**Elektrotehnički fakultet**

**Univerzitet u Beogradu**



**Projekat iz predmeta Sistemski softver**

**ASEMBLER I EMULATOR**

Nenad Babin

0585/2016

Beograd, avgust 2019. god.

SADRŽAJ

[1. UVOD 3](#_Toc16791493)

[2. OPIS REŠENJA 3](#_Toc16791494)

[2.1 Asembler 3](#_Toc16791495)

[2.2 Linker i Emulator 5](#_Toc16791496)

[3. TESTOVI 7](#_Toc16791497)

[3.1 Test 1 (petlja) 7](#_Toc16791498)

[3.2 Test 2 7](#_Toc16791499)

[3.3 Test 3 (Hello, world!) 8](#_Toc16791500)

[3.4 Test 4 (tajmer i terminal) 8](#_Toc16791501)

[3.5 Test 5 (int instrukcija) 9](#_Toc16791502)

[3.6 Test 6 (deljenje nulom) 9](#_Toc16791503)

[3.7 Test 7 (direktiva .equ) 9](#_Toc16791504)

[3.8 Test 8 (prava pristupa) 10](#_Toc16791505)

[3.9 Test 9 (std) 10](#_Toc16791506)

[3.10 Test 10 (.bss) 10](#_Toc16791507)

[3.11 Test 11 (negativne vrednosti) 11](#_Toc16791508)

[4. UPUTSTVO ZA POKRETANJE 11](#_Toc16791509)

# UVOD

Ovaj projekat je izrađen u okviru predmeta Sistemski softver koji se održava na trećoj godini Elektrotehničkog fakulteta na odseku za Softversko inženjerstvo i Računarsku tehniku i informatiku. Sam projekat je obavezan.

Cilj projekta je implementacija dvoprolaznog asemblera zasnovanog na principima rada GNU asemblera i emulatora koji će izlaz asemblera učitati i izvršavati. Asembler prima jedan ulazni tekstualni fajl, obrađuje ga i generiše tekstualni fajl koji je nalik na objektni fajl po ELF formatu. O okviru emulatora imeplementirani su i povezivač (engl. Linker) i punilac (engl. Loader). Cilj povezivača je da fajlove koji su izlaz asemblera obradi tako da spoji istoimene sekcije, prepravi sadržaj tih sekcija i prosledi ih puniocu na dodatne prepravke i učitavanje u memoriju. Povezivač još proverava da li ima nerazrešenih simbola ili višestrukih definicija simbola. Punilac vrši dodatne promene sadržaja sekcija na osnovu –place argumenata komandne linije. Ukoliko ne postoje –place argumenti, punilac smešta sekcije proizvoljnim poretkom počev od nulte adrese. Punilac takođe vrši proveru da li postoji preklapanja izmežu sekcija i da li sve sekcije mogu kompletno da stanu u memoriju. Nakon učitavanja emulator dohvata instrukciju po instrukciju iz memorije i izvršava ih.

# OPIS REŠENJA

## Asembler

Asembler se sastoji od sledećih klasa: Pass, FirstPass, SecondPass, Symbol, Section i Relocation. Klasa Pass je natklasa klasa FirstPass i SecondPass koje obrađuju prvi, odnosno drugi prolaz asemblera, respektivno. Symbol, Section i Relocation predstavljaju apstrakcije simbola, sekcije i jednog zapisa o relokaciji, respektivno.

Klasa Pass sadrži metode i statička polja koja su zajednička za oba prolaza. Ona, takođe, sadrži apstraktne metode begin i next koje definišu izvedene klase, kao i statička polja koja predstavljaju vektore simbola, relokacija i sekcija. Ti vektori se popunjavaju objektima simbola i sekcija u prvom prolazu i kasnije se koriste u drugom prolazu, dok se vektor relokacija popunjava odgovarajućim objektima u drugom prolazu. Klasa ima mogućnost parsiranja jednog tokena ulaznog fajla, dohvatanja veličine jedne isntrukcije i veličine njenih operanada, kao i pomoćne metode za konverziju stringova.

