**SISTEMSKI SOFTVER**

**PROJEKTNI ZADATAK JUN 2019.**

**autor: Stojanovic Jana 0179/2016**

**Zadatak:**

Prvi zadatak predstavlja implementaciju dvoprolaznog asemblera na Linux operativnom sistemu. Ulaz asemblera je tekstualni fajl u kojem je napisan asemblerski program. Izlaz asemblera je tekstualni fajl koji predstavlja predmetni program. Taj predmetni fajl sadrzi tabelu simbola, masinski kod, ali I relokacione zapise koje ce linker koristiti.

**Resenje:**

Za realizaciju ovog zadatka napravljene su sledece klase:

1. **MyCompiler**-klasa koja sadrzi sve najbitnije metode za obradu ulaznog fajla I ispis u izlazni fajl, kao sto su firstRun,secondRun, metode za parsiranje teksta, metode za ispis..
2. **Symbol**- klasa koja sadrzi najbitnija polja kao sto su id simbola,njegov tip,sekcija u kojoj je definisan, vrednost tog simbola(offset), dodatni flegovi koji ukazuju na to da li je taj symbol sekcija, flegovi koji ukazuju na to da je symbol dodat u tabeli direktivom .equ..
3. **MyTable**- klasa predstavlja vector svih simbola; sadrzi sve potrebne metode za radi sa simbolima, kao sto su dodavanje novog simbola, pretraga simbola po imenu, dohvatanje datog simbola…
4. **RelData-** klasa koja predstavlja jedan relokacioni zapis I sadrzi polja: adresa, tip relokacije(apsolutna ili pcrelativna), id koji se referencira zapisom(id simbola za globalni ili id sekcije za lokalni)
5. **RelTable**- klasa koja predstavlja skup svih relokacionih zapisa za datu sekciju

Metoda od koje se krece sa obradom je metoda **firstRun()** u klasi MyCompiler. Ova metoda vrsi citanje iz ulaznog fajla, red po red i vrsi njegovu obradu. Na osnovu prve reci, odredjuje se da li je rec o definiciji simbola, direktivi, instrukciji.. i na osnovu toga, obrada se prosledjuje nekoj od mnogobrojnih funkcija. Ideja prvog prolaza je da se formira tabela simbola , da se vodi racuna o ispravnom inkrementiranju location counter-a, ali i da se sve neregularne situacije otkriju i prijave(sintaksne greske, nedozvoljena adresiranja, visestruke definicije simbola,definicija eksternog simbola, nedostatak .end direktive u kodu…).

Nakon zavrsetka prvog prolaza neophodno je iterirati kroz tabelu simbola, kako bismo proverili da li su svi simboli ispravno definisani. U ovom prolazu, pored ispitivanja definisanosti simbola, vrsila sam I izracunavanje svih simbola uvedenh .equ direktivom. Ukoliko neki simbol nije definisan, ili se ne moze izracunati vrednost equ izraza, prijavljuje se greska I program zavrsava sa radom.

Metoda kod koje se vrsi drugi prolaz je metoda **secondRun**() u klasi MyCompiler. Ova metoda vrsi novo citanje ulaznog fajla i za sekcija .data i .text generise masinski kod I relokacione zapise. Greske koje ova metoda moze prijaviti su- koriscenje loseg pomeraja prilikom registarskog-indirektnog adresiranja, koriscenje simbola koji ne postoji, koriscenje prevelike vrednosti za neposredno adresiranje… Na kraju drugog prolaza se tabela simbola, masinski zapis i relokacioni zapis upisuju u izlazni fajl.

U daljem tekstu, navescu ideje koja sam usvojila, kako bih realizovala ovaj projekat:

1. **Adresiranja i tipovi operanda u instrukcijama:** *add,sub,mul,div,and,or,xor,test,cmp,mov:*  -prvi operand->Rx,simbol,Rx[pom],psw,[Rx]

-drugi operand->sva moguca adresiranja, pri cemu je bitno da bar jedan operand bude registar

*jmp,jgt,jeq,jne:*

- dozvoljeno je da operand bude registar, simbol(apsolutna relokacija) ili $simbol(pc relativna relokacija)

