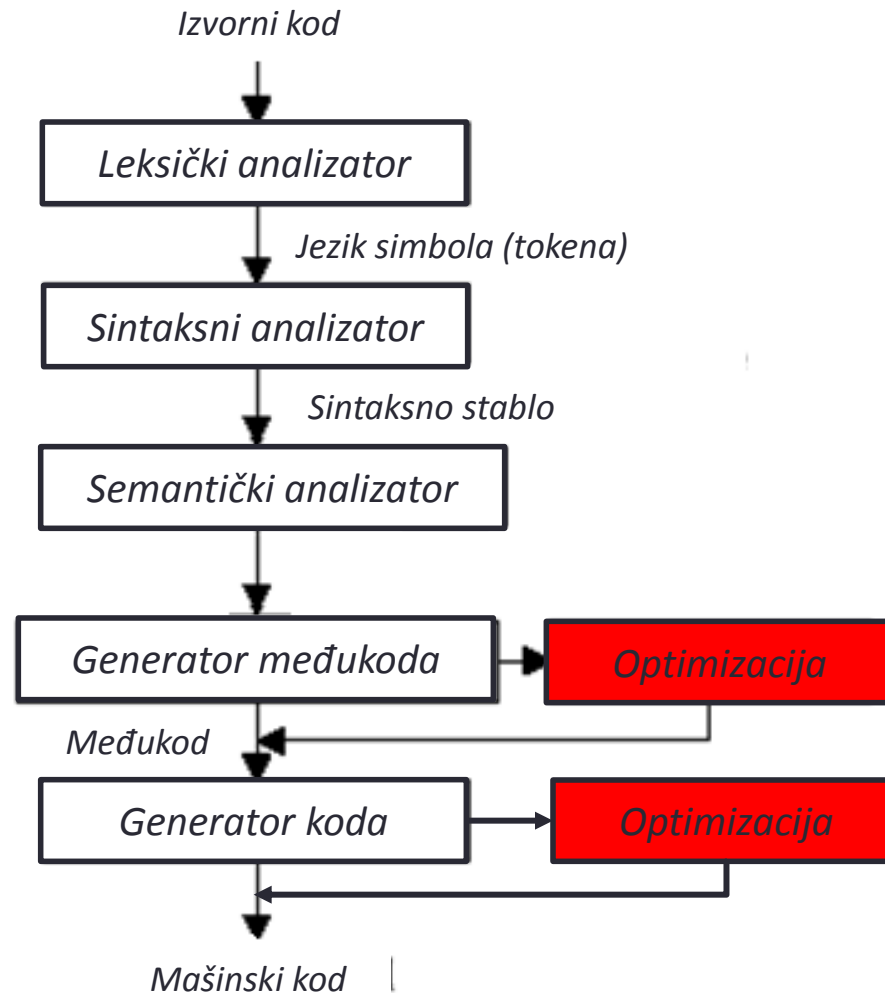


PROGRAMSI PREVODIOCI

- Optimizacija koda -

Struktura kompilatora



Osnovna (*peephole*) optimizacija

- Strategija generisanja koda naredba za naredbom često daje ciljni kod programa koji sadrži redundatne naredbe i neoptimalne konstrukcije.
- U okviru *peephole* optimizacije cilj je da se popravi kod jednosatavnim transformacijama koje se
 - izvršavaju nad kratkom sekvencom naredbi
 - brzo izvode
- Funkcionalnost koda ne sme da bude narušena.

Osnovne transformacije

- Elininisanje redundantnih naredbi.
- Optimizacija toka programa
- Algebarska uprošćenja
- Korišćenje mašinskih idioma

Eliminisanje redundantnih naredbi

Eliminisanje STORE-LOAD parova:

Često se rezultat jedne operacije odmah koristi u narednoj. Kao posledica primene formalnih algoritama za prevođenje (naredba po naredba), u takvim slučajevima se generišu parovi naredbi:

Store tk

Load tk

Ovakav skup naredbi se prosto izostavlja (eliminise).

Eliminacija se ne vrši ako je druga (Load) instrukcija obeležena (postoji skok na tu instrukciju iz drugih delova koda).

Store tk

Lab1: Load tk

Eliminisanje redundantnih naredbi

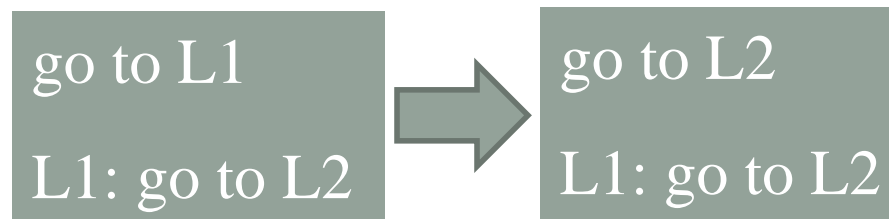
Eliminisanje STORE-LOAD parova – Primer:

| <u>Naredba</u> | <u>medjukod</u> | <u>izlazni kod</u> | <u>optimalan kod</u> |
|-----------------|-----------------|--------------------|----------------------|
| $a = b * c + d$ | $t1 := b * c$ | <i>Load b</i> | <i>Load b</i> |
| | $t2 := t1 + d$ | <i>Mul c</i> | <i>Mul c</i> |
| | $a := t2$ | <i>Store t1</i> | <i>Add d</i> |
| | | <i>Load t1</i> | <i>Store a</i> |
| | | <i>Add d</i> | |
| | | <i>Store t2</i> | |
| | | <i>Load t2</i> | |
| | | <i>Store a</i> | |

