Προηγμένα Θέματα Θεωρητικής Πληροφορικής

Επιλογή κώδικα

Νικόλαος Καββαδίας
nkavv@uop.gr

24 Μαρτίου 2010

Σημαντικά ζητήματα στη γέννηση κώδικα (1)

- Επιθυμητές ιδιότητες του γεννήτορα κώδικα (code generator)
  - Το παραγόμενο πρόγραμμα χαμηλού επιπέδου οφείλει να διατηρεί σημασιολογική ισοδυναμία με το πηγαίο πρόγραμμα
  - Τα παραγόμενο πρόγραμμα θα πρέπει να εκμεταλλεύεται τους πόρους της στοχευόμενης αρχιτεκτονικής με τον καλύτερο δυνατό τρόπο
  - Ο γεννήτορας κώδικα να μην έχει υπερβολικές απαιτήσεις σε χρόνο μεταγλώττισης ή εκτέλεσης
- Κύρια βήματα για τη γέννηση κώδικα
  - Επιλογή κώδικα (code selection)
  - Καταμερισμός καταχωρητών (register allocation)
  - Χρονοπρογραμματισμός εντολών (instruction scheduling)

Σημαντικά ζητήματα στη γέννηση κώδικα (1)

- Προκλήσεις στη γέννηση κώδικα
  - Πολλές διαδικασίες στα πλαίσια της γέννησης κώδικα έχουν εκθετική υπολογιστική πολυπλοκότητα και δεν μπορεί να συντεθεί αλγόριθμος πολυωνυμικής πολυπλοκότητας για αυτές (π.χ. καταμερισμός καταχωρητών)
  - Ανταλλαγή (trade-off) ανάμεσα στην υπολογιστική πολυπλοκότητα των χρησιμοποιούμενων ευριστικών (heuristics) και στην ποιότητα των αποτελεσμάτων
  - Η σχετική σειρά των επιμέρους διαδικασιών για τη γέννηση κώδικα (phase integration problem)

Στοιχεία της στοχευόμενης αρχιτεκτονικής για τη γέννηση κώδικα

- Επεξεργαστές περιορισμένου συνόλου εντολών (RISC: Reduced Instruction Set Computer)
  - πολλοί καταχωρητές
  - εντολές τριών διευθύνσεων
  - απλοί και λίγοι τρόποι διευθυνσιοδότησης
  - απλή αρχιτεκτονική συνόλου εντολών με ορθογώνια κωδικοποίηση
- Επεξεργαστές σύνθετου συνόλου εντολών (CISC: Complex Instruction Set Computer)
  - λίγοι καταχωρητές
  - εντολές με διαφορετικό αριθμό ορισμάτων
  - πολλοί τρόποι διευθυνσιοδότησης
  - περισσότερες από μία κατηγορίες καταχωρητών
  - εντολές με διαφορετικά μήκη κωδικοποίησης
- Άλλες αρχιτεκτονικές: μηχανές στοίβας
- ☞ Οι περισσότερες μοντέρνες αρχιτεκτονικές δεν είναι αμιγώς RISC ή CISC

Επιλογή κώδικα (1)

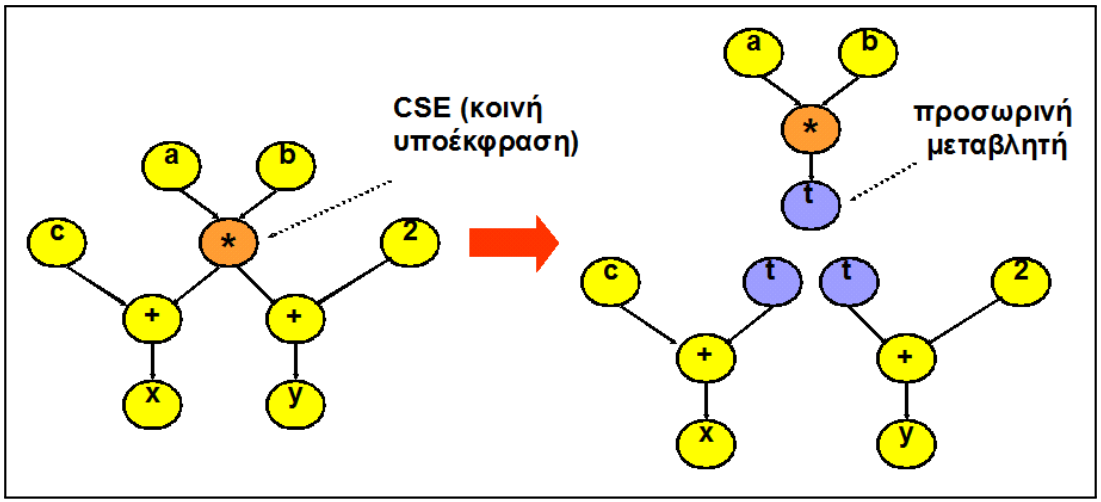
- Μετασχηματισμός από την ενδιάμεση αναπαράσταση (IR) του προγράμματος σε αναπαράσταση με εντολές συμβολομεταφραστή του στοχευόμενου επεξεργαστή
  - **Είσοδος**: γράφος ροής δεδομένων (DFG: Data Flow Graph) για κάθε βασικό μπλοκ
  - **Έξοδος**: αποδοτική αντιστοίχιση ομαδοποιήσεων από εκφράσεις επιπέδου IR σε πραγματικές εντολές του επεξεργαστή
- Η επιλογή κώδικα απευθείας σε γράφους εξάρτησης δεδομένων, οι οποίοι αποτελούν DFG είναι έμφυτα πολύπλοκη (NP-complete)
- Συνήθως κάθε DFG διαχωρίζεται σε ένα 'δάσος' (forest) από δένδρα ροής δεδομένων (DFT: Data Flow Tree) στα όρια των κοινών υποεκφράσεων (CSEs)

Επιλογή κώδικα (2)

