Projektni zadatak iz Sistemskog softvera

Dokumentacija

Andrija Cicović, 0007/2015

Odsek za softversko inženjerstvo
Elektrotehnički fakultet
Univerzitet u Beogradu
2018.

Sadržaj

1 Opis problema

1.1 Zahtevi

1.1.1 Zadatak 1

Potrebno je napisati dvoprolazni asembler za procesor opisan u nastavku. Ulaz asemblera je tekstualni fajl sa izvornim kodom, u skladu sa sintaksom opisanom u nastavku. Izlaz asemblera treba da bude predmetni program zapisan u tekstualnom fajlu, po uzoru na predmetni program prikazan na vežbama. Dozvoljeno je dodatno generisati i binarni fajl kao izlaz asemblera. Prilikom generisanja izlaznog fajla voditi se principima koje koristi GNU asembler, s tim da se sekcije smeštaju jedna za drugom, počev od adrese koja se zadaje iz komandne linije prilikom pokretanja asemblera.

1.1.2 Zadatak 2

Potrebno je napisati interpretativni emulator za procesor opisan u nastavku. Ulaz emulatora je izlaz asemblera, pri čemu je moguće zadati veći broj predmetnih programa koje je potrebno učitati, povezati i pokrenuti. Nazivi predmetnih programa se zadaju kao argumenti komandne linije. Dozvoljeno je i odvojeno napisati linker, tako da linker radi povezivanje predmetnih programa, a emulator kao argument prihvata jedan fajl koji treba da pokrene. Emulacija je moguća samo ukoliko nakon učitavanja i povezivanja nema preklapanja između učitanih fajlova, nema nerazrešenih simbola niti višestrukih definicija simbola i postoji definisan simbol START, koji definiše početnu adresu prilikom izvršavanja.

1.2 Opis okruženja

Projekat se radi pod operativnim sistemom Linux, na x86 arhitekturi, koristeći programske jezike *C*, *C++* ili asemblerski jezik (moguća je i kombinacija). Potrebni alati za programiranje su opisani u materijalima sa predavanja i vežbi.

1.3 Opis asemblerskog jezika

Za sintaksu asemblera važi:

- U jednoj liniji može biti najviše jedna komanda (instrukcija ili direktiva).
- Na početku svake linije može da stoji labela koju prati dvotačka (‘**:**’).

- Labela može da stoji i u praznoj liniji, što je ekvivalentno kao da stoji u liniji sa prvim sledećim sadržajem.
- Simboli se uvoze i izvoze tako što se na početku fajla navede direktiva `.global <ime_globalnog_simbola>, ...`
- Izvorni kod je podeljen u sekcije:
  - `.text` – sekcija koja sadrži mašinski kod
  - `.data` – sekcija koja sadrži inicijalizovane podatke
  - `.rodata` – sekcija koja sadrži inicijalizovane podatke koje nije dozvoljeno menjati
  - `.bss` – sekcija koja sadrži neinicijalizovane podatke
- Svaka sekcija se u fajlu može pojaviti najviše jednom.
- Fajl sa izvornim kodom se završava direktivom `.end`. Ostatak fajla se odbacuje.
- Direktive `.char`, `.word`, `.long`, `.align` i `.skip` imaju iste funkcionalnosti kao u GNU asembleru.

Dodatno:

- Komentar do kraja linije počinje znakom tačkazapeta (‘`;`’).

Sintaksa operanada instrukcija i argumenata direktiva:

- Neposredna vrednost 20: 20
- Vrednost simbola x: &x
- Memorijsko direktno adresiranje (labela x): x
- Vrednost sa lokacije sa adresom 20: *20
- Direktno registarsko adresiranje: r3
- Registarsko indirektno adresiranje sa pomerajem: r3[20]
- Registarsko indirektno adresiranje sa pomerajem: r3[x]
- PC relativno adresiranje sa pomerajem x: $x

1.4 Opis procesora

1.4.1 Osnovne informacije

- 16-bitni procesor sa Von-Neumann arhitekturom
- Ortogonalna, RISC arhitektura
- Adresibilna jedinica je bajt
- Raspored bajtova u reči je *little-endian* (na nižoj adresi je niži bajt)
- Veličina adresnog prostora je $2^{16}B$
- Osam 16-bitnih opštenamenskih registara, označenih sa r0...r7, pri čemu se r7 koristi kao PC (*program counter*), a r6 kao SP (*stack pointer*)
- Pored opštenamenskih registara postoji i 16-bitni registar PSW (*program status word*)
- Stek raste ka nižim adresama, a SP pokazuje na zauzetu lokaciju na vrhu steka
- Statusna reč PSW u najniža 4 bita sadrži flegove (redom od najnižeg bita):
  - Z – rezultat prethodne operacije je jednak nuli
  - O – rezultat prethodne operacije izazvao prekoračenje
  - C – rezultat prethodne operacije ima prenos
  - N – rezultat prethodne operacije je negativan
- Najviši bit u PSW služi za globalno maskiranje prekida

1.4.2 Prekidi

Počev od adrese 0 nalazi se IVT (*interrupt vector table*) sa 8 ulaza. Svaki ulaz zauzima 2 bajta i sadrži adresu odgovarajuće prekidne rutine.