*push,pop,not:*

-dozvoljeno je Rx,Rx[pom],[Rx],psw

*int:*

-broj ili &simbol

*call:*

-simbol, prilikom cega se obavezno vrsi relokacioni zapis

*shl,shr:*

-prvi operand mora biti registar

-drugi operand je broj ili &simbol

*xchg:*

-prvi I drugi operand mogu biti->Rx,Rx[pom],psw,simbol, pri cemu je bitno da bar jedan operand bude registar

1. Usvojila sam da je registarko indirektno adresiranje bez pomeraja sintaksno moguce predstaviti kao **[Rx]**
2. Za **.equ** sam usvojila da je maksimalno moguce imati 3 operanda u izrazu, da su dozvoljene operacije ‘+’ ili ‘-‘ . Kako bih izbegla situaciju kruzne zavisnosti, uvela sam ogranicenje da se u izrazu ne sme koristiti drugi .equ simbol
3. Uvela sam da su svi podaci i pomeraji na **2B.**
4. Simboli **moraju poceti slovom**, a mogu se zavrsavati brojem.
5. Nije dozvoljeno pisanje izraza **za .word i .byte** direktive, dozvoljeno je navesti broj ili neki simbol
6. Ne postoje kljucne reci sp,pc, ali postoji psw.
7. Prilikom pisanja **zareza**, bitno je da oni stoji iskljucivo uz operand: npr: add r1, r4 .byte a, b, 120
8. Zabranjeno je definisati dve labele jednu za drugom
9. Kod **apsolutne relokacije** se razlikuju dva slucaja:

-koriscenje globalnog simbola-referencira se upravo taj simbol, a na njegovo mesto u masinskom kodu upisuje “00 00” -koriscenje lokalnog simbola-referencira se sekcija u kojoj je taj simbol definisan, a na njegovo mesto u masinskom kodu se upisuje njegova vrednost iz tabele simbola

1. **Pc relativna relokacija** se javlja samo kod skokova(**branch $lab**) I tu sam usvojila sledeca pravila:

-Kada se radi o globalnom simbolu, na mestu operanda se upisuje “00 00”, a u relokacionom zapisu se referise upravo taj simbol I upisuje njegov id

-Kada se radi o lokalnom simbolu u istoj sekciji, nema potrebe za relokacijom, jer je taj pomeraj uvek konstantan, bez obzira na smestanje u memoriji

-Kada se radi o lokalnom simbolu iz druge sekcije upisuje se na mesto val-2B, gde je val vrednost iz tabele simbola, a referencira se upravo ta sekcija

**Uputstvo za prevodjenje I pokretanje:**

Komanda u terminalu za formiranje exe fajla(g++ verzija 4.9 I c++11):

**g++-4.9 -std=c++11 main.cpp MyCompiler.cpp RelData.cpp RelTable.cpp Symbol.cpp MyTable.cpp -o projekat**

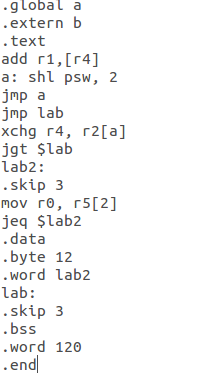
Komanda u terminalu za pokretanje exe fajla: (tekuci direktorijum je direktorijum exe fajla)

