**Асемблер**

Студент: Мићовић Марко

2019

# Увод

Од прилога, поред овог документа се могу наћи извршни фајл програма, изворни код, тест примери, скрипта за инсталирање одговарајуће верзије g++ преводиоца и makefile ради лакшег превођења изворног кода.

У директоријуму src се налази изворни код решења, у директоријуму test се налазе тест фајлови, у директоријуму bin се налазе извршни фајл, Makefile и скрипта за инсталирање одговарајуће верзије g++ преводиоца.

Програм је написан на језику C++ и за оперативни систем Linux.

Превођење и покретање под овим оперативним системом је описано у даљем тексту.

# Упутство за превођење програма

За превођење програма се користи g++ преводилац, верзија 4.9.

У слуају да немате наведени компајлер на Linux систему под којим радите, могуће је извршити инсталазију помоћу скрипте из прилога.

Превођење изворних фајлова је једноставно и ради се из командне иније.

Потребно је да поставите Makefile и src директоријум у исти директоријум и да се затим лоцирате у њему. Потом, је неопходно укуцати команду make, која ће креирати извршни фајл („make –f Makefile”, или само „make”).

Програм покрећете покретањем новонасталог фајла.

# Упутство за покретање програма

Програм се може покренути само из командне линије.

Ово се постиже на стандардан начин за оперативнеи систем Linux. Ако локација извршног фајла није додата у путању системских програма, неопходно је позиционирати се у директоријум у коме се налази извршна веерзија програма.

Програм се покреће тако што се командној линији зада име програма (./име\_програма, ако није додат у путању системских програма) и имена датотека које садрже код у асемблеру који се треба превести.

Пример покретања: “./аssembler -o output.o input.s”. Ово је једини прихватљиви формат и редослед аргумената.

# Опис решења

Овај пројекат је реализован у виду двопролазног асемблера.

У првом пролазу асемблер пролази кроз уазну датотеку ради уклањања коментара и лабела и убацивања тих лабела у табелу симбола. Сем лабела у табелу симбола се налазе и симболи уведени помоћу equ и extern директива.

За један симбол се у табели симбола памти име симбола, секција у којој се налази, померај од почетка текуће секције досег сваког симбола. Досег је глобалан, или локалан за дати модул. У првом пролазу се досег свих новодефинисаних симбола поставља на локални.

Осим табеле симбола саставља се и табела секција. За једну секцију се памти њено име, редни број и права приступа.

У другом пролазу асемблер врши превођење асемблерског кода на машински јеѕик дат по спецификацији процесора за који је асемблер писан.

У овом пролазу се врши обрађињање global директива ради постављања досега локално дефинисаних симбола.

Све инструкције се декодирају у машинске речи дужине 1 до 7 бајтова.

За све симболе коришћене у инструкцијама ће се извршити релокација. За сваку секцију се води посебна табела релокација.

Постоје и директиве byte, word, skip, align

Byte и word могу да примају један или више аргумената одвојених зарезом. Skip и align примају тачно један аргумент.

* **Byte** умеће 1 бајт података по аргументу на место на ком је дефинисана. Вредност уметнутог бајта је вредност аргумента.
* **Word**  умеће 2 бајта података по аргументу на место на ком је дефинисана. Вредност уметнутих бајтова бајта је вредност аргумента, по litle-endian принципу.
* **Skip**  прескаче онолико бајтова колико се тражи у аргументу и попуњава их нулама.
* **Align** додаје нулте бајтове све док location counter те секција није дељив са бројем и аргумента.

## Опис релокација

У систему постоје две врсте релокација.

Релативне релокације се извршавају ако за инструкције које захтевају pc релативно адресирање помоћу операнда $.

Апсолутне релокације се извршавају у преосталим случајевима.

