Generisanje koda

- Generator koda:
  - preuzima ispravan međukod i tabelu simbola, a
  - proizvodi
    1) izvršni (apsolutni) mašinski kod ili
    2) objektni (relokatibilni) mašinski kod ili
    3) asemblerski kod
- Podrazumeva se da je proizvedeni kod ispravan i da efikasno koristi resurse računara
- Generisanje koda se zasniva na određivanju načina implementacije svake naredbe međukoda pomoću naredbi koda

Generisanje koda

- Na primer, naredba hipotetskog asemblerskog jezika:

```
ADDS   x,y,z
```

  može biti implementirana pomoću sledećih naredbi asemblerskog jezika za procesor *Intel 8086*:

```
movw   x,%ax
addw   y,%ax
into
movw   %ax,z
```

- Generisanje koda može da proizvede suvišne naredbe. Na primer za međukod:

```
ADDS   x,y,z
ADDS   z,p,q
```

  generisani kod je:

Generisanje koda

```
movw   x,%ax        naredba 1
addw   y,%ax        naredba 2
into                naredba 3
movw   %ax,z        naredba 4
movw   z,%ax        naredba 5
addw   p,%ax        naredba 6
into                naredba 7
movw   %ax,q        naredba 8
```

a u njemu je peta naredba suvišna, a potencijalno i četvrta, ako se promenljiva **z** ne koristi iza dotičnog baznog bloka

- Zato se u optimizaciju koda obično uključuje parcijalna optimizacija, koja traži pojave slučajeva kao što su prethodno pomenuti

Generisanje koda

- Prilikom generisanja koda treba voditi računa o specifičnostima mašine, pa naredbu međukoda

  ```
  ADDS  x,$1,x
  ```

  treba implementirati naredbom

  ```
  incw  x
  ```

  a ne pomoću prethodno korišćene četiri naredbe
- Za generisanje koda je važno korišćenje registara (procesora), jer je registarsko adresiranje najbrži i najkraći način adresiranja
- Zato je potrebno najčešće korišćene vrednosti čuvati u registrima
- To se ostvaruje određivanjem promenljivih kojima treba dodeliti registar i zauzimanjem registara za pojedine od ovih promenljivih (*C* kompajler dozvoljava programeru da označi promenljive kojima treba dodeliti registre)

Generisanje koda - registri

- Problem:
  - napraviti međukod tako da ne koristi više privremenih promenljivih nego što ima registara
- Metoda:
  - dodeliti više promenljivih svakom registru
  - bez izmene ponašanja programa

    privr. prom. → reg.
  - mapiranje: više → jedan
  - primer programa:

```
a := c + d     više  →  jedan      r1 := r2 + r3
e := a + b     --------------->    r1 := r1 + r4
f := e – 1                        r1 := r1 – 1
```

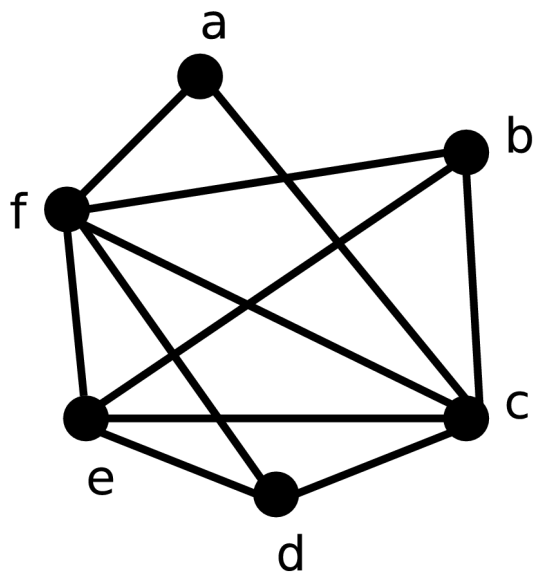
  - pretpostavka je da su promenljive **a** i **e** mrtve nakon upotrebe
  - **a**, **e** i **f** se mogu dodeliti jednom registru (**r1**)

Generisanje koda - registri

- Alokacija registara je stara koliko i kompajleri
  - prvi put je upotrebljena u originalnom FORTRAN kompajleru 1950-tih godina
  - sirovi, grubi algoritmi
  - usko grlo generisanja koda
- Proboj 1980
  - šema alokacije registara zasnovana na bojenju grafa
  - relativno jednostavna, globalna i veoma dobra u praksi
  - *privremene promenljive t1 i t2 mogu deliti isti registar ako je u bilo kojoj tački programa najviše jedna od njih živa*
  - ili
  - *ako se t1 i t2 koriste u isto vreme, onda ne mogu deliti isti registar*