Optimizacija toka

- U toku generisanja međukoda (ili izlajnog koda) često se javljaju skokovi na skokove
 - Bezuslovni skok na bezuslovni,
 - Uslovni skok na bezuslovni,
 - Bezuslovni skok na uslovni

1. Transformacija bezuslovnog skoka na bezuslovni:

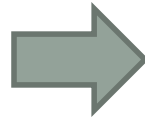


Ako posle ove transformacije nema skokova na naredbu označenu sa **L1**: onda se ona može zameniti neoznačenom naredbom, a nekad i potpuno eliminisati.

Optimizacija toka

2. Transformacija uslovnog skoka na bezuslovni:

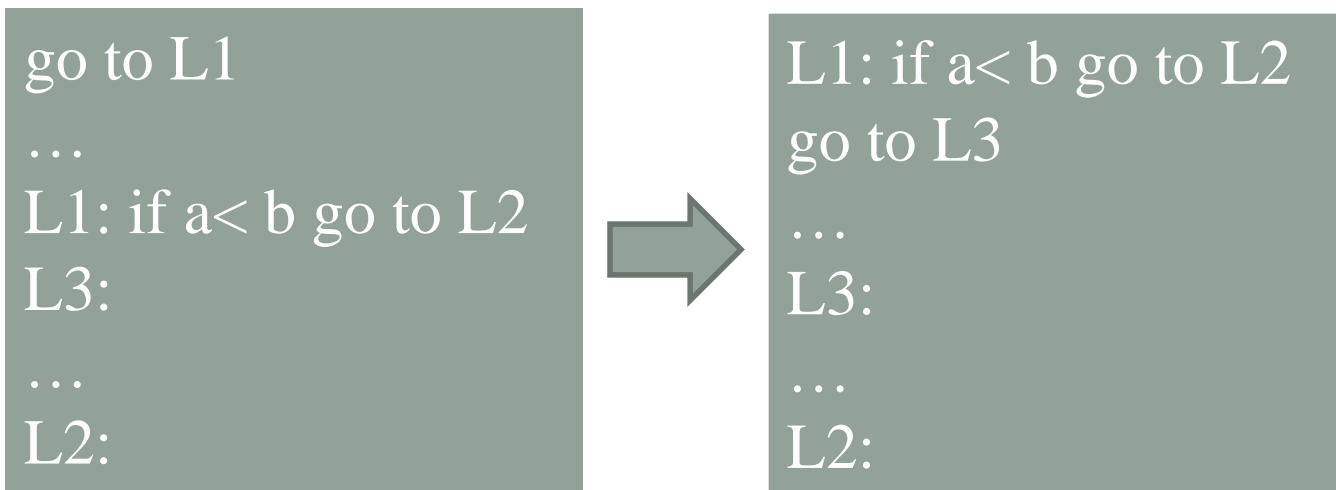
if $a < b$ go to L1
L1: go to L2



if $a < b$ go to L2
L1: go to L2

Optimizacija toka

3. Transformacija bezuslovnog skoka na uslovni:



Broj naredbi je ostao isti ali je drugi slučaj efikasniji:

- U prvom slučaju se uvek izvršava **goto** i **if** ,
- U drugom se uvek izvršava **if** , a **go to** samo kada navedeni uslov nije ispunjen

Nedosegljiv kod

- Eliminišu se naredbe koje se u određenom kontekstu neće nikada izvršiti.
 - Npr. naredba iza naredbe bezuslovnog skoka, iza return naredbe,... se nikada neće izvršiti.

Primer nedosegljivog koda u C-u:

```
# define debug 0
if (debug) {
//Stampanje poruke o gresci
}
```

Nedosegljiv kod

Međukod ove naredbe (bez obrade pretprocesorskih direktiva) bio bi:

```
if debug = 1 go to L1  
go to L2  
L1: print(...  
L2:
```

Posle eliminisanja GOTO na GOTO ova sekvenca dobija oblik:

```
if debug != 1 go to L2  
print(...  
L2:
```

Nedosegljiv kod

Posle obrade pretprocesorskih direktiva pa nakon navedenih transformacija dobija se:

```
if 0 != 1 go to L2  
  print(...  
L2:
```

Kako je uslov if naredbe uvek ispunjen uslovna naredba može biti zamenjena bezuslovnom naredbom go to L2, pa se i sekvenca naredbi za štampanje poruke u ovom slučaju može potpuno eliminisati jer će biti nedosegljiva.

Algebarska uprošćenja

1. Eliminacija operacija sa neutralnim elementom

- Sledeće naredbe mogu biti potpuno eliminisane:

$$X = X + 0$$

$$X = X * 1$$

- Uprošćenja:

$$X = Y + 0$$

zamenjuje se sa

$$X = Y$$

$$X = Y * 1$$

zamenjuje se sa

$$X = Y$$

Algebarska uprošćenja

2. Direktna redukcija - složene operacije se zamenjuju jednostavnijim koje se brže izvršavaju.

- Stepenovanje je moguće zameniti višestrukim množenjem $X^2 = X * X$
- Množenje i deljenje stepenom dvojke može se zameniti šift operacijama.
- Deljenje realnom konstantnom može se zameniti množenjem recipročnom vrednošću.

