Hipotetski asemblerski jezik

- Podrazumeva se da registri i memorijske lokacije zauzimaju po 4 bajta
- Ukupno ima 16 registara
  - oznaka registra se sastoji od oznake **%** i rednog broja registra: **%0**, **%1**, ..., **%15**
  - registri **%0** do **%12** imaju opštu namenu i služe kao radni registri
  - registar **%13** rezervisan je za povratnu vrednost funkcije
  - registar **%14** služi kao pokazivač frejma
  - registar **%15** služi kao pokazivač steka
- Labele započinju malim slovom iza koga mogu da slede mala slova, cifre i podcrta '_' (alfabet je 7 bitni *ASCII*); iza labele se navodi dvotačka
- ispred sistemskih labela se navodi znak '@'

Hipotetski asemblerski jezik

- Operandi
  - **neposredni operand** odgovara celom (označenom ili neoznačenom) broju:

    `$0` ili **`$-152`**

    a njegova vrednost vrednosti tog broja, dok

    `$labela`

    odgovara adresi labele **`labela`**
  - **registarski operand** odgovara oznaci registra, a njegova vrednost sadržaju tog registra
  - **direktni operand** odgovara labeli. Njegova vrednost odgovara adresi labele, ako ona označava naredbu i koristi se kao operand naredbe skoka ili poziva potprograma. Ako direktni operand odgovara labeli koja označava direktivu i ne koristi se kao operand naredbe skoka ili poziva potprograma, njegova vrednost odgovara sadržaju adresirane lokacije.

Hipotetski asemblerski jezik

- **indirektni operand** odgovara oznaci registra navedenoj između malih zagrada:

  `(%0)`

  a njegova vrednost sadržaju memorijske lokacije koju adresira sadržaj registra
- **indeksni operand** započinje celim (označenim ili neoznačenim) brojem ili labelom iza čega sledi oznaka registra navedena između malih zagrada:

  `-8(%14)` ili **`4(%14)`** ili **`tabela(%0)`**

  Njegova vrednost odgovara sadržaju memorijske lokacije koju adresira zbir vrednosti broja i sadržaja registra, odnosno zbir adrese labele i sadržaja registra
- operandi se dele na ulazne (neposredni, registarski, direktni, indirektni i indeksni) i izlazne (registarski, direktni, indirektni i indeksni)

Hipotetski asemblerski jezik

- **naredba poređenja** brojeva postavlja bite status registra u skladu sa razlikom prvog i drugog ulaznog operanda

```
CMPx    ulazni operand, ulazni operand
```

`x:` **S** označeni
U neoznačeni
F realni (mašinska normalizovana forma)

Hipotetski asemblerski jezik

- **naredba bezuslovnog skoka** smešta u programski brojač vrednost ulaznog operanda (omogućujući tako nastavak izvršavanja od ciljne naredbe koju adresira ova vrednost)

```
JMP    ulazni operand
```

Hipotetski asemblerski jezik

- **naredbe uslovnog skoka** smeštaju u programski brojač vrednost ulaznog operanda samo ako je ispunjen uslov određen kodom naredbe (ispunjenost uslova zavisi od bita status registra)

```
JEQ    ulazni operand
JNE    ulazni operand
JGTx   ulazni operand
JLTx   ulazni operand
JGEx   ulazni operand
JLEx   ulazni operand
```

x: **S** označeni
U neoznačeni
F realni

Hipotetski asemblerski jezik

- **naredbe rukovanja stekom** omogućuju smeštanje na vrh steka vrednosti ulaznog operanda, odnosno preuzimanje vrednosti sa vrha steka i njeno smeštanje u izlazni operand (podrazumeva se da **%15** služi kao pokazivač steka, da se stek puni od viših lokacija ka nižim i da **%15** pokazuje vrh steka)

```
PUSH   ulazni operand
POP    izlazni operand
```

Hipotetski asemblerski jezik

- **naredba poziva potprograma** smešta na vrh steka zatečeni sadržaj programskog brojača, a u programski brojač smešta vrednost ulaznog operanda:

  ```
  CALL    ulazni operand
  ```

- **naredba povratka iz potprograma** preuzima vrednost sa vrha steka i smešta je u programski brojač

  ```
  RET
  ```

Hipotetski asemblerski jezik

- **naredba za sabiranje brojeva**

`ADDx  ulazni operand, ulazni operand, izlazni operand`

omogućuje sabiranje ulaznih operanada, uz izazivanje izuzetka ako zbir ne može da stane u izlazni operand

- **naredba za oduzimanje brojeva**

`SUBx  ulazni operand, ulazni operand, izlazni operand`

omogućuje oduzimanje ulaznih operanada, uz izazivanje izuzetka ako razlika ne može da stane u izlazni operand

`x:` **`S`** označeni
`U` neoznačeni
`F` realni (mašinska normalizovana forma)

Hipotetski asemblerski jezik

- **naredba za množenje brojeva**

`MULx  ulazni operand, ulazni operand, izlazni operand`

omogućuje množenje ulaznih operanada, uz izazivanje izuzetka ako proizvod ne može da stane u izlazni operand

- **naredba za delenje brojeva**

`DIVx  ulazni operand, ulazni operand, izlazni operand`

omogućuje delenje prvog ulaznog operanda drugim i smeštanje količnika u izlazni operand

`x:` **`S`** označeni
`U` neoznačeni
`F` realni (mašinska normalizovana forma)