Klasa FirstPass implementira metode begin i next. Svaki prolaz počinje nakon poziva metoda begin. Iz ulaznog fajla se čita red po red i parsiraju se token. Jedan token prodstavlja jednu reč, tj. string iz ulaznog fajla. Klasa sadrži metode za obradu direktiva .global i .extern, odnosno za obradu .equ, .align, .skip, .word, .byte i .section direktiva. I prvom prolazu se ne generše nikav izlaz i ne analizira se smisao instrukcija, već se samo na osnovu operacionog koda i operanada odlučuje o veličini instrukcije i na osnovu toga uvećava brojač lokacija (engl. location counter). Sve direktive koje generišu sadžaj samo uvećavaju brojač lokacija u prvom prolazu, dok se u drugom prolazu generiše sam sadržaj u izlaznom fajlu. U nastavku slede opisi direktiva, kao i njihova upotreba.

.equ symbol, value - Direktiva u tabelu simbola ubacuje novi apsolutan simbol sa vrednosti value. Vrednost value može biti u heksadecimalnom ili dekadnom brojnom sistemu. Ne utiče na brojač lokacija. Simbol se može definisati direktivom pre ili posle korišćenja simbola, međutim ponavljanje direktive sa istim simbolom nije dozvoljeno. Takođe, nije dozovljeno da simbol sa istim imenom bude definisan uz pomoć labele i .equ direktive.

.equ N, -1

.equ N, 0xffff

. align value – Direktiva ubacije prazne bajtove i uvećava brojač lokacija tako da brojač lokacija ima vrednost deljivu sa value. Ukoliko je vrednost brojača lokacija već deljiva sa value, onda se brojač lokacija ne menja. Value je broj u dekadnom brojnom sistemu između 0 i 32767.

.align 4

.skip value – Direktiva uvećava brojač lokacija za value. Value je broj u dekadnom brojnom sistemu između 0 i 32767.

.skip 10

.word value – Direktiva ubacuje sadržaj value u izlazni fajl na prostor od dva bajta i uvećava brojać lokacija za dva. Value je broj u dekadnom ili heksadecimalnom brojnom sistemu. Ukoliko se value ne navede, rezerviše se prostor za jednu reč, ali je prostor od dva bajta inicijalizovan nulama.

.word 0x0001

.word 1

.word

. word symbol – Direktiva ubacuje vrednost simbola symbol u izlazni fajl na prostor od dva bajta i uvećava brojać lokacija za dva. Symbol ne mora biti definisan u ulaznom fajlu.

.word N

.byte value – Direktiva ubacuje sadržaj value u izlazni fajl na prostor od jednog bajta i uvećava brojać lokacija za jedan. Value je broj u dekadnom ili heksadecimalnom brojnom sistemu. Nije dozvoljeno .byte symbol jer vrednost simbola može biti veća od jednog bajta nakon povezivanja i punjenja. Ukoliko se value ne navede, rezerviše se prostor za jedan bajt, ali je prostor od jednog bajta inicijalizovan nulama.

.byte 3

.byte 0x03

.byte

.section sectionName [ “flags”] – Direktiva koja najavljuje početak sekcije sa nazivom sectionName. Prava pristupa se mogu proizvoljno zadati. sectionName ne sme biti text, data ili bss. Prava pristupa mogu biti: R – čitanje, W – upis, E – izvršavanje, P – sadržaj sekcije se nalazi u predmetnom programu. Flags nisu obavezni, ali se uzima da ne postoje nikakva prava pristupa nad sekcijom. sectionName se navodi bez tačke.

.section ivTable “PRW”

.section akaBSS “RW”

.section akaTEXT “PRE”

.global symbol – Direktiva koja izvozi simbol symbol. Može se navesti pre ili posle definisanja simbola. Simbol mora biti definisan u tekućem fajlu.

