ΚΕΦΑΛΑΙΟ 7 ΔΗΜΙΟΥΡΓΙΑ ΤΕΛΙΚΟΥ ΚΩΔΙΚΑ

Στόχος

Στόχος του κεφαλαίου αυτού είναι να εξηγήσει τις βασικές αρχές δημιουργίας τελικού κώδικα. Παρουσιάζει μια μηχανή (υπολογιστή) απλής αρχιτεκτονικής, τους τρόπους προσπέλασης διευθύνσεων, την δημιουργία τελικού κώδικα και απλούς τρόπους βελτιστοποίησης του κώδικα.

Προσδοκώμενα αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό θα είσαστε σε θέση να

- Εξηγήσετε τις έννοιες μεταθετός κώδικας, assembly πρόγραμμα και απόλυτο πρόγραμμα,
- Περιγράψετε ένα απλό μοντέλο υπολογιστή για την δημιουργία τελικού κώδικα και τους τρόπους προσπέλασης διευθύνσεων για τον υπολογιστή αυτό,
- Υπολογίσετε το κόστος των διαφόρων εντολών του υπολογιστή μοντέλο,
- Εξηγήσετε την έννοια της επόμενης χρήσης των ονομάτων,
- Περιγράψετε ένα απλό αλγόριθμο δημιουργίας τελικού κώδικα από ενδιάμεσο κώδικα τριών διευθύνσεων,
- Εξηγήσετε την χρήση του περιγραφέα καταχωρητών και του περιγραφέα διευθύνσεων,
- Περιγράψετε βελτιστοποιήσεις τελικού κώδικα

Έννοιες-Κλειδιά

Δημιουργός κώδικα, τελικό πρόγραμμα, απόλυτο πρόγραμμα, μεταθετό πρόγραμμα, επόμενη χρήση, περιγραφέας καταχωρητών, περιγραφέας διευθύνσεων, βελτιστοποίηση peephole.

Εισαγωγικές παρατηρήσεις

Η τελική φάση στη διαδικασία μεταγλώττισης ενός προγράμματος είναι η δημιουργία κώδικα, η οποία δεν μπορεί να συζητηθεί χωρίς λεπτομερειακή αναφορά σε συγκεκριμένο υπολογιστή. Ένας καλός αλγόριθμος δημιουργίας κώδικα μπορεί να παράγει κώδικα που "τρέχει" πολύ πιο γρήγορα από άλλον ισοδύναμο κώδικα ο οποίος παρήχθη από κάποιον άλλο όχι πολύ ευέλικτο αλγόριθμο. Η είσοδος του Δημιουργού Κώδικα, (ΔΚ), υποθέτουμε ότι θα είναι κάποια μορφή ενδιάμεσου κώδικα όπως τετράδες, τριάδες, δένδρο ή postfix μορφή τις οποίες εξετάσαμε στο Κεφάλαιο 6. Η έξοδός του είναι το τελικό πρόγραμμα το οποίο μπορεί να πάρει διάφορες μορφές: απόλυτο πρόγραμμα μηχανής, relocatable πρόγραμμα μηχανής: assembly πρόγραμμα ή τέλος πρόγραμμα σε κάποια άλλη γλώσσα προγραμματισμού.

ΕΝΟΤΗΤΑ 7.1 ΕΙΣΑΓΩΓΗ (Object Προγράμματα)

Το απόλυτο (absolute) πρόγραμμα έχει το προσόν ότι μπορεί να τοποθετηθεί σε συγκεκριμένη περιοχή της μνήμης και να εκτελεσθεί αμέσως. Μεταγλωττιστές τύπου load-and-go για προγράμματα σπουδαστών παράγουν απόλυτο κώδικα.

Ο μεταθετός (Relocatable) κώδικας επιτρέπει την χωριστή μετάφραση ρουτινών όπως και την χρήση άλλων από βιβλιοθήκες (object modules), που συνδέονται μαζί σ' ένα πρόγραμμα από κάποιο πρόγραμμα διασύνδεσης (linker). Οι περισσότεροι Μεταγλωττιστές παράγουν μεταθετό κώδικα.

Το assembly πρόγραμμα διευκολύνει τον Δημιουργό Κώδικα μία και του επιτρέπει να φτιάξει συμβολικές εντολές και μακρο-εντολές (macros). Το κόστος στην περίπτωση αυτή είναι ότι πρέπει να γίνει και assembly μετάφραση μετά την δημιουργία του assembly κώδικα.

Παράγοντας κώδικα σε κάποια ψηλή γλώσσα προγραμματισμού απλοποιεί παρά πολύ την δημιουργία του κώδικα μια και την δουλειά του δημιουργού κώδικα θα την κάνει ο Μεταγλωττιστής της γλώσσας αυτής.

Συχνά είναι χρήσιμο να υποθέτουμε ότι ο ενδιάμεσος κώδικας που αποτελεί την είσοδο στον Δημιουργό Κώδικα είναι χωρισμένος σε βασικά blocks και ότι κλήσεις σε procedures προσδιορίζονται μόνο μέσα σε blocks. Με τον όρο βασικό block εννοούμε μία ακολουθία εντολών ενδιάμεσου κώδικα που ξεκινάει να εκτελείται μόνο από την αρχή και δεν περιέχει εντολές halt ή jump (εκτός από το τέλος του).

Ακόμη υποθέτουμε ότι δεν είναι απαραίτητη κάποια φάση βελτιστοποίησης του ενδιάμεσου κώδικα.

