Jednoprolazni asembler za CISC arhitekturu

Opis rešenja:

Program koristi std::regex heder iz standardne biblioteke kako bi uporedio pročitanu liniju iz izvornog fajla sa predefinisanim regex izrazima, od kojih svaki odgovara jednom tipu naredbe ili direktive koji se mogu naći u izvornom fajlu. Ukoliko ni jedan od ovih izraza ne odgovara pročitanoj liniji, asembler obustavlja rad. Asembler inicijalizuje ove regex objekte u svom konstruktoru.

Program prihvata kao ulazne parametre putanju do fajla koji sadrži izvorni kod napisan u asembleru prema navedenoj arhitekturi, opciju -o, nakon koje se navodi putanja do izlaznog fajla koji ce predstavljati obrađen ulazni fajl. Ukoliko je neka od ovih putanja nevaljana ili je zahtevani format argumenata prekršen, asembler će obustaviti rad. Ove provere vrši metoda argumentsAnalyzer(int, char\*\*).

Asembler čita ulazni fajl red po red i svaki red uparuje sa regex objektima, na osnovu kojih zna šta predstavlja pročitana linija. Kada pronađe odgovarajući regex objekat koji prepoznaje pročitan red, skače na obradu capture grupa koje taj regex objekat definiše. Capture groupe nam omogućavaju da regex objekat ne samo koristimo kao šablon za prepoznavanje tipa pročitanog ulaza, već i da izdvojimo informacije koje su nam od značaja. Dalje su navedeni šabloni regexa koji odgovaraju mogućim tipovima direktiva i instrukcija koji se nalaze u izvornom fajlu, kao i opis podataka koje asembler izvlači iz njih i dalje obrađuje:

* komentar ili prazan red - asembler nastavlja dalje sa čitanjem fajla.
* Labela – vrši se provera da li smo unutar neke sekcije (ili smo u UNDEFINED). Ukoliko smo u sekciji, asembler proverava da li simbol postoji u tabeli simbola ili tabeli literalnih simbola i definiše simbol. Sve potrebne provere i definicije se obavljaju iz metode resolveSymbol(std::string)
* Sekcija – asembler dozvoljava proizvoljne nazive sekcija, osim nekoliko ključnih reči koje su rezervisane za ispravan rad asemblera. Pored standardne obrade novog simbola, takođe menja tekuću vrednost sekcije u kojoj se nalazi, ukoliko je prethodno bio u sekciji, pomoću locationCounter atributa dodeljuje veličinu prethodnoj sekciji.
* Equ direktiva – equ direktiva mora da se pojavi pre ulaska u prvu sekciju koju smo definisali u izvornom fajlu. Simbolički literal definisan equ direktivom se ubacuje u literalTable strukturu, a osim imena, pamti se izraz koji predstavlja vrednost ovog simboličkog literala, a koji će se izračunati nakon prvog prolaza asemblera, a pre početka backpatching-a. Asembler dozvoljava upotrebu operatora + i –, kao i literala i simbola. Nema ograničenja u vidu prekoračenja jedinice mere, tj. Zaštitu od adresne aritmetike koju programer želi da upotrebi.
* Global direktiva – mora se naći pre bilo koje sekcije. Izdvaja listu simbola i obrađuje ih jedan po jedan. Ukoliko navedeni simbol predstavlja simbolički literal ili je definisan kao eksterni simbol, prijaviće grešku. U suprotnom dodaje nov simbol u tabelu simbola, u UNDEFINED sekciji, globalne vidljivosti.
* Extern direktiva – mora se naći pre bilo koje sekcije. Izdvaja listu simbola i obrađuje ih jedan po jedan. Ukoliko navedeni simbol već postoji u tabeli simbola ili tabeli literalnih simbola, prijavljuje grešku. U suprotnom dodaje simbol u tabelu simbola, u UNDEFINED sekciji, eksterne vidljivosti
* Byte direktiva – mora se naći unutar sekcije. Ukoliko postoji labela ispred, simbol se obrađuje kao u slučaju kada imamo samo labelu. Izdvaja listu vrednosti koje će biti ugrađene u kod. Ukoliko nađe upotrebu simbola koji predstavljaju adrese u kodu, simboličkih literala čije su vrednosti veće od 1B ili koji vrše dodatne relokacije adresa, obustavlja rad. U suprotnom računa vrednost i upisuje je u memorijski vektor. Ukoliko je u pitanju literalni simbol, ostavlja njegovo računanje za kasnije i samo dodaje informaciju o lokaciji u odgovarajući ulaz tabele nepotpunih instrukcija (TII)
* Word direktiva – radi slično byte direktivi, sa izmenom što dozvoljava upotrebu simbola koji predstavljaju adrese i simboličkih literala koji su veličine 2 bajta
* Skip direktiva – vrši proveru da li se nalazi u sekciji. Razrešava labelu, ukoliko je ima. Čita literal koji predstavlja koliko bajta treba da se ugradi u memoriju. Sve bajtove inicijalizuje sa 0x90.
* Instrukcije bez operanada – proveravaju da li su u sekciji, razrešava labelu ukoliko postoji. Asembler ima spisak svih mogućih mnemonika koji spadaju u ovu grupu, te izvlači tu informaciju na osnovu odgovarajuće capture grupe, pa koristi prevodilačku tabelu da dobije odgovarajući operacioni kod koji ugrađuje u tekuću sekciju.
* Instrukcije sa jednim operandom - proveravaju da li su u sekciji, razrešava labelu ukoliko postoji. Po istom principu dobija operacioni kod, i izvlači informaciju o operandu, koji je pri uparivanju regexa prošao određenu proveru vezano za format. Pošto su moguće instrukcije koje rade sa operandom (push, pop) i instrukcije koje predstavljaju skok. Kako ove instrukcije imaju različite sintakse pri adresiranju operanda, vrši se dodatna provera u odnosu na mnemonik, na osnovu čega se odlučuje kojim novim regex objektom će se utvrditi način adresiranja i operand koji se koriste u instrukciji.
* Instrukcije sa dva operanda – identično kao instrukcije sa jednim operandom.

