Μεταγλωττιστές

Περιβάλλον Εκτέλεσης και Παραγωγή Τελικού Κώδικα

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Περιβάλλον Εκτέλεσης

Καθώς εκτελείται ένα πρόγραμμα, το ίδιο όνομα στον πηγαίο κώδικα μπορεί να αναφέρεται σε διαφορετικό αντικείμενο στην μηχανή.

Η διαχείριση αυτών των αντικειμένων γίνεται κατά την διάρκεια της εκτέλεσης του προγράμματος, από κώδικα που φορτώνεται μαζί με τον τελικό κώδικα.

Ενεργοποίηση
Activation

1. Η εκτέλεση μιας συνάρτησης ονομάζεται **ενεργοποίηση** (activation).
2. Καθώς όλες οι σύγχρονες γλώσσες υποστηρίζουν αναδρομή, πολλές ενεργοποιήσεις μπορεί να είναι ζωντανές την ίδια στιγμή.

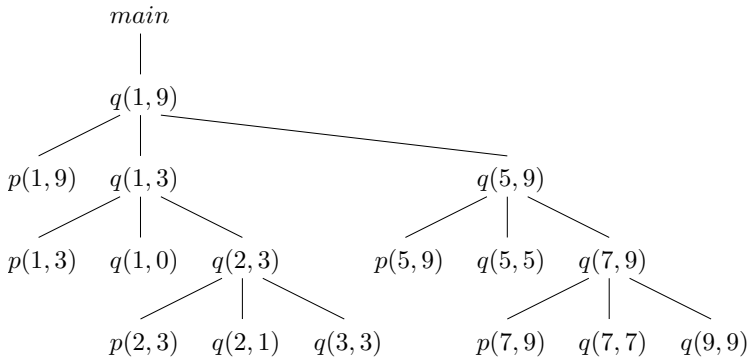
Δέντρα Ενεργοποίησης

Έστω το παρακάτω πρόγραμμα που υλοποιεί τον αλγόριθμο quicksort.

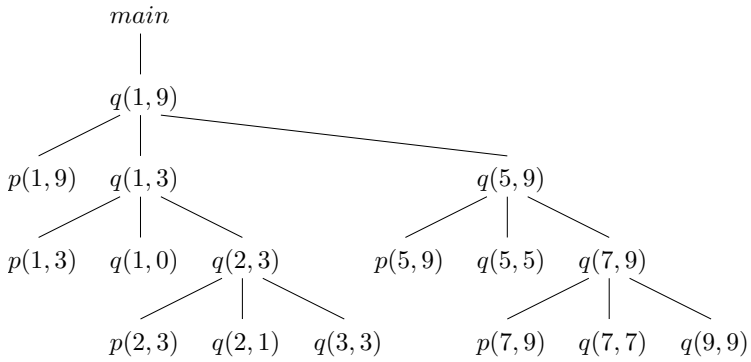
```
int partition( int l, int r ) {
    /* partition a global array */
    /* return the final position of the pivot */
}

void quicksort( int l, int r ) {
    int i;
    if ( r > l ) {
        i = partition( l, r );
        quicksort( l, i-1 );
        quicksort( i+1, r );
    }
}
```

Και έστω πως καλούμε την quicksort από την συνάρτηση main ως `quicksort`(1,9).

Δέντρα Ενεργοποίησης

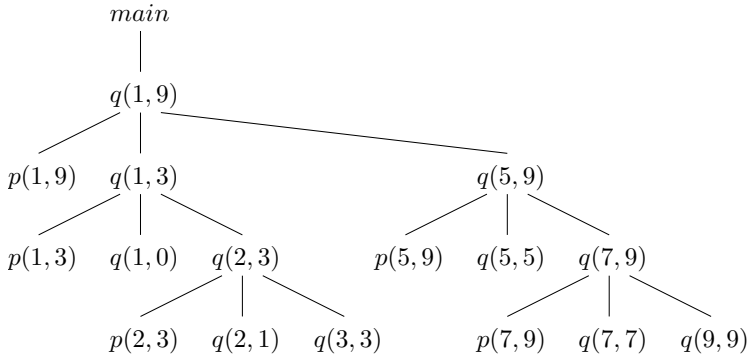


Δέντρα Ενεργοποίησης



Η ροή ελέγχου ενός προγράμματος αντιστοιχεί σε μια DFS διάσχιση του δέντρου ενεργοποίησης που ξεκινά στην ρίζα, επισκέπτεται ένα κόμβο πριν από τα παιδιά του, και επισκέπτεται τα παιδιά αναδρομικά από αριστερά προς τα δεξιά.

