άλλες βελτιστοποιήσεις. Άλλοι τύποι μετασχηματισμών υψηλού επιπέδου περιλαμβάνουν εξελιγμένους μετασχηματισμούς βρόχου όπως η εναλλαγή ένθετων βρόχων (nested loops interchange) και η τμηματοποίηση βρόχων (loops blocking) οι οποίοι οδηγούν σε καλύτερη συμπεριφορά της μνήμης· για παραδείγματα, δείτε το Κεφάλαιο 7.

«ξετύλιγμα» βρόχου (loop unrolling) Μια τεχνική για την αύξηση της απόδοσης βρόχων που προσπελάζουν πίνακες, στην οποία δημιουργούνται πολλά αντίγραφα του σώματος του βρόχου, και εντολές από διαφορετικές επαναλήψεις (iterations) χρονοπρογραμματίζονται (scheduled) μαζί.

Τοπικές και καθολικές βελτιστοποιήσεις

Μέσα στη διέλευση που είναι αφιερωμένη στην τοπική και καθολική βελτιστοποίηση, εκτελούνται τρεις κατηγορίες βελτιστοποιήσεων:

1. Η *τοπική βελτιστοποίηση* (local optimization) δουλεύει μέσα σε ένα βασικό μπλοκ (basic block). Μια διέλευση τοπικής βελτιστοποίησης εκτελείται συχνά πριν και μετά από την καθολική βελτιστοποίηση για να «καθαρίσει» τον κώδικα πριν και μετά από αυτή.
2. Η *καθολική βελτιστοποίηση* (global optimization) καλύπτει πολλά βασικά μπλοκ· θα δούμε ένα παράδειγμά της σύντομα.
3. Η καθολική *κατανομή καταχωρητών* (register allocation) κατανέμει μεταβλητές σε καταχωρητές για περιοχές του κώδικα. Η κατανομή καταχωρητών είναι κρίσιμη για την καλή απόδοση στους σύγχρονους επεξεργαστές.

Πολλές βελτιστοποιήσεις πραγματοποιούνται τόσο τοπικά όσο και καθολικά, μεταξύ των οποίων η εξάλειψη κοινών υποεκφράσεων (common subexpression elimination), η διάδοση σταθερών (constant propagation), η διάδοση αντιγράφων (copy propagation), και η μείωση της δύναμης (strength reduction). Ας δούμε μερικά απλά παραδείγματα αυτών των βελτιστοποιήσεων.

Η *εξάλειψη κοινών υποεκφράσεων* βρίσκει πολλαπλές παρουσίες της ίδιας έκφρασης και αντικαθιστά τη δεύτερη με μια αναφορά στην πρώτη. Θεωρήστε, για παράδειγμα, ένα τμήμα κώδικα που προσθέτει το 4 στο στοιχείο ενός πίνακα:

```
x[i]= x[i]+4
```

Ο υπολογισμός της διεύθυνσης του `x[i]` γίνεται δύο φορές και είναι πανομοιότυπος, αφού ούτε η αρχική διεύθυνση του `x` ούτε η τιμή του `i` αλλάζουν. Συνεπώς, ο υπολογισμός μπορεί να ξαναχρησιμοποιηθεί. Ας δούμε τον ενδιάμεσο κώδικα γι' αυτό το τμήμα, αφού επιτρέπει να γίνουν πολλές άλλες βελτιστοποιήσεις. Παρουσιάζεται ένας μη βελτιστοποιημένος ενδιάμεσος κώδικας αριστερά, και δεξιά ο κώδικας με εξάλειψη κοινών υποεκφράσεων που αντικαθιστά το δεύτερο υπολογισμό της διεύθυνσης με τον πρώτο. Σημειώστε ότι η κατανομή των καταχωρητών δεν έχει ακόμη πραγματοποιηθεί, και έτσι ο μεταγλωττιστής χρησιμοποιεί εικονικούς αριθμούς καταχωρητών, όπως `R100` εδώ.

```
# x[i] + 4
li R100,x
lw R101,i
mult R102,R101,4
add R103,R100,R102
lw R104,0(R103)
# η τιμή του x[i] είναι
# στον R104
add R105,R104,4
# x[i] =
li R106,x
lw R107,i
mult R108,R107,4
add R109,R106,R107
sw R105,0(R109)
```

```
# x[i] + 4
li R100,x
lw R101,i
mult R102,R101,4
add R103,R100,R102
lw R104,0(R103)
# η τιμή του x[i] είναι
# στον R104
add R105,R104,4
# x[i] =
sw R105,0(R103)
```

Αν η ίδια βελτιστοποίηση ήταν δυνατή σε δύο βασικά μπλοκ, θα αποτελούσε ένα παράδειγμα *καθολικής εξάλειψης κοινής υποέκφρασης*.

Ας εξετάσουμε μερικές από τις υπόλοιπες βελτιστοποιήσεις:

- Η *μείωση της δύναμης* (strength reduction) αντικαθιστά πολύπλοκες πράξεις (λειτουργίες) με απλούστερες και μπορεί να εφαρμοστεί σε αυτό το τμήμα κώδικα, αντικαθιστώντας την εντολή `mult` (πολλαπλασιασμό) με μια αριστερή ολίσθηση (left shift).
- Η *διάδοση σταθεράς* (constant propagation) και η αδελφή βελτιστοποίηση *δίπλωμα σταθεράς* (constant folding) βρίσκουν σταθερές στον κώδικα και τις διαδίδουν, συμπτύσσοντας σταθερές τιμές όποτε αυτό είναι δυνατόν.
- Η *διάδοση αντιγράφου* (copy propagation) διαδίδει τιμές οι οποίες είναι απλά αντίγραφα, εξαλείφοντας την ανάγκη για επαναφόρτωση τιμών και πιθανόν δίνοντας τη δυνατότητα για άλλες βελτιστοποιήσεις όπως η εξάλειψη κοινών υποεκφράσεων.
- Η *εξάλειψη νεκρής αποθήκευσης* (dead store elimination) βρίσκει αποθηκεύσεις τιμών οι οποίες δε χρησιμοποιούνται ξανά και τις εξαλείφει· «εξαδέλφη» της είναι η *εξάλειψη νεκρού κώδικα* (dead code elimination), η οποία βρίσκει μη χρησιμοποιούμενο κώδικα — κώδικα ο οποίος δεν μπορεί να επηρεάσει το τελικό αποτέλεσμα του προγράμματος — και τον εξαλείφει. Με την αυξημένη χρήση των μακροεντολών (macros), των προτύπων (templates), και των παρόμοιων τεχνικών που έχουν σχεδιαστεί για την επαναχρησιμοποίηση του κώδικα στις γλώσσες υψηλού επιπέδου, ο νεκρός κώδικας εμφανίζεται εντυπωσιακά συχνά.

Κατανόηση της απόδοσης του προγράμματος

Οι προγραμματιστές που ανησυχούν για την απόδοση των κρίσιμων βρόχων, ειδικά σε εφαρμογές πραγματικού χρόνου (real time applications) ή ενσωματωμένες εφαρμογές (embedded applications), συχνά ψάχνουν επίμονα τη συμβολική γλώσσα που παράγεται από ένα μεταγλωττιστή και αναρωτιούνται γιατί ο μεταγλωττιστής απέτυχε να πραγματοποιήσει μερικές καθολικές βελτιστοποιήσεις ή να κατανείμει μεταβλητές σε καταχωρητές σε όλη την έκταση ενός βρόχου. Η απάντηση συχνά βρίσκεται στην «επιταγή» ότι ο μεταγλωττιστής πρέπει να είναι συντηρητικός. Η δυνατότητα για βελτίωση του κώδικα μπορεί να φαίνεται προφανής στον προγραμματιστή, αλλά ο προγραμματιστής έχει συχνά γνώση που ο μεταγλωττιστής δεν έχει, όπως η απουσία ψευδωνυμίας (aliasing) μεταξύ δύο δεικτών (pointers) ή η απουσία παρενεργειών (side effects) από την κλήση μιας συνάρτησης. Ο μεταγλωττιστής μπορεί πραγματικά να είναι ικανός να εκτελέσει το μετασχηματισμό με μια μικρή βοήθεια, η οποία θα μπορούσε να εξαλείψει τη χειρότερη περίπτωση (worst case) συμπεριφοράς που πρέπει να υποθέσει. Αυτή η βαθιά γνώση παρουσιάζει μια σημαντική παρατήρηση: οι προγραμματιστές που χρησιμοποιούν δείκτες για να βελτιώσουν την απόδοση στην προσπέλαση μεταβλητών, και ειδικά δείκτες προς τιμές στη στοίβα οι οποίες έχουν επίσης ονόματα ως μεταβλητές ή ως στοιχεία πινάκων, πιθανόν να απενεργοποιήσουν πολλές βελτιστοποιήσεις του μεταγλωττιστή. Το τελικό αποτέλεσμα είναι ότι ο χαμηλού επιπέδου κώδικας με δείκτες μπορεί να εκτελείται όχι καλύτερα, αλλά ακόμη και χειρότερα, από τον υψηλότερου επιπέδου κώδικα που βελτιστοποιείται από το μεταγλωττιστή.

Οι μεταγλωττιστές πρέπει να είναι *συντηρητικοί*. Το πρώτο καθήκον ενός μεταγλωττιστή είναι η παραγωγή σωστού κώδικα· το δεύτερο καθήκον του είναι συνήθως η παραγωγή γρήγορου κώδικα, αν και μερικές φορές μπορεί να είναι επίσης σημαντικοί και άλλοι παράγοντες, όπως το μέγεθος του κώδικα. Κώδικας ο οποίος είναι γρήγορος αλλά λανθασμένος — για οποιονδήποτε δυνατό συνδυασμό δεδομένων εισόδου — είναι απλώς λάθος. Συνεπώς, όταν λέμε ότι ένας μεταγλωττιστής είναι «συντηρητικός» εννοούμε ότι πραγματοποιεί μια βελτιστοποίηση μόνον αν γνωρίζει με 100% βεβαιότητα ότι, για οποιεσδήποτε εισόδους, ο κώδικας θα εκτελεστεί όπως τον έγραψε ο χρήστης. Μια και οι περισσότεροι μεταγλωττιστές μεταφράζουν και βελτιστοποιούν μία συνάρτηση ή διαδικασία κάθε στιγμή, οι περισσότεροι μεταγλωττιστές, ειδικά στα χαμηλότερα επίπεδα βελτιστοποίησης, υποθέτουν το χειρότερο σχετικά με τις κλήσεις συναρτήσεων και τις παραμέτρους τους.

Καθολικές βελτιστοποιήσεις κώδικα

Πολλές καθολικές βελτιστοποιήσεις κώδικα έχουν τους ίδιους σκοπούς με εκείνες που χρησιμοποιούνται τοπικά, μεταξύ των οποίων και η εξάλειψη κοινής υποέκφρασης, η διάδοση σταθεράς, η διάδοση αντιγράφου, και η εξάλειψη της νεκρής αποθήκευσης και του νεκρού κώδικα.

Υπάρχουν άλλες δύο σημαντικές καθολικές βελτιστοποιήσεις: η κίνηση κώδικα (code motion) και η εξάλειψη επαγωγικής μεταβλητής (induction variable elimination). Και οι δύο είναι βελτιστοποιήσεις βρόχων· δηλαδή έχουν στόχο τον κώδικα στους βρόχους. Η *κίνηση κώδικα* βρίσκει τον κώδικα που είναι ανεξάρτητος από τους βρόχους: ένα συγκεκριμένο τμήμα του κώδικα υπολογίζει την ίδια τιμή σε κάθε επανάληψη του βρόχου και, γι' αυτόν το λόγο, μπορεί να υπολογιστεί μία μόνο φορά έξω από το βρόχο. Η *εξάλειψη επαγωγικής μεταβλη-*

τής είναι ένας συνδυασμός των μετασχηματισμών που μειώνουν την επιβάρυνση αριθμοδεικτοδότησης (indexing) πινάκων, ουσιαστικά αντικαθιστώντας την αριθμοδεικτοδότηση πινάκων (array indexing) με προσπελάσεις δεικτών (pointer accesses). Αντί να εξετάσουμε την εξάλειψη επαγωγικής μεταβλητής σε βάθος, παραπέμπουμε τον αναγνώστη στην Ενότητα 2.15, η οποία συγκρίνει τη χρήση αριθμοδεικτών πινάκων και δεικτών (pointers)· για τους περισσότερους βρόχους, ο μετασχηματισμός από τον πιο προφανή κώδικα με πίνακες στον κώδικα με δείκτες μπορεί να πραγματοποιηθεί από ένα σύγχρονο μεταγλωττιστή βελτιστοποίησης.

Σύνοψη της βελτιστοποίησης

Η Εικόνα 2.32 παρουσιάζει παραδείγματα τυπικών βελτιστοποιήσεων, και η τελευταία στήλη δείχνει το σημείο όπου γίνεται η βελτιστοποίηση στο μεταγλωττιστή gcc. Μερικές φορές είναι δύσκολο να ξεχωρίσουμε μερικές από τις απλές βελτιστοποιήσεις — τοπικές και εξαρτώμενες από τον επεξεργαστή βελτιστοποιήσεις — από τους μετασχηματισμούς που πραγματοποιούνται στη γεννήτρια κώδικα, ενώ μερικές βελτιστοποιήσεις γίνονται πολλές φορές, ειδικά τοπικές βελτιστοποιήσεις οι οποίες μπορεί να εκτελούνται πριν ή μετά την καθολική βελτιστοποίηση όπως και κατά τη διάρκεια της παραγωγής του κώδικα.

Όνομα βελτιστοποίησης	Ερμηνεία	Επίπεδο gcc
Υψηλό επίπεδο	*Ακριβώς ή κοντά στο επίπεδο του πηγαίου κώδικα· ανεξάρτητη από τον επεξεργαστή*	
Ενσωμάτωση διαδικασίας	Αντικατάσταση της κλήσης διαδικασίας από το σώμα της διαδικασίας	O3
Τοπική	*Μέσα σε «ευθύγραμμο» κώδικα*	
Εξάλειψη κοινής υποέκφρασης	Αντικατάσταση δύο παρουσιών του ίδιου υπολογισμού με ένα αντίγραφο	O1
Διάδοση σταθεράς	Αντικατάσταση όλων των παρουσιών μιας μεταβλητής, στην οποία ανατίθεται μια σταθερά, με τη σταθερά	O1
Μείωση ύψους στοίβας	Αναδιάρθρωση δένδρου εκφράσεων για τη μείωση των πόρων που απαιτούνται για τον υπολογισμό της έκφρασης	O1
Καθολική	*Κατά μήκος διακλάδωσης*	
Καθολική εξάλειψη κοινής υποέκφρασης	Όμοια με την τοπική, αλλά αυτή η έκδοση διασχίζει διακλαδώσεις	O2
Διάδοση αντιγράφου	Αντικατάσταση όλων των παρουσιών μιας μεταβλητής A στην οποία έχει ανατεθεί η τιμή *X* (π.χ., A = *X*) με την τιμή *X*.	O2
Κίνηση κώδικα	Αφαίρεση, από ένα βρόχο, κώδικα που υπολογίζει την ίδια τιμή σε κάθε επανάληψη του βρόχου	O2
Εξάλειψη επαγωγικής μεταβλητής	Απλοποίηση/εξάλειψη υπολογισμών διευθύνσεων πινάκων στο εσωτερικό βρόχων	O2
Εξαρτημένη από τον επεξεργαστή	*Εξαρτάται από τη γνώση του επεξεργαστή*	
Μείωση δύναμης	Πολλά παραδείγματα· αντικατάσταση ενός πολλαπλασιασμού από μια σταθερά με ολισθήσεις	O1
Χρονοπρογραμματισμός διοχέτευσης	Αναδιάταξη εντολών για τη βελτίωση της απόδοσης της διοχέτευσης	O1
Βελτιστοποίηση σχετικής απόστασης διακλάδωσης	Επιλογή της συντομότερης μετατόπισης διακλάδωσης που φτάνει στον προορισμό	O1

ΕΙΚΟΝΑ 2.32 Κύριοι τύποι βελτιστοποιήσεων και παραδείγματα για κάθε κατηγορία. Η τρίτη στήλη παρουσιάζει πότε αυτές συμβαίνουν σε διαφορετικά επίπεδα βελτιστοποίησης του μεταγλωττιστή gcc. Ο οργανισμός Gnu ονομάζει τα τρία επίπεδα βελτιστοποίησης μέσο (O1), πλήρες (O2), και πλήρες με ενσωμάτωση μικρών διαδικασιών (integration of small procedures O3).

Διασύνδεση υλικού και λογισμικού

Σήμερα, ουσιαστικά όλος ο προγραμματισμός για εφαρμογές επιτραπέζιων υπολογιστών και διακομιστών γίνεται με γλώσσες υψηλού επιπέδου, όπως και το μεγαλύτερο μέρος του προγραμματισμού για ενσωματωμένες (embedded) εφαρμογές. Αυτή η εξέλιξη σημαίνει ότι, εφόσον οι περισσότερες εντολές που εκτελούνται είναι έξοδος ενός μεταγλωττιστή, ουσιαστικά μια αρχιτεκτονική συνόλου εντολών αποτελεί στόχο ενός μεταγλωττιστή. Με το νόμο του Moore έρχεται και ο πειρασμός της προσθήκης πολύπλοκων λειτουργιών σε ένα σύνολο εντολών. Το πρόβλημα είναι ότι ίσως δεν ταιριάζουν ακριβώς με αυτό που πρέπει να παραγάγει ο μεταγλωττιστής, ή μπορεί να είναι τόσο γενικές ώστε να μην είναι γρήγορες. Για παράδειγμα, θεωρήστε τις ειδικές εντολές βρόχου που συναντώνται σε μερικούς υπολογιστές. Υποθέστε ότι, αντί για τη μείωση κατά ένα, ο μεταγλωττιστής θέλει αύξηση κατά 4, ή αντί για διακλάδωση σε περίπτωση μη μηδενικής τιμής, ο μεταγλωττιστής θέλει διακλάδωση αν ο αριθμοδείκτης (index) είναι μικρότερος ή ίσος με το όριο. Η εντολή βρόχου μπορεί να μην είναι κατάλληλη. Όταν αντιμετωπίζει τέτοια εμπόδια, ο σχεδιαστής του συνόλου εντολών μπορεί να γενικεύσει τη λειτουργία, προσθέτοντας άλλον έναν τελεστέο (operand) για τον καθορισμό του βήματος της αύξησης, και ίσως μια επιλογή για το ποια συνθήκη διακλάδωσης να χρησιμοποιήσει. Ο κίνδυνος τότε είναι μια συνηθισμένη περίπτωση, όπως για παράδειγμα η αύξηση κατά ένα, να είναι πιο αργή από μια ακολουθία απλών πράξεων (λειτουργιών).

