**DVOPROLAZNI ASEMBLER**

**(2-pass assembly)**

Autor: Strahinja Janjic

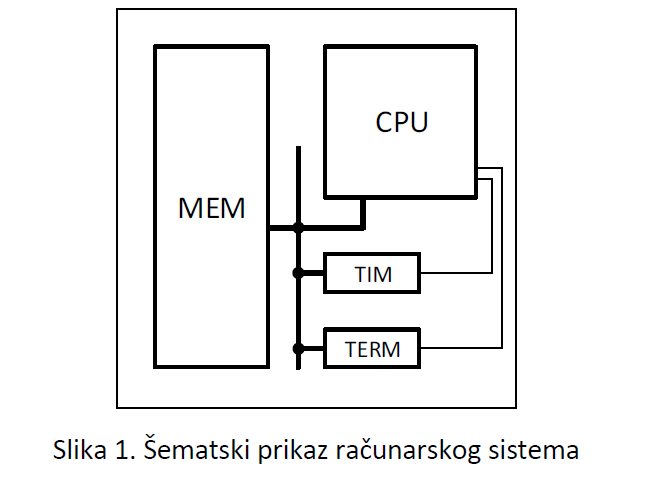
Index: 0411/2015

**UVOD**

U prilogu je data realizacija jednostavnog dvo prolaznog asemblera namenjen za računarski sistem koji ce biti opisan u nastavku. Ulaz asemblera je tekstualni fajl sa sintaksom koja će takodje biti objasnjena u daljem tekstu a izlaz je tekstualni predmetni program jako sličan elf formatu. Izlaz je takodje moguce dobiti i u binarnoj formi koja se može učitati i izvrsiti na emulatoru računarskog sistema.

**OPIS RAČUNARSKOG SISTEMA**

Računarski Sistem se sastoji od procesora, operativne memorije, tajmera I terminala. Sve komponente računarskog Sistema su međusobno povezane preko sistemske magistrale. Tajmer I terminal, kao periferije, su povezani sa procesorom I preko linija za slanje zahteva za prekid. Slika 1. Predstavlja uprošćen šematski prikaz posmatranog računarskog Sistema.

****

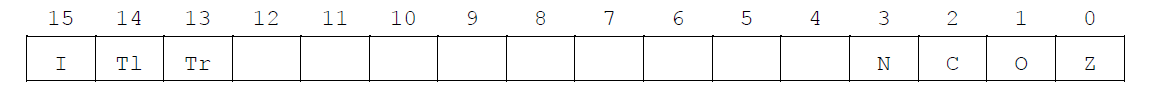
**OPIS PROCESORA**

U nastavku je opisan deo 16-bitnog dvoadresnog procesora sa Von-Neuman arhitekturom. Adresibilna jedinica je jedan bajt, a raspored bajtova u reči je little-endian. Veličina memorijskog adresnog prostora je 216B. Počev od adrese *0xFF00* memorijskog adresnog prostora nalazi se prostor veličine 256 bajtova rezervisan za memorijski mapirane registre (registri kojima se pristupa instrukcijama za pristup memorijskom adresnom prostoru). Počev od adrese *0x0000* memorijskog adresnog prostora nalazi se IVT (interrupt vector table) sa osam ulaza. Svaki ulaz zauzima dva bajta i sadrži adresu odgovarajuće prekidne rutine. Ulazi u IVT odgovaraju sledećim prekidnim rutinama:

* ulaz 0 sadrži adresu prekidne rutine koja se izvršava prilikom pokretanja odnosno resetovanja čitavog procesora (ne izvodi se kompletna sekvenca obrade prekida već se samo vrši skok na adresu koja se nalazi u okviru datog ulaza),
* ulaz 1 sadrži adresu prekidne rutine koja se izvršava ukoliko se pokuša izvršavanje nekorektne instrukcije (nepostojeći operacioni kod, neispravan način adresiranja itd.),
* ulaz 2 sadrži adresu prekidne rutine koja se izvršava kada stigne zahtev za prekid od tajmera (opis principa rada tajmera i način njegove konfiguracije dat je u zasebnom poglavlju),
* ulaz 3 sadrži adresu prekidne rutine koja se izvršava kada stigne zahtev za prekid od terminala (opis principa rada terminala dat je u zasebnom poglavlju) i
* ostali ulazi su slobodni za korišćenje od strane programera.