Δέντρα Ενεργοποίησης



Η διαχείριση των ζωντανών ενεργοποιήσεων μπορεί να γίνει με μια στοίβα, που ονομάζεται **στοίβα ελέγχου** (control stack).

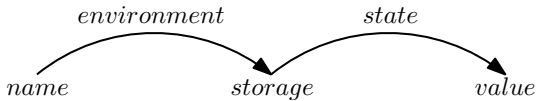
(i) Ωθούμε στην στοίβα μια εγγραφή για την ενεργοποίηση καθώς ξεκινά η ενεργοποίηση και απωθούμε την εγγραφή ως τερματίζει η ενεργοποίηση.

(ii) Με αυτό τον τρόπο η στοίβα περιέχει ανά πάσα στιγμή τους κόμβους του μονοπατιού από την τωρινή ενεργοποίηση μέχρι την ρίζα.

Συσχετισμός Ονομάτων

Bindings of Names

Ακόμη και αν ένα όνομα εμφανίζεται μια φορά σε ένα πρόγραμμα, μπορεί κατά την διάρκεια της εκτέλεσης να αναφέρεται σε διαφορετικά αντικείμενα.



Στην σημασιολογία των γλωσσών προγραμματισμού, ο όρος *περιβάλλον (environment)* αναφέρεται σε μια συνάρτηση που συσχετίζει ένα όνομα σε μία θέση αποθήκευσης, ενώ ο όρος *κατάσταση (state)* συσχετίζει μία θέση αποθήκευσης σε μια τιμή.

Συσχετισμός Ονομάτων

Περιβάλλον και κατάσταση είναι διαφορετικές έννοιες.

Μια ανάθεση όπως `pi` = 3.14 σε μια μεταβλητή `pi` αλλάζει την κατάσταση αλλά όχι και το περιβάλλον.

(i) Όταν το περιβάλλον συσχετίζει μια θέση αποθήκευσης *s* με ένα όνομα *x*, λέμε πως το *x* είναι ”δεμένο” με την *s* (bound).

(ii) Η σχέση των δύο αναφέρεται ως η συσχέτιση του *x* (binding).

Η έννοια του binding είναι η δυναμική μορφή μιας δήλωσης. Ένα όνομα τοπικής μεταβλητή σε μια συνάρτηση της C π.χ είναι ”δεμένο” σε διαφορετική θέση αποθήκευσης σε κάθε ενεργοποίηση της συνάρτησης.

Συσχετισμός Ονομάτων

στατική έννοια	δυναμική έννοια
ορισμός μια συνάρτησης	ενεργοποίηση συνάρτησης
δήλωση μεταβλητής	συσχέτιση ονόματος
εμβέλεια μιας δήλωσης	χρόνος ζωής συσχέτισης

Οργάνωση Μνήμης

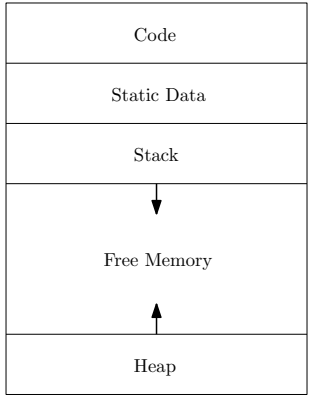
Ας υποθέσουμε πως ο μεταγλωττιστής έχει στην διάθεση του ένα κομμάτι μνήμης από το λειτουργικό σύστημα όπου θα εκτελεστεί το τελικό πρόγραμμα.

Το κομμάτι αυτό της μνήμης μπορεί να διαχωριστεί στα εξής επιμέρους κομμάτια:

1. τον τελικό κώδικα
2. διάφορα αντικείμενα
3. την στοίβα ελέγχου για την διαχείριση των ενεργοποιήσεων

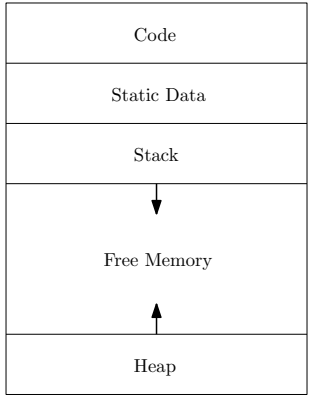
Οργάνωση Μνήμης

- το μέγεθος του κώδικα είναι σταθερό
- το μέγεθος μερικών αντικειμένων (π.χ static μεταβλητές στην C) είναι επίσης σταθερό