.extern symbol – Direktiva koja uvozi simbol symbol. Simbol ne sme biti definisan u tekućem fajlu.

U prvom prolazu se dodaju i svi simboli koji su označeni sa label: i oni zajedno sa simbolima definisanih preko .equ direktive čine tabelu simbola. Njihova vrednost je trenutna vrednost brojača lokacija. Nakon prvog prolaza imamo vrednosti svih simbola i njihovu vidljivost, kao i veličine sekcija.

Klasa SecondPass implementira metode begin i next i sadži metode za obradu direktiva, instrukcija i njihovih operanada i parsiranje i koverziju stringova. Iz ulaznog fajla se čita red po red i parsiraju se tokeni. Ukoliko se naiđe na direktivu koja generiše sadržaj, u objektu odgovarajuće sekcije se upisuje na odgovarajućoj poziciji generisani sadržaj. Direktiva .equ se ignoriše. Pri obradi direktiva .global i .extern se proverava da li su izvezeni simboli definisani, dok se za uvezene provera suprotno. Obrada instrukcija se zasniva na generisanju sadžaja u odgovarajućoj sekciji. Vezano za instrukcije se uvedene sledeće pretpostavke:

* Ukoliko nije naveden sufiks b u mnemoniku, veličina operanada je dva bajta. Ukoliko je naveden sufiks w u mnemoniku, veličina operana je dva bajta. Ukoliko je naveden sufiks b u mnemoniku, veličina operana je jedan bajt.
* Ukoliko su operandi dva bajta, nije moguće koristi viših ili nižih osam bira registara.
* Ukoliko su operandi jedan bajt, ne mora se koristiti sufiks l ili h za registre, ali se podrazumevano koristi nižih 8 bita registra.
* Ukoliko su operandi dva bajta, ne smeju se koristi sufiksi l i h za registre.
* Sa rx[0] se označava registarsko indirektno adresiranje. Kodira se kao registarsko indirektno, a ne kao registarsko indirektno sa pomerajem 0.
* Radi pojadnestavnjenja, instrukcije jmp, jeq, jne i jgt umesto dst operanda imaju src operand.

movb r2[0], 3 ⬄ movb r2l[0], 3

Za adresu u memoriji se koristi samo donjih osam bita registra r2 i broj 3 se smešta na jednu lokaciju odreženu sa donjih osam bita registra r2.

mov r2h, 7

Nije korektno. r2h – jednobajtni operand, a mov bez sufiksa sugeriše dvobajtne operande.

movw r2[127], 0x1234 ⬄ mov r2[127], 0x1234

Na dve susedne memorijske lokacije se smešta heksadecimalna vrednost 1234. Adresa se odrežuje na osnovu 16-bitne vrednosti u registru r2 koja se sabira sa pomerajem kodiranim u instrukciji – 127.

Instrukcije se mogu pisati samo u sekcijama koje imaju pravo izvršavanja. Destinacioni operand ne sme da bude neposredna vrednost ili apsolutna vrednost.

Klasa Symbol predstavlja apstrakciju simbola i sadrži informacije o imenu simbola, naziv sekcije u kojoj se nalazi (undefined ako je nedefinisan ili absoulute ako je definisan direktivom .equ), vrednost, redni broj i opseg važenja.

Klasa Section predstavlja apstrakciju sekcije i sadrži informacije o imenu sekcije, veličini sekcije, rednom broju i pravima pristupa. Svaka sekcija sadrži niz bajtova koji predstavlja sadržaj sekcije koji se popunjava tokom drugog prolaza.

Klasa Relocation predstavlja apstrakciju jednog ulaza u tabeli relokacija i sadži pokazivače ka sekciji u kojoj se nalazi i simbolu na koji se odnosi. Takođe sadrži pomeraj u odnosu na početak sekcije u kojoj treba linker da vrši prepravku i tip relokacije.