Ο τελικός κώδικας εξαρτάται πάρα πολύ από τον συγκεκριμένο υπολογιστή και το λειτουργικό του σύστημα. Έστω κατ' αρχήν ότι ο τελικός κώδικας είναι γλώσσα μηχανής. Δεν υπάρχει πρόβλημα στην κωδικοποίηση των τελεστών σε λειτουργίες της μηχανής (υπολογιστή). Κάποια προβλήματα παρουσιάζονται στην εκλογή των διευθύνσεων των έντελων (operands). Υποθέτουμε ότι έχει ήδη προσδιοριστεί (πριν την δημιουργία του κώδικα) η κάθε περιοχή δεδομένων (data area) και οι σχετικές θέσεις (offsets) κάθε ονόματος στην περιοχή. Εν γένει αν κάποιο όνομα Α έχει θέση f, τότε μία εντολή μηχανής Ι για την οποία το Α είναι έντελο θα περιέχει f στο πεδίο διεύθυνσής της. Μερικά άλλα bits της Ι όμως θα εξαρτώνται από την φύση της περιοχής δεδομένων. Για παράδειγμα, αν το Α είναι σε στατική περιοχή και δημιουργούμε μεταθετό κώδικα, τότε το μόνο που χρειάζεται είναι να επισυνάψουμε στην εντολή Ι μία ένδειξη για τον φορτωτή (loader) ότι η διεύθυνση της Ι πρέπει να μετασχηματιστεί προσθέτοντας το f στην πρώτη εντολή μηχανής που αντιστοιχεί στην procedure στην οποία ανήκει το Α.

Αν η στοχευόμενη μηχανή χρησιμοποιεί καταχωρητές βάσης (base registers), δεν θέλουμε να μετασχηματίσουμε την διεύθυνση της Ι, αλλά πρέπει να βάλουμε σε κατάλληλα bits της Ι τον αριθμό του καταχωρητή βάσης που έχει επιλεγεί για την περιοχή δεδομένων. Τέλος αν δημιουργούμε κώδικα μηχανής, τότε ο Μεταγλωττιστής πρέπει να προσθέσει την διεύθυνση αρχής της περιοχής δεδομένων του Α, την στιγμή που η εντολή Ι δημιουργείται.

Άσκηση αυτοαξιολόγησης 1 / Κεφ. 7

Προσπάθησε να συσχετίσεις τις σωστές απαντήσεις της δεξιάς στήλης.

Ο μεταθετός κώδικας		διευκολύνει πολύ τον Δημιουργό Κώδικα
Το assembly πρόγραμμα		επιτρέπει την χωριστή μετάφραση ρουτινών
Το απόλυτο πρόγραμμα		έχει το προσόν ότι μπορεί να τοποθετηθεί σε συγκεκριμένη περιοχή της μνήμης και να εκτελεσθεί αμέσως

ΕΝΟΤΗΤΑ 7.2 ΠΡΟΒΛΗΜΑΤΑ ΚΑΤΑ ΤΗΝ ΔΗΜΙΟΥΡΓΙΑ ΤΕΛΙΚΟΥ ΚΩΔΙΚΑ

Υπάρχουν τρεις περιοχές δυσκολίας, που αναφέρονται σε

- τι εντολές μηχανής να δημιουργήσουμε,
- με ποια σειρά πρέπει να γίνουν οι υπολογισμοί και
- ποιοι καταχωρητές θα χρησιμοποιηθούν.

Τι εντολές θα δημιουργήσουμε, εξαρτάται από το ρεπερτόριο εντολών της στοχευόμενης μηχανής.

Παράδειγμα 1 / Κεφ. 7

Για παράδειγμα, αν η μηχανή διαθέτει εντολή "πρόσθεσης-στην-μνήμη" (ADM), τότε η εντολή τριών διευθύνσεων A:=A+1 μπορεί να δημιουργηθεί με μία εντολή

ADM A # 1 αντί της ακολουθίας εντολών

LOAD A; ADD # l; STORE A

Η σειρά εκτέλεσης των εντολών μπορεί να έχει επίδραση στο πλήθος των καταχωρητών για ενδιάμεσα αποτελέσματα. Εμείς, κατ' αρχήν θα δημιουργούμε κώδικα για τις εντολές τριών διευθύνσεων με την σειρά με την οποία αυτές φτιάχτηκαν από τις σημασιολογικές ρουτίνες.

Τέλος θα αναφερθούμε στο πρόβλημα εκχώρησης καταχωρητών για τους υπολογισμούς, το οποίο γίνεται πιο πολύπλοκο σε μερικές μηχανές που απαιτούν ζεύγη καταχωρητών για ορισμένες πράξεις κυρίως αριθμητικές. Για παράδειγμα, η εντολή διαίρεσης DIV A,B απαιτεί ο διαιρετέος να βρίσκεται σ' ένα ζευγάρι καταχωρητών από τους οποίους ο πρώτος είναι ο A και έχει άρτια αρίθμηση (π.χ. 6), B παριστάνει τον διαιρέτη. Μετά την διαίρεση ο άρτιος καταχωρητής (A) περιέχει το υπόλοιπο και ο γειτονικός του περιττός το πηλίκο.

ΕΝΟΤΗΤΑ 7.3 ΕΝΑ ΜΟΝΤΕΛΟ ΥΠΟΛΟΓΙΣΤΗ

Όπως εξηγήσαμε προηγούμενα για την δημιουργία τελικού κώδικα χρειάζεται να αναφερθούμε σε συγκεκριμένη μηχανή. Για το λόγο αυτό θα περιγράψουμε μια μηχανή και το ρεπερτόριο εντολών της. Η μηχανή μας υποθέτουμε ότι μπορεί να προσπελάσει την μνήμη της κατά bytes και έχει συνολική μνήμη 65k bytes σε λέξεις των 16-bit. Διαθέτει οκτώ καταχωρητές γενικής χρήσης R0, R1,..,R7 με χωρητικότητα 16-bit ο καθένας. Διαθέτει δυαδικούς τελεστές της μορφής:

OP προέλευση, προορισμός

όπου κάθε OP είναι κώδικας λειτουργίας των 4-bit και προέλευση και προορισμός είναι πεδία διευθύνσεων των 6-bit το καθένα.

