Rajković Aleksa 248|2016

SISTEMSKO PROGRAMIRANJE | ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU 2019.

Dvoprolazni asembler i interpreter dizajnirani za zamišljen 16-bitni procesor

Realizacija domaćeg zadatka iz sistemskog programiranja 2019.

Sadržaj

[Uvod 1](#_Toc11100253)

[Asembler 2](#_Toc11100254)

[Proces obrade ulaznog fajla: 2](#_Toc11100255)

[Emulator 3](#_Toc11100256)

[Proces rada emulatora: 3](#_Toc11100257)

[Organizacija memorije: 5](#_Toc11100258)

[O prekidnim rutinama i IVT tabeli: 5](#_Toc11100259)

[Greške koje sistem prepoznaje i prijavljuje: 5](#_Toc11100260)

[Implementacija tajmera: 6](#_Toc11100261)

[Implementacija terminala: 6](#_Toc11100262)

[Uptutstvo 7](#_Toc11100263)

[Instalacija 7](#_Toc11100264)

[Asembler 7](#_Toc11100265)

[Emulator 7](#_Toc11100266)

[Testovi 8](#_Toc11100267)

[1-AssBasics 8](#_Toc11100268)

[2-EQU and Relocs 8](#_Toc11100269)

[3-HelloWorld 8](#_Toc11100270)

[4-DeepLinking 8](#_Toc11100271)

[5-CircDefinition 8](#_Toc11100272)

[6-SecParams 8](#_Toc11100273)

[7-Timer 8](#_Toc11100274)

[8-Kybrd 8](#_Toc11100275)

[9-15 (MulDefBAD, NoDefBad, ImmAdrBAD, StackOverflowBAD, StackUnderflowBAD, SegFaultBAD, AritFaultBAD) 8](#_Toc11100276)

[16-ProgExample 8](#_Toc11100277)

# Uvod

Projekat se bavi problematikom konstrucije asemblera i interpretera kao i njihovom implementacijom za datu zamišljenu mašinu, te je i isam podeljen na dve logičke u potpunosti modularne celine: **asembler** i **emulator**. Sve odluke o dizajnu asemblera formirane su prateći nastavni materijal predmeta kao i zvaničnu dokumentaciju GNU asemblera. Podržana sintaksa instrukcija u potpunosti je pokrivena zadatim dokumentom domaćeg zadatke, sintaksa podržava sve opisane instrukcije, direktive i modele.

# Asembler

Asembler je konstruisan kao dvoprolazni asembler koji razrešava dati kod i formira jedinstveni izlazni tekstualni fajl nalik na ELF formatu u kome se nalaze:

1. Tabela simobla – konstrusiana tako da obuhvati sve simbole koje obrađivani kod koristi i koju će linker da upotrebi kako bi formirao globalnu tabelu simbola i razreši relokacije.
2. TNS *Tabela Nedefinisanih Simbola* – konstruisanu tako da obuhvati sve simbole definisane .equ direktivom koje asembler nije mogao da razreši te ih ostavlja emulatoru
3. Tabela relokacija – konstruisana tako da emulatoru da sve potrebne informacije o zadatim relokacijama na nivou posmatranog fajla
4. Sekcije – konsturisano tako da emulatoru pruži čist sadržaj bajtova enkapsulirane sekcije unutar fajla

U izlaznom fajlu se sve gorepomenute tabele mogu jasno pregledati uz dodatno heksadecimalnu reprezentaciju svih sekcija ulaznog fajla.

Greške u sintaksi ili dvosmisleno definisane linije asmebler prijavljuje kao izuzetak i obustavlja proces prevođenja.

## Proces obrade ulaznog fajla:

1. Tokenizacija – asembler sve do kraja fajla ili do pojave .end direktive formira tokene liniju po liniju pritom ignorišiću komentare (#), sažimajući sve tabulacije u razmak a sve višestruke razmake u jedan.
2. Prvi prolaz – formira tabelu simbola tako što indeksira nove pojava istih u okviru kojih pokušava i da razreši apsolutne .equ simbole. One koje ne uspe stavlja u TNS tabelu.
3. Proverava sve TNS i Reloc. ulaze. Ako navedeni ulazi koriste simbol koji nije deklarisan prijavljuje grešku. Takođe i razrešava lokalno definisane - apsolutne .equ simbole koji su međusobno upleteni.
4. Drugi prolaz – puni sekcije po formiranom sadržaju i proverava da li postoji instrukcija koja koristi nedeklarisani simbol. Ako neka instrukcija koristi apsolutno adresiranje simbola ili pak relativno adresiranje simbola iz neokružujuće sekcije formira se odgovarajući relokacioni ulaz. Ako neka direktiva (.byte ili .word) obuhvata izraz koji koristi adresne simbole on se takođe ubacuje u tabelu relokacija.
5. Formira izlazni fajl nalik na ELF-u.

