Sistemsko programiranje

Mina Milosavljević 2014/441

Problem : Zadatak je napraviti dvoprolazni assembler ,ulaz je tekstualni fajl,koji se prevodi do linije .end,izlaz asemblera je fajl sa tabelom simbola,relokacionim tabelama I sadrzajem sekcija.

Rešenje:  
Za potrebe rešenja napravljene se klase Symbol,Section ,Relocation i Compiler.Klasa Symbol predstavlja svaki simbol u tabeli simbola.Njena polja su ime simbola,offset simbola,tip simbola,doseg,sekcija kojoj pripada i veličina simbola.

Klasa Compiler sadrži kao svoja polja tabelu simbola,tabelu realokacija i tabelu sekcija.Glavne metode klase Compiler su FirstRun ciji je zadatak obavljanje prvog prolaza asemblera i popunjavanje tabele simbola i metoda SecondPass ciji je zadatak obavljanje drugog prolaza i popunjavanje sekcija i relokacionih tabela.

Metoda FirstRun prihvata kao ulazni podatak ulazni fajl koji treba obraditi i koji sadrzi asemblerski kod.Prvi korak ove metode je parsiranje ulaza odnosno kreiranje vektora ciji su elementi vektori stringova.Svaki zaseban vektor predstavlja jedan red ulaznog fajla.Ovakav nacin obrade omogućava obradu ulaznog fajla red po red.

Nakon parsiranja ulaznog fajla započinje obrada.Prvo proveravamo da li se na početku reda nalazi labela.Ako da, onda se simbol koji predstavlja tu labelu uključuje u tabelu simbola.Nakon te provere prelazimo na switch naredbu koja ispituje naredni token.

Ako red koji obradjujemo započinje „.global“ u prvom prolazu nema obrade.Ako red započinje direktovom pozivamo metodu directiveFirstRun.DirectiveFirstRun treba za svaku odgovarajucu direktivu da definiše promenu Location Countera.Dozvoljeno je da se nakon .align direktiva nadje bilo koja decimalna vrednost i takodje moguce je jednom direktivom za podatke definisati vise podataka istovremeno.

Ako red koji obradjujemo zapocinje ključnom rečju koja definiše sekciju potrebno je kreirati nov objekat klase Section koji predstavlja sekciju u kojoj smo trenutno bili i uključiti ga u tabelu sekcija.Takodje u ovom trenutku ažuriramo veličinu sekcije koju smo upravo obradjivali.Postavljamo i pokazivač na novu sekciju kako bismo tokom obrade znali u kojoj se sekciji nalazimo i objekat nove sekcije uključujemo u tabelu simbola.

Ako red koji obradjujemo počinje nekom od instrukcija potrebno je Location Counter ažurirati zavisno od velicine instrukcije .Velicina instrukcije moze biti 2 bajta ili 4 bajta zavisno od operanada.Instrukcija je 2B ako nema operanada,ako ima jedan operand i on je registarsko direktno adresiran i ako ima 2 operanda i oba su registarsko direktno adresirani.U svakom drugom slucaju veličina instrukcije je 4 bajta.

Kada se svi redovi na predhodno opisan način obrade mozemo preći na drugi prolaz pozivom SecondPass metode.

SecondPass metoda isto kao i FirstRun metoda obradjuje ulazni fajl red po red.Ali je obrada sada drugačija.Kada naidjemo na labelu nema obrade.Ako je prvi token reda „.global“ simboli koji su navedeni nakon direktive mogu biti definisani u tabeli simbola ili ne .Ako su definisani označe se kao globalni a ako nisu definisani onda se unesu u tabelu i oznace kao globalni i nedefinisani.Kada u ulaznom fajlu naidjemo na direktive pozivamo metodu secondPassDirective ciji je zadatak da popuni pozicije u memoriji u skladu sa inicijalizovanim podacima u sekciji koju trenutno obradjujemo .Za .skip i .align direktive se sadrzaj memorije popunjava nulama.Memoriju mozemo inicijalizovati samo ako su podaci inicijalizovani bilo decimalnim operandima ili vrednošću definisanih simbola.Ako je prvi token reda sekcija potebno je samo izmeniti pokazivač koji ukazuje na sekciju koju trenutno obradjujemo.Pri svakom nailasku u kodu na novu sekciju Location Counter se postavlja na nulu.Ako je prvi token reda instrukcija pozivamo metodu secondPassInstruction.Ova metoda za cilj ima kodovanje instrukcije i njeno smeštanju u sadrzaj sekcije kojoj pripada.Obrada započinje ispitivanjem da li se instrukcije izvršava uslovno sto definiše prva 2 bita instrukcije.Zatim binarnom zapisu instrukcije dodajemo operacioni kod same instrukcije.Instrukcije se zasebno obradjuju zavisno od toga da li imaju nula,jedan ili dva operanda pa se u skladu sa tim se pozivaju odgovarajuce metode.Ako instrukcija ima manje od dva operanda odgovarajuci biti koji koduju taj operand se popunjavaju nulama.Za token operanda se prvo poziva metoda resolveAddresingTypeOperand ciji je zadatak da vrati informaciju o tipu adresiranja tog operanda kako bi znali koja su prva 2 bita koja koduju taj operand.Zatim se poziva metoda getValueOperand koja vraca informaciju o registru koji se koristi za taj operand,da li taj operand koristi dodatna dva bajta i ako ih koristi koja je njihova vrednost.U toj metodi se po potrebi kreiraju relokacione informacije pozivima metoda setAbsRelocation ako je u pitanju apsolutna realokacija i setPCrelRelocation ako je u pitanju PC relativna realokacija.PC relativna realokacija se definiše samo u slucaju da je adresiranje operanda $a gde je a neki simbol u tabeli simbola.

Nakon drugog prolaza kao podatke u objektu klase Compiler imamo popunjenu tabelu simbola,tabelu svih realokacija i tabelu svih sekcija.Realokacija kao podatke sadrzi sekciju u kojoj je definisana,tip realokacije,adresu koju treba prepraviti i simbol na koji se odnosi.Na zavrsetku obrade potrebno je u zaseban tekstualni fajl ispisati sadrzaje pomenutih tabela i zavrsiti obradu.

Testovi:

Test 1 treba da prikaze ispravan rad direktiva .align,.long,.global,.skip ,.word i .char .

**Ulaz testa 1:**

.global d,m

.text

a:

.data

dataLabela: .char 1,2,3

.skip 10

d:

.long 1500000,11

.align 4

e:

.word 6,7

.rodata

s: .char 4,5

.bss

k: .skip 8

.align 3

.end

**Izlaz testa 1:**

#Tabela Simbola

Ime Simbola |Sekcija |Offset |Scope |Tip simbola | Size|Number

.bss|bss|0|L|section|9|9

.data|data|0|L|section|28|3

.rodata|rodata|0|L|section|2|7

.text|text|0|L|section|0|1

a|text|0|L|label|0|2

d|data|13|G|label|0|5

dataLabela|data|0|L|label|0|4

e|data|24|L|label|0|6

k|bss|0|L|label|0|10

m|UND|0|G|symbol|0|11

s|rodata|0|L|label|0|8

und|UND|0|L|section|0|0

#end Symbol Table

#.ret.data

#offset | tip | vr[.data]:

#end

#.data

01 02 03 00 00 00 00 00 00 00 00 00 00 60 E3 16 00 0B 00 00 00 00 00 00 06 00 07 00

#end

#.ret.rodata

#offset | tip | vr[.rodata]:

#end

#.rodata

04 05

#end

#.ret.text

#offset | tip | vr[.text]:

#end

#.text

#end

.END

Test 2 treba da pokaze prepoznavanje svih instrukcija i postavljanje bita za uslovno izvrsavanje instrukcija.

**Ulaz testa 2:**

.global d,m

.text

a:

add r1,r1

addal r1,r2

addgt r1,r2

addne r1,r2

addeq r1,r2

add r1,10

sub r2,r3

mul c,r1

div \*20,r1

cmp r3[10],r3

and r5[e],r1

or r5[e],r1

c:

not r1,r2

push r2

pop r2

shl r4,r5

shr \*21,r1

iret

call c

.data

dataLabela: .char 1,2,3

.skip 10

d:

.long 1500000,11

.align 4

e:

.word 6,7

.rodata

s: .char 4,5

.bss

k: .skip 8

.align 4

.end

**Izlaz testa 2:**

#Tabela Simbola

Ime Simbola |Sekcija |Offset |Scope |Tip simbola | Size|Number

.bss|bss|0|L|section|8|10

.data|data|0|L|section|28|4

.rodata|rodata|0|L|section|2|8

.text|text|0|L|section|54|1

a|text|0|L|label|0|2

c|text|36|L|label|0|3

d|data|13|G|label|0|6

dataLabela|data|0|L|label|0|5

e|data|24|L|label|0|7

k|bss|0|L|label|0|11

m|UND|0|G|symbol|0|12

s|rodata|0|L|label|0|9

und|UND|0|L|section|0|0

#end Symbol Table

#.ret.data

#offset | tip | vr[.data]:

#end

#.data

01 02 03 00 00 00 00 00 00 00 00 00 00 60 E3 16 00 0B 00 00 00 00 00 00 06 00 07 00

#end

#.ret.rodata

#offset | tip | vr[.rodata]:

#end

#.rodata

04 05

#end

#.ret.text

#offset | tip | vr[.text]:

18|R\_386\_32|1

30|R\_386\_32|4

34|R\_386\_32|4

52|R\_386\_32|1

#end

#.text

C1 29 C1 2A 81 2A 41 2A 01 2A C1 20 0A 00 C5 4B CA 09 24 00 CE 09 14 00 D3 6B 0A 00 D7 A9 18 00 DB A9 18 00 DD 2A E5 40 E9 40 F9 8D FE 09 15 00 F0 00 EE 00 24 00

#end

.END

Test 3 treba da pokaze ispravno kodiranje svih instrukcija sa svim tipovima adresiranja.

**Ulaz testa 3:**

.global d,m

.text

a:

add r1,10

add r1,r2

add r4[e],r1

add r2,&c

add r2,&m

sub r2,r3[11]

mul c,r1

div \*20,r0

cmp r3[10],r3

and r1,r5[m]

or r4,\*30

c:

not r1,r4

push d

pop r2

shl r4,e

shr \*21,r1

iret

call c

.data

dataLabela: .char 1,2,3

.skip 10

d:

.long 1500000,11

.align 4

e:

.word 6,7

.rodata

s: .char 4,5

.bss

k: .skip 8

.align 4

.end

**Izlaz testa 3:**

#Tabela Simbola

Ime Simbola |Sekcija |Offset |Scope |Tip simbola | Size|Number

.bss|bss|0|L|section|8|10

.data|data|0|L|section|28|4

.rodata|rodata|0|L|section|2|8

.text|text|0|L|section|64|1

a|text|0|L|label|0|2

c|text|42|L|label|0|3

d|data|13|G|label|0|6

dataLabela|data|0|L|label|0|5

e|data|24|L|label|0|7

k|bss|0|L|label|0|11

m|UND|0|G|symbol|0|12

s|rodata|0|L|label|0|9

und|UND|0|L|section|0|0

#end Symbol Table

#.ret.data

#offset | tip | vr[.data]:

#end

#.data

01 02 03 00 00 00 00 00 00 00 00 00 00 60 E3 16 00 0B 00 00 00 00 00 00 06 00 07 00

#end

#.ret.rodata

#offset | tip | vr[.rodata]:

#end

#.rodata

04 05

#end

#.ret.text

#offset | tip | vr[.text]:

8|R\_386\_32|4

12|R\_386\_32|1

16|R\_386\_32|12

24|R\_386\_32|1

36|R\_386\_32|12

46|R\_386\_32|6

52|R\_386\_32|4

62|R\_386\_32|1

#end

#.text

C1 20 0A 00 C1 2A C3 89 18 00 C1 40 2A 00 C1 40 00 00 C5 5B 0B 00 CA 09 2A 00 CE 08 14 00 D3 6B 0A 00 D5 3D 00 00 D9 90 1E 00 DD 2C E6 00 00 00 E9 40 F9 90 18 00

FE 09 15 00 F0 00 EE 00 2A 00

#end

.END

Test 4 treba da pokaze dodatne instrukcije ret i jump i PC relativnu realokaciju kao i inicijalizovanje podataka simbolima.

**Ulaz testa 4:**

.global b,k

.text

a:

add r1,r2

jmp 20

jmp \*20

jmp m

jmp &m

jmp r1

jmp r1[2]

jmp r2[b]

jmp $m

jmp $b

jmp $d

jmp $k

ret

m: sub r3,r4

k:

.data

d:

b: .long 1,&a

.align 4

e:

.word &m,7

.rodata

.char 1,2

.word &e

.long &b

.end

**Izlaz testa 4:**

#Tabela Simbola

Ime Simbola |Sekcija |Offset |Scope |Tip simbola | Size|Number

.data|data|0|L|section|12|5

.rodata|rodata|0|L|section|8|9

.text|text|0|L|section|48|1

a|text|0|L|label|0|2

b|data|0|G|label|0|7

d|data|0|L|label|0|6

e|data|8|L|label|0|8

k|text|48|G|label|0|4

m|text|46|L|label|0|3

und|UND|0|L|section|0|0

#end Symbol Table

#.ret.data

#offset | tip | vr[.data]:

4|R\_386\_32|1

8|R\_386\_32|1

#end

#.data

01 00 00 00 00 00 00 00 2E 00 07 00

#end

#.ret.rodata

#offset | tip | vr[.rodata]:

2|R\_386\_32|5

4|R\_386\_32|7

#end

#.rodata

01 02 08 00 00 00 00 00

#end

#.ret.text

#offset | tip | vr[.text]:

12|R\_386\_32|1

16|R\_386\_32|1

26|R\_386\_32|7

34|R\_386\_PC32|7

38|R\_386\_PC32|5

42|R\_386\_PC32|4

#end

#.text

C1 2A F5 E0 14 00 F5 F0 14 00 F5 F0 2E 00 F5 E0 2E 00 F5 E9 F5 F9 02 00 F5 FA 00 00 C1 E0 0E 00 C1 E0 FE FF C1 E0 FE FF C1 E0 FE FF E9 E0 C5 6C

#end

.END