Hipotetski asemblerski jezik

- **naredba za prebacivanje vrednosti**

```
MOV    ulazni operand, izlazni operand
```

Hipotetski asemblerski jezik

- **naredba konverzije celog broja u razlomljeni broj**

```
TOF   ulazni operand, izlazni operand
```

(vrednost ulaznog operanda je celi broj, a vrednost izlaznog operanda je ekvivalentni razlomljeni broj u mašinskoj normalizovanoj formi)

- **naredba konverzije razlomljenog broja u celi broj**

```
TOI   ulazni operand, izlazni operand
```

(vrednost ulaznog operanda je razlomljeni broj u mašinskoj normalizovanoj formi, a vrednost izlaznog operanda je ekvivalentni celi broj ako je konverzija moguća, inače se izaziva izuzetak)

Hipotetski asemblerski jezik

- **direktiva zauzimanja memorijskih lokacija** omogućuje zauzimanje broja uzastopnih memorijskih lokacija koji je naveden kao njen operand

```
WORD   broj
```

Primer generisanja koda – globalne promenljive

- Globalne promenljive
  - za svaku promenljivu generisati asemblersku direktivu sa odgovarajućom labelom
  - primer globalne promenljive

```
int a;
```

  - generisani kod

```
a:
        WORD    1
```

(labela "a" ne započinje znakom "@" jer odgovara identifikatoru koga je zadao korisnik)

Primer generisanja koda – iskaz pridruživanja

- Iskaz pridruživanja
  - primer iskaza pridruživanja

    a = (-a + b) * (c + d) - e

    (podrazumeva se da su **a**, **b**, **c**, **d** i **e** celobrojne označene globalne promenljive)
  - za smeštanje međurezultata izraza koriste se radni registri

Primer generisanja koda – iskaz pridruživanja

- generisani kod za iskaz: **`a = (-a + b) * (c + d) - e`**

```
SUBS   $0,a,%0      linija 1
ADDS   %0,b,%0      linija 2
ADDS   c,d,%1       linija 3
MULS   %0,%1,%0     linija 4
SUBS   %0,e,%0      linija 5
MOV    %0,a         linija 6
```

podrazumeva se da su svi radni registri slobodni

- u *liniji 1* se zauzima radni registar **`%0`**
- on se oslobađa i ponovo zauzima u *liniji 2*
- u *liniji 3* se zauzima radni registar **`%1`**
- u *liniji 4* se oslobađaju radni registri **`%0`** i **`%1`**, a ponovo se zauzima radni registar **`%0`**
- u *liniji 5* se oslobađa i ponovo zauzima radni registar **`%0`**
- u *liniji 6* se oslobađa radni registar **`%0`**

Primer generisanja koda – iskaz pridruživanja

- radni registar se zauzima za smeštanje rezultata svakog aritmetičkog izraza sa:
  - dva operanda i jednim operatorom

    (*num_exp*, *mul_exp*)
  - unarnim operatorom i jednim operandom

    (*exp*)
- radni registar se oslobađa čim se preuzme njegova vrednost

Primer generisanja koda – iskaz pridruživanja

- pošto se međurezultati izraza koriste u suprotnom redosledu od onog u kome su izračunati, radni registri, koji se koriste za smeštanje međurezultata, se zauzimaju i oslobađaju po principu steka
- kao "pokazivač steka registara" koristi se promenljiva **`free_reg_num`**, koja sadrži broj prvog slobodnog radnog registra
- zauzimanje radnog registra se sastoji od preuzimanja vrednosti promenljive **`free_reg_num`** i njenog inkrementiranja
- oslobađanje radnog registra se sastoji od dekrementiranja promenljive **`free_reg_num`**
- treba zapaziti da je broj registra istovremeno i indeks elementa tabele simbola

Primer generisanja koda – iskaz pridruživanja

- broj zauzetog radnog registra služi kao vrednost sintaksnog pojma koji odgovara izrazu čiji rezultat radni registar sadrži
- prekoračenje broja radnih registara (**`free_reg_num > 12`**) predstavlja fatalnu grešku u radu kompajlera