Korišćenje mašinskih idioma

- Ovo je mašinski zavisna transformacija u kojoj se koriste prednosti i specifičnosti asemblerskog jezika ciljne mašine.
- Npr. neki asemblerski jezici imaju mogućnost izvršavanja auto-inkrementa ili auto-dekrementa nekog operanda pre ili posle izvršenja neke instrukcije.
 - U 8086 assembleru *loop* instrukcija dekrementira sadržaj registra cx

Dodatna optimizacija (mašinski nezavisna)

```
void quiksort (m,n)
int m,n ;
{
int i,j;
int v, x;
if (n <= m ) return;
/* deo koji optimizujemo pocinje ovde */
i = m-1; j = n; v = a[n];
while ( j ) {
do i = i +1 ; while ( a[i] < v );
do j = j -1 ; while ( a[j] > v );
if ( i >= j ) break;
x = a[i]; a[i] = a[j]; a[j] = x;
}
x = a[i]; a[i] = a[n]; a[n] = x;
/* Kraj dela koda koji nas interesuje */
quiksort (m,j-1); quiksort ( i+1, n );
}
```

Kod za QUIKSORT algoritam

Troadresni kod za analizirani deo programa

```
(1)    i := m - 1
(2)    j := n
(3)    t1 := 4 * n
(4)    v := a [t1]
(5)    i := i + 1
(6)    t2 := 4 * i
(7)    t3 := a [t2]
(8)    if t3 < v go to (5)
(9)    j := j - 1
(10)   t4 := 4 * j
(11)   t5 := a [t4]
(12)   if t5 > v go to (9)
(13)   if i <= j go to (23)
(14)   t6 := 4 * i
(15)   x := a [t6]
```

```
(16)   t7 := 4 * i
(17)   t8 := 4 * j
(18)   t9 := a [t8]
(19)   a [t7] := t9
(20)   t10 := 4 * j
(21)   a [t10] := x
(22)   go to (5)
(23)   t11 := 4 * i
(24)   x := a [t11]
(25)   t12 := 4 * i
(26)   t13 := 4 * n
(27)   t14 := a [t13]
(28)   a [t12] := t14
(29)   t15 := 4 * n
(30)   a [t15] := x
```

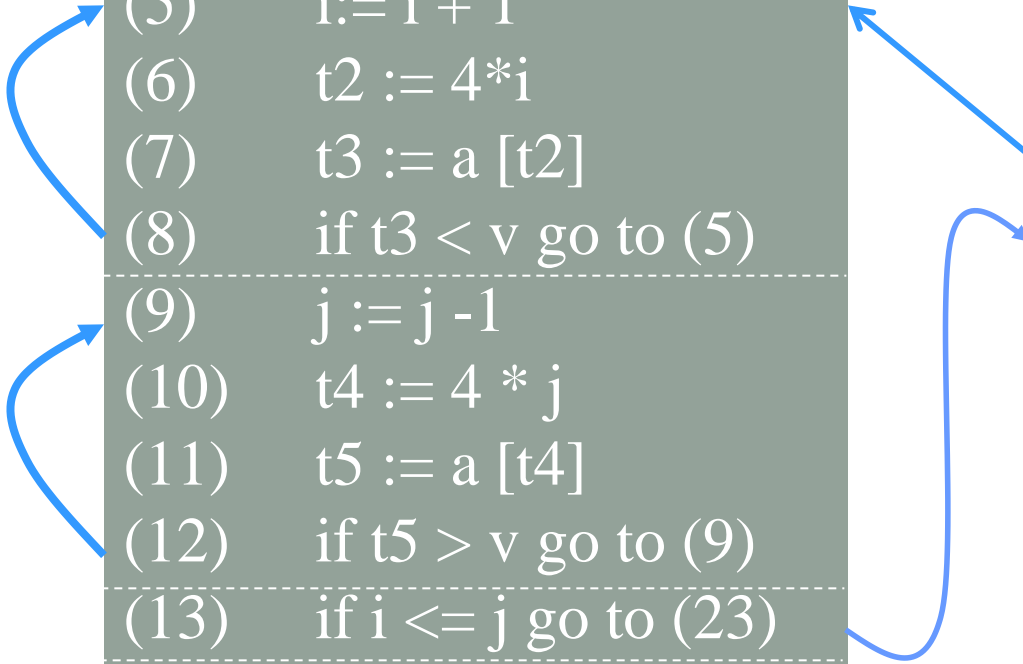
1. Korak: Kreiranje grafa toka upravljanja (*Control-flow graph* - CFG)

- Podeliti kod programa na blokove (čvorove grafa) pri čemu se unutrašnje naredbe bloka izvršavaju sekvencijalno (bez skokova)
 - Poslednja naredba u bloku može da bude naredba skoka
 - Prva naredba u bloku može da bude odredišna naredba skoka (naredba na koju se skače)
- Grane u grafu predstavljaju tok upravljanja (skokove u programu)

