Sistemski softver - Domaći zadatak - Asembler

Miloš Aćimović 2014/0146 RTI

am140146d@student.etf.bg.ac.rs
Elektrotehnički fakultet,
Univerzitet u Beogradu

31. maj 2017

Sadržaj

1 Opis projekta

Domaći zadatak ima za cilj realizaciju dvoprolaznog asemblera i emulatora za školsku arhitekturu. Sintaksa asemblera opisana je u prilogu. Dodatni zahtevi i napomene navedeni su u nastavku:

- u jednoj liniji može biti najviše jedna komanda,
- labela može da stoji i u praznoj liniji i tada je njena vrednost jednaka adresi prve sledeće instrukcije,
- simboli mogu da se uvezu ili izvezu direktivom .global. Ukoliko je simbol definisan, izvozi se. U suprotnom se uvozi. Direktiva se može navesti bilo gdje u ualznom kodu. Primer: .global <ime globalnog simbola>,...- u jednoj direktivi može da se navede i više globalnih naziva koji su odvojeni zapetama,
- fajl sa izvornim kodom se završava direktivom .end. Ostatak fajla se odbacuje (ne prevodi se),
- tipovi adresiranja označeni su u skladu sa opisom u prilogu,
- pored opisanih tipova adresiranja, u instrukciji može da se pojavi i oznaka koja počinje znakom $. Tada se očekuje da se navedenoj lokaciji pristupi PC relativnim adresiranjem (u mašinskoj instrukciji se generiše registarsko indirektno adresiranje, pri čemu se koristi registar PC i odgovarajući pomeraj). Posle znaka dolar može biti navedena adresa u numeričkom obliku ili labela. Ovakav tip adresiranja može da se pojavi u svim instrukcijama koje podržavaju registarsko indirektno adresiranje sa pomerajem
- ORG direktiva može da se navede samo pred početak sekcije i tada se njome specificira početna adresa naredne sekcije (asembler ovu vrednost koristi kao početnu adresu sekcije)
- sve upotrebe simbola koje asembler može u potpunosti da razreši, asembler razreši i ne ostavlja zapise o relokaciji
- simboli u izrazima mogu da se pojave samo u operacijama sabiranja i oduzimanja (samo se dodaju ili oduzimaju od ostatka izraza), osim u slučaju kada se razlika dva izraza pojavi u zagradama čime se dobija konstanta koja dalje može da se koristi u izrazu proizvoljnog oblika

2 Opis rešenja

2.1 Asembler

Navešću prvo algoritam asembliranja korišćen u rešenju. Asembler čita izvorni fajl liniju po liniju. I u prvom i u drugom prolazu se koristi promenljiva location counter koja ukazuje na adresu unutar tekuće sekcije. Na početku oba prolaza ona je inicijalizovana na 0 i pri nailasku na instrukciju uvećava se za veličinu instrukcije, a ukoliko je direktiva za definisanje podataka uvećava se za 1, 2, ili 4 u zavisnosti da li je direktiva *DB*, *DW* ili *DD*, respektivno. Ukoliko se u direktivi za definisanje podataka nalazi i izraz praćen rečju *DUP* onda asembler u prvom prolazu izračunava broj ponavljanja i uvećava *location_ counter* za odgovarajuću vrednost. Ukoliko linija sadrži labelu ta labela se dodaje u tabelu simbola(više o tabeli simbola u nastavku). Dalje se tumači sledeća reč i razdvaja se slučaj kada se radi o direktivi i kada se radi o instrukciji. Po nailasku definicije sekcije resetuje se *location_ counter* i pre toga se postavi veličina prethodno definisane sekcije(ako je ima) na njegovu vrednost. Pošto se u prvom prolazu može u *ORG* direktivi, tj. njenom izrazu može naći referenca unapred na simbol definisan *DEF* direktivom, obrada ove direktive je ostavljena za drugi prolaz. Kada se u drugom prolazu naiđe na direktivu ORG sračuna se njen izraz i sekciji koja joj sledi se postavi adresa na rezultat *ORG* izraza, i potom se prođe kroz tabelu simbola i svim simbolima tekuće sekcije se uveća vrednost za vrednost *ORG* izraza. Veličina instrukcije se zaključuje na osnovu načina adresiranja. Ukoliko se radi memorijskom direktnom, neposrednom ili registarsko indirektnom načinu adresiranja veličina instrukcije je 8B, inače 4B. Ukoliko se dogodi greška, bilo u prvom ili drugom prolazu asemblera ta greška se ispisuje ili na standardni izlaz ili u fajl. Rezultat prvog prolaza jeste međufajl koji sadrži sve instrukcije i direktive iz izvornog fajla osim direktive *.global* i direktive *DEF* koje su obrađene u prvom prolazu(u posebnom slučaju direktiva *DEF* je obrađena između dva prolaza prolaskom kroz tabelu neizračunljivih simbola(*DEF*)). Međufajl takođe ne sadrži labele i operandi u instrukcijama su razdvojeni znakom razmaka. Greške prijavljene u drugom prolazu se prijavljuju u odnosu na linije u međufajlu.