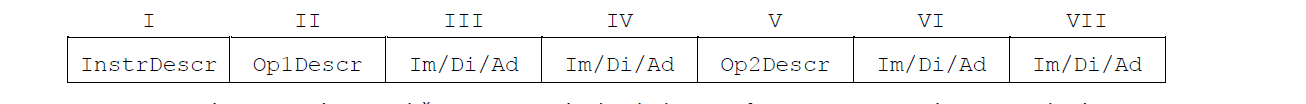
Procesor poseduje osam opštenamenskih 16-bitnih registara označenih sa *r<num>* gde *<num>* može imati vrednosti od nula do sedam. Moguće je zasebno koristiti viših ili nižih osam bita svakog od opštenamenskih registara kao 8-bitni registar označen sa *r<num>h* ili *r<num>l*, respektivno. Registar r7

se koristi kao pc registar (pokazuje na instrukciju koja se u memoriji nalazi neposredno iza trenutno izvršavane instrukcije). Registar r6 se koristi kao sp registar (pokazuje na zauzetu lokaciju na vrhu steka, a stek raste ka nižim adresama). Pored opštenamenskih registara postoji psw registar (statusna reč procesora).

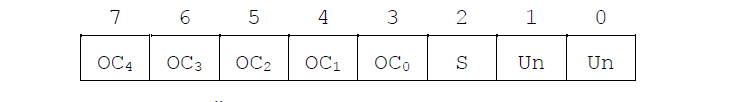


Značenje flegova u psw registru:

* Z (Zero) - rezultat prethodne operacije je nula,
* O (Overflow) – prekoračenje,
* C (Carry) - prenos,
* N (Negative) - rezultat je negativan,
* Tr (Timer) - maskiranje prekida od tajmera (0 - omogućen, 1 - maskiran),
* Tl (Terminal) - maskiranje prekida od terminala (0 - omogućen, 1 - maskiran) i
* I (Interrupt) - globalno maskiranje spoljašnjih prekida (0 - omogućeni, 1 - maskirani).

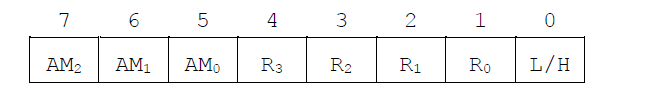
Instrukcije mogu biti veličine od jedan do sedam bajtova. Instrukcija u najopštijem slučaju ima sledeći format:

Prvi bajt instrukcije sadrži operacioni kod i dodatne informacije o instrukciji. Naredni bajtovi instrukcije koriste se za kodiranje operanada. Pojedinačni operand može zahtevati jedan, dva ili tri bajta za kodiranje u zavisnosti od načina adresiranja. Detaljan opis InstrDescr i Op<num>Descr bajtova instrukcije dat je u nastavku.



Značenje bitova InstrDescr bajta instrukcije:

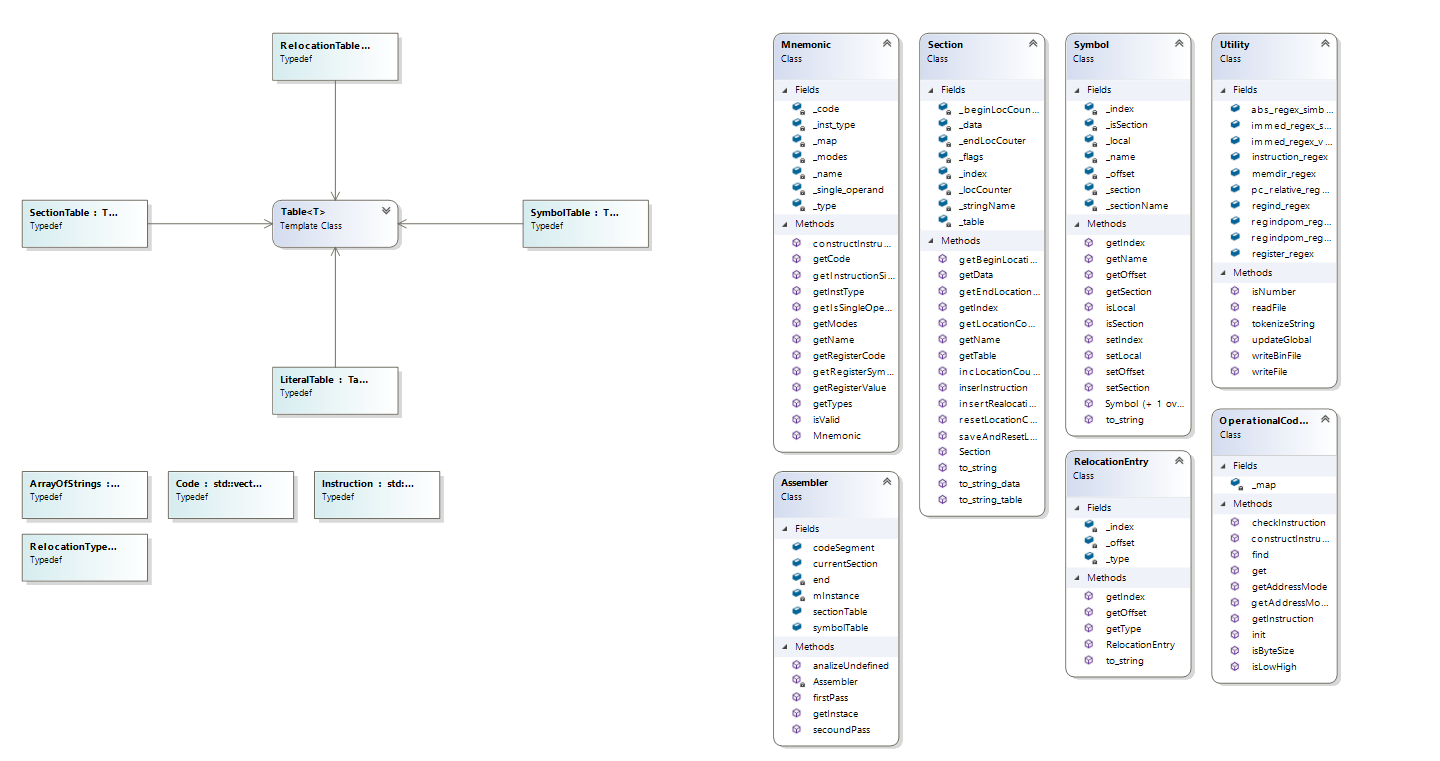
* OC4OC3OC2OC1OC0 - operacioni kod instrukcije,
* S (Size) - veličina operanada instrukcije (0 - jedan bajt; 1 - dva bajta) i
* Un (Unused) - neiskorišćeni bitovi koji imaju fiksnu vrednost nula.



Značenje bitova Op<num>Descr bajta instrukcije:

* AM2AM1AM0 - kodiran način adresiranja pri čemu adresiranje može biti:
* 0x0 - neposredno; vrednost operanda je kodirana u okviru instrukcije pomoću jednog ili dva Im/Di/Ad bajta u zavisnosti od veličine operanda; neposredno adresiranje nije validan način adresiranja za destinacioni operand,
* 0x1 - registarsko direktno; vrednost operanda nalazi se u registru čiji je broj kodiran u okviru instrukcije (nema Im/Di/Ad bajtova),
* 0x2 - registarsko indirektno bez pomeraja; vrednost operanda nalazi se u memoriji na adresi ukazanoj vrednošću registra čiji je broj kodiran u okviru instrukcije (nema Im/Di/Ad bajtova),
* 0x3 - registarsko indirektno sa 8-bitnim označenim pomerajem; vrednost operanda nalazi se u memoriji na adresi ukazanoj zbirom vrednosti registra, čiji je broj kodiran u okviru instrukcije, i vrednosti koja se nalazi u jednom Im/Di/Ad bajtu,
* 0x4 - registarsko indirektno sa 16-bitnim označenim pomerajem; vrednost operanda nalazi se u memoriji na adresi ukazanoj zbirom vrednosti registra, čiji je broj kodiran u okviru instrukcije, i vrednosti koja se nalazi u dva Im/Di/Ad bajta i
* 0x5 - memorijsko; vrednost operanda nalazi se u memoriji na adresi ukazanoj vrednošću koja se nalazi u dva Im/Di/Ad bajta,
* R2R1R0 - kodiran broj korišćenog registra (psw registar se kodira vrednošću 0xF) i
* L/H (Low/High) - naznaka da li se koristi nižih ili viših osam bita registra (0 - nižih; 1 - viših) u slučaju registarskog direktnog adresiranja za operand veličine jednog bajta.