Β ΤΟΜΟΣ 2.12 Πώς δουλεύουν οι μεταγλωττιστές: εισαγωγή

Ο σκοπός αυτής της ενότητας είναι να δώσει μια σύντομη εικόνα της λειτουργίας του μεταγλωττιστή, που θα βοηθήσει τον αναγνώστη να καταλάβει πώς ο μεταγλωττιστής μεταφράζει ένα πρόγραμμα γλώσσας υψηλού επιπέδου σε εντολές μηχανής. Μην ξεχνάτε ότι το αντικείμενο της κατασκευής ενός μεταγλωττιστή συνήθως διδάσκεται σε ένα μάθημα ενός ή δύο εξαμήνων· η εισαγωγή μας αναγκαστικά θα καλύψει μόνο τα βασικά. Το υπόλοιπο αυτής της ενότητας βρίσκεται στο δεύτερο τόμο.

2.13 Ένα παράδειγμα ταξινόμησης στη C που τα συνδυάζει όλα

Ένας κίνδυνος από την παρουσίαση κομματιών κώδικα συμβολικής γλώσσας είναι ότι δε θα έχετε καμία ιδέα σχετικά με το πώς μοιάζει ένα πλήρες πρόγραμμα συμβολικής γλώσσας. Στην ενότητα αυτή, εξάγουμε τον κώδικα του MIPS από δύο διαδικασίες γραμμένες σε C: μια για την αντιμετάθεση (swap) στοιχείων ενός πίνακα και μια για την ταξινόμησή τους (sort).

Η διαδικασία `swap`

Ας ξεκινήσουμε με τον κώδικα για τη διαδικασία `swap` που βλέπουμε στην Εικόνα 2.33. Αυτή η διαδικασία απλώς αντιμεταθέτει δύο θέσεις στη μνήμη. Όταν μεταφράζουμε από τη C στη συμβολική γλώσσα «με το χέρι», ακολουθούμε τα εξής γενικά βήματα:

1. Κατανομή καταχωρητών στις μεταβλητές του προγράμματος
2. Παραγωγή κώδικα για το σώμα της διαδικασίας
3. Διατήρηση των καταχωρητών σε όλη τη διάρκεια κλήσης της διαδικασίας

Αυτή η ενότητα περιγράφει αυτά τα τρία μέρη της διαδικασίας `swap`, και καταλήγει συναρμολογώντας όλα τα κομμάτια.

Κατανομή καταχωρητών για τη `swap`

Όπως είπαμε στη σελίδα 97, η σύμβαση του MIPS για τη μεταβίβαση παραμέτρων είναι η χρήση των καταχωρητών `$a0`, `$a1`, `$a2`, και `$a3`. Εφόσον η διαδικασία `swap` έχει μόνο δύο παραμέτρους, τις `v` και `k`, αυτές θα βρίσκονται στους καταχωρητές `$a0` και `$a1`. Η μόνη άλλη μεταβλητή είναι η `temp`, την οποία συσχετίζουμε με τον καταχωρητή `$t0` εφόσον η διαδικασία `swap` είναι μια διαδικασία-φύλλο (leaf procedure — δείτε τη σελίδα 101). Αυτή η κατανομή καταχωρητών αντιστοιχεί στις δηλώσεις μεταβλητών στο πρώτο τμήμα της διαδικασίας `swap` στην Εικόνα 2.33.

Κώδικας για το σώμα της διαδικασίας `swap`

Οι υπόλοιπες γραμμές του κώδικα C της διαδικασίας `swap` είναι

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Θυμηθείτε ότι η διεύθυνση μνήμης στον MIPS αναφέρεται σε διεύθυνση *byte*, και έτσι οι λέξεις στην πραγματικότητα απέχουν κατά 4 byte. Συνεπώς, χρειάζεται να πολλαπλασιάσουμε τον αριθμοδείκτη `k` με το 4 πριν τον προσθέσουμε στη διεύθυνση. *Το να ξεχνούμε ότι οι διευθύνσεις διαδοχικών λέξεων διαφέρουν κατά 4 αντί για 1 είναι ένα συνηθισμένο λάθος στον προγραμματισμό*

```
void swap(int v[], int k)
{
  int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

ΕΙΚΟΝΑ 2.33 Μια διαδικασία σε γλώσσα C, που αντιμεταθέτει δύο θέσεις στη μνήμη. Η επόμενη υποενότητα χρησιμοποιεί αυτή τη διαδικασία σε ένα παράδειγμα ταξινόμησης.

συμβολικής γλώσσας. Συνεπώς, το πρώτο βήμα είναι να πάρουμε τη διεύθυνση του στοιχείου `v[k]` πολλαπλασιάζοντας τον αριθμοδείκτη `k` με το 4:

```
sll $t1,$a1,2      # καταχωρητής $t1 = k * 4
add $t1,$a0,$t1    # καταχωρητής $t1 = v + (k * 4)
                   # καταχωρητής $t1 περιέχει τη διεύθυνση
                   # του στοιχείου v[k]
```

Τώρα φορτώνουμε το στοιχείο `v[k]` χρησιμοποιώντας τον καταχωρητή `$t1`, και μετά το στοιχείο `v[k+1]` προσθέτοντας 4 στον καταχωρητή `$t1`:

```
lw $t0,0($t1)      # καταχωρητής $t0 (προσωρινός)= v[k]
lw $t2,4($t1)      # καταχωρητής $t2 = v[k+1]
                   # αναφέρεται στο επόμενο στοιχείο του v
```

Μετά αποθηκεύουμε τους καταχωρητές `$t0` και `$t2` στις διευθύνσεις που έχουν αντιμετατεθεί:

```
sw $t2,0($t1)      # v[k] = καταχωρητής $t2
sw $t0,4($t1)      # v[k+1] = καταχωρητής $t0 (προσωρινός)
```

Τώρα έχουμε κατανείμει καταχωρητές και έχουμε γράψει τον κώδικα για την εκτέλεση των λειτουργιών της διαδικασίας. Αυτό που λείπει είναι ο κώδικας για τη διατήρηση των αποθηκευμένων καταχωρητών που χρησιμοποιούνται στο εσωτερικό της διαδικασίας `swap`. Εφόσον δε χρησιμοποιούμε καταχωρητές που αποθηκεύονται σε αυτή τη διαδικασία-φύλλο, δεν υπάρχει τίποτε να διατηρήσουμε.

Η πλήρης διαδικασία `swap`

Είμαστε τώρα έτοιμοι για την πλήρη ρουτίνα, η οποία περιλαμβάνει την ετικέτα της διαδικασίας και το άλμα επιστροφής (return jump). Για να κάνουμε ευκολότερη την παρακολούθηση, στην Εικόνα 2.34 προσδιορίζουμε κάθε τμήμα του κώδικα και το ρόλο του στη διαδικασία.

Η διαδικασία `sort`

Για να εξασφαλίσουμε ότι θα εκτιμήσετε την αυστηρότητα του προγραμματισμού στη συμβολική γλώσσα, θα δοκιμάσουμε ένα δεύτερο, μεγαλύτερο παράδειγμα. Στην περίπτωση αυτή, θα κατασκευάσουμε μια ρουτίνα η οποία καλεί τη ρουτίνα `swap`. Αυτό το πρόγραμμα ταξινομεί έναν πίνακα ακεραίων, χρησιμοποιώντας την ταξινόμηση φυσαλίδας (bubble sort) ή ανταλλαγής (exchange sort), η οποία είναι μια από τις απλούστερες, αν όχι τις ταχύτερες, ταξινομήσεις. Η Εικόνα 2.35 δείχνει την έκδοση του προγράμματος σε γλώσσα C. Και πάλι, παρουσιάζουμε αυτή τη διαδικασία σε διάφορα βήματα, καταλήγοντας στην πλήρη διαδικασία.

Κατανομή καταχωρητών για τη `sort`

Οι δύο παράμετροι της διαδικασίας `sort`, οι `v` και `n`, βρίσκονται στους καταχωρητές παραμέτρων `$a0` και `$a1`, και αναθέτουμε τον καταχωρητή `$s0` στη μεταβλητή `i` και τον καταχωρητή `$s1` στη μεταβλητή `j`.

Σώμα διαδικασίας

```
swap: sll $t1, $a1, 2    # καταχωρητής $t1 = k * 4
      add $t1, $a0, $t1  # καταχωρητής $t1 = v + (k * 4)
                         # καταχωρητής $t1 περιέχει τη
                         # διεύθυνση του v[k]
      lw  $t0, 0($t1)    # καταχωρητής $t0 (προσωρινός) = v[k]
      lw  $t2, 4($t1)    # καταχωρητής $t2 = v[k + 1]
                         # αναφέρεται στο επόμενο στοιχείο του v
      sw  $t2, 0($t1)    # v[k] = καταχωρητής $t2
      sw  $t0, 4($t1)    # v[k+1] = καταχωρητής $t0 (προσωρινός)
```

Επιστροφή της διαδικασίας

```
      jr  $ra            # επιστροφή στην καλούσα ρουτίνα
```

ΕΙΚΟΝΑ 2.34 Κώδικας συμβολικής γλώσσας του MIPS για τη διαδικασία `swap` της Εικόνας 2.33.

```
void sort (int v[], int n)
{
   int i, j;
   for (i = 0; i < n; i += 1) {
      for (j = i - 1; j >= 0 && v[j] > v[j + 1]; j -= 1) {
           swap(v,j);
      }
   }
}
```

ΕΙΚΟΝΑ 2.35 Μια διαδικασία σε C που πραγματοποιεί ταξινόμηση του πίνακα `v`.

Κώδικας για το σώμα της διαδικασίας `sort`

Το σώμα της διαδικασίας αποτελείται από δύο ένθετους βρόχους *for* και μία κλήση στη διαδικασία `swap` η οποία περιλαμβάνει παραμέτρους. Ας ξεδιπλώσουμε τον κώδικα από έξω προς το μέσο του.

Το πρώτο βήμα της μετάφρασης είναι ο πρώτος βρόχος *for*:

```
for (i = 0; i < n; i += 1){
```

Θυμηθείτε ότι η εντολή *for* της C έχει τρία τμήματα: ανάθεση αρχικών τιμών (initialization), έλεγχο βρόχου (loop test), και αύξηση βήματος επανάληψης (iteration increment). Χρειάζεται μόνο μία εντολή για την ανάθεση αρχικής τιμής 0 στο `i`, που είναι το πρώτο μέρος της εντολής *for*:

```
move $s0,$zero           # i = 0
```

(Θυμηθείτε ότι η εντολή `move` είναι μια ψευδοεντολή που παρέχεται από το συμβολομεταφραστή για τη διευκόλυνση των προγραμματιστών της συμβολικής γλώσσας· δείτε τη σελίδα 125). Απαιτείται επίσης μία μόνον εντολή για την αύξηση του `i`, στο τελευταίο τμήμα της εντολής *for*:

```
addi $s0,$s0,1           # i += 1
```

Έξοδος από το βρόχο πρέπει να γίνει αν η συνθήκη `i < n` *δεν* είναι αληθής ή, με άλλα λόγια, πρέπει να γίνει έξοδος αν `i ≥ n`. Η εντολή set on less than δίνει στον καταχωρητή `$t0` την τιμή 1 αν `$s0 < $a1` και 0 διαφορετικά. Εφόσον

θέλουμε να ελέγξουμε αν $s0 ≥ $a1, ακολουθούμε τη διακλάδωση αν ο καταχωρητής $t0 είναι 0. Αυτός ο έλεγχος χρειάζεται δύο εντολές:

```
for1tst:slt   $t0,$s0,$a1       # καταχωρητής $t0 = 0 αν
                                # $s0 ≥ $a1 (i ≥ n)
        beq   $t0,$zero,exit1   # μετάβαση στην exit1 αν
                                # $s0 ≥ $a1 (i ≥ n)
```

Το τελευταίο μέρος του βρόχου εκτελεί άλμα πίσω στον έλεγχο του βρόχου:

```
          j   for1tst           # άλμα στον έλεγχο του
                                # εξωτερικού βρόχου
exit1:
```

Ο σκελετός κώδικα του πρώτου βρόχου *for* είναι τότε

```
        move  $s0,$zero         # i = 0
for1tst:slt   $t0,$s0,$a1       # καταχωρητής $t0 = 0 αν
                                # $s0 ≥ $a1 (i ≥ n)
        beq   $t0,$zero,exit1   # μετάβαση στην exit1 αν
                                # $s0 ≥ $a1 (i ≥ n)
            ...
              (σώμα του πρώτου βρόχου for)
            ...
        addi  $s0,$s0,1         # i += 1
        j     for1tst           # άλμα στον έλεγχο του
                                # εξωτερικού βρόχου
exit1:
```

Τέλος! Η Άσκηση 2.14 διερευνά τη γραφή ταχύτερου κώδικα για παρόμοιους βρόχους.

Ο δεύτερος βρόχος *for* σε C είναι ως εξής:

```
for (j =i - 1; j >= 0 && v[j] > v[j + 1]; j -= 1) {
```

Το τμήμα ανάθεσης αρχικών τιμών αυτού του βρόχου αποτελείται πάλι από μία εντολή:

```
addi    $s1,$s0,-1      # j = i - 1
```

Η μείωση του j στο τέλος του βρόχου είναι επίσης μία εντολή:

```
addi    $s1,$s1,-1      # j -= 1
```

Ο έλεγχος του βρόχου έχει δύο μέρη. Βγαίνουμε από το βρόχο αν αποτύχει οποιαδήποτε από τις δύο συνθήκες, οπότε ο πρώτος έλεγχος πρέπει να προκαλέσει έξοδο από το βρόχο αν αποτύχει (j < 0):

```
for2tst:slti $t0,$s1,0          # καταχωρητής $t0 = 1 αν
                                # $s1 < 0 (j < 0)
        bne  $t0,$zero,exit2    # μετάβαση στην exit2 αν
                                # $s1<0 (j<0)
```

Αυτή η διακλάδωση θα παρακάμψει τον έλεγχο της δεύτερης συνθήκης. Αν δεν την παρακάμψει, j ≥ 0.

Ο δεύτερος έλεγχος προκαλεί έξοδο αν η συνθήκη v[j] > v[j + 1] *δεν* είναι αληθής, ή αν v[j] ≤ v[j + 1]. Πρώτα δημιουργούμε τη διεύθυνση πολλαπλασιάζοντας το j με το 4 (εφόσον χρειαζόμαστε μια διεύθυνση byte) και προσθέτοντας τη διεύθυνση βάσης του v:

```
sll  $t1,$s1,2          # καταχωρητής $t1 = j * 4
add  $t2,$a0,$t1        # καταχωρητής $t2 = v + (j * 4)
```

Τώρα φορτώνουμε το στοιχείο `v[j]`:

```
lw  $t3,0($t2)          # καταχωρητής $t3 = v[j]
```

Εφόσον γνωρίζουμε ότι το δεύτερο στοιχείο είναι απλώς η επόμενη λέξη, προσθέτουμε 4 στη διεύθυνση που βρίσκεται στον καταχωρητή `$t2` για να πάρουμε το `v[j + 1]`:

```
lw  $t4,4($t2)          # καταχωρητής $t4 = v[j + 1]
```

Ο έλεγχος `v[j]` ≤ `v[j + 1]` είναι ίδιος με τον `v[j + 1]` ≥ `v[j]`, και επομένως οι δύο εντολές του ελέγχου εξόδου είναι

```
slt $t0,$t4,$t3         # καταχωρητής $t0 = 0 αν $t4 ≥ $t3
beq $t0,$zero,exit2     # μετάβαση στην exit2 αν $t4 ≥ $t3
```

Το τελευταίο τμήμα του βρόχου εκτελεί άλμα πίσω στον έλεγχο του εσωτερικού βρόχου:

```
j    for2tst      # άλμα στον έλεγχο του εσωτερικού βρόχου
```

Αν συναρμολογήσουμε τα κομμάτια αυτά, ο σκελετός του δεύτερου βρόχου *for* είναι ως εξής:

```
        addi $s1,$s0,-1      # j = i - 1
for2tst:slti $t0,$s1,0       # καταχωρητής $t0 = 1 αν $s1 <0
                             # (j<0)
        bne  $t0,$zero,exit2 # μετάβαση στην exit2 αν $s1<0
                             # (j<0)
        sll  $t1,$s1,2       # καταχωρητής $t1 = j * 4
        add  $t2,$a0,$t1     # καταχωρητής
                             # $t2 = v + (j * 4)
        lw   $t3,0($t2)      # καταχωρητής $t3 = v[j]
        lw   $t4,4($t2)      # καταχωρητής $t4 = v[j + 1]
        slt  $t0,$t4,$t3     # καταχωρητής
                             # $t0 = 0 αν $t4 ≥$t3
        beq  $t0,$zero,exit2 # μετάβαση στην exit2 αν $t4 ≥ $t3
             ...
             (σώμα του δεύτερου βρόχου for)
             ...
        addi $s1,$s1,-1      # j -= 1
        j    for2tst         # άλμα στον έλεγχο του εσωτερικού
                             # βρόχου
exit2:
```

Η κλήση διαδικασίας στη `sort`

Το επόμενο βήμα είναι το σώμα του δεύτερου βρόχου *for*:

```
swap(v,j);
```

Η κλήση της διαδικασίας swap είναι αρκετά εύκολη:

```
jal  swap
```

Μεταβίβαση παραμέτρων στη sort

Το πρόβλημα παρουσιάζεται όταν θέλουμε να μεταβιβάσουμε παραμέτρους, επειδή η διαδικασία sort χρειάζεται τις τιμές στους καταχωρητές $a0 και $a1, ενώ και η διαδικασία swap χρειάζεται τις παραμέτρους της στους ίδιους αυτούς καταχωρητές. Μια λύση είναι η αντιγραφή των παραμέτρων της διαδικασίας sort σε άλλους καταχωρητές νωρίτερα στη διαδικασία, ώστε οι καταχωρητές $a0 και $a1 να γίνουν διαθέσιμοι για την κλήση της διαδικασίας swap. (Αυτή η αντιγραφή είναι γρηγορότερη από την αποθήκευση και την επαναφορά στη στοίβα). Πρώτα αντιγράφουμε τους καταχωρητές $a0 και $a1 στους καταχωρητές $s2 και $s3 κατά τη διάρκεια της διαδικασίας:

```
move    $s2,$a0   # αντιγραφή της παραμέτρου $a0 στον $s2
move    $s3,$a1   # αντιγραφή της παραμέτρου $a1 στον $s3
```

Κατόπιν, μεταβιβάζουμε τις παραμέτρους στη swap με τις εξής δύο εντολές:

```
move    $a0,$s2   # η πρώτη παράμετρος της swap είναι η v
move    $a1,$s1   # η δεύτερη παράμετρος της swap είναι η j
```

Διατήρηση των καταχωρητών στη sort

Ο μόνος κώδικας που απομένει είναι η αποθήκευση και η επαναφορά των καταχωρητών. Σαφώς, πρέπει να αποθηκεύσουμε τη διεύθυνση επιστροφής στον καταχωρητή $ra, εφόσον η sort είναι μια διαδικασία και καλείται και αυτή. Η διαδικασία sort χρησιμοποιεί επίσης τους αποθηκευμένους καταχωρητές $s0, $s1, $s2, και $s3, οπότε αυτοί πρέπει να έχουν αποθηκευτεί. Ο πρόλογος της διαδικασίας sort είναι τότε

