

1. Redovarjanje

PREVJALNIK

jezik, ki so podobni kot C / pascal / C (niso OOP, funkcionalni, logični, tolmačeni)

nestapelje podobno

- ne bomo porabili o kerem verzijam

če kerma rečenica = basic C komajce

1 teden = 1 poglavje v "Compiler implementation in Java"
optional "complex principles, techniques and tools"

Keratvoda DN $n - \text{naloge} = \sum_{i=0}^{n-1} i - \text{naloge}$

če naredi rezultato

8 druge na izhovca = vzet je easy

kpt: spremeni svoj compiler tako da -

Vzaj 50% narejenih DN

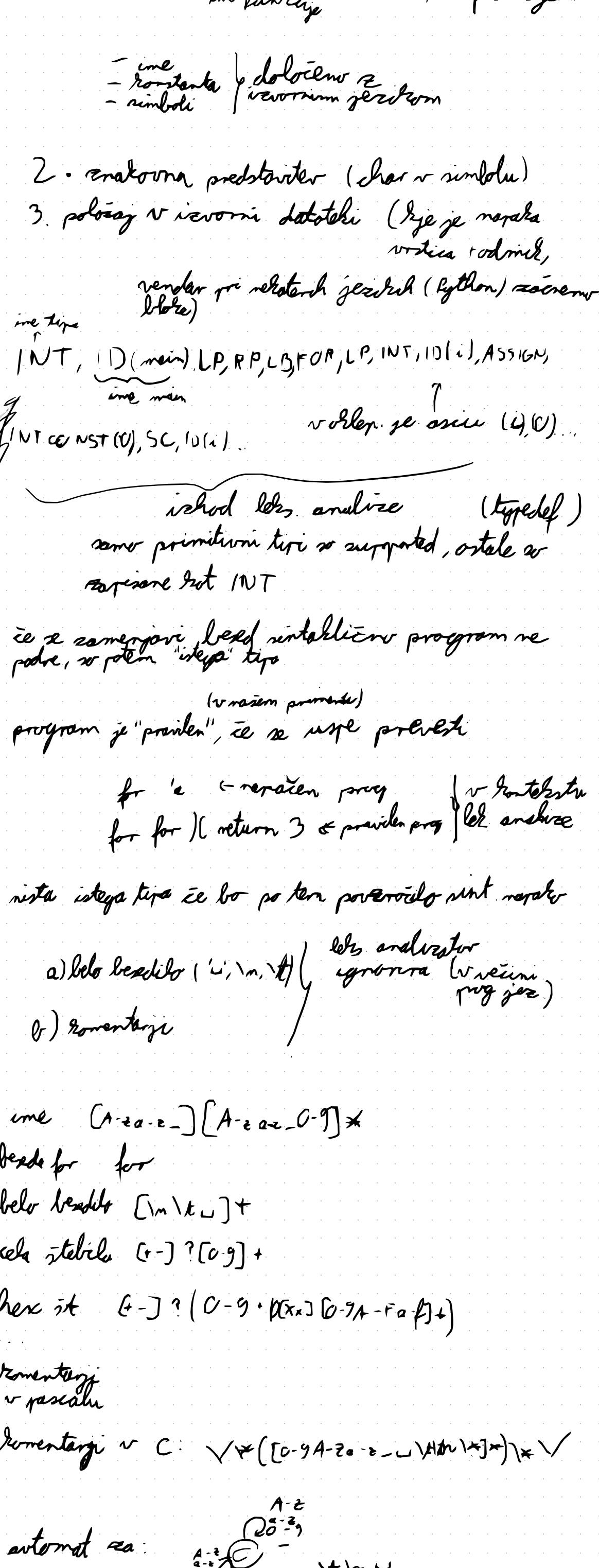
50 vaje + 50 vajet
druge pozitivne

Ne vajah se venci + pogleda

Struktura prevajalnika

struktura je podobna cevovoda

vstop: izvorna koda



1 taza = 1 teden (nazen semantične analize)

od letes. in do gen. vmesne koda = sprednji del (od izvornega jaz. do vmesne koda)

ad vmesne koda do izvira koda = zadnji del

(last prevajalniški člen)

GCC stil

C++ → vmesna koda → ARM → RISC

fortran → vmesna koda → RISC

ada → vmesna koda → RISC

fortran → vmesna koda → RISC

ada → vmes

2. Redavanya

G brez om.
kontekst odr. gr.

T S

LCA

linearna

Sintaksa analiza

vhod

vhod razoredje leks. simbolov

KNG SA
L G LCA
lin. gramatika

vhod: dlevo napravo (sled nejednake)

razoredje producija & napravak

dle vseh napravak

skrajna leva rep. (RM)
skrajna desna rep. (LM)

-> vsejne
skrajna desna vsejne
vsejne & skrajna leva
vsejne simbol

Sintakso jezika opisemo = KNG (in njeni,
je pa "najbolji" formalizem) pravilno
syntax specification

KI JE NE DVOJUMNA

$E \rightarrow (E) \mid M * E \mid C * N \mid N$

$\Gamma \rightarrow A \mid \Gamma + A \mid A + \Gamma$

$A \rightarrow ATN \mid NT_A \mid N$

$N \rightarrow C \cdot g$

algoritmi:

- LL: left to right scan producing
the left most derivation

ale: naredimo levo sled napravak
od zadnjih napred

- LR: L. producing the right most derivation
in reverse order

- PEG: parsing expression grammars

- sintaktični tabulatordi (syntax condensors)

kanonici LL: LAL, SLL, ALL(*), GLL, ...

denoncani LR: CALR, SLR, ..., GLR, ...

LL je veliko izboljši od LR, je pa laže
za implementacijo?

na nekaj bomo učarabili LALR / ALL

ne nem

LL je ima boljše obvezovanje o napravah
 kot LR

LL omogoča "backtracking"

LR in LL sta linearne v časovni kompleksnosti

LL algoritam:

primer: $S \rightarrow AB$

$A \rightarrow aA \mid b$

$B \rightarrow ab \mid ba$

predpostavka: vhod je razenjiz AB

$S \rightarrow AB$ \xrightarrow{aaAB} je razgledano simbol

\xrightarrow{abAB} na producijo

\xrightarrow{baAB} v nasprotno smer gremo

\xrightarrow{bbAB}

\xrightarrow{aB}

\xrightarrow{bB}

\xrightarrow{a}

\xrightarrow{b}

void parseA() {

switch (read()) {

case 'a': parseA(); break;

case 'b': read('b'); break;

default: error(); } }

void parseB() {

switch (read()) {

case 'a': read('a'); break;

case 'b': read('b'); break;

default: error(); } }

redukcija spuščanje

primer: void parseB() {

read('a'); } }

read('b');

read('a');

read('b');

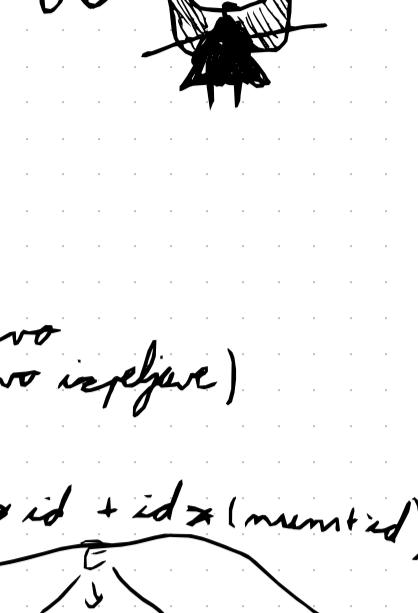
3. Predeavanje

Abstraktna sintaksna drvo

drvo izpeljav

"Zakaj hodite na fiks?" Da riješi so ni druga zobje polomil,
da ni načelo teda mi.

