Опис проблема:

Потребно је креирати двопролазни асемблер који преводи изворни текстуални фајл, у предметни програм. Асемблер треба да одради 2 пролаза кроз задати фајл и да, у случају да нема неких грешака, преведе и генерише излазни фајл који садржи табелу симбола, потребне релокационе табеле и потребне машинске записе за дате инструкције, директиве итд.

Рјешење:

Асемблер је програм који у својој главној функцији прави два пролаза. Током првог пролаза прикупља и генерише записе о свим симболима, било да су глобални или локални. У другом пролазу генерише одговарајуће записе везане за секције, и табеле релокационих записа за сваку секцију која их садржи. За бсс секцију се не генеришу записи. Програм такође генерише посебну табелу која садржи величине секција. Ово првенствено служи да би назначило линкеру колике су секције, и колика је бсс секција која нема никакве записе. Програм исписује поруку о завршетку на главном испису. Ако је све исправно, програм ће генерисати фајл izlaz.txt, у сурпотном ће генерисати фајл log.txt са поруком о грешци. У случају вишеструког покретања програма који има грешку, лог фајл ће садржати и записе о свим претходним грешкама. Асемблер подржава писање израза у директивама.

Упутство за коришћење:

У исти фолдер ставити извршни фајл и фајл за превођење са екстензијом .txt. Програм покренути из терминала. У истом фолдеру ће се генерисати излазни фајл. У конзоли се откуца име текстуалног фајла а затим почетна адреса од које се секције записују редом.

Претпоставке:

Jump инструкција може да користи само PC релативно адресирање и меморијско директно, са тиме да меморијско директно је само у таквој синтакси (нпр. jmpeq a), док се у запису записује код за непосредно адресирање, пошто инструкција скока треба да скочи на ту инструкцију. У зависности од адресирања се преводи са add кодом ако је у питању PC релативно адресирање, а у супротном се преводи као инструкција mov. Свака инструкција садржи услов у имену и није битно да ли се тај услов налази испред или иза самог имена инструкције (нпр. jmpeq a, eqjmp ). Такође једина инструкција која подржава непосредни податак као први операнд јесте push инструкција која га том приликом ставља на стек.

Спроведени су тестови под називима: test1.txt, test2.txt, test3.txt. Њихови изворни кодови су дати у прилогу. Циљ ових тестова јесте да покажу различите излазе за разне инструкције и за све типове адресирања, као и за директиве са подршком за рачунање израза. Тестови су направљени тако да немају грешку, а у конзоли је за адресу куцана адреса 100. У даљем тексту су дати њихови изворни кодови као и излази.

Тест1:

.global a,b,c

.text

mov r3, -20

.skip 4

.align 4

h:

.long 4

a: .long e-h

gtmov r5,&a

mov r5[a],r3

.data

e:.word a - e + g + \_h

mov r5[a],r4

mov r5,20

g: .long z

mov r5,r4[20]

.long 2

f: .char n

.rodata

z: .long 2

.bss

n: .word 1

.end

Излаз:

#Tabela simbola

#ime #sekcija #vr #dostupnost #rb

UND 0 l 0

.text text 100 l 1

.data data 132 l 2

.rodata rodata 155 l 3

.bss bss 159 l 4

a text 20 g 5

b UND -1 g 6

c UND -1 g 7

h text 16 l 8

e data 0 l 9

g data 10 l 10

f data 22 l 11

z rodata 0 l 12

n bss 0 l 13

h UND -1 g 15

#Sekcije i njihove velicine

.text 32

.data 23

.rodata 4

.bss 2

#.text

00 34 ec ff

00 00 00 00

00 00 00 00 00 00 00 00

04 00 00 00

00 00 00 00

a0 b5 fe ff

ab 37 fe ff

.#data

1a 00

ac 37 fe ff

00 34 14 00

00 00 00 00

bc 35 14 00

02 00 00 00

00

.#rodata

02 00 00 00

#.rel.text

20 R\_386\_16 15

20 R\_386\_16 2

26 R\_386\_16 5

30 R\_386\_PC16 5

#.rel.data

0 R\_386\_16 5

0 R\_386\_16 1

4 R\_386\_PC16 5

10 R\_386\_16 3

22 R\_386\_16 4

Тест2:

.global \_start

.data

niz:

.long 1,-2,5-3,-4,5

.text

\_start:

push r0

mov r0, 65

mov r0, 0

mov r1, 5

mul r1, 2

mov r2, 0

loop: cmp r2, r1

moveq r7, &cont

mov r3, r2[niz]

addgt r0, r3

incr: add r2, 2

jmp loop

cont:

pop r0

ret

.rodata

.char -2, 12, 17

.end

Излаз:

#Tabela simbola

#ime #sekcija #vr #dostupnost #rb

UND 0 l 0

.data data 100 l 1

.text text 120 l 2

.rodata rodata 160 l 3

start UND -1 g 4

niz data 0 l 5

\_start text 0 l 6

loop text 22 l 7

incr text 32 l 8

cont text 36 l 9

#Sekcije i njihove velicine

.data 20

.text 40

.rodata 3

#.text

00 25

00 34 41 00

00 34 00 00

00 34 05 00

00 08 02 00

00 34 00 00

49 11

e0 35 22 00

7a 35 fe ff

0b 81

00 00 02 00

00 34 14 00

00 29

e0 28

.#data

01 00 00 00

fe ff ff ff

02 00 00 00

fc ff ff ff

05 00 00 00

.#rodata

fe

0c

11

#.rel.text

26 R\_386\_16 2

30 R\_386\_PC16 1

40 R\_386\_16 2

Тест3:

.global a,b,c

.text

call getchar

a:

call scanf

sub r3,&a

moveq r2, r4[b]

addne r1, r5[32]

push 20

mov r3,\*20

eqmov &a,r5

aladd r3, $c

gtadd r1, z

jmp a

ret

iret

.data

b: .long 2

c: .long 3

.bss

z:

.long 3

.end

Излаз:

#Tabela simbola

#ime #sekcija #vr #dostupnost #rb

UND 0 l 0

.text text 100 l 1

.data data 148 l 2

.bss bss 156 l 3

a text 4 g 4

b data 0 g 5

c data 4 g 6

z bss 0 l 7

getchar UND -1 g 8

scanf UND -1 g 9

#Sekcije i njihove velicine

.text 48

.data 8

.bss 4

#.text

00 ee fe ff

00 ee fe ff

60 05 fe ff

5c 35 fe ff

3d 41 20 00

00 24 14 00

70 35 14 00

0d 34 fe ff

7f c1 fe ff

30 81 fe ff

00 34 fe ff

e0 28

.#data

02 00 00 00

03 00 00 00

#.rel.text

2 R\_386\_16 8

6 R\_386\_16 9

10 R\_386\_16 4

14 R\_386\_PC16 5

30 R\_386\_16 4

34 R\_386\_PC16 6

38 R\_386\_16 3

42 R\_386\_16 4