Μεταγλωττιστές II

Γέννηση ενδιάμεσης αναπαράστασης

Νικόλαος Καββαδίας
nkavv@uop.gr

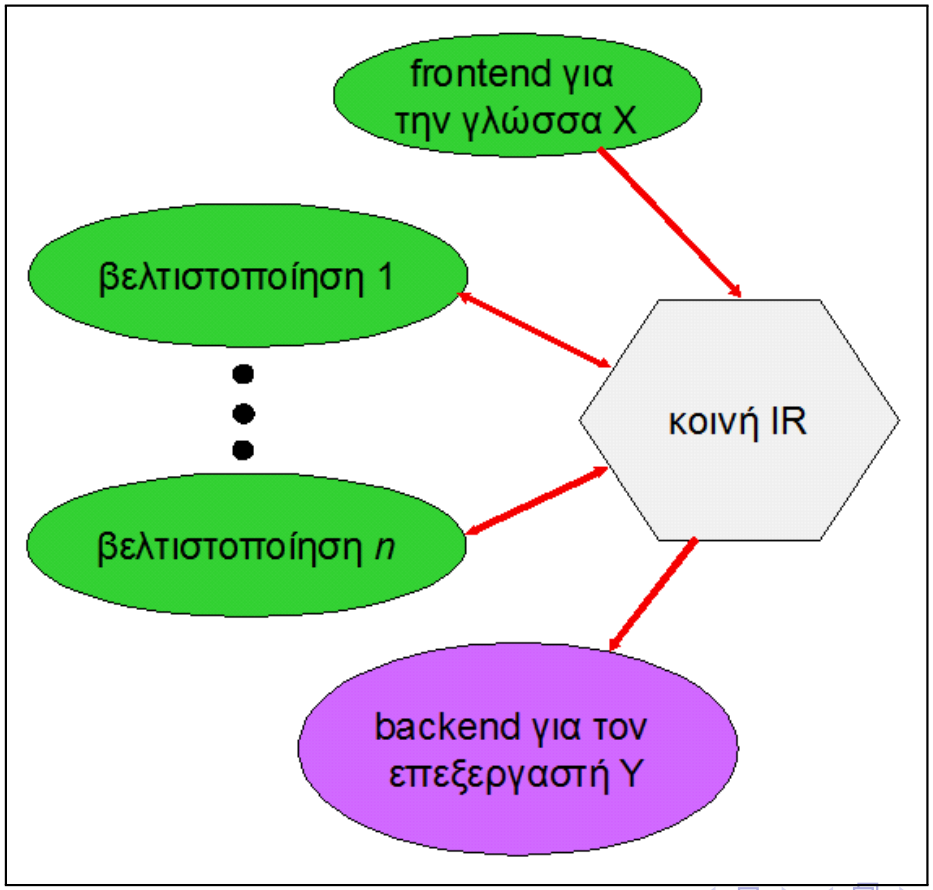
10 Νοεμβρίου 2010

Η έννοια της ενδιάμεσης αναπαράστασης

- Ενδιάμεση αναπαράσταση (IR: intermediate representation): απλοποιημένη, σημασιολογικά ισοδύναμη μορφή του πηγαίου προγράμματος
- Είναι η 'κοινή' (koine) γλώσσα η οποία χρησιμοποιείται για την επικοινωνία πληροφορίας για το πηγαίο πρόγραμμα από το frontend στο backend του μεταγλωττιστή
- Παράγεται από τα αφηρημένα συντακτικά δένδρα (AST: Abstract Syntax Tree) στα οποία αποδομείται το πηγαίο πρόγραμμα από το frontend
- Στο επίπεδο της IR υλοποιούνται μετασχηματισμοί για την εφαρμογή βελτιστοποιήσεων ανεξάρτητων από την αρχιτεκτονική του στοχευόμενου επεξεργαστή

Άποψη του μεταγλωττιστή από την πλευρά της IR

- Η εξαγωγή της IR είναι το αποτέλεσμα της λεκτικής, συντακτικής και σημασιολογικής ανάλυσης του πηγαίου προγράμματος



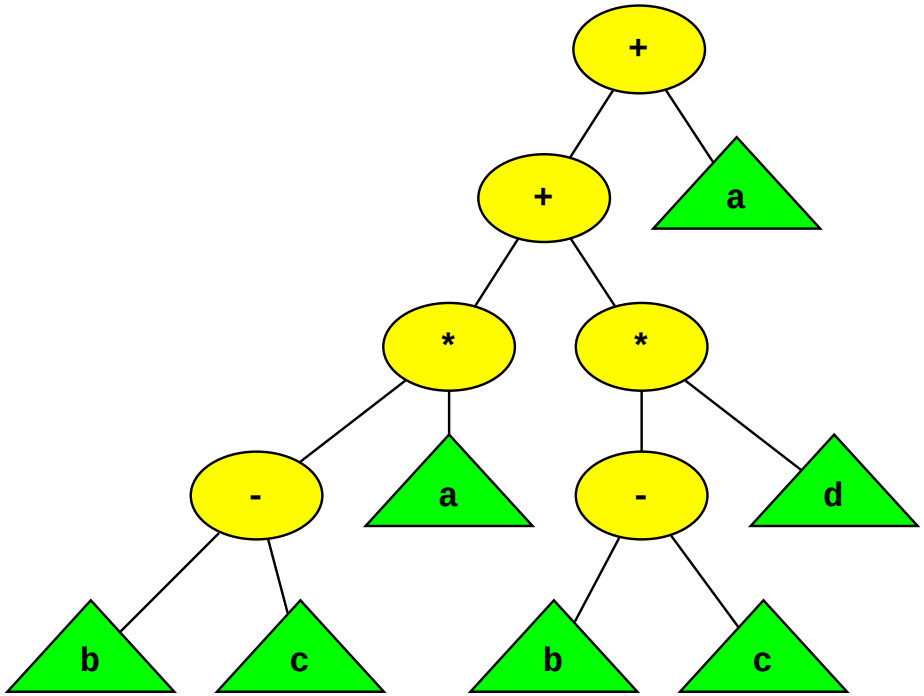
Υποβιβασμός (αποδόμηση) σύνθετων εκφράσεων

- Προγραμματιστικές δομές οι οποίες αποδομούνται σε απλούστερες
  - εντολές ελέγχου και επανάληψης
  - προσπελάσεις σε στοιχεία πίνακα
  - προσπελάσεις σε στοιχεία σύνθετων δομών δεδομένων (όπως είναι η `struct` στην ANSI C και η `record` στην Pascal)
  - σύνθετες αριθμητικές εκφράσεις
  - Παράδειγμα: ανάγνωση του στοιχείου του πίνακα `a` στη διεύθυνση `i*WIDTH+(j+2)` και ανάθεση στην μεταβλητή `b`

```
b = a[i][j+2]; =>    t1 = j+2;
                     t2 = WIDTH * i;
                     t3 = t1 + t2;
                     t4 = 4 * t3;
                     t5 = address(a);
                     t6 = t4 + t5;
                     b  = *t6;
```

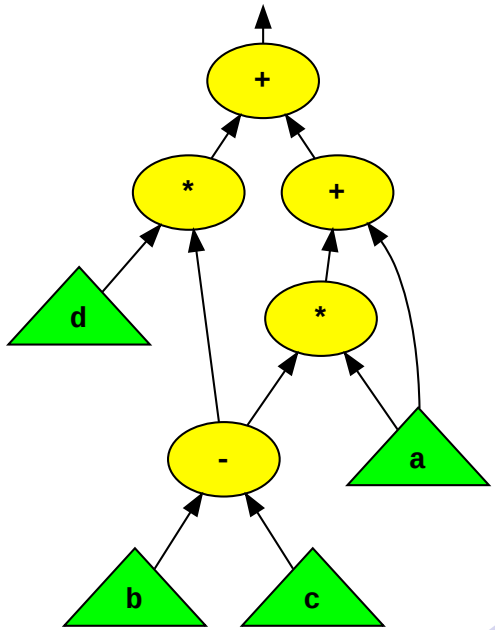
Αφηρημένο συντακτικό δένδρο (AST: Abstract Syntax Tree)