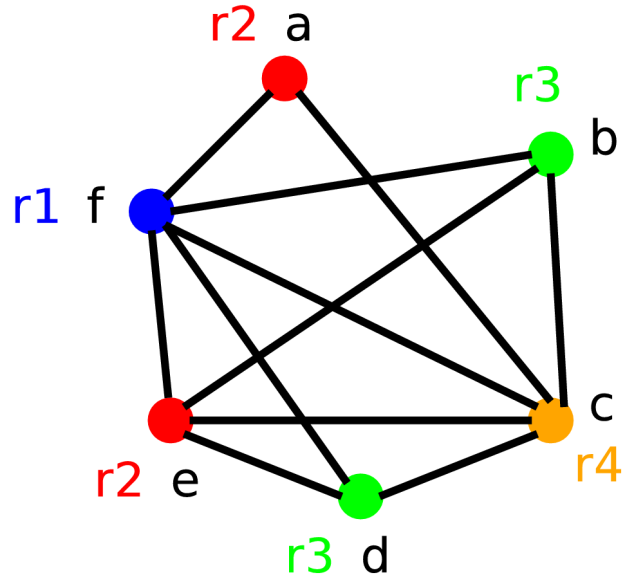
Generisanje koda - registri

- **RIG: *register interference graph***
  - čvorove grafa predstavljaju promenljive koje su dodeljene registrima
  - međusobno su povezani čvorovi promenljivih koje se istovremeno koriste
  - 2 promenljive mogu biti dodeljene istom registru ako nisu povezane u grafu
    - a i b
    - a i d
    - b i d
  - daje globalnu sliku o potrebi za registrima
  - posle pravljenja RIG grafa, algoritam alokacije registara je nezavisan od arhitekture



Generisanje koda - registri

- **algoritam bojenja grafa** *(graph coloring)*
  - prvo RIG
  - onoliko boja koliko ima radnih registara
  - čvorovi grafa se boje tako da međusobno povezani čvorovi budu obojeni različitim bojama
  - na ovaj način se sprečava da za dve promenljive, koje se istovremeno koriste bude zauzet isti radni registar



Generisanje koda - registri

- Šta se dešava kada bojenje grafa ne uspe?
  - u tom slučaju se ne mogu držati sve vrednosti u registrima
  - neke vrednosti se **prelivaju** u memoriju (*spilling*)
  - na neki način se odabere čvor koji će se preliti u memoriju
    - bilo koji izbor je ispravan
    - pitanje je performansi
  - pre svake operacije koja čita **`f`** se ubacuje: **`f := load fa`**
  - posle svake operacije koja piše **`f`** se ubacuje: **`store f, fa`**
  - onda se ponovo pokuša bojenje grafa
    - možda je potrebno još prelivanja u memoriju
- Alokacija registara je jedan od najznačajnijih poslova koje radi kompajler
  - od načina alokacije registara zavise performanse i efikasnost

Generisanje koda

- Za uspešno generisanje koda potrebno je
  - pamtiti kako se koristi svaki radni registar (za šta se uvodi posebna struktura podataka)
  - pamtiti za svaku vrednost (konstante, promenljive, funkcije, ...) gde se nalazi (zato se koristi tabela simbola)
  - pripremiti generisanje naredbe zauzimanjem radnog registra i pronalaženjem gde se nalaze potrebne vrednosti
  - generisati naredbu (vodeći računa da se odabere najbolja alternativa, ako se mogu birati razne naredbe i načini adresiranja)
  - ažurirati podatke o korišćenju radnih registara i o mestima smeštanja vrednosti