```
addi   $sp,$sp,-20    # δημιουργία χώρου στη στοίβα για
                      # 5 καταχωρητές
sw     $ra,16($sp)    # αποθήκευση του $ra στη στοίβα
sw     $s3,12($sp)    # αποθήκευση του $s3 στη στοίβα
sw     $s2,8($sp)     # αποθήκευση του $s2 στη στοίβα
sw     $s1,4($sp)     # αποθήκευση του $s1 στη στοίβα
sw     $s0,0($sp)     # αποθήκευση του $s0 στη στοίβα
```

Η ουρά της διαδικασίας απλώς αντιστρέφει όλες αυτές τις εντολές, και στη συνέχεια προσθέτει μια εντολή jr για την επιστροφή της.

Η πλήρης διαδικασία sort

Τώρα συναρμολογούμε όλα τα κομμάτια στην Εικόνα 2.36, φροντίζοντας να αντικαταστήσουμε τις αναφορές στους καταχωρητές $a0 και $a1 στους βρόχους *for* με αναφορές στους καταχωρητές $s2 και $s3. Και πάλι για να διευκολύνουμε την παρακολούθηση του κώδικα, προσδιορίζουμε το σκοπό κάθε τμήματος του κώδικα στη διαδικασία. Στο παράδειγμα αυτό, οι 9 γραμμές του κώδικα C της διαδικασίας sort έγιναν 35 γραμμές στη συμβολική γλώσσα του MIPS.

Αποθήκευση καταχωρητών

```
sort:   addi  $sp,$sp,-20      # δημιουργία χώρου στη στοίβα για 5 καταχωρητές
        sw    $ra,16($sp)      # αποθήκευση του $ra στη στοίβα
        sw    $s3,12($sp)      # αποθήκευση του $s3 στη στοίβα
        sw    $s2,8($sp)       # αποθήκευση του $s2 στη στοίβα
        sw    $s1,4($sp)       # αποθήκευση του $s1 στη στοίβα
        sw    $s0,0($sp)       # αποθήκευση του $s0 στη στοίβα
```

Σώμα διαδικασίας

Μετακίνηση παραμέτρων	`move $s2,$a0` # αντιγραφή της παραμέτρου $a0 στον $s2 `move $s3,$a1` # αντιγραφή της παραμέτρου $a1 στον $s3
Εξωτερικός βρόχος	`move $s0,$zero` # i = 0 `for1tst:slt $t0,$s0,$a1` # καταχωρητής $t0 = 0 αν $s0 ≥ $a1 (i ≥ n) `beq $t0,$zero,exit1` # μετάβαση στην exit1 αν $s0 ≥ $a1 (i ≥ n)
Εσωτερικός βρόχος	`addi $s1, $s0,-1` # j = i – 1 `for2tst:slti $t0,$s1,0` # καταχωρητής $t0 = 1 αν $s1 <0 (j<0) `bne $t0,$zero,exit2` # μετάβαση στην exit2 αν $s1<0 (j<0) `sll $t1,$s1,2` # καταχωρητής $t1 = j * 4 `add $t2,$a0,$t1` # καταχωρητής $t2 = v + (j * 4) `lw $t3,0($t2)` # καταχωρητής $t3 = v[j] `lw $t4,4($t2)` # καταχωρητής $t4 = v[j + 1] `slt $t0,$t4,$t3` # καταχωρητής $t0 = 0 αν $t4 ≥ $t3 `beq $t0,$zero,exit2` # μετάβαση στην exit2 αν $t4 ≥ $t3
Μεταβίβαση παραμέτρων και κλήση	`move $a0,$s2` # η πρώτη παράμετρος της swap είναι η v `move $a1,$s1` # η δεύτερη παράμετρος της swap είναι η j `jal swap` # ο κώδικας της swap φαίνεται # στην Εικόνα 2.34
Εσωτερικός βρόχος	`addi $s1,$s1,-1` # j -= 1 `j for2tst` # άλμα στον έλεγχο του εσωτερικού βρόχου
Εξωτερικός βρόχος	`exit2: addi $s0, $s0, 1` # i += 1 `j for1tst` # άλμα στον έλεγχο του εξωτερικού βρόχου

Επαναφορά καταχωρητών

```
exit1:  lw    $s0, 0($sp)      # επαναφορά του $s0 από τη στοίβα
        lw    $s1, 4($sp)      # επαναφορά του $s1 από τη στοίβα
        lw    $s2, 8($sp)      # επαναφορά του $s2 από τη στοίβα
        lw    $s3,12($sp)      # επαναφορά του $s3 από τη στοίβα
        lw    $ra,16($sp)      # επαναφορά του $ra από τη στοίβα
        addi  $sp,$sp, 20      # επαναφορά του δείκτη στοίβας
```

Επιστροφή διαδικασίας

```
        jr    $ra              # επιστροφή στην καλούσα ρουτίνα
```

ΕΙΚΟΝΑ 2.36 Κώδικας συμβολικής γλώσσας του επεξεργαστή MIPS για τη διαδικασία `sort` της Εικόνας 2.35 στη σελίδα 142.

Επιπλέον ανάπτυξη: Μια βελτιστοποίηση κατάλληλη γι' αυτό το παράδειγμα είναι η παρεμβολή διαδικασίας (procedure inlining), που αναφέραμε στην Ενότητα 2.11. Αντί για τη μεταβίβαση ορισμάτων σε παραμέτρους και την κλήση του κώδικα με μια εντολή `jal`, ο μεταγλωττιστής αντιγράφει τον κώδικα από το σώμα της διαδικασίας `swap`, όπου η κλήση στη διαδικασία εμφανίζεται στον κώδικα. Η παρεμβολή της διαδικασίας στον κώδικα θα οδηγούσε στην αποφυγή τεσσάρων εντολών σε αυτό το παράδειγμα. Το μειονέκτημα της βελτιστοποίησης με παρεμβολή είναι ότι ο μεταγλωττισμένος κώδικας είναι μεγαλύτερος αν η εμβόλιμη διαδικασία καλείται από διαφορετικές θέσεις. Μια τέτοια επέκταση του κώδικα θα μπορούσε να οδηγήσει σε *χαμηλότερη* απόδοση αν αύξανε το ρυθμό αστοχίας της κρυφής μνήμης (cache miss rate)· δείτε το Κεφάλαιο 7.

Οι μεταγλωττιστές του MIPS κρατούν πάντοτε χώρο στη στοίβα για την περίπτωση που πρέπει να αποθηκευτούν τα ορίσματα, με αποτέλεσμα στην πραγματικότητα να μειώνουν πάντα τον καταχωρητή `$sp` (δείκτη στοίβας) κατά 16 ώστε να δημιουργήσουν χώρο και για τους τέσσερις καταχωρητές ορίσματος (16 byte). Ένας λόγος είναι ότι η C παρέχει μια επιλογή `vararg` η οποία επιτρέπει σε ένα δείκτη να πάρει, για παράδειγμα, το τρίτο όρισμα σε μια διαδικασία. Όταν ο μεταγλωττιστής συναντά τη σπάνια εντολή `vararg`, αντιγράφει τους τέσσερις καταχωρητές ορίσματος στη στοίβα, στις τέσσερις δεσμευμένες θέσεις.

Κατανόηση της απόδοσης του προγράμματος

Η Εικόνα 2.37 παρουσιάζει την επίδραση της βελτιστοποίησης του μεταγλωττιστή στην απόδοση (performance), το χρόνο μεταγλώττισης (compile time), τους κύκλους ρολογιού (clock cycles), το πλήθος εντολών (instruction count), και τον αριθμό κύκλων ρολογιού ανά εντολή (Clock cycles Per Instruction — CPI) του προγράμματος ταξινόμησης (sort). Σημειώστε ότι ο μη βελτιστοποιημένος κώδικας έχει τον καλύτερο αριθμό κύκλων ανά εντολή, και η βελτιστοποίηση O1 το χαμηλότερο αριθμό εντολών, ενώ η βελτιστοποίηση O3 οδηγεί στον ταχύτερο κώδικα, υπενθυμίζοντάς μας ότι ο χρόνος είναι το μόνο ακριβές μέτρο της απόδοσης του προγράμματος.

Η Εικόνα 2.38 συγκρίνει την επίδραση των γλωσσών προγραμματισμού, της μεταγλώττισης έναντι της διερμηνείας, και των αλγορίθμων στην απόδοση των ταξινομήσεων. Η τέταρτη στήλη δείχνει ότι το μη βελτιστοποιημένο πρόγραμμα C είναι 8,3 φορές ταχύτερο από το διερμηνευόμενο κώδικα Java για την ταξινόμηση φυσαλίδας (bubble sort). Η χρήση τού «ακριβώς στην ώρα» μεταγλωττιστή Java (Just In Time Java compiler) κάνει τον κώδικα Java 2,1 φορές ταχύτερο από το μη βελτιστοποιημένο κώδικα C, και μέσα σε ένα συντελεστή 1,13 για τον πιο βελτιστοποιημένο κώδικα C. (Η επόμενη ενότητα δίνει περισσότερες λεπτομέρειες για τη διερμηνεία σε σχέση με τη μεταγλώττιση της Java, και τον κώδικα Java και MIPS για την ταξινόμηση φυσαλίδας.) Οι λόγοι δεν είναι τόσο κοντά όσο για τη γρήγορη ταξινόμηση (Quicksort) στη στήλη 5, πιθανόν επειδή είναι δυσκολότερη η απόσβεση του κόστους της μεταγλώττισης κατά το χρόνο εκτέλεσης, με το συντομότερο χρόνο εκτέλεσης. Η τελευταία στήλη παρουσιάζει την επίδραση ενός καλύτερου αλγορίθμου, ο οποίος δίνει τρεις τάξεις μεγέθους αύξηση της απόδοσης για την ταξινόμηση 100.000 στοιχείων. Σε σύγκριση ακόμη και με τη διερμηνευόμενη Java στη στήλη 5 με το μεταγλωττιστή C στο υψηλότερο επίπεδο βελτιστοποίησης στη στήλη 4, η Γρήγορη Ταξινόμηση κερδίζει την Ταξινόμηση Φυσαλίδας κατά έναν παράγοντα 50 (0,05 × 2468 ή 123 αντί 2,41).

Βελτιστοποίηση gcc	Σχετική απόδοση	Κύκλοι ρολογιού (εκατομμύρια)	Αριθμός εντολών (εκατομμύρια)	CPI
καμία	1,00	158.615	114.938	1,38
O1 (μέση)	2,37	66.990	37.470	1,79
O2 (πλήρης)	2,38	66.521	39.993	1,66
O3 (ενσωμάτωση διαδικασίας)	2,41	65.747	44.993	1,46

ΕΙΚΟΝΑ 2.37 Σύγκριση της απόδοσης, του αριθμού εντολών, και του CPI για τη βελτιστοποίηση του μεταγλωττιστή στην ταξινόμηση Φυσαλίδας (Bubble Sort). Το πρόγραμμα ταξινόμησε 100.000 λέξεις, με τυχαίες αρχικές τιμές του πίνακα. Τα προγράμματα εκτελέστηκαν σε επεξεργαστή Pentium 4 με ρυθμό ρολογιού 3,06 GHz και δίαυλο συστήματος (system bus) 533 MHz με 2GB μνήμης PC2100 DDR SDRAM. Ο υπολογιστής χρησιμοποιούσε λειτουργικό σύστημα Linux έκδοσης 2.4.20.

Γλώσσα	Μέθοδος εκτέλεσης	Βελτιστο-ποίηση	Σχετική απόδοση ταξινόμησης Φυσαλίδας	Σχετική απόδοση Γρήγορης ταξινόμησης	Ταχύτητα Γρήγορης ταξινόμησης σε σχέση με ταξινόμηση Φυσαλίδας
C	μεταγλωττιστής	καμία	1,00	1,00	2468
	μεταγλωττιστής	O1	2,37	1,50	1562
	μεταγλωττιστής	O2	2,38	1,50	1555
	μεταγλωττιστής	O3	2,41	1,91	1955
Java	διερμηνευτής	—	0,12	0,05	1050
	μεταγλωττιστής «ακριβώς στην ώρα»	—	2,13	0,29	338

ΕΙΚΟΝΑ 2.38 Απόδοση των δύο αλγορίθμων ταξινόμησης σε C και Java, με τη χρήση διερμηνείας και μεταγλωττιστών βελτιστοποίησης σε σχέση με τη μη βελτιστοποιημένη έκδοση της C. Η τελευταία στήλη δείχνει το πλεονέκτημα στην απόδοση της Γρήγορης ταξινόμησης (Quicksort) σε σχέση με την ταξινόμηση Φυσαλίδας (Bubble Sort) για κάθε γλώσσα και επιλογή εκτέλεσης. Τα προγράμματα αυτά εκτελέστηκαν στο ίδιο σύστημα με αυτά της Εικόνας 2.37. Η εικονική μηχανή Java (JVM) είναι της Sun έκδοση 1.3.1, και ο «ακριβώς στην ώρα» μεταγλωττιστής (JIT) είναι ο Sun Hotspot έκδοση 1.3.1.

Β ΤΟΜΟΣ 2.14 Υλοποίηση μιας αντικειμενοστρεφούς γλώσσας

αντικειμενοστρεφής γλώσσα (object-oriented language) Μια γλώσσα προγραμματισμού που προσανατολίζεται σε αντικείμενα αντί για ενέργειες, ή σε δεδομένα αντί για λογική.

Αυτή η ενότητα απευθύνεται σε αναγνώστες που ενδιαφέρονται να μάθουν με ποιο τρόπο εκτελείται μια **αντικειμενοστρεφής γλώσσα** (objected-oriented language) όπως η Java σε μια αρχιτεκτονική MIPS. Παρουσιάζει τους κώδικες byte Java που χρησιμοποιούνται για τη διερμηνεία, και τον κώδικα του MIPS για την έκδοση Java κάποιων τμημάτων κώδικα C που παρουσιάστηκαν σε προηγούμενες ενότητες, μεταξύ των οποίων και η Ταξινόμηση Φυσαλίδας (Bubble Sort). Το υπόλοιπο αυτής της ενότητας βρίσκεται στο Παράρτημα Ε του δεύτερου τόμου.

2.15 Πίνακες ή δείκτες;

Ένα δύσκολο θέμα για κάθε νέο προγραμματιστή είναι η κατανόηση των δεικτών (pointers). Η σύγκριση κώδικα συμβολικής γλώσσας που χρησιμοποιεί πίνακες και αριθμοδείκτες πινάκων (array indices) με κώδικα συμβολικής γλώσσας που χρησιμοποιεί δείκτες (pointers) προσφέρει βαθύτερη γνώση σχετικά με τους δείκτες. Η ενότητα αυτή παρουσιάζει εκδόσεις σε C και συμβολική γλώσσα του MIPS δύο διαδικασιών καθαρισμού (clear) μιας σειράς λέξεων στη μνήμη: η μια χρησιμοποιεί αριθμοδείκτες πινάκων και η άλλη δείκτες. Η Εικόνα 2.39 παρουσιάζει τις δύο διαδικασίες σε C.

Ο σκοπός αυτής της ενότητας είναι να παρουσιάσει τον τρόπο με τον οποίο οι δείκτες αντιστοιχίζονται σε εντολές του MIPS, και όχι να υποστηρίξει ένα συγκεκριμένο τρόπο προγραμματισμού. Θα δούμε την επίδραση των βελτιστοποιήσεων των σύγχρονων μεταγλωττιστών σε αυτές τις δύο διαδικασίες στο τέλος της ενότητας.

Έκδοση της clear με πίνακες

Ας ξεκινήσουμε από την έκδοση `clear1` με πίνακες, εστιάζοντας στο σώμα του βρόχου και αγνοώντας τον κώδικα σύνδεσης της διαδικασίας. Υποθέτουμε ότι

οι δύο παράμετροι `array` και `size` βρίσκονται στους καταχωρητές `$a0` και `$a1`, και ότι η `i` κατανέμεται στον καταχωρητή `$t0`.

```
clear1(int array[], int size)
{
    int i;
    for (i = 0; i < size; i += 1)
        array[i] = 0;
}

clear2(int *array, int size)
{
    int *p;
    for (p = &array[0]; p < &array[size]; p = p + 1)
        *p = 0;
}
```

ΕΙΚΟΝΑ 2.39 Δύο διαδικασίες της γλώσσας C για το μηδενισμό όλων των στοιχείων ενός πίνακα. Η `clear1` χρησιμοποιεί αριθμοδείκτες (indices), ενώ η `clear2` δείκτες (pointers). Η δεύτερη διαδικασία απαιτεί κάποια περαιτέρω εξήγηση για όσους δεν είναι εξοικειωμένοι με τη C. Η διεύθυνση μιας μεταβλητής σημειώνεται με `&`, και η αναφορά στο αντικείμενο που δείχνει ένας δείκτης με `*`. Οι δηλώσεις δηλώνουν ότι τα `array` και `p` είναι δείκτες προς ακέραιους (integers). Το πρώτο μέρος του βρόχου *for* στη διαδικασία `clear2` αναθέτει τη διεύθυνση του πρώτου στοιχείου του πίνακα `array` στο δείκτη `p`. Το δεύτερο μέρος του βρόχου *for* ελέγχει αν ο δείκτης δείχνει πέρα από το τελευταίο στοιχείο του `array`. Η αύξηση ενός δείκτη κατά ένα, στο τελευταίο μέρος του βρόχου *for*, σημαίνει μετακίνηση του δείκτη στο επόμενο αντικείμενο στη σειρά του δηλωμένου μεγέθους. Εφόσον ο `p` είναι ένας δείκτης προς ακεραίους, ο μεταγλωττιστής θα παραγάγει εντολές του MIPS για την αύξηση του `p` κατά τέσσερα, τον αριθμό των byte που περιέχονται σε έναν ακέραιο του MIPS. Η ανάθεση τιμής στο βρόχο τοποθετεί το 0 στο αντικείμενο που δείχνει ο `p`.

Η ανάθεση αρχικών τιμών στο `i`, το πρώτο μέρος του βρόχου *for*, είναι απλή:

```
        move    $t0,$zero     # i = 0 (καταχωρητής $t0 = 0)
```

Για να δώσουμε στο στοιχείο `array[i]` την τιμή 0, πρέπει πρώτα να πάρουμε τη διεύθυνσή του. Καταρχήν πολλαπλασιάζουμε το `i` με το 4 για να πάρουμε τη διεύθυνση byte:

```
loop1:sll     $t1,$t0,2     # $t1 = i * 4
```

Αφού η διεύθυνση αρχής του πίνακα βρίσκεται σε έναν καταχωρητή, πρέπει να την προσθέσουμε στον αριθμοδείκτη ώστε να πάρουμε τη διεύθυνση του στοιχείου `array[i]`, χρησιμοποιώντας μια εντολή πρόσθεσης:

```
        add     $t2,$a0,$t1   # $t2 = διεύθυνση του
                              # στοιχείου array[i]
```

(Αυτό το παράδειγμα είναι μια ιδανική κατάσταση διευθυνσιοδότησης με αριθμοδείκτες — indexed addressing· δείτε την ενότητα Σε μεγαλύτερο βάθος της Ενότητας 2.20 στη σελίδα 166.) Τέλος, σε αυτή τη διεύθυνση μπορούμε να αποθηκεύσουμε το 0:

```
        sw      $zero,0($t2)  # array[i] = 0
```

Αυτή η εντολή είναι το τέλος του σώματος του βρόχου, και έτσι το επόμενο βήμα είναι η αύξηση του `i`:

```
    addi $t0,$t0,1        # i = i + 1
```

Ο έλεγχος του βρόχου ελέγχει αν το i είναι μικρότερο από το size:

```
        slt  $t3,$t0,$a1      # $t3 = (i < size)
        bne  $t3,$zero,loop1  # μετάβαση στη loop1 αν (i < size)
```

Έχουμε τώρα δει όλα τα μέρη της διαδικασίας. Ακολουθεί ο κώδικας του MIPS για το μηδενισμό ενός πίνακα με αριθμοδείκτες:

```
        move $t0,$zero        # i = 0
loop1:sll  $t1,$t0,2          # $t1 = i * 4
        add  $t2,$a0,$t1      # $t2 = διεύθυνση του
                              # στοιχείου array[i]
        sw   $zero,0($t2)     # array[i] = 0
        addi $t0,$t0,1        # i = i + 1
        slt  $t3,$t0,$a1      # $t3 = (i < size)
        bne  $t3,$zero,loop1  # μετάβαση στη loop1 αν (i < size)
```

(Αυτός ο κώδικας δουλεύει εφόσον το μέγεθος — size — είναι μεγαλύτερο από 0.)

Έκδοση της clear με δείκτες

Η δεύτερη διαδικασία, που χρησιμοποιεί δείκτες, κατανέμει τις δύο παραμέτρους array και size στους καταχωρητές $a0 και $a1 και το p στον καταχωρητή $t0. Ο κώδικας για τη δεύτερη διαδικασία ξεκινάει με την ανάθεση του p στη διεύθυνση του πρώτου στοιχείου του πίνακα:

```
        move $t0,$a0          # p = διεύθυνση του στοιχείου
                              # array[0]
```

Ο κώδικας που ακολουθεί είναι το σώμα του βρόχου *for*, ο οποίος απλώς αποθηκεύει την τιμή 0 στο δείκτη p:

```
loop2:sw   $zero,0($t0)       # Memory[p] = 0
```

Η εντολή αυτή υλοποιεί το σώμα του βρόχου, ώστε ο επόμενος κώδικας να είναι η αύξηση του μετρητή επανάληψης, η οποία αλλάζει το δείκτη p ώστε να δείχνει στην επόμενη λέξη:

```
        addi $t0,$t0,4        # p = p + 4
```

Η αύξηση ενός δείκτη κατά 1 σημαίνει τη μετακίνησή του στο επόμενο αντικείμενο στη σειρά στη C. Εφόσον ο p είναι ένας δείκτης προς ακεραίους, καθένας από τους οποίους χρησιμοποιεί τέσσερα byte, ο μεταγλωττιστής αυξάνει τον p κατά 4.

Ο έλεγχος του βρόχου είναι το επόμενο τμήμα κώδικα. Το πρώτο βήμα είναι ο υπολογισμός της διεύθυνσης του τελευταίου στοιχείου του πίνακα array. Αρχίζουμε με τον πολλαπλασιασμό της size με το 4 για να πάρουμε τη διεύθυνση byte του:

```
        add  $t1,$a1,$a1      # $t1 = size * 2
        add  $t1,$t1,$t1      # $t1 = size * 4
```

και μετά προσθέτουμε το γινόμενο στη διεύθυνση αρχής του πίνακα array για να πάρουμε τη διεύθυνση της πρώτης λέξης *μετά* τον πίνακα:

```
        add  $t2,$a0,$t1      # $t2 = διεύθυνση του
                              # στοιχείου array[size]
```

Ο έλεγχος του βρόχου γίνεται απλώς για να διαπιστωθεί αν ο δείκτης p είναι μικρότερος από τη διεύθυνση του τελευταίου στοιχείου του πίνακα.

```
        slt    $t3,$t0,$t2       # $t3 = (p < &array[size])
        bne    $t3,$zero,loop2   # μετάβαση στη loop2 αν
                                 # (p < &array[size])
```

Με όλα τα κομμάτια συμπληρωμένα, μπορούμε να παρουσιάσουμε την έκδοση με δείκτες του κώδικα για το μηδενισμό ενός πίνακα.

```
            move  $t0,$a0          # p = διεύθυνση του στοιχείου
                                   # array[0]
      loop2:sw    $zero,0($t0)     # Memory[p] = 0
            addi  $t0,$t0,4        # p = p + 4
            add   $t1,$a1,$a1      # $t1 = size * 2
            add   $t1,$t1,$t1      # $t1 = size * 4
            add   $t2,$a0,$t1      # $t2 = διεύθυνση του στοιχείου
                                   # array[size]
            slt   $t3,$t0,$t2      # $t3 = (p < &array[size])
            bne   $t3,$zero,loop2  # μετάβαση στη loop2 αν
                                   # (p < &array[size])
```

Όπως στο πρώτο παράδειγμα, ο κώδικας αυτός υποθέτει ότι η τιμή του μεγέθους size είναι μεγαλύτερη από 0.

Σημειώστε ότι το πρόγραμμα αυτό υπολογίζει τη διεύθυνση του τέλους του πίνακα array σε κάθε επανάληψη του βρόχου, παρόλο που αυτή δεν αλλάζει. Μια ταχύτερη έκδοση του κώδικα μεταφέρει αυτόν τον υπολογισμό έξω από το βρόχο:

```
            move  $t0,$a0          # p = διεύθυνση του στοιχείου
                                   # array[0]
            sll   $t1,$a1,2        # $t1 = size * 4
            add   $t2,$a0,$t1      # $t2 = διεύθυνση του στοιχείου
                                   # array[size]
      loop2:sw    $zero,0($t0)     # Memory[p] = 0
            addi  $t0,$t0,4        # p = p + 4
            slt   $t3,$t0,$t2      # $t3 = (p < &array[size])
            bne   $t3,$zero,loop2  # μετάβαση στη loop2 αν
                                   # (p < &array[size])
```

Σύγκριση των δύο εκδόσεων της clear

Η σύγκριση των δύο ακολουθιών κώδικα δίπλα-δίπλα παρουσιάζει τη διαφορά μεταξύ των αριθμοδεικτών πινάκων και των δεικτών (οι αλλαγές οι οποίες εισάγονται από την έκδοση που βασίζεται σε δείκτες είναι επισημασμένες).

```
      move $t0,$zero       # i = 0
loop1:sll  $t1,$t0,2       # $t1 = i * 4
      add  $t2,$a0,$t1     # $t2 = # &array[i]
      sw   $zero,0($t2)    # array[i] = 0
      addi $t0,$t0,1       # i = i + 1
      slt  $t3,$t0,$a1     # $t3 = # (i < size)
      bne  $t3,$zero,loop1 # μετάβαση
                           # loop1
```