- Na ulazu 0 se nalazi adresa prekidne rutine koja se izvršava prilikom pokretanja, odnosno prilikom resetovanja procesora.
- Na ulazu 1 se nalazi adresa prekidne rutine koja se izvršava periodično sa periodom od jedne sekunde. Prekidna rutina na ulazu 1 se izvršava periodično samo ako je bit 13 u PSW registru postavljen na vrednost 1, u suprotnom se ne izvršava.

- Na ulazu 2 se nalazi adresa prekidne rutine koja se izvršava prilikom pokušaja izvršenja nekorektne instrukcije.
- Na ulazu 3 se nalazi adresa prekidne rutine koja se izvršava kada se pritisne neki taster, i tada je potrebno očitati kod pritisnutog tastera.

Ostali ulazi su slobodni za korišćenje od strane programera.

1.4.3 Rad sa periferijama

Poslednjih 128 bajtova adresnog prostora rezervisano je za periferije. Na adresi 0xFFFE nalazi se registar podataka izlazne periferije. Upisom *ASCII* koda na adresu 0xFFFE na ekranu se na tekućoj poziciji ispisuje odgovarajući znak, osim u slučaju upisa vrednosti 0x10, kada se tekuća pozicija kursora premešta na početak sledećeg reda. Na adresi 0xFFFC nalazi se registar podataka ulazne periferije. Kada se pritisne neki taster, *ASCII* kod pritisnutog tastera se upisuje u ovaj registar, nakon čega sistem generiše prekid.

1.4.4 Instrukcijski skup

Sve instrukcije su veličine 2 bajta, izuzev instrukcija koje u sebi sadrže neposredni podatak, apsolutnu ili PC relativnu adresu, kada je veličina instrukcije 4 bajta.

Sve instrukcije se izvršavaju uslovno, pri čemu je uslov određen sa prva (najviša) 2 bita svake instrukcije. Naredna 4 bita rezervisana su za operacioni kod instrukcije. Ostatak instrukcije podeljen je na dva polja od po 5 bitova – *dst* i *src*. Oba polja se kodiraju na isti način, opisan u nastavku, s tim što **neposredno adresiranje nije validan način adresiranja u polju *dst***. Na mnemonike instrukcija dodaju se sufiksi koji označavaju uslov koji treba da bude ispunjen kako bi se instrukcija izvršila. Najviša dva bita instrukcije kodiraju koji od 4 uslova iz tabele je u pitanju.

Mnemonik	*Op. kod*	*Objašnjenje*	*Flegovi*
add	$0x0$	$dst = dst + src$	Z, O, C, N
sub	$0x1$	$dst = dst - src$	Z, O, C, N
mul	$0x2$	$dst = dst \cdot src$	Z, N
div	$0x3$	$dst = dst \,/\, src$	Z, N
cmp	$0x4$	$dst - src$	Z, O, C, N
and	$0x5$	$dst = dst \,\&\, src$	Z, N
or	$0x6$	$dst = dst \mid src$	Z, N
not	$0x7$	$dst = \sim src$	Z, N
test	$0x8$	$dst \,\&\, src$	Z, N
push	$0x9$	mem[sp −= 2]=*src*	
pop	$0xA$	*dst*=mem[sp], sp += 2	
call	$0xB$	push pc, pc = *src*	
iret	$0xC$	pop psw, pop pc	
mov	$0xD$	$dst = src$	Z, N
shl	$0xE$	$dst <<= src$	Z, C, N
shr	$0xF$	$dst >>= src$	Z, C, N

Tabela 1: Pregled instrukcijskog skupa

Sufiks	*Kod*	*Uslov*
eq	$0b00$	==
ne	$0b01$	!=
gt	$0b10$	> (označeno)
al	$0b11$	true (bezuslovno)

Tabela 2: Pregled uslovnih sufiksa

Dodatne napomene:

- Sve aritmetičke operacije se izvode tako da odgovaraju označenim celim brojevima.
- Operacije pomeranja su aritmetička pomeranja.
- Instrukcije **cmp** i **test** ne čuvaju direktni rezultat odgovarajuće operacije, nego samo u skladu sa rezultatom postavljaju nove vrednosti flegova u PSW.
- Asembler poseduje i pseudoinstrukciju **ret** (povratak iz potprograma) koja se prevodi odgovarajućom **pop** instrukcijom.
- Asembler poseduje i pseudoinstrukciju **jmp** (bezuslovni skok), koja se prevodi instrukcijom **mov** ili **add**, u zavisnosti od tipa adresiranja odredišta.