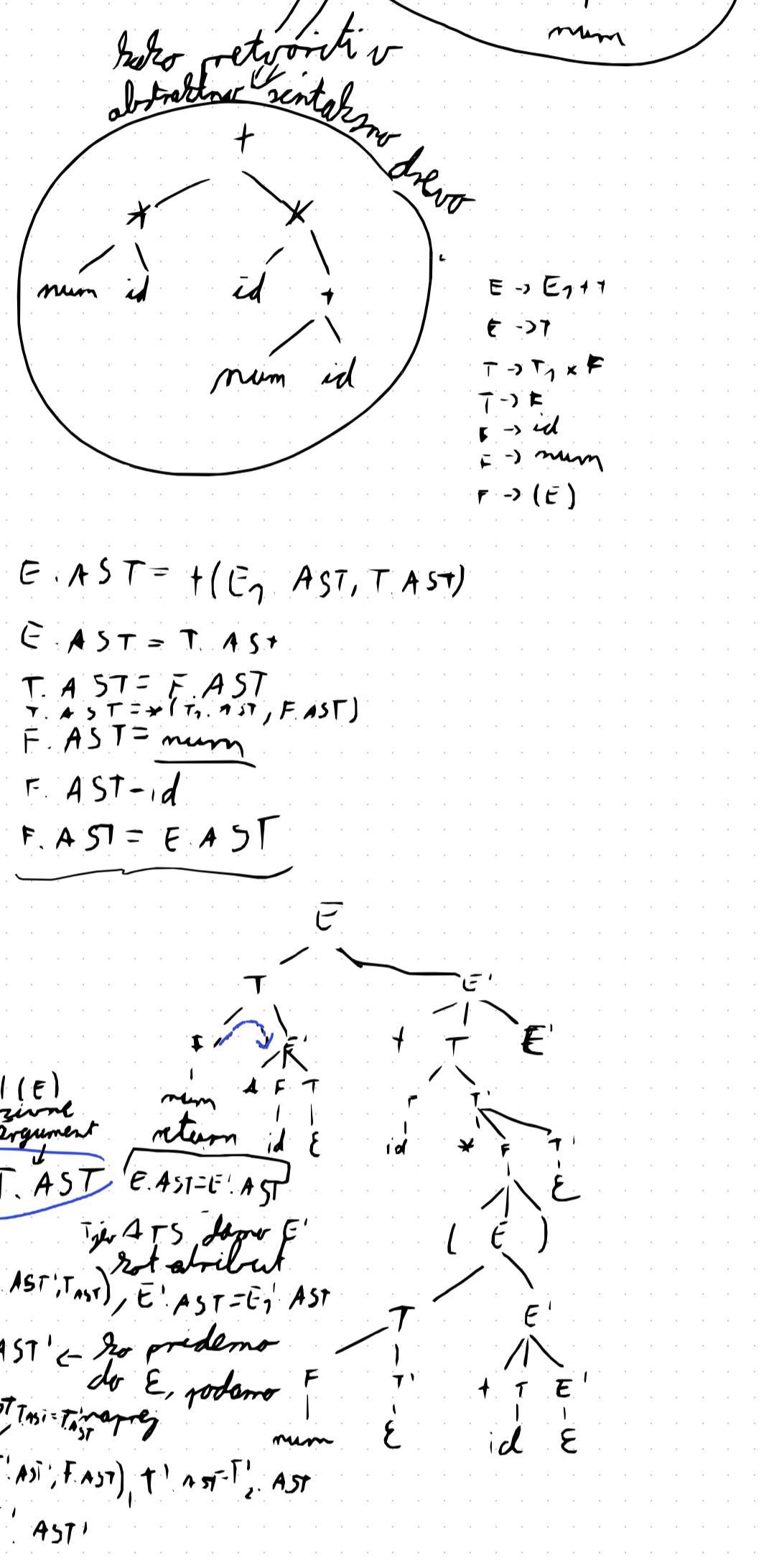
-Slovenj 2025-



ABSTRAKTNA SINTAKSA

VHOD: drvo izpeljav

IZHOD: abstraktno sintaksno drvo
(vsebot, vse generativne posredstva drvo izpeljav)



$$E \cdot AST = + (E_1 \cdot AST, T \cdot AST)$$

$$E \cdot AST = T \cdot AST$$

$$T \cdot AST = F \cdot AST$$

$$T \cdot AST = + (T_1 \cdot AST, F \cdot AST)$$

$$F \cdot AST = num$$

$$F \cdot AST = id$$

$$F \cdot AST = E \cdot AST$$

$$E \rightarrow T E'$$

$$E' \rightarrow T E'$$

$$T \rightarrow F T'$$

$$T' \rightarrow X F T_1 / E$$

$$F \rightarrow id \mid num \mid (E)$$

$$\text{param rekurzivni argument}$$

$$E' \cdot AST = T \cdot AST$$

$$E_1 \cdot AST = + (E_1 \cdot AST; T_1 \cdot AST), E_1 \cdot AST = E_1 \cdot AST$$

$$E_1 \cdot AST = E_1 \cdot AST \leftarrow \text{do predemo}$$

$$T_1 \cdot AST = F \cdot AST, T_1 \cdot AST = T_1 \cdot AST$$

$$T_2 \cdot AST = + (T_1 \cdot AST, F \cdot AST), T_2 \cdot AST = T_2 \cdot AST$$

$$T' \cdot AST = T' \cdot AST$$

$$F \cdot AST = id$$

$$F \cdot AST = num$$

ast parse E()

ast t - parse T()

return parse E();

ast e' - parse E'(d)

return e'

}

ast parse E'(ast t (levi))

switch (t) {

case '+' skip();

ast t Desni = parse T();

return parse E'(+ (levi, t Desni));

case 'E':

return tDesni;

}

}

vrednost atributa na lev stran je određena od vrednosti atributa na desni

zveznica razumevanja

AST je poddržan atribut

A $\rightarrow X_1 X_2 \dots X_m$

$$\text{attr}(A) = f(\text{attr}(X_1), \text{attr}(X_2), \dots, \text{attr}(X_{m-1}))$$

A $\rightarrow X_1 X_2 \dots X_m$

$$\text{attr}(X_n) = f(\text{attr}(A), \text{attr}(X_1), \dots, \text{attr}(X_{n-1}))$$

Sintaktičko usmerjeno prevoanje

od izvirne rabe do abstraktne sintaksne drvece

in nato prevoanje poddržava

ordje atributne gramatike

KNB + atributi + semantika pravila za izvoz atributor

LR logično mnenje od LR, LR nudi logički opis neke red LR

4. Preverjanje

Semanticka analiza

Vvod: abstraktne sintaksne drevo

khod: abstraktov sintaksno drevo + attribute

spet rezerviramo je
odstran od prav. znaka

reg. z dojema

- rezervirajo imen:

- ugotovimo za vsako ime: kje je def / uporabljen
vsak uporabljen imen bo lahko nelye def

- preverjajo tipov: ali so vse kljuci operanjem v

oddrijujoce pravilnem sistemu tipov
oddrijuje redogljive kode (rezervir, zato da zmanj
porude, a jarev redosredz.)

- analiza strožnosti v kodi (strictness razmerj)

nonstrict - je
strict -

Oloje lahko delamo:

- statično (pred izvajanjem) - strožje, hitreje in
- dinamično (med izvajanjem) - bolj splošna pravila,
potem je

Zaradi statične strožje = kot din?

Ker je problem rezervirljivosti

semanticka analiza ignorira it

int x = 1; $x = 1$ | x : int
x = x + 1; $x = x + 1$

problem

- imenski prostor (namespace)
- doseg (scope)



Semicenje

```
int a
int f(int) {
    return a + b;
}
int g() {
    return a + b;
}
```

Razresavanje imen:

simbolna tabela

prestva imena v definicijo

med obhodom AST

- na mestu definicije imena

ustvarim (ime, def) v sim. tabl.

če je ime vedno ^{def} javno naredilo

če je ime vedno ^{def} nova definicija tako

- na mestu uporabe imena

če def. obstaja: usmerimo razredce na def

če def. ne obstaja: javimo napaka

0(1)

→ ins: dodaj imo, def v simtab ali grem napaka

0(1) find: vrne def dana za imena ali jarev napaka

0(1) new_scope: zacine novo podrojno dosega

0(1) old_scope: konča trenutno podrojno dosega

če želimo izbrisati scope, zacнемo pa

a, izbrisemo vse do takih link, potem pa

sebe izbrisie / poppe

5. Predavanje naprij. Semantična analiza

za

Tip je mrežica vrednosti

(čeli sterile, plenjocia sterile, ...)

noben ne pravi, da tip ne more biti sterilen $\times \{T, F\}$

algebrajski tipi (pointer \rightarrow C je - struct/union)

