Programski prevodioci: Vežbe 7

Sadržaj

1. Uvod

U ovoj nedelji biće prikazana implementacija generisanja koda. Odnosno, biće prikazano na koji način MICKO kompajler konvertuje kod napisan u miniC jeziku u ekvivalentan kod u Hipotetsko asemblerskom jeziku.

2. Generisanje koda

Generisanje koda vrši se pozivom `code` makroa. Navedeni makro se koristi identično kao i makroi `err`, `warn`, ili funkcija `printf`. Razlika se ogleda u tome da `code` makro ne ispisuje prosleđene argumente na standardni izlaz, već u datoteku `output.asm`, u kojoj će se nakon kompajliranja nalaziti izgenerisani asemblerski program.

3. Hipotetski asemblerski jezik

Hipotetski asemblerski jezik nije asemblerski jezik koji se može direktno izvršavati na računaru, već se ovakvi programi izvršavaju u simulatoru—Hipsim.

Da bismo mogli da napravimo odgovarajuće asemblerske programe, potrebno je prethodno upoznati se sa registrima koje Hipsim poseduje.

Registar	**Namena**
`%0`, `%1`, ..., `%12`	Registri opšte namene

Registar	Namena
%13	Registar u kojem se čuva povratna vrednost funkcije
%14	Pokazivač na stek frejm
%15	Pokazivač na vrh steka

Svi registri i memorijske lokacije zauzimaju po 4 bajta.

3.1. Registri opšte namene

Registre opšte namene ćemo koristiti za čuvanje međurezultata u izrazu.

Na primer, posmatrajmo sledeći miniC iskaz dodele:

```
a = b + 5 + c - d;
```

Jedan iskaz dodele u miniC jeziku može biti proizvoljne dužine, odnosno može postojati proizvoljan broj sabiraka sa desne strane jednakosti.

Sa druge strane, asemblerske naredbe ADD i SUB prihvataju tačno 3 operanda:

```
ADDx op1, op2, op3 ①
SUBx op1, op2, op3 ②
```

① Naredba ADD radi op1 + op2 i rezultat smešta u op3.

② Naredba SUB radi op1 - op2 i rezultat smešta u op3.

Sufiks x treba da bude ili slovo S ili slovo U, ako su operandi označene odnosno neoznačene vrednosti, respektivno.

Prema tome, ekvivalent gore navedenom miniC iskazu će biti čitav niz asemblerskih naredbi:

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

① Pod pretpostavkom da su svi brojevi tipa int, sve naredbe u ovom primeru imaju sufiks S. Prvo se sabira vrednost promenljive b sa konstantom 5. Konstante se zapisuju sa predznakom $. Prvi slobodan registar (u ovom primeru registar %0) koristimo za pamćenje međurezultata.

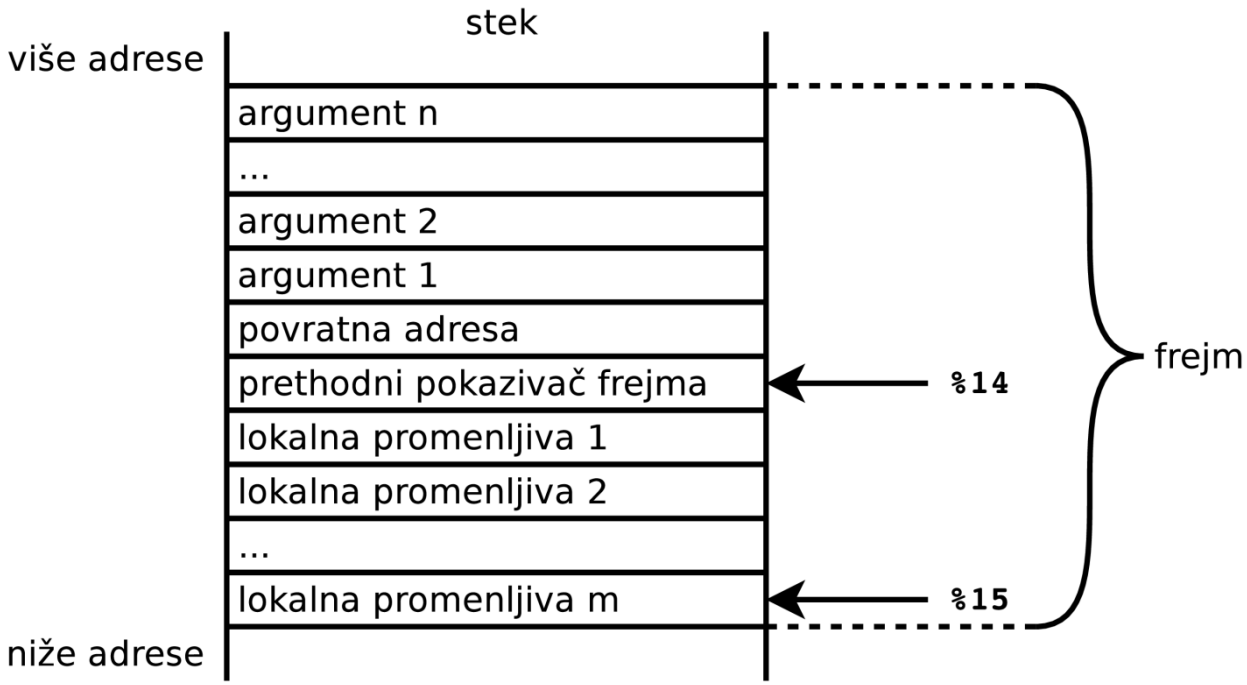
② Na prethodni međurezultat dodajemo vrednost promenljive c. U ovom trenutku, vrednost iz registra %0 je iskorišćena, pa je registar moguće osloboditi, jer više nije neophodan. Prema tome, prvi slobodan registar je opet registar %0, pa novi međurezultat takođe smeštamo u taj isti registar.