# Тестови

## Тест 1:

.text

.global find\_x

.extern b

find\_x:

.word 0x3

push K

.align 4

jmp b

mov r3, r2[12] ; r3 = n

sub r3, 1 ; r3--

mov r1, r2[16] ; r1 = x

mov r0, r2[8] ; r0 = &array

cmp r3, 0 ; if n <= 0 goto exit

jgt b

a: mov r2, r0[4\*2]

cmp r3, r2

jeq l2

jmp b

l2: mov r0, r1

pop r2

ret

.section newsection, rx

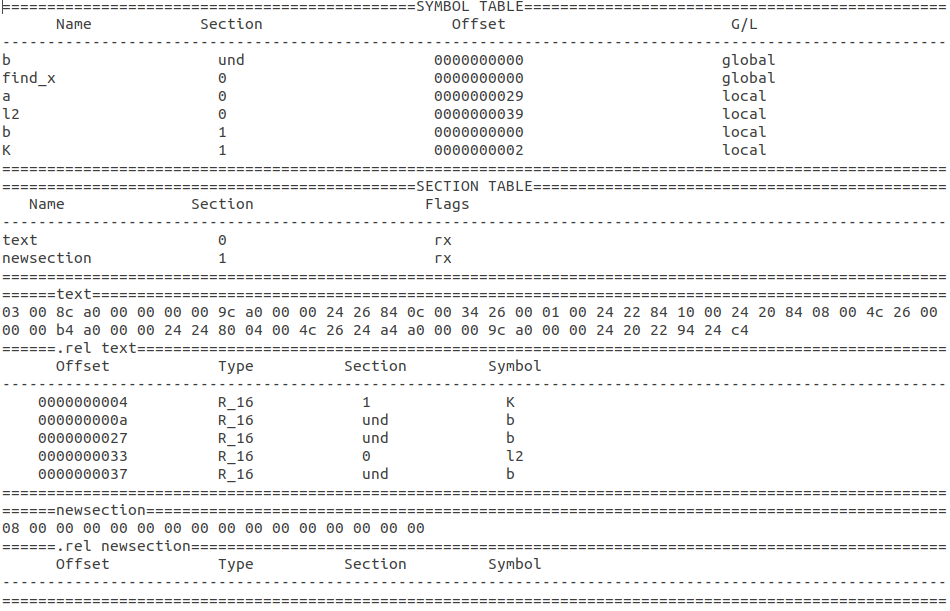
b: .word 8

K: .align 0x10 ,0xff , 0x10

.align 2,

.end

## Izlaz:



## Тест 2:

.data ; Pocetak data sekcije.

ulazni\_format: ; String za formatiranje ulaza scanf funkcije

.byte 0x25, 105, 045, 0x69, 0 ; "%i%i\0"

izlazni\_format: ; String za formatiranje izlaza printf funkcije

.byte 0x25, 0x69, 0x20, 0x2B ; "%i +" +

.byte 0x20, 0x25, 0x69, 0x20 ; " %i " +

.byte 0x3d, 0x20, 0x25, 0x69 ; "= %i" +

.byte 0x5C, 0x6e, 0x0 ; "\n\0" = "%i + %i = %i\n"

;============================================================================================

.text ; pocetak sekcije sa izvrsnim kodom

.extern write

.extern write

.extern scanf

.extern printf

.extern exit

.global start

start:

call main

push 0

call exit ; ekvivalentno: exit(0);

saberi: ; Primjer funkcije koja koristi

; stdcall konvenciju pozivanja.

push r5 ; bp

mov r5, sp

mov r0, r1[4]

add r0, r1[6] ; Rezultat ostaje u r0.

pop r5

pop r4

pop r4 ; Pri izlasku, u slucaju fiksnog broja parametara,

ret ; funkcija je duzna da oslobodi prostor koji su

; argumenti zauzimali na steku.

main:

push r5 ; bp

mov r5, sp

mov r0, r1

sub r0, 2 ; Adresa prve pomocne promjenljive (ebp-4)

push r0 ; Treci argument za scanf poziv.