Primer generisanja koda – funkcija

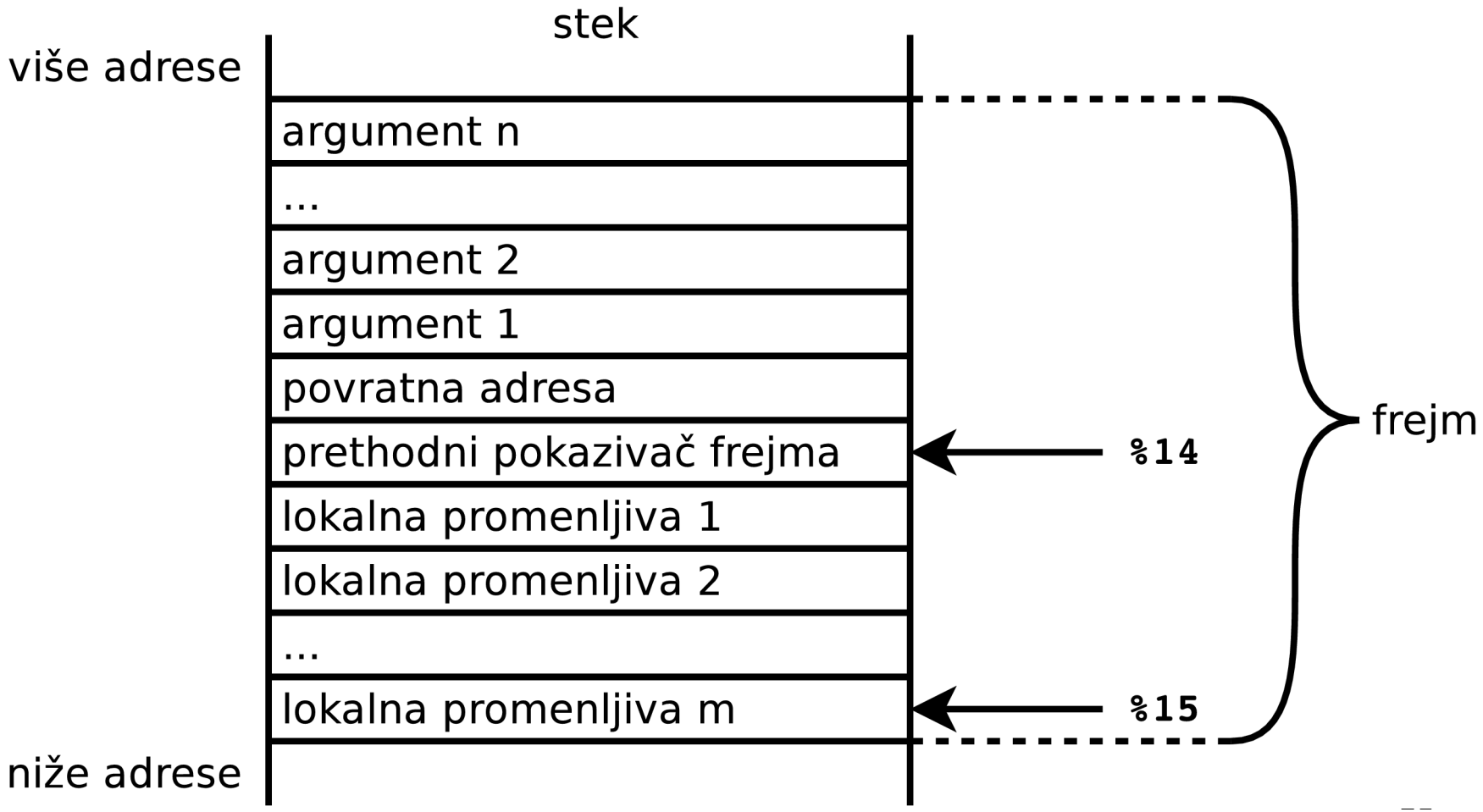
- Funkcija
  - primer funkcije

```
int f(int x, int y) {
  int z;
  return x + y;
}
```

  - za smeštanje povratne vrednosti funkcije koristi se radni registar **%13**

Primer generisanja koda – funkcija

- Funkcija, nakon poziva, lokalne promenljive i parametre čuva u stek frejmu:



Primer generisanja koda – funkcija

- generisani kod

```
f:                                          linija 1
        PUSH   %14                          linija 2
        MOV    %15,%14                      linija 3
        SUBS   %15,$4,%15                   linija 4
@f_body:                                    linija 5
        ADDS   8(%14),12(%14),%0            linija 6
        MOV    %0,%13                       linija 7
        JMP    @f_exit                      linija 8
@f_exit:                                    linija 9
        MOV    %14,%15                      linija 10
        POP    %14                          linija 11
        RET                                 linija 12
```

Primer generisanja koda – funkcija

- u *linijama 2 i 3* se postavlja pokazivač frejma
- u *liniji 4* se zauzima prostor na steku za lokalnu promenljivu **z**
  - brojač lokalnih promenljivih **var_num**
  - veličina prostora za lokalne promenljive je **var_num * 4**
  - prostor se zauzima samo ako je **var_num > 0**
- u *linijama 6 i 7* se računa povratna vrednost funkcije i smešta u registar **%13**
  - ako funkcija ne sadrži **return** iskaz, kao povratna vrednost funkcije služi zatečeni sadržaj registra **%13**, koji je nepoznat u vreme definisanja funkcije
- u *liniji 10* se oslobađa prostor za lokalne promenljive
- u *liniji 11* se vraća prethodna vrednost u pokazivač frejma

Primer generisanja koda – poziv funkcije

- Poziv funkcije
  - primer poziva funkcije

    `a = f(a + b, c – d)`

    (podrazumeva se da su **`a`**, **`b`**, **`c`** i **`d`** celobrojne označene globalne promenljive)
  - generisani kod

```
ADDS   a,b,%0        linija 1
SUBS   c,d,%1        linija 2
PUSH   %1            linija 3
PUSH   %0            linija 4
CALL   f             linija 5
ADDU   %15,$8,%15    linija 6
MOV    %13, a        linija 7
```

Primer generisanja koda – poziv funkcije

- u *linijama 1 i 2* se računaju argumenti i čuvaju u radnim registrima (pretpostavka je da su svi radni registri slobodni)
- u *linijama 3 i 4* argumenti se smeštaju na stek
- u *liniji 6* oslobađa se prostor koji su zauzimali argumenti
  - veličina oslobađanog prostora je **`arg_num * 4`**
  - ovaj prostor se oslobađa samo ako je **`arg_num > 0`**
- u *liniji 7* se isporučuje povratna vrednost funkcije

Primer generisanja koda – poziv funkcije

- ako su argumenti konstante ili promenljive, kao u slučaju poziva

```
a = f(1,b);
```