1.4.5 Načini adresiranja

U poljima od 5 bitova za *dst* i *src* prva (viša) dva bita kodiraju tip adresiranja. U tipovima adresiranja u kojima je potreban registar, preostala 3 bita kodiraju registar. U slučaju da je za adresiranje potreban i podatak koji predstavlja adresu, pomeraj ili neposrednu vrednost, zapisuje se u poslednja dva bajta instrukcije (tada je instrukcija dugačka 4 bajta). U jednoj instrukciji samo jedan operand može da zahteva dva dodatna bajta, u suprotnom je u pitanju greška.

Adresiranje	***Kod***	***Objašnjenje***
Neposredno ili PSW	$0b00$	Ukoliko je u preostala 3 bita zapisano 0x7 koristi se PSW, u suprotnom je u dodatna 2 bajta zapisan neposredni podatak.
Registarsko direktno	$0b01$	Preostala 3 bita kodiraju registar koji se koristi.
Memorijsko direktno	$0b10$	Preostala 3 bita se ne koriste, u dodatna 2 bajta zapisana je adresa.
Registarsko indirektno sa pomerajem	$0b11$	Preostala 3 bita sadrže registar koji se koristi pri adresiranju, a u 2 dodatna bajta zapisan je označeni pomeraj.

Tabela 3: Pregled načina adresiranja

1.5 Napomene

- Ovaj odeljak se najviše sastoji od teksta projektnog zadatka postavljenog na sajt predmeta Sistemski softver, koji važi počev od juna 2018. godine.

2 Opis rešenja

2.1 Asembler

2.1.1 Lekser i parser

Tokom procesa leksičke analize (tokenizacije) lekser klasifikuje tekst izvornog koda u sledeće kategorije tokena:

```
EOF, NEWLINE, COMMENT, DEC_LITERAL, SYMBOL,
INSTRUCTION, DIRECTIVE, REGISTER, COLON, COMMA, PLUS, MINUS,
AMPERSAND, ASTERISK, LEFT_BRACKET, RIGHT_BRACKET, DOLLAR
```

Ukoliko postoje leksičke greške koje onemogućavaju ispravnu tokenizaciju koda, biće prijavljene od strane leksera, nakon čega se obustavlja dalje prevođenje asemblerskog koda. Lekser ume i da prepozna situaciju u kojoj se poslednja linija ulaznog fajla se izvornim kodom ne završava znakom za novi red – tada se automatski dodaje još jedan `NEWLINE` token kako bi svi redovi bili ispravno terminisani.

Nakon tokenizacije, parser započinje proces sintaksne analize tokom koje grupiše tokene u kompleksnije strukture, definisane specifikacijom sintakse asemblerskog jezika. Zahvaljujući relativno jednostavnoj sintaksi, parser je implementiran kao *recursive descent parser*, koji se sastoji od uzajamno rekurzivnih procedura, od kojih svaka parsira jednu sintaksnu konstrukciju. Zbog ovakvog načina parsiranja, implementacija parsera dosta podseća na definiciju jezika u nekoj od sintaksnih notacija. Parser će prijaviti prvu sintaksnu grešku na koju naiđe, nakon čega se obustavlja dalje prevođenje.

Definicija asemblerskog jezika u EBNF sintaksnoj notaciji:

```
MODULE = {LINE} EOF .
LINE = [LABEL] [COMMAND] [COMMENT] NEWLINE .
LABEL = SYMBOL ":" .
COMMAND = MNEMONIC [PARAM {"," PARAM}] .
MNEMONIC = (INSTRUCTION | DIRECTIVE) .
PARAM = (LITERAL | SYMVAL | SYMBOL | MEMLIT | REGADR | PCREL) .
LITERAL = [("+"|"-")]DEC_LITERAL .
SYMVAL = "&" SYMBOL .
MEMLIT = "*" LITERAL .
REGADR = REGISTER ["["(LITERAL | SYMBOL)"]"].
PCREL = "$" SYMBOL .
```

2.1.2 Struktura relokativnog objektnog fajla

Asembler kao rezultat prevođenja generiše binarni relokativni objektni fajl, u dobro definisanom formatu. Takođe, postoji opcija i generisanja tekstualnog relokativnog objektnog fajla koji može na čitljiv način da prikaže sadržaj fajla. Format binarnog relokativnog objektnog fajla je sledeći:

1. Tabela simbola zapisana u sledećem formatu:
   (a) 32-bitni neoznačen ceo broj k koji predstavlja broj ulaza u tabeli simbola
   (b) k zapisa (ulaza u tabeli simbola) odgovarajuće veličine koji formiraju tabelu simbola
2. 32-bitni neoznačen ceo broj r koji predstavlja broj tabela sa relokativnim zapisima
3. r tabela sa relokativnim zapisima zapisanih u sledećem formatu:
   (a) 32-bitni neoznačen ceo broj m koji predstavlja broj relokacionih zapisa u tabeli
   (b) 16-bitni neoznačen ceo broj s koji predstavlja broj ulaza u tabeli simbola sekcije na koju se odnosi relokacioni zapis
   (c) m relokacionih zapisa odgovarajuće veličine koji formiraju tabelu relokacionih zapisa
4. Tabela sa zaglavljima sekcija odgovarajuće veličine koja sadrži informacije o broju sekcija n (maks. 4) i 4 zapisa koji daju informacije o broju ulaza sekcije u tabeli simbola, veličini sekcije i naznačenoj adresi za učitavanje sekcije u memoriju
5. n sekcija zapisanih u vidu niza bajtova koje formiraju relokativni objektni kod

Na primeru potprograma napisanog u asemblerskom jeziku koji računa najveći zajednički delilac parametara prosleđenih kroz registre r1 i r2 prikazan je sadržaj binarnog relokativnog objektnog fajla i izgled tekstualnog relokativnog objektnog fajla.

```
;gcd.s - Greatest Common Divisor (GCD)
.text
.global gcd
gcd:
            cmp r2, r1
            jmpeq stop
            jmpgt less
            sub r1, r2
            jmp gcd
less:       sub r2, r1
            jmp gcd
stop:       mov r0, r1
            ret
.end
```

Sadržaj binarnog relokativnog fajla objektnog, prikazan korišćenjem hex. editora `xxd`:

```
00000000: 0500 0000 0000 0000 0000 0000 0000  ..............
0000000e: 0000 0000 0000 0000 0000 0000 0000  ..............
0000001c: 0000 0000 0000 0000 0000 0000 0000  ..............
0000002a: 0000 0000 0100 0100 0100 0000 0000  ..............
00000038: 2e54 4558 5400 0000 0000 0000 0000  .TEXT..........
00000046: 0000 0000 0000 0000 0000 0000 0000  ..............
00000054: 0000 0000 0200 0200 0100 0000 0100  ..............
00000062: 6763 6400 0000 0000 0000 0000 0000  gcd...........
00000070: 0000 0000 0000 0000 0000 0000 0000  ..............
0000007e: 0000 0000 0300 0200 0100 1000 0000  ..............
0000008c: 6c65 7373 0000 0000 0000 0000 0000  less..........
0000009a: 0000 0000 0000 0000 0000 0000 0000  ..............
000000a8: 0000 0000 0400 0200 0100 1600 0000  ..............
000000b6: 7374 6f70 0000 0000 0000 0000 0000  stop..........
000000c4: 0000 0000 0000 0000 0000 0000 0000  ..............
000000d2: 0000 0000 0100 0000 0400 0000 0100  ..............
000000e0: 0000 0000 0400 0100 0100 0000 0800  ..............
000000ee: 0100 0200 0000 0e00 0200 0300 0000  ..............
000000fc: 1400 0200 0100 0000 0100 c3bf c3bf  ..............
0000010a: 1a00 0000 0000 0000 0000 0000 0000  ..............
00000118: 0000 0000 0000 0000 0000 0000 0000  ..............
00000126: c391 4935 c3b0 1600 c2b5 c3b0 1000  ..I5..........
00000134: c385 2ac3 b5c3 b000 00c3 8549 c3b5  ..*........I..
00000142: c3b0 0000 c3b5 09c3 a9c3 a00a       ............
```

Izgled tekstualnog relokativnog objektnog fajla:

```
### SYMBOL TABLE ###
+-------------------------------------------------------+
| INDEX|   NAME|     TYPE|  SECTION|   VALUE|     BIND|
+-------------------------------------------------------+
|     0|       |    UNDEF|        0|       0|    LOCAL|
|   0x1|  .TEXT|  SECTION|      0x1|       0|    LOCAL|
|   0x2|    gcd|   SYMBOL|      0x1|       0|   GLOBAL|
|   0x3|   less|   SYMBOL|      0x1|    0x10|    LOCAL|
|   0x4|   stop|   SYMBOL|      0x1|    0x16|    LOCAL|
+-------------------------------------------------------+
### .TEXT RELOCATION TABLE ###
+-------------------------------------------------------+
| INDEX| RELOCATION_TYPE|    OFFSET|            SYMBOL|
+-------------------------------------------------------+
|     0|  REL_TYPE_ABS16|       0x4|               0x1|
|   0x1|  REL_TYPE_ABS16|       0x8|               0x1|
|   0x2|  REL_TYPE_ABS16|       0xe|               0x2|
|   0x3|  REL_TYPE_ABS16|      0x14|               0x2|
+-------------------------------------------------------+
### SECTION HEADER TABLE ###
+-------------------------------------------------------+
|          SECTION_INDEX|      SIZE|     LOAD_ADDRESS|
+-------------------------------------------------------+
|                    0x1|      0x1a|            0xffff|
+-------------------------------------------------------+
### .TEXT SECTION ###
d1 49 35 f0 16 00 b5 f0 10 00 c5 2a f5 f0 00 00 c5 49
f5 f0 00 00 f5 09 e9 e0
```

Tekstualni izlazni fajlovi su samo reprezentativni, sa svrhom da na čitljiv način prikažu strukturu i sadržaj relokativnih objektnih fajlova. Oni neće biti kasnije korišćeni od strane linkera ili emulatora.

