**Optimizacija koda**

## Nakon generisanja prve verzije koda uvek vrši optimizacija koda čiji je cilj da se smanji broj naredbi i memorijski prostor koji generisani kod zahteva, i da se dobije što efikasniji kod..

## Pored ove osnovne optimizacije, koja je praktično obavezna, obično se primenjuje i čitav niz dodatnih optimizacija (neobaveznih).

## Osnovno pravilo kod svake optimizacije koda je da ne sme da se naruši ispravnost koda.

## Osnovna optimizacija koda (Piphole optimizacija)

Piphole (kod se posmatra kroz uski prozor)

Vrši se već na nivou međukoda i najvećim delom je mašinski nezavisna. Više puta se prolazi kroz kod i nastoji da se primene jednostavne transformacije za optimizaciju. U jednom trenutku posmatra se samo mali deo koda i nastoji da se izmeni tako da se smanji broj naredbi, složenije naredbe zamene jednostavnijim ili utiče na memorijski prostor koji kod zahteva.

Transformacije koda koje se vrše u okviru Piphol optimizacije se mogu svrstati u sledećih nekoliko grupa:

1. Eliminisanje redundantnih naredbi.

2. Optimizacija toka programa

1. Algebarska uprošćenja
2. Korišćenje mašinskih idioma

**1.1 Eliminisanje redundantnih naredbi**

Ovom grupom obuhvaćene su transformacije kojima se iz koda izbacuju suvišne naredbe ili grupe naredbi. Posledica je samog automatizma koji se primenjuje kod generisanja koda

eliminisu se ne eliminisu se

ST T1 ST T1

LD T1 Lab1: LD T1

**1.2 Optimizacija toka**

U toku generisanja međukoda često se javljaju skokovi na skokove, skokovi na uslovne skokove ili uslovni skokovi na bezuslovne skokove. (posledica ugnjezdavanja petlji i if naredbi – sve kombinacije) U mnogim takvim slučajevima moguće su redukcije.

bezuslovni na bezuslovni zamena

go to L1 go to L2

L1: go to L2 L1: go to L2

Uslovni skok na bezuslovni

if a< b go to L1

L1: go to L2

Bezuslovni skok na uslovni:

glupo

**Nedosegljiv kod**

Eliminišu se naredbe koje se praktično u određenom kontekstu neće nikada izvršiti

# define debug 0

if (debug) {Stampanje poruke o greski }

if debug / = 1 go to L2

print(... //eliminise se

L2:

**1.3 Algebarska uprošćenja**

Naredbe koje se mogu zameniti jednostavnijim naredbama ili potpuno izbaciti

X = X + 0 X = Y + 0

X = X \* 1 X = Y \* 1

**Direktna redukcija**

Složene operacije zamenjuju jednostavnijima koje se brže izvršavaju

* Stepenovanje se zamenjuje višestrukim množenjem X2 = X\*X
* Množenje i deljenje stepenom dvojke se zamenjuje šift operacijma.
* Deljenje realnom konstantnom može se zamenjuje množenjem sa konstantom čija je vrednost jednaka recipročnoj vrednosti te konstante, što je nekad brže.

**1.4 Korišćenje mašinskih idioma**

Osnovna optimizacija može da uključije i neke mašinski zavisne transformacije, kojima se koriste prednosti i specifičnosti asemblerskog jezika mašine za koju se generiše kod

* samoindeksiranja,
* samoinkrementiranja,
* specifičnih adresiranja,
* specifičnih registara procesora i
* korišćenje steka.

## Dodatna optimizacija

Niz dodatnih transformacija koje se izvršavaju na nivou međukoda ili na nivou samog asemblerskog koda. (primer je HUGE)

**2.1 Zajednički podizrazi na nivou blokova**

Eliminisanje suvišnih temporarnih promenljivih u bloku. Nastoji se da se identifikuju promenljive koje u suštini imaju istu vrednost i da se umesto uvođenja nove promenljive koristi stara.

blok

t6 := 4 \* i

x := a [t6]

t7 := 4 \* i //eliminiše se

* 1. **Zajednički podizrazi na globalnom nivou**

Zajednički podizrazi mogu da se pronalaze i na globalnom nivou tako što se posmatraju zavisnosti između blokova i vrednosti uvedene u jednom bloku koriste u narednim

**2.3 Prostiranje copy naredbi**

f:=g

Efekat ove naredbe se može postići tako što se svuda u kodu gde se pojavljuje f ubacuje g. Prilikom optimizacije koda prvo se iyvršava ta transformacija, a nakon nje je moguće izbaciti copy naredbu kao redundantnu.

**2.4 Eliminisanje neaktivnog koda**

Neaktivne naredbe su naredbe koje se nikada neće izvršiti ili nemaju nikakav efekat na program.

Ovakve naredbe i promenljive često nastaju kao posledica prostiranja copy naredbi

**2.5 Optimizacija petlji**

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.

Na primer sledeće tehnike se mogu lako primeniti:

1. Pomeranje koda (*Code Motion*)
2. Eliminisanje indukcionih promenljivih
3. Direktna redukcija

**-Pomeranje koda (*Code Motion*)**

Transformacija se sastoji u tome da se iz petlje izbacuju izrazi čija se vrednost ne menja u. Ako se izračunavanje takvih izraza prebaci u deo za inicijalizaciju petlje onda će se oni izračunavati samo jednom umesto da se izračunavaju prilikom svakog prolaza kroz petlju.

while (x+y > 0) -> z=x+y while (z > 0)

**-Eliminisanje indukcionih promenljivih**

Indukcione promenljive su promenljive čija promena izaziva promenu drugih promenljivih., kada vrednost jedne promenljive direktno zavisi od druge.

i = i + 1

t2 :=4\*i.

* t2 := t2 + 4.

**-Direktna redukcija**

Direktna redukcija se odnosi na transformacije kojima se složenije (skuplje) naredbe zamenjuju jednostavnijim (jeftinijim) naredbama