Nakon izvršavanja se generiše tekstrualni fajl koji liči na izlaz školskog ELF formata. Takođe se generiše i assembly.log fajl koji sadži tok izvršavanja i pogodan je da razjasni u kom je trenutku došlo do greške ukoliko do nje dođe, a poruka o grešci nije dovoljno jasna.

## Linker i Emulator

Linker i emulator se sastoje od sledećih klasa: ObjectFile, Linker, Emulator, AccessRights, Memory, Symbol, Section i Relocation.

Klase Memory, Symbol i Relocation su identične delu zadatka sa asemblerom.

Klasa ObjectFile predstavlja apstrakciju objektnog fajla. Klasa parsira jedan teksutalni fajl koji predstavlja objektni fajl i prikuplja informacije o simbolima, sekcijama, relokacijama i sadržajima sekcija. Svaka instanca sadrži i ime fajla koji je ulaz programa. Nakon parsiranja svaki objekat odgovara jednom ulaznom fajlu i sadži vektore objekata sekcija, simbola i relokacija koji odgovaraju datom fajlu.

Klasa Linker predstavlja apstrakciju povezivača i punioca. Instanca klase prvo proverava da li ima višestrukih definicija simbola, nerazrešenih simbola i da li je definisan simbol \_start. Ukoliko su ispunjeni zahtevi, pokušava se učitavanje sadržaja sekcija u memoriju na osnovu –place parametara. Ukoliko –place parametara nema, sve sekcije se nakon spajanja smeštaju u proizvoljnom redosledu počev od adrese 0. Ukoliko postoje parametri, proverava se da li postoji preklapanja između sekcija i da li sve sekcije mogu da stanu u memoriju. –place parametri se navode na sledeći način:

-place=.ivTable@0x0000

Ispravno.

-place=.ivTable@0x00

Nije ispravno. Nakon @ mora da se navede heksadecimalni broj u formatu 0xnnnn gde je n heksadecimalna cifra.

-place=.ivTable@0

Nije ispravno. Nije dozvoljen broj u dekadnom brojnom sistemu nakon @.

Ukoliko nema problema, vrše se preprake u sadržajima svih sekcija tako da se programi mogu uspešno učitati na odgovarajuće adrese. Linker kreira i objekte klase AccessRights o kojima će kasnije biti reči. Nakon završetka linkovanja, main program u objekat klase Memory učitava sadržaj sekcija. O klasi Memory će kasnije biti reči.

Klasa AccessRights predstavlja apstrakciju prava pristupa i sadrži informacije koji deo memorije (od-do) ima koje privilegije pristupa. Privilegije su: R – čitanje, W – pisanje i E – izvršavanje. Objekte ove klase kasnije koristi Memory i Emulator kako bi proveravali ispravnost pristupa prilikom izvršavanja.

Klasa Memory predstavlja apstrakciju memorije i ima mogućnost upisa bloka podataka (koristi se pri inicijalizaciji), upisa i čitanja jednog bajta i upisa i čitanja dva bajta. Objekat kalse Memory koristi Emulator u vidu emulirane memorije. Memory sadži vektor dužine 216 koji se sastoji od bajtova.

Klasa Emulator predstavlja apstrakciju emulatora i srž je zadataka. Instanca klase Emulator predstavlja intepratativni emulator koji sadži metode za dohvatanje i izvršavanje instrukcije, dohvatanje operanada, upis rezultata izvršavanja, čitanja i upisa u registre, dohvatanje i postavljanje bita PSW registra. Emulator poseduje glavnu nit koja dohvata instrukcije i izvršava ih, nit koja osluškuje prekid od tastature i nit koja predstavlja tajmer i koja nakon proteklog vremena poziva prekidnu rutinu u ulazu dva u tabeli prekidnih rutina. Postoji niz u kome se pamte signali da je do prekida došlo kako bi se prekidi nakon izvršavanja instrukcije mogli obraditi. Tajmer i terminal rade sve dok radi i procesor koji dohvata instrukcije, dekoduje instrukcije, dohvata operande, izvršava isntrukciju i na kraju obrađuje prekide. Prekidi od tastature i terminala se prihvataju ako je I bit postavljen na 0 i ako su odgovarajući biti (Tr za tajmer i Tl za terminal) postavljeni takođe na 0.