**DIZAJN ASEMBLERA**

****

Sam asembler funkcioniše tako što se pozivaju metode firstPass() i secoundPass() iz glavne (singleton) klase Assembler. U prvom prolaz se formiraju tabele sekcija, i tabele simbola. Tabela sekcija sadrzi ime, velicinu, pocetak i kraj, samo masinski kod sekcije, kao i relokacionu tabelu (Odnosno listu reklokacionih zapisa za sve simbole koji se nalaze unutar te sekcije). Tabela simbola sadrzi ime, offset, sekciju kojoj simbol pripada za svaki simobl na koji se naidje tokom parsiranja koda.

Sve table unutar projekta realizovane su template klasom Table u kojoj su definisane stadnardne metode koje omogucavaju rad sa standardnom bibliotekom.

Pored ovih klasa postoje i pomocne klase OperationalCodeTable i Mnemonic. Metode iz klase OperationalCodeTable služe za prepoznavanje instrukcije iz stringa koji joj se prosledi, kao i odredjivanje velicine same instrukcije. Klasa Mnemonic sadrizi metode za konstrukciju mašinske instrukcije kao i prepoznavanje adresiranja i kreiranja samih relokacionih zapisa.

Svim ovim klasama se na usluzi nalazi staticka Utility klasa koja sadrzi standardne operacije citanja, tokenizacije, ispisa u tekstualni fajl kao i ispisa u binarni fajl. Takodje u ovoj klasi su definisani regex-i koji se koriste pri prepoznavanju instrukcije adresiranja I velicine operanda.

**SINTAKSA ASEMBLERA**

- jedna linija izvornog koda sadrži najviše jednu asemblersku naredbu/direktivu,

- labela, koja se završava dvotačkom, se može naći na početku linije izvornog koda nakon proizvoljnog broja belih znakova,

- labela može da stoji samostalno, bez prateće asemblerske naredbe/direktive u istoj liniji izvornog koda,

što je ekvivalentno tome da stoji u liniji izvornog koda sa prvim sledećim sadržajem,

- simboli se izvoze i uvoze navođenjem asemblerskih direktiva .global <ime\_simbola> i .extern <ime\_simbola>, respektivno, pri čemu u okviru jedne direktive može da se navede i više simbola razdvojenih zapetama,

- direktiva ekvivalencije .equ <ime\_simbola>, <vrednost> uvodi simbol sa datom vrednošću,

- celokupan izvorni kod je podeljen na sekcije definisane pomoću sledećih asemblerskih direktiva: .text – sekcija sa mašinskim kodom,

* .text – sekcija sa mašinskim kodom,
* .data – sekcija za podatke sa inicijalnim vrednostima,
* .bss – sekcija za podatke bez inicijalnih vrednosti,
* .section <ime\_sekcije> [, "<flegovi>"] – sekcija sa navedenim proizvoljnim imenom (datu asemblersku direktivu je moguće navesti više puta sa različitim imenom),

- fajl sa izvornim kodom se završava (ostatak fajla se odbacuje tj. ne prevodi se) pomoću asemblerske direktive .end,

- asemblerske direktive .byte, .word, .align i .skip imaju identične funkcionalnosti kao u okviru GNU asemblera,

- ostatak sintakse, ukoliko nije definisan u prilogu, definisati po sopstvenom nahođenju.