Μια και τα πεδία αυτά των 6-bit δεν είναι αρκετά μεγάλα για να επιτρέπουν άμεση προσπέλαση όλης της μνήμης, ορισμένοι συνδυασμοί από bits στα πεδία αυτά καθορίζουν ότι ορισμένες λέξεις που ακολουθούν μία εντολή θα περιέχουν έντελα και/ή διευθύνσεις. Υποθέτουμε τους παρακάτω τρόπους προσπέλασης διευθύνσεων, που παρουσιάζονται στην μνημονική τους μορφή της assembly γλώσσας (operand addressing modes).

1. r (register mode). Ο καταχωρητής r περιέχει το έντελο.
2. *r (indirect register mode). Ο καταχωρητής r περιέχει την διεύθυνση στην οποία βρίσκεται το έντελο.
3. X(r) (Indexed mode). Η διεύθυνση του έντελου βρίσκεται προσθέτοντας την τιμή X, που βρίσκεται στην επόμενη λέξη μετά την εντολή, και το περιεχόμενο του καταχωρητή r.

4. * X(r) (indirect indexed mode). Η τιμή X, που βρίσκεται στην λέξη που ακολουθεί την εντολή, προστίθεται στα περιεχόμενα του r για να δώσει την διεύθυνση της λέξης που περιέχει την διεύθυνση του έντελου.
5. #X (immediate). Η λέξη που ακολουθεί την εντολή περιέχει το έντελο.
6. X (absolute). Η διεύθυνση του X ακολουθεί την εντολή.

Μεταξύ άλλων θα χρησιμοποιήσουμε τους παρακάτω κώδικες λειτουργίας.

MOV (μετακίνησε την προέλευση στον προορισμό)

ADD (πρόσθεσε την προέλευση στον προορισμό)

SUB (αφαίρεσε την προέλευση από τον προορισμό)

Με μερικά παραδείγματα θα ξεκαθαρίσουμε την έννοια του μήκους (σε λέξεις) των εντολών. Θα θεωρήσουμε ότι το κόστος μίας εντολής εκφράζεται από το μήκος της σε λέξεις.

Παράδειγμα 2 / Κεφ. 7

Ακολουθούν παραδείγματα εντολών μηχανής.

- MOV R0, R1 μεταφέρει τα περιεχόμενα του R0 στην R1 και έχει κόστος 1, μία και απασχολεί μία μόνο θέση μνήμης.
- MOV R5, M μεταφέρει τα περιεχόμενα του R5 στην θέση μνήμης M και έχει κόστος 2, μία και η διεύθυνση της θέσης μνήμης M βρίσκεται στην λέξη που ακολουθεί την εντολή (absolute addressing).

Άσκηση αυτοαξιολόγησης 2 / Κεφ. 7

Βρείτε τι κόστος έχει η κάθε μια από τις παρακάτω εντολές και εξηγήστε γιατί.

1. ADD #1, R3
2. SUB 4(R0), *5(R1)

Δραστηριότητα 1 / Κεφ. 7

Για να αντιληφθούμε καλύτερα μερικές από τις δυσκολίες στην δημιουργία κώδικα, για αυτή τη μηχανή ας υποθέσουμε ότι θέλουμε να φτιάξουμε κώδικα για μία τετράδα της μορφής A:=B+C όπου τα B και C είναι απλές μεταβλητές σε συγκεκριμένες θέσεις μνήμης με τα ίδια ονόματα. Προσπαθήστε, χωρίς να κοιτάξετε την συνέχεια του κειμένου, να καταγράψετε διάφορες ακολουθίες εντολών που μπορούμε να δημιουργήσουμε και με τι κόστος για την κάθε ακολουθία.

Μπορούμε να δημιουργήσουμε τις παρακάτω ακολουθίες εντολών με το αντίστοιχο κόστος τους, υποθέτοντας ότι οι καταχωρητές R1 και R2 περιέχουν τις τιμές των B και C αντίστοιχα και η τιμή του B δεν χρειάζεται μετά την εκχώρηση.

1. MOV B, R0
   ADD C, R0
   MOV R0, A Κόστος 6
   ==========
2. MOV B, A
   ADD C, A Κόστος 6
   ==========
3. MOV *R1, *R0
   ADD *R2, *R0 Κόστος 2

Και άλλες ακολουθίες εντολών είναι δυνατές. Για παράδειγμα, καλύτερος κώδικας μπορεί να παραχθεί για την εντολή A:=B+C αν δημιουργήσουμε την απλή εντολή ADD Rj, Ri, με κόστος 1, και αφήσουμε το αποτέλεσμα A στον Ri. Αυτό είναι δυνατό μόνο όταν ο Ri περιέχει το B, ο Rj περιέχει το C και το B δεν χρειάζεται (not live) μετά την εντολή.

Παρατηρούμε λοιπόν ότι για να παράγουμε "καλό" κώδικα για αυτή τη μηχανή (όπως και τις περισσότερες) πρέπει να κάνουμε καλή χρήση στις δυνατότητές της για προσπέλαση διευθύνσεων.

