kompajler **micko**

- direktorijum `code-gen`
  - = implementacija generisanja koda za miniC programski jezik
  - kompajler prevodi sa miniC jezika na hipotetski asemblerski jezik (HAJ)
- **`make`**
  - će napraviti program **`micko`**
  - **`$./micko <test.mc`**
    - izlaz će biti datoteka **`output.asm`**
    - koja sadrži izgenerisan kod na HAJ
- **`make test`**
  - izlaz će biti datoteke **`test-ok1.asm, test-ok2.asm, ...`**
    - pod uslovom da je kompajliranje prošlo bez grešaka

simulator **hipsim**

- primer upotrebe
  - **`$./hipsim < output.asm`**
    - **Any key** → step by step
    - **Ctrl + C** → exit
  - prozor u kome se simulator izvršava, ako nije dovoljno visok, treba povećati da se vidi sve što je prikazano
  - **`$./hipsim -r < output.asm`**
    - izvršava ceo kod i samo prikazuje povratnu vrednost
  - ako se u test-ok.mc fajl doda linija

    `//RETURN: 123`
    - tada će “make test“ pokrenuti i simulator i uporediti dobijenu vrednost sa onom zadatom u RETURN (u ovom primeru, sa 123)

implementacija generisanja koda

- generisanje koda znači:
  - kada se prepoznaju iskazi miniC koda,
  - treba ih na drugaciji nacin (na drugom jeziku) zapisati u izlazni fajl
- generisanje koda se implementira u parseru
  - kao akcija C koda
  - **`code("MOV %d...",x);`**
    - upisuje zadati tekst u izlazni fajl
    - sintaksa ista kao za C-ov printf

implementacija generisanja koda

- postoje više gotovih funkcija za generisanje koda
  - **codegen.c i .h**
    - `gen_cmp(op1_idx, op2_idx);`
    - `gen_mov(in_idx, out_idx);`
    -
    - `gen_sym_name(int index);`
    -
    - `take_reg(void);`
    - `free_reg(void);`
    - `free_if_reg(idx);`

primer generisanja koda

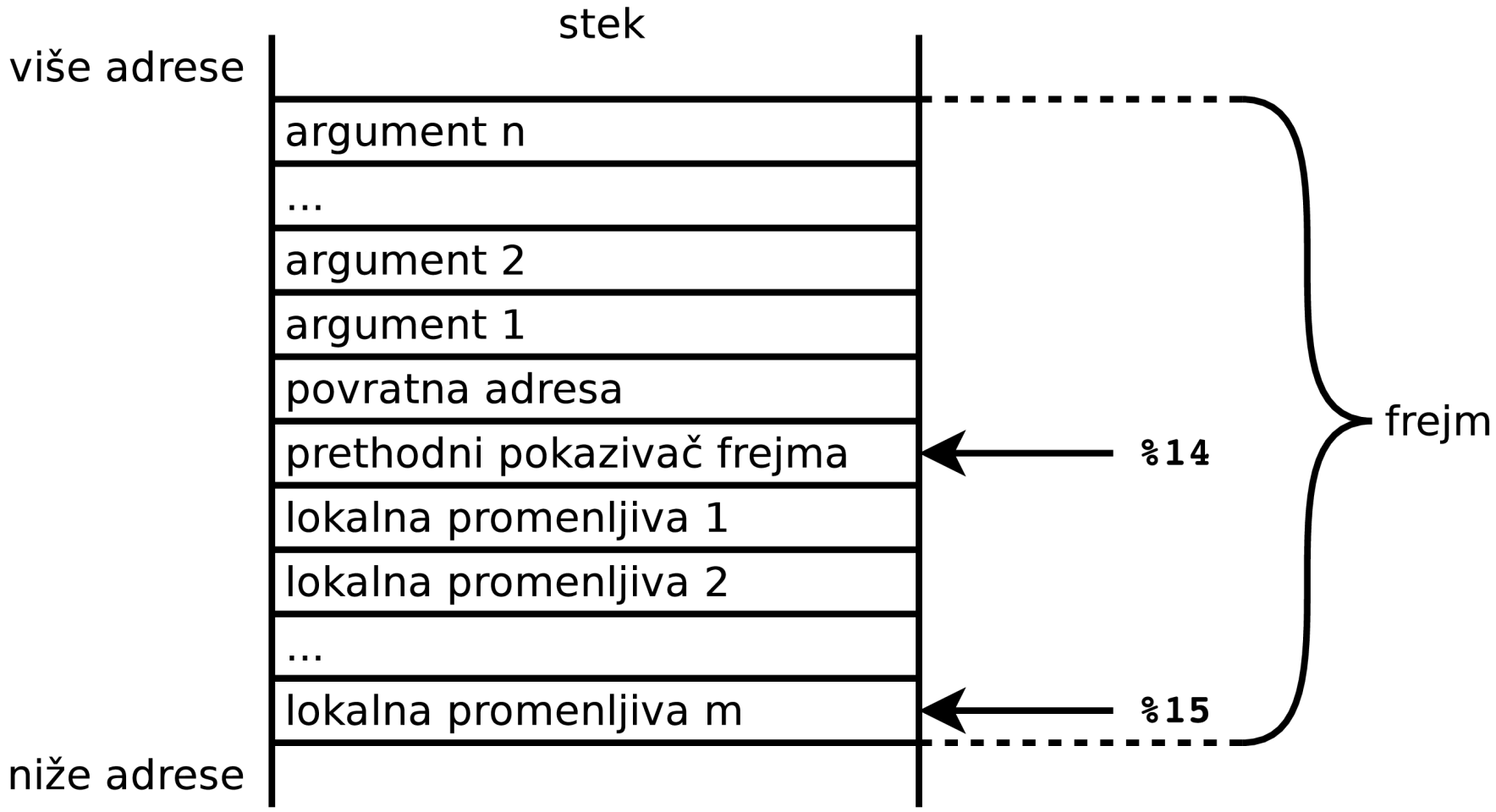
- primer miniC koda iz **test.mc**

```
int main() {
  int x;
  int y;

  y = 5;
  x = y - 2;

  return x;
}
```

