

kompajler **micko**

- direktorijum `code-gen`
 - = implementacija generisanja koda za miniC programski jezik
 - kompajler prevodi sa miniC 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
- ili ako se uradi **make test**
 - izlaz će biti datoteke **test-ok1.asm, test-ok2.asm, ...**
 - pod uslovom da je kompajliranje prošlo bez grešaka

primer generisanja koda

□ primer miniC koda
`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
```

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);`

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 promenljive

- `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 lokacija koje se zauzimaju
 - 1 int/unsigned promenljiva = 1 lokacija

zadatak - globalne promenljive

<code>int a;</code>	<code>a:</code>	<code>WORD 1</code>
<code>unsigned b;</code>	<code>b:</code>	<code>WORD 1</code>
<code>int main(int m)</code> <code>{</code>	<code>main:</code>	<code>PUSH %14</code> <code>MOV %15,%14</code> <code>SUBU %15,\$4,%15</code>
<code>int x;</code>	<code>@main_body:</code>	<code>ADDS 8(%14),a,%0</code> <code>MOV %0,-4(%14)</code> <code>MOV -4(%14),%13</code> <code>JMP @main_exit</code>
<code>x = m + a;</code>	<code>@main_exit:</code>	<code>MOV %14,%15</code> <code>POP %14</code> <code>RET</code>
<code>return x;</code>		
<code>}</code>		

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