2.2 Uputstvo za prevođenje i pokretanje asemblera

Asembler se prevodi komandom:

```
make
```

Asmebler se pokreće:

```
./assembler putanja_do_fajla/naziv_izvornog_fajla.s
```

Ukoliko se želi da se prijave greške ili uspešnog prevođenja ispisuje u poseban fajl to se navodi na sledeći način.

```
./assembler naziv_izvornog_fajla.s -log naziv_log_fajla.txt
```

Primer pokretanja asemblera bi bio(za izvorni program u istom direktorijumu kao i asembler):

```
./assembler sample.s
```

ili

```
./assembler sample.s -log log.txt
```

2.3 Tabela simbola

Za tabelu simbola je uzeta struktura podataka *niz vektora simbola*. Izvršeno je preslikavanje mogućih početnih karaktera simbola na indekse niza, pa se u jednom bucketu(jednom vektoru) nalaze svi simboli koji počinju istim karakterom. Bucket-a ima 54. Prvih 26 odgovaraju malim slovima alfabeta a drugih 26 odgovara velikim slovima. Sledeća dva odgovaraju donjoj crti i tački kojom počinju imena sekcija. Ovakvom struktrom je u određenim slučajevima smanjen prostor pretrage a time i ubrzana pretraga(npr. pri pretraživanju sekcija dovoljno je proći kroz niz simbola koji predstavljaju sekcije(počinju tačkom)). Pošto je operacija pretraživanja tabele česta, ovo čini ovu strukturom pogodnom u vremenskom pogledu, po cenu neiskorišćenog prostora za bucket koji možda nema simbole. Za niz je iskorišćena C++ STL struktura vector. Simbol je modeliran sledećom C-ovskom strukturom.

```
typedef struct Symbol{
    uint32_t num;
    char *name;
    int32_t sec_num;
    uint32_t addr;
    char flag;
    uint32_t sec_size;
    std::vector<char> flags;
    Symbol(char* n, uint32_t address): num(0), name(n), sec_num(0),
    addr(address), flag('L'), sec_size(0), flags(0){}
} Symbol;
```

- *num* predstavlja broj simbola u tabeli simbola;
- *name* predstavlje ime simbola;
- *sec_num* broj sekcije u kojoj je simbol definisan;
- *addr* vrednost simbola;
- *flag* označava da li je simbol lokalan ili globalan

sledeća polja su relevantna za sekcije:

- *sec_size* veličina sekcije
- *flags* niz karaktera
  - W - sekcija je dozvoljena za upis i čitanje,
  - A - za sekciju se odvaja prostor,
  - P - sekcija je prisutna i za nju je generisan sadržaj,
  - X - sekcija se može izvršavati,
  - F - pozicija sekcije unutar emulirane memorije je fiksna.