**INSTRUKCIJE**

halt

xchg dst, src

int dst

mov dst, src

add dst, src

sub dst, src

mul dst, src

div dst, src

cmp dst, src

not dst

and dst, src

or dst, src

xor dst, src

test dst, src

shl dst, src

shr dst, src

push src

pop dst

jmp dst

jeq dst

jne dst

jgt dst

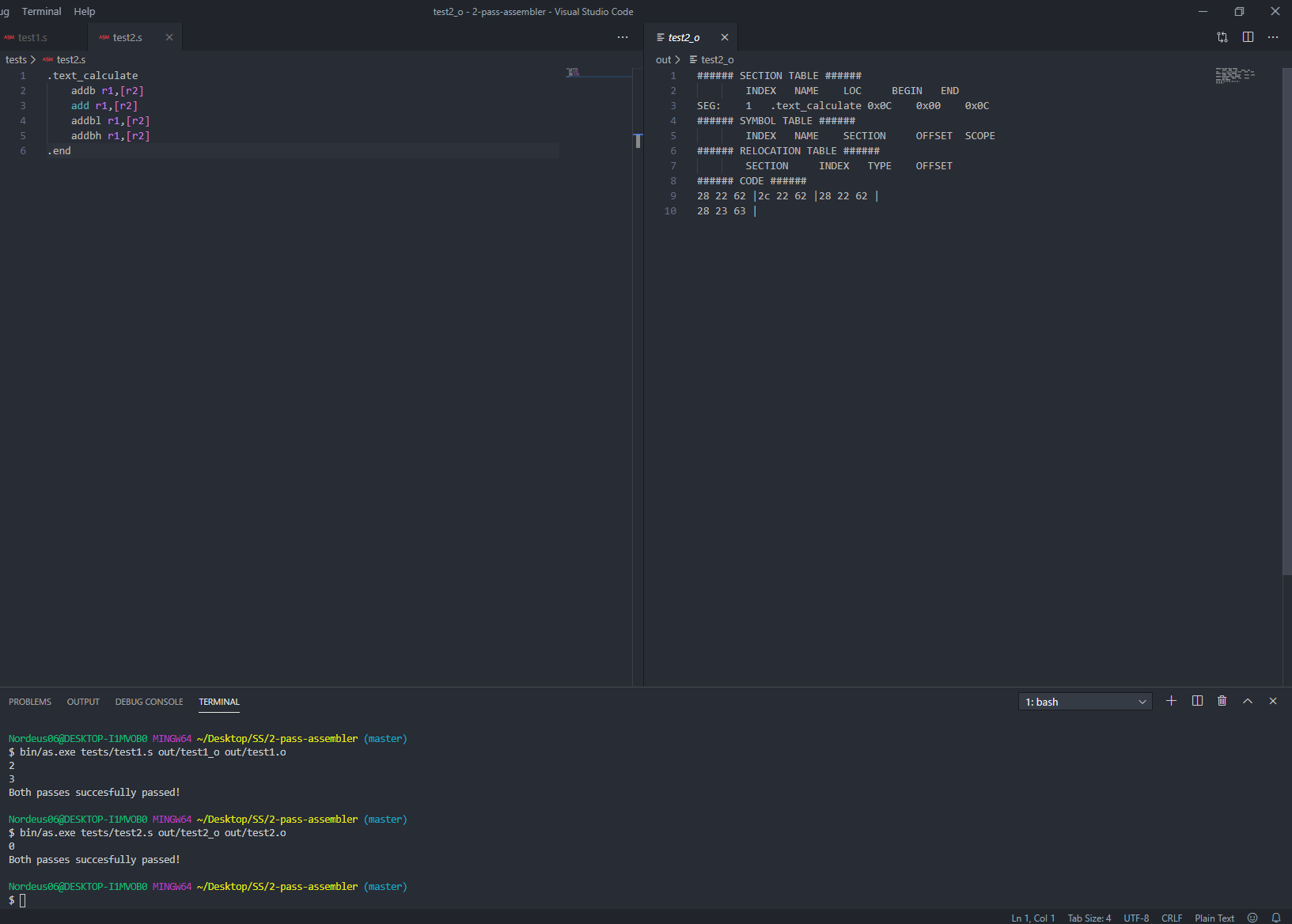
call dst

ret

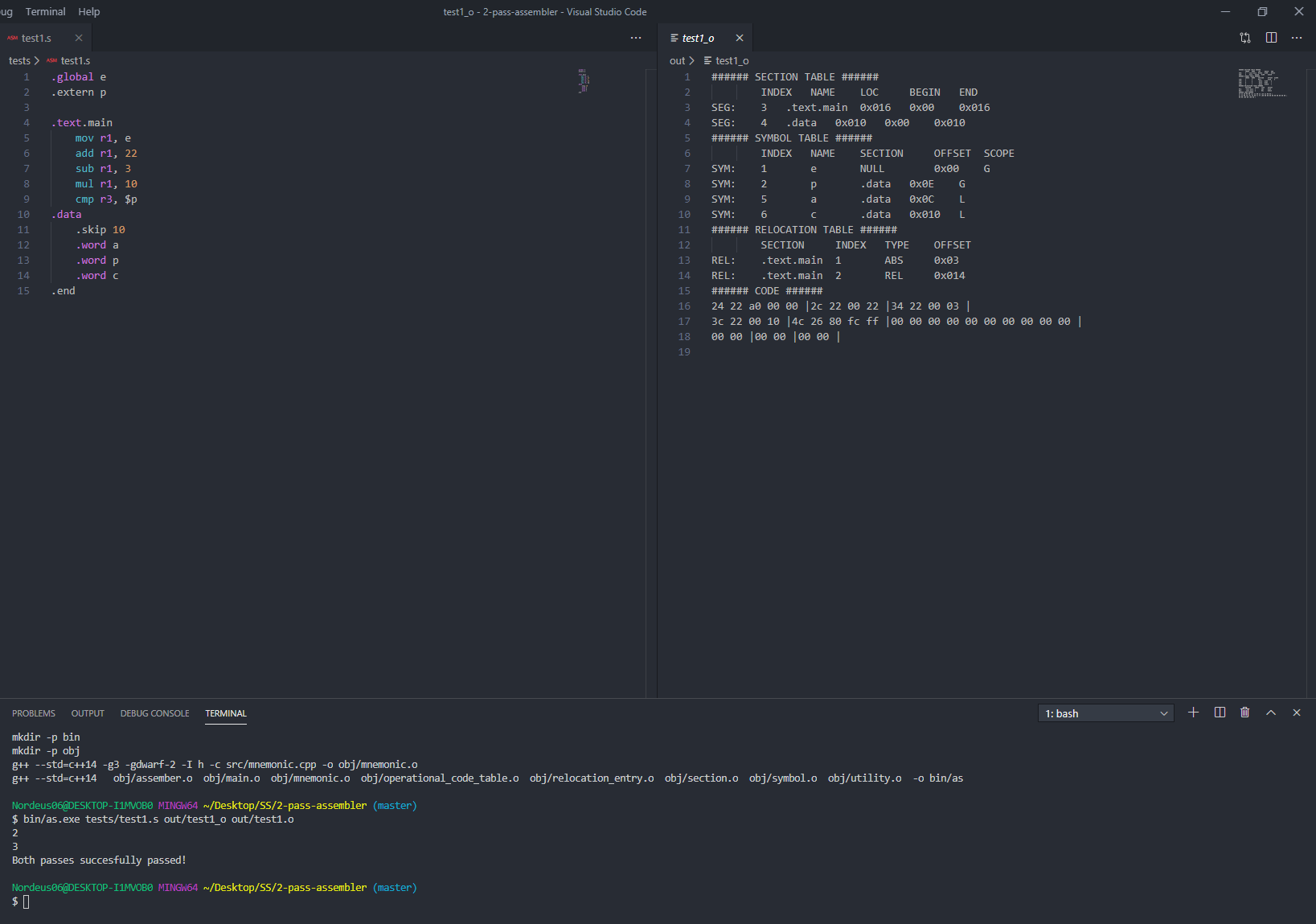
iret

* sve aritmetičke operacije se izvode tako da odgovaraju označenim celim brojevima,
* iza mnemonika asemblerske naredbe, bez belih znakova, može se navesti sufiks b ili w kako bi se naznačila veličina operanada date instrukcije,
* nstrukcije cmp i testnigde ne čuvaju direktni rezultat odgovarajuće operacije, već samo u skladu sa rezultatom postavljaju nove vrednosti flegova u psw registru i
* kombinacije instrukcija i operanada, za koje ne postoji razumno tumačenje, smatrati greškom.
* Sintaksa operanada u okviru asemblerskih naredbi:
* <val> - neposredno adresiranje vrednosti <val>
* &<symbol\_name> - neposredno adresiranje vrednosti simbola <symbol\_name>
* r<num> - registarsko direktno adresiranje
* r<num>[<val>] - registarsko indirektno sa označenim pomerajem
* r<num>[<symbol\_name>] - registarsko indirektno sa označenim pomerajem
* $<symbol\_name> - pc relativno adresiranje simbola <symbol\_name>
* <symbol\_name> - apsolutno adresiranje simbola <symbol\_name>
* \*<val> - apsolutno adresiranje podatka u memoriji na adresi ukazanoj vrednošću <val>