- Φυσική αναπαράσταση της εξόδου του συντακτικού αναλυτή για το πηγαίο πρόγραμμα
- Οι κόμβοι αναπαριστούν λεκτικές μονάδες του πηγαίου προγράμματος
- Παράδειγμα: το AST για την έκφραση $a + a \times (b - c) + (b - c) \times d$



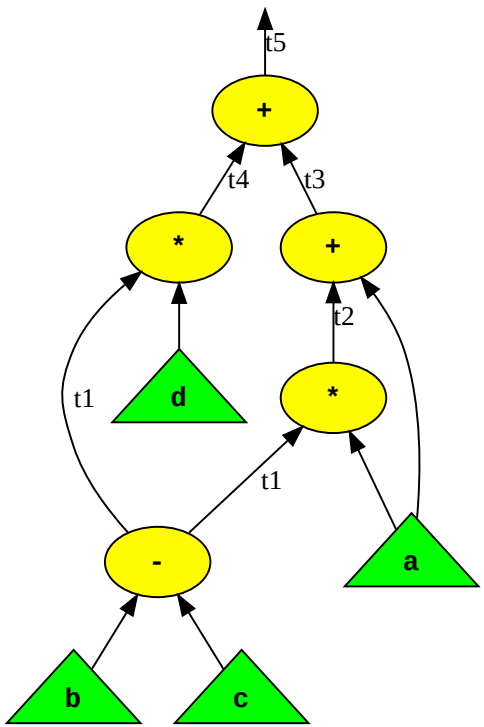
Κατευθυνόμενοι ακυκλικοί γράφοι (DAGs) για την αναπαράσταση εκφράσεων

- Η αναπαράσταση DAG (Directed Acyclic Graph) μιας έκφρασης αναγνωρίζει τις κοινές υποεκφράσεις (CSE: Common SubExpression), δηλ. εκείνες τις εκφράσεις που εμφανίζονται περισσότερες από μία φορά στο AST
- Ένας κόμβος N που αναπαριστά μια CSE έχει περισσότερους από έναν κόμβους-παιδιά
- Σε ένα AST η κοινή υποέκφραση εμφανίζεται σε τόσα αντίτυπα όσες φορές εμφανίζεται στην αρχική έκφραση
- Παράδειγμα: το DAG για την έκφραση $a + a \times (b - c) + (b - c) \times d$



Κώδικας τριών διευθύνσεων (TAC: Three Address Code)

- Γραμμικοποιημένη αναπαράσταση ενός AST ή DAG
- Ακολουθία απλών εντολών, οι οποίες χρησιμοποιούν κατά μέγιστο τρία ορίσματα: δύο ορίσματα ανάγνωσης και ένα εγγραφής
- Ο κώδικας TAC θυμίζει τις εντολές συμβολομεταφραστή ενός απλού επεξεργαστή RISC με τη διαφορά ότι γίνεται αναφορά σε συμβολικές θέσεις αποθήκευσης
- Για το παράδειγμα:




```
t1 = b - c;
t2 = a * t1;
t3 = a + t2;
t4 = t1 * d;
t5 = t3 + t4;
```

Περιγραφή εντολών και διευθύνσεων σε κώδικα TAC (1)

- Ο κώδικας τριών διευθύνσεων αποτελείται από εντολές που δρουν πάνω σε δεδομένα τα οποία υποδεικνύονται από διευθύνσεις
- Οι διευθύνσεις αυτές σε μεταγενέστερο στάδιο της μεταγλώττισης αποδίδουν μεταβλητές που αποθηκεύονται σε φυσικούς καταχωρητές ή σε θέσεις μνήμης δεδομένων
- Οι διευθύνσεις σε εντολές TAC μπορεί να είναι ονόματα, σταθερές, ή προσωρινές μεταβλητές
  - Ονόματα: όπως ονόματα μεταβλητών του πηγαίου προγράμματος (μεταβλητές καθολικής ή τοπικής εμβέλειας)
  - Σταθερές: αριθμητικές ή συμβολικές
  - Προσωρινές μεταβλητές: τις παράγει ο μεταγλωττιστής κατά την αποδόμηση του πηγαίου προγράμματος

Περιγραφή εντολών και διευθύνσεων σε κώδικα TAC (2)

- Τύποι εντολών σε TAC
  - Συμβολικές ετικέττες (symbolic labels) της μορφής `LABELn:` για τον καθορισμό συγκεκριμένων σημείων στη ροή του προγράμματος
    - Στοχεύονται από εντολές μεταφοράς ροής ελέγχου
    - Παίρνουν οριστικές τιμές (δείκτη εντολής) από ένα δεύτερο πέρασμα (backpatching)
  - Εντολές ανάθεσης της μορφής: `x = y op z;` όπου *op* είναι τελεστής αριθμητικός, λογικός ή σύγκρισης
  - Εντολές ανάθεσης για μοναδιαίους τελεστές: `x = op y;` όπως είναι οι τελεστές αντιστροφής, λογικής άρνησης, και μετατροπής τύπου δεδομένων (casting)
  - Εντολές αντιγραφής της μορφής `x = y;`

Περιγραφή εντολών και διευθύνσεων σε κώδικα TAC (3)

- Τύποι εντολών σε TAC (συνέχεια)
  - Άλματα χωρίς συνθήκη: `goto LABEL;`
  - Άλματα υπό συνθήκη της μορφής `if (x) goto LABEL;` και `if (!x) goto LABEL;`
  - Κλήση υποπρογράμματος `y = p(x1, x2, ..., xn)` με n ορίσματα

```
param x1;
param x2;
...
param xn;
call p, n;
// or
y = call p, n;
```

  - Επιστροφή τιμής (χρήση σε υποπρογράμματα): `return y;`
  - Αναθέσεις σε στοιχεία πίνακα όπως `x = y[i];` και `x[i] = y;` που εκφράζουν λειτουργίες φόρτωσης από και αποθήκευσης σε μνήμη
  - Δεικτοδοτημένες αναθέσεις
    - `x = &y;` η ανάθεση της διεύθυνσης της μεταβλητής y στην x
    - `x = *y;` η τιμή που βρίσκεται στη διεύθυνση y ανατίθεται στη μεταβλητή x
    - `*x = y;` η τιμή η οποία δεικτοδοτείται από τον δείκτη x ισούται πλέον με y

Μεταγλώττιση πηγαίου κώδικα σε TAC

- Έστω το εξής τμήμα πηγαίου κώδικα

```
i = 0;
do
{
  i = i + 1;
} while (a[i] < v);
```

- Μετατροπή σε TAC

```
    i = 0;
L:
    i = i + 1;
    t0 = a[i];
    if (t0 < v) goto L;
```

- Παραδείγματα χρήσης TAC βρίσκουμε στους μεταγλωττιστές GCC (υπό τη μορφή GIMPLE) και LANCE [Leupers, 2003] ως εκτελέσιμο κώδικα C χαμηλού επιπέδου

Αναπαράσταση ενδιάμεσου κώδικα σε 'τετράδες' (quadruples ή quads)

- Η 'τετράδα' αποτελεί δομή δεδομένων κατάλληλη για χρήση από το μεταγλωττιστή προκειμένου τη διατήρηση της πληροφορίας κώδικα TAC
- Οι τετράδες έχουν ένα-προς-ένα αντιστοιχία με τις εντολές TAC και αποτελούνται από τα εξής επιμέρους στοιχεία: *op* για τον τελεστή, arg_1, arg_2 για τα δύο έντελα ανάγνωσης, και *result* για το έντελο εγγραφής στο οποίο γράφεται τυχόν αποτέλεσμα της εντολής. Ένα ή περισσότερα από τα πεδία $arg_1, arg_2, result$ μπορεί να είναι κενά
- Τα στοιχεία μιας τετράδας υλοποιούνται ως μέλη μιας `struct` της C
- Για παράδειγμα η εντολή `z = x + y;` αντιπροσωπεύεται από την τετράδα `('+', x, y, z)`
- Για το παράδειγμα:

```
t1 = b - c;
t2 = a * t1;
t3 = a + t2;
t4 = t1 * d;
t5 = t3 + t4;
```

	op	arg_1	arg_2	*result*
0	subtract	b	c	t1
1	multiply	a	t1	t2
2	add	a	t2	t3
3	multiply	t1	d	t4
4	add	t3	t4	t5

Το βασικό μπλοκ (BB: basic block) (1)