③ Ista situacija je i u ovom koraku. Preuzeta je vrednost iz registra `%0`, pa on ponovo postaje slobodan, što znači da međurezultat opet možemo sačuvati u taj isti registar.

④ Na kraju, finalni rezultat desne strane jednakosti je potrebno sačuvati u promenljivu `a` koja se nalazi sa leve strane jednakosti.

3.2. Stek i stek frejm

U primeru u prethodnoj sekciji, prikazano je da se lokalnim promenljivim `a`, `b`, `c` i `d` u asembleru pristupa tako što se samo navede njihov naziv. Ovo je urađeno da ne bismo komplikovali primer previše, međutim ovo nije tačno. Kao što vam je verovatno već poznato sa predmeta Arhitektura računara, lokalne promenljive i parametri funkcije se zapravo nalaze na steku.



Kao što je već rečeno, registar `%14` predstavlja pokazivač na stek frejm. Drugim rečima, registar `%14` je referentna tačka na osnovu koje pristupamo lokalnim promenljivim i parametrima.

Na slici iznad vidimo da se prva lokalna promenljiva nalazi na lokaciji odmah ispod referentne tačke. Prema tome, da bismo joj pristupili, potrebno je uzeti referentnu tačku i pomeriti se za `4` bajta niže, odnosno potrebno je napisati `-4(%14)`.

Analogno, druga lokalna promenljiva se u asemblerskom kodu zapisuje kao `-8(%14)`, treća kao `-12(%14)`, i tako dalje.

Parametri se nalaze na višim adresama, pa prvom parametru pristupamo kao `8(%14)`.

Podsetnik: U miniC jeziku funkcija može imati maksimalno 1 parametar.

4. Pomoćne funkcije

Kao što je već rečeno, generisanje koda se svodi na ispisivanje ekvivalentnog asemblerskog koda

upotrebom `code` makroa.

Apsolutno sve je moguće uraditi samo upotrebom ovog makroa, međutim, kako bi se olakšalo generisanje koda, postoje i neke pomoćne funkcije.

Zaglavlja pomoćnih funkcija su data u datoteci `codegen.h`:

```
int  take_reg(void); ①
void free_reg(void); ②
void free_if_reg(int reg_index);  ③

void gen_sym_name(int index); ④

void gen_cmp(int operand1_index, int operand2_index); ⑤

void gen_mov(int input_index, int output_index); ⑥
```

① Zauzima prvi slobodni registar i vraća njegov indeks u tabeli simbola.

② Oslobađa poslednji zauzeti registar.

③ Oslobađa registar čiji je indeks prosleđen.

④ Ispisuje simbol koji se nalazi na prosleđenom indeksu u tabeli simbola. Funkcija ispisuje simbol u odgovarajućem obliku, u zaivnosti od njegove vrste.

⑤ Generiše kompletnu `CMP` naredbu.

⑥ Generiše kompletnu `MOV` naredbu.