Uzeta je pretpostavka da stek pokazuje na posledlju zauzetu lokaciju, da na početku vrh steka pokazuje na adresu 0xff00 i da stek raste ka nižim adresama.

# TESTOVI

Svi testovi su linkovani sa stdo.txt koji predstavlja svojevrsnu biblioteku. stdo.txt sadži tabelu prekidnih rutina, prekidne rutine, pomoćne funckije getchar, printstr koje se mogu pozvati iz korisničkog programa, simbole dataOut i dataIn na čijim lokacijama se nalaze vrednosti u memoriji koje odgovaraju dataOut i dataIn registrima. Za sve testove je dostupan link.log i exec.log u kojima se nalazi rezultat linkovanja i tok izvršavanja programa. Program emulatora pokrenut sa prvim parametrom false nema exec.log fajl.

## Test 1 (petlja)

Asembliranje: ./as test1.txt test1o.txt

Izvršavanje: ./emu true –place=.ivTable@0x0000 –place=.text@0x1000 test1o.txt stdo.txt

Test prikazuje:

* Korišćenje direktive .section
* Registarsko direktno adresiranje
* Memorijsko direktno adresniranje
* Registrasko indirektno adresiranje
* Neposredno adresiranje vrednosti 1
* Neposredno adresiranje vrednosti simbola label
* Definisanje simbola pomoću .equ direktive
* Definisanje simbola pomoći labele
* Korišćenje instrukcije sa operandima od jednog bajta
* Korišćenje registra r1 kao osmobitnog registra
* Aritmeriče operacije
* Instrukciju uslovnog skoka
* Povratak iz potprograma
* Korišćenje .data sekcije
* Korišćenje .word direktive
* R\_16 relokacija
* Korišćenje .global direktive
* -place argumente

Program u deset iteracija na lokacije 0x00f0 do 0x00fa upisuje vrednosti od 10 do 1.

## Test 2

Asembliranje: ./as test2.txt test2o.txt

Izvršavanje: ./emu true –place=.ivTable@0x0000 –place=.text@0x0064 –place=.data@0x0200 test2o.txt stdo.txt

Test prikazuje:

* Korišćenje direktive .skip
* Apsolutno adresiranje podatka na adresi 0x0200
* Pomeračku instrukciju
* Registrasko indirektno adresiranje gde se kao pomeraj koristi vrednost simbola
* Push i pop instrukcije
* Korišćenje specijalnog imena registra (PSW)
* Relativno adresiranje simbola
* R\_PC16 relokaciju

Program rezerviše mesta za dve reči u .data sekciji. Pošto je .data sekcija na adresi 0x0200, dohvatanjem podatka sa te adrese dohvata se zavravo vrednost 1 koja je prethodno inicijalizovana u .data sekciji. Nakon pomeranja u levo, na adresu 0x0202 se upisuje vrednost 2 (dva bajta alocirana .skip naredbom). Nakon toga se uz pomoć push i pop naredbi upisuje vrednost 2 u PSW registar.

## Test 3 (Hello, world!)

Asembliranje: ./as test3.txt test3o.txt

Izvršavanje: ./emu true –place=.ivTable@0x0000 test3o.txt stdo.txt

Test prikazuje:

* Korišćenje dataOut registra
* Linkovanje bez –place argumenata (osim jednog koji obezbeđuje korektnost pri izvršavanju)
* Korišćenje .extern direktive

Program ispisuje pozdrav Hello, world! tako što se upisuju vrednosti ASCII karaktera u dataOut registar.