- Παράδειγμα:
  - Έστω DFG το οποίο αντιστοιχεί στον εξής κώδικα:

```
x = a*b + c;
y = a*b + 2;
```

  - Στην περίπτωση αυτή, η κοινή υποέκφραση είναι το αποτέλεσμα του γινομένου `a` × `b`
  - Γίνεται ανάθεση της CSE σε προσωρινή μεταβλητή `t = a*b`



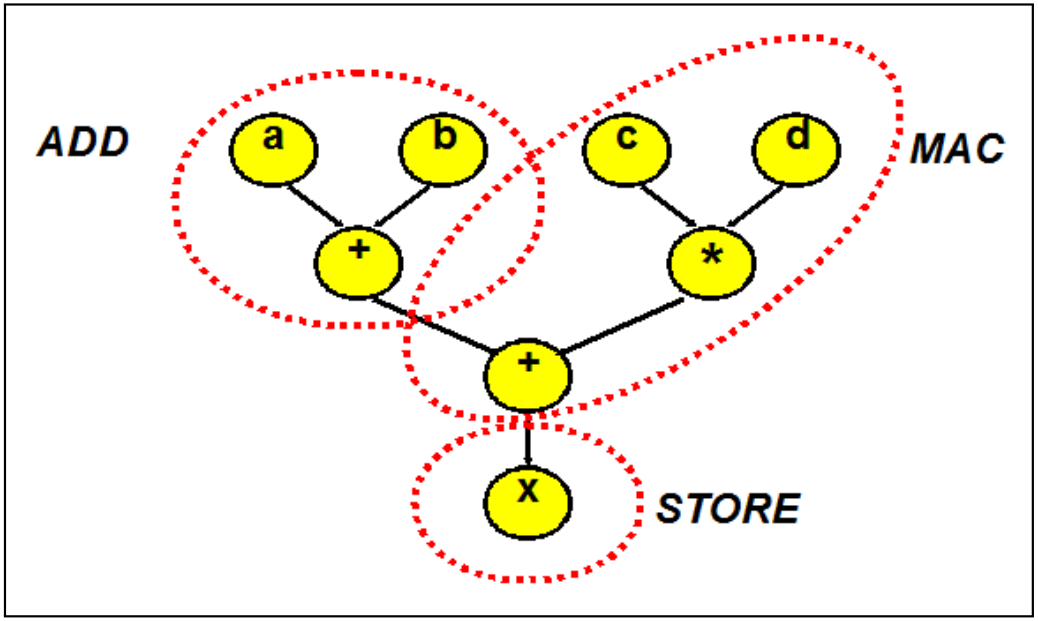
Κάλυψη δένδρου (tree covering)

- Η επιλογή κώδικα από DFTs μπορεί να θεωρηθεί ως πρόβλημα *κάλυψης* τους από υποδένδρα τα οποία έχουν ένα-προς-ένα αντιστοιχία με πραγματικές εντολές του επεξεργαστή
- Η διαδικασία ελέγχεται από κάποιο μετρικό κόστους (cost metric) όπως για παράδειγμα το μέγεθος του παραγόμενου κώδικα, ο χρόνος εκτέλεσης στον επεξεργαστή, ή η καταναλισκόμενη ισχύς στον επεξεργαστή κατά την εκτέλεση του προγράμματος
- Υπάρχουν αλγοριθμικές τεχνικές και εργαλεία που τις εφαρμόζουν για τη γέννηση κώδικα με βέλτιστη κάλυψη δένδρων

Παράδειγμα κάλυψης δένδρου

- Παραγόμενος κώδικας για το απεικονιζόμενο DFT

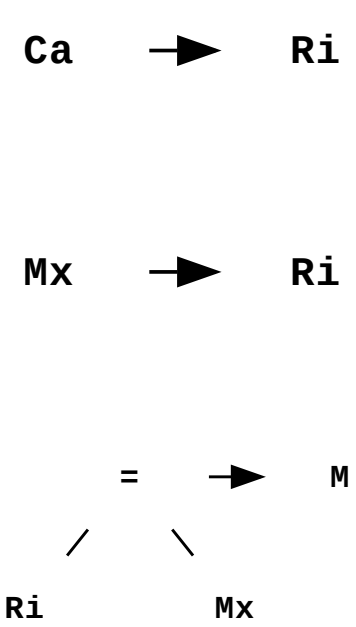
```
LOAD  R1, a
LOAD  R2, b
ADD   R1, R1, R2
LOAD  R2, c
LOAD  R3, d
MUL   R2, R2, R3
ADD   R2, R1, R2
STORE x, R2
```



Αρχή λειτουργίας της επιλογής κώδικα με κάλυψη δένδρου

1. Δοθέντος ενός δένδρου ροής δεδομένων
2. Γίνεται αρίθμηση όλων των κόμβων του δένδρου
3. Πραγματοποιείται ταύτιση μοτίβων (pattern matching) πάνω στο δένδρο προκειμένου να επιλεγεί η βέλτιστη περίπτωση επανεγγραφής (μετασχηματισμού) του δένδρου λόγω αντικατάστασης ενός υποδένδρου
   - Η διάβαση του δένδρου μπορεί να γίνει είτε bottom-up είτε top-down
   - Λαμβάνεται υπόψη το μετρικό κόστους που αντιστοιχείται σε κάθε επανεγγραφή
   - Χρησιμοποιείται κάποια τεχνική για την επιλογή του υποδένδρου που πρόκειται να καλυφθεί και να αντιστοιχηθεί με εντολή assembly
     - Μη βέλτιστες τεχνικές: μέγιστη 'μπουκιά' (maximal munch)
     - Βέλτιστες τεχνικές: TWIG, BURS (Bottom Up Rewrite System)

Κανόνες επανεγγραφής δένδρου για μία υποθετική μηχανή (1)