(podrazumeva se da su **a** i **b** celobrojne označene globalne promenljive)

onda nisu potrebni radni registri, jer se konstante i promenljive direktno mogu smeštati na stek:

```
PUSH   b
PUSH   $1
CALL   f
ADDU   %15,$8,%15
MOV    %13,a
```

Primer generisanja koda – poziv funkcije

- pošto argumentima odgovaraju ili radni registri ili konstante ili promenljive, svakom argumentu je pridružen indeks elementa tabele simbola koji je rezervisan ili za pomenuti radni registar, ili za pomenutu konstantu ili za pomenutu promenljivu
- pošto se argumenti na stek smeštaju od poslednjeg ka prvom, zgodno je na posebnom **steku argumenata** kompajlera čuvati njihove indekse
- ovi indeksi se smeštaju na stek argumenata od prvog ka poslednjem argumentu, tako da su pripremljeni za generisanje naredbi koje smeštaju argumente na stek
- za čuvanje broja argumenata iz poziva funkcije zgodno je uvesti poseban **stek broja argumenata** kompajlera (ovaj stek omogućuje da se pamte brojevi argumenata kada se u pozivu neke funkcije kao argument javi opet poziv funkcije)

Primer generisanja koda – poziv funkcije

- ako su argumenti pozivi novih funkcija

  ```
  a = f(1,f2(3));
  ```

  (podrazumeva se da je **a** celobrojna označena globalna promenljiva, a da funkcija **f2** ima jedan parametar tipa **int**)
- generisani kod

```
PUSH   $3              linija 1
CALL   f2              linija 2
ADDU   %15,$4,%15      linija 3
MOV    %13,%0          linija 4
PUSH   %0              linija 5
PUSH   $1              linija 6
CALL   f               linija 7
ADDU   %15,$8,%15      linija 8
MOV    %13,a           linija 9
```

Primer generisanja koda – poziv funkcije

- kada se u pozivu neke funkcije kao argument javi opet poziv funkcije, prvo se izvršava poziv te funkcije da bi se izračunala vrednost odgovarajućeg argumenta
- u *linijama 1, 2 i 3* se izvršava poziv funkcije **f2** da bi se izračunala vrednost drugog argumenta poziva funkcije **f**

- u slučaju pojave poziva funkcije na mestu argumenta poziva neke druge funkcije zgodno je uvesti poseban **stek poziva funkcija** kompajlera
- ovaj stek omogućuje da se pamte indeksi (imena) funkcija u tabeli simbola čiji pozivi su se javili kao argument poziva neke funkcije

Primer generisanja koda – iskaz poziva funkcije

- Iskaz poziva funkcije
  - primer iskaza poziva funkcije

```
f(1,2);
```

  - generisani kod

```
PUSH   $2
PUSH   $1
CALL   f
ADDU   %15,$8,%15
```

Primer generisanja koda – **`if`** iskaz

- **`if`** iskaz sa **`else`** delom
  - primer **`if`** iskaza

```
if(a > b)
   a = 1;
else
   a = 2;
```

(podrazumeva se da su **`a`** i **`b`** celobrojne označene globalne promenljive)

  - generisani kod

Primer generisanja koda – **if** iskaz

```
@if0:                          linija 1
        CMPS   a,b             linija 2
        JLES   @false0         linija 3
@true0:                        linija 4
        MOV    $1,a            linija 5
        JMP    @exit0          linija 6
@false0:                       linija 7
        MOV    $2,a            linija 8
@exit0:                        linija 9
```

Primer generisanja koda – **`if`** iskaz

- labele u generisanom kodu moraju biti jedinstvene
  - svaka labela se završava brojem
  - promenljiva **`label_number`** sadrži aktuelni broj labele
  - jednoznačni brojevi se dobijaju inkrementiranjem promenljive **`label_number`**
  - broj uz labelu **`if`** se čuva jer se koristi i uz labele
    - **`@true`**
    - **`@exit`**
  - promenljiva **`false_label_number`** sadrži aktuelni broj **`@false`** labele
  - neophodan je zaseban brojač **`false_label_number`** pošto u jednom logičkom izrazu može biti više **`false`** labela

Primer generisanja koda – **`if`** iskaz

- **`if`** iskaz sa **`else`** delom
  - primer **`if`** iskaza

```
if(a > b && c > d)
   a = 1;
else
   a = 2;
```

(podrazumeva se da su **a**, **b**, **c** i **d** celobrojne označene globalne promenljive)

  - generisani kod (podrazumeva se da se generisanje koda nastavlja na prethodni primer)

Primer generisanja koda – **`if`** iskaz

```
@if1:
        CMPS    a,b         linija 1
        JLES    @false1     linija 2
        CMPS    c,d         linija 3
        JLES    @false1     linija 4
@true1:                     linija 5
        MOV     $1,a        linija 6
        JMP     @exit1      linija 7
@false1:                    linija 8
        MOV     $2,a        linija 9
@exit1:                     linija 10
                            linija 11
```

Primer generisanja koda – **`if`** iskaz

- **`if`** iskaz sa **`else`** delom
  - primer **`if`** iskaza

```
if(c > d || a > c)
   a = 1;
else
   a = 2;
```

(podrazumeva se da su **a**, **b**, **c** i **d** celobrojne označene globalne promenljive)

  - generisani kod (podrazumeva se da se generisanje koda nastavlja na prethodni primer)