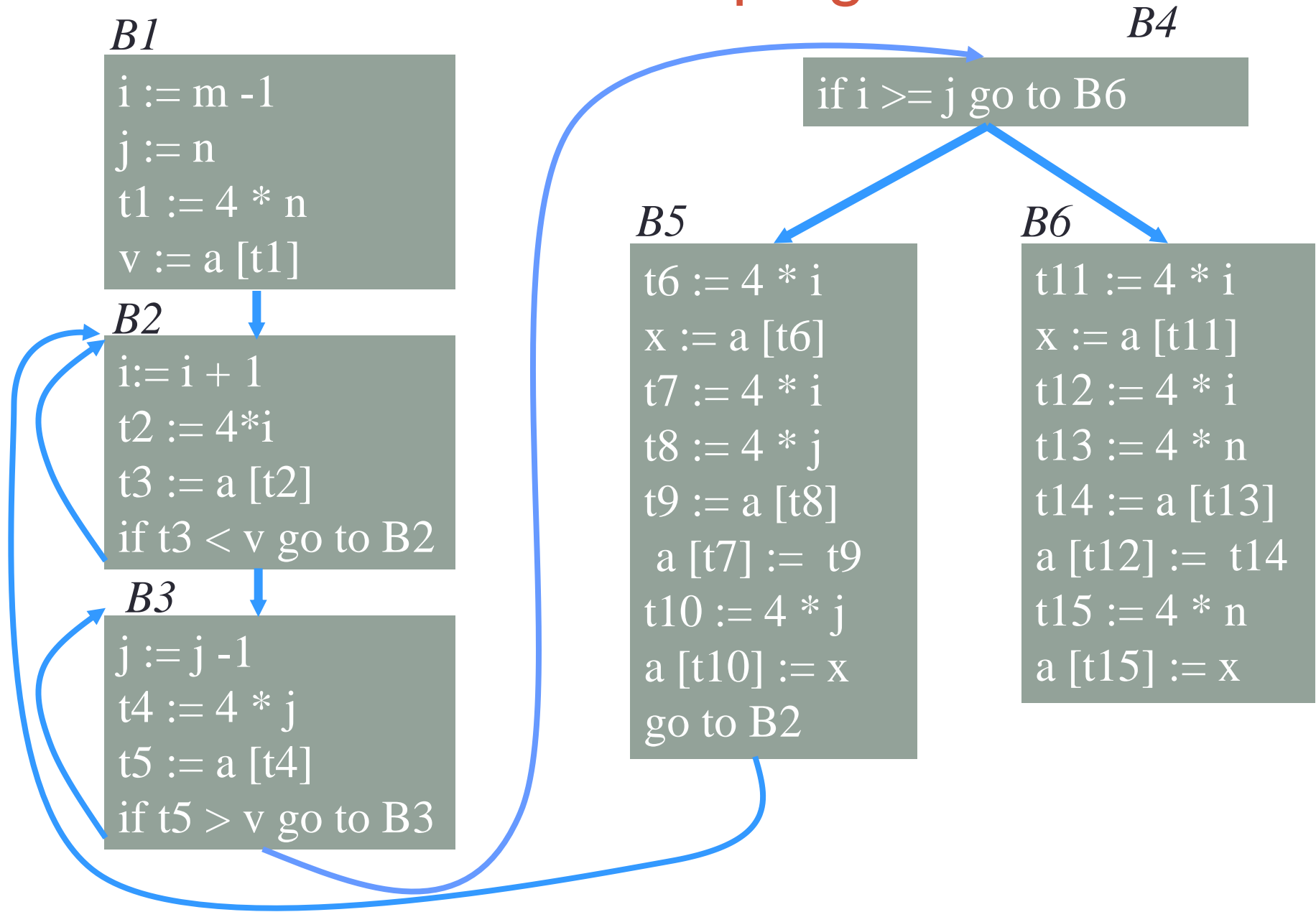
Podela koda na blokove

```
(1)    i := m - 1
(2)    j := n
(3)    t1 := 4 * n
(4)    v := a [t1]
-----
(5)    i := i + 1
(6)    t2 := 4 * i
(7)    t3 := a [t2]
(8)    if t3 < v go to (5)
-----
(9)    j := j - 1
(10)   t4 := 4 * j
(11)   t5 := a [t4]
(12)   if t5 > v go to (9)
-----
(13)   if i <= j go to (23)
-----
(14)   t6 := 4 * i
(15)   x := a [t6]
```

```
(16)   t7 := 4 * i
(17)   t8 := 4 * j
(18)   t9 := a [t8]
(19)   a [t7] := t9
(20)   t10 := 4 * j
(21)   a [t10] := x
(22)   go to (5)
-----
(23)   t11 := 4 * i
(24)   x := a [t11]
(25)   t12 := 4 * i
(26)   t13 := 4 * n
(27)   t14 := a [t13]
(28)   a [t12] := t14
(29)   t15 := 4 * n
(30)   a [t15] := x
```



CFG za analizirani deo programa



Mašinski nezavisna optimizacija

- Lokalna – na nivou jednog bloka
- Globalna – na nivou celog potprograma

Zajednički podizrazi na lokalnom nivou

Izraz E naziva se zajedničkim ako je pre toga već bio izračunat. Npr. u bloku B5 izračunato je $t6$ kao $4*i$, a posle toga se ista vrednost pamti i kao $t7$. Isto važi i za $t8$ i $t10$. Na svim mestima gde se koristi $t7$ može da stoji $t6$, a umesto $t10$ može da stoji $t8$.