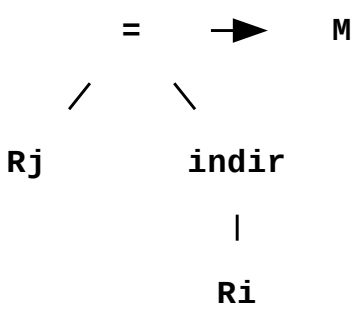



```
LOAD Ri, #a
```

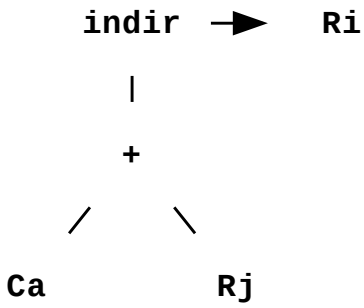
```
LOAD Ri, x
```

```
STORE x, Ri
```

Κανόνες επανεγγραφής δένδρου για μία υποθετική μηχανή (2)

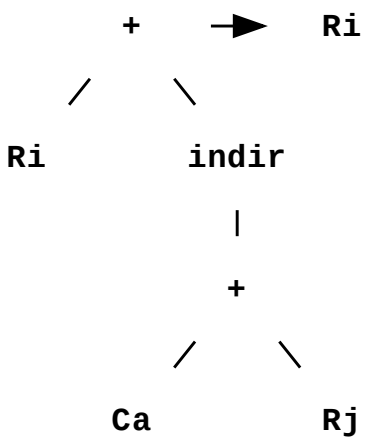



```
STORE Ri, Rj
```

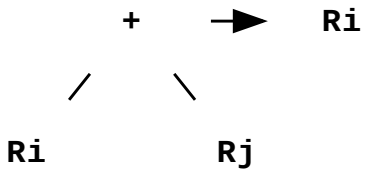



```
LOAD Ri, a(Rj)
```

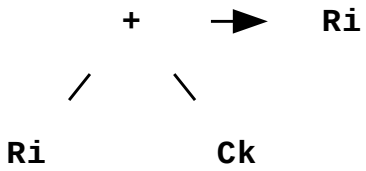
Κανόνες επανεγγραφής δένδρου για μία υποθετική μηχανή (3)




```
ADD Ri, Ri, a(Rj)
```



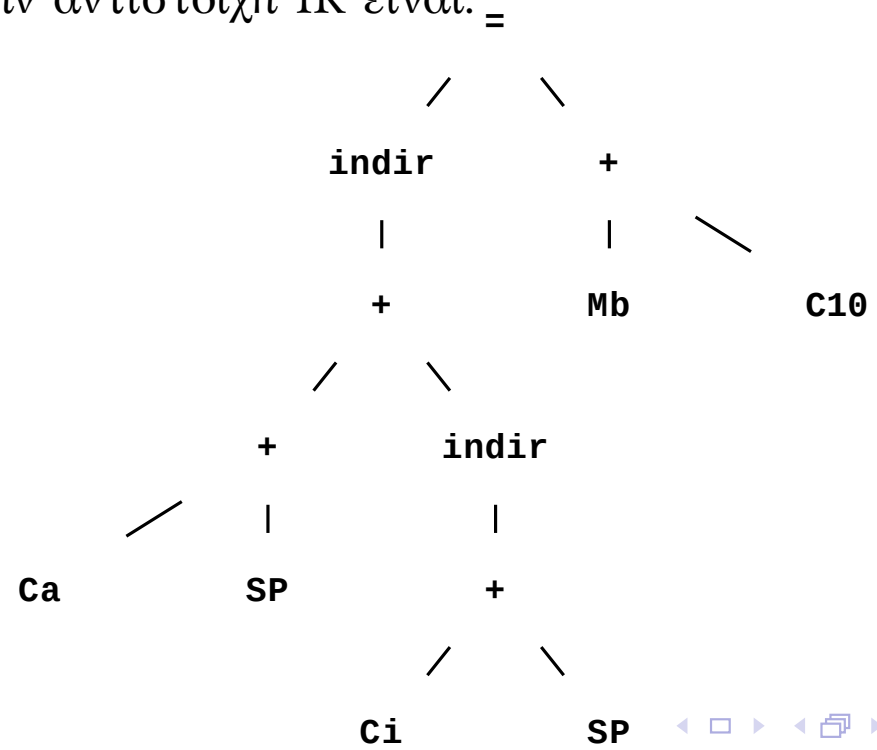

```
ADD Ri, Ri, Rj
```




```
ADD Ri, Ri, #k
```

Σχηματισμός δένδρου για μια σύνθετη έκφραση

- Έστω η έκφραση `a[i] = b + 10;` όπου
  - ο πίνακας `a` είναι αποθηκευμένος στη στοίβα, η οποία δεικτοδοτείται με αναφορά τα περιεχόμενα του δείκτη στοίβας (*SP*)
  - η μεταβλητή `b` είναι καθολικής εμβέλειας (στη μνήμη)
  - η διεύθυνση του πρώτου στοιχείου του πίνακα `a` και η τιμή του δείκτη `i` βρίσκονται σε θέσεις μνήμης σχετικές ως προς την τιμή του *SP*
- Το δένδρο για την αντίστοιχη IR είναι:



Γέννηση κώδικα με πλακόστρωση (tiling) του αρχικού δένδρου

- Δοθέντος ενός δένδρου, εφαρμόζονται οι παραπάνω κανόνες επανεγγραφής (rewriting rules) για την πλακόστρωση των αντίστοιχων υποδένδρων
- Όταν ένα μοτίβο ταιριάζει με ένα υποδένδρο, τότε η ισοδύναμή του έκφραση αντικαθιστά το αντίστοιχο υποδένδρο, και εκτελούνται οι τυχόν ενέργειες (όπως γέννηση κώδικα assembly με ανάθεση καταχωρητών) που συνδέονται με τον κανόνα
- Η διαδικασία συνεχίζεται μέχρις ότου το αρχικό δένδρο να μειωθεί σε ένα μονήρη κόμβο ο οποίος να αντιστοιχεί στην ισοδύναμη έκφραση κάποιου κανόνα ή μέχρις ότου να μην μπορεί να βρεθεί κάποια νέα αντικατάσταση
- Στην πράξη, υπάρχουν εργαλεία (γεννήτορες γεννητόρων κώδικα) που εξασφαλίζουν τη βέλτιστη κάλυψη ενός δένδρου (BURG, IBURG, OLIVE)