Primer generisanja koda – **if** iskaz

```
@if2:
        CMPS    c,d             linija 1
        JGTS    @true2          linija 2
@false2:                        linija 3
        CMPS    a,c             linija 4
        JLES    @false3         linija 5
@true2:                         linija 6
        MOV     $1,a            linija 7
        JMP     @exit2          linija 8
@false3:                        linija 9
        MOV     $2,a            linija 10
@exit2:                         linija 11
                                linija 12
```

Primer generisanja koda – **`if`** iskaz

- **`if`** iskaz sa **`else`** delom
  - primer **`if`** iskaza

```
if(a > b && c > d || a > c && b > d || a > e)
   a = 1;
else
   a = 2;
```

(podrazumeva se da su **`a`**, **`b`**, **`c`**, **`d`** i **`e`** celobrojne označene globalne promenljive)

  - generisani kod

Primer generisanja koda – **if** iskaz

```
@if0:                          linija 1
      CMPS  a,b                linija 2
      JLES  @false0            linija 3
      CMPS  c,d                linija 4
      JGTS  @true0             linija 5
@false0:                       linija 6
      CMPS  a,c                linija 7
      JLES  @false1            linija 8
      CMPS  b,d                linija 9
      JGTS  @true0             linija 10
@false1:                       linija 11
      CMPS  a,e                linija 12
      JLES  @false2            linija 13
@true0:                        linija 14
      MOV   $1,a               linija 15
      JMP   @exit0             linija 16
@false2:                       linija 17
      MOV   $2,a               linija 18
@exit0:                        linija 19
```

Primer generisanja koda – **`if`** iskaz

- Pošto se kao **`then`** iskaz **`if`** iskaza može pojaviti novi **`if`** iskaz i tako dalje, neophodno je sačuvati zatečenu vrednost promenljive **`label_number`**, jer će ona biti izmenjena u toku generisanja koda za novi **`if`** iskaz (zbog sekvencijalnog generisanja koda zatečena vrednost će biti naknadno korišćena)
- Iz istih razloga se mora sačuvati i vrednost promenljive **`false_label_number`** pošto se ona koristi uz poslednju labelu **`@false`** u delu koda koji još nije izgenerisan za spoljašnji **`if`** iskaz

Primer generisanja koda – **`if`** iskaz

- za čuvanje vrednosti ovih promenljivih se koristi poseban **stek labela** kompajlera
- na njega se ove vrednosti smeštaju pre tretiranja **`then`** iskaza
- nakon tretiranja **`then`** iskaza obe vrednosti se preuzimaju sa steka labela
- isto važi i za vrednost promenljive **`label_number`** pre i nakon tretiranja **`else`** iskaza

Primer generisanja koda – `if` iskaz

- Od *linije 2* do *linije 13* je opisano određivanje vrednosti logičkog izraza
- generisanje naredbi poređenja je vezano za pojam *`rel_exp`*
- kao vrednost ovog pojma služi vrednost relacionog operatora, da bi se na osnovu nje mogla izgenerisati ispravna naredba uslovnog skoka
- njeno generisanje je vezano za *&&* operator pojma *`and_exp`*, || operator pojma *`log_exp`* ili za pojam *`log_exp`*

Primer generisanja koda – **`if`** iskaz

- **`if`** iskaz bez **`else`** dela
  - primer **`if`** iskaza

```
if(a > b)
   a = 1;
```

(podrazumeva se da su **`a`** i **`b`** celobrojne označene globalne promenljive)

  - generisani kod

Primer generisanja koda – **if** iskaz

```
@if0:
      CMPS  a,b
      JLES  @false0
@true0:
      MOV   $1,a
      JMP   @exit0
@false0:
@exit0:
```

Primer generisanja koda – **`if`** iskaz

- **`if`** iskaz bez **`else`** dela
  - primer **`if`** iskaza

```
if(a > b && c > d || a > c && b > d || a > e)
   a = 1;
```

(podrazumeva se da su **`a`**, **`b`**, **`c`**, **`d`** i **`e`** celobrojne označene globalne promenljive)

  - generisani kod

Primer generisanja koda – **`if`** iskaz

```
@if0:                                   linija 1
        CMPS    a,b                     linija 2
        JLES    @false0                 linija 3
        CMPS    c,d                     linija 4
        JGTS    @true0                  linija 5
@false0:                                linija 6
        CMPS    a,c                     linija 7
        JLES    @false1                 linija 8
        CMPS    b,d                     linija 9
        JGTS    @true0                  linija 10
@false1:                                linija 11
        CMPS    a,e                     linija 12
        JLES    @false2                 linija 13
@true0:                                 linija 14
        MOV     $1,a                    linija 15
        JMP     @exit0                  linija 16
@false2:                                linija 17
@exit0:                                 linija 18
```

- Od *linije 2* do *linije 13* je opisano određivanje vrednosti logičkog izraza

Primer generisanja koda – **`while`** iskaz

- **`while`** iskaz
  - primer **`while`** iskaza

```
while (a != b)
   if (a > b)
      a = a - b;
   else
      b = b - a;
```

(podrazumeva se da su **`a`** i **`b`** celobrojne označene globalne promenljive)

  - za generisanje koda primenjuje se pristup objašnjen u **`if`** iskazu
  - generisani kod

Primer generisanja koda – **while** iskaz

```
@while0:
        CMPS    a,b
        JEQ     @false0
@true0:
@if1:
        CMPS    a,b
        JLES    @false1
@true1:
        SUBS    a,b,%0
        MOV     %0,a
        JMP     @exit1
@false1:
        SUBS    b,a,%0
        MOV     %0,b
@exit1:
        JMP     @while0
@false0:
@exit0:
```