sub r0, 0x2 ; Adresa druge pomocne promjenljive (ebp-8)

push r0 ; Drugi argument za scanf poziv

push $ulazni\_format ; Prvi argument za scanf poziv

call scanf

add r6, 6 ; Oslobadjanje prostora koji su zauzimali argumenti.

push r5[-4] ; Dohvatanje prvog ucitanog broja - 2. arg. za "saberi".

push r5[-2] ; Dodavanje drugog ucitanog broja - 1. arg. za "saberi".

call saberi ; Rezultat vraca u registru eax.

push r0 ; 4. argument - izracunata suma

push r5[-2] ; 3. argument - drugi ucitani broj

push r5[-4] ; 2. argument - prvi ucitani broj

push $izlazni\_format ; 1. argument - adresa stringa za formatiranje

call printf

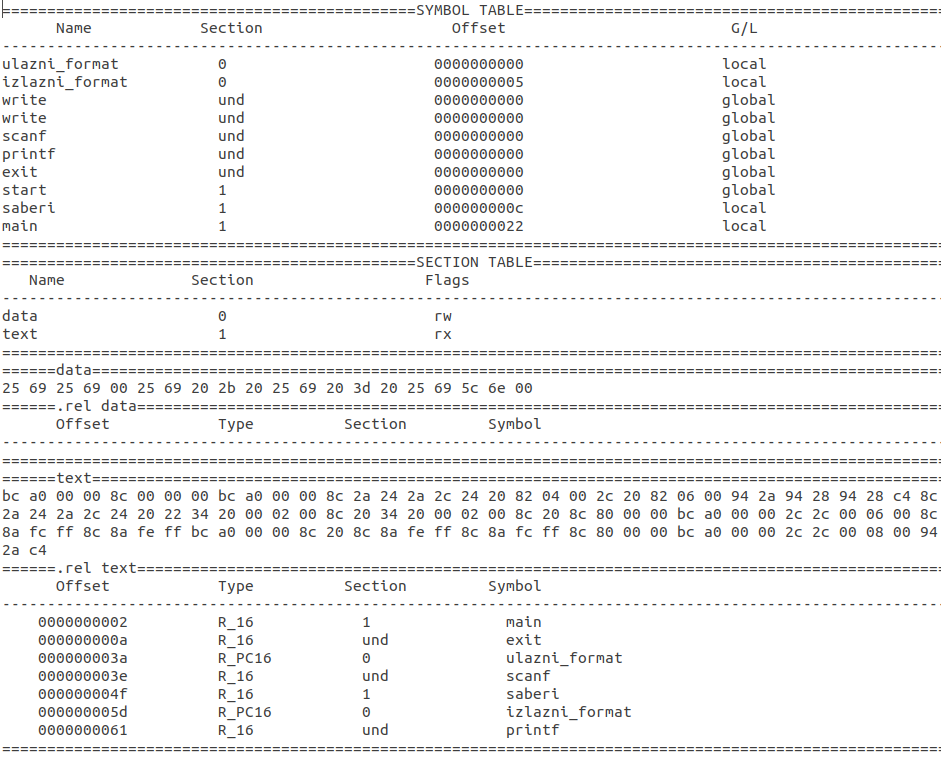
add r6, 8

pop r5

ret

.end

## Излаз:



## Тест 3:

; main.s - Bubblesort algorithm demonstration.