Άσκηση αυτοαξιολόγησης 3 / Κεφ. 7

Σε συνέχεια των παραπάνω, αν υποθέσουμε ότι ο καταχωρητής Ri περιέχει το B και ότι το C είναι σε θέση μνήμης με όνομα C μπορείτε να περιγράψετε μια ή περισσότερες ακολουθίες κώδικα, με το αντίστοιχο κόστος τους, για την A:=B+C; Υπόδειξη: η χρήση των καταχωρητών μειώνει το κόστος των ακολουθιών.

ΕΝΟΤΗΤΑ 7.4 ΕΝΑΣ ΑΠΛΟΣ ΔΗΜΙΟΥΡΓΟΣ ΚΩΔΙΚΑ

Θα εξετάσουμε ένα απλοποιημένο ΔΚ κάνοντας τις παρακάτω υποθέσεις:

1) χρησιμοποιούνται τετράδες σαν είσοδος στον ΔΚ,
2) ο ΔΚ "θυμάται" αν κάποιο από τα έντελα της τετράδας είναι σε κάποιο καταχωρητή,
3) για κάθε τελεστή υπάρχει αντίστοιχος κώδικας μηχανής και,
4) υπολογιζόμενα αποτελέσματα μπορούν να παραμένουν σε καταχωρητές για όσο χρειάζεται και αποθηκεύονται αλλού όταν οι καταχωρητές χρειάζονται για άλλες πράξεις ή όταν βρισκόμαστε πριν από κλήση procedure, jump ή εντολή με επιγραφή.

Στην συνέχεια θα συζητήσουμε μερικές διευκολύνσεις που χρειαζόμαστε για τον αλγόριθμο του Δ.Κ.

Επόμενη Χρήση ονομάτων σε τετράδες που καθορίζεται ως εξής: Έστω ότι η τετράδα i εκχωρεί μία τιμή στο A. Αν η τετράδα j έχει το A σαν έντελο και ο έλεγχος μπορεί να περάσει από το i στο j, χωρίς ενδιάμεσα να υπάρχουν εκχωρήσεις στο A, τότε λέμε ότι η τετράδα j "χρησιμοποιεί" την τιμή του A που υπολογίστηκε στο j.

Χρειαζόμαστε να προσδιορίσουμε για κάθε τετράδα A:=B op C ποιες είναι οι επόμενες χρήσεις των A, B και C και για διευκόλυνση εξετάζουμε χρήσεις μόνο μέσα στο βασικό block στο οποίο βρίσκεται η τετράδα.

Ο αλγόριθμος που καθορίζει τις επόμενες χρήσεις κάνει μία οπισθοανίχνευση μέσα στο βασικό block. Έστω ότι σε κάποιο σημείο φθάνουμε στην τετράδα

i: A:=B op C.

Στην συνέχεια κάνουμε τα εξής:

1. Επισυνάπτουμε στην τετράδα i τις πληροφορίες που βρήκαμε στον Πίνακα συμβόλων σχετικά με την επόμενη χρήση και την διάρκεια ζωής (liveness) των A,B και C.
2. Θέτουμε στον πίνακα συμβόλων, για το A, τις ενδείξεις: "νεκρό" και "δεν έχει επόμενη χρήση".
3. Στον πίνακα συμβόλων θέτουμε στα B και C ότι είναι "ζωντανά" και i στις "επόμενες χρήσεις" τους. Σημειώνουμε ότι τα βήματα (2) και (3) δεν μπορούν να αντιμετατεθούν διότι το A μπορεί να είναι το ίδιο με το B ή το C.

Αν η τετράδα έχει την μορφή A:=B ή A:=op B, τα βήματα παραμένουν όπως είναι και αγνοείται το C.

Περιγραφέας Καταχωρητών (Register Descriptor). Για να κάνουμε εκχώρηση καταχωρητών (register allocation) πρέπει να διατηρούμε ένα περιγραφέα καταχωρητών ο οποίος γνωρίζει τι βρίσκεται κάθε στιγμή μέσα στους καταχωρητές. Συμβουλευόμαστε τον περιγραφέα κάθε φορά που χρειαζόμαστε ένα καινούριο καταχωρητή. Υποθέτουμε ότι αρχικά ο περιγραφέας δείχνει ότι όλοι οι καταχωρητές είναι κενοί (διαθέσιμοι). Καθώς προχωράει η διαδικασία δημιουργίας κώδικα για το βασικό block κάθε καταχωρητής μπορεί να κρατάει, την τιμή κανενός ή πολλών ονομάτων.

Περιγραφέας διευθύνσεων (Address Descriptor). Για κάθε όνομα μέσα στο βασικό block θα διατηρούμε ένα περιγραφέα διεύθυνσης ο οποίος γνωρίζει την θέση (ή θέσεις) στην οποία βρίσκεται η τρέχουσα τιμή του ονόματος κατά τον χρόνο εκτέλεσης. Η θέση αυτή μπορεί να είναι ένας καταχωρητής, μία θέση σε μια στοίβα, μία διεύθυνση μνήμης ή κάποιο σύνολο από αυτά. Η πληροφορία αυτή μπορεί να φυλάσσεται στον πίνακα συμβόλων και χρησιμοποιείται για να καθορίσει τον τρόπο προσπέλασης του ονόματος.

Άσκηση αυτοαξιολόγησης 4 / Κεφ. 7

Σαν ορισμό για την *επόμενη χρήση* ονόματος μπορούμε να θεωρήσουμε τον εξής: "Ένα όνομα A σε μια τετράδα i έχει επόμενη χρήση στην τετράδα j άν εμφανίζεται στην τετράδα αυτή (j)". Διορθώστε, συμπληρώστε ή αλλάξτε τον ορισμό αυτό εάν κατά την άποψή σας δεν είναι ακριβής.