## Test 4 (tajmer i terminal)

Asembliranje: ./as test4.txt test4o.txt

Izvršavanje: ./emu true –place=.ivTable@0x0000 test4o.txt stdo.txt

Test prikazuje:

* Korišćenje tajmera, tj. podešavanje da se oglašava nakon dve sekunde
* Postavljanje nove prekidne rutine koja će se izvršavati umesto prekidne rutine tajmera
* Instrukcija povratka iz prekidne rutine - iret
* Čitanje iz dataIn registra i upis u dataOut registar
* Poziv printstr potprograma koji ispisuje string koji mu se prosledi
* Poziv getchar potprograma koji blokira program i čeka da se unese karakter sa tastature.

Program umesto prekidne rutine tajmera postavlja novu prekidnu rutinu koja nakon svake dve sekunde ispisuje *timer interrupt...*. Glavni deo programa zatim učitava znak po znak sa tastature i ispisuje ga na ekran, sve dok korisnik ne unese slovo *e*. Nakon toga se ispisuje string *End!*.

## Test 5 (int instrukcija)

Asembliranje: ./as test5.txt test5o.txt

Izvršavanje: ./emu true –place=.ivTable@0x0000 test5o.txt stdo.txt

Test prikazuje:

* Korišćenje int instrukcije
* Korišćenje and instrukcije

Program prvo postavlja bit Tr na 1 i tako maskira prekide od tajmera, a u timer\_cfg registar se upisuje 7. Postavlja se nova prekidna rutina u ulaz 4. A zatim se sofverskim prekidom poziva prekidna rutina u ulazu 4 i izvršava se newRoutine. Zatim se softverskim prekidom poziva i prekidna rutina od tajmera. Tajmer na lokaciji čije je ime currentTime vodi evidenciju o proteklom vremenu. Proteklo vreme se može dobiti tako što se vrednost na toj lokaciji pomnoži sa 500 i dobija se broj milisekundi koliko je proteklo od uključivanaj sistema. Prekidna rutina tajmera čitanjem vrednost timer\_cfg registra misli da je proteklo 60s i dodaje na currentTime 120. ASCII kod 120 odgovara karakteru *x*, tako da je taj karakter ispisuje na ekran upisom u r0 registar (dataOut je zapravo mapiran na r0).

## Test 6 (deljenje nulom)

Asembliranje: ./as test6.txt test6o.txt

Izvršavanje: ./emu true –place=.ivTable@0x0000 test6o.txt stdo.txt

Test prikazuje:

* Korišćenje aritmetičke operacije
* Izvršavanje programa nakon što se dogodi deljenje nulom – poziv prekidne rutine u ulazu 1

Program učitava u registre r1 i r2, 2 i 0, respektivno i pokušava deljenje nulom. Poziva se prekidna rutina u ulazu 1, ispisuje se poruka *Error while executing. Halting.* i završava izvršavanje programa.

## Test 7 (direktiva .equ)

Asembliranje: ./as test7const.txt test7consto.txt

./as test7main.txt test7maino.txt

Izvršavanje: ./emu true –place=.ivTable@0x0000 test7consto.txt test7maino.txt stdo.txt

Test prikazuje:

* Definisanje konstanti direktivnom .equ i njihovo korišćenje u drugom fajlu
* Prepravku koju linker radi kako bi se koristila vrednost simbola kao neposredna vrednost
* Korišćenje negativnih brojeva

Program u odgovarajuće registre učitava vrednosti simbola definisanih preko .equ direktive i ispisuje karakter *C* na ekranu. Iako se u izlaznom main fajlu na mestu korišćenja simbola kodira memorijsko direktno adresiranje, linker radi prepravku jer se radi o simbolu definisanom preko .equ direktive i kaže da se radi o neposrednom adresiranju i upisuje vrednost simbola koja je apsolutna (vrednost simbola se kodira direktno u instrukciji kao neposredna veličina).

