Sistemski softver

Domaći zadatak: Dvoprolazni asembler

Septembar 2017.

Ivana Vučković 293/2013

Cilj domaćeg zadatka je napraviti dvoprolazni asembler, koji ce od ulaznog fajla cija je sintaksa opisana u tekstu projekta napraviti izlazni predmetni program. Projekat se sastoji iz dve glavne funkcije koje vrse obradu prvog prolaza i drugog prolaza.

U prvom prolazu je cilj pronaci sve simbole u ulaznom tekstualnom fajlu, gde simbol moze biti neka labela, sekcija ili simbol definisan direktivom DEF, uneti ih u tabelu simbola i odrediti kojoj sekciji pripada odredjeni simbol i njegovu vrednost. U prvom prolazu takodje mora da se vrši obrada izraza koji se mogu naći pre DUP direktive i posle DEF direktive zbog validnog racunanja Location Countera i adresa simbola. Takodje imamo i obradu ORG direktive koja predstavlja stvarnu pocetnu adresu nekog segmenta, odnosno sekcije. Jedina obrada instrukcija u prvom prolazu jeste racunanje velicine tih instrukcija jer naš asembler radi sa instrukcijama koje mogu biti velicine 32b ili 64b u zavisnosti od načina adresiranja.

U drugom prolazu se radi obradjivanje pocetnih vrednosti promenljivih, kao i instrukcije detaljno. Tada se pravi tabela zapisa o relokacijama i pise sadrzaj sekcije u kom se mogu naci masinske instrukcije i pocetne vrednosti promenljivih zapisane heksadecimalno. Relokativni zapis za neki simbol pravimo kada nam izraz zavisi od tog simbola. Ukoliko je taj simbol globalni relociramo sam simbol upisujuci redni broj simbola iz tabele simbola, a ukoliko je lokalni, upisujemo sekciju. Razlog tome je sto linker vidi samo globalne simbole dok lokalne ne moze. Tu vodimo računa da li je se radi o apsolutnoj relokaciji ili relativnoj. Kod pc relativnog adresiranja koristimo relativnu relokaciju. Kod apsolutne linker dodaje vrednost simbola koji relociramo, dok kod relativne dodaje vrednost simbola i oduzima vrednost reda u kome je doslo do relokacije, odnosno reda koji je potrebno prepravljati. Kod instrukcija moramo voditi racuna o tipu operanda, o nacinu adresiranja kao i o relokacijama.

Pokretanje projekta: pod Linux operativnim sistemom, najpre je potrebno instalirati g++ na sledeći način:

sudo add-apt-repository ppa:ubuntu-toolchain-r/test

sudo apt-get update

sudo apt-get install gcc-4.9

sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.9 50

sudo apt-get install g++-4.9

sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.9 50

Potom možemo ili napraviti makefile ili iz terminala komandom

g++ -std=c++0x Mape.cpp TabelaSimbola.cpp SimboliUlaz.cpp Obrada.cpp Postfix.cpp RelZapis.cpp Sadrzaj.cpp main.cpp -o prog

prevesti nas kod, a zatim komandom ./Program ulaz.txt izlaz.txt pokrenemo izvrsavanje našeg projekta.

Testovi

Test1

Cilj ovog testa je da pokaze rad sa DUP i DEF direktivama, sekcijama, inicijalizacijom podataka, gde imamo slucajeve sta se sve moze naci kao pocetna vrednost nekog podatka i kako se tretira ako je izraz. Takodje i rad sa nekim aritmetičkim instrukcijama.

.data.1

a DEF 7

b DEF 0x20

.global lab1

lab1:

DW b-a

lab2:

c DEF 0b11

lab3: DB (10-9)\*3 DUP lab1 + 3

.text.1

INT R10

RET

ADD R3, R4, R5

lab4: XOR R1,R6,R14

ASR R2,R3,R4

CALL R6

ORG 0x100

.data

d DEF 10

DW lab2 + lab5-lab1

lab5: DD lab1-9, -2, ?

.bss

DD ?