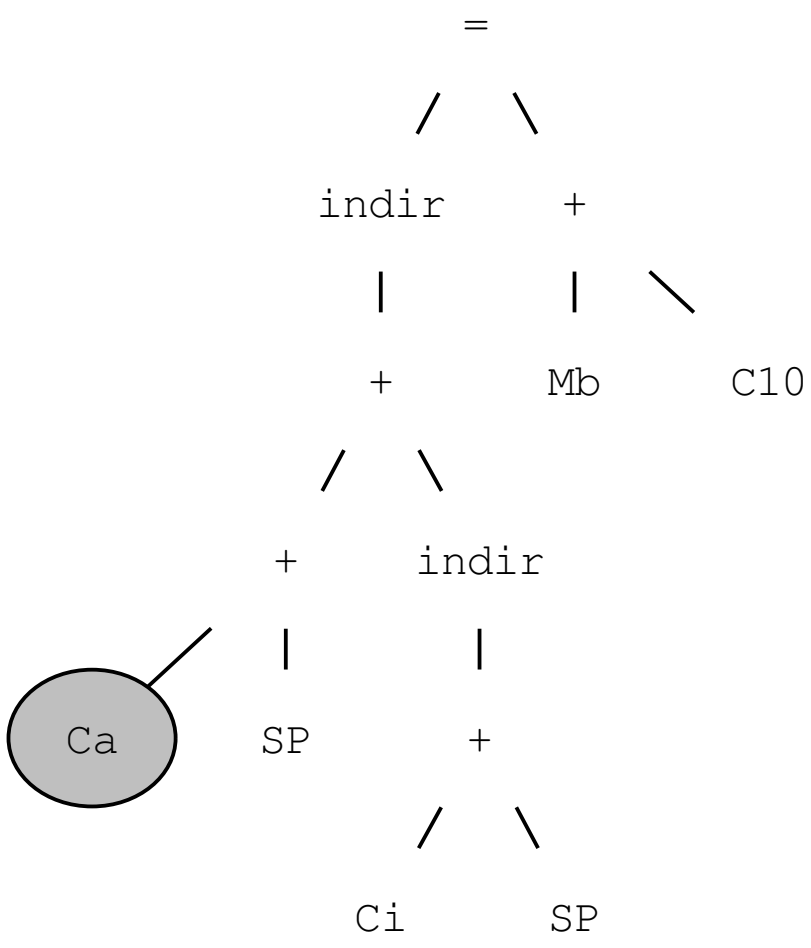
Γέννηση κώδικα με πλακόστρωση του αρχικού δένδρου: Βήμα 1

- Παραγόμενος κώδικας

```
LOAD  R0, #a
```

- Εφαρμογή κανόνα 1

Ca → **Ri**



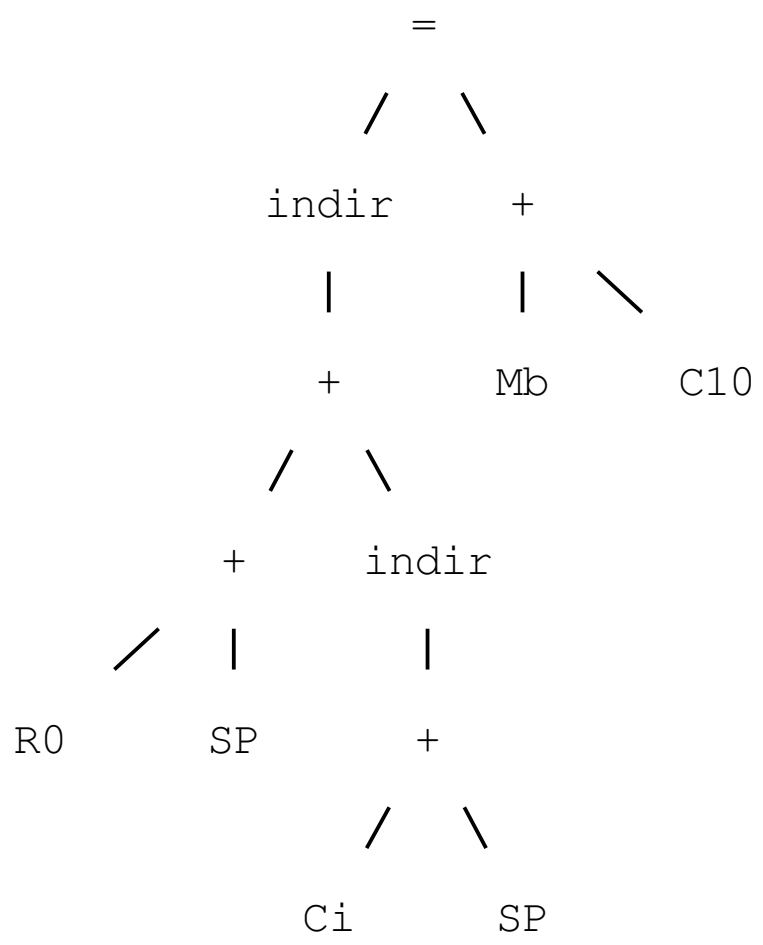
Γέννηση κώδικα με πλακόστρωση του αρχικού δένδρου: Βήμα 1