- Ένα βασικό μπλοκ αποτελεί μία ακολουθία διαδοχικών εντολών με ένα σημείο εισόδου και ένα σημείο εξόδου με τις ακόλουθες ιδιότητες:
  - Η ροή ελέγχου μπορεί να μεταφερθεί στο βασικό μπλοκ μόνο διαμέσου της 1ης εντολής του
  - Η ροή ελέγχου εξέρχεται από το βασικό μπλοκ από την τελευταία εντολή του η οποία είναι η μόνη που επιτρέπεται να μεταφέρει τη ροή εκτέλεσης του προγράμματος σε μη διαδοχική διεύθυνση
- Το βασικό μπλοκ αποτελεί τη θεμελιώδη μονάδα μεταγλώττισης στην οποία εφαρμόζονται βασικές βελτιστοποιήσεις και θεμελιώδεις φάσεις της μεταγλώττισης όπως η επιλογή εντολών (instruction selection) και ο καταμερισμός καταχωρητών (register allocation)

Το βασικό μπλοκ (BB: Basic Block) (2)

- Παράδειγμα

```
1  data = inp;
2  count = 0;
3
4  while (data != 0)
5  {
6    count += (data & 0x1);
7    data = data >> 0x1;
8  }
```

```
1  L1:
2    data = inp;
3    count = 0;
4
5  L2:
6    if (data != 0) goto L3;
7    else goto L_EXIT;
8
9  L3:
10   t0 = data & 0x1;
11   count = count + t0;
12   data = data >> 0x1;
13   goto L2;
14
15 L_EXIT:
16   ...
```

Το βασικό μπλοκ (BB: basic block) (3)

- Η έννοια του βασικού μπλοκ χρησιμοποιείται από το επίπεδο της IR μέχρι και τον τελικό κώδικα (assembly) για τη στοχευόμενη μηχανή
- Η ροή ελέγχου μεταφέρεται στο BB από προηγηθέντα (predecessor) BB και μεταφέρεται σε διάδοχα (successor) BB
- Σημεία αλλαγής ροής ελέγχου
  - Άμεσες ή έμμεσες διακλαδώσεις (branches) και άλματα (jumps) με ή χωρίς συνθήκη
  - Κλήσεις υποπρογραμμάτων
  - Εξαιρέσεις (exceptions)

Ορισμός του Γράφου Εξάρτησης Δεδομένων (DDG: Data-Dependence Graph) ενός BB

- Ένας Γράφος Εξάρτησης Δεδομένων (DDG) είναι ο κατευθυνόμενος ακυκλικός γράφος (DAG) που αντιπροσωπεύει τις εξαρτήσεις δεδομένων σε μια ακολουθία διαδοχικών εντολών

Definition (DDG βασικού μπλοκ)

Καλείται $DDG(V, E)$ ο κατευθυνόμενος ακυκλικός γράφος που αναπαριστά τη ροή και τις εξαρτήσεις δεδομένων σε δοθέν βασικό μπλοκ · οι κόμβοι V αναπαριστούν στοιχειώδεις λειτουργίες και οι ακμές E αναπαριστούν εξαρτήσεις δεδομένων. Ο γράφος DDG αντιστοιχείται σε έναν γράφο $DDG^+(V \bigcup V^+, E \bigcup E^+)$, ο οποίος περιλαμβάνει επιπρόσθετους κόμβους V^+ και επιπρόσθετες ακμές E^+. Οι επιπρόσθετοι κόμβοι V^+ αναπαριστούν μεταβλητές (έντελα/ορίσματα) εισόδου και εξόδου από το βασικό μπλοκ. Οι επιπρόσθετες ακμές E^+ συνδέουν κόμβους του V^+ με κόμβους του V, και κόμβους V με του V^+.

Γράφος Ροής Ελέγχου (CFG: Control Flow Graph)

- Ένας Γράφος Ροής Ελέγχου (CFG) αποτελεί μια αναπαράσταση όλων των διαδρομών τις οποίες μπορεί να διαβεί η ροή εκτέλεσης του προγράμματος. Οι κορυφές του CFG αποτελούν βασικά μπλοκ και οι ακμές του αντιπροσωπεύουν άλματα στη ροή ελέγχου
- Ένα CFG αντιπροσωπεύει ένα υποπρόγραμμα (π.χ. function της C) και έτσι ένα πηγαίο πρόγραμμα αναπαρίσταται από ένα σύνολο από CFG
- Ένα CFG μπορεί να αποτελεί κυκλικό γράφο
- Ειδική σημασία στη λειτουργία αλγορίθμων ανάλυσης ροής ελέγχου και δεδομένων σε CFG έχουν το μπλοκ εισόδου και το μπλοκ εξόδου του, τα οποία μπορεί και να είναι εικονικά (source, sink blocks)

Γράφος Ροής Ελέγχου-Δεδομένων (CDFG: Control-Data Flow Graph)

- Η IR είναι οργανώσιμη σε μορφή Γράφου Ροής Ελέγχου-Δεδομένων (CDFG) για κάθε συνάρτηση του πηγαίου προγράμματος
- Ο CDFG είναι ένας ιεραρχικός γράφος δύο επιπέδων
  - Ανώτερο επίπεδο: CFG που έχει ως κορυφές τα BB της συνάρτησης και ως ακμές τις εξαρτήσεις ελέγχου μεταξύ τους
  - Κατώτερο επίπεδο: BB που αντιπροσωπεύεται από τον DDG του
- Γενικά ο CDFG μπορεί να οριστεί ως η ΔΙΑΤΕΤΑΓΜΕΝΗ ΤΡΙΑΔΑ (3-tuple) $CDFG(V, E, D)$ όπου: V το σύνολο των λειτουργιών, σταθερών τιμών και μεταβλητών ή ορισμάτων εισόδου/εξόδου σε αυτό το τμήμα του προγράμματος, E οι σχέσεις ανάμεσα σε κορυφές του συνόλου V, και D οι έμφυτες χρονικές καθυστερήσεις (π.χ. σε ns ή σε κύκλους μηχανής) για την ολοκλήρωση των λειτουργιών V

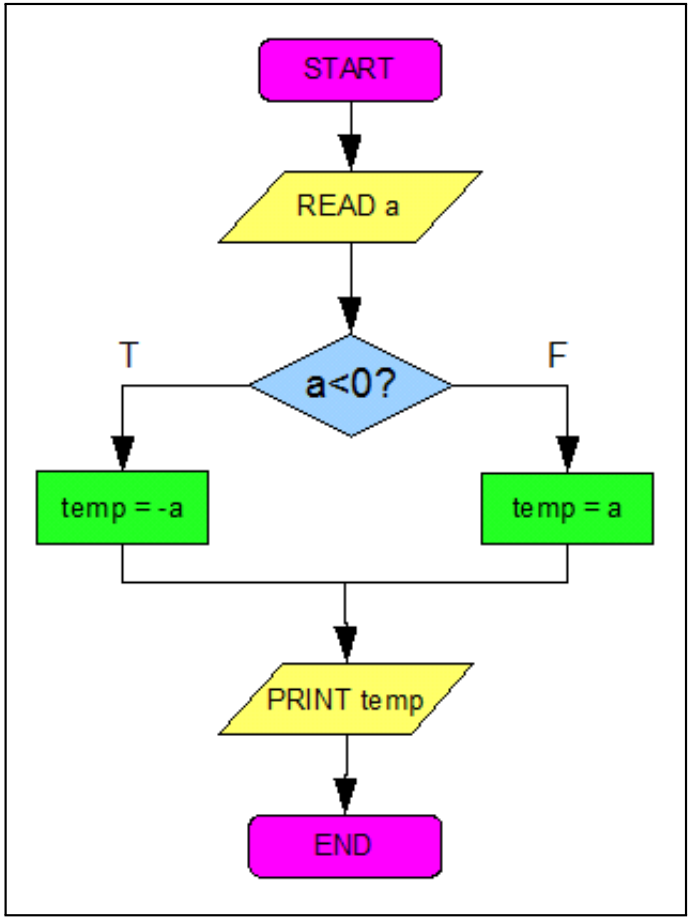
Ολοκληρωμένο παράδειγμα: Από την C στο CDFG (1)

- Συνάρτηση υπολογισμού απόλυτης τιμής σε C

```
int iabs(int a)
{
  int temp;

  if (a < 0)
    temp = -a;
  else
    temp = a;

  return temp;
}
```