7.4.1 Ο ΑΛΓΟΡΙΘΜΟΣ ΔΗΜΙΟΥΡΓΙΑΣ ΚΩΔΙΚΑ

Περιγράφουμε τώρα τον αλγόριθμο. Έχει δοθεί μία ακολουθία τετράδων οι οποίες αποτελούν ένα βασικό block. Για κάθε τετράδα της μορφής A:=B op C εκτελούνται τα παρακάτω βήματα:

1. Κάλεσε μια function GETREG( ) για να προσδιορίσει την θέση L στην οποία θα πρέπει να εκτελεσθεί ο υπολογισμός B op C. Η L συνήθως είναι κάποιος register αλλά μπορεί να είναι και θέση μνήμης.
2. Συμβουλεύσου τον περιγραφέα διεύθυνσης του B για να προσδιορίσεις το B', την τρέχουσα θέση του B (μπορεί να είναι μία ή περισσότερες). Προτίμησε τον καταχωρητή για B' αν η τιμή του B είναι στην μνήμη και σε κάποιο καταχωρητή. Αν η τιμή του B δεν είναι στην L, δημιούργησε την εντολή MOV B', L για να βάλεις κόπια του B στην L.
3. Δημιούργησε την εντολή OP C', L όπου C' είναι η τρέχουσα θέση του C. Ενημέρωσε τον περιγραφέα διεύθυνσης του A ώστε να δείχνει ότι κατά τον χρόνο εκτέλεσης θα περιέχει την τιμή του A.
4. Αν οι τρέχουσες τιμές των B και/ή C δεν έχουν επόμενες χρήσεις, δεν είναι ζωντανές κατά την έξοδο από το block και βρίσκονται σε καταχωρητές, άλλαξε τον περιγραφέα καταχωρητών έτσι ώστε να δείχνει ότι μετά την εκτέλεση της A:=BopC, οι καταχωρητές αυτοί δεν θα περιέχουν πλέον το B και / ή το C, αντίστοιχα.

Όταν έχουμε τελειώσει με την επεξεργασία όλων των τετράδων του βασικού block, αποθηκεύουμε με εντολές MOV, εκείνα τα ονόματα που είναι ζωντανά κατά την έξοδο του block και δεν βρίσκονται στις θέσεις μνήμης τους. Για να γίνει αυτό χρησιμοποιούμε τον περιγραφέα καταχωρητών για να προσδιορίζουμε τι ονόματα έχουν μείνει στους καταχωρητές, τον περιγραφέα διεύθυνσης για να

προσδιορίσουμε αν το ίδιο όνομα βρίσκεται ήδη στην θέση μνήμης του και την πληροφορία για τις ζωντανές μεταβλητές για να καθορίσουμε αν το όνομα πρέπει να φυλαχτεί ή όχι.

Αν η τρέχουσα τετράδα έχει μοναδιαίο τελεστή, τα βήματα του αλγόριθμου είναι ανάλογα με τα παραπάνω.

Δραστηριότητα 2 / Κεφ. 7

Μία ειδική περίπτωση είναι η τετράδα A:=B. Για την περίπτωση αυτή προσπαθήστε να δώσετε περιγραφικά τι λειτουργίες θα πρέπει να κάνει ο Δημιουργός Κώδικα και τι κώδικα θα κατασκευάσει, χωρίς να κοιτάξετε την απάντηση που δίνεται στη συνέχεια.

Αν το B είναι σε κάποιο καταχωρητή, απλώς άλλαξε τους περιγραφείς καταχωρητών και διεύθυνσης έτσι ώστε να καταγράφουν ότι η τιμή του A βρίσκεται τώρα μόνο στον καταχωρητή που έχει την τιμή του B. Αν το B δεν έχει επόμενη χρήση και δεν είναι ζωντανό κατά την έξοδο του block, ο καταχωρητής δεν έχει πλέον την τιμή του B.

Αν το B είναι στην μνήμη πρέπει να χρησιμοποιήσουμε την GETREG για να βρούμε ένα καταχωρητή στον οποίο να φορτώσουμε το B και να κάνουμε τον καταχωρητή αυτό να είναι η θέση του A. Αλλιώς, μπορούμε να δημιουργήσουμε μία εντολή MOV B, A που είναι προτιμότερο αν η τιμή του A δεν έχει επόμενη χρήση στο block.

Παράδειγμα 3 / Κεφ. 7

Η έκφραση W:=(A-B)+(A-C)+(A-C)

θα μπορούσε να είχε μεταφρασθεί στην παρακάτω ακολουθία εντολών τριών διευθύνσεων.

T: = A-B
U: = A-C
V: = T+U
W: = V+U

Με το W να είναι ζωντανό στο τέλος. Ο Αλγόριθμος δημιουργίας Κώδικα θα παράγει την παρακάτω ακολουθία του σχήματος 7.1.

Εντολές	Δημιουργούμενος Κώδικας	Περιγραφέας Καταχωρητών	Περιγραφέας Διεύθυνσης
-	-	Καταχωρητές κενοί	-
T: = A-B	MOV A, R0 SUB B, R0	R0 περιέχει T	T στον R0
U: = A-C	MOV A, R1 SUB C, R1	R0 περιέχει T R1 περιέχει U	T στον R0 U στον R1
V:= T+U	ADD R1, R0	R0 περιέχει V R1 περιέχει U	U στον R1 V στον R0
W:=V+U	ADD R1, R0 MOV R0, W	R0 περιέχει W -	W στον R0 W στον R0 και στην μνήμη.