2.1.3 Formiranje tabele simbola - prvi prolaz

Prilikom prvog prolaza asemblera rade se provere na semantičke greške (neodgovarajući broj operanada instrukcija, nepravilni tipovi adresiranja...) i formira se tabela simbola. Ukoliko asembler pronađe semantičku grešku, prijaviće je i obustaviti proces prevođenja.

Direktiva `.global` obrađuje se tokom prvog prolaza, sa ciljem da se obezbedi podrška za deklarisanje simbola globalnim bilo gde u fajlu, ne obavezno pre prve upotrebe, tako da je na početku drugog prolaza tabela simbola u potpunosti izgrađena. Asembler ovde otkriva i greške višestrukog definisanja simbola, što uključuje i višestruko definisanje sekcija, jer nije dozvoljeno da se ista sekcija pojavi više od jednom unutar jednog fajla.

2.1.4 Generisanje koda - drugi prolaz

Prilikom drugog prolaza asemblera formiraju se tabela sa zaglavljima sekcija, relokacione tabele za svaku sekciju na osnovu korišćenih simbola i generiše se relokativni objektni kod na osnovu sadržaja sekcija.

Tabela sa zaglavljima sekcija sadrži informaciju o broju sekcija i 4 zaglavlja (za svaku potencijalnju sekciju po jedno) koja sadrže informacije o ulazu koji sekcija zauzima u tabeli simbola, veličini sekcije i adresi na koju sekcija treba da se učita. Rezervisana vrednost `0xffff` označava da za sekciju nije specificirano od koje adrese se učitava, tako da linker kasnije treba da odredi tu adresu.

Za svaku definisanu sekciju generiše se tabela sa relokacionim zapisima. Za svaki simbol korišćen unutar sekcije za koji asembler ne zna vrednost generisaće se relokacioni zapis u relokacionoj tabeli te sekcije. Svaka relokaciona tabela odnosi se na neku sekciju, a svaki relokacioni zapis (ulaz u tabeli) sadrži informacije o tome na kom pomeraju u odnosu na početak sekcije treba izvršiti relokaciju, tipu relokacije i ulazu u tabeli simbola za simbol na koji se odnosi relokacija. Na pomenuti pomeraj u odnosu na početak sekcije upisuje se označena celobrojna vrednost p umesto vrednosti simbola. Kada linker izgradi sopstvenu tabelu simbola, vrednost p će sabrati sa vrednošću koju izračuna na osnovu tipa relokacije i upisati novodobijenu vrednost na pomeraj zadat u relokacionom zapisu.

Nakon što se generišu tabela sa zaglavljima sekcija, relokacione tabele i relokativni objektni kod, u izlazni fajl se ispisuju svi podaci koji su definisani formatom relokativnog objektnog fajla. Dobijeni izlazni fajl je nakon toga spreman za povezivanje od strane linkera.

2.1.5 Dodatne napomene

- Mnemonici koji predstavljaju instrukcije, direktive i mnemonike su *case-insensitive*. Imena simbola (labela) su *case-sensitive*.
- PC-relativno adresiranje dozvoljeno je koristiti samo u instrukcijama skoka i poziva potprograma.
- Registarsko indirektno adresiranje nije dozvoljeno koristiti u instrukcijama skoka i poziva potprograma (ovo je implicitno moguće jedino korišćenjem PC-relativnog adresiranja).
- Omogućeno je koristiti mnemonike instrukcija bez sufiksa za uslovno izvršavanje, što je ekvivalentno korišćenju mnemonika sa sufiksom za bezuslovno izvršavanje.

- Dodata je pseudoinstrukcija HALT, koja obustavlja izvršavanje programa.
- Korišćenje registarskog direktnog adresiranja u instrukcijama skoka i poziva potprograma je dozvoljeno, pri čemu će u PC biti upisana vrednost sadržana unutar naznačenog registra.

2.2 Linker

2.2.1 Struktura izvršnog fajla

Linker kao rezultat povezivanja generiše binarni izvršni fajl, u dobro definisanom formatu. Takođe, postoji opcija i generisanja tekstualnog izvršnog fajla, koji može na čitljiv način da prikaže sadržaj fajla. Format binarnog izvršnog fajla je sledeći.