- **output.asm**

```
main:
  PUSH    %14
  MOV     %15,%14
  SUBS    %15,$8,%15
@main_body:
  MOV     $5,-8(%14)
  SUBS    -8(%14),$2,%0
  MOV     %0,-4(%14)
  MOV     -4(%14),%13
  JMP     @main_exit
@main_exit:
  MOV     %14,%15
  POP     %14
  RET
```

primer generisanja koda



zadatak - globalne promenljive

- globalne promenljive se mogu pojaviti u bloku pre funkcija

```
int x;
unsigned y;
int main() {
  ...
}
```

- deklaracija globalne promenljive izgleda isto kao i deklaracija lokalne promenjive
  - `int x;`

zadatak - globalne promenljive

- globalni i lokalni identifikatori mogu imati isto ime
  - treba ih razlikovati u TS
    - **`kind`** polje
- podsetnik: zauzimanje prostora za lokalne promenljive
  - na stek frejmu (*runtime*)
  - 1 int/unsigned promenljiva = 1 element steka
- zauzimanje prostora za globalne promenljive
  - može se uraditi u toku kompajliranja
  - direktiva za zauzimanje memorije u HAJ je
    - **`WORD n`**
    - n - broj lokacja koje se zauzimaju
    - 1 int/unsigned promenljiva = 1 lokacija

zadatak - globalne promenljive

```
int  a;

unsigned b;

int main(int m)
{

    int x;

    x = m + a;

    return x;
}
```

```
a:
        WORD 1
b:
        WORD 1
main:
        PUSH   %14
        MOV    %15,%14
        SUBU %15,$4,%15
@main_body:
        ADDS   8(%14),a,%0
        MOV    %0,-4(%14)
        MOV    -4(%14),%13
        JMP    @main_exit
@main_exit:
        MOV    %14,%15
        POP    %14
        RET
```

testiranje zadataka

- za korektne miniC programe generisani HAJ kod mora biti sintaksno ispravan
  - simulator prilikom parsiranja proverava sintaksu
- generisani kod se mora izvršavati tako da daje ispravan rezultat
  - na primer, za sledeći kod:

```
int main() {
   int x;
   int y;
   x = 100 - 7;
   y = x - 50 + 42;
   return y;
}
```

  - se očekuje da izlazni kod nakon izvršavanja bude 85