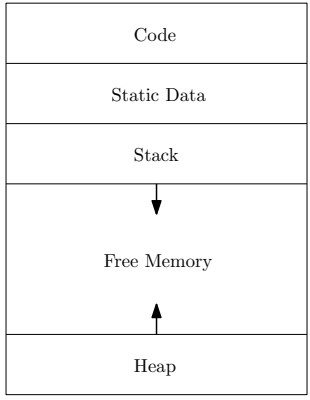
Οργάνωση Μνήμης

- υλοποιήσεις γλωσσών όπως η C χρησιμοποιούν παραλλαγές την στοίβας ελέγχου για να ελέγχουν την ενεργοποίηση των συναρτήσεων



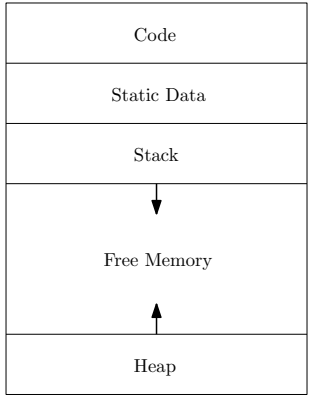
Οργάνωση Μνήμης

- με την κλήση μια συνάρτησης, η εκτέλεση μιας ενεργοποίησης σταματάει και πληροφορίες σχετικά με την κατάσταση της μηχανής όπως ο καταχωρητής γραμμής προγράμματος (program counter), οι τιμές των καταχωρητών, κ.τ.λ. αποθηκεύονται στην στοίβα
- όταν ο έλεγχος επιστρέψει από μια κλήση, η ενεργοποίηση μπορεί να επαναεκκινηθεί αφού οι διάφοροι καταχωρητές πάρουν τις παλιές τιμές τους, και ο καταχωρητής γραμμής προγράμματος δείχνει στην επόμενη εντολή μετά την κλήση
- αντικείμενα που η διάρκεια ζωής τους είναι ίδια με την διάρκεια ενεργοποίησης μπορούν να αποθηκευτούν και αυτά μέσα στην στοίβα, μαζί με οποιαδήποτε άλλη πληροφορία σχετίζεται με την ενεργοποίηση



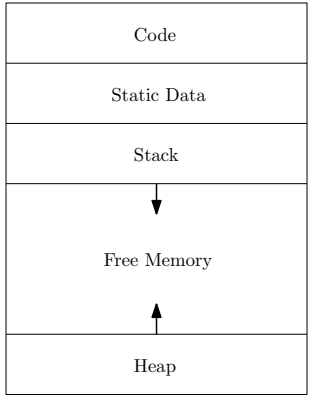
Οργάνωση Μνήμης

- ένα ξεχωριστό κομμάτι μνήμης που ονομάζεται σωρός (heap) χρησιμοποιείται για την αποθήκευση των υπόλοιπων στοιχείων
- για παράδειγμα οι γλώσσες που επιτρέπουν δυναμική καταχώρηση μνήμης κατά την διάρκεια της εκτέλεσης χρησιμοποιούν τον σωρό για την αποθήκευση αυτών των αντικειμένων



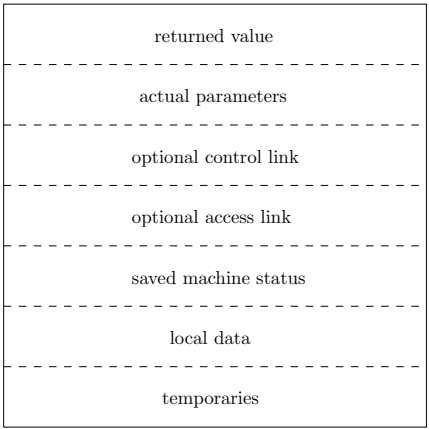
Οργάνωση Μνήμης

- η στοίβα και ο σωρός μεγαλώνουν προς την αντίθετη κατεύθυνση για να γίνεται καλύτερη διαχείριση χώρου



Εγγραφές Ενεργοποίησης

Όλη η πληροφορία που είναι απαραίτητη σε μια συγκεκριμένη κλήση μιας συνάρτησης αποθηκεύεται σε ένα συνεχόμενο κομμάτι μνήμης που ονομάζεται *εγγραφή ενεργοποίησης* (activation record ή frame).