Tip nem pove se datotemu se lahko dela in operatore,

ki jih lahko uporabimo

tipizacija

statična

dinamika

- v času prevojnja

- v času izvajanja

- lahko povrni tip in

- spremenljivke nemajo

spremenljivke

tipa

JAVA

class A { a(); }

class A extends A { a'(); }

A x = new A'();

recuren lahko videti

int a

int x

int y

x = y + 2 * a

INT

7. Predavanje

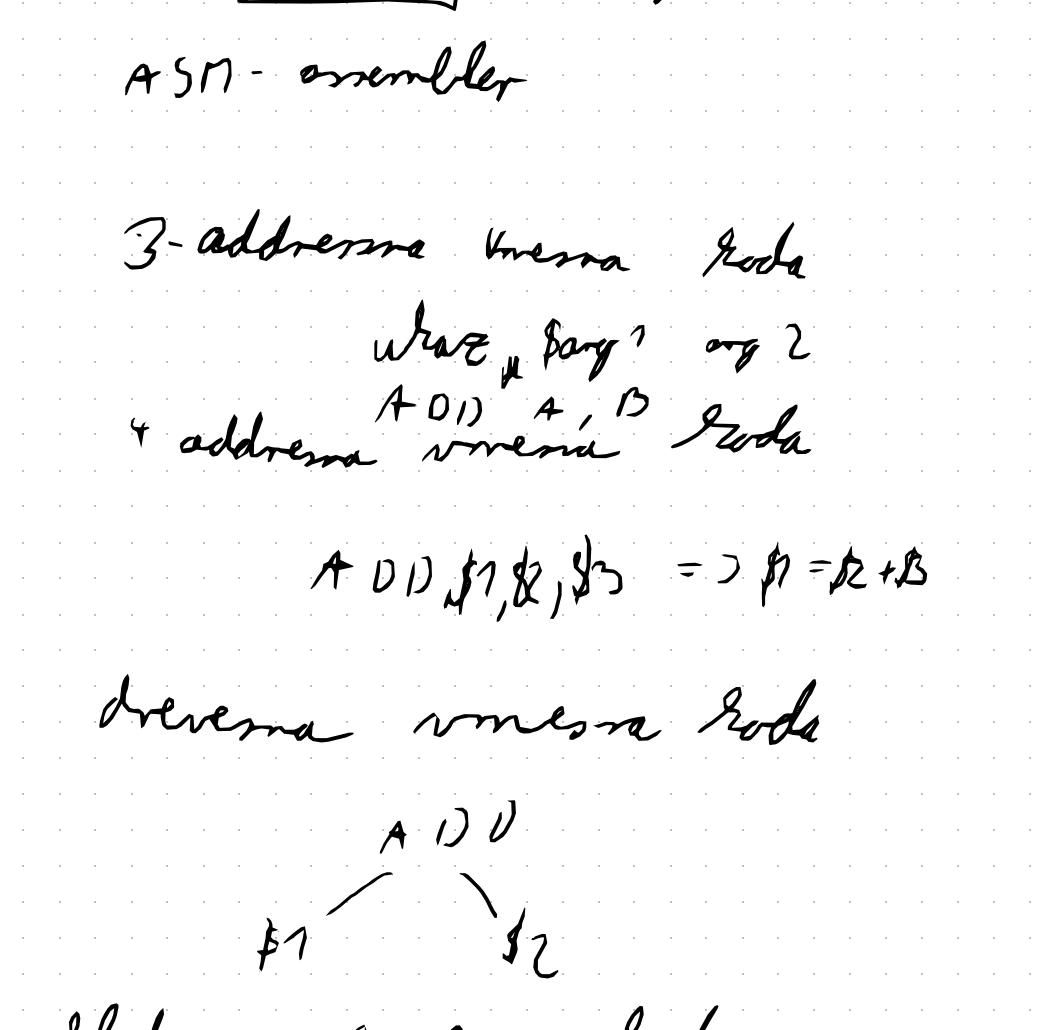
Generirajte vmesne kode

Vhod: A ST + attributi

Izход: vmesna koda za posamezne funkcije

PREVOV

vsečenem razini FUNKCIJE



3-addr resne vmesne kode

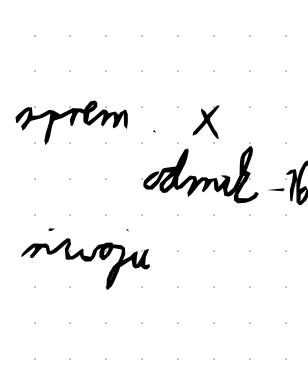
uraz "bag" arg 2

A[0] A[B]

4-addr resne vmesne kode

$$A[0], \$1, \$2, \$3 \Rightarrow \$1 = \$2 + \$3$$

drevna resna vmesna koda

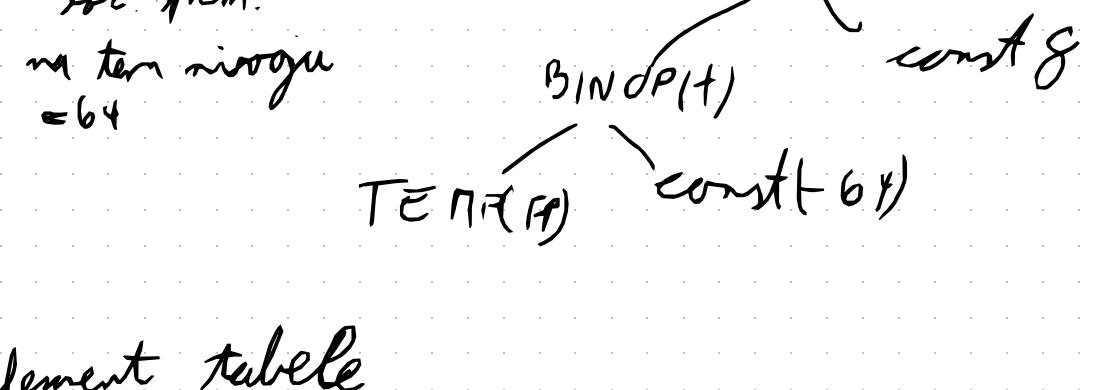


sladovna resna vmesna koda
java byte code

enostavni prevozniški



dejavniki



torek gledamo

za zante

varaz

const 12 => const(12)

CINOP

BINOP

CALL

NAME

MEM

12 * 3 + 4 => BINOP

BINOP *

const 4

const(12) const(3)

NAME

NAME (-x)

NAME

NAME (-y)

NAME

NAME (-z)

NAME

NAME (-w)

NAME

NAME (-v)

NAME

NAME (-u)

NAME

NAME (-t)

NAME

NAME (-s)

NAME

NAME (-r)

NAME

NAME (-q)

NAME

NAME (-p)

NAME

NAME (-o)

NAME

NAME (-n)

NAME

NAME (-m)

NAME

NAME (-l)

NAME

NAME (-k)

NAME

NAME (-j)

NAME

NAME (-i)

NAME

NAME (-h)

NAME

NAME (-g)

NAME

NAME (-f)

NAME

NAME (-e)

NAME

NAME (-d)

NAME

NAME (-c)

NAME

NAME (-b)

NAME

NAME (-a)

NAME

NAME (-z)

NAME

NAME (-y)

NAME

NAME (-x)

NAME

NAME (-w)

NAME

NAME (-v)

NAME

NAME (-u)

NAME

NAME (-t)

NAME

NAME (-s)

NAME

NAME (-r)

NAME

NAME (-o)

NAME

NAME (-n)

NAME

NAME (-m)

NAME

NAME (-l)

NAME

NAME (-k)

NAME

NAME (-j)

NAME

NAME (-i)

NAME

NAME (-h)

NAME

NAME (-g)

NAME

NAME (-f)

NAME

NAME (-e)

NAME

NAME (-d)

NAME

NAME (-c)

NAME

NAME (-b)

NAME

NAME (-a)

NAME

NAME (-z)

NAME

NAME (-y)

NAME

NAME (-x)

NAME

NAME (-w)

NAME

NAME (-v)

NAME

NAME (-u)

NAME

NAME (-t)

NAME

NAME (-s)

NAME

NAME (-r)

NAME

NAME (-o)

NAME

NAME (-n)

NAME

NAME (-m)

NAME

NAME (-l)

NAME

NAME (-k)

NAME

NAME (-j)

NAME

NAME (-i)

NAME

NAME (-h)

NAME

NAME (-g)

NAME

NAME (-f)

NAME

NAME (-e)

NAME

NAME (-d)

NAME

NAME (-c)

NAME

NAME (-b)

NAME

NAME (-a)

NAME

NAME (-z)

NAME

NAME (-y)

NAME

NAME (-x)

NAME

NAME (-w)

NAME

NAME (-v)

NAME

NAME (-u)

NAME

NAME (-t)

NAME

NAME (-s)

NAME

NAME (-r)

NAME

<

8. Redavanya
Linearisatje V.
1 1 1 1 1 1 1 1

gnezdem tří funkcií

$$f_1(f_2(f_3(b)))$$

1. SL za f_1 na sladce }
2. SL za f_2 na sladce }

• what SExpr

• what CJump

problem: 2 různov. řeš. podána, ale už třetí

city :
more
/ \ ah Es

CALL
a non-void
function

- Ull je odvoden od procesorja, nato ga
imenujemo zatem obveznosti.