Primer generisanja koda – **`while`** iskaz

- **`while`** iskaz
  - primer **`while`** iskaza

```
while(a > b && c > d || a > c && b > d || a > e)
    a = a + e;
```

    (podrazumeva se da su **`a`**, **`b`**, **`c`**, **`d`** i **`e`** celobrojne označene globalne promenljive)
  - za generisanje koda primenjuje se pristup objašnjen u **`if`** iskazu
  - generisani kod

Primer generisanja koda – **while** iskaz

```
@while0:                          linija 1
        CMPS    a,b               linija 2
        JLES    @false0           linija 3
        CMPS    c,d               linija 4
        JGTS    @true0            linija 5
@false0:                          linija 6
        CMPS    a,c               linija 7
        JLES    @false1           linija 8
        CMPS    b,d               linija 9
        JGTS    @true0            linija 10
@false1:                          linija 11
        CMPS    a,e               linija 12
        JLES    @false2           linija 13
@true0:                           linija 14
        ADDS    a,e,%0            linija 15
        MOV     %0,a              linija 16
        JMP     @while0           linija 17
@false2:                          linija 18
@exit0:                           linija 19
```

- Od *linije 2* do *linije 13* je opisano određivanje vrednosti logičkog izraza

Primer generisanja koda – **break** iskaz

- **break** iskaz
  - primer **break** iskaza

```
while(a < 5) {
   if(a == b)
      break;
   a = a + 1;
}
```

(podrazumeva se da su **a** i **b** celobrojne označene globalne promenljive)

  - podrazumeva se da se **break** iskaz sme naći samo unutar **while** iskaza
  - generisani kod

Primer generisanja koda – **`break`** iskaz

```
@while0:
        CMPS   a,$5
        JGES   @false0
@true0:
@if1:
        CMPS   a,b
        JNE    @false1
@true1:
        JMP    @exit0          //break
        JMP    @exit1
@false1:
@exit1:
        ADDS   a,$1,%0
        MOV    %0,a
        JMP    @while0
@false0:
@exit0:
```

Primer generisanja koda – **`continue`** iskaz

- **`continue`** iskaz
  - primer **`continue`** iskaza

```
while(a < 5) {
   if(a == b)
      continue;
   a = a + 1;
}
```

(podrazumeva se da su **`a`** i **`b`** celobrojne označene globalne promenljive)

  - podrazumeva se da se **`continue`** iskaz sme naći samo unutar **`while`** iskaza
  - generisani kod

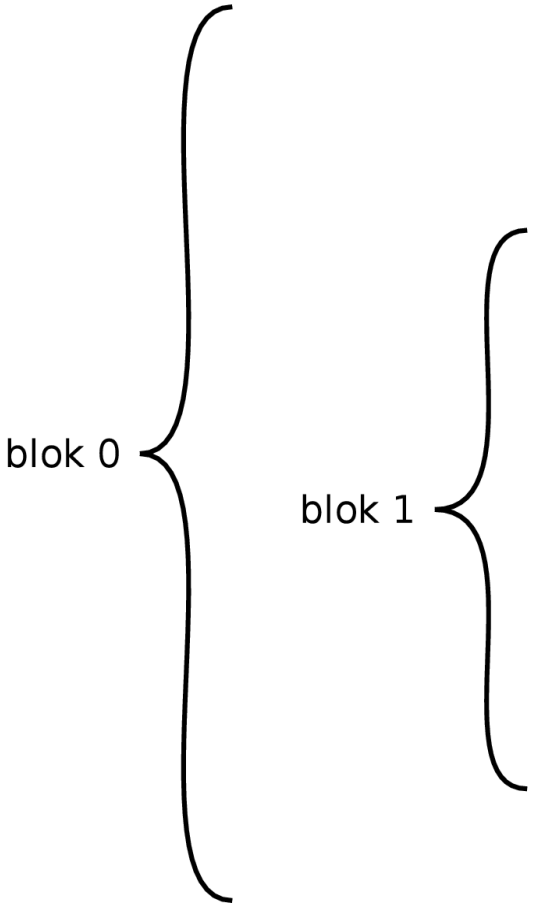
Primer generisanja koda – **continue** iskaz

```
@while0:
        CMPS   a,$5
        JGES   @false0
@true0:
@if1:
        CMPS   a,b
        JNE    @false1
@true1:
        JMP    @while0        //continue
        JMP    @exit1
@false1:
@exit1:
        ADDS   a,$1,%0
        MOV    %0,a
        JMP    @while0
@false0:
@exit0:
```

Generisanje koda – C blokovi

- C blokovi
  - blokovi se međusobno razlikuju po rednom broju koji im dodeljuje kompajler
  - u tabeli simbola za svaku lokalnu promenljivu bloka mora biti vezan redni broj bloka
  - kompajler koristi redne brojeve blokova kod određivanja područja važenja identifikatora (kada u bloku **n** naiđe na neku promenljivu, kompajler traži tu promenljivu za blok **n**, a ako je traženje neuspešno, ono se ponavlja za prethodni blok):

Generisanje koda – C blokovi




```
int main
{
  int x = 0;
  int y = 0;
  int z = 0;
  {
    int y = 1;
    {
      int z = 2;
      print(x,y,z);
    }
    {
      int z = 3;
      print(x,y,z);
    }
    print(x,y,z);
  }
  print(x,y,z);
}
```