- Παραγόμενος κώδικας

```
LOAD  R0, #a
```

- Εφαρμογή κανόνα 1

Ca → **Ri**

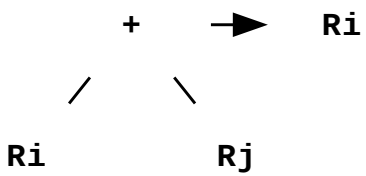


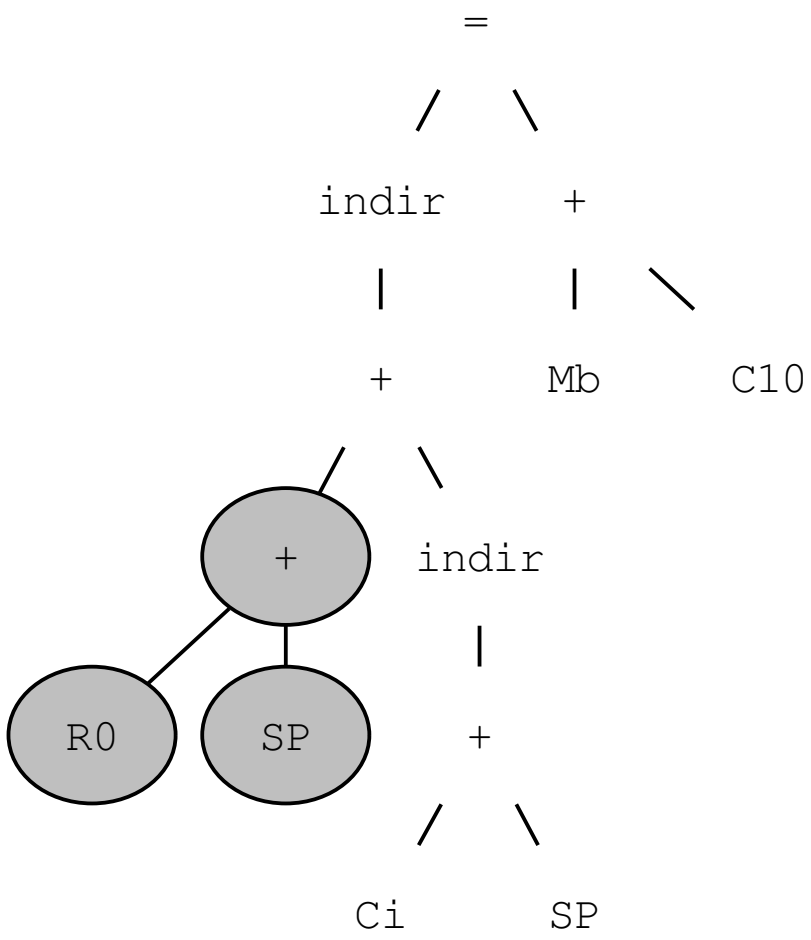
Γέννηση κώδικα με πλακόστρωση του αρχικού δένδρου: Βήμα 2

- Παραγόμενος κώδικας

```
LOAD   R0, #a
ADD    R0, R0, SP
```

- Εφαρμογή κανόνα 7



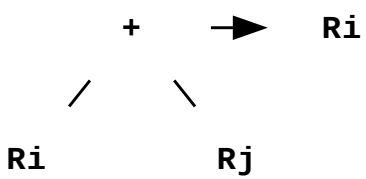


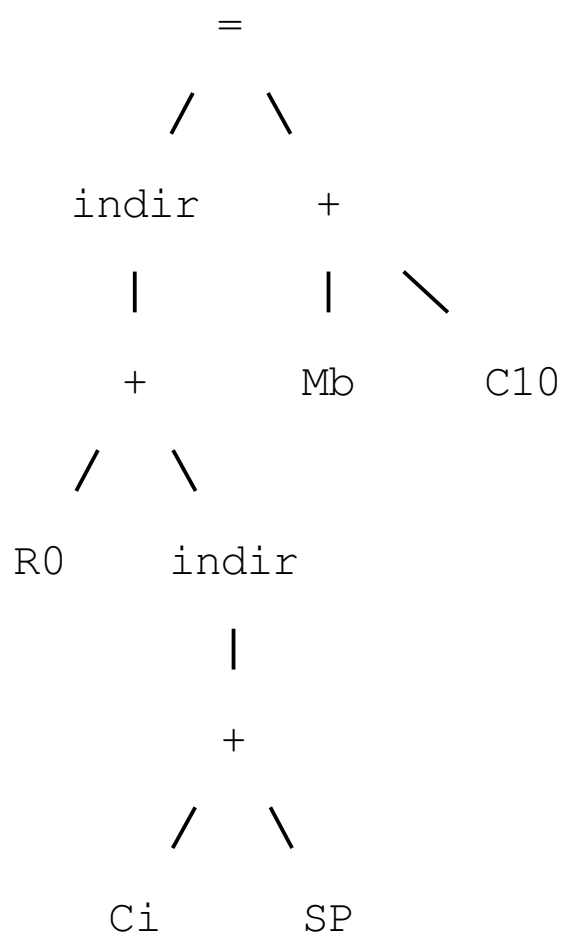
Γέννηση κώδικα με πλακόστρωση του αρχικού δένδρου: Βήμα 2