# Emulator

Emulator je dizajniran kao interpretativni emulator zadate mašine. Za datu listu fajlova i opcionih parametara početnih adresa sekcija, emulator započinje povezivanje objektnih fajlova delegirajući ovaj posao instanci klase Linker. Linker uspešno povezuje fajlove akko:

1. Nema nerazrešenih simbola
2. Nema dvostrukih definicija
3. Nema ciklične definicije
4. Ima definicije globalnog \_start simbola tj. ulazne tačke programa
5. Ima definicije prekidnih rutina

Emulator pre nego što pozove Linker prevodi /usr/bin/IVTSetup.s fajl i podmeće ga u proces povezivanja kako bi \_start proglasio globalnim i uvezao prekidne rutine. Ovaj fajl se mora naći u pomenutoj putanji pre pokretanja emulacije, inače se proces obustavlja. Korisniku je dozvoljeno da podrazumevani IVTSetup.s fajli iz project/bin foldera promeni i instalira novi pozivanjem skripte project/install.sh. Ovime se odgovornost za ispravno funkcionisanje sistema, tj. IVT tabele, delegira korisniku. Kao što je u postavci navedeno ulazima 0,1,2,3 odgovaraju prekidne rutine startovanja procesa, rutine grešaka sistema, tajmera i signala pritiskanovog znaka na tastaturi respektivno, dok su ostali ostavljeni korisniku na korišćenje.

Ako je lanac fajlova prevelik da bi se smestio u prostor dodeljen za kod ili ako je iz bilo kog razloga nemoguće postaviti željenu konfiguraciju adresa parametrizovanih sekcija (preklapanja, zalaženja u zaštićenu zonu), Emulator prijavljuje grešku.

Ako određene sekcije sadrže .align direktivu takvu da se ne poklapa sa poravnanjem parametrizovane početne adrese sekcije, Linker menja zadatu početnu adresu tako da ista bude poravnata sa maksimalnim koeficijentom poravnanja posmatrane sekcije na nivou objektnog fajla i o ovome ne obaveštava korisnika.

## Proces rada emulatora:

1. Pozivanje Linkera za zadatu listu fajlova i parametara, koji dodatno formira i sliku memorije po kojoj će se se program izvršavati :
   1. Razrešavanje globalnih simbola tako što se za svaki globalno definisan simbol traži definicija, ako se ovde naiđe na višestruku definiciju ili se ista ni ne nađe prijavljuje se greška. Ova faza proverava kako i lokalne tabele simbola tako i lokalne TNS tabele.
   2. Razrešavannje sekcija tj. objedinjavanje sekcija istog imena iz svih ulaznih fajlova. Prvo se obrađuju parametrizovane pa one za koje nije navedena početna adresa. Proces alokacije prostora jeste dizajniran po principu parametrizovane kontinualne alokacije prostora, sa dodatnim optimizacijama, koju obavlja klasa MemAllocator. U postavci projekta je navedeno da se neparametrizovane sekcije smeštaju odmah nakon što se postave parametrizovane tj. na najvišu sledeću slobodnu adresu, no ovo direktno utiče na pojavu moguće eksterne fragmentacije prostora te se u cilju smanjivanja fragmentacije prostora, nakon obrade parametrizovanih sekcija, prostot popunjava gde je to moguće bilo na višim ili nižim adresama.  
      Nakon alokacije i konačnog formiranja početnih adresa sekcija, svaka vrednost simbola prethodno definisanog lokalno na nivou fajla sabira se sa adresom sekcije u kojoj je definisan.
   3. Mapiranje dobijenih vrednosti globalnih simbola u sve fajlove koje iste i uvoze.
   4. Razrešavanje TNS tabele, tako što se za svaki ulaz iterira kroz sve ulazne fajlove u kojima se dato ime nalazi. Ako je gledajući lokalnu tabelu simbola, koja sada sigurno ima sve definicje koje su se mogle razrešiti, moguće sračunati ulaz TNS tabele tj. izraz koji je za nju vezan, Linker sračunava TNS ulaz i unapređuje ga u ulaz Tabele simbola. Dodatno ako je dati, sada sračunat, TNS ulaz bio proglašen globalnim na primer :.global a  
      .extern b,c  
      .equ a,b+cOn se ponovo mapira u sve objektne fajlove koji isti i koriste.   
      Iteracije se ponavljaju po istom principu dokle god postoji promena, tj. dokle god je barem jedan TNS ulaz razrešen.  
      Kada se iz petlje izađe zbog nerazrešavanja ni jednog TNS ulaza, proverava se da li su sve TNS tabele sada parzne. Ako nisu ciklična definicija je pronađena bilo na lokalu bilo na nivou upletenih definicija i deklaracija ulaznih fajlova. Linker ovo prijavljuje navodeći listu nerazrešenih jednakosti.
   5. Razrešavanje relokacijonih ulaza svih ulaznih fajlova. Ako je algoritam stigao do ove tačke nju sa sigurnoću i obavlja po standardnim mehanizmima opisanim u materijlima predmeta. Dodatno *školska* reloakcija je apstrahovana *relokacijom izraza*. Ovo je dodato u cilju obrađivaja izraza tipa:  
      .extern arrStructs, sizeOfStruct  
      pointers:

.word arrStructs+0\*sizeOfStruct, arrStructs+1\*sizeOdStruct, arrStructs+2\*SizeOfStruct

* 1. Punjenje alociranog niza od 64KB po formiranim sekcijma.
  2. Traženje \_global ulazne tačke i \_IVT#n prekidnih rutina.

1. Emulator preuzima sliku formirane memorije, puni prvih 16B adresama prekidnih rutina, podešava početnu konfiguraciju tajmera i pokreće osluškujuću demonsku nit tastature .
2. Program uvek otpočinje u prekidnoj rutini #1 gde se za povratne parametre postavljaju : 0 za početni PSW kontekst i &\_start za povratnu adresu. Preczino govoreći emulator postavlja:  
     
   PC = adr(\_start)  
   SP = 0x0010  
   PSW = 0x0000  
     
   U prekidnoj rutine na stek podmeću inicijalne vrednosti povranih parametara.

Organizacija memorije:   
  
IVT\_SEGMENT [0x0000 – 0x000F] 16B  
STACK\_SEGMENT [0x0010 – 0x200F] 8KB ~ 12%64KB  
CODE\_SEGMENT [0x2010 – 0xFEFF] 55KB  
MMREGS\_SEGMENT[0xFF00 - 0xFFFF] 256B

## O prekidnim rutinama i IVT tabeli:

Konstruisano je ukupno osam ulaza ove table tako svaki ulaz odgovara određenoj prekidnoj rutini onako kako je to navedeno u postavci zadatka. Bitno je napomenuti da je kritični čvor projekta fajl IVTSetup.s. On mora postojati u direktorijumu /usr/bin datog okruženja zbog potrebe Emulatora da isti i prevede pre početka povezivanja i emulacije. Ulazna tačka Emulatora (njegov main) sistemskim pozivom system(“assembler -o IVTSetup.o /usr/bin/IVTSetup.s”);  
prevodi tabelu i podmeće je u fazu povezivanja. Ažuriranje IVTSetup.s iz projektnog bin direktorijuma se u /usr/bin direktorijumu ažurira pomoću poziva skripte projcet/bin/install.sh ili jednostavnim kopiranjem.

Prekidna rutina \_IVT0 jeste ona koja se poziva prilikom restovanja procesa tj. njegovog startovanja. I sam Emulator kreće iz ove rutine u kojoj se manipuliše stekom i na nejga postavljaju adresa ulazne tačke i početni kontekst. Opasno je menjati date linije koda u ovoj rutini ili narušavati stek jer će direktno uticati na ispravan rad emulacije.

Prekidna rutina \_IVT1 jeste ona koja se poziva prilikom nailaska na grešku od koje se sitem ne može oporaviti. Ona jednostavno ispisuje #ERROR# i obstavlja izvršavanje. Opasno je menjati sadržaj ove rutine tako da se iz nje može izaći. Ona se uvek poziva pri greškama koje trebaju i završiti izvršavanje.

## Greške koje sistem prepoznaje i prijavljuje:

Emulator sve greške obrađuje i prijavljuje na identičan način. Ispisujući poruku o grešci i ulazeći u ulazeći u prekidnu rutinu jedan:

* Unrecognized Instruction OpCode – neprepoznatljiv bajt koji identifikuje operaciju instrukcije
* Immediate addressing fault – neposredan operand naveden kao destinacija promene
* Stack Overflow/Underflow – pucanje steka
* Segmentation Fault – nelegalan pristup adresi, tj. nekompaktibilan pristup adresi u poređenju sa RWX flegovima
* Arithmetic Fault – deljenje nulom prepoznato.

## Implementacija tajmera:

Pretopstavke su sledeće:

CPU\_SPEED = 200MHz

REG\_ACCESS = 1Clk

MEM\_ACCESS = 5Clk

Za pristup memoriji uzeto je samo 5 ciklusa jer virtuelno postoje samo 2 jednobajtno pristupne periferije pa je arbitraža krajnje jednostavna i brza. Svaki pristup resursima reguliše unutrašnji brojač emulatora koji se na kraju svake instrukcije upoređuje sa nulom u cilju odlučivanja koliko je cikulsa do tog trenutka proteklo i da li je u poređenju sa trenutom vrednošću perioda generisanja prekida tajmera (timer\_cfg) potrebno ući u prekidnu rutinu tajmera (identično kao na materijalima sa predavanja). Dodatno, Emulator pre ulaska u prekid tajmera isti i maskira kako ne bi dozvolio moguće samougnježdavanje ako je iz nekog slučaja rutina tajmera vremenski dovoljno duga da se u nju ponovo uđe. Ovime se sistem štiti od beskonačne petlje.