```
      move $t0,$a0         # p = &array[0]
      sll  $t1,$a1,2       # $t1 = size * 4
      add  $t2,$a0,$t1     # $t2 = &array[size]
loop2:sw   $zero,0($t0)    # Memory[p] = 0
      addi $t0,$t0,4       # p = p + 4
      slt  $t3,$t0,$t2     # $t3 =
                           # (p < &array[size])
      bne  $t3,$zero,loop2 # μετάβαση loop2
```

Η έκδοση αριστερά πρέπει να έχει τον «πολλαπλασιασμό» και την άθροιση μέσα στο βρόχο, επειδή το `i` αυξάνεται και κάθε διεύθυνση πρέπει να επανυπολογίζεται από το νέο αριθμοδείκτη· η έκδοση δεξιά, η οποία χρησιμοποιεί δείκτη στη μνήμη, αυξάνει απευθείας τον `p`. Η έκδοση με δείκτες μειώνει τις εκτελούμενες εντολές ανά επανάληψη, από 7 σε 4. Αυτή η βελτιστοποίηση «με το χέρι» αντιστοιχεί στη βελτιστοποίηση μείωσης της δύναμης (ολίσθηση αντί πολλαπλασιασμού) και στην εξάλειψη επαγωγικής μεταβλητής (κατάργηση των υπολογισμών διευθύνσεων πινάκων στο εσωτερικό βρόχων) του μεταγλωττιστή.

Επιπλέον ανάπτυξη: Ο μεταγλωττιστής της C θα πρόσθετε έναν έλεγχο για να εξασφαλίσει ότι το μέγεθος `size` είναι μεγαλύτερο από 0. Ένας τρόπος θα ήταν η προσθήκη μιας εντολής άλματος (jump) ακριβώς πριν από την πρώτη εντολή του βρόχου στην εντολή `slt`.

Κατανόηση της απόδοσης του προγράμματος

Οι άνθρωποι συνήθως διδάσκονται να χρησιμοποιούν δείκτες στη C ώστε να παίρνουν καλύτερη απόδοση από αυτή που είναι δυνατή με τους πίνακες: «Χρησιμοποιήστε δείκτες, ακόμη και αν δεν μπορείτε να καταλάβετε τον κώδικα». Οι σύγχρονοι μεταγλωττιστές βελτιστοποίησης μπορούν να παραγάγουν εξίσου καλό κώδικα συμβολικής γλώσσας και για κώδικα που χρησιμοποιεί πίνακες. Οι περισσότεροι προγραμματιστές σήμερα προτιμούν να αφήνουν τη βαριά δουλειά για το μεταγλωττιστή.

Η ομορφιά βρίσκεται εξολοκλήρου στο μάτι αυτού που κοιτάζει.
Margaret Wolfe Hungerford, *Molly Bawn,* 1877

2.16 Πραγματικότητα: εντολές τής IA-32

Οι σχεδιαστές συνόλων εντολών μερικές φορές παρέχουν ισχυρότερες λειτουργίες από αυτές που υπάρχουν στο MIPS. Ο στόχος είναι γενικά η μείωση του αριθμού των εκτελούμενων εντολών από το πρόγραμμα. Ο κίνδυνος είναι αυτή η μείωση να πραγματοποιηθεί σε βάρος της απλότητας, αυξάνοντας το χρόνο που χρειάζεται για την εκτέλεση ενός προγράμματος επειδή οι εντολές είναι πιο αργές. Αυτός ο μεγαλύτερος χρόνος εκτέλεσης μπορεί να είναι το αποτέλεσμα ενός μεγαλύτερου κύκλου ρολογιού ή της απαίτησης περισσότερων κύκλων ρολογιού σε σχέση με μια απλούστερη ακολουθία (δείτε την Ενότητα 4.8).

Ο δρόμος προς την πολυπλοκότητα των λειτουργιών ενέχει επομένως πολλούς κινδύνους. Για την αποφυγή αυτών των προβλημάτων, οι σχεδιαστές κινήθηκαν προς απλούστερες εντολές. Η Ενότητα 2.17 παρουσιάζει τις παγίδες της πολυπλοκότητας.

Η αρχιτεκτονική Intel IA-32

Ο MIPS ήταν το όραμα μιας μικρής ομάδας το 1985· τα κομμάτια αυτής της αρχιτεκτονικής ταιριάζουν ωραία μεταξύ τους, και όλη η αρχιτεκτονική μπορεί να περιγραφεί περιληπτικά. Αυτό δεν ισχύει για την αρχιτεκτονική IA-32· είναι το προϊόν διαφορετικών ανεξάρτητων ομάδων οι οποίες την εξέλισσαν για περίπου 20 χρόνια, προσθέτοντας νέες δυνατότητες στο αρχικό σύνολο εντολών όπως κάποιος θα πρόσθετε ρούχα σε μια γεμάτη βαλίτσα. Τα σημαντικά ορόσημα της αρχιτεκτονικής IA-32 είναι τα εξής:

- **1978:** Ανακοινώνεται η αρχιτεκτονική του επεξεργαστή Intel 8086 ως μια επέκταση συμβατή με τη συμβολική γλώσσα τού τότε επιτυχημένου Intel 8080, ενός μικροεπεξεργαστή 8 bit. Ο 8086 έχει αρχιτεκτονική 16 bit, με όλους τους εσωτερικούς καταχωρητές του να διαθέτουν εύρος 16 bit. Σε αντίθεση με το MIPS, οι καταχωρητές έχουν αποκλειστικές χρήσεις και, επομένως, ο 8086 δε θεωρείται μια αρχιτεκτονική **γενικών καταχωρητών** ή **καταχωρητών γενικού σκοπού** (general purpose register).

γενικός καταχωρητής ή **καταχωρητής γενικού σκοπού** (general purpose register — GPR) Ένας καταχωρητής που μπορεί να χρησιμοποιηθεί για διευθύνσεις ή δεδομένα σχεδόν με οποιαδήποτε εντολή.

- **1980:** Ανακοινώνεται ο συνεπεξεργαστής κινητής υποδιαστολής (floating point coprocessor) Intel 8087. Αυτή η αρχιτεκτονική επεκτείνει τον 8086 με περίπου 60 εντολές κινητής υποδιαστολής. Αντί για τη χρήση καταχωρητών, βασίζεται σε στοίβα (δείτε τις Ενότητες 2.19 και 3.9).
- **1982:** Ο 80286 επέκτεινε την αρχιτεκτονική του 8086 αυξάνοντας το χώρο διευθύνσεων στα 24 bit και δημιουργώντας ένα λεπτομερές μοντέλο χαρτογράφησης μνήμης και προστασίας (δείτε το Κεφάλαιο 7), και προσθέτοντας μερικές εντολές για τη συμπλήρωση του συνόλου εντολών και το χειρισμό του μοντέλου προστασίας.
- **1985:** Ο 80386 επέκτεινε την αρχιτεκτονική του 80286 στα 32 bit. Επιπλέον μιας αρχιτεκτονικής 32 bit με καταχωρητές των 32 bit και χώρο διευθύνσεων 32 bit, ο 80386 πρόσθεσε νέους τρόπους διευθυνσιοδότησης και επιπλέον λειτουργίες. Οι εντολές που προστέθηκαν κάνουν τον 80386 σχεδόν μια μηχανή καταχωρητών γενικού σκοπού. Η αρχιτεκτονική του 80386 πρόσθεσε επίσης υποστήριξη σελιδοποίησης (paging) εκτός από την κατατμημένη διευθυνσιοδότηση (segmented addressing — δείτε το Κεφάλαιο 7). Όπως ο 80286, έτσι και ο 80386 διαθέτει μια κατάσταση λειτουργίας για την εκτέλεση προγραμμάτων του 8086 χωρίς αλλαγές.
- **1989-95:** Οι επόμενοι, 80486 το 1989, Pentium το 1992, και Pentium Pro το 1995 είχαν στόχο την υψηλότερη απόδοση, με τέσσερις μόνο επιπλέον εντολές στο ορατό στο χρήστη σύνολο εντολών, που είναι: τρεις βοηθητικές για την πολυεπεξεργασία (multiprocessing — Κεφάλαιο 9) και μία εντολή μετακίνησης υπό συνθήκη (conditional move).
- **1997:** Μετά τη διάθεση των επεξεργαστών Pentium και Pentium Pro, η Intel ανακοίνωσε ότι θα επέκτεινε τις αρχιτεκτονικές τους με Επεκτάσεις Πολυμέσων (MMX — Multimedia Extensions). Αυτό το νέο σύνολο 57 εντολών χρησιμοποιεί τη στοίβα κινητής υποδιαστολής για την επιτάχυνση εφαρμογών πολυμέσων και επικοινωνιών. Οι εντολές MMX συνήθως λειτουργούν σε πολλά μικρού μήκους στοιχεία δεδομένων κάθε χρονική στιγμή, με τον τρόπο των αρχιτεκτονικών μίας εντολής, πολλών δεδομένων (Single Instruction Multiple Data — SIMD — δείτε το Κεφάλαιο 9). Ο Pentium II δεν εισήγαγε νέες εντολές.
- **1999:** Η Intel πρόσθεσε άλλες 70 εντολές, με το όνομα Επεκτάσεις Συνεχούς Ροής SIMD (Streaming SIMD Extensions — SSE) ως τμήμα της αρχιτεκτονικής του επεξεργαστή Pentium III. Οι κύριες αλλαγές ήταν η προσθήκη οκτώ ξεχωριστών καταχωρητών, ο διπλασιασμός του εύρους τους στα 128 bit, και η προσθήκη ενός τύπου δεδομένων κινητής υποδιαστολής απλής ακρίβειας (single precision floating-point). Έτσι, μπορούν να πραγματοποιούνται παράλληλα τέσσερις πράξεις κινητής υποδιαστολής 32 bit. Για τη βελτίωση της απόδοσης της μνήμης, οι SSE περιλάμβαναν εντολές εκ των προτέρων προσκόμισης της κρυφής μνήμης (cache

prefetch) και εντολές συνεχούς ροής αποθήκευσης που παρακάμπτουν τις κρυφές μνήμες και γράφουν απευθείας στη μνήμη.

- **2001:** Η Intel πρόσθεσε άλλες 144 εντολές, αυτή τη φορά με το όνομα SSE2. Ο νέος τύπος δεδομένων είναι αριθμητικής διπλής ακρίβειας, και επιτρέπει την εκτέλεση ζευγών πράξεων κινητής υποδιαστολής 64 bit παράλληλα. Σχεδόν όλες αυτές οι 144 εντολές είναι εκδόσεις των εντολών MMX και SSE οι οποίες λειτουργούν σε 64 bit παράλληλα. Η αλλαγή αυτή, όχι μόνον επιτρέπει περισσότερες λειτουργίες πολυμέσων, αλλά δίνει στο μεταγλωττιστή ένα διαφορετικό στόχο για λειτουργίες κινητής υποδιαστολής από την αρχιτεκτονική μοναδικής στοίβας. Οι μεταγλωττιστές μπορούν να επιλέξουν να χρησιμοποιήσουν τους οκτώ καταχωρητές SSE σαν καταχωρητές κινητής υποδιαστολής όπως εκείνοι που υπάρχουν σε άλλους υπολογιστές. Αυτή η αλλαγή εκτόξευσε στα ύψη τις επιδόσεις των πράξεων κινητής υποδιαστολής στον επεξεργαστή Pentium 4, τον πρώτο μικροεπεξεργαστή που περιλάμβανε εντολές SSE2.
- **2003:** Αυτή τη φορά, ήταν μια άλλη εταιρεία εκτός της Intel που ενίσχυσε την αρχιτεκτονική IA-32. Η AMD ανακοίνωσε ένα σύνολο αρχιτεκτονικών επεκτάσεων για την αύξηση του χώρου διευθύνσεων, από 32 σε 64 bit. Όπως κατά τη μετάβαση από ένα χώρο διευθύνσεων 16 bit σε έναν 32 bit το 1985 με τον 80386, η αρχιτεκτονική AMD64 διευρύνει όλους τους καταχωρητές στα 64 bit. Επίσης, αυξάνει τον αριθμό των καταχωρητών σε 16 και τον αριθμό των καταχωρητών SSE των 128 bit σε 16. Η βασική αλλαγή της αρχιτεκτονικής συνόλου εντολών είναι η προσθήκη ενός νέου τρόπου (ή κατάστασης) λειτουργίας που ονομάζεται *εκτενής κατάσταση λειτουργίας* (long mode) και επαναπροσδιορίζει την εκτέλεση όλων των εντολών IA-32 με διευθύνσεις και δεδομένα 64 bit. Για τη διευθυνσιοδότηση του μεγαλύτερου αριθμού καταχωρητών, προστίθεται ένα νέο πρόθεμα στις εντολές. Ανάλογα με το πώς μετράτε, η εκτενής κατάσταση λειτουργίας προσθέτει επίσης 4 έως 10 νέες εντολές και εγκαταλείπει 27 παλιές. Η σχετική ως προς PC (PC-relative) διευθυνσιοδότηση δεδομένων είναι άλλη μία επέκταση. Ο AMD64 έχει ακόμα μία κατάσταση λειτουργίας που είναι όμοια με την αρχιτεκτονική IA-32 (*κληρονομημένη κατάσταση λειτουργίας* — legacy mode) και μια κατάσταση που περιορίζει τα προγράμματα χρήστη στην αρχιτεκτονική IA-32 αλλά επιτρέπει σε λειτουργικά συστήματα να χρησιμοποιούν την αρχιτεκτονική AMD64 (*κατάσταση συμβατότητας* — compatibility mode). Αυτές οι καταστάσεις λειτουργίας επιτρέπουν μια πιο ομαλή μετάβαση σε διευθυνσιοδότηση 64 bit από ό,τι η αρχιτεκτονική IA-64 της HP/Intel.
- **2004:** Η Intel συνθηκολογεί και αγκαλιάζει την αρχιτεκτονική AMD64, μετονομάζοντας τη σε Τεχνολογία Επεκτεταμένης Μνήμης 64 (Extended Memory 64 Technology — EM64T). Η κύρια διαφορά είναι ότι η Intel πρόσθεσε μια ατομική εντολή σύγκρισης και αντιμετάθεσης, η οποία μάλλον θα έπρεπε να είχε συμπεριληφθεί στην αρχιτεκτονική του AMD64. Την ίδια στιγμή, η Intel ανακοίνωνε άλλη μία γενιά επεκτάσεων μέσων (media extensions). Η SSE3 προσθέτει 13 εντολές για την υποστήριξη πολύπλοκων αριθμητικών πράξεων, λειτουργιών γραφικών σε πίνακες δομών, κωδικοποίησης βίντεο, μετατροπής κινητής υποδιαστολής, και συγχρονισμού νημάτων (thread synchronization — δείτε το Κεφάλαιο 9). Η AMD θα προσφέρει αρχιτεκτονική SSE3 σε επόμενα ολοκληρωμένα κυκλώματα και σχεδόν σίγουρα θα προσθέσει την ατομική

εντολή αντιμετάθεσης που λείπει από την αρχιτεκτονική του AMD64, ώστε να διατηρήσει τη δυαδική συμβατότητα με την Intel.

Αυτή η ιστορία δείχνει την επίδραση των «χρυσών χειροπέδων» της συμβατότητας στην αρχιτεκτονική IA-32, αφού η υπάρχουσα βάση λογισμικού σε κάθε βήμα ήταν πολύ σημαντική για να τεθεί σε κίνδυνο με σημαντικές αλλαγές αρχιτεκτονικής.

Ανεξάρτητα από τις καλλιτεχνικές αποτυχίες της αρχιτεκτονικής IA-32, κρατήστε στο μυαλό σας ότι υπάρχουν πολύ περισσότερες εκδόσεις αυτής της οικογένειας στους επιτραπέζιους υπολογιστές από οποιαδήποτε άλλη αρχιτεκτονική, με μια αύξηση κατά 100 εκατομμύρια το χρόνο. Παρόλα αυτά, αυτή η κατακερματισμένη γενεαλογία οδήγησε σε μια αρχιτεκτονική που είναι δύσκολο να εξηγηθεί και αδύνατον να αγαπηθεί.

Καθίστε καλά γι' αυτό που πρόκειται να δείτε! *Μην* προσπαθήσετε να διαβάσετε αυτή την ενότητα με την προσοχή που θα χρειαζόταν για να γράψετε προγράμματα για IA-32· αντίθετα, στόχος μας είναι η εξοικείωσή σας με τις δυνατότητες και τις αδυναμίες της δημοφιλέστερης αρχιτεκτονικής επιτραπέζιων υπολογιστών του κόσμου.

Αντί να παρουσιάσουμε το πλήρες σύνολο εντολών 16 και 32 bit, σε αυτή την ενότητα επικεντρωνόμαστε στο υποσύνολο των 32 bit το οποίο ξεκίνησε με τον 80386, αφού αυτό το τμήμα της αρχιτεκτονικής είναι αυτό που χρησιμοποιείται. Αρχίζουμε με τους καταχωρητές και τους τρόπους διευθυνσιοδότησης, προχωρούμε στις πράξεις μεταξύ ακεραίων, και καταλήγουμε με μια εξέταση της κωδικοποίησης των εντολών.

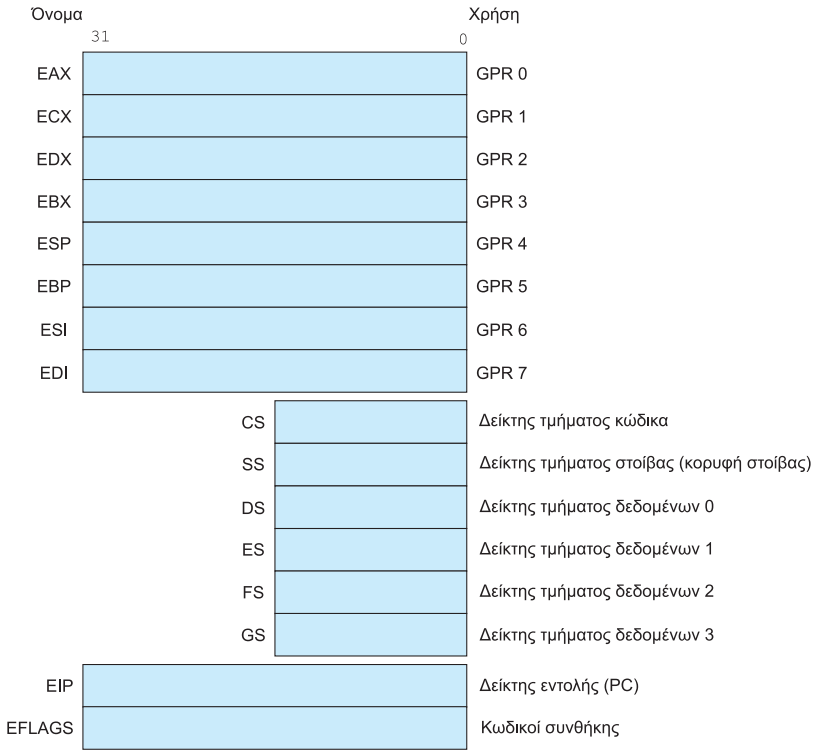
Καταχωρητές και τρόποι διευθυνσιοδότησης δεδομένων της IA-32

Οι καταχωρητές του 80386 δείχνουν την εξέλιξη του συνόλου εντολών (Εικόνα 2.40). Ο 80386 επέκτεινε όλους τους καταχωρητές των 16 bit (εκτός από τους καταχωρητές τμήματος — segment registers) στα 32 bit, με πρόθεμα το *E* στο όνομά τους ως ένδειξη της έκδοσης των 32 bit. Θα τους αποκαλέσουμε καταχωρητές γενικού σκοπού (general purpose registers — GPR). Ο 80386 περιέχει μόνο 8 GPR. Αυτό σημαίνει ότι τα προγράμματα του MIPS μπορούν να χρησιμοποιήσουν τετραπλάσιους.

Οι αριθμητικές και οι λογικές εντολές και οι εντολές μεταφοράς δεδομένων είναι εντολές δύο τελεστέων, οι οποίες επιτρέπουν τους συνδυασμούς που παρουσιάζονται στην Εικόνα 2.41. Υπάρχουν δύο σημαντικές διαφορές εδώ. Οι αριθμητικές και λογικές εντολές της αρχιτεκτονικής IA-32 πρέπει να έχουν έναν τελεστέο που ενεργεί τόσο ως προέλευση όσο και ως προορισμός· ο MIPS επιτρέπει ξεχωριστούς καταχωρητές για προέλευση και προορισμό. Αυτός ο περιορισμός ασκεί μεγαλύτερη πίεση στους περιορισμένους καταχωρητές, εφόσον ένας καταχωρητής προέλευσης πρέπει να τροποποιηθεί. Η δεύτερη σημαντική διαφορά είναι ότι ένας από τους τελεστέους μπορεί να βρίσκεται στη μνήμη. Με αυτόν τον τρόπο, ουσιαστικά κάθε εντολή μπορεί να έχει τον έναν τελεστέο στη μνήμη, σε αντίθεση με τους επεξεργαστές MIPS και PowerPC.

Οι επτά τρόποι διευθυνσιοδότησης δεδομένων στη μνήμη, που περιγράφονται λεπτομερώς παρακάτω, διαθέτουν δύο μεγέθη διευθύνσεων στο εσωτερικό της εντολής. Αυτές οι λεγόμενες *μετατοπίσεις* (displacements) μπορεί να είναι 8 ή 32 bit.

Αν και ένας τελεστέος μνήμης μπορεί να χρησιμοποιεί οποιονδήποτε τρόπο διευθυνσιοδότησης, υπάρχουν περιορισμοί σχετικά με το ποιοι *καταχωρητές*



ΕΙΚΟΝΑ 2.40 Το σύνολο καταχωρητών του 80386. Με αρχή τον 80386, οι οκτώ επάνω καταχωρητές επεκτάθηκαν στα 32 bit και μπορούσαν επίσης να χρησιμοποιηθούν ως καταχωρητές γενικού σκοπού.

Τύπος τελεστέου προέλευσης/προορισμού	Δεύτερος τελεστέος προέλευσης
Καταχωρητής	Καταχωρητής
Καταχωρητής	Άμεσος
Καταχωρητής	Μνήμη
Μνήμη	Καταχωρητής
Μνήμη	Άμεσος

ΕΙΚΟΝΑ 2.41 Τύποι εντολών για τις αριθμητικές και λογικές εντολές και τις εντολές μεταφοράς δεδομένων. Η αρχιτεκτονική IA-32 επιτρέπει τους συνδυασμούς που βλέπετε εδώ. Ο μόνος περιορισμός είναι η απουσία τρόπου διευθυνσιοδότησης από μνήμη σε μνήμη. Οι άμεσοι τελεστέοι μπορεί να έχουν μήκος 8, 16, ή 32 bit· ο καταχωρητής μπορεί να είναι ένας από τους 14 κύριους καταχωρητές της Εικόνας 2.40 (όχι ο EIP ή ο EFLAGS).

μπορούν να χρησιμοποιηθούν σε κάθε κατάσταση. Η Εικόνα 2.42 δείχνει τους τρόπους διευθυνσιοδότησης της αρχιτεκτονικής IA-32 και ποιοι καταχωρητές γενικού σκοπού δεν μπορούν να χρησιμοποιηθούν με κάθε τρόπο, καθώς και πώς θα παίρνατε το ίδιο αποτέλεσμα με τη χρήση εντολών του MIPS.

Πράξεις ακεραίων της αρχιτεκτονικής IA-32

Ο 8086 παρέχει υποστήριξη για τύπους δεδομένων τόσο 8 bit (*byte*) όσο και 16 bit (*λέξη* — word). Ο 80386 προσθέτει διευθύνσεις και δεδομένα 32 bit (*διπλές λέξεις* — double words) στην αρχιτεκτονική IA-32. Οι διακρίσεις στους τύπους των δεδομένων ισχύουν τόσο για τις πράξεις με καταχωρητές όσο και για τις προσπελάσεις μνήμης. Σχεδόν κάθε πράξη (λειτουργία) δουλεύει και σε δεδομένα 8 bit και σε μεγαλύτερο μέγεθος δεδομένων. Το μέγεθος καθορίζεται από την κατάσταση λειτουργίας και είναι είτε 16 bit είτε 32 bit.

Τρόπος	Περιγραφή	Περιορισμοί καταχωρητών	Ισοδύναμο στο MIPS
Έμμεσος μέσω καταχωρητή	Η διεύθυνση είναι σε έναν καταχωρητή	εκτός ESP και EBP	`lw $s0,0($s1)`
Βάσης με μετατόπιση 8 ή 32 bit	Η διεύθυνση είναι τα περιεχόμενα του καταχωρητή βάσης συν τη μετατόπιση	εκτός ESP και EBP	`lw $s0,100($s1) # ≤16-bit # μετατόπιση`

Βάσης με κλιμακούμενο αριθμοδείκτη	Η διεύθυνση είναι Βάση + ($2^{\text{Κλίμακα}}$ × Αριθμοδείκτης) Κλίμακα=0, 1, 2, ή 3.	Βάση: οποιοσδήποτε GPR Αριθμοδείκτης: όχι ESP	`mul $t0,$s2, 4`
			`add $t0,$t0,$s1`
			`lw $s0,0($t0)`

Βάσης με κλιμακούμενο αριθμοδείκτη και μετατόπιση 8 ή 32 bit	Η διεύθυνση είναι Βάση + ($2^{\text{Κλίμακα}}$ × Αριθμοδείκτης) + μετατόπιση Κλίμακα=0, 1, 2, ή 3.	Βάση: οποιοσδήποτε GPR Αριθμοδείκτης: όχι ESP	`mul $t0,$s2, 4`
			`add $t0,$t0,$s1`
			`lw $s0,100($t0) # ≤16-bit # μετατόπιση`

ΕΙΚΟΝΑ 2.42 Τρόποι διευθυνσιοδότησης της αρχιτεκτονικής IA-32 των 32 bit με περιορισμούς καταχωρητών και ισοδύναμο κώδικα MIPS. Ο τρόπος διευθυνσιοδότησης Βάσης με κλιμακούμενο αριθμοδείκτη (Base plus Scaled Index) ο οποίος δε συναντάται στους επεξεργαστές MIPS και PowerPC, περιλαμβάνεται για την αποφυγή πολλαπλασιασμών με το τέσσερα (συντελεστής κλίμακας 2) κατά τη μετατροπή ενός αριθμοδείκτη καταχωρητή σε διεύθυνση byte (δείτε τις Εικόνες 2.34 και 2.36). Ένας συντελεστής κλίμακας 1 χρησιμοποιείται για δεδομένα 16 bit, και ένας συντελεστής κλίμακας 3 χρησιμοποιείται για δεδομένα 64 bit. Ένας συντελεστής κλίμακας 0 σημαίνει ότι η διεύθυνση δεν αλλάζει μέγεθος. Αν η μετατόπιση είναι μεγαλύτερη από 16 bit στο δεύτερο ή τέταρτο τρόπο, η ισοδύναμη κατάσταση λειτουργίας του MIPS θα απαιτούσε δύο επιπλέον εντολές: μια `lui` για τη φόρτωση των ανώτερων 16 bit της μετατόπισης και μια `add` για την άθροιση του ανώτερου τμήματος της διεύθυνσης με τον καταχωρητή βάσης `$s1`. Η Intel δίνει δύο διαφορετικά ονόματα σε αυτό που ονομάζεται τρόπος διευθυνσιοδότησης βάσης — Based addressing mode: Βάσης (Based) και Αριθμοδείκτη (Indexed) — αλλά είναι ουσιαστικά όμοιες και εδώ τις συνδυάζουμε.)

Σαφώς κάποια προγράμματα θέλουν να χειρίζονται δεδομένα και των τριών μεγεθών, και έτσι οι αρχιτέκτονες του 80386 παρέχουν ένα βολικό τρόπο καθορισμού κάθε έκδοσης χωρίς σημαντική επέκταση του μεγέθους του κώδικα. Αποφάσισαν ότι στα περισσότερα προγράμματα κυριαρχούν δεδομένα είτε των 16 είτε των 32 bit, και έτσι είναι λογικό να υπάρχει δυνατότητα καθορισμού ενός μεγάλου προεπιλεγμένου (default) μεγέθους. Αυτό το προεπιλεγμένο μέγεθος δεδομένων επιλέγεται από ένα bit στον καταχωρητή τμήματος κώδικα (code segment register). Για να παρακάμψουμε το προεπιλεγμένο μέγεθος δεδομένων, προσαρτάται στην εντολή ένα πρόθεμα 8 bit που πληροφορεί τη μηχανή να χρησιμοποιήσει το άλλο μεγάλο μέγεθος γι' αυτή την εντολή.

Η λύση του προθέματος είναι δανεισμένη από τον 8086, ο οποίος επιτρέπει πολλά προθέματα για την τροποποίηση της συμπεριφοράς των εντολών. Τα τρία αρχικά προθέματα παρακάμπτουν τον προεπιλεγμένο καταχωρητή τμήματος, κλειδώνουν το δίαυλο για την υποστήριξη ενός σηματοφόρου (semaphore — δείτε το Κεφάλαιο 9), ή επαναλαμβάνουν την εντολή που ακολουθεί μέχρι ο καταχωρητής ECX να καταλήξει στο 0. Γι' αυτό το τελευταίο πρόθεμα, υπήρχε η πρόθεση να συνοδεύει μια εντολή μετακίνησης byte για τη μετακίνηση μεταβλητού αριθμού byte. Ο 80386 επίσης πρόσθεσε ένα πρόθεμα για την παράκαμψη του προεπιλεγμένου μεγέθους διεύθυνσης.

Οι πράξεις ακεραίων της αρχιτεκτονικής IA-32 μπορούν να χωριστούν σε τέσσερις κύριες κατηγορίες:

1. Εντολές μετακίνησης δεδομένων, που περιλαμβάνουν τις move, push, και pop
2. Αριθμητικές και λογικές εντολές, που περιλαμβάνουν λειτουργίες ελέγχου, και αριθμητικές πράξεις με ακέραιους και δεκαδικούς

3. Εντολές ελέγχου ροής (control flow), μεταξύ των οποίων διακλαδώσεις υπό συνθήκη (conditional branches), άλματα χωρίς συνθήκη (unconditional jumps), κλήσεις (calls), και επιστροφές (returns)
4. Εντολές συμβολοσειρών (string instructions), μεταξύ των οποίων μετακίνησης και σύγκρισης συμβολοσειρών.

Οι δύο πρώτες κατηγορίες δεν έχουν ιδιαίτερο ενδιαφέρον, εκτός από το γεγονός ότι οι εντολές αριθμητικών και λογικών πράξεων επιτρέπουν ο προορισμός να είναι είτε καταχωρητής είτε θέση μνήμης. Η Εικόνα 2.43 παρουσιάζει μερικές τυπικές εντολές της αρχιτεκτονικής IA-32 και τις λειτουργίες τους.

Οι διακλαδώσεις υπό συνθήκη (conditional branches) στην αρχιτεκτονική IA-32 βασίζονται στους *κωδικούς συνθήκης* (condition codes) ή *σημαίες* (flags). Οι κωδικοί συνθήκης ορίζονται ως παρενέργεια μιας πράξης (λειτουργίας)· οι περισσότερες χρησιμοποιούνται για να συγκρίνουν την τιμή ενός αποτελέσματος με το 0. Οι διακλαδώσεις στη συνέχεια ελέγχουν τους κωδικούς συνθήκης. Το επιχείρημα υπέρ των κωδικών συνθήκης είναι ότι εμφανίζονται ως μέρος κανονικών πράξεων και είναι πιο γρήγορος ο έλεγχός τους σε σχέση με τη σύγκριση καταχωρητών, όπως κάνει ο MIPS για τις εντολές `beq` και `bne`. Το επιχείρημα εναντίον των κωδικών συνθήκης είναι ότι η σύγκριση με το 0 αυξάνει το χρόνο εκτέλεσης της πράξης, εφόσον χρησιμοποιεί επιπλέον υλικό μετά την πράξη, και συχνά ο προγραμματιστής πρέπει να χρησιμοποιήσει εντολές σύγκρισης για τον έλεγχο μιας τιμής η οποία δεν είναι το αποτέλεσμα κάποιας πράξης. Επιπλέον, διευθύνσεις διακλάδωσης σχετικές ως προς PC πρέπει να οριστούν σε αριθμό byte αφού, σε αντίθεση με τον MIPS, οι εντολές του 80386 δεν έχουν όλες μήκος 4 byte.

Οι εντολές συμβολοσειρών είναι μέρος της γενεαλογίας της αρχιτεκτονικής IA-32 από τον 8080 και δεν εκτελούνται με συνήθη τρόπο στα περισσότερα προγράμματα. Είναι συνήθως πιο αργές από τις ισοδύναμες ρουτίνες λογισμικού (δείτε την πλάνη της σελίδας 161).

Η Εικόνα 2.44 παραθέτει μερικές από τις εντολές πράξεων ακεραίων της αρχιτεκτονικής IA-32. Πολλές από αυτές είναι διαθέσιμες τόσο σε μορφή byte όσο και σε μορφή λέξης.

Εντολή	Λειτουργία
`JE name`	`αν ίσο (κωδικός συνθήκης) {EIP=name}· EIP-128 ≤ name < EIP+128`
`JMP name`	`EIP=name`
`CALL name`	`SP=SP-4· M[SP]=EIP+5· EIP=name·`
`MOVW EBX,[EDI+45]`	`EBX=M[EDI+45]`
`PUSH ESI`	`SP=SP-4· M[SP]=ESI`
`POP EDI`	`EDI=M[SP]· SP=SP+4`
`ADD EAX,#6765`	`EAX= EAX+6765`
`TEST EDX,#42`	Κωδικός συνθήκης (σημαίες) = EDX συν 42
`MOVSL`	`M[EDI]=M[ESI]· EDI=EDI+4· ESI=ESI+4`

ΕΙΚΟΝΑ 2.43 Μερικές τυπικές εντολές της IA-32 και οι λειτουργίες τους. Στην Εικόνα 2.44 παρουσιάζεται μια λίστα συχνών πράξεων. Η εντολή `CALL` αποθηκεύει την τιμή του καταχωρητή EIP για την επόμενη εντολή στη στοίβα. (Ο καταχωρητής EIP είναι ο μετρητής προγράμματος της Intel.)

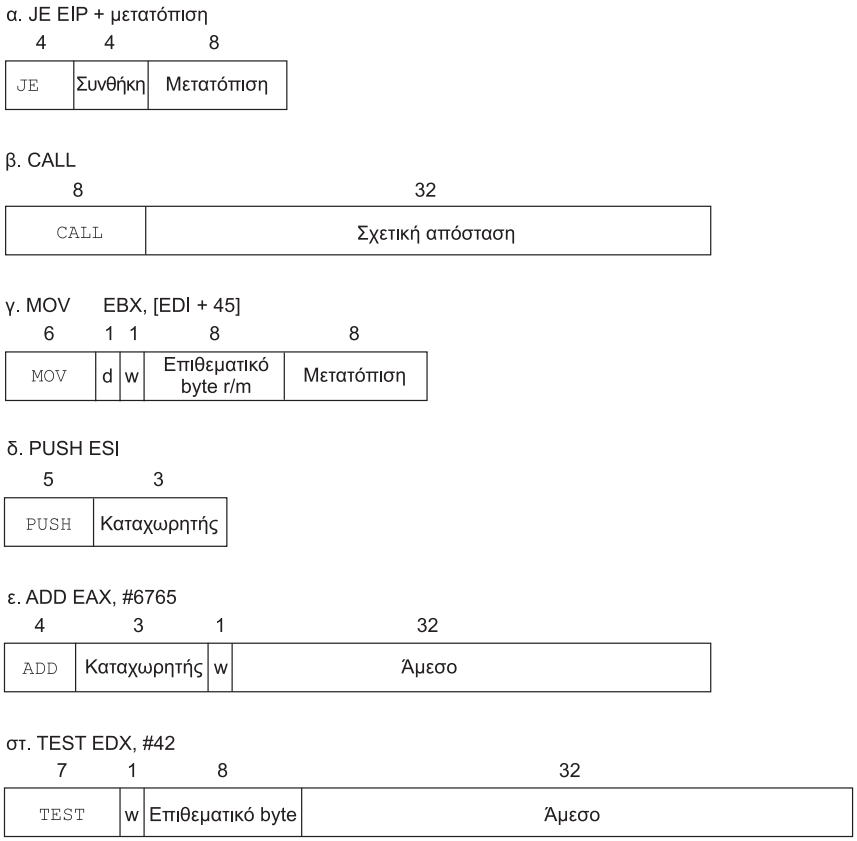
Εντολή	Σημασία
Έλεγχος	**Διακλαδώσεις υπό συνθήκη και χωρίς συνθήκη**