Generisanje koda

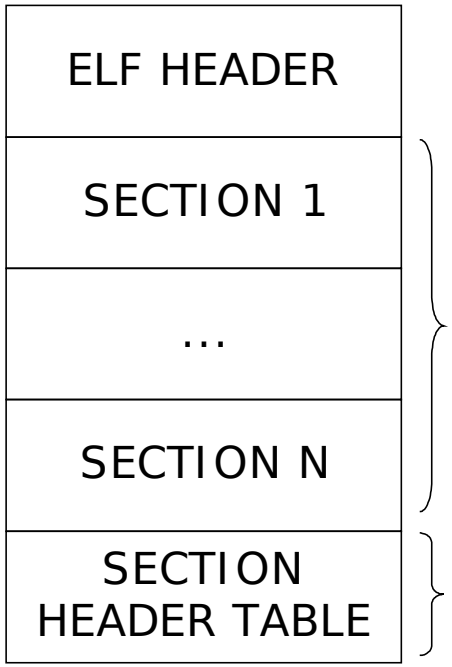
- Rukovanje radnim registrima može uticati na izbor redosleda (nezavisnih) naredbi (čiji međusobni položaj je proizvoljan)
- Postoje **generatori generatora koda** *(code-generator generators)* koji automatizuju pravljenje generatora koda *(burg, iburg)*

Optimizacija koda

- Zavisi od optimizacije međukoda
- Mašinski zavisna
- Obuhvata parcijalnu optimizaciju
- Ima za cilj da na najbolji način iskoristi specifičnosti računara
- U modernim kompajlerima optimizacija koda je najveća i najkompleksnija faza kompajliranja

Format objektne datoteke

- ELF *(Executable and Linkable Format)* format objektne datoteke (*UNIX/LINUX*):



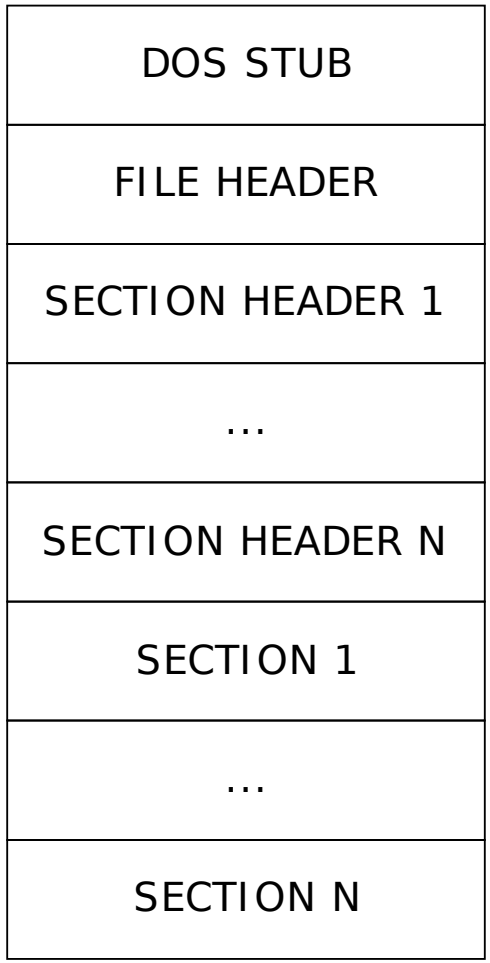
ELF zaglavlje se uvek nalazi na pocetku datoteke i opisuje organizaciju datoteke

Sekcija sadrži naredbe, podatke, tabelu simbola, informacije o relokaciji, itd.
Neke od specijalnih sekcija koje se mogu naci u ELF objektnoj datoteci su: .text (naredbe), .data (inicijalizovane promenljive), .bss (neinicijalizovane promenljive), .debug (debug informacije), .symtab (tabela simbola), itd.

Sve sekcije u objektnoj datoteci se mogu naci pomocu ove tabele. Svako polje tabele odgovara jednoj sekciji u datoteci, i sadrži ime, tip, pocetnu adresu, velicinu, poravnanje itd.

Format objektne datoteke

- PE *(Portable Executable)* format objektne datoteke (*Windows*):



Svaka PE datoteka je izvršiva pod MS-DOS-om.

Zaglavlje datoteke govori za koju mašinu je binarni kod predviden, koliko sekcija sadrži, vreme linkovanja, da li sadrži izvršni kod ili DLL, itd.

Zaglavlje svake sekcije opisuje da li sadržaj sekcije predstavlja izrvšiv kod, inicijalizovane promenljive, neinicijalizovane promenljive, da li se promenljive samo mogu citati ili se u njih može i pisati, itd.

Svaka sekcija sadrži podatke o poravnanju, podatke o vrsti sadržaja, da li može biti "shared" itd., i sam sadržaj - ono što je neophodno za izvršavanje programa.
Postoji nekoliko predefinisanih sekcija: .text (sadrži kod), .bss, .rdata, .data (sadrže neinicijalizovane, read-only i ostale promenljive), .rsrc (sadrži informacije o resursima), .debug (sadrži debug informacije).