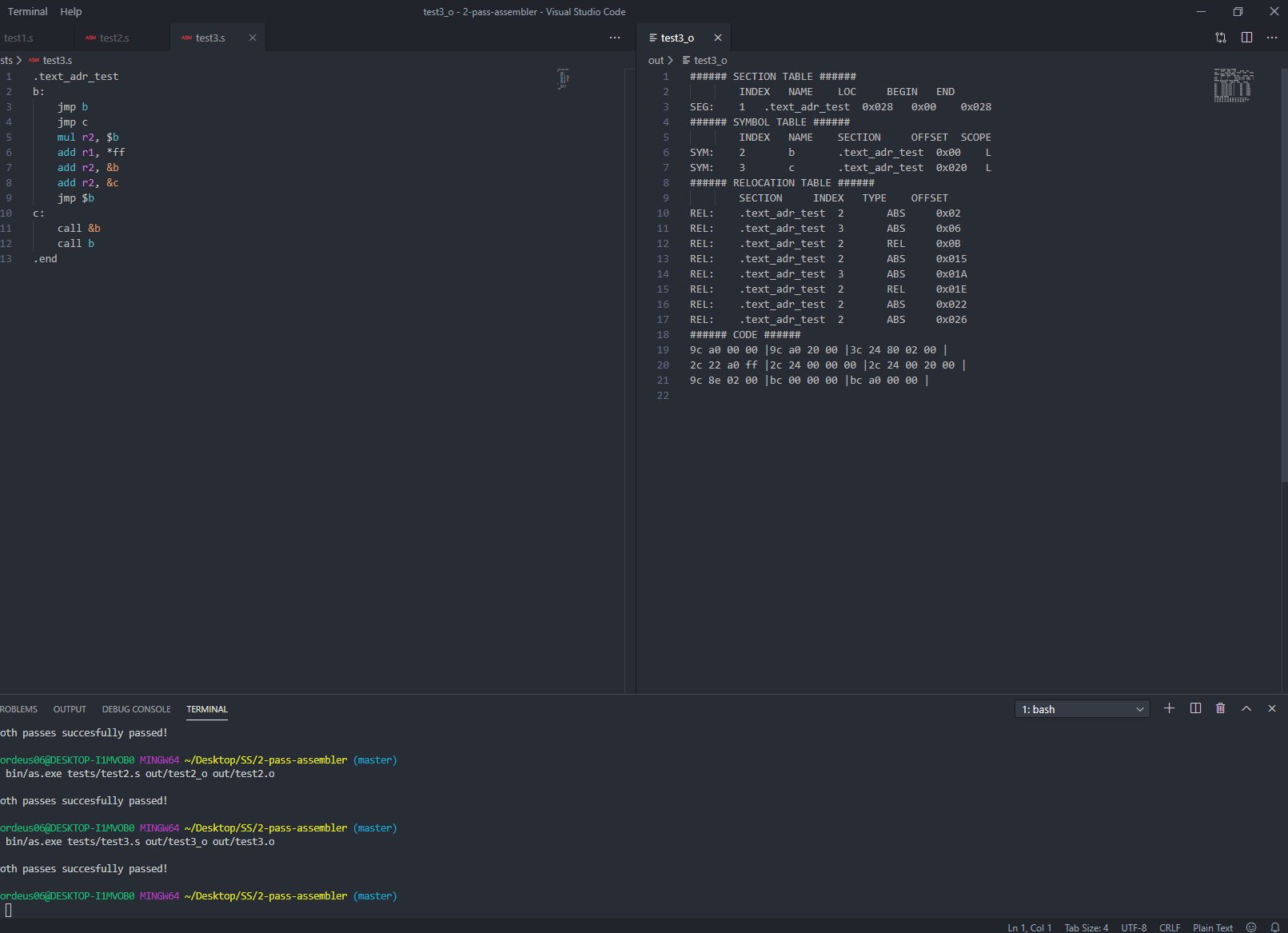
**TEST 1**

Prvi test testira standardne operacije sa jednostavnim adresiranjima i globalnim i ekstrenim simbolima.