заподиши по д

- starve
- ignore

ENV
 P T3

MOVE

TEMP17 → CALL F3

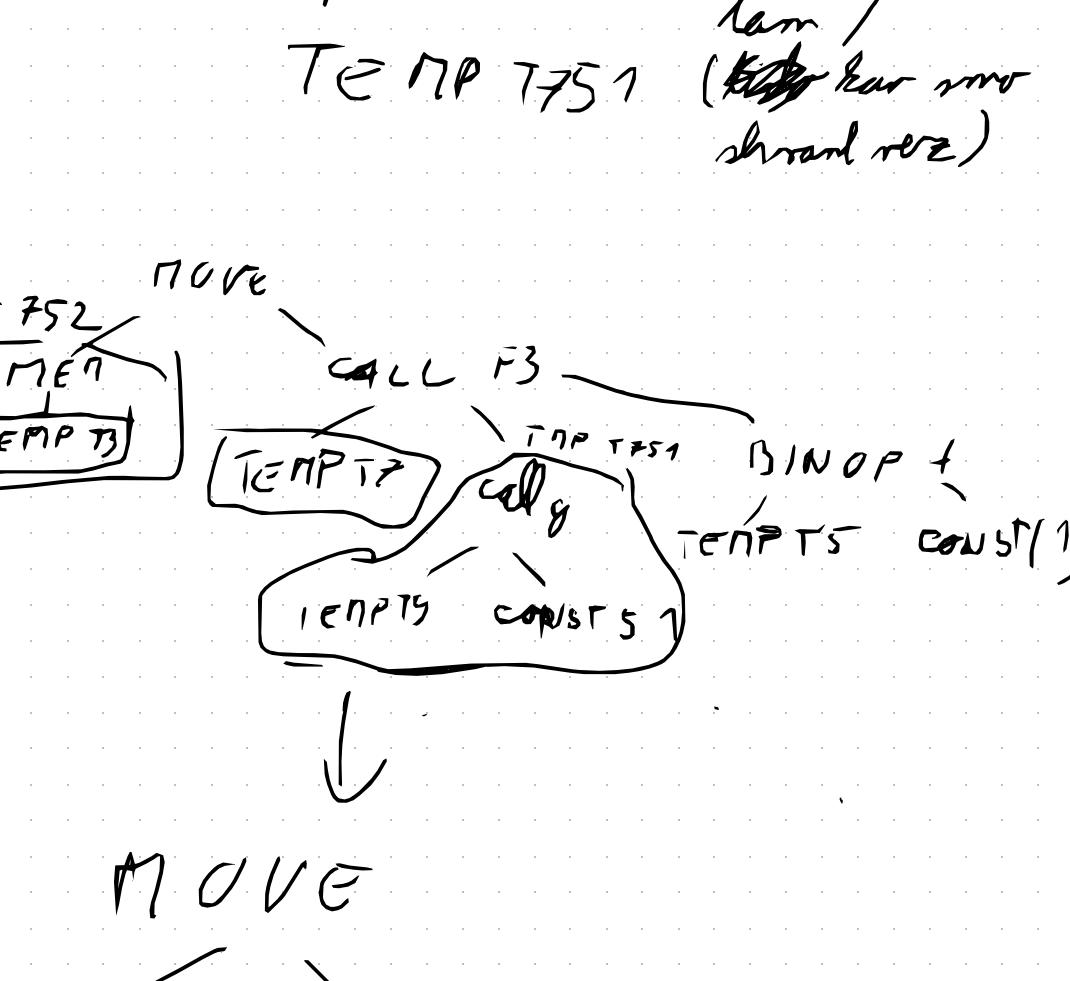
TEMP17 → call g

TEMP5 → BINOP +

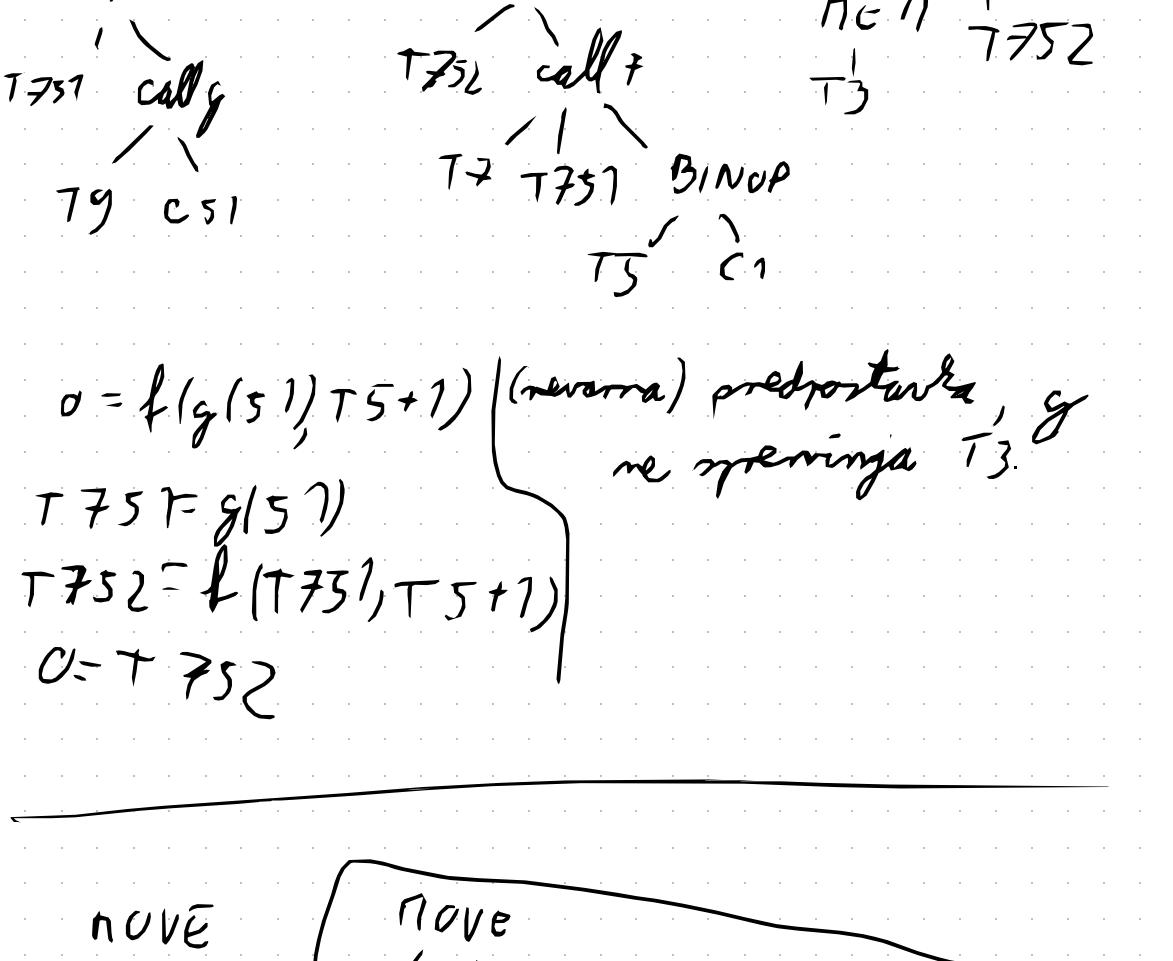
TEMP5 → CONST17

TEMP17 → CONST17

LOVE
call

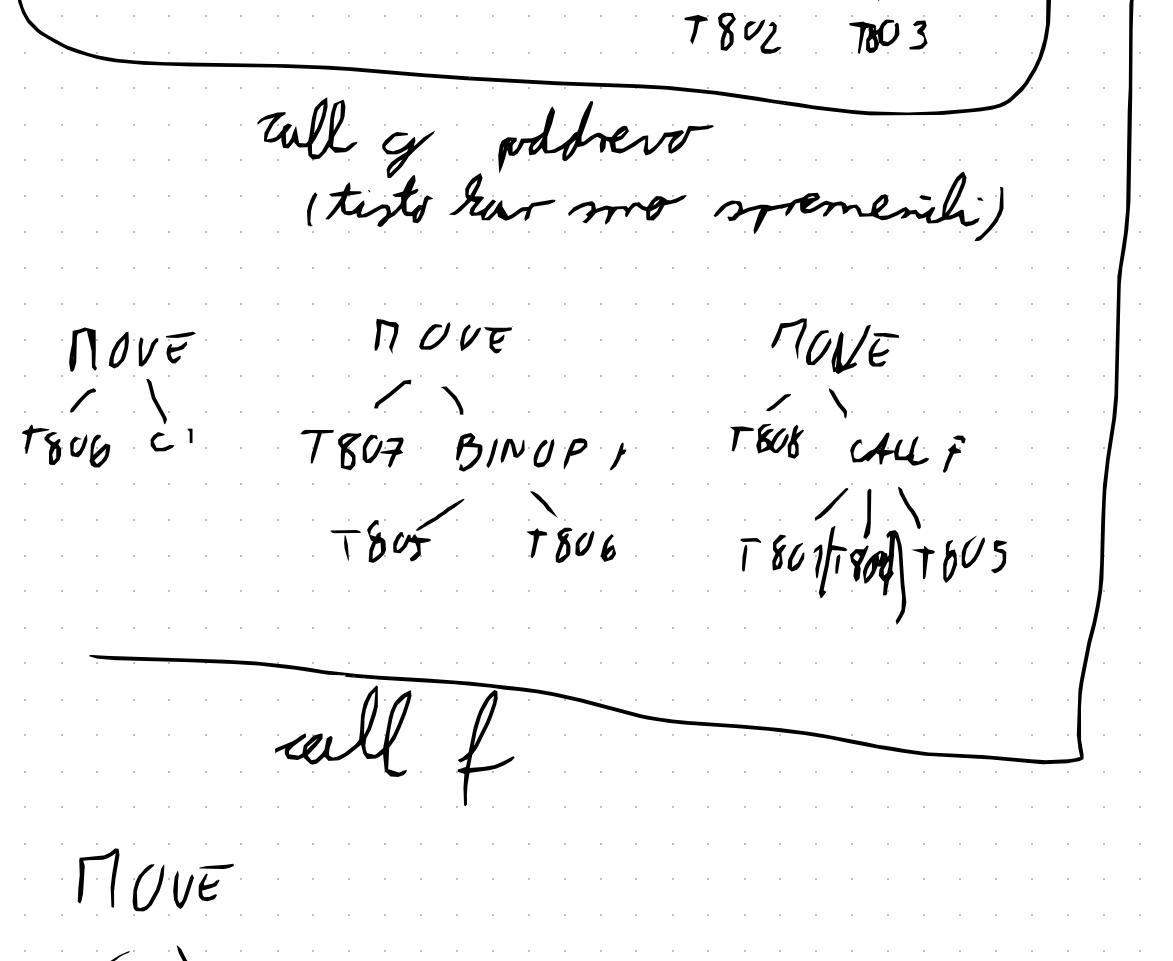


A graph on a grid background showing two bell-shaped curves. The top curve is labeled "T 3" and the bottom curve is labeled "Move". The "Move" curve has a sharp vertical drop at its right end.