2.4 Tabela neizračunljivih simbola

Pošto se može dogoditi da se preko jedne simboličke konstante definiše druga i njihovo razrešavanje bi zahtevali više prolaza, ukoliko ne može da se u *DEF* direktivi razreši simbol izračunavanjem izraza onda se dati simbol koji se definiše kao i njegov izraz dodaju u TNS(Tabela Neizračunljivih Simbola) kako bi se kraju prvog prolaza, većim brojem prolaza kroz ovu tabelu razrešile vrednosti svih neizračunljivih simbola i time ušlo sa svima simbolima razrešenim u drugi prolaz. Dodavanje u TNS koja je organizovana kao ulančana lista se radi preko sledeće strukture:

```
typedef struct TNSymbol {
        struct TNSymbol* next;
        char* def_sym;
        char* expr;
        TNSymbol(char* n, char* ex) : def_sym(n), expr(ex), next(NULL){}
}TNSymbol;
```

- *next* polje koje predstavlja pokazivač na sledeći element liste
- *def_ sym* polje koje predstavlja simbol koji se definiše
- *next* polje koje predstavlja izraz preko kojeg se simbol definiše

Potpis funkcije za razrešenje vrednosti simbola iz TNS:

```
void resolve_TNS(TNSymbol** tns, std::vector<std::vector<Symbol*>>& symtbl);
```

2.5 Tumačenje direktive

Sve se obrađuju u prvom i drugom prolazi, sa izuzetkom direktiva *.global* i *DEF* koje se obrađuju samo u prvom prolazu i direktiva *ORG* koja se obrađuje samo u drugom prolazu. Potpisi funkcija koje vrše obradu direktive u prvom i drugom prolazu respektivno:

```
void translate_directive_pass_one(const char* name, char* buf,
FILE* output, std::vector<std::vector<Symbol*>>& symtbl,uint32_t& location_counter);

void translate_directive_pass_two(const char* name, char* buf,
std::vector<std::vector<Symbol*>>& symtbl, std::vector<RelTable*>& reltbls,
uint32_t& location_counter, int& rel_ind, char* prev_name);
```

2.6 Tumačenje instrukcije

Sledeća funkcija određuje o kom načinu adresiranja se radi u instrukciji i na osnovu toga uvećava *location_counter* za odgovarajuću vrednost:

```
void location_counter_update(char** args, int num_args, uint32_t& location_counter);
```

Ova funkcija određuje o kojoj se funkciji radi na osnovu mnemonika i poziva odgovarajuću funkciju za obradu instrukcije:

```
int translate_inst(const char* name, char** args, int num_args,
std::vector<std::vector<Symbol*>>& symtbl, std::vector<RelTable*>& reltbls,
 int rel_ind, uint32_t& location_counter);

int trans_ar_log(uint8_t opcode, char** args, int num_args, std::vector<RelTable*>& reltbls,
int rel_ind, uint32_t& location_counter);

int trans_not(uint8_t opcode, char** args, int num_args, std::vector<RelTable*>& reltbls,
int rel_ind, uint32_t& location_counter);

int trans_stack(uint8_t opcode, char** args, int num_args, std::vector<RelTable*>& reltbls,
int rel_ind, uint32_t& location_counter);

int trans_int(uint8_t opcode, char** args, int num_args, std::vector<RelTable*>& reltbls,
int rel_ind, uint32_t& location_counter);

int trans_ret(uint8_t opcode, char** args, int num_args, std::vector<RelTable*>& reltbls,
 int rel_ind, uint32_t& location_counter);

int trans_uncond_branch(uint8_t opcode, char** args, int num_args,
std::vector<std::vector<Symbol*>>& symtbl, std::vector<RelTable*>& reltbls,
 int rel_ind, uint32_t& location_counter);

int trans_cond_branch(uint8_t opcode, char** args, int num_args,
std::vector<std::vector<Symbol*>>& symtbl, std::vector<RelTable*>& reltbls,
 int rel_ind, uint32_t& location_counter);

int trans_load(uint8_t opcode,const char* name, char** args, int num_args,
std::vector<std::vector<Symbol*>>& symtbl, std::vector<RelTable*>& reltbls,
int rel_ind, uint32_t& location_counter);

int trans_store(uint8_t opcode,const char* name, char** args, int num_args,
std::vector<std::vector<Symbol*>>& symtbl, std::vector<RelTable*>& reltbls,
 int rel_ind, uint32_t& location_counter);
```

Nakon određenog načina adresiranja operanada poziva se odgovarajuća varijanta ove funkcije koja izračunava izraz unutar operanda i pravi podizraz i proverava njegovu korektnost pozivom funkcije *check_sub*. Prva funkcija se poziva za varijantu registarskog indirektnog a druga u slučajevima memorijskog direktnog, neposrednog, registarkog indirektnog sa pomerajem i PC relativnog adresiranja. Povratna vrednost funkcije je vrednost druge duple reči instrukcije. Parametar *sub_res* prosleđen po referenci nakon poziva sadrži rezultat podizraza.

```
int32_t resolve_addr_mode(char* arg, std::vector<int32_t>& literals, int32_t tmp,
bool& ret, uint32_t& sym_num, int& r, std::vector<std::vector<Symbol*>>& symtbl, int32_t& sub_res);

int32_t resolve_addr_mode(char* arg, std::vector<int32_t>& literals, int32_t tmp,
 bool& ret, uint32_t& sym_num, std::vector<std::vector<Symbol*>>& symtbl, int32_t& sub_res);
```

2.7 Izračunavanje izraza

Pošto se u izrazu mogu naći simboli(apsolutni ili relativni) od početnog izraza se formira *podizraz* koji ima ulogu provere korektnosti izraza. Podizraz se formira na sledeći način: simboli(koji imaju format labele) se menjaju brojem 1 a literali se menjaju brojem 0, i ukoliko po izračunavanju izraza se dobije 1 ili 0 izraz se smatra korektnim, u suprotnom izraz nije korektan. Ukoliko je izraz korektan prebacuje se postfiksnu notaciju i izračunava se njegova vrednost. Potpis funkcije za proveravanje korektnosti podizraza:

```
int32_t check_sub(char* expr, std::vector<Symbol*>& symbols, bool& ret, uint32_t& sym_num);
```

3 Testovi

Prvi test. Prevođenje iz foldera *bin*:

```
./assembler ../tests/test1.s
```

```
ORG 0x0
.data
DD sp ; stek inicijalna vrednost
DD 2 DUP ? ; prvi i drugi ulaz su trenutno prazni
DD fault ; sadrzi adresu funkcije handler greske
DD timer ; sadrzi adresu timer funkcije
DD keypress ; sadrzi adresu handler pritisnutog tastera
DD 36 DUP ?
.text.0
out_reg DEF 0xA0
in_reg DEF 0xA4
.global START
START:
LOAD R0, #0 ; ucitava 0 u registar R0
x: JZ R0,x ; apsolutni skok na lokaciju x
.text.1
fault:
LOADUB R2, #'g'
STOREB R2, out_reg
LOADUB R2, #'r'
STOREB R2, out_reg
LOADUB R2, #'e'
STOREB R2, out_reg
LOADUB R2, #'s'
STOREB R2, out_reg
LOADUB R2, #'k'
STOREB R2, out_reg
LOADUB R2, #'a'
STOREB R2, out_reg
LOADUB R2, #'\n'
STOREB R2, out_reg
INT R0
.text.2
keypress:
PUSH R2
LOADUB R2, #'p'
STOREB R2, out_reg
LOADUB R2, #'r'
STOREB R2, out_reg
```

```
LOADUB R2, #'i'
STOREB R2, out_reg
LOADUB R2, #'t'
STOREB R2, out_reg
LOADUB R2, #'i'
STOREB R2, out_reg
LOADUB R2, #'s'
STOREB R2, out_reg
LOADUB R2, #'n'
STOREB R2, out_reg
LOADUB R2, #'u'
STOREB R2, out_reg
LOADUB R2, #'t'
STOREB R2, out_reg
LOADUB R2, #' '
STOREB R2, out_reg
LOADUB R2, in_reg
STOREB R2, out_reg
LOADUB R2, #'\n'
STOREB R2, out_reg
POP R2
RET
.text.3
timer:
PUSH R2
LOADUB R2, #'t'
STOREB R2, out_reg
LOADUB R2, #'i'
STOREB R2, out_reg
LOADUB R2, #'m'
STOREB R2, out_reg
LOADUB R2, #'e'
STOREB R2, out_reg
LOADUB R2, #'r'
STOREB R2, out_reg
LOADUB R2, #'\n'
STOREB R2, out_reg
POP R2
RET
RET
.bss
sp:
DB 0x1000 DUP ?
.end
```