.data

.global X

.equ X, 40

array:

.word 10, 42, 8, 34, 22 , 2, 1, 87, 99, 53, 67, 10, 69

.word 22, 80, 11, 75, 25, 0, 12

size: .word X

.text

.extern print\_hex

.extern println

.global START

START:

push &array

push size

call bubblesort

add r6, 4

mov r1, 0

loop: push r1[array]

call print\_hex

add r6, 2

call println

add r1, 2

cmp size, r1

jgt loop

exit: halt

bubblesort: ; assumes size is greater than 0

push r1

push r2

push r3

push r4

mov r3, r6[10]

add r3, r6[12]

; sub r3, 2

next\_iter: cmp r3, r6[10]

jeq $return

mov r1, r6[10]

mov r2, r1

inner\_iter: add r2, 2

cmp r2, r3

jeq $inner\_iter\_end

mov r4, r2[0]

cmp r4, r1[0]

jgt $skip\_swap

push r1

push r2

call $swap

add r6, 4

skip\_swap: mov r1, r2

jmp $inner\_iter

inner\_iter\_end: sub r3, 2

jmp $next\_iter

return:

pop r4

pop r3

pop r2

pop r1

ret

swap:

push r1

push r2

push r3

mov r1, r6[8]

mov r2, r6[10]

mov r0, r1[0]

mov r3, r2[0]