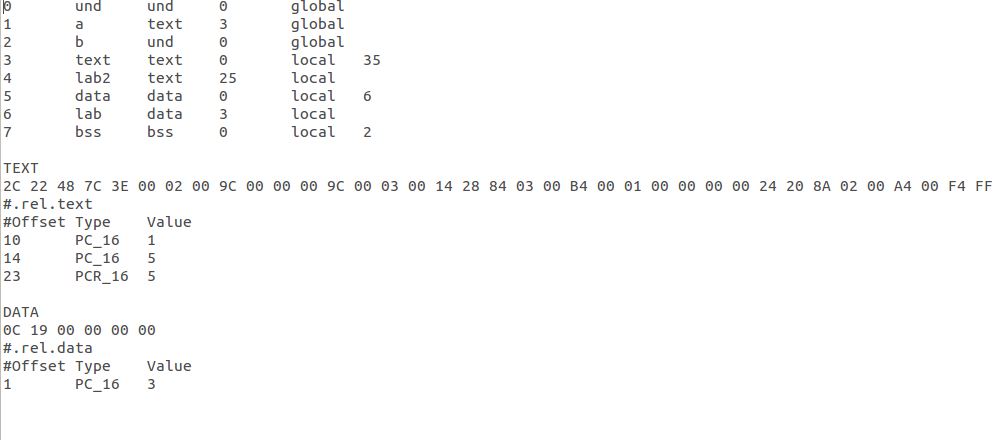
**projekat ulaz.txt izlaz.txt**

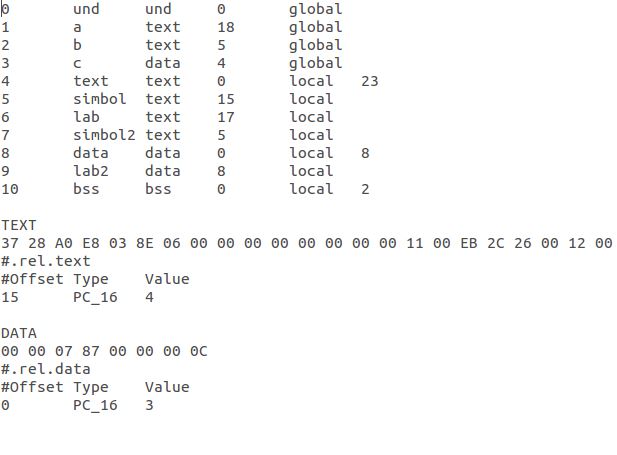
Nakon izvrsavanja programa, u konzoli ce biti ispisan isti sadrzaj kao i sadrzaj izlaznog fajla.U slucaju neke greske i neregularne situacije, u konzoli ce biti ispisano koja je greska u pitanju.

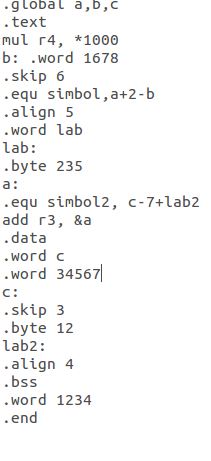
**Testovi:**

**Test1**-Ideja je da se prikazu razlicite instrukcije, ali i instrukcije skokova sa 3 slucaja-apsolutna relokacija, pc relativna relokacija i slucaj kada nije potrebno praviti relokativni zapis