## Test 8 (prava pristupa)

Asembliranje: ./as test8.txt test8o.txt

Izvršavanje: ./emu true –place=.ivTable@0x0000 test8.txt stdo.txt

Test prikazuje:

* Korišćenje direktive .byte
* Skok na kod koji nije napisan u sekciji koja ima pravo izvršavanja
* Izvršavanje prekidne rutine u ulazu 1 i zaustavljanje procesora

Program u .data sekciji kodira sledeće instrukcije:

mov r0, 0x0045

ret

Main program skače na ovaj kod u .data sekciji i pokušava da ga izvrši. Program se zaustavlja jer sadžaj u .data sekciji nema pravo izvršavanja i tu se poziva prekidna rutina u ulazu 1 i ispisuje poruka *Access not allowed. Halting.* nakon čega se procesor zaustavlja.

## Test 9 (std)

Asembliranje: ./as std.txt stdo.txt

Izvršavanje: -

Test prikazuje:

* Korišćenje .word symbol
* Korišćenje registarskog indirektnog adresiranja sa osmobitnim pomerajem
* Korišćenje registra sp sa specijalnim imenom (sp[4])
* Implementaciju getchar i printstr potprograma
* cmp instrukciju
* Uslovne skokove (na određeni deo koda)

Ovaj test nije moguće izvršavati jer iako postoji simbol \_start, nije definisan simbol main. Izlaz asemblera se kasnije može linkovati sa ostalim objektnim fajlovima i izvršavati. definisan je simbol \_start od koga počinje izvršavanje programa i tada se poziva potprogram main. Definisane su i prekidne rutine tajmera, terminala, kao i prekidna rutina koja se izvršava kada dođe do neregulatnosti pri izvršavanju programa.

## 3.10 Test 10 (.bss)

Asembliranje: ./as test10.txt test10o.txt

Izvršavanje: ./emu true –place=.ivTable@0x0000 test10o.txt stdo.txt

Test prikazuje:

* Korišćenje direktive .align
* Korišćenje direktive .bss
* Korišćenje imena sekcije kao simbola
* Registarsko indirektno adresiranje sa osmobitnim pomerajem i registarsko indirektno adresiranje bez pomeraja (različito se kodira, iako se isto piše).

Program alocira neinicijalizovan prostor u memoriji od 4 bajta, zatim se vrši poravnanje tako da brojač lokacija bude deljiv sa 8, tako da se ukupno zauzima 8 bajtova. Na te zauzete lokacije se zatim upisuju reči 1, 2, 3 i ponovo 1.

## 3.11 Test 11 (negativne vrednosti)

Asembliranje: ./as test11.txt test11o.txt

Izvršavanje: ./emu true –place=.ivTable@0x0000 test11o.txt stdo.txt

Test prikazuje:

* Korišćenje instrukcije not
* Korišćenje negativnog broja u instrukciji
* Korišćenje instrukcije xchg
* Korišćenje PC relativnog adresiranja kod istrukcije sa jednim operandom
* Postojanje dve različite sekcije sa mogućnosti izvršavanja
* Aritmetiku sa negativnim operandima
* Korišćenje donjih 8 bita registra u instrukciji sa jednobajtnim operandom

Program u registre r1 i r2 upisuje dva negativna broja, zatim zamenjuje vrednosti u registrima, negira donjih 8 bita u registru r2, sabira vrednosti iz registara i smešta rezultat u registar r1.

# UPUTSTVO ZA POKRETANJE

Kako bi se virtuelna mašina pripremila za rad, potrebno je izvršiti komande iz ubuntu.txt fajla. Nakon što su sve komande izvršene, instaliran je g++ 6. Komande se mogu videti i na [linku](https://ovenproof-linux.blogspot.com/2016/09/upgrade-gcc-and-g-in-ubuntu.html?fbclid=IwAR3ukxyhKBFdI_IYl_M-dgBEWTrFGZ28h-e5UpjtsgKGGddAZy0zQLd8cek).

Izvorne fajlove prekopirati na Desktop i pokrenuti Terminal. U Terminalu izvršiti sledeće komande:

cd Desktop

cd as

cd Debug

make

Nakon izvršenih komandi, u folderu Debug će se naći izršni program as. Isti postupak ponoviti i za emulator, samo umesto as, treba napisati emu.