Σχήμα 7.1 Ακολουθία δημιουργηθέντα κώδικα για το block

W := (A-B) + (A-C) + (A-C)

Στο σχήμα 1 δεν φαίνεται το γεγονός ότι τα A, B και C είναι πάντοτε στην μνήμη. Ακόμη υποθέτουμε ότι οι προσωρινές μεταβλητές T, U και V δεν βρίσκονται στην μνήμη εκτός εάν τις βάλουμε εμείς χρησιμοποιώντας εντολές MOV.

Η πρώτη κλήση της GETREG επιστρέφει R0 σαν τη θέση στην οποία θα γίνει ο υπολογισμός του T. Μία και το A δεν βρίσκεται στον R0 δημιουργούμε εντολές MOV A, R0 και SUB B, R0. Στην συνέχεια ενημερώνουμε τον περιγραφέα καταχωρητών να δείχνει ότι ο R0 περιέχει T. Η δημιουργία του κώδικα προχωράει έτσι μέχρι και την τελευταία τετράδα. Σημειώνουμε ότι ο R1 εκκενώνεται διότι το U δεν έχει επόμενη χρήση. Τέλος δημιουργούμε MOV R0, W για να φυλάξουμε την ζωντανή μεταβλητή W κατά το τέλος του block.

7.4.2 ΔΗΜΙΟΥΡΓΙΑ ΚΩΔΙΚΑ ΓΙΑ ΑΛΛΟΥΣ ΤΥΠΟΥΣ ΕΝΤΟΛΩΝ

Έστω ότι έχει εκχωρηθεί στατική μνήμη για το B και έχουμε την δεικτοδοτημένη εντολή A: = B[I]. Αυτή μπορεί να υλοποιηθεί επιλέγοντας κάποιο καταχωρητή L για το A, με την βοήθεια της GETREG και στην συνέχεια εκτελώντας τις:

```
MOV I', L
MOV B(L), L          με κόστος 4
```

αν το I δεν βρίσκεται σε καταχωρητή και I' είναι μία θέση (μνήμη ή stack) για το I. Αν το I βρίσκεται στον R0 εκδίδουμε την:

```
MOV B(R0), L         με κόστος 2
```

Παρόμοια η A[I] := B υλοποιείται με τις

```
MOV I', L
MOV B,A(L)           με κόστος 4
```

εφ' όσον το I δεν βρίσκεται σε καταχωρητή και I' είναι μία θέση για το I. Ενώ αν το I βρίσκεται στον R0 έχουμε:

```
MOV B, A(R0)                 με κόστος 2
```

Για την δημιουργία κώδικα για άλλους τύπους εντολών μπορείτε να συμβουλευθείτε την βιβλιογραφία. Σε κάθε περίπτωση, η αρχιτεκτονική δομή του υπολογιστή αποτελεί τον καθοριστικότερο παράγοντα στη δημιουργία τελικού κώδικα.

ΕΝΟΤΗΤΑ 7.5 ΕΚΧΩΡΗΣΗ ΚΑΤΑΧΩΡΗΤΩΝ

Οι εντολές μηχανής που κάνουν χρήση μόνο καταχωρητών είναι μικρότερες (σε πλήθος λέξεων) και πιο γρήγορες από εκείνες που κάνουν αναφορά σε έντελα. Είναι επομένως προφανές ότι η καλύτερη χρήση των καταχωρητών θα αποδώσει καλύτερο τελικό κώδικα. Γι αυτό και ενδιαφερόμαστε για τεχνικές οι οποίες καθορίζουν ποια ονόματα ενός προγράμματος θα μπουν σε καταχωρητές (Register allocation) και ποιος καταχωρητής θα εκχωρηθεί σε ποιο όνομα (register assignment). Μια προσέγγιση στο πρόβλημα αυτό του καταμερισμού και εκχώρησης των καταχωρητών είναι να εκχωρήσουμε συγκεκριμένους τύπους ποσοτήτων σε ένα τελικό πρόγραμμα σε ορισμένους καταχωρητές. Για παράδειγμα, μπορούμε να αποφασίσουμε ότι θα εκχωρήσουμε μία ομάδα καταχωρητών για συνδέσεις υποπρογραμμάτων, μία άλλη ομάδα για καταχωρητές βάσης, άλλη για

αριθμητικούς υπολογισμούς κάποιο καταχωρητή για να περιέχει τον δείκτη της στοίβας εκτέλεσης (run-time stack) και λοιπά.

Η προσέγγιση αυτή έχει το πλεονέκτημα ότι απλοποιεί τον σχεδιασμό ενός μεταγλωττιστή, από την άλλη όμως μεριά εφαρμοζόμενη με αυστηρότητα έχει το μειονέκτημα ότι κάποιοι καταχωρητές μένουν αχρησιμοποίητοι για μεγάλα διαστήματα του τελικού κώδικα ενώ δημιουργεί και πολλές άχρηστες φορτώσεις και εκφορτώσεις (εντολές LOAD και STORE ή MOV για την μηχανή που περιγράψαμε).

Παρά την σπουδαιότητα του προβλήματος καταμερισμού και εκχώρησης των καταχωρητών δεν θα ασχοληθούμε περισσότερο μαζί του στο βιβλίο αυτό.