**TEST 2**

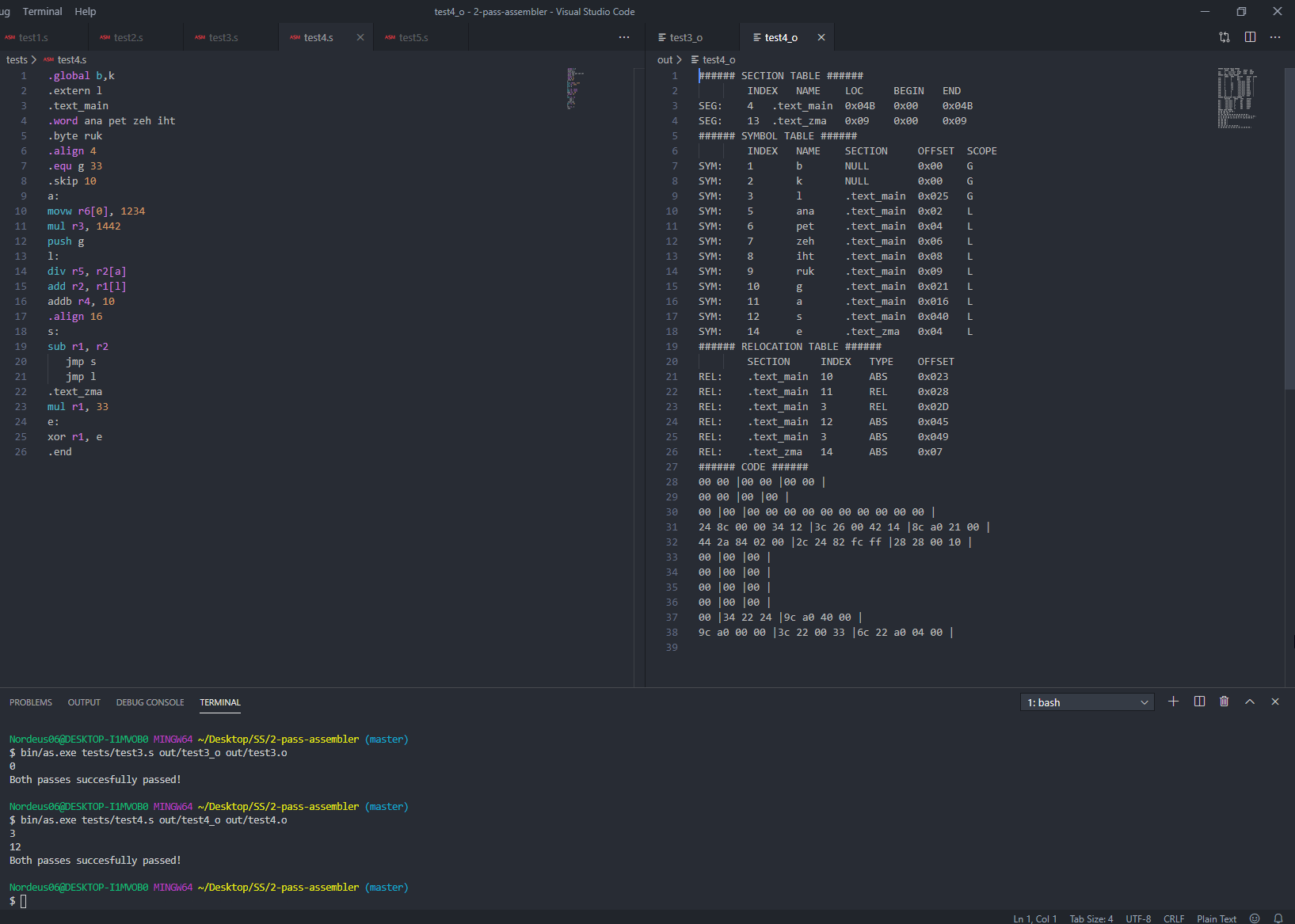
****Drugi test testira generisanje koda pri različitim širinama podataka i smestanjima na više i nize adrese

**TEST 3**

Treći test testira adresiranja i kao i kreiranje tabele realokacija pri tim različitim adresiranjima.

**TEST 4**

Ovo je jedan veći primer koda u dve sekcije gde imamo razne primere adresiranja kao i sve vec vidjenje operacije na jednom mestu.



**TEST 5**

Ovaj test je primer asemblerskog koda koji ima dve funkcije jednu za sabiranje drugu za oduzimanje.

