Sistemski softver

Projekat: Dvoprolazni asembler

Lazar Vasić 2013/0298

septembar 2016

Opis problema

Od ulaznog fajla koji sadrži sintaksu sličnu kao kod gnu asemblera, trebamo da dobijemo izlazni fajl koji sadrzi tabelu simbola, table realokacija i izgled kodiranog dela (ELF) u memoriji.
Ovaj postupak se radi u 2 prolaza. U prvom formiramo tabelu simbola koju čine lokalne i globalne promenljive i sekcije. Dok u drugom prolazu kodiramo instruckije i pravimo tabele realokacija za svaku sekciju.

Resenje:

U startu programa , se vrsi formatiranje ulaznog texta, brisu se komentari, prazni redovi… fajl se cita do .end direktive, sve nakon toga se nece tretirati da postoji. Svaka procitana linije se cuva u vektor stringova i salje se Instruction_Parseru na parsiranje (prvi i drugi prolaz). Svaki red se cita i svaka reč se proverava da li je ona sekcija memonik , labela , ili nesto drugo. U slučaju da nije nista od onoga sto nam treba u program ta reč/ linija se ignorise. Svaka instrukcija i direktiva imaju svoju sintaksu i u slucaju da nije upisan odgovarajuci program javice se izuzetak i ispistai u terminalu/konzoli koja instrukcija je bacila koji izuzetak. U projektu postoje dosta izuzetaka u cilju da se lakse popravi greska. Ako postoji.
Sve instruckije se izvode iz klase Instcution, i overriduje metodu read_instuction(). Ovo je vako zamisljeno da bi bilo lakse i preglenije parsiranje instruckija.
Neke instrkcuje mogu da imaju labele kao neki operand (call, ldc, ldch, ldcl i ldr) i u drgom prolazu prilikom nailaska na labale u instrukciji pravice se i neki ulaz u tabele realokacija.
Za tebelu realokacija postoje 5 razlicitih tipova adresiranja, (R_386_PC32, R_386_32, R_386_LONG32 , R_386_H16, R_386_L16), prva 2 se koriste kao I u zadatku 9 za apsolutno I relativno adresiranje, 3. se koristi kod izraza u long direktivi kada imamo sabiranje zato sto ne znamo gde ce punilac da smesti nas program , I poslednja dva adresiranja su apsolutna adresiranja kao R_386_32 jedina razlika je ta sto ovim dajemo linkeru informaciju o tome da li radimo sa visim ili nizim vrednostima (ldch ili ldcl).

Direktiva .long pored svojih vrednosti kao vrednost moze da ima I izraz. Predpostavka je da u izrazu moze da ima 2 operanda koji moraju biti iz istih sekcija , u suprotom ne mozemo da dobijemo relokatibilan izraz. U izrazima se mogu javljati + I – kao operacije.

Instuckija call koristi pc registar ako se to ne navede I kod pc registra se ugradjuje u operacion kod instrukcije.
Kod instkcuija za pomera shl I shr moraju da se navedu << ili >> respektivno kako I neposredna vredsnost za koju se pomera. Greska je ako shl ima >>. Takodje znaci << >> moraju biti odvojeni blanko znakom od registra.

Immed (neposredne) vrednosti se pisu #vrednost. Obavezan je znak #.

Kod instuckija ldr ili str uz registar mozemo da pisemo postfix/prefix inkrement/derkement na prvom registru.

U jednom input fajlu moze da postoji jedna jedinstvena sekcija. Greska je ako imamo dve sekcije sa istim imenom npr. .text.sek I zatim ponovo .text.sek

NAPOMENA: Kada se u direktivama javlajaju vise podataka .byte 1,2,3 ili .long 1213,2131, ne dolazi do greske ali ako ima malo vise podataka npr. Byte 1,2,3,3,4,45,6,6,7 dolazi do pucanja programa I javaljanja greske stack smashed. Ovo se ne tretira kao izuzetak vec se program nasilno prekida. Ako se ovo javi znaci da neka od direktiva ima vise podataka nego sto moze , ovo se resava tako sto se od jedne diretkive naprave vise.

Upustvo za pokretanje

Pošno na virtuelnoj mašini ne postoji instaliran gcc , g++ (g++ nam je potreban) najpre moraju da se dovuku sa internate.
Komande:

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
sudo apt-get remove g++
sudo apt-get install g++-5
sudo ln -s /usr/bin/g++-5 /usr/bin/c++
```

Buildovanje projekta radimo alatom cmake/make, takonje i njega moramo da instaliramo na VM.
Komande

```
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:george-edison55/cmake-3.x
sudo apt-get update
sudo apt-get install cmake
sudo apt-get upgrade // ovo ne mora da se radi jer ce da krene da se ažuriraju svi programi ali je preporucljivo
```

Kada instaliramo sve ovo imacemo potrebne alate da povežemo i pokrenemo projekat.

Upustvo:

1) Kopira se projekat (4 foldera) npr. na destkopu (može i nedge drugde), naci main.cpp i u njemu promeniti output_file i input_file, staviti adresu zeljenog testa.
2) Otvorimo terminal i nadjemo putanju projekta i udjemo u doc folder posto se tu nalazi skripta za pokretanje cmake alata

```
cd /home/laza/Desktop/ss_project/doc/
```

3) u terminalu komadnom

```
mkdir cmake && cd cmake/ && cmake .. && make && ./ss
```

pravimo cmake direktorijum, ulazimo u taj diretkorijum pokrecemo skriptu CmakeList.txt koja se nalazi u doc folderu i radimo make, tj povezujemo sve h i cpp fajlove. Na kraju se komadnom ./ss izvrsava projekat. Izvrsni fajl ss se pokrece za vrednosti output i input fajlova koje su navedene u main.cpp, ako treba da se promeni output i input fajlovi onda se u main.cpp promeni njihova putanja.
4) komandom
gedit ../../input.txt ../../output.txt
ili
emacs ../../input.txt ../../output.txt
ili
nano ../../input.txt ../../output.txt
ili
vi ../../input.txt ../../output.txt

otvaramo input i oputut fajlove, ovi fajlovi treba da budu upareni sa onim fajlovima iz main.cpp (ovde je navedena putanja do tests fajla u kome se nalaze 3 testa (input fajla) i odgovrajuci output fajlovi.

Testovi:

1)input1.txt output1.txt
2)input2.txt output2.txt
3)input3.txt output3.txt

Ovi fajlovi se nalaze u tests folderu. Zbog nekompatibilnosti formata kada se ubacie u .doc dokument tabele ne igzledaju razumljivo. U testovima je u komentarima (oznaka za komentar @) napisano sta koja linija radi.