.end

Izlaz1:

#TabelaSimbola

SEG 0 UND 0 0x0 0x0 L

SEG 1 data.1 1 0x0 0x5 LRW

SYM 2 a -1 0x7 L

SYM 3 b -1 0x20 L

SYM 4 lab1 1 0x0 G

SYM 5 lab2 1 0x2 L

SYM 6 c -1 0x3 L

SYM 7 lab3 1 0x2 L

SEG 8 text.1 8 0x0 0x18 LRX

SYM 9 lab4 8 0xC L

SEG 10 data 10 0x100 0xE LRW

SYM 11 d -1 0xA L

SYM 12 lab5 10 0x102 L

SEG 13 bss 13 0x0 0x4 LRW

#rel.data.1

0x2 A 4

0x3 A 4

0x4 A 4

data.1

19 00 03 03 03

#rel.text.1

text.1

00 0A 00 00 01 00 00 00 30 03 21 40 37 01 33 80

3A 02 19 00 03 06 00 00

#rel.data

0x102 A 4

data

04 01 F7 FF FF FF FE FF FF FF 00 00 00 00

#rel.bss

bss

#end

Test2

Cilj ovog testa je da pokaže rad sa instrukcijama skoka i načinima adresiranja. I u ovom testu možemo videti rad sa podacima gde imamo izraze i negativne vrednosti kod inicijalizacije.

.data.1

.global lab1, lab6

a DEF 4

b DEF a-3

lab1: DD -8

lab2:

c DEF 0x9

DW (c-a)\*2

ORG lab2-lab1 +100

.text.1

PUSH R5

JGZ R1, R7

JZ R2, [R9]

JLEZ R3, [R4+ lab2-lab1+0x5]

JNZ SP, [R2+ 0b010]

lab3: POP R8

.data.2

DD 10 DUP 1234

DB lab3 -11

.text.2

JMP lab1

JMP [R5+0x9]

JMP lab2+6-0b01

.end

Izlaz2:

#TabelaSimbola

SEG 0 UND 0 0x0 0x0 L

SEG 1 data.1 1 0x0 0x6 LRW

SYM 2 lab1 1 0x0 G

SYM 3 lab6 0 0x0 G

SYM 4 a -1 0x4 L

SYM 5 b -1 0x1 L

SYM 6 lab2 1 0x4 L

SYM 7 c -1 0x9 L

SEG 8 text.1 8 0x68 0x20 LRX

SYM 9 lab3 8 0x84 L

SEG 10 data.2 10 0x0 0x29 LRW

SEG 11 text.2 11 0x0 0x18 LRX

#rel.data.1

data.1

F8 FF FF FF 0A 00

#rel.text.1

text.1

20 05 00 00 06 01 38 00 04 42 48 00 09 E3 20 00

09 00 00 00 05 F0 10 00 02 00 00 00 21 08 00 00

#rel.data.2

data.2

D2 04 00 00 D2 04 00 00 D2 04 00 00 D2 04 00 00

D2 04 00 00 D2 04 00 00 D2 04 00 00 D2 04 00 00

D2 04 00 00 D2 04 00 00 79

#rel.text.2

0x4 A 2

0x14 A 1

text.2

02 C0 00 00 00 00 00 00 02 E5 00 00 09 00 00 00

02 C0 00 00 09 00 00 00

#end

Test 3

Cilj ovog testa je rad sa instrukcijama LOAD/STORE, PUSH/POP i razlicitim tipovima operanada koji se mogu naći u instrukcijama LOAD/STORE.

.data.1

a DEF 6

b DEF 0x20

.global lab1,lab6,lab9

lab1:

DW b-a

lab2:

c DEF 0b11

lab3: DB (b-a-24)\*3 DUP lab1 + 3

.text.1

LOADUB R1, R2

LOADSB R0, [R4]

LOADUW R5, #200

lab4: LOADSW R5, [R3 + ( lab2 - lab1 ) \* 2 ]

STOREB R4, 0x500

STOREW R8, lab2-lab1

lab5: STORE R7, $lab2

ADD R3, R4, R5

PUSH R2

MUL R1,R2,R3

DIV R1,R2,R3

MOD R4, R5, R6

lab7: POP R12

.data.2

d DEF 10

DW lab9 ;lab9 je externi simbol

lab8: DD -4, ?, ?