Generisanje koda – C blokovi

- rezultat štampanja

```
0  1  2
0  1  3
0  1  0
0  0  0
```

- Lokalne promenljive blokova se čuvaju u frejmu bloka na steku (za razliku od frejma poziva funkcije, frejm bloka ne sadrži argumente kao ni povratnu vrednost)
- Frejm bloka se stvara na ulazu u blok, a uništava na izlasku iz bloka

Generisanje koda - slogovi

- Slogovi *(C struct, Pascal record)*
  - za svaki slog je potrebna posebna tabela simbola, koja sadrži njegova polja sa relativnom pozicijom u slogu kao dodatnim atributom
  - za slog

```
struct {
    int x;
    int y;
 } z
```

je potrebno zauzeti **2** lokacije. Iskazu

```
z.y = z.x;
```

odgovara kod

```
MOV   $z,%0
MOV   0(%0),4(%0)
```

(podrazumeva se da je radni registar **%0** slobodan i da je u njega smeštena početna adresa sloga **z**)

Generisanje koda - nizovi

- Nizovi
  - za svaki niz u tabeli simbola treba registrovati i broj njegovih elemenata
  - za niz

    ```
    int n[10];
    ```

    je potrebno zauzeti **10** lokacija. Iskazu

    ```
    n[0] = n[1];
    ```

    odgovara kod

    ```
    MOV   $n,%0
    MOV   4(%0),0(%0)
    ```

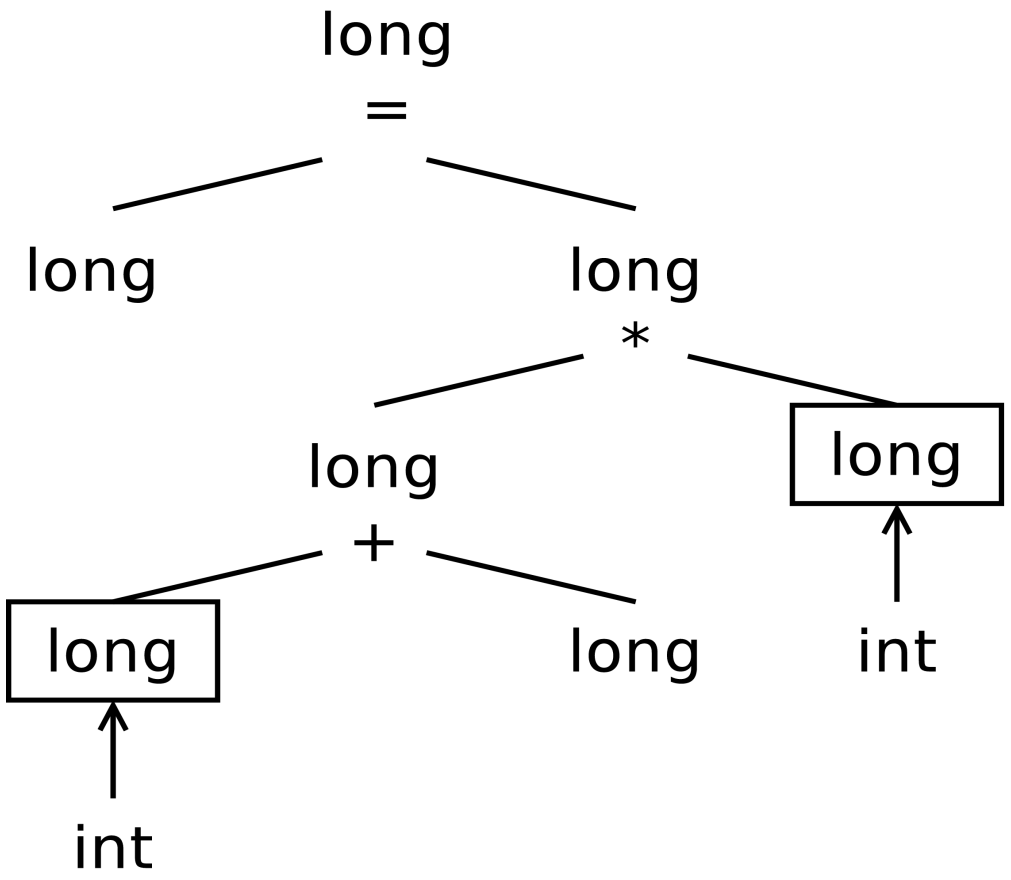
    (podrazumeva se da su radni registar **%0** slobodan i da je u njega smeštena početna adresa niza **n**)