- Παραγόμενος κώδικας

```
LOAD   R0, #a
ADD    R0, R0, SP
```

- Εφαρμογή κανόνα 7



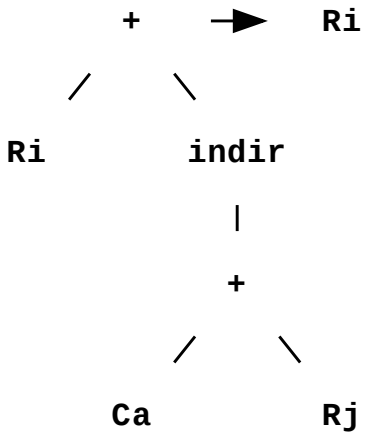


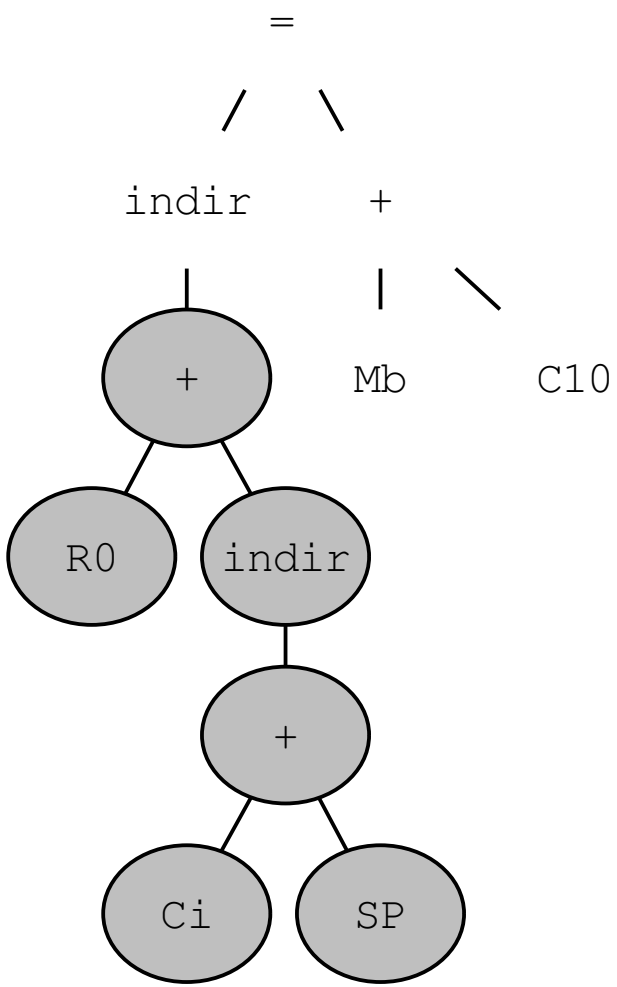
Γέννηση κώδικα με πλακόστρωση του αρχικού δένδρου: Βήμα 3

- Παραγόμενος κώδικας

```
LOAD   R0, #a
ADD    R0, R0, SP
ADD    R0, R0, i(SP)
```

- Εφαρμογή κανόνα 6



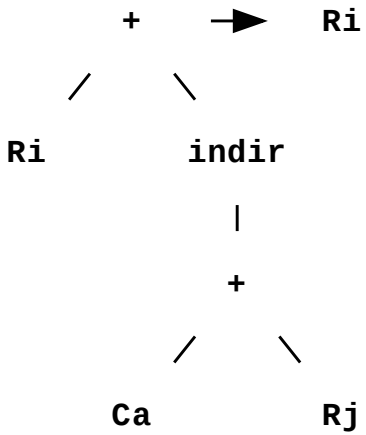


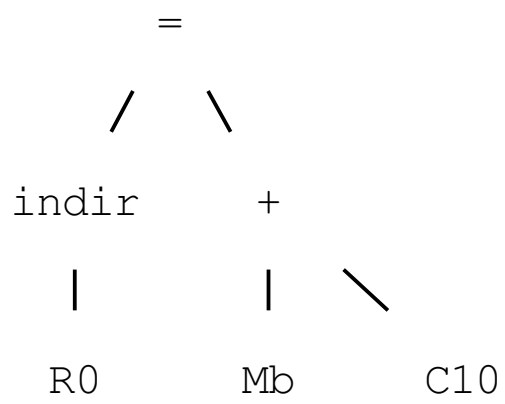
Γέννηση κώδικα με πλακόστρωση του αρχικού δένδρου: Βήμα 3

- Παραγόμενος κώδικας

```
LOAD   R0, #a
ADD    R0, R0, SP
ADD    R0, R0, i(SP)
```

- Εφαρμογή κανόνα 6

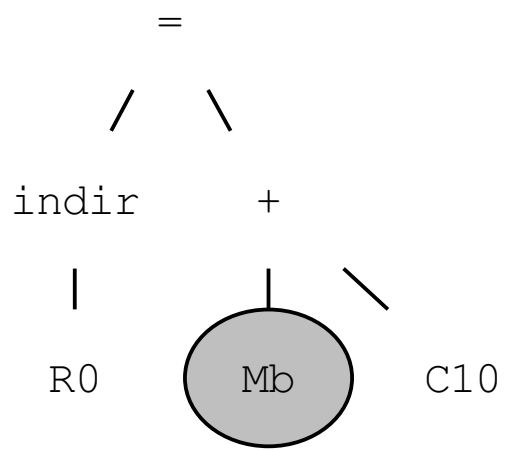




Γέννηση κώδικα με πλακόστρωση του αρχικού δένδρου: Βήμα 4

- Παραγόμενος κώδικας

```
LOAD   R0, #a
ADD    R0, R0, SP
ADD    R0, R0, i(SP)
LOAD   R1, b
```

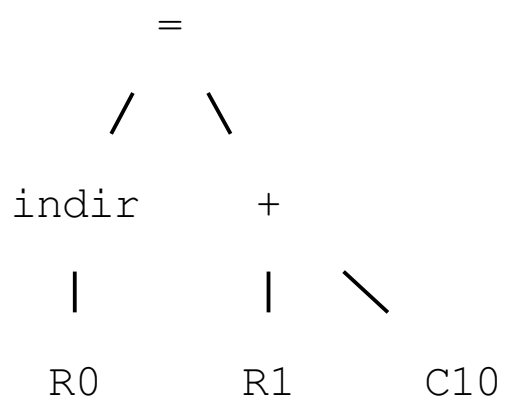


- Εφαρμογή κανόνα 2

Γέννηση κώδικα με πλακόστρωση του αρχικού δένδρου: Βήμα 4

- Παραγόμενος κώδικας

```
LOAD   R0, #a
ADD    R0, R0, SP
ADD    R0, R0, i(SP)
LOAD   R1, b
```