1. Tabela simbola zapisana u sledećem formatu:
   (a) 32-bitni neoznačen ceo broj k koji predstavlja broj ulaza u tabeli simbola
   (b) k zapisa (ulaza u tabeli simbola) odgovarajuće veličine koji formiraju tabelu simbola
2. Tabela zaglavlja programa zapisana u sledećem formatu:
   (a) 32-bitni neoznačen ceo broj s koji predstavlja broj ulaza u tabeli zaglavlja programa
   (b) s zapisa o segmentima koda odgovarajuće veličine koji formiraju tabelu zaglavlja programa. Svaki zapis nosi informacije o tome koji ulaz u tabeli simbola ogovara simbolu sekcije (segmenta), adresi učitavanja segmenta i veličini segmenta
3. s segmenata zapisanih u vidu niza bajtova koji formiraju objektni (mašinski) kod

Na primeru potprograma napisanog u asemblerskom jeziku koji računa najveći zajednički delilac parametara prosleđenih kroz registre r1 i r2 i glavnog programa, napisanog u odvojenom fajlu, koji poziva ovaj potprogram, prikazan je sadržaj tekstualnog izvršnog fajla. (izlaza linkera).

```
;main.s - GCD test program
.data
M:              .word 36
N:              .word 27
GCD:            .word
.text
.global gcd
.global START
START:
                mov r1, M
                mov r2, N
                call gcd
                mov GCD, r0
                halt
.end
```

Izgled tekstualnog relokativnog objektnog fajla `main.s`, gde je prilikom asembliranja eksplicitno zadato da sekcije treba učitavati počev od adrese `0x200`:

```
### SYMBOL TABLE ###
+-------------------------------------------------------+
|  INDEX|  NAME|     TYPE|  SECTION|   VALUE|     BIND|
+-------------------------------------------------------+
|      0|      |    UNDEF|        0|       0|    LOCAL|
|    0x1| .DATA|  SECTION|      0x1|       0|    LOCAL|
|    0x2|     M|   SYMBOL|      0x1|       0|    LOCAL|
|    0x3|     N|   SYMBOL|      0x1|     0x2|    LOCAL|
|    0x4|   GCD|   SYMBOL|      0x1|     0x4|    LOCAL|
|    0x5| .TEXT|  SECTION|      0x5|       0|    LOCAL|
|    0x6|   gcd|   SYMBOL|        0|       0|   GLOBAL|
|    0x7| START|   SYMBOL|      0x5|       0|   GLOBAL|
+-------------------------------------------------------+
### .DATA RELOCATION TABLE ###
+-------------------------------------------------------+
|  INDEX| RELOCATION_TYPE|   OFFSET|           SYMBOL|
+-------------------------------------------------------+
+-------------------------------------------------------+
### .TEXT RELOCATION TABLE ###
+-------------------------------------------------------+
|  INDEX| RELOCATION_TYPE|   OFFSET|           SYMBOL|
+-------------------------------------------------------+
|      0|  REL_TYPE_ABS16|      0x2|              0x1|
|    0x1|  REL_TYPE_ABS16|      0x6|              0x1|
|    0x2|  REL_TYPE_ABS16|      0xa|              0x6|
|    0x3|  REL_TYPE_ABS16|      0xe|              0x1|
+-------------------------------------------------------+
```

```
### SECTION HEADER TABLE ###
+-----------------------------------------------------+
|           SECTION_INDEX|       SIZE|     LOAD_ADDRESS|
+-----------------------------------------------------+
|                      0x1|        0x6|             0x200|
|                      0x5|       0x14|             0x206|
+-----------------------------------------------------+
### .DATA SECTION ###
24 00 1b 00 00 00
### .TEXT SECTION ###
f5 30 00 00 f5 50 02 00 ee 08 00 00 f6 08 04 00 d8 e0 10 00
```

Izgled tekstualnog izvršnog fajla dobijenog povezivanjem binarnih relokativnih objektnih fajlova `gcd.o` i `main.o`:

```
### SYMBOL TABLE ###
+-------------------------------------------------------------+
|  INDEX|        NAME|      TYPE|  SECTION|    VALUE|     BIND|
+-------------------------------------------------------------+
|      0|            |     UNDEF|        0|        0|    LOCAL|
|    0x1| .SEGMENT.1|   SECTION|      0x1|   0x100|    LOCAL|
|    0x2|         gcd|    SYMBOL|      0x1|   0x100|   GLOBAL|
|    0x3| .SEGMENT.2|   SECTION|      0x3|   0x200|    LOCAL|
|    0x4| .SEGMENT.3|   SECTION|      0x4|   0x206|    LOCAL|
|    0x5|       START|    SYMBOL|      0x4|   0x206|   GLOBAL|
+-------------------------------------------------------------+
### PROGRAM HEADER TABLE ###
+-----------------------------------------------------+
|      SEGMENT_INDEX|               SIZE|     LOAD_ADDRESS|
+-----------------------------------------------------+
|                0x1|               0x1a|             0x100|
|                0x3|                0x6|             0x200|
|                0x4|               0x14|             0x206|
+-----------------------------------------------------+
### .SEGMENT.1 SECTION ###
d1 49 35 e0 16 01 b5 e0 10 01 c5 2a f5 e0 00 01 c5 49 f5 e0
00 01 f5 09 e9 e8
### .SEGMENT.2 SECTION ###
24 00 1b 00 00 00
### .SEGMENT.3 SECTION ###
f5 30 00 02 f5 50 02 02 ee 08 00 01 f6 08 04 02 d8 e0 10 00
```