JNZ, JZ	Άλμα αν ισχύει η συνθήκη, στη διεύθυνση EIP + σχετική απόσταση 8 bit· JNE (για το JNZ), JE (για το JZ) είναι εναλλακτικά ονόματα
JMP	Άλμα χωρίς συνθήκη – σχετική απόσταση 8 ή 16 bit
CALL	Κλήση υπορουτίνας – σχετική απόσταση 16 bit· διεύθυνση επιστροφής τοποθετείται στη στοίβα
RET	Εξάγει την διεύθυνση επιστροφής από τη στοίβα και κάνει άλμα σε αυτή
LOOP	Διακλάδωση βρόχου – μείωση του ECX· άλμα στον EIP + μετατόπιση 8 bit αν ECX ≠ 0
Μεταφορά δεδομένων	**Μετακίνηση δεδομένων μεταξύ καταχωρητών ή μεταξύ καταχωρητή και μνήμης**
MOV	Μετακίνηση μεταξύ δύο καταχωρητών ή μεταξύ καταχωρητή και μνήμης
PUSH, POP	Τοποθέτηση τελεστέου προέλευσης στη στοίβα· εξαγωγή τελεστέου από την κορυφή της στοίβας σε έναν καταχωρητή
LES	Φόρτωση του ES και ενός από τους καταχωρητές γενικού σκοπού από τη μνήμη
Αριθμητικές και λογικές πράξεις	**Αριθμητικές και λογικές πράξεις με τη χρήση των καταχωρητών δεδομένων και της μνήμης**
ADD, SUB	Πρόσθεση της προέλευσης στον προορισμό· αφαίρεση της προέλευσης από τον προορισμό· μορφή καταχωρητή-μνήμης
CMP	Σύγκριση προέλευσης και προορισμού· μορφή καταχωρητή-μνήμης
SHL, SHR, RCR	Αριστερή ολίσθηση, δεξιά λογική ολίσθηση· δεξιά ολίσθηση με τον κωδικό συνθήκης κρατουμένου ως γέμισμα (fill)
CBW	Μετατροπή του byte στα 8 δεξιότερα bit του EAX σε λέξη 16 bit στα δεξιά του EAX
TEST	Λογικό AND μεταξύ προέλευσης και προορισμού, ορίζει τους κωδικούς συνθήκης
INC, DEC	Αύξηση προορισμού, μείωση προορισμού
OR, XOR	Λογικό OR αποκλειστικό OR μορφή καταχωρητή-μνήμης
Συμβολοσειρά	**Μετακίνηση μεταξύ τελεστέων συμβολοσειρών · το μήκος δίνεται από ένα επανάληψης πρόθεμα**
MOVS	Αντιγράφει από τη συμβολοσειρά προέλευσης στον προορισμό αυξάνοντας τους ESI και EDI· μπορεί να επαναληφθεί
LODS	Φορτώνει ένα byte, λέξη, ή διπλή λέξη μιας συμβολοσειράς στον καταχωρητή EAX

ΕΙΚΟΝΑ 2.44 Μερικές τυπικές λειτουργίες στην αρχιτεκτονική IA-32. Πολλές λειτουργίες (πράξεις) χρησιμοποιούν μορφή καταχωρητή-μνήμης, όπου είτε η προέλευσης είτε ο προορισμός μπορεί να είναι η μνήμη και το άλλο μπορεί να είναι ένας καταχωρητής ή ένας άμεσος τελεστέος.

Κωδικοποίηση εντολών της αρχιτεκτονικής IA-32

Κρατήσαμε το χειρότερο για το τέλος: η κωδικοποίηση των εντολών στον 80386 είναι πολύπλοκη, με πολλές διαφορετικές μορφές εντολών. Οι εντολές για τον 80386 μπορεί να ποικίλουν από ένα byte, όταν δεν υπάρχουν τελεστέοι, έως 17 byte.

Η Εικόνα 2.45 δείχνει τη μορφή εντολής για διάφορα παραδείγματα εντολών τής Εικόνας 2.43. Το byte του κωδικού λειτουργίας (opcode) συνήθως περιέχει ένα bit που λέει αν ο τελεστέος είναι 8 ή 32 bit. Για κάποιες εντολές, ο κωδικός λειτουργίας μπορεί να περιλαμβάνει τον τρόπο διευθυνσιοδότησης και τον καταχωρητή· αυτό ισχύει σε πολλές εντολές που έχουν τη μορφή «καταχωρητής = καταχωρητής – λειτουργία – άμεσος τελεστέος». Άλλες εντολές χρησιμοποιούν ένα «επιθεματικό byte» (postbyte) ή επιπλέον byte κωδικού λειτουργίας,



ΕΙΚΟΝΑ 2.45 Τυπικές μορφές εντολών της IA-32. Η Εικόνα 2.46 δείχνει την κωδικοποίηση του επιθεματικού byte. Πολλές εντολές περιέχουν το πεδίο w του 1 bit, το οποίο λέει αν η λειτουργία είναι ενός byte ή μιας διπλής λέξης. Το πεδίο d στην εντολή `MOV` χρησιμοποιείται σε εντολές που μπορούν να μετακινήσουν δεδομένα προς και από τη μνήμη και δείχνει την κατεύθυνση της μετακίνησης. Η εντολή `ADD` απαιτεί 32 bit για το άμεσο πεδίο επειδή σε κατάσταση λειτουργίας 32 bit οι άμεσοι τελεστέοι είναι είτε 8 είτε 32 bit. Το άμεσο πεδίο στην εντολή `TEST` έχει μήκος 32 bit επειδή δεν υπάρχει άμεσος τελεστέος 8 bit για έλεγχο της κατάστασης λειτουργίας 32 bit. Συνολικά, οι εντολές μπορούν να ποικίλουν σε μήκος από 1 έως 17 byte. Το μεγάλο μήκος προέρχεται από επιπλέον προθέματα του 1 byte, τόσο με έναν άμεσο τελεστέο 4 byte και μια διεύθυνση μετατόπισης 4 byte, όσο και έναν κωδικό λειτουργίας 2 byte, και τον προσδιοριστή κατάστασης λειτουργίας κλιμακούμενου αριθμοδείκτη, ο οποίος προσθέτει ένα ακόμη byte.

με την ετικέτα «mod, reg, r/m», το οποίο περιέχει τις πληροφορίες για τον τρόπο διευθυνσιοδότησης. Αυτό το επιθεματικό byte χρησιμοποιείται για πολλές από τις εντολές που χρησιμοποιούν τη μνήμη. Ο τρόπος διευθυνσιοδότησης «βάση συν κλιμακούμενος αριθμοδείκτης» (base plus scaled index) χρησιμοποιεί ένα δεύτερο επιθεματικό byte, με την ετικέτα «sc, index, base».

Η Εικόνα 2.46 δείχνει την κωδικοποίηση των δύο προσδιοριστών διεύθυνσης (address specifiers) των επιθεματικών byte για την κατάσταση λειτουργίας 16 και 32 bit. Δυστυχώς, για να γίνει πλήρως κατανοητό ποιοι καταχωρητές και ποιοι τρόποι διευθυνσιοδότησης είναι διαθέσιμοι, χρειάζεται να δείτε την κωδικοποίηση όλων των τρόπων διευθυνσιοδότησης και, σε μερικές περιπτώσεις, ακόμη και την κωδικοποίηση των εντολών.

reg	w = 0	w = 1		r/m	mod = 0		mod = 1		mod = 2		mod = 3
		16b	32b		16b	32b	16b	32b	16b	32b	
0	AL	AX	EAX	0	addr=BX+SI	=EAX	*ίδια διεύθυνση με mod=0 + disp8*	*ίδια διεύθυνση με mod=0 + disp8*	*ίδια διεύθυνση με mod=0 + disp16*	*ίδια διεύθυνση με mod=0 + disp32*	*ίδια με πεδίο reg*
1	CL	CX	ECX	1	addr=BX+DI	=ECX					
2	DL	DX	EDX	2	addr=BP+SI	=EDX					
3	BL	BX	EBX	3	addr=BP+SI	=EBX					
4	AH	SP	ESP	4	addr=SI	*=(sib)*	SI+disp8	*(sib)+disp8*	SI+disp8	(*sib*)+disp32“	–
5	CH	BP	EBP	5	addr=DI	=disp32	*BP+dis*	*2DI+disp8*	*DI+disp16*	EBP+disp32	–
6	DH	SI	ESI	6	addr=disp16	=ESI	BP+disp8	*ESIBP+disp8*	BP+disp16	ESI+disp32	–
7	BH	DI	EDI	7	addr=BX	=EDI	BX+disp8	*EDIBX+disp8*	BX+disp16	EDI+disp32	–

ΕΙΚΟΝΑ 2.46 Η αρχιτεκτονική του πρώτου προσδιοριστή διεύθυνσης της αρχιτεκτονικής IA-32, «*mod, reg, r/m*». Οι τέσσερις πρώτες στήλες παρουσιάζουν την κωδικοποίηση του πεδίου reg των 3 bit, που εξαρτάται από το bit w του κωδικού λειτουργίας (opcode) και από το αν η μηχανή είναι σε κατάσταση λειτουργίας 16 bit (8086) ή 32 bit (80386). Οι υπόλοιπες στήλες εξηγούν τα πεδία mod και r/m. Η σημασία του πεδίου r/m 3 bit εξαρτάται από την τιμή στα 2 bit του πεδίου mod και το μέγεθος της διεύθυνσης. Βασικά, οι καταχωρητές που χρησιμοποιούνται στον υπολογισμό της διεύθυνσης παρουσιάζονται στην έκτη και στην έβδομη στήλη, κάτω από την περίπτωση mod = 0, με την πρόσθεση μιας μετατόπισης 8 bit στην περίπτωση mod = 1 και μιας μετατόπισης 16 bit ή 32 bit στην περίπτωση mod = 2, ανάλογα με τον τρόπο διευθυνσιοδότησης. Οι εξαιρέσεις είναι η περίπτωση όπου r/m = 6 όταν mod = 1 ή mod = 2 σε κατάσταση λειτουργίας 16 bit οπότε επιλέγεται ο καταχωρητής BP συν τη μετατόπιση· η περίπτωση όπου r/m = 5 όταν mod = 1 ή mod = 2 σε κατάσταση λειτουργίας 32 bit οπότε επιλέγεται ο καταχωρητής EBP συν τη μετατόπιση· και η περίπτωση όπου r/m = 4 σε κατάσταση λειτουργίας 32 bit όταν mod = 3, όπου (sib) σημαίνει χρήση του τρόπου διευθυνσιοδότησης κλιμακούμενου αριθμοδείκτη (scaled index) που παρουσιάζεται στην Εικόνα 2.42. Όταν mod = 3, το πεδίο r/m δείχνει έναν καταχωρητή, χρησιμοποιώντας την ίδια κωδικοποίηση όπως το πεδίο reg σε συνδυασμό με το bit w.

Συμπεράσματα για την αρχιτεκτονική IA-32

Η Intel διέθετε ένα μικροεπεξεργαστή 16 bit δύο χρόνια πριν από την εμφάνιση των πιο κομψών αρχιτεκτονικών των ανταγωνιστών της όπως η 68000 της Motorola, και αυτό το γρήγορο ξεκίνημα οδήγησε στην επιλογή του 8086 ως κεντρικής μονάδας επεξεργασίας (central processing unit — CPU) για τον προσωπικό υπολογιστή της IBM (IBM PC). Οι μηχανικοί της Intel γενικά παραδέχονται ότι η αρχιτεκτονική IA-32 είναι πιο δύσκολη στην κατασκευή της συγκριτικά με μηχανές όπως ο MIPS, αλλά η πολύ μεγαλύτερη αγορά σημαίνει ότι η Intel μπορεί να διαθέσει περισσότερους πόρους ώστε να βοηθήσει να ξεπεραστεί η πρόσθετη πολυπλοκότητα. Αυτό που λείπει από την αρχιτεκτονική IA-32 σε στυλ αντισταθμίζεται σε ποσότητα, κάνοντας την όμορφη κάτω από το σωστό πρίσμα.

Το καλό είναι ότι τα πιο συχνά χρησιμοποιούμενα τμήματα της αρχιτεκτονικής IA-32 δεν είναι δύσκολο να υλοποιηθούν, όπως απέδειξε η Intel βελτιώνοντας ραγδαία την απόδοση των προγραμμάτων χειρισμού ακεραίων από το 1978. Για να πάρουν αυτή την απόδοση, οι μεταγλωττιστές πρέπει να αποφύγουν τα τμήματα της αρχιτεκτονικής που είναι δύσκολο να υλοποιηθούν με ταχύτητα.

2.17 Πλάνες και παγίδες

Πλάνη: Πιο ισχυρές εντολές σημαίνει υψηλότερη απόδοση.

Μέρος της ισχύος της αρχιτεκτονικής Intel IA-32 είναι τα προθέματα που μπορούν να τροποποιήσουν την εκτέλεση της επόμενης εντολής. Ένα πρόθεμα μπορεί να επαναλάβει την επόμενη εντολή μέχρι ένας μετρητής να φτάσει στην τιμή 0. Έτσι, για τη μετακίνηση δεδομένων στη μνήμη, φαίνεται ότι η φυσική ακο-

λουθία εντολών είναι η χρήση της εντολής move με το πρόθεμα επανάληψης για την πραγματοποίηση μετακινήσεων 32 bit από μνήμη σε μνήμη.

Μια εναλλακτική μέθοδος, η οποία χρησιμοποιεί τις καθιερωμένες εντολές που υπάρχουν σε όλους τους υπολογιστές, είναι η φόρτωση των δεδομένων στους καταχωρητές και στη συνέχεια η αποθήκευση των καταχωρητών ξανά στη μνήμη. Αυτή η δεύτερη έκδοση του συγκεκριμένου προγράμματος, με την επανάληψη του κώδικα για τη μείωση της επιβάρυνσης του βρόχου, αντιγράφει περίπου 1,5 φορές ταχύτερα. Μια τρίτη έκδοση, η οποία χρησιμοποίησε τους μεγαλύτερους καταχωρητές κινητής υποδιαστολής αντί για τους ακέραιους καταχωρητές της IA-32, αντιγράφει περίπου 2 φορές γρηγορότερα από την πολύπλοκη εντολή.

Πλάνη: Γράψτε σε συμβολική γλώσσα για να πάρετε την υψηλότερη απόδοση.

Κάποτε οι μεταγλωττιστές γλωσσών προγραμματισμού παρήγαγαν απλοϊκές ακολουθίες εντολών· η αυξανόμενη πολυπλοκότητα των μεταγλωττιστών σημαίνει ότι το κενό μεταξύ του μεταγλωττιζόμενου κώδικα και του κώδικα που παράγεται «με το χέρι» κλείνει γρήγορα. Στην πραγματικότητα, για να ανταγωνιστεί τους σύγχρονους μεταγλωττιστές, ο προγραμματιστής συμβολικής γλώσσας χρειάζεται να καταλάβει σε βάθος τις έννοιες των Κεφαλαίων 6 και 7 (για τη διοχέτευση — pipelining — του επεξεργαστή και την ιεραρχία της μνήμης).

Αυτή η μάχη μεταξύ μεταγλωττιστών και προγραμματιστών συμβολικής γλώσσας είναι μια κατάσταση στην οποία οι άνθρωποι χάνουν έδαφος. Για παράδειγμα, η C παρέχει στον προγραμματιστή μια ευκαιρία να δώσει στο μεταγλωττιστή μια υπόδειξη (hint) σχετικά με το ποιες μεταβλητές να κρατήσει στους καταχωρητές αντί να τις τοποθετήσει στη μνήμη. Όταν οι μεταγλωττιστές ήταν κακής ποιότητας ως προς την κατανομή των καταχωρητών (register allocation), αυτές οι υποδείξεις ήταν ζωτικής σημασίας για την απόδοση. Για την ακρίβεια, κάποια εγχειρίδια της C αφιέρωναν ένα σημαντικό μέρος τους σε παραδείγματα τα οποία χρησιμοποιούν αποδοτικά υποδείξεις για τους καταχωρητές. Σήμερα, οι μεταγλωττιστές τής C γενικά αγνοούν τέτοιες υποδείξεις επειδή ο μεταγλωττιστής κάνει καλύτερη δουλειά στην κατανομή των καταχωρητών από τον προγραμματιστή.

Ακόμη *και αν* το γράψιμο «με το χέρι» οδηγεί σε γρηγορότερο κώδικα, οι κίνδυνοι από τον προγραμματισμό σε συμβολική γλώσσα είναι ο μεγαλύτερος χρόνος κωδικοποίησης και αποσφαλμάτωσης, η απώλεια της φορητότητας, και η δυσκολία συντήρησης τέτοιου κώδικα. Ένα από τα λίγα ευρύτατα αποδεκτά αξιώματα της μηχανικής λογισμικού (software engineering) είναι ότι η κωδικοποίηση απαιτεί περισσότερο χρόνο αν γράφετε περισσότερες γραμμές, και η γραφή ενός προγράμματος σε συμβολική γλώσσα σαφώς χρειάζεται πολύ περισσότερες γραμμές σε σχέση με τη C. Επιπλέον, αφού κωδικοποιηθεί, το πρόγραμμα διατρέχει τον κίνδυνο να γίνει δημοφιλές. Τέτοια προγράμματα ζουν πάντα μεγαλύτερο διάστημα από το αναμενόμενο, πράγμα το οποίο σημαίνει ότι κάποιος θα πρέπει να ενημερώνει τον κώδικα για αρκετά χρόνια και να τον κάνει να δουλεύει με νέες εκδόσεις λειτουργικών συστημάτων και νέα μοντέλα μηχανών. Η γραφή τους σε μια γλώσσα υψηλότερου επιπέδου αντί για τη συμβολική, όχι μόνον επιτρέπει σε μελλοντικούς μεταγλωττιστές να προσαρμόζουν τον κώδικα για μελλοντικές μηχανές, αλλά κάνει ευκολότερη και τη συντήρηση του λογισμικού και επιτρέπει στο πρόγραμμα να εκτελείται σε περισσότερες μάρκες υπολογιστών.

Παγίδα: Να ξεχνούμε ότι οι διευθύνσεις διαδοχικών λέξεων σε μηχανές με διευθυνσιοδότηση byte δε διαφέρουν κατά ένα.

Οι προγραμματιστές συμβολικής γλώσσας έχουν κοπιάσει πολύ για σφάλματα που έγιναν λόγω της παραδοχής ότι η διεύθυνση της επόμενης λέξης μπορεί να υπολογιστεί μέσω της αύξησης της διεύθυνσης σε έναν καταχωρητή κατά ένα αντί κατά το μέγεθος της λέξης σε byte. Η γνώση είναι δύναμη!

Παγίδα: Η χρήση ενός δείκτη προς μια αυτόματη μεταβλητή έξω από τη διαδικασία που την ορίζει.

Ένα συνηθισμένο λάθος στη χρήση των δεικτών είναι η μεταβίβαση ενός αποτελέσματος μιας διαδικασίας που περιέχει δείκτη προς έναν πίνακα ο οποίος είναι τοπικός σε αυτή τη διαδικασία. Σύμφωνα με την αρχή της στοίβας στην Εικόνα 2.16, η μνήμη που περιέχει τον τοπικό πίνακα θα ξαναχρησιμοποιηθεί αμέσως μετά την επιστροφή από τη διαδικασία. Οι δείκτες προς αυτόματες μεταβλητές μπορούν να οδηγήσουν σε χάος.

2.18 Συμπερασματικές παρατηρήσεις

Το λιγότερο είναι περισσότερο.
Robert Browning, *Andrea del Sarto*, 1855

Οι δύο αρχές του υπολογιστή αποθηκευμένου προγράμματος (stored-program computer) είναι η χρήση εντολών που δεν ξεχωρίζουν από αριθμούς και η χρήση μεταβλητής μνήμης για προγράμματα. Αυτές οι αρχές επιτρέπουν σε μία μόνο μηχανή να βοηθάει επιστήμονες περιβάλλοντος, οικονομικούς συμβούλους, και συγγραφείς στις ειδικότητές τους. Η επιλογή ενός συνόλου εντολών που η μηχανή μπορεί να καταλάβει απαιτεί μια λεπτή ισορροπία μεταξύ του αριθμού των εντολών που απαιτούνται για την εκτέλεση ενός προγράμματος, του αριθμού των κύκλων ρολογιού που απαιτούνται από μία εντολή, και της ταχύτητας του ρολογιού. Τέσσερις σχεδιαστικές αρχές οδηγούν τους σχεδιαστές συνόλων εντολών στην επίτευξη αυτής της λεπτής ισορροπίας:

1. *Η απλότητα ευνοεί την κανονικότητα (regularity).* Η κανονικότητα είναι το κίνητρο για πολλά χαρακτηριστικά τού συνόλου εντολών του MIPS: η διατήρηση όλων των εντολών σε ένα μέγεθος, η απαίτηση για τρεις πάντα τελεστέους καταχωρητών στις αριθμητικές εντολές, και η διατήρηση των πεδίων των καταχωρητών στο ίδιο μέρος σε κάθε μορφή εντολής.
2. *Το μικρότερο είναι ταχύτερο.* Η επιθυμία για ταχύτητα είναι ο λόγος που ο MIPS έχει 32 καταχωρητές αντί για πολύ περισσότερους.
3. *Κάνε τη συνηθισμένη περίπτωση γρήγορη.* Παραδείγματα γρήγορης υλοποίησης κοινών περιπτώσεων του MIPS περιλαμβάνουν τη σχετική διευθυνσιοδότηση ως προς PC για διακλαδώσεις υπό συνθήκη, και την άμεση διευθυνσιοδότηση για σταθερούς τελεστέους.
4. *Η καλή σχεδίαση απαιτεί καλούς συμβιβασμούς.* Ένα παράδειγμα από τον MIPS ήταν ο συμβιβασμός μεταξύ της παροχής μεγάλων διευθύνσεων και σταθερών στις εντολές, και της διατήρησης όλων των εντολών στο ίδιο μήκος.

Επάνω από αυτό το επίπεδο της μηχανής βρίσκεται η συμβολική γλώσσα, μια γλώσσα που μπορεί να διαβάσει ο άνθρωπος. Ο συμβολομεταφραστής τη μεταφράζει σε δυαδικούς αριθμούς που οι μηχανές μπορούν να καταλάβουν, και «επεκτείνει» το σύνολο των εντολών δημιουργώντας συμβολικές εντολές οι οποίες δεν υπάρχουν στο υλικό. Για παράδειγμα, σταθερές ή διευθύνσεις που

είναι πολύ μεγάλες διαιρούνται σε τμήματα κατάλληλου μεγέθους, συνηθισμένες παραλλαγές εντολών παίρνουν το δικό τους όνομα, και ούτω καθεξής. Η Εικόνα 2.47 παραθέτει τις εντολές του MIPS που έχουμε καλύψει μέχρι τώρα, τόσο πραγματικές όσο και ψευδοεντολές.

Αυτές οι εντολές «δεν έχουν γεννηθεί ίσες»· η δημοτικότητα λίγων κυριαρχεί στις πολλές. Για παράδειγμα, η Εικόνα 2.48 παρουσιάζει τη δημοτικότητα κάθε κατηγορίας εντολών για το σύνολο μετροπρογραμμάτων SPEC2000. Η ποικίλη δημοτικότητα των εντολών παίζει σημαντικό ρόλο στα κεφάλαια για την απόδοση, τη διαδρομή δεδομένων, τη μονάδα ελέγχου, και τη διοχέτευση.

Εντολές του MIPS	Όνομα	Μορφή	Ψευδοεντολές του MIPS	Όνομα	Μορφή
add	add	R	move	move	R
subtract	sub	R	multiply	mult	R
add immediate	addi	I	multiply immediate	multi	I
load word	lw	I	load immediate	li	I
store word	sw	I	branch less than	blt	I
load half	lh	I	branch less than or equal	ble	I
store half	sh	I	branch greater than	bgt	I
load byte	lb	I	branch greater than or equal	bge	I
store byte	sb	I			
load upper immediate	lui	I			
and	and	R			
or	or	R			
nor	nor	R			
and immediate	andi	I			
or immediate	ori	I			
shift left logical	sll	R			
shift right logical	srl	R			
branch on equal	beq	I			
branch on not equal	bne	I			
set less than	slt	R			
set less than immediate	slti	I			
jump	j	J			
jump register	jr	R			
jump and link	jal	J			

ΕΙΚΟΝΑ 2.47 Το σύνολο εντολών του επεξεργαστή MIPS που έχουμε καλύψει μέχρι τώρα, με τις πραγματικές εντολές στα αριστερά και τις ψευδοεντολές στα δεξιά. Το Παράρτημα Α (Ενότητα Α.10, σελίδα 113) περιγράφει την πλήρη αρχιτεκτονική του MIPS. Η Εικόνα 2.27 παρουσιάζει περισσότερες λεπτομέρειες της αρχιτεκτονικής του MIPS που αποκαλύψαμε σε αυτό το κεφάλαιο.

Κατηγορία εντολής	Παραδείγματα του MIPS	Αντιστοιχία με γλώσσα υψηλού επιπέδου	Συχνότητα	
			Ακέραιες	Κιν. υποδ.
Αριθμητικές πράξεις	`add, sub, addi`	Πράξεις (λειτουργίες) σε εντολές ανάθεσης τιμής	24%	48%
Μεταφορά δεδομένων	`lw, sw, lb, sb, lui`	Αναφορές σε δομές δεδομένων, όπως πίνακες	36%	39%
Λογικές πράξεις	`and, or, nor, andi, ori, sll, srl`	Πράξεις (λειτουργίες) σε εντολές ανάθεσης τιμής	18%	4%
Διακλάδωση υπό συνθήκη	`beq, bne, slt, slti`	Εντολές if και βρόχοι	18%	6%
Άλμα	`j, jr, jal`	Κλήσεις διαδικασιών, επιστροφές, και εντολές case/switch	3%	0%

ΕΙΚΟΝΑ 2.48 Κατηγορίες εντολών του επεξεργαστή MIPS, παραδείγματα, αντιστοιχία με δομές γλώσσας προγραμματισμού υψηλού επιπέδου, και ποσοστό εντολών του MIPS που εκτελούνται ανά κατηγορία για το μέσο όρο πέντε μετροπρογραμμάτων χειρισμού ακεραίων SPEC2000 και πέντε μετροπρογραμμάτων χειρισμού αριθμών κινητής υποδιαστολής SPEC2000. Η Εικόνα 3.26 παρουσιάζει το ποσοστό των ξεχωριστών εκτελούμενων εντολών του MIPS.

Κάθε κατηγορία εντολών του MIPS είναι συσχετισμένη με δομές που εμφανίζονται στις γλώσσες προγραμματισμού:

- Οι αριθμητικές εντολές αντιστοιχούν σε πράξεις (λειτουργίες) που συναντώνται σε εντολές ανάθεσης τιμής (assignment statements).
- Οι εντολές μεταφοράς δεδομένων είναι πιο πιθανό να παρουσιαστούν κατά το χειρισμό δομών δεδομένων όπως πίνακες ή δομές (structures).
- Οι υπό συνθήκη διακλαδώσεις χρησιμοποιούνται σε εντολές *if* και σε βρόχους.
- Τα άλματα χωρίς συνθήκη χρησιμοποιούνται σε κλήσεις διαδικασιών και επιστροφές (returns) και για εντολές *case/switch*.

Αφού εξηγήσουμε την αριθμητική των υπολογιστών στο Κεφάλαιο 3, θα αποκαλύψουμε περισσότερα για την αρχιτεκτονική του συνόλου εντολών του MIPS.

Β ΤΟΜΟΣ 2.19 Ιστορική προοπτική και πρόσθετες πηγές

Αυτή η ενότητα παρέχει μια σύνοψη της εξέλιξης των αρχιτεκτονικών συνόλων εντολών στο χρόνο, και δίνει επίσης μια σύντομη ιστορία των γλωσσών προγραμματισμού και των μεταγλωττιστών. Οι αρχιτεκτονικές συνόλου εντολών περιλαμβάνουν αρχιτεκτονικές συσσωρευτών (accumulator architectures), αρχιτεκτονικές γενικών (general purpose) καταχωρητών, αρχιτεκτονικές στοίβας, και μια σύντομη ιστορία της αρχιτεκτονικής IA-32. Εξετάζουμε επίσης αμφιλεγόμενα θέματα αρχιτεκτονικών υπολογιστών για γλώσσες υψηλού επιπέδου. Η ιστορία των γλωσσών προγραμματισμού περιλαμβάνει τις Fortran, Lisp, Algol, C, Cobol, Pascal, Simula, Smalltalk, C++, και Java, και η ιστορία των μεταγλωττιστών περιλαμβάνει τα βασικά ορόσημα και τους πρωτοπόρους που τα πέτυχαν. Το υπόλοιπο αυτής της ενότητας βρίσκεται στο δεύτερο τόμο.

2.20 Ασκήσεις

Το Παράρτημα Α παρουσιάζει τον προσομοιωτή του MIPS, που αποτελεί βοήθημα γι' αυτές τις ασκήσεις. Μολονότι ο προσομοιωτής δέχεται ψευδοεντολές, προσπαθήστε να μη χρησιμοποιήσετε ψευδοεντολές για όσες ασκήσεις σάς ζητούν να παραγάγετε κώδικα για τον MIPS. Ο στόχος σας είναι να μάθετε το πραγματικό σύνολο εντολών του MIPS και, αν σας ζητηθεί να μετρήσετε τις εντολές, η μέτρησή σας θα πρέπει να αντιστοιχεί στις πραγματικές εντολές που θα εκτελεστούν και όχι στις ψευδοεντολές.

Υπάρχουν μερικές περιπτώσεις που πρέπει να χρησιμοποιούνται οι ψευδοεντολές (για παράδειγμα, η ψευδοεντολή `la` όταν μια πραγματική τιμή δεν είναι γνωστή κατά το χρόνο της συμβολομετάφρασης). Σε πολλές περιπτώσεις, είναι αρκετά βολικές και οδηγούν σε πιο ευανάγνωστο κώδικα (για παράδειγμα, οι εντολές `li` και `move`). Αν επιλέξετε να χρησιμοποιήσετε ψευδοεντολές γι' αυτούς τους λόγους, προσθέστε μία ή δύο προτάσεις στη λύση σας δηλώνοντας ποιες ψευδοεντολές χρησιμοποιήσατε και γιατί.

2.1 [15] <§2.4> Για περισσότερη εξάσκηση: Μορφές εντολών

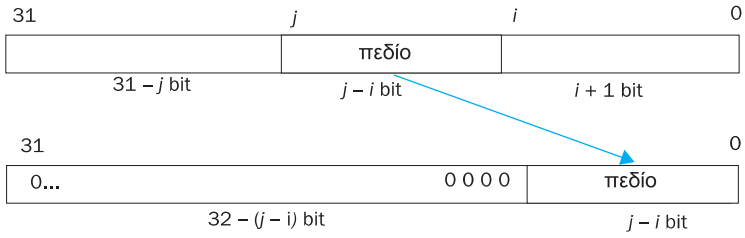
2.2 [5] <§2.4> Ποιο δυαδικό αριθμό αναπαριστά ο δεκαεξαδικός αριθμός 7fff $fffa_{hex}$; Ποιο δεκαδικό αριθμό αναπαριστά;

2.3 [5] <§2.4> Ποιο δεκαεξαδικό αριθμό αναπαριστά ο δυαδικός αριθμός $1100\ 1010\ 1111\ 1110\ 1111\ 1010\ 1100\ 1110_{two}$;

2.4 [5] <§2.4> Γιατί ο MIPS δε διαθέτει εντολή άμεσης αφαίρεσης (subtract immediate);

2.5 [15] <§2.5> Για περισσότερη εξάσκηση: Κώδικας MIPS και λογικές λειτουργίες

2.6 [15] <§2.5> Κάποιοι υπολογιστές έχουν ειδικές εντολές για την εξαγωγή ενός οποιουδήποτε πεδίου από έναν καταχωρητή 32 bit και την τοποθέτησή του στα λιγότερο σημαντικά bit ενός καταχωρητή. Η παρακάτω εικόνα παρουσιάζει την επιθυμητή λειτουργία:



Βρείτε τη συντομότερη ακολουθία εντολών MIPS που εξάγει ένα πεδίο για τις σταθερές τιμές $i = 5$ και $j = 22$ από τον καταχωρητή `$t3` και το τοποθετεί στον καταχωρητή `$t0`. (Υπόδειξη: μπορεί να γίνει με δύο εντολές).

2.7 [10] <§2.5> Για περισσότερη εξάσκηση: Λογικές πράξεις στον MIPS

2.8 [20] <§2.5> Σε μεγαλύτερο βάθος: Πεδία bit στη C

2.9 [20] <§2.5> Σε μεγαλύτερο βάθος: Πεδία bit στη C

2.10 [20] <§2.5> Σε μεγαλύτερο βάθος: Πίνακες αλμάτων

2.11 [20] <§2.5> Σε μεγαλύτερο βάθος: Πίνακες αλμάτων

2.12 [20] <§2.5> Σε μεγαλύτερο βάθος: Πίνακες αλμάτων

2.13 [10] <§2.6> Κατασκευάστε ένα γράφημα ροής ελέγχου (control flow graph — όπως αυτό που φαίνεται στην Εικόνα 2.11) για το επόμενο τμήμα κώδικα C ή Java:

```
for (i=0; i<x; i=i+1)
    y = y + i;
```

2.14 [10] <§2.6> Για περισσότερη εξάσκηση: Γραφή κώδικα συμβολικής γλώσσας

2.15 [25] <§2.7> Υλοποιήστε τον επόμενο κώδικα C στο MIPS, υποθέτοντας ότι η `set_array` είναι η πρώτη συνάρτηση που καλείται:

```
int i;
void set_array(int num) {
    int array[10];
    for (i=0; i<10; i++){
        array[i] == compare(num,i);
    }
}
int compare(int a, int b){
    if (sub(a,b)>=0)
        return 1;
    else
        return 0;
}
int sub (int a,int b){
    return a-b;
}
```

Βεβαιωθείτε ότι θα χειριστείτε τους δείκτες στοίβας και πλαισίου κατάλληλα. Ο κώδικας των μεταβλητών κατανέμεται στη στοίβα, και η μεταβλητή `i` αντιστοιχεί στον καταχωρητή `$s0`. Σχεδιάστε την κατάσταση της στοίβας πριν από την κλήση της `set_array` και κατά τη διάρκεια της κλήσης κάθε συνάρτησης. Δείξτε τα ονόματα των καταχωρητών και των αποθηκευμένων στη στοίβα μεταβλητών και σημειώστε τη θέση των `$sp` και `$fp`.

2.16 [30] <§2.7> Σε μεγαλύτερο βάθος: Αναδρομή ουράς

2.17 [30] <§2.7> Σε μεγαλύτερο βάθος: Αναδρομή ουράς

2.18 [20] <§2.7> Σε μεγαλύτερο βάθος: Αναδρομή ουράς

2.19 [5] <§2.8> Η Iris και η Julie σπουδάζουν μηχανικοί υπολογιστών που μαθαίνουν τα σύνολα χαρακτήρων ASCII και Unicode. Βοηθήστε τες να συλλαβίσουν τα ονόματά τους και το δικό σας μικρό όνομα τόσο στο σύνολο χαρακτήρων ASCII (με τη χρήση δεκαδικής σημειογραφίας) όσο και στο σύνολο χαρακτήρων Unicode (με τη χρήση δεκαεξαδικής σημειογραφίας και του βασικού Λατινικού συνόλου χαρακτήρων).

2.20 [10] <§2.8> Υπολογίστε τις δεκαδικές τιμές byte που αποτελούν την τερματιζόμενη με μηδενικό (null) χαρακτήρα αναπαράσταση κατά ASCII της επόμενης συμβολοσειράς:

```
A byte is 8 bits
```

2.21 [30] <§§2.7, 2.8> Για περισσότερη εξάσκηση: Γραφή κώδικα MIPS και συμβολοσειρές ASCII

2.22 [20] <§§2.7, 2.8> Για περισσότερη εξάσκηση: Γραφή κώδικα MIPS και συμβολοσειρές ASCII

2.23 [20] <§§2.7, 2.8> {Ασκ. 2.22} Για περισσότερη εξάσκηση: Γραφή κώδικα MIPS και συμβολοσειρές ASCII

2.24 [30] <§§2.7, 2.8> Για περισσότερη εξάσκηση: Γραφή κώδικα MIPS και συμβολοσειρές ASCII

2.25 <§2.8> Για περισσότερη εξάσκηση: Σύγκριση C και Java με τη γλώσσα MIPS

2.26 <§2.8> Για περισσότερη εξάσκηση: Μετάφραση του MIPS σε C

2.27 <§2.8> Για περισσότερη εξάσκηση: Κατανόηση του κώδικα MIPS

2.28 <§2.8> Για περισσότερη εξάσκηση: Κατανόηση του κώδικα MIPS

2.29 [5] <§§2.3, 2.6, 2.9> Προσθέστε σχόλια στον επόμενο κώδικα του MIPS και περιγράψτε με μία πρόταση τι υπολογίζει. Υποθέστε ότι οι `$a0` και `$a1` χρησιμοποιούνται για την είσοδο, και αρχικά περιέχουν τους ακέραιους *a* και *b*, αντίστοιχα. Υποθέστε ότι ο καταχωρητής `$v0` χρησιμοποιείται για την έξοδο.

```
                add   $t0, $zero, $zero
loop:           beq   $a1, $zero, finish
                add   $t0, $t0, $a0
                sub   $a1, $a1, 1
                j     loop
finish:         addi  $t0, $t0, 100
                add   $v0, $t0, $zero
```

2.30 [12] <§§2.3, 2.6, 2.9> Το επόμενο τμήμα κώδικα επεξεργάζεται δύο πίνακες και παράγει μια σημαντική τιμή στον καταχωρητή `$v0`. Υποθέστε ότι κάθε πίνακας αποτελείται από 2500 λέξεις και έχει αριθμοδείκτες 0 έως 2499, ότι οι διευθύνσεις βάσης των πινάκων αποθηκεύονται στους καταχωρητές `$a0` και `$a1` αντίστοιχα, και ότι τα μεγέθη τους (2500) αποθηκεύονται στους καταχωρητές `$a2` και `$a3`, αντίστοιχα. Προσθέστε σχόλια στον κώδικα και περιγράψτε με μία πρόταση τι κάνει αυτός ο κώδικας. Συγκεκριμένα, ποια τιμή θα επιστραφεί στον καταχωρητή `$v0`;

```
                sll   $a2, $a2, 2
                sll   $a3, $a3, 2
                add   $v0, $zero, $zero
                add   $t0, $zero, $zero
outer:          add   $t4, $a0, $t0
                lw    $t4, 0($t4)
                add   $t1, $zero, $zero
inner:          add   $t3, $a1, $t1
                lw    $t3, 0($t3)
                bne   $t3, $t4, skip
                addi  $v0, $v0, 1
```

```
skip:           addi   $t1, $t1, 4
                bne    $t1, $a3, inner
                addi   $t0, $t0, 4
                bne    $t0, $a2, outer
```

2.31 [10] <§§2.3, 2.6, 2.9> Υποθέστε ότι ο κώδικας της Άσκησης 2.30 εκτελείται από μια μηχανή με ρολόι συχνότητας 2 GHz, η οποία απαιτεί τον παρακάτω αριθμό κύκλων για κάθε εντολή:

Εντολή	Κύκλοι
add,addi,sll	1
lw, bne	2

Στη χειρότερη περίπτωση, πόσα δευτερόλεπτα θα απαιτούσε η εκτέλεση αυτού του κώδικα;

2.32 [5] <§2.9> Δείξτε τη μία εντολή του MIPS ή την ελάχιστη ακολουθία εντολών για την επόμενη εντολή της C:

```
b = 25 | a;
```

Υποθέστε ότι το `a` αντιστοιχεί στον καταχωρητή `$t0` και το `b` αντιστοιχεί στον καταχωρητή `$t1`.

2.33 [10] <§2.9> Για περισσότερη εξάσκηση: Μετάφραση από C σε MIPS

2.34 [10] <§§ 2.3, 2.6, 2.9> Το επόμενο πρόγραμμα προσπαθεί να αντιγράψει λέξεις, από τη διεύθυνση που είναι αποθηκευμένη στον καταχωρητή `$a0`, στη διεύθυνση που είναι αποθηκευμένη στον καταχωρητή `$a1`, μετρώντας τον αριθμό των λέξεων στον καταχωρητή `$v0`. Το πρόγραμμα σταματάει την αντιγραφή όταν βρίσκει μια λέξη ίση με το 0. Δε χρειάζεται να διατηρήσετε τα περιεχόμενα των καταχωρητών `$v1`, `$a0`, και `$a1`. Η τελευταία λέξη πρέπει να αντιγράφεται αλλά να μη μετράται.

```
      addi $v0, $zero, 0     # ανάθεση αρχικών τιμών στο μετρητή
loop:lw   $v1, 0($a0)        # ανάγνωση της επόμενης λέξης από
                             # την προέλευση
      sw   $v1, 0($a1)       # εγγραφή στον προορισμό
      addi $a0, $a0, 4       # αύξηση δείκτη στην επόμενη
                             # προέλευση
      addi $a1, $a1, 4       # αύξηση δείκτη στον επόμενο
                             # προορισμό
      beq  $v1, $zero, loop # βρόχος αν λέξη που αντιγράφηκε!= 0
```

Υπάρχουν πολλά σφάλματα σε αυτό το πρόγραμμα του MIPS· διορθώστε τα και δημιουργήστε μια έκδοση του προγράμματος χωρίς σφάλματα. Όπως σε πολλές ασκήσεις αυτού του κεφαλαίου, ο ευκολότερος τρόπος για να γράψετε προγράμματα του MIPS είναι να χρησιμοποιήσετε τον προσομοιωτή που περιγράφεται στο Παράρτημα Α.

2.35 [10] <§§2.2, 2.3, 2.6, 2.9> Για περισσότερη εξάσκηση: Αντίστροφη μετάφραση από MIPS σε C

2.36 <§2.9> Για περισσότερη εξάσκηση: Μετάφραση από C σε MIPS

2.37 [25] <§2.10> Όπως είπαμε στη σελίδα 125 (στην Ενότητα «Συμβολομεταφραστής»), οι ψευδοεντολές δεν αποτελούν μέρος του συνόλου εντολών του MIPS αλλά συχνά εμφανίζονται σε προγράμματα του MIPS. Για κάθε ψευδοεντολή στον επόμενο πίνακα, δημιουργήστε μια ελάχιστη ακολουθία πραγματικών εντολών του MIPS για τον ίδιο σκοπό. Είναι πιθανό να χρειαστεί να χρησιμοποιήσετε τον καταχωρητή `$at` για κάποιες από τις ακολουθίες. Στον επόμενο πίνακα, ο χαρακτηρισμός `big` αναφέρεται σε ένα συγκεκριμένο αριθμό που απαιτεί 32 bit για την αναπαράστασή του, και ο χαρακτηρισμός `small` αναφέρεται σε έναν αριθμό που μπορεί να χωρέσει σε 16 bit.

Ψευδοεντολή	Αποτέλεσμα
`move $t1, $t2`	`$t1 = $t2`
`clear $t0`	`$t0 = 0`
`beq $t1, small, L`	`if ($t1 = small) go to L`
`beq $t2, big, L`	`if ($t2 = big) go to L`
`li $t1, small`	`$t1 = small`
`li $t2, big`	`$t2 = big`
`ble $t3, $t5, L`	`if ($t3 <= $t5) go to L`
`bgt $t4, $t5, L`	`if ($t4 > $t5) go to L`
`bge $t5, $t3, L`	`if ($t5 >= $t3) go to L`
`addi $t0, $t2, big`	`$t0 = $t2 + big`
`lw $t5, big($t2)`	`$t5 = Memory[$t2 + big]`

2.38 [5] <§§2.9, 2.10> Με την προϋπόθεση ότι κατανοείτε τη σχετική διευθυνσιοδότηση ως προς PC, εξηγήστε για ποιο λόγο θα μπορούσε ένας συμβολομεταφραστής να έχει προβλήματα με την άμεση υλοποίηση της εντολής διακλάδωσης στην παρακάτω ακολουθία κώδικα:

```
here:           beq $s0,$s2,there
...
there:          add $s0,$s0,$s0
```

Δείξτε πώς ο συμβολομεταφραστής θα ξαναέγραφε την ίδια ακολουθία κώδικα για να λύσει αυτά τα προβλήματα.

2.39 <§2.10> Για περισσότερη εξάσκηση: Ψευδοεντολές MIPS

2.40 <§2.10> Για περισσότερη εξάσκηση: Σύνδεση κώδικα MIPS

2.41 <§2.10> Για περισσότερη εξάσκηση: Σύνδεση κώδικα MIPS

2.42 [20] <§2.11> Βρείτε ένα μεγάλο πρόγραμμα γραμμένο σε C (για παράδειγμα, το gcc, που μπορείτε να το αποκτήσετε από τη διεύθυνση *http://gcc.gnu.org*) και μεταγλωττίστε το δύο φορές· μια με βελτιστοποιήσεις (χρησιμοποιήστε το `-O3`) και μια χωρίς. Συγκρίνετε τους χρόνους μεταγλώττισης και εκτέλεσης του προγράμματος. Είναι τα αποτελέσματα αυτά που περιμένατε;

2.43 [20] <§2.12> Για περισσότερη εξάσκηση: Βελτίωση των τρόπων διευθυνσιοδότησης του MIPS

2.44 [10] <§2.12> Για περισσότερη εξάσκηση: Βελτίωση των τρόπων διευθυνσιοδότησης του MIPS

2.45 [10] <§2.12> Σε μεγαλύτερο βάθος: Ο PowerPC των IBM/Motorola

2.46 [15] <§§2.6, 2.13> Η μετάφραση του MIPS για το τμήμα C (ή Java)

```
while (save[i] == k)
    i += 1;
```

της σελίδας 92 (Ενότητα «Μεταγλώττιση ενός βρόχου while της C») χρησιμοποιεί κάθε φορά τόσο μια διακλάδωση υπό συνθήκη όσο και ένα άλμα χωρίς συνθήκη στο βρόχο. Μόνον οι κακής ποιότητας μεταγλωττιστές θα παρήγαγαν κώδικα με αυτή την επιβάρυνση του βρόχου. Υποθέτοντας ότι αυτός ο κώδικας είναι σε Java (και όχι σε C), ξαναγράψτε τον κώδικα της συμβολικής γλώσσας ώστε να χρησιμοποιεί κάθε φορά το πολύ μία διακλάδωση ή ένα άλμα στο βρόχο. Επιπλέον, προσθέστε κώδικα για την πραγματοποίηση του ελέγχου της Java για δείκτες εκτός ορίων και βεβαιωθείτε ότι αυτός ο κώδικας χρησιμοποιεί κάθε φορά το πολύ μία διακλάδωση ή ένα άλμα στο βρόχο. Πόσες εντολές εκτελούνται πριν και μετά τη βελτιστοποίηση αν ο αριθμός των επαναλήψεων του βρόχου είναι 10 και η τιμή του `i` δεν είναι ποτέ εκτός ορίων;

2.47 [30] <§§2.6, 2.13> Θεωρήστε το παρακάτω τμήμα κώδικα Java:

```
for (i=0; i<=100; i=i+1)
    a[i] == b[i] + c;
```

Υποθέστε ότι οι `a` και `b` είναι πίνακες λέξεων και η διεύθυνση βάσης τού `a` είναι στον καταχωρητή `$a0` και η διεύθυνση βάσης του `b` είναι στον καταχωρητή `$a1`. Ο καταχωρητής `$t0` είναι συσχετισμένος με την `i` και ο καταχωρητής `$s0` με την τιμή τής `c`. Μπορείτε επίσης να υποθέσετε ότι όλες οι σταθερές διευθύνσεων που χρειάζεστε είναι διαθέσιμες να φορτωθούν από τη μνήμη. Γράψτε τον κώδικα για τον MIPS. Πόσες εντολές εκτελούνται κατά την εκτέλεση αυτού του κώδικα αν δεν υπάρχουν εκτός ορίων εξαιρέσεις για τους πίνακες. Πόσες αναφορές στη μνήμη δεδομένων θα γίνουν κατά την εκτέλεση;

2.48 [5] <§2.13> Γράψτε κώδικα για τον MIPS για τη μέθοδο `compareTo` της Java (που υπάρχει στην Εικόνα 2.35 της σελίδας 142).

2.49 [15] <§2.17> Κατά το σχεδιασμό συστημάτων μνήμης, είναι χρήσιμη η γνώση της συχνότητας των αναγνώσεων της μνήμης σε σύγκριση με τις εγγραφές, όπως και η συχνότητα των προσπελάσεων εντολών σε σύγκριση με αυτές των δεδομένων. Χρησιμοποιώντας τις πληροφορίες του μέσου μίγματος εντολών του MIPS για το πρόγραμμα SPEC2000int της Εικόνας 2.48 (στη σελίδα 165), βρείτε τα εξής:

α. Το ποσοστό των συνολικών προσπελάσεων μνήμης (δεδομένων και εντολών) που αφορούν δεδομένα.
β. Το ποσοστό των συνολικών προσπελάσεων μνήμης (δεδομένων και εντολών) που αφορούν αναγνώσεις. Υποθέστε ότι τα δύο τρίτα των μεταφορών δεδομένων είναι φορτώσεις.

2.50 [10] <§2.17> Πραγματοποιήστε τους ίδιους υπολογισμούς της Άσκησης 2.49, αλλά αντικαταστήστε το πρόγραμμα SPEC2000int με το πρόγραμμα SPEC2000fp.

2.51 [15] <§2.17> Υποθέστε ότι έχετε κάνει τις εξής μετρήσεις για το μέσο αριθμό κύκλων ρολογιού ανά εντολή (CPI) για τις εντολές:

Εντολή	Μέσο CPI
Αριθμητικές πράξεις	1,0 κύκλος ρολογιού
Μεταφορά δεδομένων	1,4 κύκλοι ρολογιού
Διακλάδωση υπό συνθήκη	1,7 κύκλοι ρολογιού
Άλμα	1,2 κύκλοι ρολογιού

Υπολογίστε το πραγματικό CPI για τον MIPS. Βρείτε το μέσο όρο των συχνοτήτων εντολών για τα προγράμματα SPEC2000int και SPEC2000fp της Εικόνας 2.48 (σελίδα 165) για να πάρετε το μίγμα των εντολών.

2.52 [20] <§2.18> Σε μεγαλύτερο βάθος: Στυλ συνόλων εντολών

2.53 [20] <§2.18> Σε μεγαλύτερο βάθος: Στυλ συνόλων εντολών

2.54 [10] <§2.18> Σε μεγαλύτερο βάθος: Ο υπολογιστής μοναδικής εντολής

2.55 [20] <§2.18> Σε μεγαλύτερο βάθος: Ο υπολογιστής μοναδικής εντολής

2.56 [5] <§2.19> Η ιδέα του αποθηκευμένου προγράμματος (stored-program), που εισήχθη στα τέλη της δεκαετίας του 1940, έφερε μια σημαντική αλλαγή στον τρόπο σχεδιασμού και λειτουργίας των υπολογιστών. Ποιο είναι ένα πιθανό παράδειγμα μηχανής μη αποθηκευμένου προγράμματος (nonstored-program), και ποια είναι τα προβλήματα μιας τέτοιας μηχανής; Πώς αυτά τα προβλήματα μπορούν να ξεπεραστούν από μια μηχανή αποθηκευμένου προγράμματος;

2.57 [5] <§2.19> Σε μεγαλύτερο βάθος: Ο PowerPC των IBM/Motorola

2.58 [15] <§2.19> Σε μεγαλύτερο βάθος: Ο PowerPC των IBM/Motorola

2.59 [15] <§2.19> Σε μεγαλύτερο βάθος: Λογικές εντολές