Generisanje koda – konverzije tipova

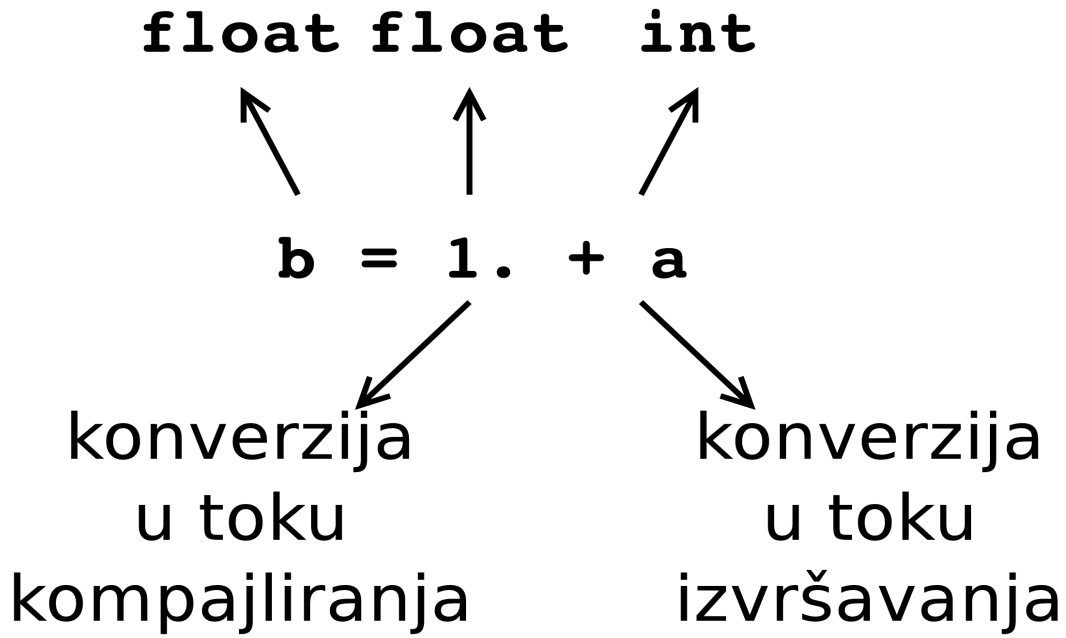
- Konverzije tipova u izrazima
  - u izrazima sa promenljivim i konstantama raznih ali kompatibilnih tipova (**`int`** i **`long`** ili **`int`** i **`float`**) potrebno je obaviti podrazumevajuće konverzije tipova (**`int`** u **`long`** ili **`int`** u **`float`**) pre korišćenja vrednosti promenljivih ili konstanti
  - konverzije tipova se zasnivaju na određivanju tipova izraza i iskaza. Za određivanje tipova izraza (iskaza) potrebno je izraze (iskaze) predstaviti u obliku grafa čiji čvorovi sadrže tipove podizraza (podiskaza).
  - konverzija se obavlja (ako je moguća) kada se prepozna binarni izraz sa operandima raznih ali kompatibilnih tipova, ili kada se prepozna dodela vrednosti jednog tipa promenljivoj različitog kompatibilnog tipa
  - radi konverzije kompajler generiše odgovarajuću naredbu za konverziju

Generisanje koda – konverzije tipova

long = (int + long) * int



Generisanje koda – konverzije tipova



Generisanje koda – vrednost logičkog izraza

- Pridruživanje vrednosti logičkih izraza
  - iskazu

```
bool = a > b && c > d || a > c && b > d || a > e;
```

(podrazumeva se da su sve promenljive globalne, celobrojne i označene) odgovara kod

Generisanje koda – vrednost logičkog izraza

```
        CMPS   a,b
        JLES   @false0
        CMPS   c,d
        JGTS   @true0
@false0:
        CMPS   a,c
        JLES   @false1
        CMPS   b,d
        JGTS   @true0
@false1:
        CMPS   a,e
        JLES   @false2
@true0:
        MOV    $1,bool
        JMP    @exit0
@false2:
        MOV    $0,bool
@exit0:
```

(podrazumeva se da su brojevi uz labele jedinstveni)

Generisanje koda – **switch** iskaz

- **switch** iskaz
  - iskazu

```
switch(state) {
   case 10 : state = 1;
             break;
   case 20 : state = 2;
             break;
   default : state = 0;
}
```

    (podrazumeva se da je **state** globalna celobrojna označena promenljiva) odgovara kod
  - napomena: realizovani kompajler je jednoprolazni, pa je u skladu sa tim i napravljen sledeći asemblerski kod

Generisanje koda – **switch** iskaz

```
@switch0:
            JMP     @test0
@case0_0:
            MOV     $1,state
            JMP     @exit0
@case0_1:
            MOV     $2,state
            JMP     @exit0
@default0:
            MOV     $0,state
            JMP     @exit0
@test0:
            CMPS    state,$10
            JEQ     @case0_0
            CMPS    state,$20
            JEQ     @case0_1
            JMP     @default0
@exit0:
```

(podrazumeva se da je prvi broj **0** uz labele jedinstven)

- da su izostavljeni **break** iskazi, bile bi izostavljene i prve dve naredbe **JMP exit0**

Generisanje koda – **`for`** iskaz

- **`for`** iskaz
  - iskazima

```
suma = 0;
for(i = 0; i <= 5; i++)
   suma = suma + i;
```

(podrazumeva se da su **`suma`** i **`i`** globalne, celobrojne i označene promenljive) odgovara kod

Generisanje koda – **for** iskaz

```
                MOV   $0,suma
                MOV   $0,i
@for0:
                CMPS  i,$5
                JGTS  @exit0
                ADDS  suma,i,%0
                MOV   %0,suma
                ADDS  i,$1,i
                JMP   @for0
@exit0:
```

(podrazumeva se da su brojevi uz labele jedinstveni)

MICKO (MIkro C KOmpajler)

- putanja: **`primeri/mC/micko/`**

- definicije konstanti (**`defs.h`**)
- Lex specifikacija (**`scanner.l`**)
- Bison specifikacija (**`parser.y`**)
- implementacija tabele simbola (**`symtab.h`**, **`symtab.c`**)
- implementacija steka (**`stack.h`**, **`stack.c`**)
- funkcije za semantičku analizu (**`sem.h`**, **`sem.c`**)
- funkcije za generisanje koda (**`codegen.h`**, **`codegen.c`**)