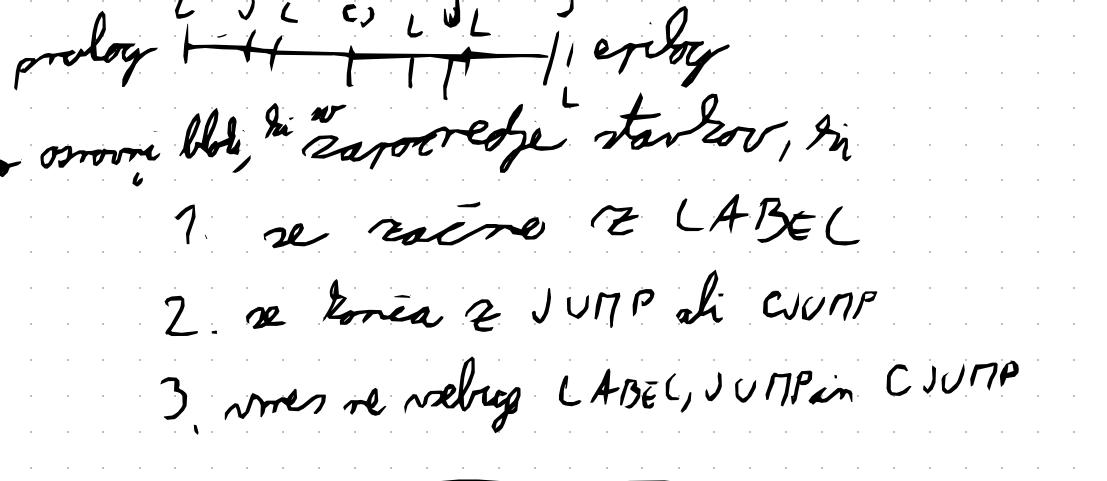


MOVE
 MOVE
 MOLE

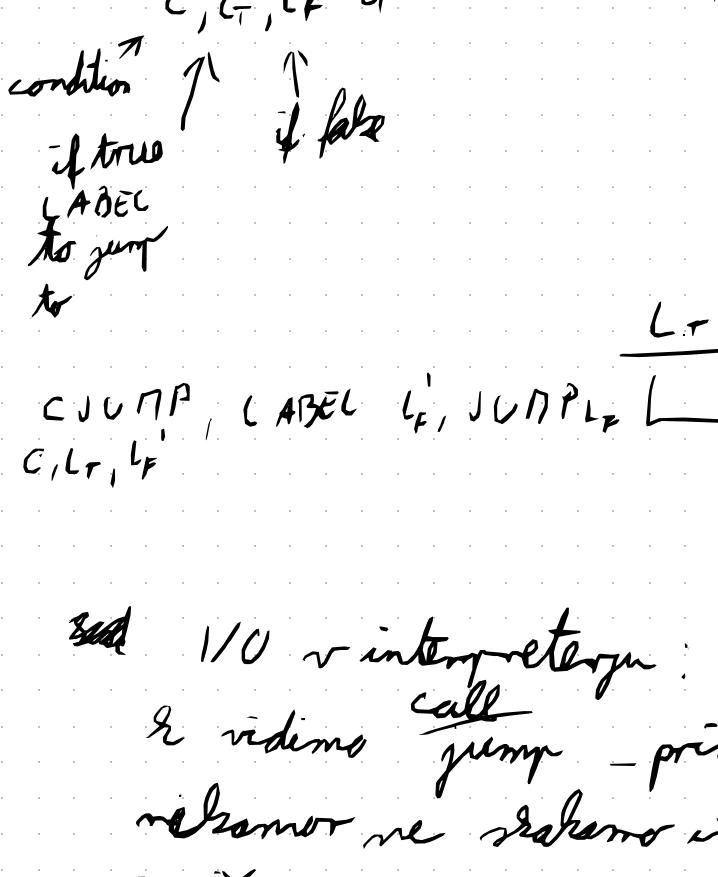
T802 T9 T803 CS7 T804 calling



$\pi \in \Pi$
T808
T800
primer za CJUMP C, label L, label E



9. Prevaranje



zad 1/0 v interpretatorju:

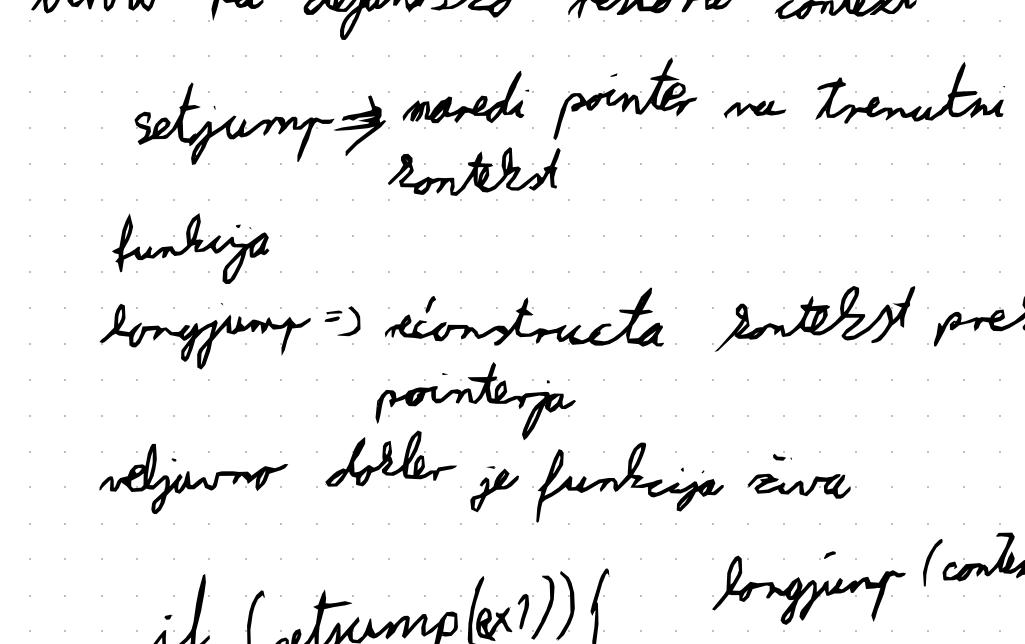
če vidimo call - print,

nesamov ne zakrovim to direktno
izpisemo

Izjeme:

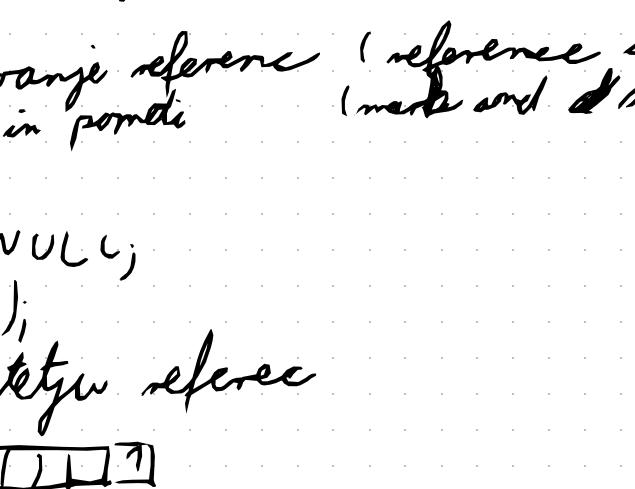
C setjmp - try - catch
longjmp - throw

related



doseemo 1 slican razvod
a time in dedenovu, če v try
funkciji lahko hanlamo

do zgodil
throw x trenutki = P in SP poleg se od tiste,
ki lahko hanlamo ta throw



v f2

throw EX1

pri try - catch se hrani PC counter,
ki pride do catch(..) stavlja

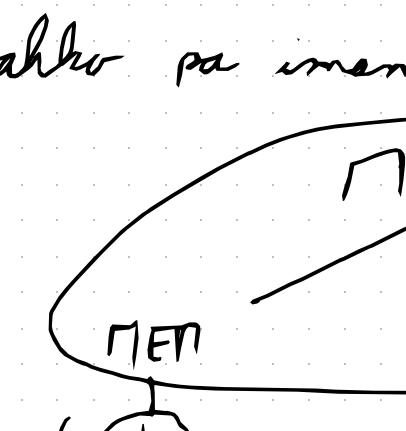
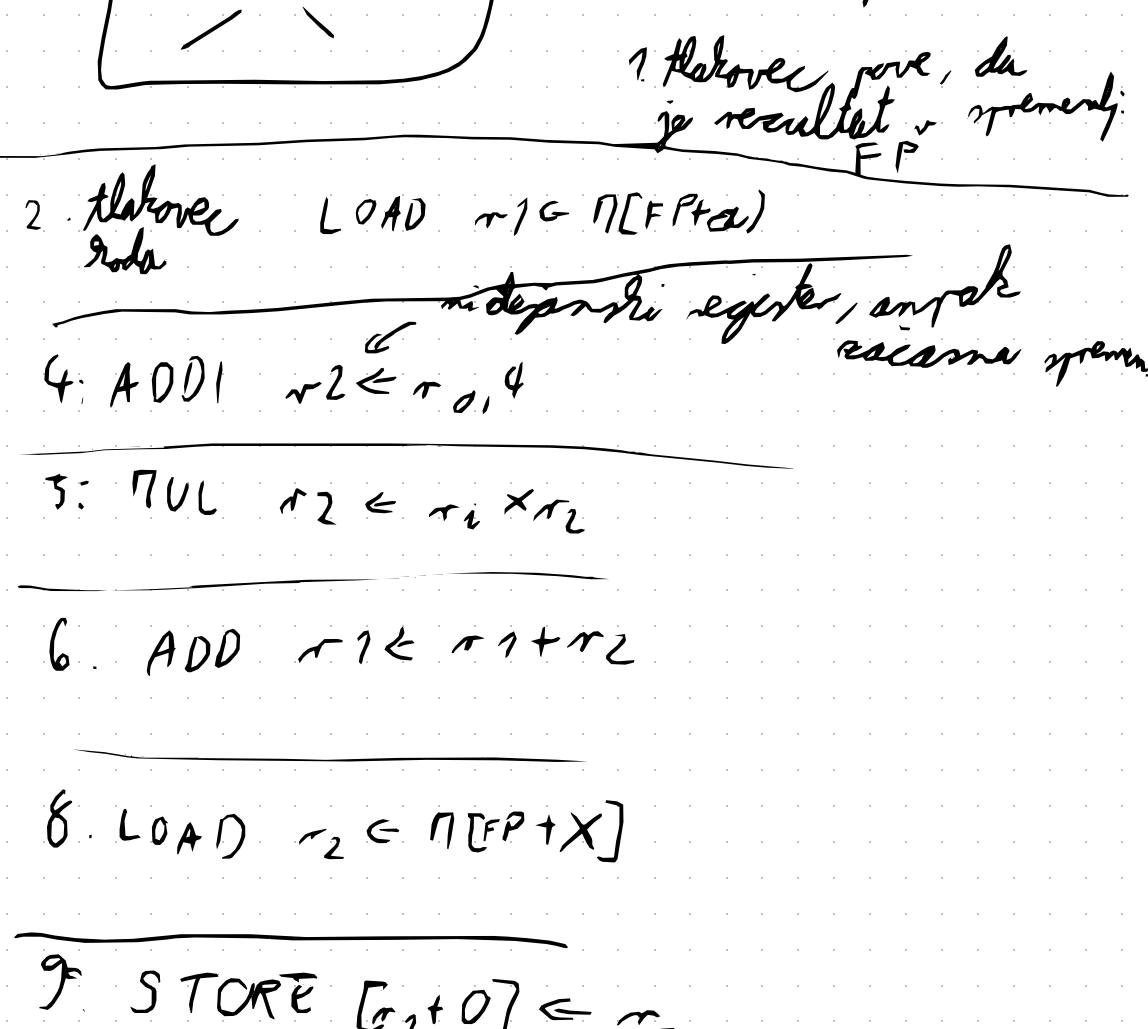
10. Predavanje

Generiranje strojnih urazov

VHOD: (za posamezne funkcije) razredjevanje končnih drevov (za C JUMP je fata labila, ne ker im delu MOVE je TEPP)

vihod: (za pos funkc.) razredjevanje strojnih urazov in zato spremen

primer iz knjige



? blatorec vose, da je rezultat v segmentu FP

2. blatorec LOAD r1 < r1[FPT+a]