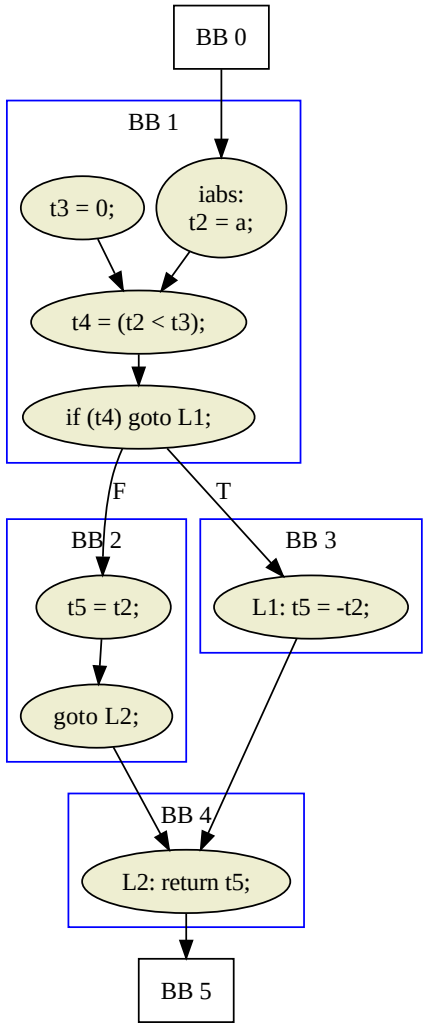


Ολοκληρωμένο παράδειγμα: Από την C στο CDFG (2)

- Κώδικας TAC για τη συνάρτηση `iabs`

```
iabs:
  t2 = a;
  t3 = 0;
  t4 = (t2 < t3);
  if (t4) goto L1;
  t5 = t2;
  goto L2;
L1:
  t5 = -t2;
L2:
  return t5;
```

- Το CDFG της `iabs`



Ενδιάμεση αναπαράσταση με χρήση Στατικής Απλής Ανάθεσης (SSA: Static Single Assignment)

- Η Στατική Απλή Ανάθεση (SSA) [Cytron, 1991] αποτελεί ΙΔΙΟΤΗΤΑ μιας IR σε σχέση με το πως επιλέγουμε να χειριστούμε τις μεταβλητές

Definition (Ιδιότητα της στατικής απλής ανάθεσης)

Ένα πρόγραμμα βρίσκεται σε μορφή SSA αν κάθε μεταβλητή ορίζεται στατικά το πολύ μία φορά, δηλ. δεν υπάρχουν δύο τοποθεσίες στο πρόγραμμα οι οποίες να αναθέτουν στην ίδια μεταβλητή

- TAC

```
p = a + b;
q = p - c;
p = q * d;
p = e - p2;
q = p + q;
```

- SSA-TAC

```
p1 = a + b;
q1 = p1 - c;
p2 = q1 * d;
p3 = e - p2;
q2 = p3 + q1;
```

Η συνάρτηση ϕ (1)

- Η ίδια μεταβλητή μπορεί να οριστεί μέσω δύο ή παραπάνω μονοπατιών ροής ελέγχου σε ένα πρόγραμμα
- Έστω το τμήμα κώδικα

```
if (flag)
  x = -1;
else
  x = 1;
y = x * a;
```

- Στη μορφή SSA χρησιμοποιούμε διαφορετικά ονόματα για τη μεταβλητή x στα δύο μονοπάτια
- Η συνάρτηση ϕ (phi) επανασυνδέει τις $x1, x2$ στη $x3$ έχοντας την τιμή της μίας ή της άλλης ανάλογα με την τιμή της συνθήκης *flag*
- Έτσι έχουμε

```
if (flag)
  x1 = -1;
else
  x2 = 1;
x3 = phi(x1, x2);
y = x3 * a;
```

Η συνάρτηση ϕ (2)

- Η συνάρτηση ϕ φέρει τόσα ορίσματα όσα είναι και τα βασικά μπλοκ τα οποία είναι προηγηθέντα (predecessors) του BB (οδηγούν τη ροή ελέγχου) στο οποίο πρόκειται να χρησιμοποιηθεί (δηλ. όσες είναι οι εισερχόμενες ακμές)
- Κάθε όρισμα συνδέεται με μοναδικό τρόπο με ένα και μόνο από τα προηγηθέντα BB
- Το αποτέλεσμα μιας συνάρτησης ϕ είναι εκείνη η εκδοχή της μεταβλητής η οποία προέρχεται από το μονοπάτι ροής ελέγχου το οποίο τελικά οδηγεί κατά την εκτέλεση του προγράμματος στο εν λόγω BB
- Οι συναρτήσεις ϕ τοποθετούνται πριν από τις πραγματικές εντολές σε ένα BB και θεωρείται ότι υπολογίζονται ταυτόχρονα

Απλή μέθοδος για την κατασκευή της μορφής SSA

- Αλγόριθμος δύο φάσεων [Aycock and Horspool, 2000] για την απλή γέννηση SSA
  1. Εισαγωγή ϕ-συναρτήσεων για κάθε μεταβλητή σε κάθε BB με αριθμό ορισμάτων ίσο με τον αριθμό των εισερχόμενων ακμών
  2. Φάση ελαχιστοποίησης των ϕ-συναρτήσεων
     - 2.1 Διαγραφή των ϕ-συναρτήσεων της μορφής: $V_i \leftarrow \phi(V_i, V_i, \ldots, V_i)$
     - 2.2 Διαγραφή των ϕ-συναρτήσεων της μορφής

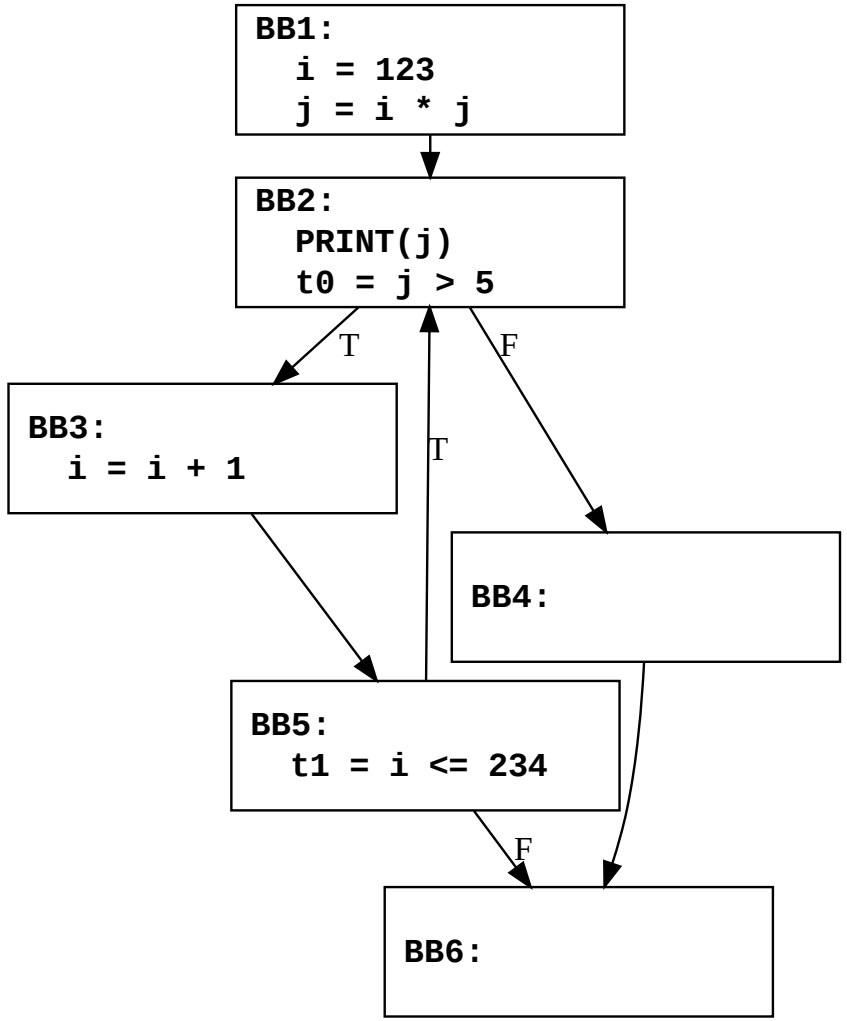
$$V_i \leftarrow \phi(V_{x1}, V_{x2}, \ldots, V_{xk}), \qquad \text{όπου} \qquad x_1, x_2, \ldots, x_k \in \{i, j\}$$