**Test2**-Ideja je da se prikazu razliciti nacini koriscenja .word, .byte i .equ direktive



**Test3:** Ideja je prikazati sto vise instrukcija I razlicite dozvoljene nacine adresiranja

**.global a,b**

**.extern c**

**.text**

**.skip 3**

**lab3:**

**add r2[-10],r5**

**xchg r5[-2],r0**

**xchg r1[3],psw**

**jmp $lab3**

**xchg r1,r2**

**jne $labb**

**jne $b**

**jeq a**

**jgt lab**

**call a**

**call c**

**call lab**

**not r5[a]**

**push r2[23]**

**push psw**

**pop [r4]**

**push r3**

**shl r1,3**

**shr psw, &b**

**lab:**

**.byte 12**

**.byte lab**

**.equ novi,3**

**.align 4**

**int &a**

**ret**

**int 6**

**a:**

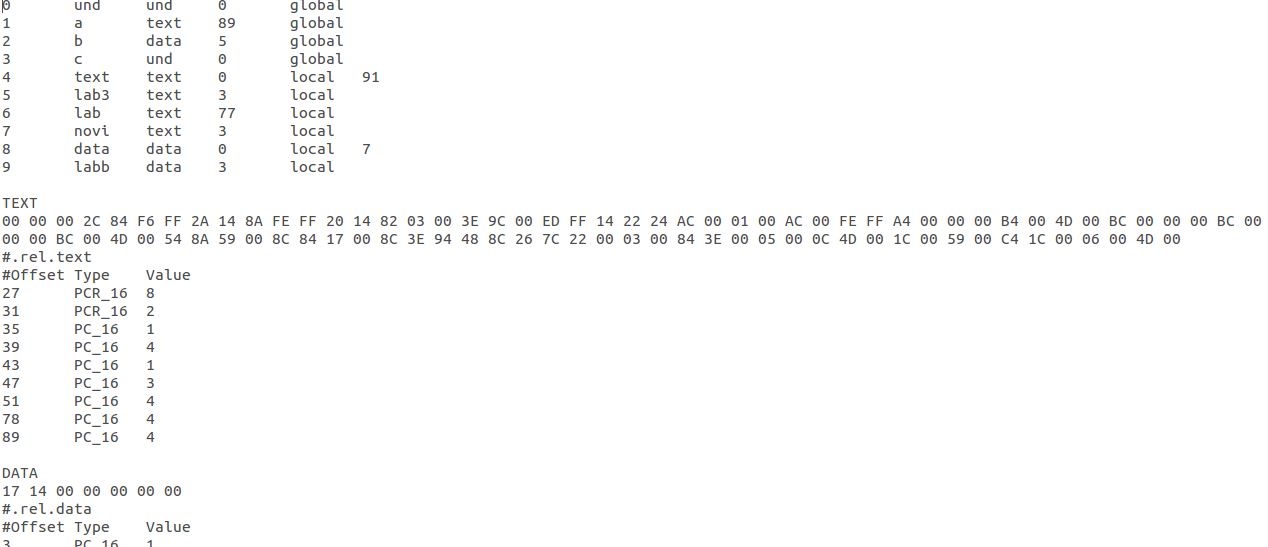
**.word lab**

**.data**

**.byte 23,20,0**

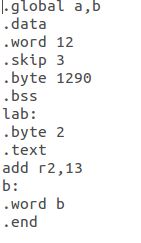
**labb: .word a**

**b: .skip 2**

**.end**

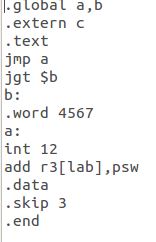
**Test4:** Sledeci test sluzi da bi prikazao gresku koja moze da se registruje nakon prvog prolaza-u ovom slucaju je to nedefinisan globalni simbol nakon prvog prolaza

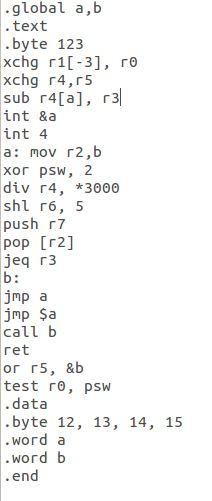
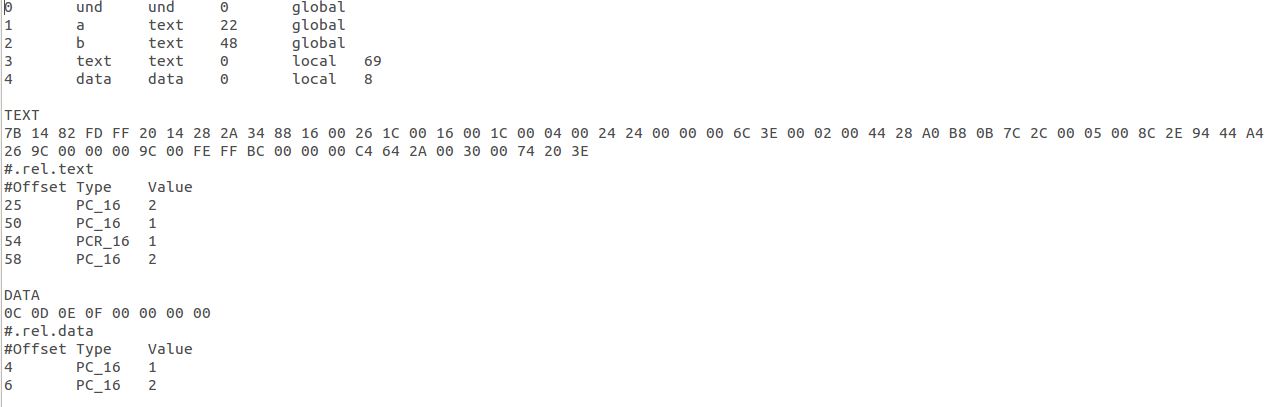
Izlaz iz programa I ispis na konzoli: **Error!Definition missing for global symbol!**



**Test5**: Ideja je da se prikaze greska koja moze nastati nakon drugog prolaza. U ovom slucaju je to koriscenje simbola koji ne postoji u tabeli.

Izlaz iz programa I ispis na konzoli: **Error! Symbol doesn’t exist in symbol table!**



**Test6:** Razlicite instrukcije sa razlicitim nacinima adresiranja