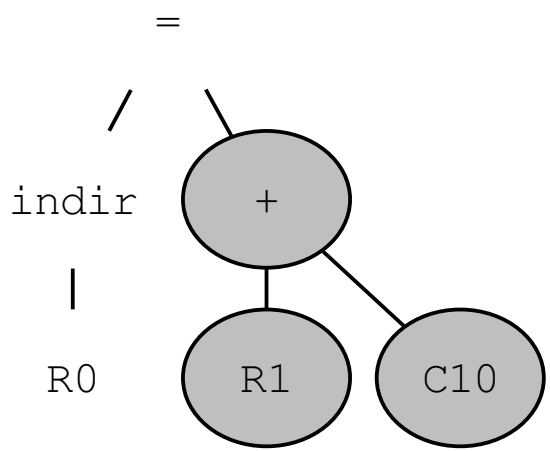


- Εφαρμογή κανόνα 2

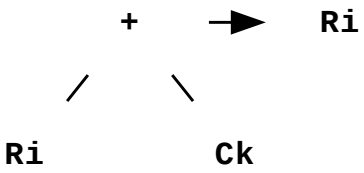
Γέννηση κώδικα με πλακόστρωση του αρχικού δένδρου: Βήμα 5

- Παραγόμενος κώδικας

```
LOAD   R0, #a
ADD    R0, R0, SP
ADD    R0, R0, i(SP)
LOAD   R1, b
ADD    R1, R1, #10
```



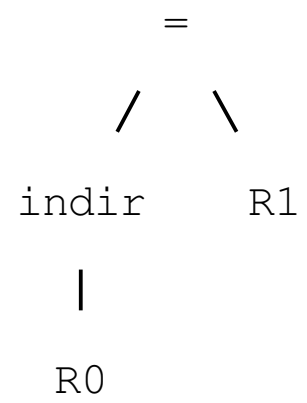
- Εφαρμογή κανόνα 8



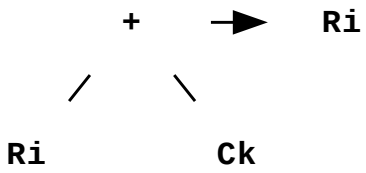
Γέννηση κώδικα με πλακόστρωση του αρχικού δένδρου: Βήμα 5

- Παραγόμενος κώδικας

```
LOAD   R0, #a
ADD    R0, R0, SP
ADD    R0, R0, i(SP)
LOAD   R1, b
ADD    R1, R1, #10
```



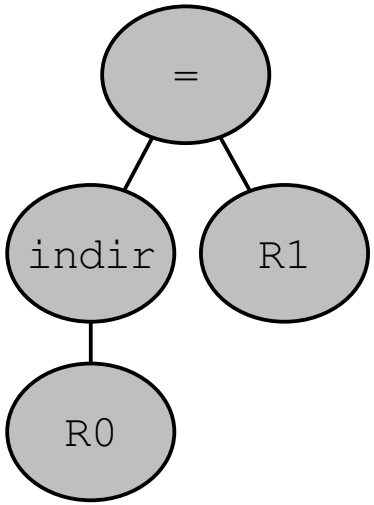
- Εφαρμογή κανόνα 8



Γέννηση κώδικα με πλακόστρωση του αρχικού δένδρου: Βήμα 6

- Παραγόμενος κώδικας

```
LOAD  R0, #a
ADD   R0, R0, SP
ADD   R0, R0, i(SP)
LOAD  R1, b
ADD   R1, R1, #10
STORE R0, R1
```



- Εφαρμογή κανόνα 4

```
      =    ->    M
    /   \
  Rj     indir
           |
          Ri
```

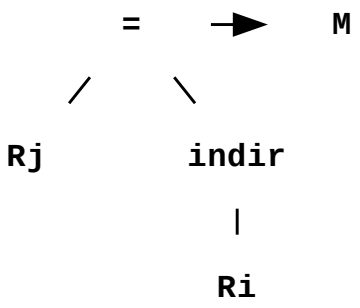
Γέννηση κώδικα με πλακόστρωση του αρχικού δένδρου: Βήμα 6

- Παραγόμενος κώδικας

```
LOAD  R0, #a
ADD   R0, R0, SP
ADD   R0, R0, i(SP)
LOAD  R1, b
ADD   R1, R1, #10
STORE R0, R1
```

DONE!

- Εφαρμογή κανόνα 4



Κάλυψη δένδρου με τον αλγόριθμο της *μέγιστης μπουκιάς* (maximal munch)

- Ο αλγόριθμος maximal munch για την κάλυψη δένδρου αποτελεί μία άπληστη (greedy) μέθοδο για την επίλυση του προβλήματος
- Ροή του αλγορίθμου
  1. Ξεκινώντας από τον κόμβο ρίζα βρίσκουμε το μεγαλύτερο υποδένδρο
  2. Κάλυψη του κόμβου ρίζας και πιθανώς και άλλων κόμβων με το πρώτο υποδένδρο
  3. Επανάληψη για κάθε υποδένδρο
- Καλά αποτελέσματα για σύνολα εντολών που είναι αμιγώς RISC
- Η πλακόστρωση που επιτυγχάνεται δεν είναι εγγυημένα βέλτιστη

Επιλογή κώδικα με τεχνικές δυναμικού προγραμματισμού

- Κοινό στοιχείο των τεχνικών που χρησιμοποιούν δυναμικό προγραμματισμό είναι η ανάθεση κόστους σε κάθε κόμβο του δένδρου
- Η διάβαση του δένδρου γίνεται από κάτω προς τα πάνω (bottom-up traversal)
  1. Για κάθε πλακόστρωση t με κόστος c η οποία παρουσιάζει ταύτιση στον κόμβο n
  2. Το c_i ισούται με το κόστος κάθε υποδένδρου το οποίο αντιστοιχεί στους κόμβους-φύλλα (leaves) του t
  3. Το κόστος του n ισούται με $c + \sum c_i$
- Με διάβαση από πάνω προς τα κάτω (top-down) επιλέγεται η πλακόστρωση με το μικρότερο κόστος

Γέννηση επιλογέα κώδικα με το εργαλείο OLIVE (1)