Σε γλώσσες όπως η C είναι συνήθες να ωθείται μια εγγραφή ενεργοποίησης στην στοίβα όταν καλείται μια συνάρτηση και να απωθείται όταν επιστρέψει ο έλεγχος στην συνάρτηση που έκανε την κλήση.

Αυτή είναι μια γενική μορφή, μπορεί να διαφέρει από γλώσσα σε γλώσσα ανάλογα με τις ανάγκες

Προσωρινές Τιμές (Temporaries)

- π.χ τιμές που προκύπτουν από τον υπολογισμό κάποιων αριθμητικών εκφράσεων και οι οποίες για κάποιο λόγο δεν μπορούν να αποθηκευτούν σε καταχωρητές

Εγγραφές Ενεργοποίησης

Προσωρινές Τιμές (Temporaries)

- π.χ τιμές που προκύπτουν από τον υπολογισμό κάποιων αριθμητικών εκφράσεων και οι οποίες για κάποιο λόγο δεν μπορούν να αποθηκευτούν σε καταχωρητές

Τοπικά Δεδομένα (Local Data)

- που ανήκουν στην συνάρτηση στην οποία ανήκει η εγγραφή ενεργοποίησης

Εγγραφές Ενεργοποίησης

Προσωρινές Τιμές (Temporaries)

- π.χ τιμές που προκύπτουν από τον υπολογισμό κάποιων αριθμητικών εκφράσεων και οι οποίες για κάποιο λόγο δεν μπορούν να αποθηκευτούν σε καταχωρητές

Τοπικά Δεδομένα (Local Data)

- που ανήκουν στην συνάρτηση στην οποία ανήκει η εγγραφή ενεργοποίησης

Αποθηκευμένη Κατάσταση Μηχανής (Saved Machine Status)

- με πληροφορίες για την κατάσταση της μηχανής αμέσως πριν την κλήση της συνάρτησης. Συνήθως περιέχει την διεύθυνση επιστροφής, και τις τιμές των καταχωρητών πριν την κλήση.

Δείκτης Πρόσβασης (Access Link)

- ένας δείκτης που πολλές φορές χρειάζεται για την πρόσβαση σε δεδομένα που χρειάζεται η συνάρτηση, αλλά δεν ανήκουν σε αυτή, π.χ ανήκουν σε άλλη εγγραφή ενεργοποίησης
- παράδειγμα είναι η υποστήριξη nested functions, δηλαδή συναρτήσεων που έχουν δηλωθεί μέσα σε άλλες συναρτήσεις

Εγγραφές Ενεργοποίησης

Δείκτης Πρόσβασης (Access Link)

- ένας δείκτης που πολλές φορές χρειάζεται για την πρόσβαση σε δεδομένα που χρειάζεται η συνάρτηση, αλλά δεν ανήκουν σε αυτή, π.χ ανήκουν σε άλλη εγγραφή ενεργοποίησης
- παράδειγμα είναι η υποστήριξη nested functions, δηλαδή συναρτήσεων που έχουν δηλωθεί μέσα σε άλλες συναρτήσεις

Δείκτης Ελέγχου (Control Link)

- δείκτης που δείχνει στην εγγραφή ενεργοποίησης της συνάρτησης που κάλεσε την συνάρτηση που εκτελείται

Τιμές Επιστροφής (Returned Values)

- χώρος για την τιμή επιστροφής της συνάρτησης. Πολλές φορές χρησιμοποιείται κάποιος ειδικός καταχωρητής για αυτή την δουλειά.

Εγγραφές Ενεργοποίησης

Τιμές Επιστροφής (Returned Values)

- χώρος για την τιμή επιστροφής της συνάρτησης. Πολλές φορές χρησιμοποιείται κάποιος ειδικός καταχωρητής για αυτή την δουλειά.

Παράμετροι (Actual Parameters)

- χώρος για τις τιμές των παραμέτρων της συνάρτησης. Πολλές φορές οι παράμετροι είναι αποθηκευμένες σε καταχωρητές για μεγαλύτερη ταχύτητα.

Ακολουθίες Κλήσεων και Επιστροφής

Calling & Return Sequences

Η κλήση συναρτήσεων υλοποιείται από τις **ακολουθίες κλήσεων**, που είναι κώδικας ο οποίος δεσμεύει μια εγγραφή ενεργοποίησης στην στοίβα και εισάγει πληροφορίες μέσα στα πεδία της.