Posle obrade ulaznog reda, ukoliko je korišćen neki simbol koji još nije definisan, ili ukoliko se koristi simbolički literal, pravi se nov zapis u odgovarajućem ulazu TII. Ukoliko je poznat simbol, dodaje se odgovarajući zapis o relokaciji.

Kada se dođe do kraja fajla, ili direktive .end, naredni korak je računanje svih vrednosti simboličkih literala koji su navedeni u izvornom fajlu. Izraz koji je korišćen za opis simboličkog literala može da sadrži literale kao i simbole. Asembler ugrađuje vrednosti koje može u vrednost simboličkog literala, a pored toga čuva vektor sa potrebnim relokacijama i njihovim operacijama, koje treba ugraditi na mesto korišćenja simboličkog literala.

Kada su izračunati simbolički literali, asembler može da zakrpi kod pomoću TII. Ključ u TII ulazu je naziv simbola ili simboličkog literala koji je korišćen, dok je vrednost struktura koja čuva vrednost o sekciji, offset-u, operaciji (+ ili -), broju bajtova vrednosti koju ugrađuje i tipu relokacije. Asembler vrši određene provere, da vidi da li se informacija o zakrpi poklapa sa vrednošću koja treba da se ugradi. Nakon toga ugrađuje vrednosti i dodaje potrebne relokacije u relokacionu tabelu.

Asembler kao izlaz pravi fajl .o ekstenzije. Informacije u njemu su sledeće:

* Tabela simbola, sa svim potrebnim informacijama
* Tabela literal, koja sadrži vrednost literalnog simbola i relokacije koje on nosi sa sobom
* Relokacione tabele za svaku sekciju u kojoj postoji barem jedna relokacija
* Vrednosti sekcija, prvo naziv sekcije i broj bajtova, a potom mašinski kod ukoliko sekcija ima sadržaj

U nastavku su dati testovi, prvo ulazni fajl, potom izlaz generisan od strane asemblera:

miniProgram.s

.global \_start

.extern labela1,labela2,exit,arraybase

.equ simbolLiteral,0x1234+labela2-labela1+op1

.text

\_start:

jmp \*main(%pc)

mov $0x0, %r0

call exit

main: push %r5

mov %sp, %r5

mov 0x8(%r5), %r2

mov (%r2), storage

mov op1, %r0

mov $1, %r1

xchg %r0, %r1

add %r1, %r0

call function

pop %r5

ret

function:

mov op1, %r1

shl $2, %r2

loop:

add arraybase(%r2), %r1

sub $1, %r2

test %r2, %r2

jne loop

mov %r1, %r0

ret

.bss

.skip 0x20

.data

op1:.byte 0xff,1

labela5:

.word labela5,0x1234

.word 0xffff,-512

storage:

.word 0

.end

miniProgram.o

%SYMBOL TABLE%

Symbol Symbol number Section Offset Type Size SymbolType

text 6 text 0 local 82 section

bss 12 bss 0 local 32 section

data 13 data 0 local 12 section

loop 11 text 61 local 0 label

labela2 3 UNDEFINED 0 extern 0 label

op1 9 data 0 local 0 label

storage 8 data 10 local 0 label

main 7 text 13 local 0 label

arraybase 5 UNDEFINED 0 extern 0 label

exit 4 UNDEFINED 0 extern 0 label

function 10 text 51 local 0 label

\_start 1 text 0 global 0 label

labela1 2 UNDEFINED 0 extern 0 label

labela5 14 data 2 local 0 label

%EQU SYMBOLS%

Symbol Value Relocations

simbolLiteral 4660 +3 -2 +13

%RELOCATION TABLE% - section data

Symbol number Offset Operation Relocation type

13 2 + R\_16

%RELOCATION TABLE% - section text

Symbol number Offset Operation Relocation type

4 11 + R\_16

13 26 + R\_16

13 30 + R\_16

6 46 + R\_16

13 53 + R\_16

5 63 + R\_16

6 76 + R\_16

.data 12

ff 01 02 00 34 12 ff ff 00 fe 00 00

.bss 32

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

.text 82

28 6e 09 00 64 00 00 00 20 20 00 00 00 48 2a 64

2c 2a 64 6a 08 00 24 64 44 80 0a 00 64 80 00 00

20 64 00 01 00 22 5c 20 22 6c 22 20 20 00 33 00

50 2a 10 64 80 00 00 22 bc 00 02 00 24 6c 64 00

00 22 74 00 01 00 24 b4 24 24 38 00 3d 00 64 22

20 10

randomTest.s

.global a,c

.extern e,f

.equ dvanaest,12

.equ expr,-s+mam-0x1234+10

.data

a:

.skip 0x10

b: .word a,0x4321

.byte 0xff,-1

.word expr

c:

#komentar

.text

start:

push %r5

mov %sp, %r5

test %r0, %r0

d: add dvanaest(%r5), %r0

sub %r0, %r0

call \*start(%pc) #PC relativan skok

int d

jmp \*d

g:

shlb $1, %r0l

shrb %r0h, $1

s:

mam:

.end

randomTest.o

%SYMBOL TABLE%

Symbol Symbol number Section Offset Type Size SymbolType

data 5 data 0 local 24 section

text 7 text 0 local 36 section

a 1 data 0 global 0 label

d 9 text 8 local 0 label

c 2 data 24 global 0 label

e 3 UNDEFINED 0 extern 0 label

f 4 UNDEFINED 0 extern 0 label

start 8 text 0 local 0 label

b 6 data 16 local 0 label

g 10 text 28 local 0 label

s 11 text 36 local 0 label

mam 12 text 36 local 0 label

%EQU SYMBOLS%

Symbol Value Relocations

expr -4650 -7 +7

dvanaest 12

%RELOCATION TABLE% - section text

Symbol number Offset Operation Relocation type

7 22 + R\_16

7 26 + R\_16

%RELOCATION TABLE% - section data

Symbol number Offset Operation Relocation type

1 16 + R\_16

7 22 + R\_16

7 22 - R\_16

.text 36

48 2a 64 2c 2a b4 20 20 6c 6a 0c 00 20 74 20 20

20 6e ec ff 18 00 08 00 28 80 08 00 b8 00 01 20

c0 21 00 01

.data 24

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

00 00 21 43 ff ff d6 ed

testPCrel.s

.global labela5,labela4

.extern labela6

.text

.skip 0x10

labela1:

jmp \*labela2(%pc)

labela4:

labela2:

call \*labela1(%pc)

addw labela5(%r7), labela3(%r7)

subb %r0h, labela6

jgt \*labela4(%pc)

.data

.skip 16

labela3: .word 0x1234

labela5:

.end

testPCrel.o

%SYMBOL TABLE%

Symbol Symbol number Section Offset Type Size SymbolType

text 4 text 0 local 40 section

data 8 data 0 local 18 section

labela5 1 data 18 global 0 label

labela4 2 text 20 global 0 label

labela6 3 UNDEFINED 0 extern 0 label

labela3 7 data 16 local 0 label

labela1 5 text 16 local 0 label

labela2 6 text 20 local 0 label

%EQU SYMBOLS%

Symbol Value Relocations

%RELOCATION TABLE% - section text

Symbol number Offset Operation Relocation type

1 26 + R\_PC16

8 29 + R\_PC16

3 34 + R\_16

.data 18

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

34 12

.text 40

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

28 6e 00 00 20 6e f8 ff 6c 6e fb ff 6e 0e 00 70

21 80 00 00 40 6e ec ff