3. ADDI r2 < r1+a, r2

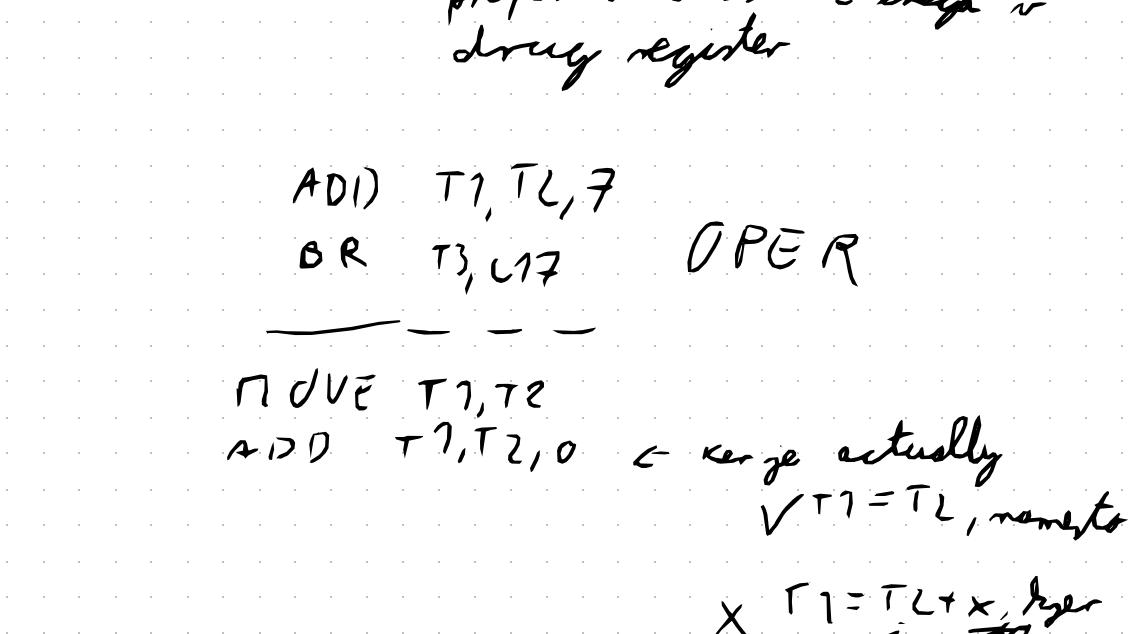
4. MUL r2 < r1 * r2

5. ADD r1 < r1+r2

6. ADDI r2 < FP+X

7. MOVE r1 < r1[r2]

8. STORE [r1+0] < r2



2. LOAD r1 < r1[FPT+a]

4. ADDI r2 < r1+a

5. MUL r2 < r1 * r2

6. ADD r1 < r1+r2

7. ADDI r2 < FP+X

8. MOVE r1 < r1[r2]

9. STORE [r1+0] < r2

10. MOVE r1 < r1[r2]

11. STORE [r1+0] < r2

12. MOVE r1 < r1[r2]

13. STORE [r1+0] < r2

14. MOVE r1 < r1[r2]

15. STORE [r1+0] < r2

16. MOVE r1 < r1[r2]

17. STORE [r1+0] < r2

18. MOVE r1 < r1[r2]

19. STORE [r1+0] < r2

20. MOVE r1 < r1[r2]

21. STORE [r1+0] < r2

22. MOVE r1 < r1[r2]

23. STORE [r1+0] < r2

24. MOVE r1 < r1[r2]

25. STORE [r1+0] < r2

26. MOVE r1 < r1[r2]

27. STORE [r1+0] < r2

28. MOVE r1 < r1[r2]

29. STORE [r1+0] < r2

30. MOVE r1 < r1[r2]

31. STORE [r1+0] < r2

32. MOVE r1 < r1[r2]

33. STORE [r1+0] < r2

34. MOVE r1 < r1[r2]

35. STORE [r1+0] < r2

36. MOVE r1 < r1[r2]

37. STORE [r1+0] < r2

38. MOVE r1 < r1[r2]

39. STORE [r1+0] < r2

40. MOVE r1 < r1[r2]

41. STORE [r1+0] < r2

42. MOVE r1 < r1[r2]

43. STORE [r1+0] < r2

44. MOVE r1 < r1[r2]

45. STORE [r1+0] < r2

46. MOVE r1 < r1[r2]

47. STORE [r1+0] < r2

48. MOVE r1 < r1[r2]

49. STORE [r1+0] < r2

50. MOVE r1 < r1[r2]

51. STORE [r1+0] < r2

52. MOVE r1 < r1[r2]

53. STORE [r1+0] < r2

54. MOVE r1 < r1[r2]

55. STORE [r1+0] < r2

56. MOVE r1 < r1[r2]

57. STORE [r1+0] < r2

58. MOVE r1 < r1[r2]

59. STORE [r1+0] < r2

60. MOVE r1 < r1[r2]

61. STORE [r1+0] < r2

62. MOVE r1 < r1[r2]

63. STORE [r1+0] < r2

64. MOVE r1 < r1[r2]

65. STORE [r1+0] < r2

66. MOVE r1 < r1[r2]

67. STORE [r1+0] < r2

68. MOVE r1 < r1[r2]

69. STORE [r1+0] < r2

70. MOVE r1 < r1[r2]

71. STORE [r1+0] < r2

72. MOVE r1 < r1[r2]

73. STORE [r1+0] < r2

74. MOVE r1 < r1[r2]

75. STORE [r1+0] < r2

76. MOVE r1 < r1[r2]

77. STORE [r1+0] < r2

78. MOVE r1 < r1[r2]

79. STORE [r1+0] < r2

80. MOVE r1 < r1[r2]

81. STORE [r1+0] < r2

82. MOVE r1 < r1[r2]

83. STORE [r1+0] < r2

84. MOVE r1 < r1[r2]

85. STORE [r1+0] < r2

86. MOVE r1 < r1[r2]

87. STORE [r1+0] < r2

88. MOVE r1 < r1[r2]

89. STORE [r1+0] < r2

90. MOVE r1 < r1[r2]

91. STORE [r1+0] < r2

92. MOVE r1 < r1[r2]

93. STORE [r1+0] < r2

94. MOVE r1 < r1[r2]

95. STORE [r1+0] < r2

96. MOVE r1 < r1[r2]

97. STORE [r1+0] < r2

98. MOVE r1 < r1[r2]

99. STORE [r1+0] < r2

100. MOVE r1 < r1[r2]

101. STORE [r1+0] < r2

102. MOVE r1 < r1[r2]

103. STORE [r1+0] < r2

104. MOVE r1 < r1[r2]

105. STORE [r1+0] < r2

106. MOVE r1 < r1[r2]

107. STORE [r1+0] < r2

108. MOVE r1 < r1[r2]

109. STORE [r1+0] < r2

110. MOVE r1 < r1[r2]

111. STORE [r1+0] < r2

112. MOVE r1 < r1[r2]

113. STORE [r1+0] < r2

114. MOVE r1 < r1[r2]

115. STORE [r1+0] < r2

116. MOVE r1 < r1[r2]

117. STORE [r1+0] < r2

118. MOVE r1 < r1[r2]

119. STORE [r1+0] < r2

120. MOVE r1 < r1[r2]

121. STORE [r1+0] < r2

122. MOVE r1 < r1[r2]

123. STORE [r1+0] < r2

124. MOVE r1 < r1[r2]

125. STORE [r1+0] < r2

126. MOVE r1 < r1[r2]

127. STORE [r1+0] < r2

128. MOVE r1 < r1[r2]

129. STORE [r1+0] < r2

130. MOVE r1 < r1[r2]

131. STORE [r1+0] < r2

132. MOVE r1 < r1[r2]

133. STORE [r1+0] < r2

134. MOVE r1 < r1[r2]

135. STORE [r1+0] < r2

136. MOVE r1 < r1[r2]

137. STORE [r1+0] < r2

138. MOVE r1 < r1[r2]

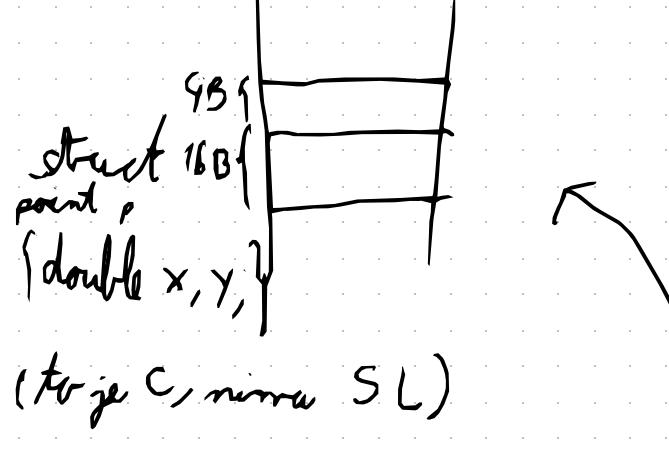
139. STORE [r1+0] < r2

11. Predavanje

calling convention

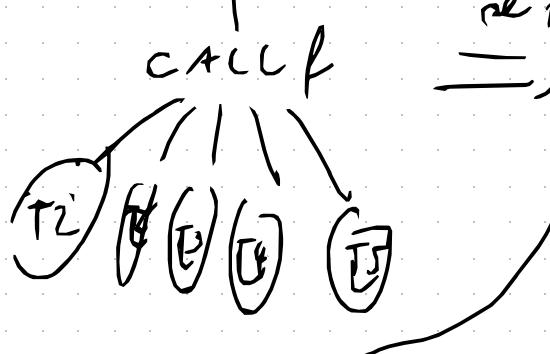
primer: ~~one~~ 3 argy gogo se v reg,
ostali preto slade

int f (int a, int b, struct point p, int c, int d)
 R R S
 \$0 \$1 \$2 +\$0
 +C S +\$2 +\$0 }



če ga voljeliči f, je to delom ravn

g



se poslita v zapovede
=) stojík ukazov

code(T1) = res T1

code(T2) = res T2

code(T3) = res T3

code(T4) = res T4

code(T5) = res T5

LD T6,[T3+0]

STO T6,[SP,C]

LD T6,[T3+8]

STO T6[T3+8]

pushed

dodamo
rodo, da slavni

ay

NEM

STO T5,[SP+16]