mov r1[0], r3

mov r2[0], r0

pop r3

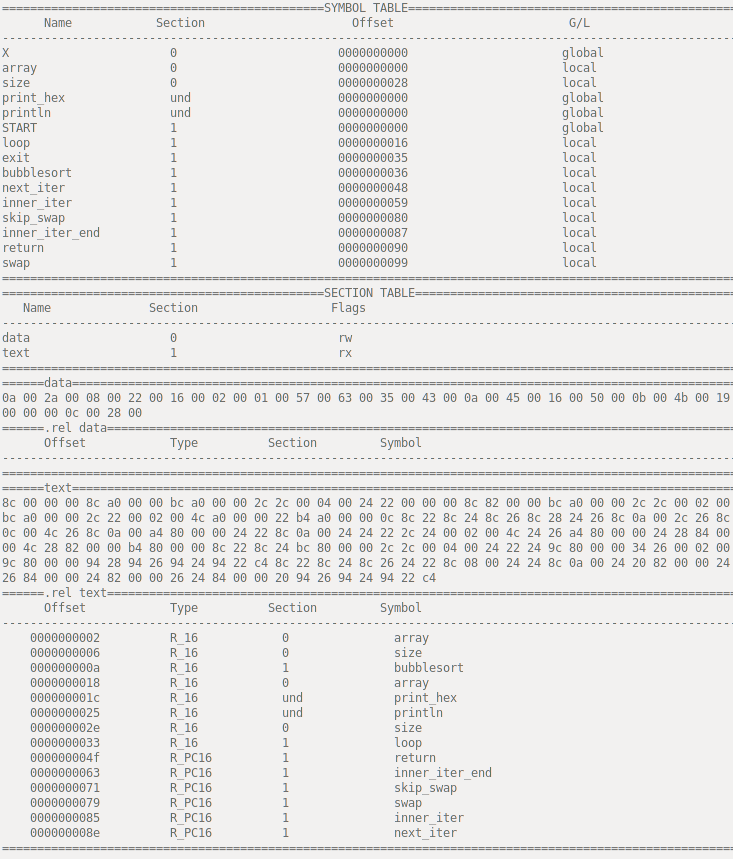
pop r2

pop r1

ret

.end

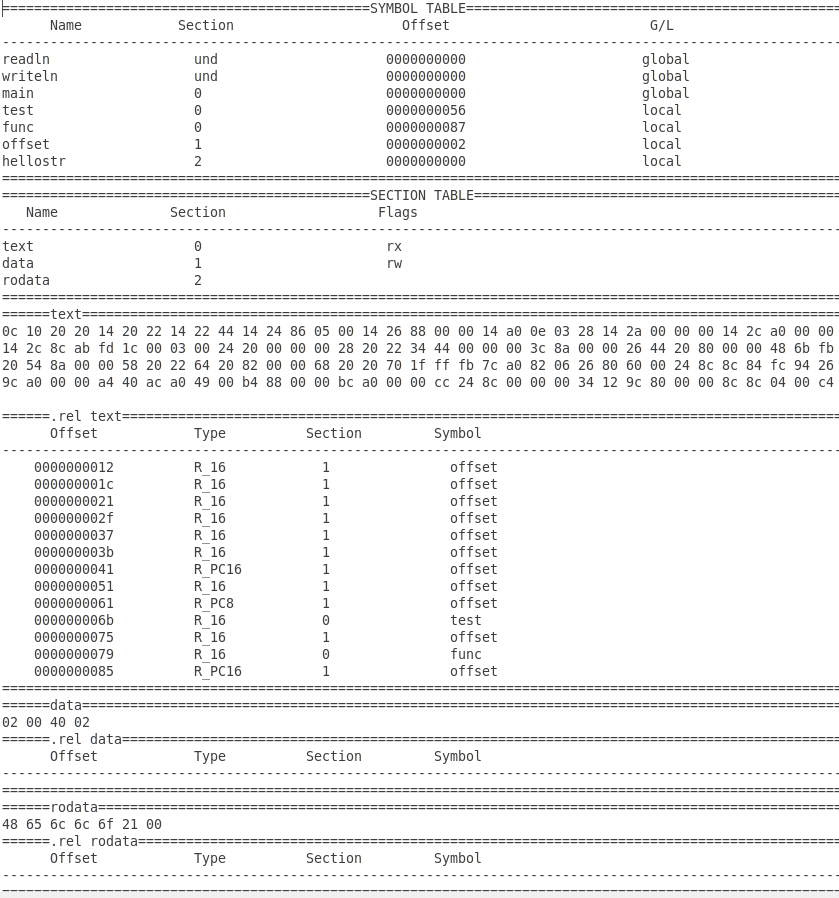
## Излаз:



## Тест 4:

.text  
  
.global main  
.extern readln  
.extern writeln  
  
main:  
 halt  
 xchgb r0h, r0l  
 xchg r0, r1  
 xchgw r1, r2[]  
 xchgw r2, r3[5]  
 xchg r3, r4[offset]  
 xchgw \*782, r4  
 xchg r5, &offset  
 xchg sp, offset  
 xchg sp, r6[-597]  
 int 3  
 mov r0, &offset  
 addb r0h, r1l  
 sub r2[], &offset  
 mul r5[offset], r3  
 div r0, $offset  
 cmpb sp[-5], r0h  
 not r5[0]  
 andb r0h, r1l  
 orw r0, r1[offset]  
 xorb r0h, r0h  
test:  
 testb r0l, -5  
 shl \*0x682, r3  
 shrb $offset, r2h  
 push sp[-892]  
 pop r3  
 jmp test  
 jeq r0[]  
 jne \*73  
 jgt r4[offset]  
 call func  
 iret  
 movw sp[0], 0x1234  
 jmp $offset  
func:  
 push sp[4]  
 ret  
  
.data  
.word 2  
offset: .word 576  
  
.section rodata   
  
hellostr: .byte 0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x21, 0x0  
  
.end

## Излаз:



## Тест 5:

.data  
test: .skip 100, 0xff  
.align 8, 03, 2  
test2: .skip 0x43, 057  
.skip 0b10110110  
.byte 0b00100010, 074, -135, 0xf2  
.word 0b0101010101010101, 0356, 6421, 0x3f2a  
 .word 56, 031 , 65536 , 0x3f2a  
  
.equ num, 45  
  
.text  
.align 2, 057, 1   
  
.global MaIn  
.extern printf, test, t2est  
  
 MaIn:   
test3: mov r0, &num add r5, r2[0] test r2[0x5], r0 and r4[536], r0 subb r2l, r3h xchgw r0, sp halt .section data2, rw  
 n: .word 0x195f  
 TESTMatch: .word 0x195f   
 ;.L0: #  
 \_testLab\_el12.3test:   
 023test:   
 t$test: test\_:   
  
.section rodata , a  
.section testsection2 , rwx   
  
.end  
  
junkdata: .word 0x54

## Излаз:

