Projektni zadatak iz predmeta Sistemski Softver

**JEDNOPROLAZNI ASEMBLER**

Student:

Tamara Jevđević 2015/0492

**Opis rešenja:**

U zadatku je obrađen jednoprolazni asembler na sledeći način: Nakon što korisnik unese komandu asembler –o ulaz1.o ulaz1.s, tekst koji je u fajlu ulaz1.s se obradi i rezultat obrade se upiše u fajl ulaz1.o.

Obrada se sastoji od jednog prolaza kroz ulazni fajl, pri čemu se uzima linija po linija, obrađuje se, i njen heksadecimalni kod se upisuje u izlazni fajl.

Linje se obrađuju na različite načine u zavisnosti od tipa linije, koji može biti Sekcija, Direktiva, Instrukcija, Prazna.

Sekcija se sastoji od ključne reči „section“ i naziva sekcije u nastavku.

Od direktiva, postoje sledeće:

-global, koja označava da se izvozi simbol, ne upisuje ništa u izlazni fajl

-extern, koja označava da se uvozi simbol, ne upisuje ništa u izlazni fajl

-byte, ispisuju se argumenti u izlazni fajl, u veličini jednog bajta

-word, ispisuju se argumenti u izlazni fajl, u veličini dva bajta

-skip, kojom se ispisuje određen broj nula u onolikom broju bajtova koliki je argument koji stoji uz skip

-equ, koja služi da se nekom simbolu dodeli konstantna vrednost, ne upisuje ništa u izlazni fajl

-end, koja označava kraj ulaznog fajla, sve posle ove direktive u ulaznom fajlu će biti zanemareno

Ukoliko je instrukcija, obrađuje se u zavisnosti od načina adresiranja.

Ukoliko je prazna linija, odmah se prelazi na sledeću.

U projektu postoji klasa Asembler koja se poziva iz glavne, main funkcije i ona će primiti ulazni fajl i od njega napraviti izlazni.

U konstruktoru klase Asembler, se poziva funkcija koja koristi unordered\_maps kako bi se sačuvali parovi:

(naziv instrukcije-kod naziva instrukcije),

(naziv registra-kod registra)

(naziv načina adresiranja-kod načina adresiranja)

Takođe, u toj funkciji se prave nizovi nulaparametarskih, jednoparametarskih i dvoparametarskih instrukcija, kao i niz skokova, kako bi se kasnije moglo proveravati u koju grupu spada instrukcija.

Još, u konstruktoru se pravi jedna tabela simbola i jedna tabela sekcija, koje su jedinstvene za ceo ulazni tj. izlazni fajl, i poziva se funkcija jedanJediniProlaz();

Funkcija jedanJediniProlaz() zapravo uzima liniju po liniju i uz pomoć pomoćnih klasa, pravi izlazni fajl.

Zahvaljujući pomoćnoj klasi ParserLinije, svakoj liniji se uklone komentari, zatim beli znaci, zatim se ostatak raščlani na delove(labela, sekcija, direktiva, instrukcija, izraz, brParam, kodInstrukcije), od kojih će u određenom trenutku, u zavisnosti od tipa linije, samo određeni delovi(zapravo oni koji su potrebni) imati validnu vrednost. Te delove kasnije dohvatamo i sastavljamo izlazni kod.

Prilikom nailaska na simbol, on se ubacuje u tabelu simbola, u izlazni fajl se upisuju nule u onoliko bajtova koliko je potrebno, a u listuPojavljivanja, koju ima svaki simbol, se upisuje lokacija gde se simbol pojavljuje.

Takođe, čuva se i relokacioni zapis u tabeliRelokacionih zapisa koja se odnosi na sekciju u kojoj se simbol pojavio(tekuću sekciju).

Kasnije, kada se bude znala tačna vrednost simbola, ona će se i postaviti u izlazni fajl, svuda gde se taj simbol pojavljuje, ukoliko je to potrebno.

Ukoliko je situacija takva da se simbol koristi pre nego što se definiše, u tom slučaju, simbol će biti nedefinisan, a kada se naiđe na njegovu definiciju, tom prilikom će se njegova vrednost pomeraja i sekcija kojoj pripada setovati u tabeli simola. Tada se, takođe, prolazi kroz sve relokacione zapise i ukoliko je u pitanju lokalni simbol, za redni broj relokacionog zapisa se postavlja redni broj sekcije u kojoj se simbol nalazi. Ukoliko je globalni simbol, onda je redni broj u relokacionom zapisu zapravo redni broj tog simbola u tabeli simbola. Kako ne bi došlo do problema ukoliko se simbol tek kasnije proglasi globalnim, svaki put kada se simbol proglasi globalnim/eksternim, setuje se redni broj relokacionog zapisa koji se odnosi na taj simbol.

U tabeli simbola čuvaju se i sekcije, dodatno, za simbole koji su ustvari sekcije, čuva se i veličina, koja predstavlja broj bajtova od početka sekcije do početka sledeće sekcije, odnosno od početka, do direktive end, kada je u pitanju poslednja sekcija. Veličina sekcije se računa kada naiđemo na sledeću sekciju, odnosno na direktivu end.

Pri nailasku na sekciju, postavljamo je u tabelu simbola, jer je i to jedna vrsta simbola, takođe je stavljamo u tabelu sekcija i tom prilikom se pravi nova sekcija i njena lista relokacionih zapisa.

Tada se resetuje se brojač „pomeraj“, koji broji bajtove unutar sekcije i dohvataju se „bajtoviZaIspis“ te sekcije, koji se popunjavaju bajtovima koji predstavljaju rezultat kodiranja ulaznog fajla, odnosno to je predmetni program tekuće sekcije.

Pri nailasku na instrukciju, ParserLinije je rasparča na naziv i operande ukoliko ih ima, zatim se svi delovi posebno kodiraju.

Tu se razlikuju nazivi instrukcija, veličine instrukcija, načini adresiranja, i u odnosu na to se prave kodovi za izlazni fajl.

Pri nailasku na direktivu, ona se obrađuje u skladu sa tim šta ona radi i u odnosu na to se prave kodovi za izlazni fajl.

Pri nailasku na direktivu end, pored postavljanja veličine poslednje sekcije iz ulaznog fajla, poziva se funkciju “sredi”, koja ima zadatak da proveri da li su svi simboli u tabeli simbola, koji nisu extern ili equSimboli, definisani, ukoliko to nije slučaj, imamo grešku, jer se koristi nedefinisan simbol i program prekida rad.

Zatim, ova funkcija prolazi kroz sve equ simbole i razrešava njihove vrednosti. Ovo se radi sve dok se ne razreše svi equ simboli. Ukoliko se desi da se u jednom prolazu ne razreši nijedan equ simbol, a ima još nerazrešenih, to znači da imamo beskonačnu petlju i tada program javlja grešku i prekida svoj rad.

Dalje, ova funkcija razrešava sva pojavljivanja tako što prolazi kroz sve simbole i za svaki kroz njegovu listu pojavljivanja, koja sadrži mesto gde se simbol pojavio, sekciju u kojoj se pojavio i veličinu na kojoj treba ispisati simbol.

U svakoj iteraciji, u “bajtoviZaIspis” sekcije koja je u tekućem pojavljivanju, na mesto koje je sačuvano u tekućem pojavljivanju, i u veličini koja je sačuvana u tekućem pojavljivanju, upisuje se određena vrednost. Razlikuju se slučajevi kad je u pitanju pcRel adresiranje za lokalni simbol, pcRel adresiranje za globalni simbol, aps adresiranje za lokalni simbol, aps adresiranje za globalni simbol, u zavisnosti od toga, upisujemo razliku trenutnog PC-ja i veličine operanda, odnosno samo negativnu vrednost veličine operanda, odnosno nule, odnosno vrednost pomeraja simbola.

I na kraju, funkcija “sredi” poziva funkcije za ispis tabele simbola, tabela relokacionih zapisa i predmetnih programa.

**Klase:**

**Asembler** – glavna klasa, koja uzima liniju po liniju, šalje je parseru linije od kog dobija samo potrebne delove linije i kodira ih. Usput, popunjava tabelu simbola, tabelu sekcija, tabelu relokacionih zapisa, koje na kraju obrade ispisuje.

**ParserLinije** – prima izvornu liniju, obrađu je i njene delove čuva

**Simbol** – ima naziv, broj sekcije kojoj pripada, vidljivost(“l” – lokalni simbol ili “g” – globalni ili eksterni simbol), redni broj u tabeli simbola(jedinstven za svaki simbol), pomeraj(mesto definisanja simbola, u odnosu na početak sekcije u kojoj je definisan), takođe, simbol ima i dodatna polja, ali se ona ne koriste za sve vrste simbola(vrste: equ simbol, sekcija, “obican”, extern)

**Sekcija** – ima naziv i broj(isti kao redni broj tog simbola u tabeli simbola), čuva listu relokacionih zapisa

**RelokacioniZapis** – ima imeSimbola, pomeraj(mesto na koje se odnosi relokacioni zapis), tip(R\_8-apsolutno, jedan bajt, R\_16- apsolutno, dva bajta, R\_PC8-PC relativno, jedan bajt, R\_PC16-PC relativno dva bajta), redniBroj(redni broj simbola u tabeli simbola ako je global/extern simbol, odnosno sekcija simbola ako je lokalni simbol), veličina(čuva koliko bajtova je u pitanju)

**TabelaRelokacionihZapisa** – čuva listu relokacionih zapisa

**TabelaSekcija** – čuva listu sekcija

**TabelaSimbola** – čuva listu simbola

**Korišćeni alat:**

Okruženje Visual Studio Code

Kompajler g++(verzija 9.3.0) – instaliran pomoću:

$sudo apt install build-essential

Provera verzije:

$gcc –version

Komanda za kompajliranje: g++ asembler.cpp main.cpp parserLinije.cpp relokacioniZapis.cpp simbol.cpp tabelaRelokacionihZapisa.cpp tabelaSekcija.cpp tabelaSimbola.cpp –o asembler

asembler – dobijeni izvršni fajl pomoću kog testiramo na sledeći način:

./asembler –o izlaz.o ulaz.s

**TESTOVI:**

TEST adresiranjeinstrukcija.s

.extern e

.section sekcija

mov b,%r1

movw c,x

movb e,a(%r1)

sub c, b(%r5)

movb $a, a

movb $b, b

x: jmp \*x

jne \*2

jmp \*c

d: movw $-5,d

.section data

a: .byte 1

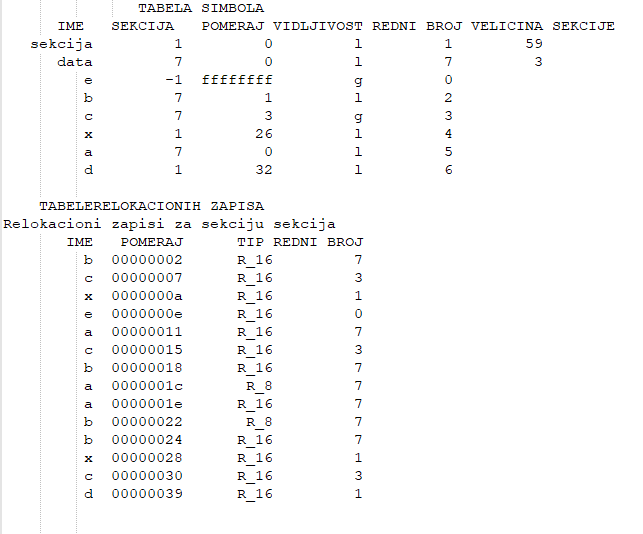
b: .word 2

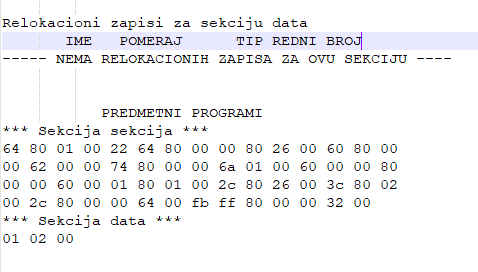
.global c

c:

.end

IZLAZ:





**TEST bytewordskip.s**

.global c

.extern f

.section text

.byte a

.byte 5

.byte c, f

.byte 5, 6

.word 8, 5

.word a, c

.skip 10

x: .byte r

.section data

.byte 5,c

a: .byte 0x20, -0x05

b: .byte 0x5, -1, a

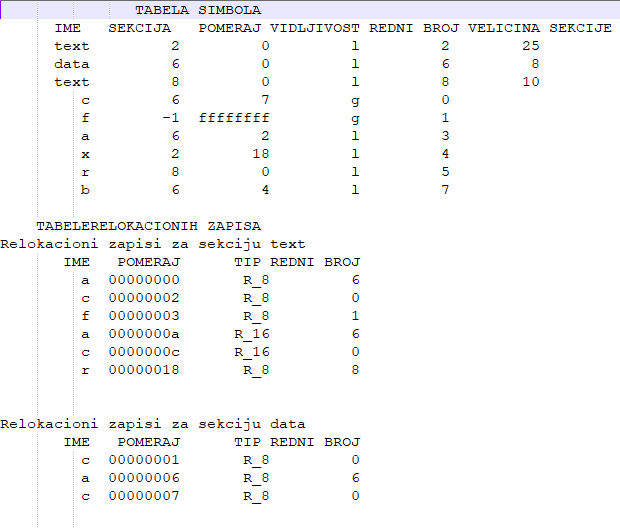
c: .byte c

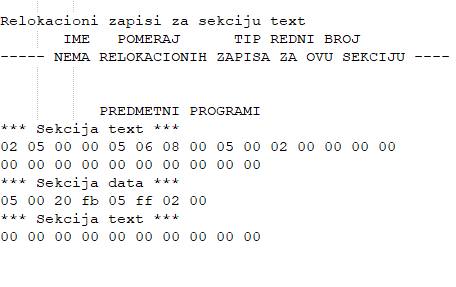
.section text

r: .skip 0x000A

.end

**IZLAZ:**





**TEST equ.s :**

.section data

.equ tamara, +10+0xA - 15 + a #12 = 0x0c

.equ marko, +0xA + 9 - b #12 = 0x0c

.equ pavle, 20 - 7 + c #118 = 0x76

.equ jovana, +0xAB + 6 -d# = 170 = 0xaa

.global k

.equ r, 0x78

.equ k, 7

.word 10

.word 5

.word 0x9

.word -0x0008

.equ b, 7

.equ c, r - 15

.section text

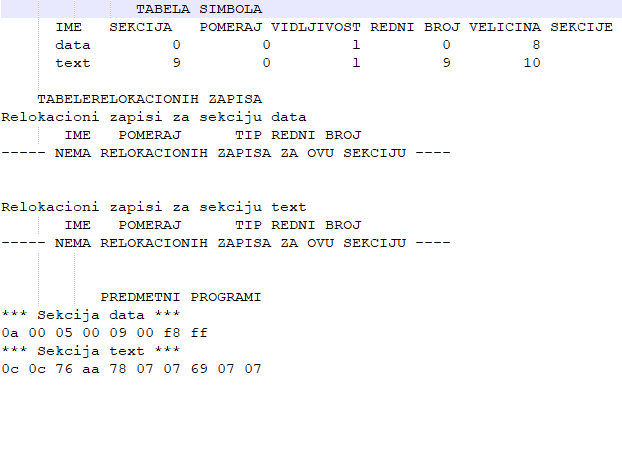
.equ a, k

.equ d, a

.byte tamara, marko, pavle, jovana, r, k, b, c, a, d

.end

**IZLAZ:**



**TEST pcrel.s:**

.section data

.global d

.extern c# bvjbhjv

.global x#ffj

#fffdfd

a: .byte 0x0002551

b: .word -0x54

d: .word 80

.section text

y: .byte d

x: .byte 0x6

z: .byte 8

.section data

p: movw c(%pc),y

t: jmp \*t(%pc)

tamara: xchg a, b

jne \*d

sub a(%pc), 5

mov $3, (%r6)

.end

Izlaz testa pcrel.s:

**IZLAZ:**