  3. Απαρίθμηση τιμών (value numbering) για τη δεικτοδότηση κάθε νέου ορισμού μιας μεταβλητής (επαναλαμβάνεται)

Παράδειγμα πηγαίου προγράμματος και το TAC IR του

- Πηγαίο πρόγραμμα

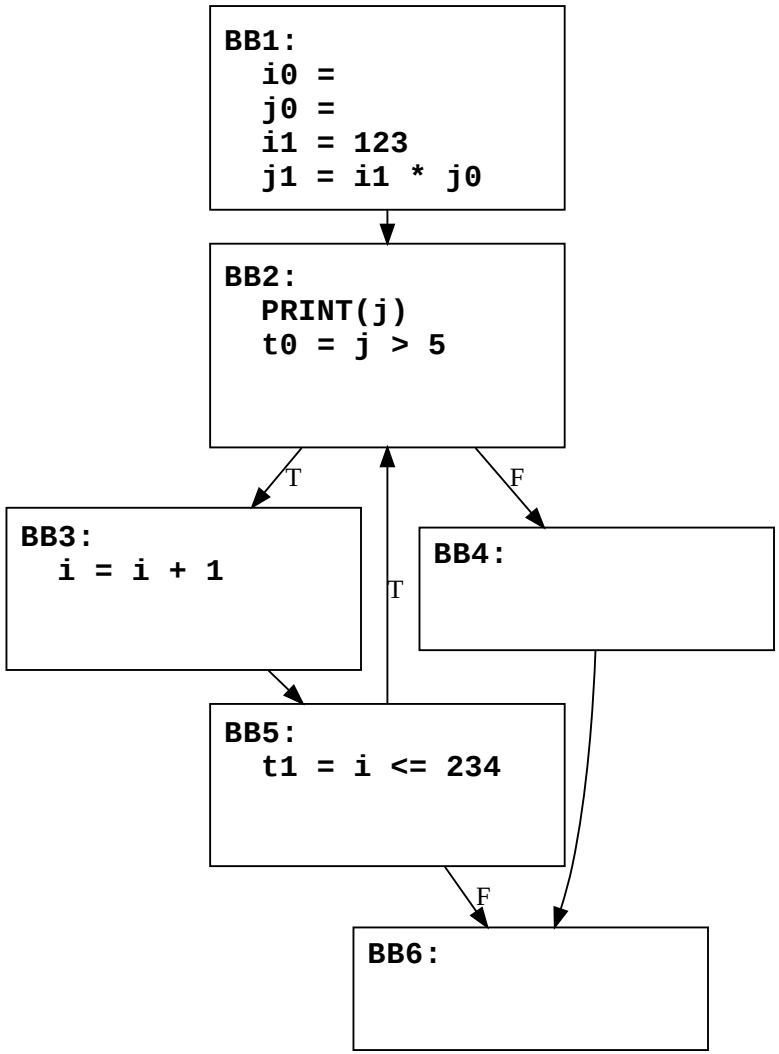
```
i = 123;
j = i * j;
do
{
  PRINT(j);
  if (j > 5)
  {
    i = i + 1;
  }
  else
  {
    break;
  }
} while (i <= 234);
```



Εξαγωγή SSA από πηγαίο πρόγραμμα: Βήμα 1

- IR

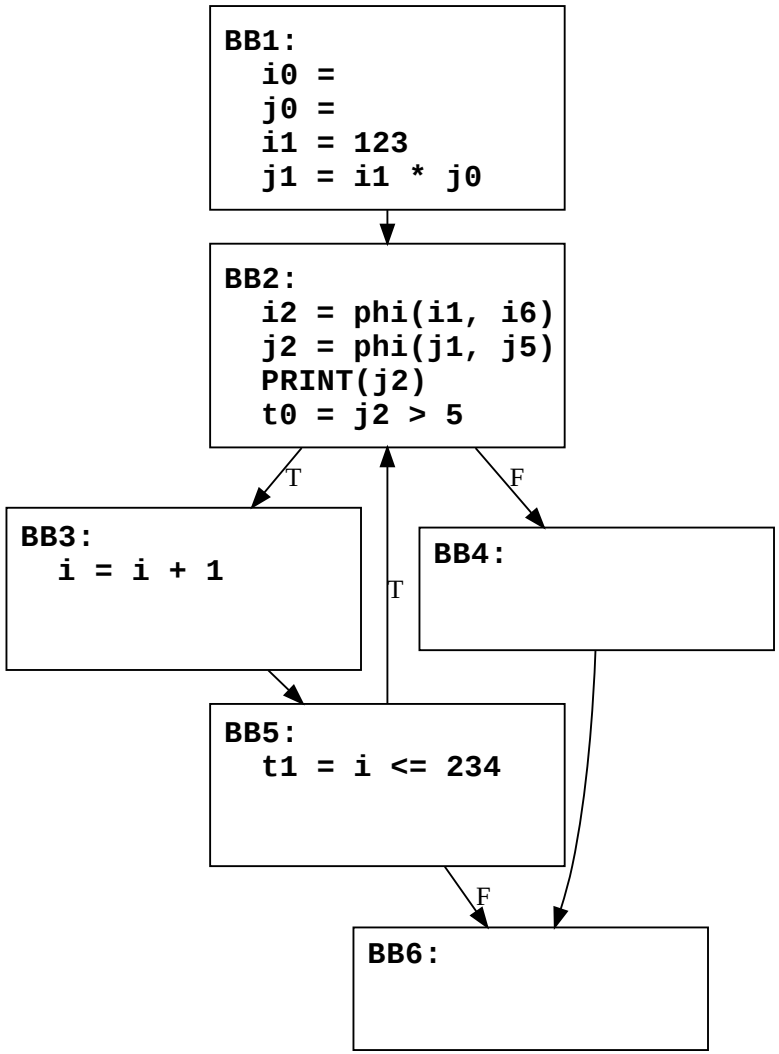
```
BB1:
  i1 = 123;
  j1 = i1 * j0;
BB2:
  PRINT(j);
  t0 = j > 5;
  if (t0) goto BB3; else goto BB4;
BB3:
  i = i + 1;
  goto BB5;
BB4:
  goto BB6;
BB5:
  t1 = i <= 234;
  if (t1) goto BB2; else goto BB6;
BB6:
```



Εξαγωγή SSA από πηγαίο πρόγραμμα: Βήμα 2

- IR

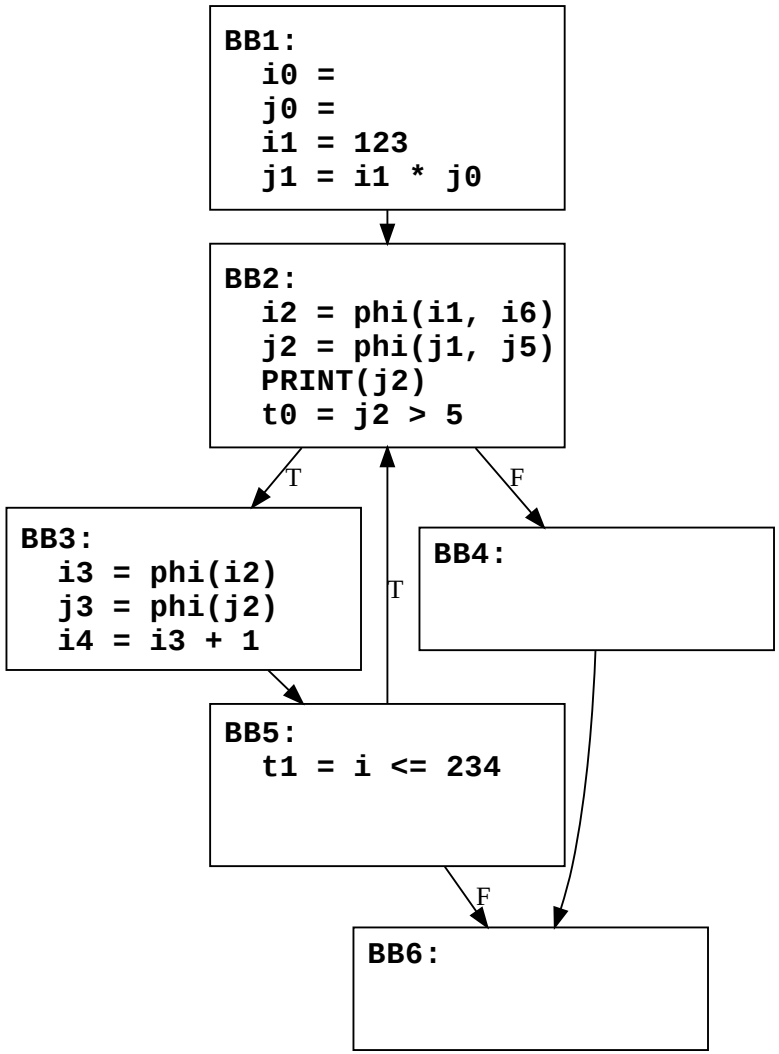
```
BB1:
  i1 = 123;
  j1 = i1 * j0;
BB2:
  i2 = phi(i1, i6);
  j2 = phi(j1, j5);
  PRINT(j2);
  t0 = j2 > 5;
  if (t0) goto BB3; else goto BB4;
BB3:
  i = i + 1;
  goto BB5;
BB4:
  goto BB6;
BB5:
  t1 = i <= 234;
  if (t1) goto BB2; else goto BB5;
BB6:
```



Εξαγωγή SSA από πηγαίο πρόγραμμα: Βήμα 3

- IR

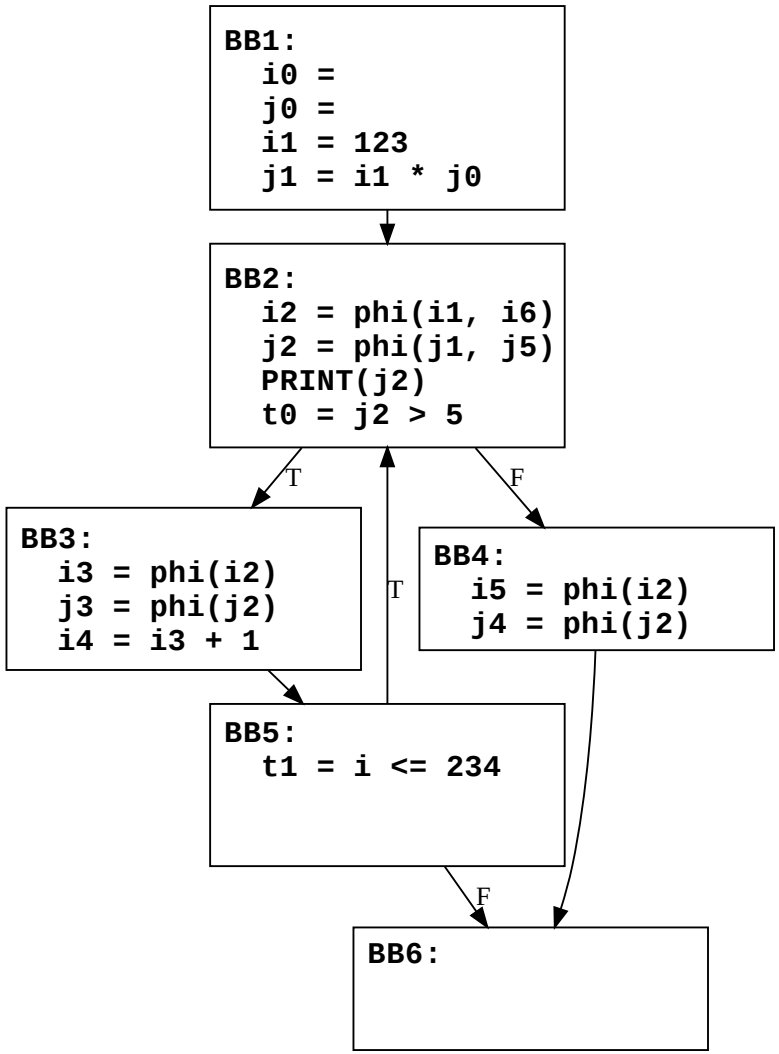
```
BB1:
  i1 = 123;
  j1 = i1 * j0;
BB2:
  i2 = phi(i1, i6);
  j2 = phi(j1, j5);
  PRINT(j2);
  t0 = j2 > 5;
  if (t0) goto BB3; else goto BB4;
BB3:
  i3 = phi(i2); j3 = phi(j2);
  i4 = i3 + 1;
  goto BB5;
BB4:
  goto BB6;
BB5:
  t1 = i <= 234;
  if (t1) goto BB2; else goto BB5;
BB6:
```



Εξαγωγή SSA από πηγαίο πρόγραμμα: Βήμα 4

- IR

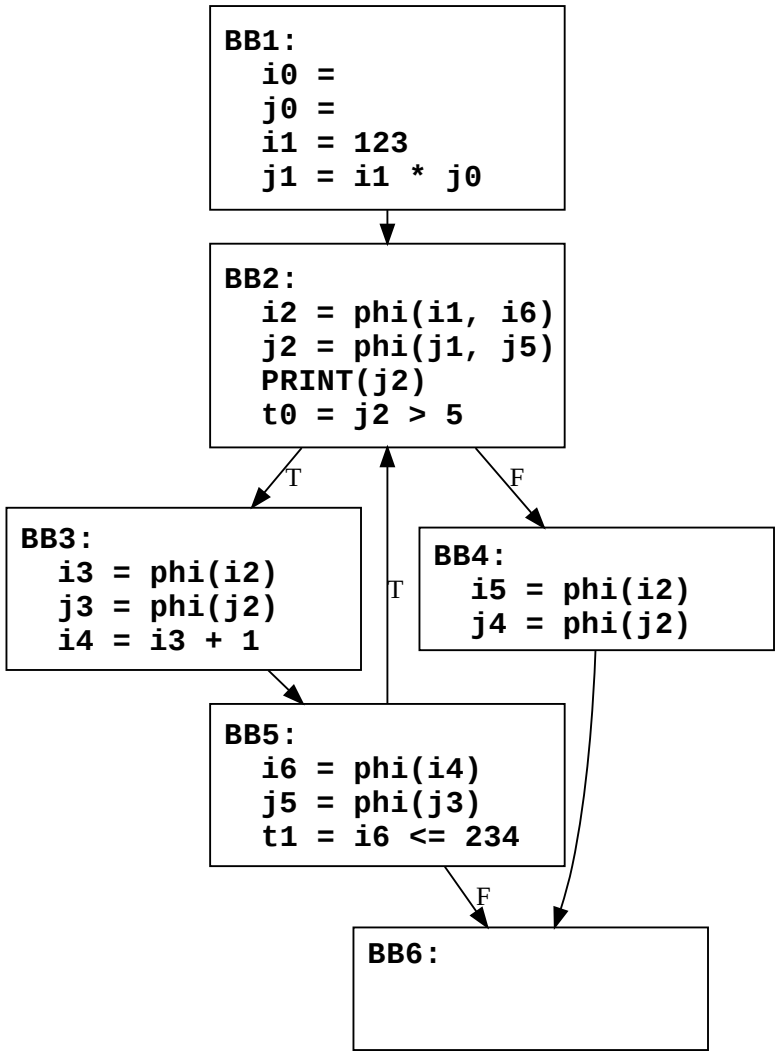
```
BB1:
  i1 = 123;
  j1 = i1 * j0;
BB2:
  i2 = phi(i1, i6);
  j2 = phi(j1, j5);
  PRINT(j2);
  t0 = j2 > 5;
  if (t0) goto BB3; else goto BB4;
BB3:
  i3 = phi(i2); j3 = phi(j2);
  i4 = i3 + 1;
  goto BB5;
BB4:
  i5 = phi(i2); j4 = phi(j2);
  goto BB6;
BB5:
  t1 = i <= 234;
  if (t1) goto BB2; else goto BB5;
BB6:
```