Μια **ακολουθία επιστροφής** είναι παρόμοιος κώδικας ο οποίος επιστρέφει την μηχανή στην προηγούμενη κατάσταση της ώστε να μπορεί να συνεχίσει η εκτέλεση μετά την επιστροφή μιας συνάρτησης.

Ακολουθίες Κλήσεων και Επιστροφής

Calling & Return Sequences

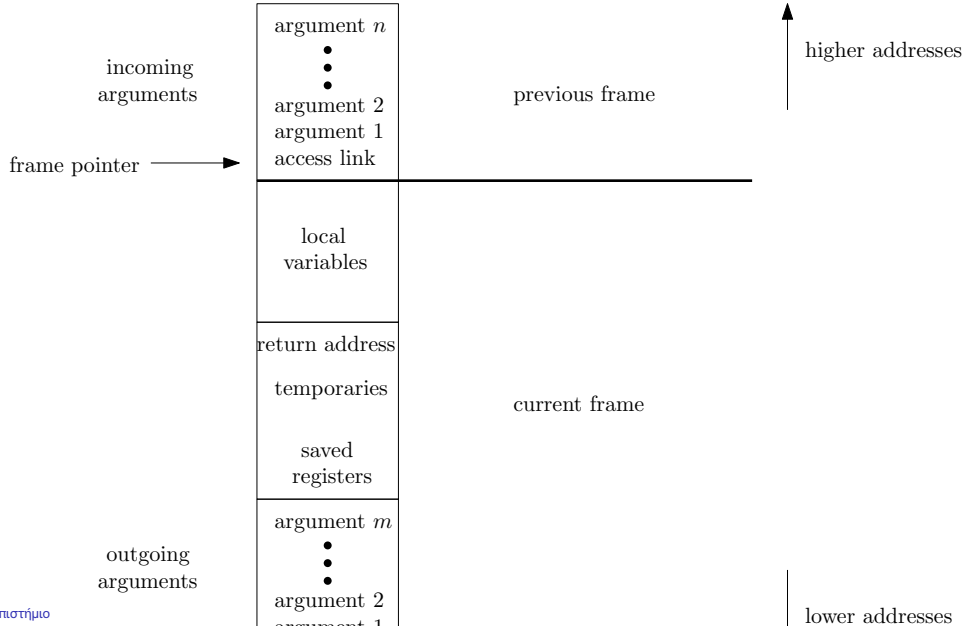
Οι ακολουθίες κλήσεων και επιστροφής και η ακριβής δομή της εγγραφής ενεργοποίησης, μπορεί να διαφέρει σημαντικά από υλοποίηση σε υλοποίηση ακόμη και για την ίδια γλώσσα.

Σε γενικές γραμμές ο κώδικας μιας ακολουθίας κλήσεων χωρίζεται σε δύο μέρη μεταξύ

1. της συνάρτησης που πραγματοποιεί την κλήση (**caller**)
2. και της συνάρτησης που καλείται (**callee**).

Δεν υπάρχει πάντα σαφής διαχωρισμός μεταξύ αρμοδιοτήτων, γενικά προσπαθούμε να δώσουμε όσες περισσότερες αρμοδιότητες στην συνάρτηση που καλείται (callee).

Παράδειγμα Εγγραφής Ενεργοποίησης

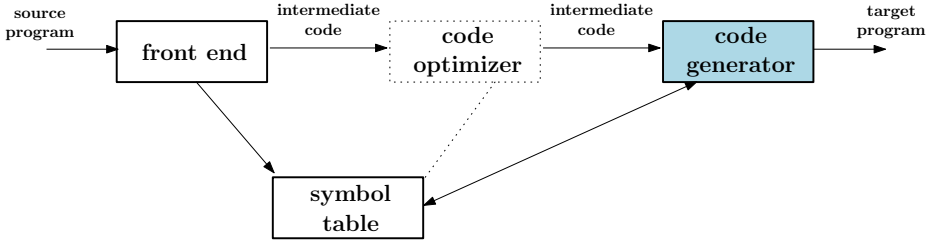


Ακολουθίες Κλήσεων και Επιστροφής

Για λεπτομέρειες σχετικά με την ακολουθία κλήσεων της γλώσσας C δείτε το `http://www.cs.bell-labs.com/who/dmr/clcs.html` .