T0 call f

LD \$0,T1

LD \$1,T2

LD \$2,T4

a,f,c so
reg v reg

OBNOVIMO C4U f
STO \$0,\$0
resultat = \$0, če lahko, drugače
po sladku

12 Predevaranje

Activnost spremenljivk

vhod: razredje stognih vektorov z
zacetnimi spremenljivkami

izhod: -1^1 in interferencijski graf

c: stog

a: 0

b: 0

c: 0

d: 0

e: 0

f: 0

g: 0

h: 0

i: 0

j: 0

k: 0

l: 0

m: 0

n: 0

o: 0

p: 0

q: 0

r: 0

s: 0

t: 0

u: 0

v: 0

w: 0

x: 0

y: 0

z: 0

aa: 0

bb: 0

cc: 0

dd: 0

ee: 0

ff: 0

gg: 0

hh: 0

ii: 0

jj: 0

kk: 0

ll: 0

mm: 0

nn: 0

oo: 0

pp: 0

qq: 0

rr: 0

ss: 0

tt: 0

uu: 0

vv: 0

ww: 0

xx: 0

yy: 0

zz: 0

aa: 0

bb: 0

cc: 0

dd: 0

ee: 0

ff: 0

gg: 0

hh: 0

ii: 0

jj: 0

kk: 0

ll: 0

mm: 0

nn: 0

oo: 0

pp: 0

qq: 0

rr: 0

ss: 0

tt: 0

uu: 0

vv: 0

ww: 0

xx: 0

yy: 0

zz: 0

aa: 0

bb: 0

cc: 0

dd: 0

ee: 0

ff: 0

gg: 0

hh: 0

ii: 0

jj: 0

kk: 0

ll: 0

mm: 0

nn: 0

oo: 0

pp: 0

qq: 0

rr: 0

ss: 0

tt: 0

uu: 0

vv: 0

ww: 0

xx: 0

yy: 0

zz: 0

aa: 0

bb: 0

cc: 0

dd: 0

ee: 0

ff: 0

gg: 0

hh: 0

ii: 0

jj: 0

kk: 0

ll: 0

mm: 0

nn: 0

oo: 0

pp: 0

qq: 0

rr: 0

ss: 0

tt: 0

uu: 0

vv: 0

ww: 0

xx: 0

yy: 0

zz: 0

aa: 0

bb: 0

cc: 0

dd: 0

ee: 0

ff: 0

gg: 0

hh: 0

ii: 0

jj: 0

kk: 0

ll: 0

mm: 0

nn: 0

oo: 0

pp: 0

qq: 0

rr: 0

ss: 0

tt: 0

uu: 0

vv: 0

ww: 0

xx: 0

yy: 0

zz: 0

aa: 0

bb: 0

cc: 0

dd: 0

ee: 0

ff: 0

gg: 0

hh: 0

ii: 0

jj: 0

kk: 0

ll: 0

mm: 0

nn: 0

oo: 0

pp: 0

qq: 0

rr: 0

ss: 0

tt: 0

uu: 0

vv: 0

ww: 0

xx: 0

yy: 0

zz: 0

aa: 0

bb: 0

cc: 0

dd: 0

ee: 0

ff: 0

gg: 0

hh: 0

ii: 0

jj: 0

kk: 0

ll: 0

mm: 0

nn: 0

oo: 0

pp: 0

qq: 0

rr: 0

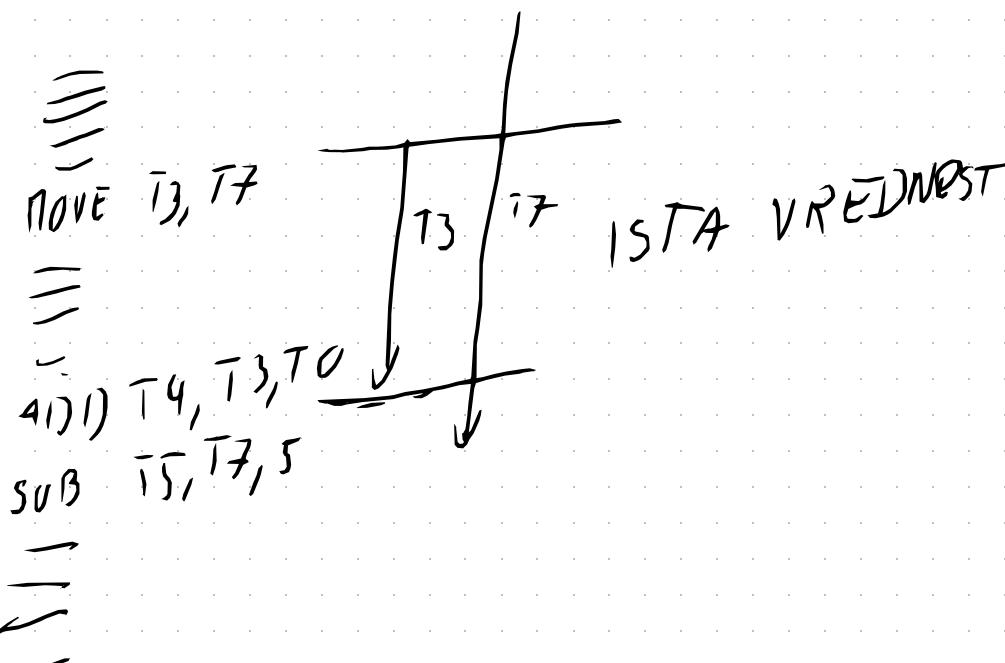
ss: 0

tt: 0

73 Predavanje

bez mo ddali?
bez spraviti calling convention
v registre

~~vezak~~ move je funkcij



če imamo 2 vrstic, tater sestevki se menjat stevilo barv (K), ju lahko denuj razen

ZAKAJ? zdrževanje teh je $O(n^2)$, ker je n ist vrstic, proti barvom je $O(c^n)$

ZAKAJ NE? graf postane bolj gost, ne predstavlja dovolj na performancem

14. Predavanje

FP + SP register se bavljamo u ovom "

Napisati ecdog in prolog

predavanje "putting it all together"

1. podatkovni deli - staticne spremnje, adrese
2. programski deli - konstante, logika, struktura

teretno je rezervirano za razne funkcije

vrata funkcija ima:

1. električni zapisi < TETN Frame } = spremanje
2. rezultat vrata jedinica & v zbirnik

v zbirniku

funkcija

ima predavajuća
prolog

entry
dodatak od
vezanja
gratov

izlaz

exit

ecdog

imo predavajuća

prolog | koj se naredi

- shranje registre

- ustvari električni zapisi } rečnik je samo
- vrati na l entry je odgovarajuće sklopljenje

- restavruje registre

- podstavlja električni zapisi

- prepravlja rezultat

- vrati u R C U R N (jump)

malor dodati

void grant(int m, int n){

int i, j, r, x;

if (n < m) return;

i = m - 1; j = n; r = a[m];

while (1) {

do {
i = i + 1;
while (a[i] < j)}

do { j = j - 1; while (a[j] > r);

if (i >= j) break;

x = a[i]; a[i] = a[j]; a[j] = x;

}

x = a[i]; a[i] = a[m]; a[m] = a[x]

grant(m, j); grant(i, n);

z operacijom (\geq) generira se

izlazni del

vezujući

strength reduction

B1

$i = m - 1$

$j = n$

$t1 = 4 \times n$

$r = a[t1]$

$t2 = a$

$t4 = t1$

B2

$i = i + 1$

$t2 = 4 \times i + t2$

$t3 = a[t2]$

$t4 < r$

goto B2

B3

$j = j - 1$

$t4 = j$

$t5 = a[t4]$

$t3 > r$

goto B3

B4

$t2 = t4$

if $i >= j$ goto B6

B5

$t6 = 4 \times i$

$x = a[t6]$

$t7 = 4 \times i$

$t8 = 4 \times j + t1$

$t9 = a[t8]$

$t10 = t9$

$t11 = 4 \times j$

$a[t11] = t10$

goto B2

B6

$t11 = 4 \times i$

$t12 = 4 \times m$

$t13 = a[t12]$

$t14 = t13$

$a[t14] = t11$

$t15 = 4 \times m$

$a[t15] = t13$

$t16 = t15$

$t17 = t16$

$t18 = t17$

ne zmeri fajn je razmisliti spremnjivanje

ekvivalentne varijable u vremenskoj 2 množenju

glede na rezultat isti podizvareti rezultat

implementacija je ne spremnjiva

(i, t2), (j, t4)

induction variables - spremnjivanje je u

spremnjivanje potpisuje se

varijabli

copy propagation

locates variables - de spremnji se kroz

spremnjivanje

(i, t2), (j, t4)