Εξαγωγή SSA από πηγαίο πρόγραμμα: Βήμα 5

- IR

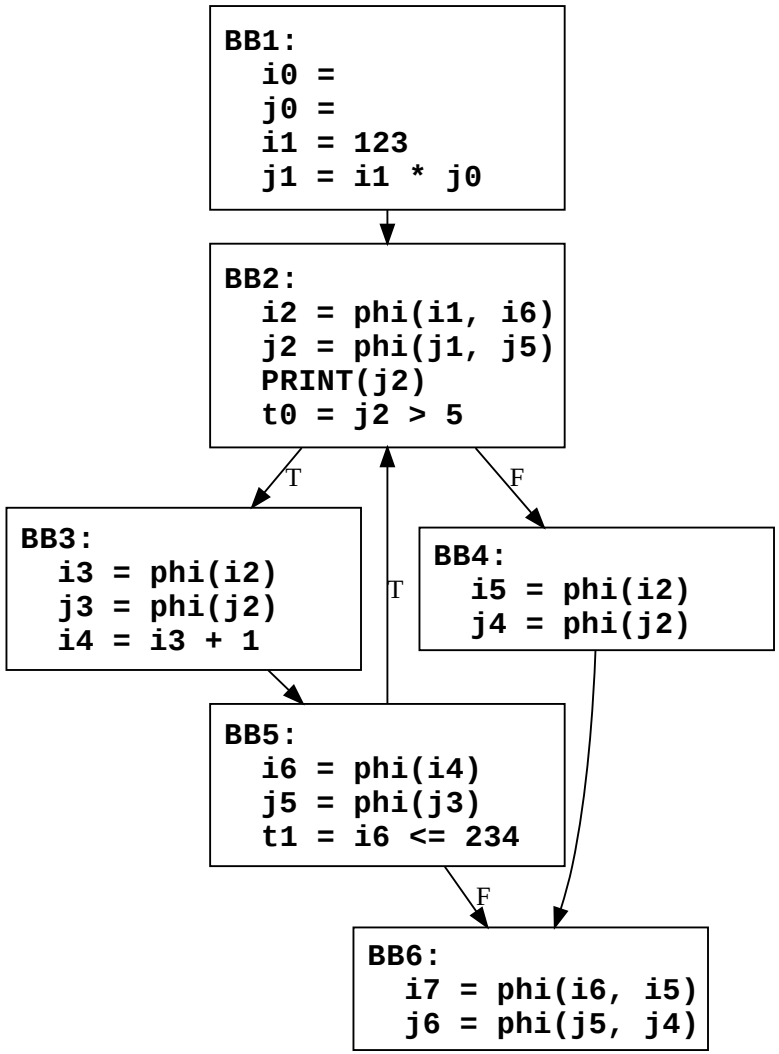
```
BB1:
  i1 = 123;
  j1 = i1 * j0;
BB2:
  i2 = phi(i1, i6);
  j2 = phi(j1, j5);
  PRINT(j2);
  t0 = j2 > 5;
  if (t0) goto BB3; else goto BB4;
BB3:
  i3 = phi(i2); j3 = phi(j2);
  i4 = i3 + 1;
  goto BB5;
BB4:
  i5 = phi(i2); j4 = phi(j2);
  goto BB6;
BB5:
  i6 = phi(i4); j5 = phi(j3);
  t1 = i6 <= 234;
  if (t1) goto BB2; else goto BB5;
BB6:
```



Εξαγωγή SSA από πηγαίο πρόγραμμα: Βήμα 6

- IR

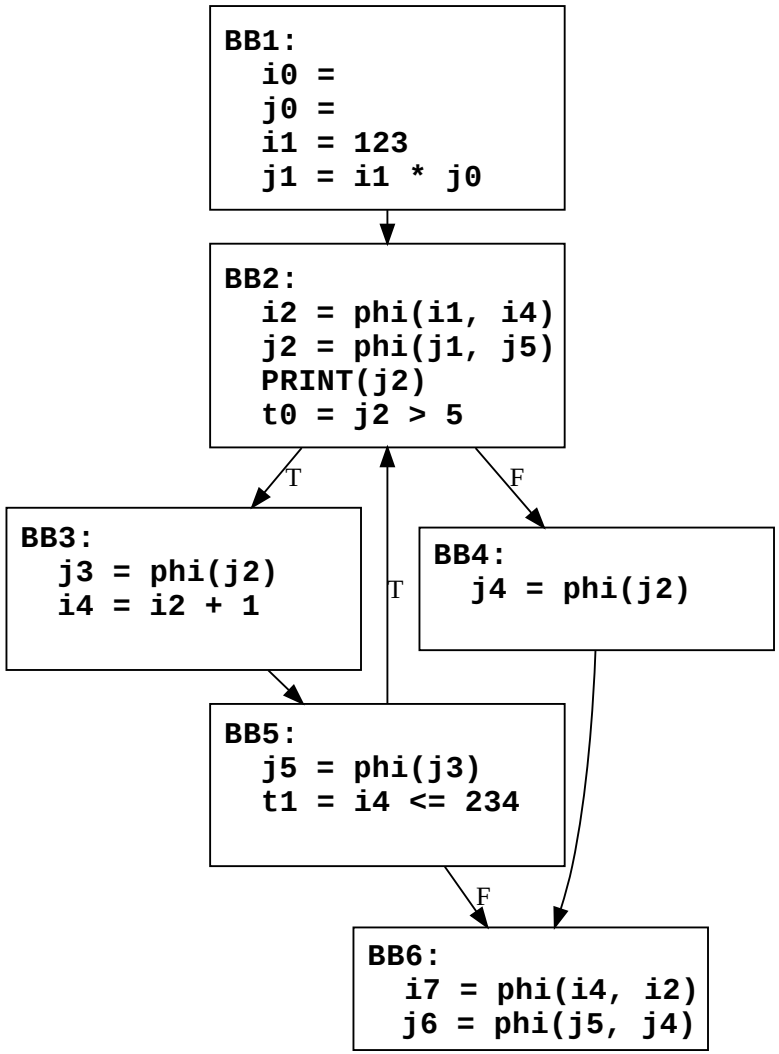
```
BB1:
  i1 = 123;
  j1 = i1 * j0;
BB2:
  i2 = phi(i1, i6);
  j2 = phi(j1, j5);
  PRINT(j2);
  t0 = j2 > 5;
  if (t0) goto BB3; else goto BB4;
BB3:
  i3 = phi(i2); j3 = phi(j2);
  i4 = i3 + 1;
  goto BB5;
BB4:
  i5 = phi(i2); j4 = phi(j2);
  goto BB6;
BB5:
  i6 = phi(i4); j5 = phi(j3);
  t1 = i6 <= 234;
  if (t1) goto BB2; else goto BB5;
BB6:
  i7 = phi(i6, i5);
  j6 = phi(j5, j4);
```



Εξαγωγή SSA από πηγαίο πρόγραμμα: Βήμα 7

- Εξουδετέρωση περιττών εκδοχών (versions) της μεταβλητής i

$$\left.\begin{array}{l} i_2 \leftarrow \phi(i_1, i_6) \\ i_3 \leftarrow \phi(i_2) \\ i_5 \leftarrow \phi(i_2) \\ i_6 \leftarrow \phi(i_4) \\ i_7 \leftarrow \phi(i_6, i_5) \end{array}\right\} \left[\begin{array}{l} i_3 \equiv i_2 \\ i_5 \equiv i_2 \\ i_6 \equiv i_4 \end{array}\right] \Longrightarrow \left\{\begin{array}{l} i_2 \leftarrow \phi(i_1, i_4) \\ i_7 \leftarrow \phi(i_4, i_2) \end{array}\right.$$