Γενικά η χρήση μιας ”κλασσικής” εγγραφής ενεργοποίησης επιτρέπει σε μια γλώσσα την κλήση συναρτήσεων γραμμένες σε άλλη γλώσσα προγραμματισμού.

Παραγωγή Τελικού Κώδικα



(i) Η τελευταία φάση μεταγλώττισης πέρνει ως είσοδο κώδικα σε ενδιάμεση μορφή.

(ii) Πολλές φορές πέρνει είσοδο από μια φάση βελτιστοποίησης του ενδιάμεσου κώδικα.

Χαρακτηριστικά Γεννήτριας Τελικού Κώδικα

Τα χαρακτηριστικά που ζητάμε από μια γεννήτρια τελικού κώδικα είναι:

(i) η έξοδος πρέπει να είναι σωστή

(ii) η έξοδος πρέπει να είναι υψηλής ποιότητας, δηλαδή να χρησιμοποιεί σωστά τους διαθέσιμους πόρους της μηχανής

(iii) η γεννήτρια τελικού κώδικα πρέπει επίσης να είναι αποδοτική

(iv) τέλος θέλουμε η γεννήτρια να είναι εύκολα υλοποιήσιμη και εύκολα διατηρήσιμη.

Βέλτιστος Κώδικας

Μαθηματικά το προβλημα παραγωγής βέλτιστου τελικού κώδικα είναι **μη-αποφασίσιμο** (**undecidable**).

Επίσης πολλά υποπροβλήματα του όπως η ανάθεση καταχωρητών (register allocation) είναι NP-Complete.

Βέλτιστος Κώδικας

Στην πράξη είμαστε ευχαριστημένοι με ευριστικές μεθόδους (heuristics) που παράγουν καλό κώδικα, αλλά όχι απαραίτητα βέλτιστο κώδικα.

Η επιλογή των ευριστικών μεθόδων είναι σημαντική αφού μια προσεκτικά σχεδιασμένη γεννήτρια κώδικα μπορεί να παράξει κώδικα που είναι αρκετές φορές ταχύτερος από κώδικα που παράγεται από μια πρόχειρα σχεδιασμένη γεννήτρια.

Θέματα Σχεδίασης Γεννήτριας Τελικού Κώδικα

(i) Είσοδος Γεννήτριας
(ii) Έξοδος Γεννήτριας
(iii) Επιλογή Εντολών
(iv) Ανάθεση Καταχωρητών
(v) Σειρά Εκτέλεσης

Είσοδος στην Γεννήτρια Τελικού Κώδικα

Είναι μια ενδιάμεση μορφή του προγράμματος που έχει παραχθεί από τις αρχικές φάσης της μεταγλώττισης (front-end) μαζί με πληροφορίες που έχουν συλλεχθεί στον πίνακα συμβόλων.

Η ενδιάμεση μορφή μπορεί να είναι οποιαδήποτε από τις:

1. αναπαράσταση 3-διευθύνσεων όπως τετράδες, τριάδες, έμμεσες τριάδες
2. αναπαράσταση εικονικής μηχανής όπως bytecodes και κώδικας μηχανής στοίβας
3. γραφική αναπαράσταση όπως συντακτικά δέντρα ή κατευθυνόμενα ακυκλικά γραφήματα

Έξοδος της Γεννήτριας Τελικού Κώδικα

Το σύνολο εντολών μηχανής που είναι διαθέσιμο, επηρεάζει σημαντικά το έργο κατασκευής μιας αποδοτικής γεννήτριας τελικού κώδικα, η οποία να παράγει αποδοτικά προγράμματα.

Τα πιο συνήθη σύνολα εντολών μηχανής είναι

- RISC (reduced instruction set computer)
- CISC (complex instruction set computer)
- μηχανή στοίβας.

RISC και CISC

Μια μηχανή RISC έχει συνήθως πολλούς καταχωρητές, εντολές 3-διευθύνσεων, απλές μορφές διευθυνσιοδότησης, και σχετικά απλές εντολές.

Μια μηχανή CISC έχει συνήθως λίγους καταχωρητές, εντολές 2-διευθύνσεων, πολλές μεθόδους διευθυνσιοδότησης, πολλές κατηγορίες καταχωρητών, εντολές μεταβαλλόμενου μήκους, και εντολές με παρενέργειες.