B5 pre transformacije

```
t6 := 4 * i
x := a [t6]
t7 := 4 * i
t8 := 4 * j
t9 := a [t8]
  a [t7] := t9
t10 := 4 * j
a [t10] := x
go to B2
```

B5 posle transformacije

```
t6 := 4 * i
x := a [t6]
t8 := 4 * j
t9 := a [t8]
  a [t6] := t9
a [t8] := x
go to B2
```

Zajednički podizrazi na lokalnom nivou

U bloku B6 izračunato je t11 kao $4*i$, a posle toga se ista vrednost pamti i kao t12. Isto važi i za t13 i t15. Na svim mestima gde se koristi t12 može da stoji t11, a umesto t15 može da stoji t13.

B6 pre transformacije

```
t11 := 4 * i
x := a [t11]
t12 := 4 * i
t13 := 4 * n
t14 := a [t13]
a [t12] := t14
t15 := 4 * n
a [t15] := x
```

B6 posle transformacije

```
t11 := 4 * i
x := a [t11]
t13 := 4 * n
t14 := a [t13]
a [t11] := t14
a [t13] := x
```


Zajednički podizrazi na globalnom nivou

B1

```
i := m - 1  
j := n  
t1 := 4 * n  
v := a [t1]
```

B2

```
i := i + 1  
t2 := 4 * i  
t3 := a [t2]  
if t3 < v go to B2
```

B3

```
j := j - 1  
t4 := 4 * j  
t5 := a [t4]  
if t5 > v go to B3
```

Zajednički podizrazi mogu da se pronalaze i na globalnom nivou ako promenljive koje u njima učestvuju ne menjaju vrednost u međuvremenu.

```
t1 := 4 * n, t13 := 4 * n;    t4 := 4 * j, t8 := 4 * j;  
t2 := 4 * i, t6 := 4 * i, t11 := 4 * i;
```