ΕΝΟΤΗΤΑ 7.6 ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ PEEPHOLE

Αποτελεί μία τεχνική βελτιστοποίησης που συναντάται σε αρκετούς Μεταγλωττιστές, για να ξεπεράσει τις δυσκολίες που συναντώνται στην συντακτικά κατευθυνόμενη δημιουργία ενδιάμεσου ή object κώδικα. Η μέθοδος δουλεύει εξετάζοντας μικρές περιοχές του κώδικα (peepholes), ο οποίος δεν είναι απαραίτητο να είναι συνεχής. Χαρακτηριστικό της μεθόδου είναι το γεγονός ότι κάθε βελτιστοποίηση δημιουργεί συνθήκες για πάρα πέρα βελτιώσεις. Γι αυτό η μέθοδος δουλεύει με διαδοχικά περάσματα πάνω στον κώδικα.

Περιττές φορτώσεις και εκφορτώσεις.

Για παράδειγμα, στην ακολουθία.

1. MOV R0, A
2. MOV A, R0

μπορούμε να διαγράψουμε την εντολή (2), εκτός εάν η (2) έχει κάποια επιγραφή. Είναι ασφαλές να διαγράψουμε την (2) μόνο όταν αποτελούν μέρος ενός βασικού block. Κώδικας όπως η παραπάνω ακολουθία (1) και (2) δημιουργείται αν μεταφράζουμε πάντοτε τις εντολές του τύπου A:=B op C σε MOV B, R0; op C, R0; MOV R0, A. Τότε η μετάφραση των A:=B+C και D:=A+B θα δημιουργούσε τέτοιο κώδικα.

Μη προσπελάσιμος κώδικας.

Μπορούμε, και πρέπει, να αποσύρουμε κώδικα ο οποίος δεν προσπελαύνεται και ο οποίος δημιουργήθηκε είτε από τον προγραμματιστή, είτε και έμμεσα από τον ίδιο τον μεταγλωττιστή στα πλαίσια άλλων βελτιστοποιήσεων (πχ σαν αποτέλεσμα της μετάδοσης σταθερών –constant propagation). Σε τέτοια περίπτωση καταλήγουμε να έχουμε κώδικα της μορφής

```
        Goto L1
        /* μη προσπελάσιμος κώδικας  */
    L1:
```

ο οποίος πρέπει φυσικά να αποσυρθεί μια και δεν εκτελείται ποτέ.

Πολλαπλά Άλματα.

Δεν είναι ασυνήθιστο αλγόριθμοι δημιουργίας ενδιάμεσου κώδικα να δημιουργούν εντολές αλλαγής ροής (goto) σε άλλες εντολές αλλαγής ροής, υπό συνθήκη ή μη. Μπορούμε να απαλλαγούμε από αυτά στον ενδιάμεσο κώδικα αντικαθιστώντας:

```
goto L1
```

...

L1:goto L2

με το:

goto L2

...

L1: goto L2

Αν τώρα δεν υπάρχουν άλματα στην L1 (μπορούμε να μετράμε το πλήθος των jumps στην θέση του πίνακα συμβόλων που αντιστοιχεί στην label αυτή) μπορούμε να εξαλείψουμε την L1:goto L2, με την προϋπόθεση ότι προηγείται κάποιο jump χωρίς συνθήκη.

Παρόμοια, η

if A<B goto L1

...

L1: goto L2

μπορεί να αντικατασταθεί από την:

if A<B goto L2

...

L1: goto L2

Άσκηση αυτοαξιολόγησης 5 / Κεφ. 7

Σε αρκετές περιπτώσεις διάφοροι Δημιουργοί Κώδικα δημιουργούν κώδικα όπως:

1) A:=A+0
2) A:=A*1
3) A:=2*A
4) A:=B^2

Μπορείτε να σκεφθείτε και να προτείνετε βελτιστοποιήσεις για τέτοιου είδους κώδικα;

ΣΥΝΟΨΗ ΚΕΦΑΛΑΙΟΥ

Στο κεφάλαιο αυτό μελετήσαμε την δημιουργία τελικού κώδικα. Είδαμε ένα απλό υπολογιστή, περιγράψαμε μερικές εντολές από το ρεπερτόριο εντολών του και τρόπους προσπέλασης των διευθύνσεών του. Εξηγήσαμε την έννοια του κόστους μιας εντολής, σαν τον αριθμό λέξεων που χρειάζονται για την εσωτερική αναπαράσταση της εντολής. Στη συνέχεια, για να μπορέσουμε να περιγράψουμε ένα αλγόριθμο δημιουργίας κώδικα, εξηγήσαμε την επόμενη χρήση ονομάτων μέσα σε ένα βασικό block, τον περιγραφέα καταχωρητών και τον περιγραφέα διευθύνσεων. Δώσαμε τον αλγόριθμο δημιουργίας κώδικα για μια εντολή τριών διευθύνσεων γενικής μορφής A:=BopC και αναφερθήκαμε σύντομα στην δημιουργία κώδικα για άλλους τύπους εντολών. Τέλος, αναφερθήκαμε στην βελτιστοποίηση peephole και είδαμε πώς μπορούμε να απαλλαγούμε από περιττές εντολές, μη προσπελάσιμο κώδικα, πολλαπλά άλματα και πώς να εκμεταλλευθούμε τις ιδιομορφίες του υπολογιστή.

ΑΠΑΝΤΗΣΕΙΣ ΑΣΚΗΣΕΩΝ ΑΥΤΟΑΞΙΟΛΟΓΗΣΗΣ

Απάντηση άσκησης αυτοαξιολόγησης 1 / Κεφ. 7

Η σωστή συσχέτιση της αριστερής με την δεξιά στήλη δίνεται παρακάτω

Ο μεταθετός κώδικας		διευκολύνει τον Δημιουργό Κώδικα
Το assembly πρόγραμμα		επιτρέπει την χωριστή μετάφραση ρουτινών
Το απόλυτο πρόγραμμα		έχει το προσόν ότι μπορεί να τοποθετηθεί σε συγκεκριμένη περιοχή της μνήμης και να εκτελεσθεί αμέσως

Αν η απάντησή σας είναι ίδια με την παραπάνω, μπράβο σας. Αν όμως κάνατε διαφορετικές συσχετίσεις τότε πρέπει να ξαναδιαβάσετε πιο προσεκτικά την ενότητα 7.1.