Μηχανές Στοίβας

Μια μηχανή στοίβας πραγματοποιεί λειτουργίες διαβάζοντας τους τελεστέους από την κορυφή μιας στοίβας.

Για να πραγματοποιήσουμε μια λειτουργία, όπως πρόσθεση πρέπει να ωθήσουμε (push) τους τελεστέους στην στοίβα και να εκτελέσουμε την λειτουργία της πρόσθεσης.

Το αποτέλεσμα της εντολής αποθηκεύεται επίσης στην κορυφή της στοίβας.

Μηχανές Στοίβας

Για να πετύχουν οι μηχανές στοίβας υψηλή απόδοση αποθηκεύουν συνήθως την κορυφή της στοίβας σε καταχωρητές.

Η μηχανές στοίβας σχεδόν εξαφανίστηκαν καθώς θεωρήθηκε πως είναι περιοριστικές και χρειάζονται πολλές διαδικασίες swap και copy.

(i) Η αρχιτεκτονική αυτή όμως ”αναστήθηκε” με την υλοποίηση του **Java Virtual Machine (JVM)**.

(ii) Στην συνέχεια αναπτύχθηκε και το **CLR (Common Language Runtime)** που είναι μια μηχανή στοίβας για τις τεχνολογίες **.NET** .

Java Virtual Machine

Το JVM είναι ένα πρόγραμμα διερμηνέας (interpreter) για Java **bytecodes**, μια ενδιάμεση γλώσσα που παράγεται από μεταγλωττιστές Java.

Μέσω του JVM παρέχεται η συμβατότητα του κώδικα μεταξύ διαφόρων μηχανών, αφού αρκεί να υλοποιήσουμε ένα JVM για μια καινούρια μηχανή ώστε να υποστηρίζεται η γλώσσα Java. Αυτός είναι και ένας από τους λόγους για την επιτυχία της Java.

Java Virtual Machine

Για να ξεπεραστεί το υψηλό τίμημα στην απόδοση που εισάγει το JVM (περίπου $\times 10$), κατασκευάστηκαν **just-in-time (JIT)** Java μεταγλωττιστές.

Οι μεταγλωττιστές αυτοί μετατρέπουν το bytecode σε γλώσσα μηχανής την ώρα της εκτέλεσης του προγράμματος.

Μια άλλη τεχνική για καλύτερη απόδοση των Java προγραμμάτων είναι η απευθείας μεταγλώττιση σε κώδικα μηχανής, χωρίς την παραγωγή bytecodes.

Κώδικας Μηχανής

Απόλυτος (absolute)

Η παραγωγή απόλυτου (absolute) κώδικα μηχανής επιτρέπει την εύκολη εκτέλεση του αφού αρκεί να αντιγραφεί σε ένα σταθερό σημείο της μνήμης και μετά μπορεί να εκτελεστεί.

(i) Η παραγωγή όμως μετατοπίσιμου (relocatable) κώδικα μηχανής μας επιτρέπει να μεταγλωττίζουμε υποπρογράμματα ξεχωριστά.

(ii) Ένα σύνολο μετατοπίσιμων υποπρογραμμάτων μπορούν να συνδεθούν μεταξύ τους (linked) και να φορτωθούν για εκτέλεση από ένα επιπλέον πρόγραμμα που ονομάζεται πρόγραμμα σύνδεσης/φόρτωσης.

(iii) Παρόλο που πρέπει να πληρώσουμε το έξτρα κόστος της σύνδεσης και φόρτωσης, κερδίζουμε μεγάλη ευελίξία μπορώντας να μεταγλωττίζουμε προγράμματα ξεχωριστά και να καλούμε άλλα προγράμματα από ένα υποπρόγραμμα.

Η παραγωγή κώδικα σε μορφή assembly γλώσσας κάνει ευκολότερη την παραγωγή ενδιάμεσου κώδικα.

Μπορούμε να χρησιμοποιήσουμε συμβολικές εντολές, και να χρησιμοποιήσουμε τις μακροεντολές του assembler για την παραγωγή κώδικα.

Το τίμημα είναι το βήμα της εκτέλεσης του assembler μετά την παραγωγή του κώδικα.

Επιλογή Εντολών

Η γεννήτρια τελικού κώδικα πρέπει να μετασχηματίσει το πρόγραμμα ενδιάμεσης αναπαράστασης (IR) σε μια σειρά εντολών που να μπορεί να εκτελεστεί από την μηχανή.