Više primera programa napisanih na asemblerskom jeziku može se naći unutar foldera `examples/` koji se nalazi u korenskom direktorijumu projekta.

2.2.2 Prvi prolaz - spajanje objektnih fajlova

Linker u prvom prolazu sastavlja sopstvenu tabelu simbola, koja se sastoji od simbola za sekcije (segmente) i globalnih eksportovanih (izvezenih) simbola svih fajlova koje je dobio kao ulaz, spaja sadržaje sekcija svih fajlova u jedan radni fajl i formira listu relokacionih zahteva sastavljenu čitajući relokacione zapise iz relokacionih tabela pronađenih u ulaznim fajlovima.

Ukoliko je u relokativnom objektnom fajlu za sekciju zapisana adresa od koje je treba učitati, linker će početak sekcije povezati sa tom adresom. U suprotnom, linker učitava sekcije počev od unapred određene adrese, pritom svaku sekciju za koju u relokativnom objektnom fajlu nije navedeno od koje adrese treba da se učita će nadovezati na prethodnu takvu sekciju. Linker na ovaj način vezuje početak svake sekcije za neku adresu, odnosno vrednost simbola koji predstavlja sekciju (vrednost sekcije) postavlja na tu adresu.

Istovremeno, linker ažurira i vrednosti svih globalnih eksportovanih simbola u odnosu na novu vrednost sekcije kojoj simbol pripada. U ovom momentu, ukoliko se dogodi takva situacija za neki simbol, biće prijavljena greška višestrukog definisanja simbola.

Sekcije iz ulaznih fajlova prepisuju se u radni fajl u vidu **segmenata** sa dodeljenom početnom adresom i veličinom. Termin *sekcija* i *segment* se ovde koriste ekvivalentno, s tim što se sekcija odnosi na objektni kod u relokativnom objektnom fajlu a segment na objektni kod u izvršnom fajlu (mašinski kod).

Lista relokacionih zahteva se sastavlja tako što se relokacioni zapisi prepravljaju koristeći ulaze u tabeli simbola linkera, umesto ulaza iz lokalnih tabela simbola za ulazne fajlove.

2.2.3 Drugi prolaz - prepravljanje objektnog koda

U drugom prolazu, linker za svaki relokacioni zahtev pokuša da izvrši prepravku objektnog koda. Ukoliko nije definisan simbol na koji se odnosi relokacije, biće prijavljena greška.

Nakon prepravke objektnog koda, u tabeli simbola proverava se da li je definisan simbol START koji predstavlja adresu prve instrukcije programa. Ukoliko nije, linker prijavljuje grešku.

Nakon toga, za sve segmente koda definisane sekcijama iz ulaznih fajlova proverava se da li postoji preklapanje bilo koja dva segmenta. Ukoliko postoji, linker prijavljuje grešku.

Radni fajl se, na kraju, ispisuje u izlazni fajl, koji je nakon toga spreman da bude učitan od strane emulatora.

2.2.4 Dodatne napomene

- Omogućena je opcija koja linkeru nalaže da ignoriše adrese za učitavanje sekcija zapisane u relokativnim objektnim fajlovima. U ovom slučaju, linker samostalno raspoređuje sekcije po memoriji počevši od unapred određene adrese.

- Linker je u stanju da prepozna situaciju u kojoj je segmet previše veliki da bi se uspešno učitao od zadate početne adrese (nema dovoljno memorije do kraja adresnog prostora). U ovom slučaju biće prijavljena greška i prekinuće se proces povezivanja.

- Implementacija linkera očekuje na ulazu ispravne, neoštećene binarne relokativne objektne fajlove. Provere validnosti sadržaja ulaznih fajlova nisu realizovane.

2.3 Emulator

2.3.1 Rad emulatora

Emulator čita binarne izvršne fajlove generisane od strane linkera i svaku instrukciju sprovodi kroz četiri faze: *fetch*, *decode*, *execute* i *interrupt*. Pored emuliranja rada procesora, emulira se i memorija, rad ulaznog i izlaznog uređaja, i hardverskog tajmera.