Απάντηση άσκησης αυτοαξιολόγησης 2 / Κεφ. 7

1. Η εντολή ADD #1, R3 προσθέτει το 1 στα περιεχόμενα του καταχωρητή R3 και έχει κόστος 2 λέξεων. Η δεύτερη λέξη περιέχει τον ακέραιο αριθμό 1.
2. Η εντολή SUB 4(R0), *5(R1) αφαιρεί το ((R0) +4) από το (((R1) +5)), όπου (x) δηλώνει το περιεχόμενο του καταχωρητή ή της θέσης X. Το αποτέλεσμα αποθηκεύεται στην διεύθυνση προορισμού *5(R1). Το κόστος της είναι 3 λέξεις μία και οι σταθερές 4 και 5 φυλάσσονται στις επόμενες δύο λέξεις μετά την εντολή.

Αν η απάντησή σας είναι ίδια με την παραπάνω, συγχαρητήρια, έχετε κατανοήσει τόσο την έννοια του κόστους μιας εντολής, όσο και τους τρόπους προσπέλασης διευθύνσεων του υποθετικού υπολογιστή. Αν η απάντησή σας είναι διαφορετική, τότε θα πρέπει να ξαναδιαβάσετε την ενότητα 7.3 και να προσπαθήσετε πάλι.

Απάντηση άσκησης αυτοαξιολόγησης 3 / Κεφ. 7

Με τις υποθέσεις της άσκησης μπορούμε να δημιουργήσουμε την ακολουθία

```
ADD C, Ri
```

η οποία έχει κόστος 2 λέξεων, ή την ακολουθία

```
MOV C, Rj
ADD Rj, Ri
```

με κόστος 3 και με την υπόθεση ότι το B δεν χρειάζεται στην συνέχεια (είναι νεκρό). Η δεύτερη ακολουθία γίνεται ελκυστική αν η τιμή του C πρόκειται να ξαναχρησιμοποιηθεί στην συνέχεια (μέσα στο βασικό block) οπότε μπορούμε να την πάρουμε από τον Rj. Εν γένει, υπάρχει μεγάλος αριθμός από διαφορετικές περιπτώσεις που πρέπει να εξετάζονται κατά την δημιουργία κώδικα.

Απάντηση άσκησης αυτοαξιολόγησης 4 / Κεφ. 7

Ο σωστός ορισμός πρέπει να συμπληρωθεί με τα εξής

1. Οι τετράδες πρέπει να βρίσκονται μέσα στο ίδιο βασικό block,
2. Η τετράδα j έπεται της i,
3. Το A στην τετράδα j πρέπει να μην είναι διεύθυνση προορισμού της τετράδας j διότι τότε θα είναι νεκρό και δεν θα έχει επόμενη χρήση (οπότε θα πρέπει να εκδοθεί εντολή καταχώρησης στο A).

Απάντηση άσκησης αυτοαξιολόγησης 5 / Κεφ. 7

Για τις δύο πρώτες περιπτώσεις μπορούμε προφανώς να εξαλείψουμε τις εντολές αυτές.

Στην τρίτη περίπτωση, πολλαπλασιασμού επί δύο όπως στην A:=2*A μπορούμε να αντικαταστήσουμε τον πολλαπλασιασμό με πρόσθεση (A:=A+A) ή με εντολή μετατόπισης (shift) αν το έντελο Α είναι σταθερής υποδιαστολής.

Για την τέταρτη περίπτωση σημειώνουμε ότι ορισμένες εντολές μηχανής θεωρούνται "φθηνότερες" από άλλες. Για παράδειγμα είναι φθηνότερο να υλοποιήσουμε την A^2 με A*A αντί μίας κλήσης σε κάποια ρουτίνα ύψωσης σε δύναμη.

Παρόμοια πολλαπλασιασμός σταθερής υποδιαστολής ή διαίρεση με μία δύναμη του δύο είναι προτιμότερο να υλοποιηθεί με εντολή μετατόπισης.

Με την ευκαιρία να σημειώσουμε ότι στις περισσότερες περιπτώσεις δημιουργίας κώδικα μπορούμε να εκμεταλλευθούμε ιδιομορφίες του υπολογιστή. Κάποιοι υπολογιστές μπορεί να διαθέτουν hardware εντολές για την υλοποίηση συγκεκριμένων λειτουργιών αποδοτικότερα. Η χρήση αυτών των εντολών μπορεί να βελτιώσει τον χρόνο εκτέλεσης του προγράμματος σημαντικά. Για παράδειγμα υπάρχουν εντολές που προσθέτουν ή αφαιρούν την σταθερά 1(ένα) στο (από) το έντελο πριν ή μετά την χρήση της τιμής του. Η χρήση της δυνατότητας αυτής βελτιώνει σημαντικά την ποιότητα του κώδικα όταν μπαίνει ή βγαίνει ένα στοιχείο σε κάποια στοίβα, όπως στην περίπτωση που περνάμε τις παραμέτρους μιας ρουτίνας. Ακόμη μπορούμε να κάνουμε χρήση της δυνατότητας αυτής στον κώδικα για εντολές της μορφής X:=X+1.