5. Kompajliranje, pokretanje i testiranje

Kao i do sada, rešenje se kompajlira i pokreće na sledeći način:

```
make
./micko < ulaz.mc
```

Ako nisu postojale greške u kodu, kompajler generiše asemblerski kod u datoteci `output.asm`. Dobijenu datoteku treba proslediti simulatoru:

```
./hipsim -r < output.asm
```

Simulator će izvršiti program i ispisati povratnu vrednost `main` funkcije.

Ukoliko se ne navede `-r` opcija prilikom poziva simulatora, pokrenuće se interaktivno izvršavanje programa, što može biti pogodno za debagovanje rešenja.

Takođe, moguće je pokrenuti sve testove automatski, upotrebom sledeće komande:

```
make test
```

Testovi treba na vrhu da sadrže komentar u sledećem obliku:

```
//RETURN: očekivana_vrednost ①
```

① `očekivana_vrednost` predstavlja vrednost koju ispravno rešenje treba da vrati iz `main` funkcije.

Ukoliko testovi sadrže ovakve komentare, `make test` će automatski pokrenuti simulator za svaki test i uporediti povratnu vrednost izvršenog programa sa očekivanom vrednosti. Ukoliko se rezultati ne poklapaju, test ne prolazi i prikazuje se u crvenoj boji.

6. Zadaci

6.1. Zadatak 1: globalne promenljive

Proširiti gramatiku globalnim promenljivim i napraviti generisanje koda za njih.

Primer deklaracije:

```
int a;
```

Izgenerisani kod za datu deklaraciju:

```
a:
     WORD    1
```

Ako je a globalna promenljiva, tada za iskaz dodele:

```
a = a + 1;
```

treba da bude izgenerisan sledeći kod:

```
ADDS    a,$1,%0
MOV     %0,a
```

Realizovati semantičku proveru:

1. Promenljiva ne sme biti prethodno deklarisana.

POMOĆ: Globalni i lokalni identifikatori mogu imati isto ime, pa se u tabeli simbola moraju razlikovati.

6.2. Zadatak 2: postinkrement iskaz

Napraviti generisanje koda za postinkrement iskaz.

Primer iskaza:

```
  x++;
```

Primer izgenerisanog koda (za prvu lokalnu promenljivu):

```
ADDS    -4(%14),$1,-4(%14)
```

Primer izgenerisanog koda (za globalnu promenljivu):

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

Realizovati semantičku proveru:

1. Postinkrement operator može da se primeni samo na promenljive (lokalne i globalne) i parametre.

6.3. Zadatak 3. postinkrement izraz

Napraviti generisanje koda za postinkrement unutar numeričkih izraza.

Primer:

```
int main() {
    int x;
    int y;
    x = 3;
    y = x++ + x++ + 42;
    return x + y;
}
```

Izlazni kod treba da proizvede rezultat 54. Generisanje operacije za inkrement treba da bude nakon obrade kompletnog numeričkog izraza.

Izgenerisani kod za:

```
    y = x++ + x++ + 42;
```

```
ADDS    -4(%14),-4(%14),%0  //num_exp
ADDS    %0,$42,%0           //num_exp
ADDS    -4(%14),$1,-4(%14)  //++
ADDS    -4(%14),$1,-4(%14)  //++
MOV      %0,-8(%14)         //assign
```

Realizovati semantičku proveru:

1. Postinkrement operator može da se primeni samo na promenljive (lokalne i globalne) i parametre.

7. Napomena za rešavanje zadataka

Svi zadaci se rešavaju sledećim redosledom:

- Dodati nove tokene na vrh `.y` datoteke.
- Definisati regularne izraze u `.l` datoteci za nove tokene.
- Proširiti gramatiku jezika tako da sintaksno podržava novu konstrukciju.
- Dodati semantičke provere.
- Osmisliti, za 1 konkretan primer, kako ekvivalentan asemblerski kod treba da izgleda.
- Uopštiti asemblerski kod iz prethodnog koraka i implementirati generisanje koda.

8. Projekat

U dokumentu [https://docs.google.com/document/d/e/2PACX-1vTh-HFm3Ujw2s6Oa_bWZGeYQM87NHUpS2L90rq72DQd0QigfpDDax0-Rh01PLNKKiwx7iGQUV3Ouecu/pub] se nalaze osnovne informacije vezane za projekat i zajednički zadaci koje je potrebno da svi implementiraju. Svake nedelje dokument će biti dopunjavan sa obaveznim delovima koje je potrebno implementirati.