B4

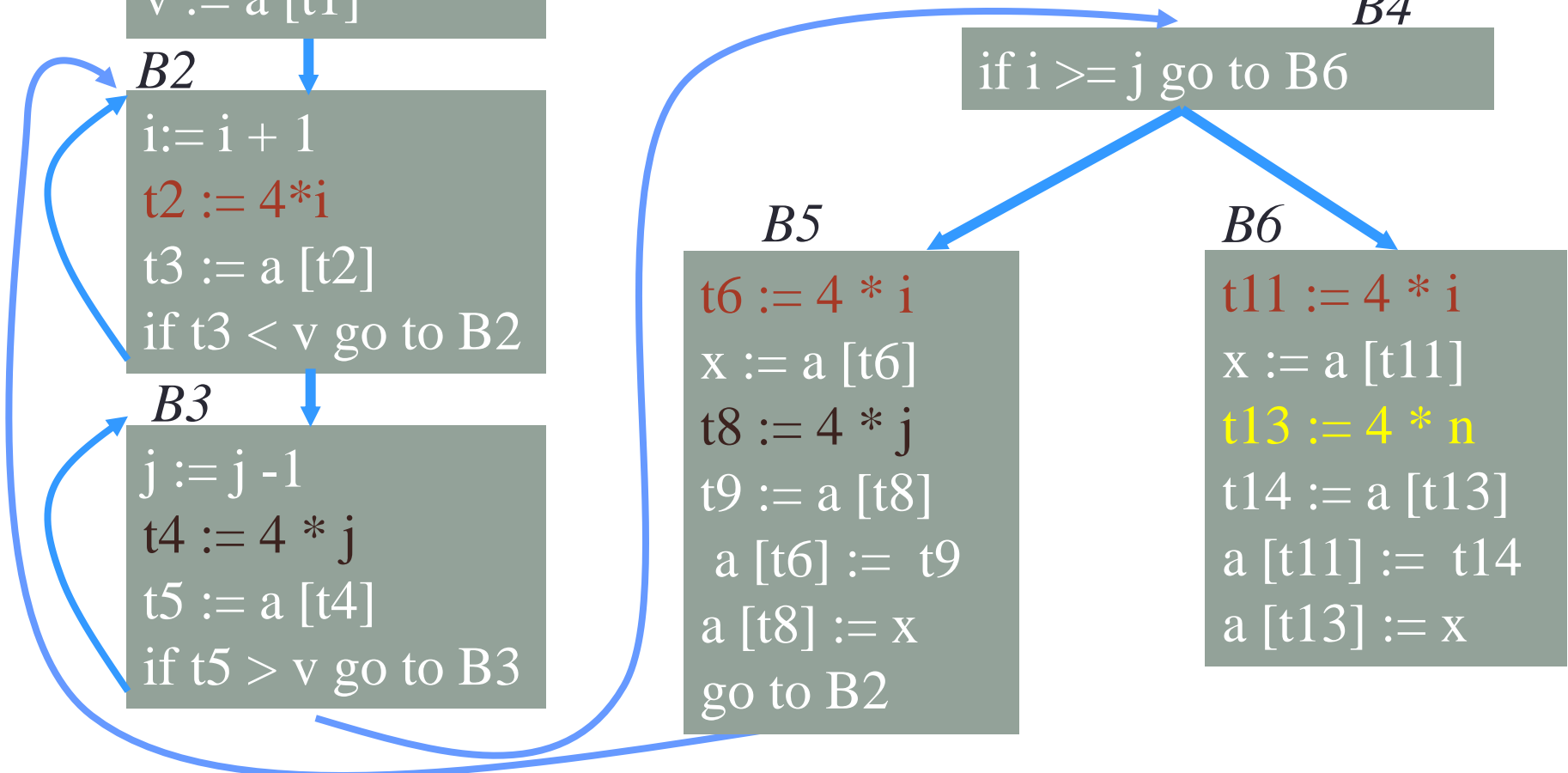
```
if i >= j go to B6
```

B5

```
t6 := 4 * i  
x := a [t6]  
t8 := 4 * j  
t9 := a [t8]  
a [t6] := t9  
a [t8] := x  
go to B2
```

B6

```
t11 := 4 * i  
x := a [t11]  
t13 := 4 * n  
t14 := a [t13]  
a [t11] := t14  
a [t13] := x
```

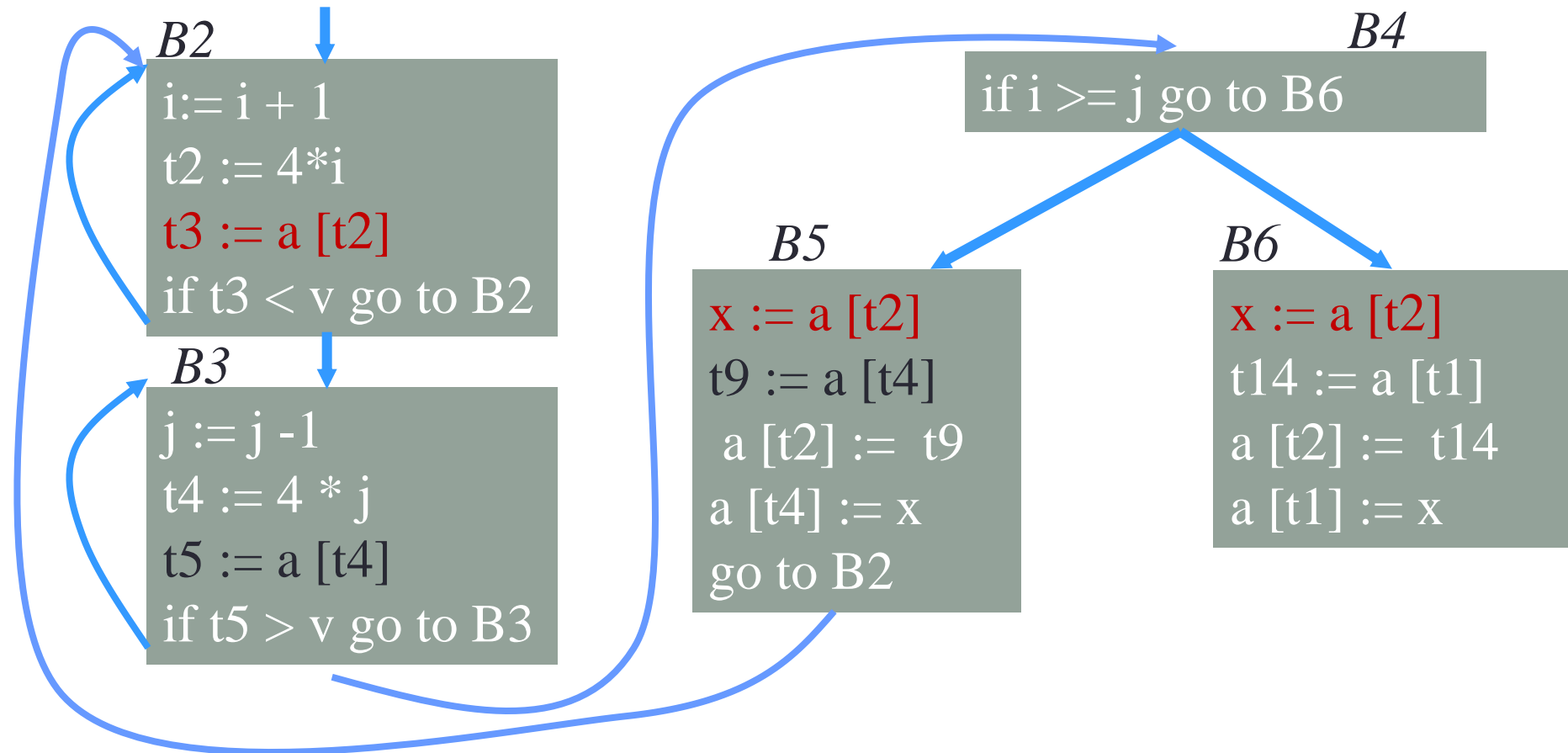


Zajednički podizrazi na globalnom nivou

Nakon eliminacije prethodnih zajedničkih podizraza uočavamo nove:

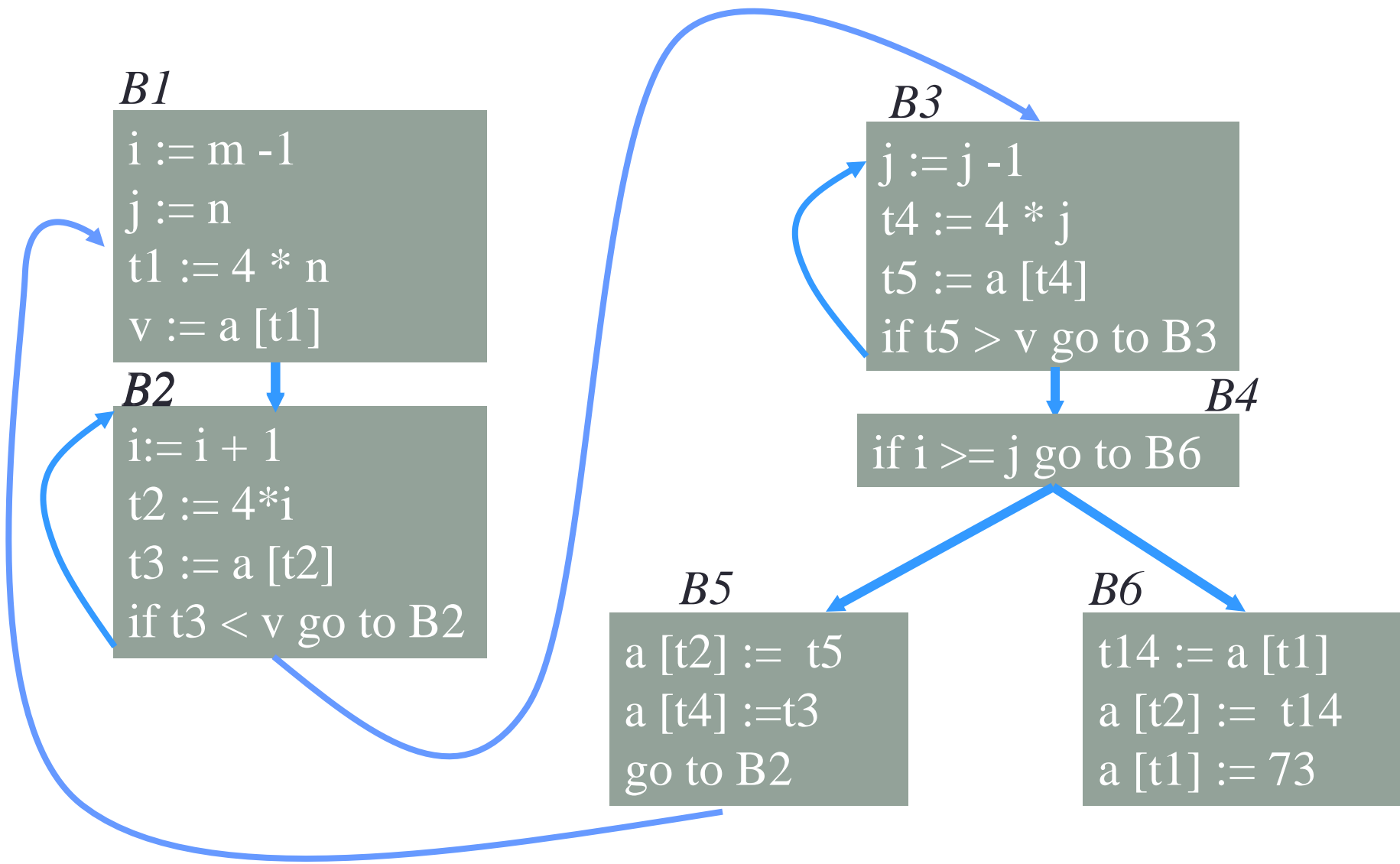
B2: $t3 := a[t2]$, B5: $x := a[t2]$, B6: $x := a[t2]$.

B3: $t5 := a[t4]$, B5: $t9 := a[t4]$.



Zajednički podizrazi na globalnom nivou

Kod nakon eliminacije svih zajedničkih podizraza:



Prostiranje copy naredbe

Posle naredbe oblika $f := g$

svuda gde se koristi f stavi g , a ova naredba eliminiše.

B5 pre transformacije

```
x := t3  
a [t2] := t5  
a [t4] := x  
go to B2
```

*B5 posle prostiranja
copy naredbe $x := t3$*

```
x := t3  
a [t2] := t5  
a [t4] := t3  
go to B2
```

Prostiranje copy naredbe

B6 pre transformacije

```
x := t3  
t14 := a [t1]  
a [t2] := t14  
a [t1] := x
```

*B6 после простiranja
copy naredbe $x := t3$*

```
x := t3  
t14 := a [t1]  
a [t2] := t14  
a [t1] := t3
```

Eliminisanje neaktivnog koda

Neaktivne naredbe su naredbe koje se nikada neće izvršiti ili nemaju nikakav efekat na program. Takođe mogu da postoje i neaktivne (mrtve) promenljive koje se nikada ne koriste u programu. Ovakve naredbe i promenljive često nastaju kao posledica prostiranja copy naredbi. U prethodnom primeru to je bio slučaj sa naredbom $x := t3$ koju smo eliminisali i u bloku B5 i u bloku B6.