## Implementacija terminala:

Što se izlaza tiče, čim se pristupi na upis adresi data\_out memorijski mapiranog registra reguliše se reakcijom oblika cout<<mem[data\_out] .

Ulaz je malo složeniji i koristi posebnu demonsku nit čiji je jedini cilj da se blokira na ulaznom toku konzole dok ne pročita char. Nakon toga jednostavno deljenom promenljivom signalizira emulatorskoj niti da je nešto pročitano i da se u prekidnu rutinu odgovarajućeg ulaza treba ući. Naravno demonska nit ostaje blokirane sve dok emulatorska nit ne prihvati preneti char.

# Uptutstvo

## Instalacija

Pre korišćenja bilo koje komande mora se pokrenuti install.sh skripta koja kopra izvršne fajlove iz project/bin foldera u /usr/bin folder. Ovime se omogućava jednostavno pozivanje asemblera i linkera ali isto takoi kopiranje IVTSetup.s fajla. Svaki put kada se želi promeniti konfiguracija IVT tabele nužno je pokrenuti ovu skriput ili jednostavno prekopirati IVTSetup.s u /usr/bin folder.

## Asembler

Asembler se pravilno poziva komandom:

assembler inputFile -o outputFile  
 assembler -o outputFile inputFile  
  
 Bilo koji drugi ili nejednoznačni format biće prijavljen kao greška

## Emulator

Emulator se pravilno poziva komandom:  
  
 emulator inputFile0, [inputFilei]\*, [-place<.sectionName>=0x<hexValue>]\*

Gde se opcijom -place parametrizuje početna adresa navedene sekcije. Bilo koji drug ili nejednoznačni format biće prijavljen kao greška. Izostavljanje liste fajlova biće prijavljeno kao greška.

# Testovi

U folderu project/test dati su svi testovi koji prikazuju funkcionalnosti sistema. Pokretanjem skripti pokreće se navedeni test koji u sebi sadrži detaljan opis demonstriranog.

## 1-AssBasics

Prikazuje proizvod prevođenja ulaznog fajla sa velikim brojem podržanih sintaksnih formi i načina adresiranja. Prikazuje osnovne tabele, kako se one formiraju i kako izgledaju ulazi.

## 2-EQU and Relocs

Detaljno prikazuje kako se formiraju TNS i Reloc. tablea za dati ulazni fajl

## 3-HelloWorld

Klasičan *Pozdrav svetu* program koji ujedno i demonstrira izlaz na terminal ali i pozivanja funkcije definisane u drugom fajlu.

## 4-DeepLinking

Prikazuje šta je u stanju Linker da razreši. Dosta upletene definicije simbola među fajlovima tako da su neke .equ a neke adresne i sve zavisne od nekih osim izlazne tačke razrešavanja.

## 5-CircDefinition

Prikazuje kako Linker(Emulator) reaguje na pojavu ciklične definicije.

## 6-SecParams

Na jednostavan način koji u kombinaciji upoređivanja adresa simbola i neposrednih vrednosti na ekran štampa parametrizovane adrese sekcija pozvane na određeni način.

## 7-Timer

Testira tajmer rutinu i prikazuje kako korisnik istu može i menjati. Program se javlja sa ispisom *tick!* na svake 10s.

## 8-Kybrd

Testira rutinu ulaza terminala i prikazuje kako korisnik istu može i menjati. Program ispisuje *Key pressed: <char>* akko je val(char)!=val(LINE\_FEED)

## 9-15 (MulDefBAD, NoDefBad, ImmAdrBAD, StackOverflowBAD, StackUnderflowBAD, SegFaultBAD, AritFaultBAD)

Testiraju nevalidne situacije u kojoj se bilo asembler bilo linker bilo emulator mogu naći. Nazivi su samoopisujući. Očekivani izlasi su prijavljivanja nekorektnih stanja.

## 16-ProgExample

Prikazuje rad malo složenijeg programa. Koji za lokalno alocirani niz bajtova *ThisIsNotAnagram* poziva i relativno i apsolutno funkcije iz međusobno različitih fajlova. Prvo poziva *printLine* koji štampa alociran niz sve do pojave terminišućeg znaka, onda poziva *invert* funkciju koja dati niz obrće u redosledu, pa ponovo *printLine* kako bi ispisao modifikovan niz.