- Το εργαλείο OLIVE διαβάζει τις προδιαγραφές του συνόλου εντολών με τη μορφή γραμματικής δένδρου (tree grammar)
- Παράγει πηγαίο κώδικα ANSI C ενός εξειδικευμένου, για την αρχιτεκτονική, επιλογέα κώδικα
- Ο επιλογέας κώδικα είναι βέλτιστος όσον αφορά την κάλυψη DFTs
- Ο επιλογέας κώδικα μπορεί να ενσωματωθεί στο συνολικό μεταγλωττιστή
- Το εργαλείο OLIVE χρησιμοποιεί την τεχνική TWIG

Γέννηση επιλογέα κώδικα με το εργαλείο OLIVE (2)

Τμηματική προδιαγραφή για τον OLIVE

```
stm: cs_STORE(reg,reg)
{
  $cost[0].cost = 1 + $cost[2].cost + $cost[3].cost;
}={
  $action[2]();
  $action[3]();
  printf("\n\tSTORE reg,reg");
};

reg: cs_PLUS(reg,reg)
{
  $cost[0].cost = 1 + $cost[2].cost + $cost[3].cost;
}={
  $action[2]();
  $action[3]();
  printf("\n\tADD reg,reg,reg");
};

reg: cs_READARG
{
  $cost[0].cost = 1;
}={
  printf("\n\tMV reg,%s",CS_EXPSYM($1)->Name());
};
```

Παράδειγμα κάλυψης DFT με το εργαλείο OLIVE (1)

Πηγαίος κώδικας C

```
int a;

void main(int b, int c)
{
  int d, e, f, g;

  a = b + c - d * e / f >> g;
}
```

TAC IR εκφρασμένη σε υποσύνολο της C

```
int a;

void main(int b_3, int c_4)
{
  int t1, t2, t3, t4, t5, t6;
  int c_4, d_6, e_7, f_8, g_9;

  t1 = b_3 + c_4;
  t2 = d_6 * e_7;
  t3 = t2 / f_8;
  t4 = t1 - t3;
  t5 = t4 >> g_9;
  t6 = &a;
  *t6 = t5;
}
```

Παράδειγμα κάλυψης DFT με το εργαλείο OLIVE (2)

Κειμενική μορφή ενός DFT

```
(cs_STORE [stm 7 '*t6 = t5;']
  (cs_ADDR [exp 22 'a' int] addr of 'a')
  (cs_SHR [exp 20 't4 >> g_9' int]
    (cs_MINUS [exp 16 't1 - t3' int]
      (cs_PLUS [exp 4 'b_3 + c_4' int]
        (cs_READARG [exp 2 'b_3' int] arg no 1)
        (cs_READARG [exp 3 'c_4' int] arg no 2))
      (cs_DIV [exp 12 't2 / f_8' int]
        (cs_MULT [exp 8 'd_6 * e_7' int])
          (cs_READ [exp 6 'd_6' int])
          (cs_READ [exp 7 'e_7' int]))
        (cs_READ [exp 11 'f_8' int])))
     (cs_READ [exp 19 'g_9' int])))
```

Κώδικας assembly για έναν υποθετικό επεξεργαστή

```
MV    reg, @a
MV    reg, b_3
MV    reg, c_4
ADD   reg, reg, reg
MV    reg, d_6
MV    reg, e_7
MUL   reg, reg, reg
MV    reg, f_8
DIV   reg, reg, reg
SUB   reg, reg, reg
MV    reg, g_9
SHR   reg, reg, reg
STORE reg, reg
```

Παραδείγματα επιλογής κώδικα από IR

- Τυπικές εντολές: φόρτωση από μνήμη, αποθήκευση στη μνήμη, άλμα σε διεύθυνση με ή χωρίς συνθήκη, αριθμητικές και λογικές εντολές, εντολές σύγκρισης
- Τρόποι διευθυνσιοδότησης: με δεικτοδότηση (indexed addressing), έμμεση διευθυνσιοδότηση (indirect addressing), άμεση (direct)

Assembly για την έκφραση `x = y + z;`

```
LOAD  R0, y      // R0 = y
ADD   R0, R0, z  // R0 = R0 + z
STORE x, R0      // x  = R0
```

Για την έκφραση `b = a[i];`

```
LOAD  R1, i      // R1 = i
MUL   R1, R1, 4  // R1 = R1 * 4
LOAD  R2, a(R1)  // R2 = MEM(a + R1)
STORE b, R2      // x  = R2
```

Για την έκφραση `*p = y;`

```
LOAD  R1, p      // R1 = p
LOAD  R2, y      // R2 = y
STORE 0(R1), R2  // MEM(0 + R1) = R2
```

Αναφορές του μαθήματος Ι


A. V. Aho, R. Sethi, and J. D. Ullman, *Μεταγλωττιστές: Αρχές, Τεχνικές και Εργαλεία*, με την επιμέλεια των: Αγγελος Σπ. Βώρος και Νικόλαος Σπ. Βώρος και Κων/νος Γ. Μασσέλος, **κεφάλαια 8.9, 8.9.1, 8.9.2**, Εκδόσεις Νέων Τεχνολογιών, 2008. Website for the English version: `http://dragonbook.stanford.edu`


R. Leupers, O. Whalen, M. Hahenauer, T. Kogel, and P. Marwedel, "An executable intermediate representation for retargetable compilation and high-level code optimization," in *Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS 2003)*, Samos, Greece, July 21-23 2003, pp. 120–125.


A. V. Aho, M. Ganapathi, and S. W. K. Tjiang, "Code generation using tree matching and dynamic programming," *ACM Transactions on Programming Languages and Systems*, vol. 11, no. 4, pp. 491–516, October 1989.


C. W. Fraser, D. R. Hanson, and T. A. Proebstring, "Engineering a simple, efficient code-generator generator," *ACM Letters on Programming Languages and Systems*, vol. 1, no. 3, pp. 213–226, September 1992.

Αναφορές του μαθήματος II


SPAM Research Group, "SPAM Compiler User's Manual," including a description of OLIVE in Chapter 4, September 23, 1997.


IBURG, a tree parser generator. [Online]. Available:
http://www.cs.princeton.edu/software/iburg/