Η πολυπλοκότητα υπολογισμού αυτής της αντιστοίχισης εξαρτάται από μια σειρά παραγόντων όπως

- το επίπεδο της ενδιάμεσης μορφής
- την φύση του συνόλου εντολών της μηχανής
- το επίπεδο ποιότητας του τελικού κώδικα μηχανής.

Επιλογή Εντολών

Η απλή στρατηγική του να έχουμε ένα συγκεκριμένο κομμάτι κώδικα για κάθε κομμάτι της ενδιάμεσης μορφής δεν οδηγεί σε πολύ αποτελεσματικό κώδικα.

π.χ κάθε εντολή της μορφής $x = y + z$ όπου τα x, y και z είναι στατικά αποθηκευμένα θα μπορούσε να μετατραπεί σε

```
LD  R0, y            (load y in register R0)
ADD R0, R0, z        (add z to R0)
ST  x, R0            (store R0 to x)
```

Η στρατηγική όμως αυτή παράγει πολλά περιττά load και store.

Επιλογή Εντολών

π.χ οι δύο παρακάτω εντολές

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

μετατρέπεται σε

```
LD  R0, b           // R0 = b
ADD R0, R0, c       // R0 += c
ST  a, R0           // a = R0
LD  R0, a           // R0 = a
ADD R0, R0, e       // R0 += e
ST  d, R0           // d = R0
```

Η 4η εντολή είναι άχρηστη αφού έχουμε ήδη το *a* φορτωμένο στον καταχωρητή R0.

Σε περίπτωση που δεν χρησιμοποιείται ξανά το *a* τότε και 3η εντολή είναι περιττή.

Ποιότητα Τελικού Κώδικα

Η ποιότητα του κώδικα που παράγει μια γεννήτρια εξαρτάται συνήθως από

- την ταχύτητα του
- το μέγεθος του.

(i) Στις περισσότερες μηχανές, ένα πρόγραμμα σε ενδιάμεση μορφή μπορεί να υλοποιηθεί με πολλές διαφορετικές ακολουθίες εντολών, με σημαντική διαφορά σε κόστος.

(ii) Μια απλή μετάφραση μπορεί λοιπόν να οδηγήσει σε σωστό αλλά καθόλου αποδεκτό από πλευράς αποδοτικότητας κώδικα.

Ποιότητα Τελικού Κώδικα

Πολλές φορές μια μηχανή έχει ειδικές εντολές για διάφορες περιπτώσεις.

π.χ μια μηχανή μπορεί να έχει μια εντολή ”αύξησης κατά 1” οπότε ο κώδικας $a = a + 1$ θα μπορούσε να υλοποιηθεί ως

```
INC a
```

αντί για

```
LD  R0, a           // R0 = a
ADD R0, R0, #1      // R0 = R0 + 1
ST  a, R0           // a = R0
```

Σε γενικές γραμμές θα μοντελοποιήσουμε το πρόβλημα παραγωγής κώδικα ως πρόβλημα βελτιστοποίησης όπου οι εντολές θα έχουν κάποιο καθορισμένο κόστος.

Επιλογή Εντολών

Το πρόβλημα της επιλογής εντολών μοντελοποιείται συνήθως ως πρόβλημα επικάλυψης ενός δέντρου (tree tiling).

Λύνεται ή με άπληστους αλγορίθμους ή με δυναμικό προγραμματισμό.

Ανάθεση Καταχωρητών

Ένα από τα κυριότερα προβλήματα στην παραγωγή κώδικα είναι να αποφασίσουμε ποιες τιμές να κρατήσουμε σε καταχωρητές.

(i) οι καταχωρητές είναι η πιο γρήγορη μνήμη ενός υπολογιστή
(ii) δεν υπάρχουν όμως αρκετοί για την αποθήκευση όλων των τιμών
(iii) οι εντολές που αφορούν καταχωρητές είναι μικρότερες και γρηγορότερες

Η σωστή χρήση καταχωρητών είναι πολύ σημαντική για την καλή απόδοση του τελικού κώδικα.

Ανάθεση Καταχωρητών

Το πρόβλημα της βέλτιστης ανάθεσης μεταβλητών σε καταχωρητές είναι δύσκολο.

(i) Μαθηματικά το πρόβλημα είναι NP-Complete.

(ii) Μοντελοποιείται ως πρόβλημα χρωματισμού γραφημάτων (graph coloring).