Izlaz asembliranja:

```
Assembly of the source file completed successfully.
```

```
#TabelaSimbola
SEG 1 .data 1 0x00000000 0x000000a8 AWPF
SEG 2 .text.0 2 0x00000000 0x00000010 AXP
SEG 3 .text.1 3 0x00000000 0x00000074 AXP
SEG 4 .text.2 4 0x00000000 0x000000cc AXP
SEG 5 .text.3 5 0x00000000 0x00000070 AXP
SEG 6 .bss 6 0x00000000 0x00001000 AW
SYM 7 fault 3 0x00000000 L
SYM 8 in_reg -1 0x000000a4 L
SYM 9 keypress 4 0x00000000 L
SYM 10 out_reg -1 0x000000a0 L
SYM 11 sp 6 0x00000000 L
SYM 12 timer 5 0x00000000 L
SYM 13 x 2 0x00000008 L
SYM 14 START 2 0x00000000 G
#rel.data
0x00000000 A 6
0x0000000c A 3
0x00000010 A 5
0x00000014 A 4
.data
00 00 00 00 89 7f 00 00 78 0b 82 e7 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
#rel.text.0
0x0000000c A 2
.text.0
00 00 80 10 00 00 00 00 00 00 c0 04 08 00 00 00
#rel.text.1
.text.1
18 10 80 10 67 00 00 00 18 10 c0 11 a0 00 00 00
18 10 80 10 72 00 00 00 18 10 c0 11 a0 00 00 00
18 10 80 10 65 00 00 00 18 10 c0 11 a0 00 00 00
18 10 80 10 73 00 00 00 18 10 c0 11 a0 00 00 00
18 10 80 10 6b 00 00 00 18 10 c0 11 a0 00 00 00
18 10 80 10 61 00 00 00 18 10 c0 11 a0 00 00 00
18 10 80 10 0a 00 00 00 18 10 c0 11 a0 00 00 00
00 00 00 00
#rel.text.2
.text.2
00 00 02 20 18 10 80 10 70 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 72 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 69 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 74 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 69 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 73 00 00 00 18 10 c0 11
```

```
a0 00 00 00 18 10 80 10 6e 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 75 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 74 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 20 00 00 00 18 10 c0 11
a0 00 00 00 18 10 c0 10 a4 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 0a 00 00 00 18 10 c0 11
a0 00 00 00 00 00 02 21 00 00 00 01
#rel.text.3
.text.3
00 00 02 20 18 10 80 10 74 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 69 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 6d 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 65 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 72 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 0a 00 00 00 18 10 c0 11
a0 00 00 00 00 00 02 21 00 00 00 01 00 00 00 01
#end
```

Drugi test. Prevođenje iz foldera *bin*:

```
./assembler ../tests/test2.s
```

```
ORG 0x0
.data
DD sp ; stek inicijalna vrednost
DD 2 DUP ? ; prvi i drugi ulaz su trenutno prazni
DD fault ; sadrzi adresu funkcije handler greske
DD timer ; sadrzi adresu timer funkcije
DD keypress ; sadrzi adresu handler pritisnutog tastera
DD 36 DUP ?
.text.0
out_reg DEF 0xA0
.global START
START:
LOADUB R0, #'1'
LOAD R4, #10
PUSH R4
PUSH R0
CALL sub ;rezultat je u R0
POP R10
POP R10
l: JZ R3, \$l
.text.1
fault:
LOAD R0, #0
INT R0
.text.2
keypress:
RET
.text.3
timer:
PUSH R2
LOADUB R2, #'t'
STOREB R2, out_reg
LOADUB R2, #'i'
STOREB R2, out_reg
LOADUB R2, #'m'
STOREB R2, out_reg
LOADUB R2, #'e'
STOREB R2, out_reg
LOADUB R2, #'r'
STOREB R2, out_reg
LOADUB R2, #'\n'
STOREB R2, out_reg
POP R2
RET
.text.4 ; potprogram za neko izracunavanje
sub:
PUSH R13
LOAD R13, SP
LOAD R3, #1
LOAD R0, [R13 -11] ; R0 = '1'
LOAD R1, [R13 - 15] ; R1 = '10'
```

```
loop:
STOREB R0, out_reg
ADD R0, R0, R3
SUB R1, R1, R3
JNZ R1, loop
POP R13
RET
.bss
sp:
DB 0x1000 DUP ?
.end
```

Izlaz asembliranja:

```
Assembly of the source file completed successfully.
```

```
#TabelaSimbola
SEG 1 .data 1 0x00000000 0x000000a8 AWPF
SEG 2 .text.0 2 0x00000000 0x00000030 AXP
SEG 3 .text.1 3 0x00000000 0x0000000c AXP
SEG 4 .text.2 4 0x00000000 0x00000004 AXP
SEG 5 .text.3 5 0x00000000 0x0000006c AXP
SEG 6 .text.4 6 0x00000000 0x00000040 AXP
SEG 7 .bss 7 0x00000000 0x00001000 AW
SYM 8 fault 3 0x00000000 L
SYM 9 keypress 4 0x00000000 L
SYM 10 l 2 0x00000028 L
SYM 11 loop 6 0x00000020 L
SYM 12 out_reg -1 0x000000a0 L
SYM 13 sub 6 0x00000000 L
SYM 14 sp 7 0x00000000 L
SYM 15 timer 5 0x00000000 L
SYM 16 START 2 0x00000000 G
#rel.data
0x00000000 A 7
0x0000000c A 3
0x00000010 A 5
0x00000014 A 4
.data
00 00 00 00 d5 7f 00 00 78 eb 81 c0 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
#rel.text.0
0x0000001c A 6
0x0000002c R 2
.text.0
18 00 80 10 31 00 00 00 00 20 80 10 0a 00 00 00
00 00 04 20 00 00 00 20 00 00 c0 03 00 00 00 00
00 00 0a 21 00 00 0a 21 00 18 f1 04 24 00 00 00
#rel.text.1
.text.1
00 00 80 10 00 00 00 00 00 00 00 00
#rel.text.2
.text.2
00 00 00 01
#rel.text.3
.text.3
00 00 02 20 18 10 80 10 74 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 69 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 6d 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 65 00 00 00 18 10 c0 11
a0 00 00 00 18 10 80 10 72 00 00 00 18 10 c0 11
```

```
a0 00 00 00 18 10 80 10 0a 00 00 00 18 10 c0 11
a0 00 00 00 00 00 02 21 00 00 00 01
#rel.text.4
0x00000034 A 6
.text.4
00 00 0d 20 00 68 10 10 00 18 80 10 01 00 00 00
00 00 ed 10 f5 ff ff ff 00 08 ed 10 f1 ff ff ff
18 00 c0 11 a0 00 00 00 c0 00 00 30 c0 08 01 31
00 08 c0 05 20 00 00 00 00 00 0d 21 00 00 00 01
#end
```

Treći test. Prevođenje iz foldera *bin*:

```
./assembler ../tests/test3.s
```

```
ORG 0x0
.data
DD 42 DUP ?
.text
out_reg DEF 0xA0
.global START ; program za testiranje TNS-a
A DEF B
B DEF C
C DEF D
D DEF 'A'
START:
LOAD R0, #A
LOAD R3, #26
LOAD R4, #1
loop:
STOREB R0, out_reg
ADD R0, R0, R4
SUB R3, R3, R4
JNZ R3, \$loop
INT R3
.end
```

Izlaz asembliranja:

```
Assembly of the source file completed successfully.
```

```
#TabelaSimbola
SEG 1 .data 1 0x00000000 0x000000a8 AWPF
SEG 2 .text 2 0x00000000 0x00000034 AXP
SYM 3 loop 2 0x00000018 L
SYM 4 out_reg -1 0x000000a0 L
SYM 5 A -1 0x00000041 L
SYM 6 B -1 0x00000041 L
SYM 7 C -1 0x00000041 L
SYM 8 D -1 0x00000041 L
SYM 9 START 2 0x00000000 G
#rel.data
.data
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
#rel.text
0x0000002c R 2
.text
00 00 80 10 41 00 00 00 00 18 80 10 1a 00 00 00
00 20 80 10 01 00 00 00 18 00 c0 11 a0 00 00 00
00 01 00 30 00 19 03 31 00 18 f1 05 14 00 00 00
00 00 03 00
#end
```

Četvrti test. Prevođenje iz foldera *bin*:

```
./assembler ../tests/testerror1.s
```

```
.text
ADD R0, R0, R1, R2 ; dodatni operandi
label:
OR R0, R1, R2
JZ R0
3hello: ;neispravna labela
ASL R0, R1, R2 ASR R1, R2, R3 ;vise instrukcija u jednom redu
label:
SUB R1, R1, R1  ;visestruka definicija labele
l1: l2: ; vise labela u jednoj liniji

ADD R17, R2, R3 ; nepostojeci registar
STORE R1, #0x123 ;ne sme neposredno
LOAD R1, 0x8000080800080 ; preveliki broj
```

Izlaz asembliranja:

```
Error: extra argument on line 2: R2
Error: invalid label on line 6: 3hello
Error: extra argument on line 7:  R3
Error: name label already exists.
Error: to few arguments.
Error: on line 3:
jz R0
Error: on line 5:
l2:
Error: register doesn't exist.
Error: on line 6:
add R17 R2 R3
Error: on line 7:
store R1 #0x123
One or more errors happend during assembly.
```

```
#TabelaSimbola
SEG 1 .text 1 0x00000000 0x00000028 AXP
SYM 2 label 1 0x00000000 L
SYM 3 l1 1 0x0000000c L
#rel.text
.text
80 08 00 36 40 08 01 31 00 08 c0 10 80 00 80 80
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
#end
```

Peti test. Prevođenje iz foldera *bin*:

```
./assembler ../tests/testerror2.s
```

```
.text
.global START
START: LOAD R0, #3
PUSH R0
CALL sub
POP R13
LOAD R0, #0
INT R0
.end
```

Izlaz asembliranja:

```
Assembly of the source file completed successfully.
```

```
#TabelaSimbola
SEG 1 .text 1 0x00000000 0x00000024 AXP
SYM 0 sub 0 0x00000000 L
SYM 2 START 1 0x00000000 G
#rel.text
0x00000010 A 0
.text
00 00 80 10 03 00 00 00 00 00 00 20 00 00 c0 03
00 00 00 00 00 00 0d 21 00 00 80 10 00 00 00 00
00 00 00 00
#end
```

Šesti test. Prevođenje iz foldera *bin*:

```
./assembler ../tests/testerror3.s
```

```
; overrlapping sections
ORG 0x0
.data
DD 300 DUP ?
ORG 0x30
.text
.global START
START: LOAD R0, #0
INT R0
```

Izlaz asembliranja:

```
Assembly of the source file completed successfully.
```

```
#TabelaSimbola
SEG 1 .data 1 0x00000000 0x000004b0 AWPF
SEG 2 .text 2 0x00000030 0x0000000c AXPF
SYM 3 START 2 0x00000030 G
#rel.data
.data
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 a1 d2 01 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 81 d2 01 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
#rel.text
.text
00 00 80 10 00 00 00 00 00 00 00 00
#end
```