*B5 posle eliminisanja
naredbe $x := t3$*

```
a [t2] := t5  
a [t4] := t3  
go to B2
```

*B6 posle eliminisanja
naredbe $x := t3$*

```
t14 := a [t1]  
a [t2] := t14  
a [t1] := t3
```

Optimizacija petlji

- Optimizacija petlji zauzima važno mesto u optimizaciji programa.
- Vreme izvršavanja programa može se značajno smanjiti ukoliko se smanji broj naredbi unutar petlji čak i ako se pri tome poveća broj naredbi van petlje.
- Tehnike za optimizaciju petlji:
 - Pomeranje (migracija) koda (Code Motion)
 - Eliminisanje indukcionih promenljivih
 - Direktna redukcija

Pomeranje koda

- Transformacija se sastoji u tome da se iz petlje izbacuju izrazi čija se vrednost ne menja u petlji (ne zavisi od broja prolaza kroz petlju). Takvi izrazi se izbacuju van petlje tako da se izračunavaju samo jednom.

Primer:

```
while (i <= limit - 2)
```

Transformiše se u

```
t = limit - 2
```


```
while (i <= t )
```


Indukcione promenljive

- Indukcione promenljive su promenljive čija promena izaziva promenu drugih promenljivih. Vrednost jedne promenljive direktno zavisi od druge. Ove zavisnosti se mogu iskoristiti za direktnu redukciju koda.

Npr. u bloku B3 imamo $j = j - 1$, a zatim $t4 := 4 * j$. Ova naredba može da bude zamenjena naredbom: $t4 := t4 - 4$

B3 pre transformacije



```
j := j - 1  
t4 := 4 * j  
t5 := a [t4]  
if t5 > v go to B3
```

B3 posle transformacije



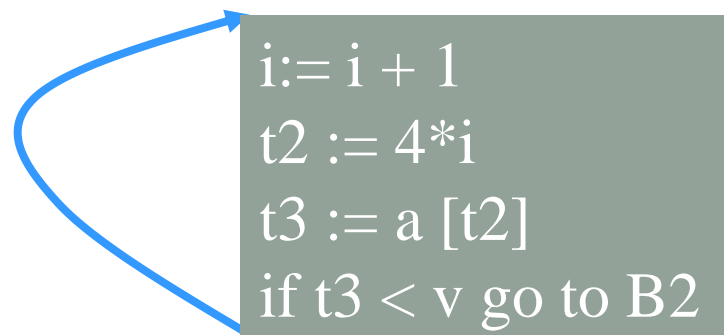
```
t4 := 4 * j
```

```
t4 := t4 - 4  
t5 := a [t4]  
if t5 > v go to B3
```

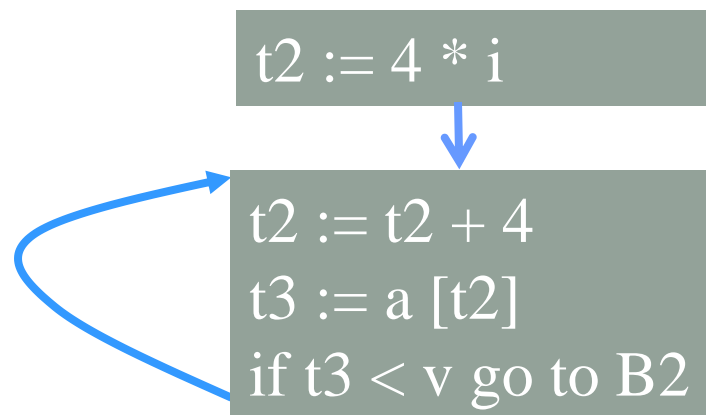
Indukcione promenljive

- Slična redukcija može da se izvršena i bloku B2 gde imamo naredbe $i = i + 1$, a zatim $t2 := 4*i$. Ova naredba može da bude zamenjena naredbom $t2 := t2 + 4$.

B2 pre transformacije



B2 posle transformacije



Direktna redukcija

- Transformacija kojom se složenije (skuplje) naredbe zamenjuju jednostavnijim (jeftinijim) naredbama
- U našem primeru, direktna redukcija je primenjena u sprezi sa eliminisanjem indukcionih promenljivih.
 - U okviru petlji u blokovima B2 i B3, naredbe množenja $t2 := 4 * i$ i $t4 := 4 * j$ zamenjene su jednostavnijim naredbama oduzimanja $t4 := t4 - 4$ i sabiranja $t2 := t2 + 4$.

Konačno rešenje

B1

```
1: i := m - 1  
2: j := n  
3: t1 := 4 * n  
4: v := a [t1]  
5: t2 := 4*i  
6: t4 := 4*j
```

B2

```
7: t2 := t2+4  
8: t3 := a [t2]  
9: if t3 < v go to B2
```

B3

```
10: t4 := t4-4  
11: t5 := a [t4]  
12: if t5 > v go to B3
```

B4

```
13: if i >= j go to B6
```

B5

```
14: a [t2] := t5  
15: a [t4] := t3  
16: go to B2
```

B6

```
17: t14 := a [t1]  
18: a [t2] := t14  
19: a [t1] := 73
```