.end

Izlaz3:

#TabelaSimbola

SEG 0 UND 0 0x0 0x0 L

SEG 1 data.1 1 0x0 0x8 LRW

SYM 2 a -1 0x6 L

SYM 3 b -1 0x20 L

SYM 4 lab1 1 0x0 G

SYM 5 lab6 0 0x0 G

SYM 6 lab9 0 0x0 G

SYM 7 lab2 1 0x2 L

SYM 8 c -1 0x3 L

SYM 9 lab3 1 0x2 L

SEG 10 text.1 10 0x0 0x48 LRX

SYM 11 lab4 10 0x10 L

SYM 12 lab5 10 0x28 L

SYM 13 lab7 10 0x44 L

SEG 14 data.2 14 0x0 0xE LRW

SYM 15 d -1 0xA L

SYM 16 lab8 14 0x2 L

#rel.data.1

0x2 A 4

0x3 A 4

0x4 A 4

0x5 A 4

0x6 A 4

0x7 A 4

data.1

1A 00 03 03 03 03 03 03

#rel.text.1

0x2C R 1

text.1

10 01 10 18 10 40 20 38 10 80 28 08 C8 00 00 00

10 E3 28 28 04 00 00 00 11 C0 20 18 00 05 00 00

11 C0 40 08 02 00 00 00 11 F1 38 00 FE FF FF FF

30 03 21 40 20 02 00 00 32 01 10 C0 33 01 10 C0

34 04 29 80 21 0C 00 00

#rel.data.2

0x0 A 6

data.2

00 00 FC FF FF FF 00 00 00 00 00 00 00 00

#end

Test4

Cilj ovog testa je da se vidi rad sa instrukcijama koje u sebi sadrze $, gde se koristi PC relativno adresiranje i relativna relokacija. Ovde se vide slucajevi kada se labela na koju se skace može naci u istoj sekciji i u drugoj sekciji od one u kojoj se nalazi instrukcija u kojoj je korišćena. I kako izgleda relokacija u tim slucajevima i da li uopste postoji. Može se takodje videti i rad sa labelama koje se nalaze u okviru sekcije cija je početna adresa odredjena ORG direktivom.

.text.1

.global lab2,lab9

JMP $lab2

lab1: PUSH R2

lab2: POP R3

INT R2

JGEZ R5, $lab3

ORG 0x70

.text.2

.global lab4

lab3: CALL lab1+7

JZ R6, $lab1

lab4: LOAD R3, $(lab2 + 0b111)

JMP 0x2+ 0b11 - 4+ lab2+lab3-lab1

RET

.data.1

k DEF 2

lab5:

DD lab4 - lab3 + lab2 - 0b01 + k\*2

DB lab4-lab3 DUP 0b011

.data.2

a DEF 1

DB ?, ?, ?, 3\*1-9, (a+2)/3

.end

Izlaz4:

#TabelaSimbola

SEG 0 UND 0 0x0 0x0 L

SEG 1 text.1 1 0x0 0x1C LRX

SYM 2 lab2 1 0xC G

SYM 3 lab9 0 0x0 G

SYM 4 lab1 1 0x8 L

SEG 5 text.2 5 0x70 0x24 LRX

SYM 6 lab4 5 0x80 G

SYM 7 lab3 5 0x70 L

SEG 8 data.1 8 0x0 0x14 LRW

SYM 9 k -1 0x2 L

SYM 10 lab5 8 0x0 L

SEG 11 data.2 11 0x0 0x5 LRW

SYM 12 a -1 0x1 L

#rel.text.1

0x18 R 5

text.1

02 F1 00 00 04 00 00 00 20 02 00 00 21 03 00 00

00 02 00 00 07 E5 88 00 FC FF FF FF

#rel.text.2

0x74 A 1

0x7C R 1

0x84 R 2

text.2

03 C0 00 00 0F 00 00 00 04 E6 88 00 04 00 00 00

10 F1 18 00 03 00 00 00 02 C0 00 00 75 00 00 00

01 00 00 00

#rel.data.1

0x0 A 2

data.1

13 00 00 00 03 03 03 03 03 03 03 03 03 03 03 03

03 03 03 03

#rel.data.2

data.2

00 00 00 FA 01

#end