Emulator iz ulaznog fajla najpre pročita tabelu simbola i tabelu zaglavlja programa. Nakon toga, na osnovu zapisa iz tabele zaglavlja simbola, čita segmente koda iz ulaznog fajla i učitava ih u emuliranu memoriju počev od zadate adrese.

Adresa prve instrukcije koja se upisuje u PC je vrednost simbola START koju emulator traži u tabeli simbola. Emulator zatim ciklično izvršava sledeći niz akcija:

1. Dohvatanje instrukcije (*fetch*): Sa adrese upisane u registar PC čita se instrukcija odgovarajuće veličine i smešta se u jedan ili dva instrukcijska registra veličine 2 bajta.

2. Dekoduju se operandi instrukcije (*decode*), i po potrebi se adresa operanda u emuliranoj memoriji zapisuje u registar MAR (*memory address register*). Ukoliko je detektovana nevalidna instrukcija, biće generisan prekid.

3. Na osnovu instrukcijskog koda (nad operandima) se izvršava instrukcija *execute*, i po potrebi upisuje rezultat.

4. Ispituje se da li je došlo do prekida (*interrupt*). Ukoliko jeste, poziva se odgovarajuća prekidna rutina, pod uslovom da prekidi nisu maskirani, i da je pokazivač na prekidnu rutinu validan (nije *nullpointer*).

5. Ukoliko je rezultat upisan u memoriju na adresu registra izlaznog uređaja, šalje se signal izlaznom uređaju da je novi podatak za ispis spreman.

6. Ispituje se ulazni uređaj radi provere da li je spreman novi ulazni podatak. Ukoliko jeste, biće generisan prekid.

7. Ispituje se tajmer radi provere da li je došlo do periodičnog tajmerskog prekida.

Ukoliko je pročitana instrukcija koja u programskoj statusnoj reči postavlja indikator završetka programa (pseudoinstrukcija HALT u asemblerskom jeziku), obustavlja se emuliranje programa.

2.3.2 Dodatne napomene

- Implementacija emulatora očekuje na ulazu ispravan, neoštećen binarni izvršni fajl. Provera validnosti sadržaja ulaznog fajla nije realizovana.

- Podsistem za zaštitu memorije ne postoji. Program ima punu slobodu da u bilo koju lokaciju memorije upiše bilo koju vrednost.

- Ne postoji podsistem za pamćenje prekida. Uvek će biti obrađen samo poslednji generisan prekid u slučaju da se generiše više prekida u kratkom vremenskom intervalu.

3 Uputstvo za prevođenje izvornog koda

Kako bi se projekat uspešno preveo, potrebno je najpre preuzeti izvorni kod sa GitHub repozitorijuma kom se može pristupiti preko sledećeg URL-a: https://github.com/cicovic-andrija/ETF-System-Software Jedan od načina na koji se može preuzeti izvorni kod je korišćenjem alata `git`, unutar željenog direktorijuma:

```
$ git clone https://github.com/cicovic-andrija/ETF-System-
    Software.git
```

Ukoliko ne postoji, treba napraviti direktorijum `bin/` unutar `home/` direktorijuma tekućeg korisnika:

```
$ mkdir -p ~/bin
```

Program se prevodi upotrebom alata `make`. Izvršavanjem komande:

```
$ make
```

prevode se asembler, linker, emulator od strane alata `gcc`, i generiše se dokumentacija od strane alata `pdflatex`. Izvršni fajlovi alata smeštaju se u direktorijum `./bin/` i kopiraju u `home/bin/` direktorijum tekućeg korisnika (pod pretpostavkom da postoji). Dokumentacija u PDF formatu smešta se u direktorijum `./doc/pdf/`.

Moguće je i svaku komponentu prevesti pojedinačno upotrebom jedne od narednih komandi:

```
$ make assembler
$ make linker
$ make emulator
$ make doc
```

`Makefile` unutar korenskog direktorijuma projekta je *top-level* makefile, koju upravlja makefile-ovima unutar direktorijuma `assembler/`, `linker/`, `emulator/` i `doc/`. Izvršavanjem komande:

```
$ make
```

unutar jednog od ovih direktorijuma prevodi se odgovarajuća komponenta projekta.

Pravilo `clean` definisano je unutar svakog makefile-a. Upotrebom ovog pravila brišu se neželjeni fajlovi unutar stabla direktorijuma projekta.

Nakon prevođenja, alati su spremni za upotrebu od strane programera. Ukoliko se asembler ili linker pozovu uz korišćenje opcije `-h` biće prikazano kratko korisničko uputstvo. Emulator se poziva sa jednim ulaznim fajlom; ukoliko se pozove bez ulaznih fajlova, ispisaće kratko korisničko uputstvo.