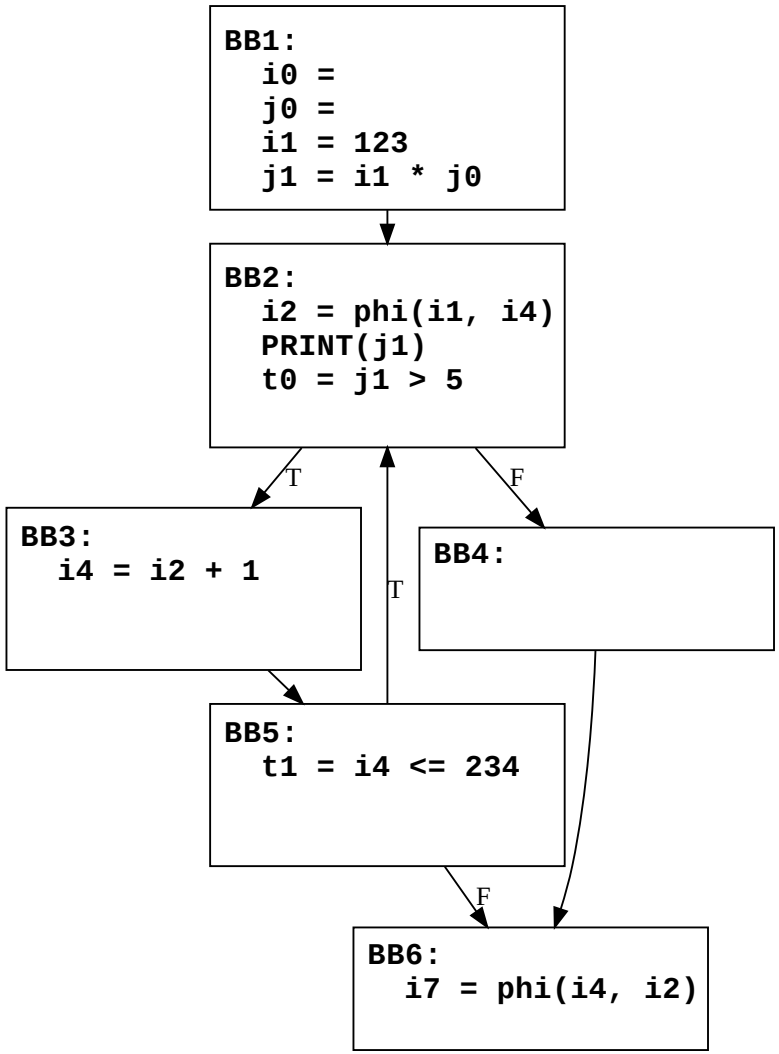


Εξαγωγή SSA από πηγαίο πρόγραμμα: Βήμα 8

- Εξουδετέρωση περιττών εκδοχών (versions) της μεταβλητής j

$$\left.\begin{array}{l} j_2 \leftarrow \phi(j_1, j_5) \\ j_3 \leftarrow \phi(j_2) \\ j_4 \leftarrow \phi(j_2) \\ j_5 \leftarrow \phi(j_3) \\ j_6 \leftarrow \phi(j_5, j_4) \end{array}\right\} \left[\begin{array}{l} j_3 \equiv j_2 \\ j_4 \equiv j_2 \\ j_5 \equiv j_4 \equiv j_3 \end{array}\right] \Longrightarrow$$

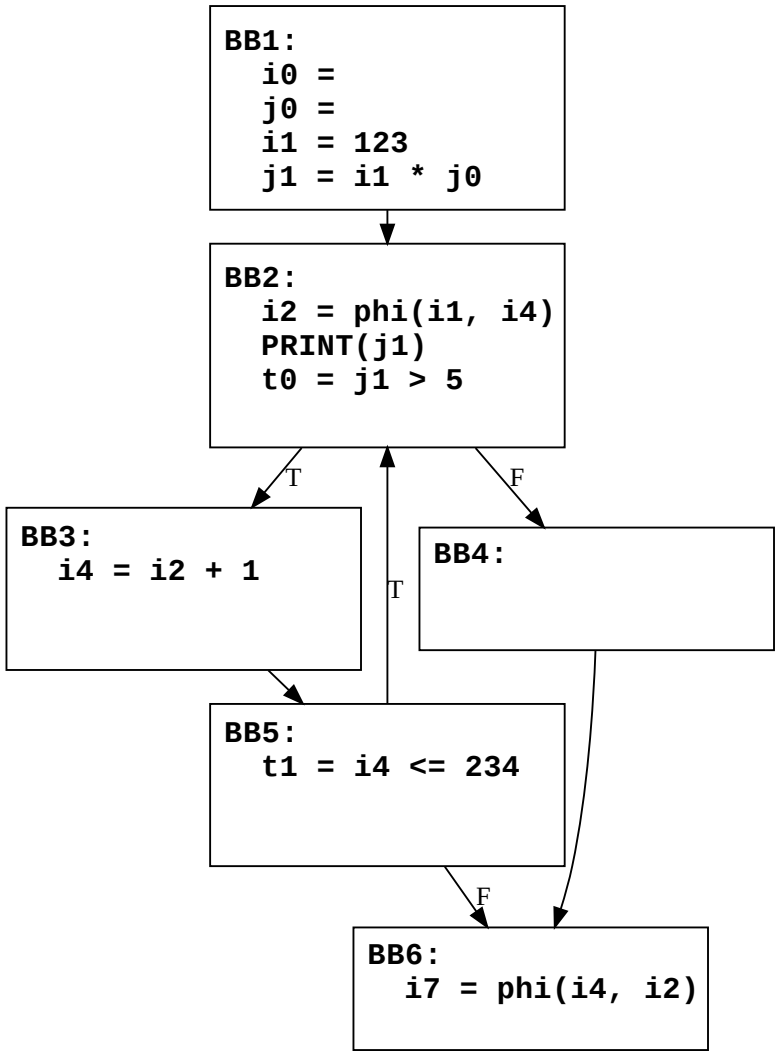
$$j_2 \leftarrow \phi(j_1, j_2) \left[\; j_2 \equiv j_1 \;\right] \Longrightarrow \text{none}$$



Μετά την εξαγωγή SSA από πηγαίο πρόγραμμα

- IR

```
BB1:
  i1 = 123;
  j1 = i1 * j0;
BB2:
  i2 = phi(i1, i4);
  PRINT(j1);
  t0 = j1 > 5;
  if (t0) goto BB3; else goto BB4;
BB3:
  i4 = i2 + 1;
  goto BB5;
BB4:
  goto BB6;
BB5:
  t1 = i4 <= 234;
  if (t1) goto BB2; else goto BB5;
BB6:
  i7 = phi(i4, i2);
```



Κίνητρα για τη χρήση SSA

- Η μορφή SSA προτιμάται στις μοντέρνες σχεδιάσεις μεταγλωττιστών (GCC 4.5.1, LLVM 2.7, Machine-SUIF 2, libfirm 1.17.0) για τους εξής λόγους:
  - Οι αναλύσεις ροής δεδομένων (data flow analyses) είναι περισσότερο αποδοτικές
  - Διευκολύνεται η εφαρμογή συγκεκριμένων τύπων βελτιστοποιήσεων
  - Ορισμένες αναλύσεις και βελτιστοποιήσεις όπως η εξουδετέρωση κοινής υποεκφράσεως (CSE) είναι έμφυτες στην αναπαράσταση (πραγματοποιούνται κατά την εξαγωγή της SSA)
  - Οι αλυσίδες χρήσης-ορισμού (def-use chains) είναι εμφανείς
  - Οι αλυσίδες ορισμού-χρήσης (use-def chains) είναι ευκολότερο να αναπαρασταθούν

Αναφορές του μαθήματος Ι

A. V. Aho, R. Sethi, and J. D. Ullman, *Μεταγλωττιστές: Αρχές, Τεχνικές και Εργαλεία*, με την επιμέλεια των: Άγγελος Σπ. Βώρος και Νικόλαος Σπ. Βώρος και Κων/νος Γ. Μασσέλος, **κεφάλαια 6, 6.1.1, 6.2, 6.2.1, 6.2.4, 8.4, 9.6.1**, 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.

R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck, "Efficiently computing static single assignment form and the control dependence graph," *ACM Transactions on Programming Languages and Systems*, vol. 13, no. 4, pp. 451–490, October 1991.

Αναφορές του μαθήματος II


J. Aycock and N. Horspool, "Simple generation of static single assignment form," in *Proceedings of the 9th International Conference in Compiler Construction*, Berlin, Germany, March 2000, pp. 110–124.


Machine-SUIF research compiler. [Online]. Available: `http://www.eecs.harvard.edu/hube/research/machsuif.html`


The GNU compiler collection homepage. [Online]. Available: `http://gcc.gnu.org`


LLVM (Low-Level Virtual Machine) compiler homepage. [Online]. Available: `http://www.llvm.org`


LANCE C compiler. [Online]. Available:

`http://www.lancecompiler.com`