**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА» (СПбГУТ)**

КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Дисциплина: «Машинно-зависимые языки программированияе»

Лабораторная работа №3.

**Тема: «Ввод-вывод на ассемблере»**

Выполнил:

Студент группы ИКПИ-21

Козлова А. И.

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Принял:

Старший преподаватель кафедры ПИиВТ

Анохин Ю. В.

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_

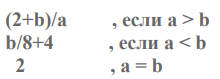
**А. *Постановка задачи***

Реализовать целочисленные вычисления, сделанные в лабораторной работе №1 или №2(по указанию преподавателя), полностью используя язык Ассемблера для организации корректного ввода-вывода информации:

• исходные данные должны вводиться с проверкой правильности вводимых символов;

• входные данные и результат должны быть проверены на область допустимых значений;

• при наличии ошибки должно быть выдано соответствующее сообщение.



**Б. *Разработка алгоритма***

С помощью глобальных переменных в C и external-объявленных переменных будет осуществляться обмен данными между модулями на Assembler и C. С помощью операций mov, div, sub, add, модуль на Assembler будет высчитывать необходимые значения.

**В. *Описание программы***

Варианты программы для разных типов данных. Программа принимает на вход числа – a, b, затем выдаёт результат, высчитанный на С и на Assembler

**Д. Контрольный расчёт**

a = 5, b = 6

Результат = 6/8 + 4 = 4;

a = 6, b = 5

Результат = (2 + 5)/6 = 1;

a = 4, b = 4

Результат = 2;

**Д. *Результаты работы программы***

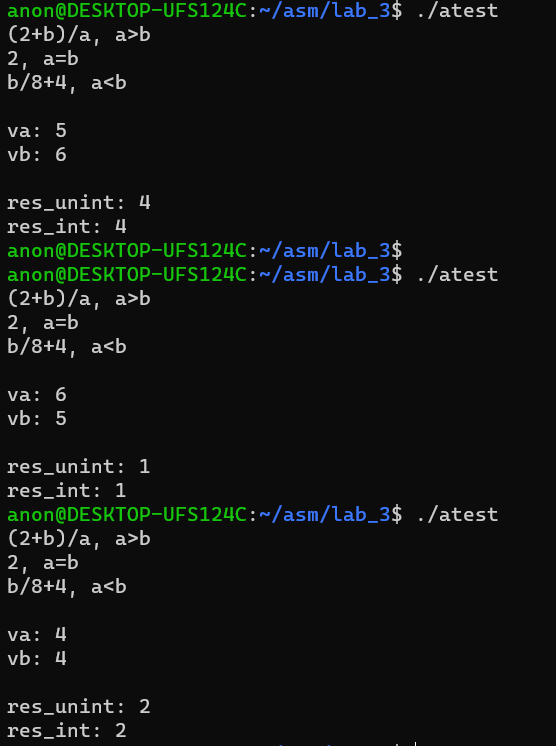


рис 1. Результаты работы программы

**E. Код программы.**

**Asm**

%macro print 1

mov rsi, %1

call \_print

%endmacro

%macro isDigit 1

mov rsi, %1

call is\_digit

test rsi, rsi

%endmacro

%macro stoi 1

mov rsi, %1

call str\_to\_int

%endmacro

%macro checkForInt 1

mov rsi, %1

call check\_for\_int

test rsi, rsi

%endmacro

%macro itos 2

movsx rax, %1

mov rsi, %2

call int\_to\_str

%endmacro

section .data

input\_a db 'va: ', 0

input\_b db 'vb: ', 0

err\_d db 'Division by zero!', 10, 0

endl db 10, 0

tz\_1 db '(2+b)/a, a>b', 10, 0

tz\_2 db '2, a=b', 10, 0

tz\_3 db 'b/8+4, a<b', 10, 0

outp\_res\_unint db 'res\_unint: ', 0

outp\_res\_int db 'res\_int: ', 0

eia\_int db 'Invalid int!', 10, 0

eia\_str db 'Invalid string!', 10, 0

err\_num db 'Invalid number!', 10, 0

sign db 0

section .bss

res\_int resb 4

res\_unint resb 4

va resb 4

vb resb 4

buf resb 16

section .text

;(b+a)/(a-1) a>b

;23 a=b

;b\*b/8 a<b

;(2+b)/a , если a > b

;b/8+4 , если a < b

;2 , a = b

global \_start

\_start:

mov rbp, rsp; for correct debugging

print tz\_1

print tz\_2

print tz\_3

print endl

print input\_a

call input

isDigit buf

jz inv\_arg\_int

stoi buf

checkForInt rdx

jz inv\_num

mov [va], rdx

print input\_b

call input

isDigit buf

jz inv\_arg\_int

stoi buf

checkForInt rdx

jz inv\_num

mov [vb], rdx

print endl

call unint\_func

unint\_func:

xor eax, eax

xor ebx, ebx

xor ecx, ecx

xor edx, edx

mov eax, [va]

mov ebx, [vb]

cmp eax, 65535

jg unint\_inv

cmp ebx, 65535

jg unint\_inv

cmp eax, 0

jl unint\_inv

cmp ebx, 0

jl unint\_inv

unint:

cmp eax, ebx

je equal

ja umore

jb uless

ret

intt\_func:

xor eax, eax

xor ebx, ebx

xor ecx, ecx

xor edx, edx

mov eax, [va]

mov ebx, [vb]

cmp eax, 32767

jg int\_inv

cmp ebx, 32767

jg int\_inv

cmp eax, -32768

jl int\_inv

cmp ebx, -32768

jl int\_inv

intt:

cmp eax, ebx

je equal

jg more

jl less

ret

equal:

mov dword [res\_unint], 2

mov dword [res\_int], 2

itos dword [res\_unint], buf

print outp\_res\_unint

print buf

print endl

itos dword [res\_int], buf

print outp\_res\_int

print buf

print endl

call \_exit

;a>b

umore:

xor eax, eax

xor ebx, ebx

xor ecx, ecx

xor edx, edx

mov ax, [vb]

mov bx, [va]

add ax, 2

;cdqe

test ebx, ebx

jz divz

div ebx

;add eax, ebx

; dec ebx

; test ebx, ebx

; jz divz

; div ebx

mov [res\_unint], eax

itos dword [res\_unint], buf

print outp\_res\_unint

print buf

print endl

call intt\_func

more:

xor eax, eax

xor ebx, ebx

xor ecx, ecx

xor edx, edx

mov ax, [vb]

add ax, 2

;add ax, [va]

cwde

;mov ecx, eax

mov bx, [va]

cwd

test bx, bx

jz divz

idiv bx

cwd

mov [res\_int+2], dx

mov [res\_int], ax

;dec eax

; jz divz

; mov ebx, eax

; mov eax, ecx

; cdq

; idiv ebx

;mov [res\_int], eax

itos dword [res\_int], buf

print outp\_res\_int

print buf

print endl

call \_exit

uless:

xor eax, eax

xor ebx, ebx

xor ecx, ecx

xor edx, edx

mov ax, [vb]

mov ecx, 8

div ecx

;cwde

add eax, 4

;mul eax

; mov bx, 8

; div ebx

mov [res\_unint], eax

itos dword [res\_unint], buf

print outp\_res\_unint

print buf

print endl

call intt\_func

less:

xor eax, eax

xor ebx, ebx

xor ecx, ecx

xor edx, edx

mov ax, [vb]

cwde

mov cx, 8

cwd

idiv cx

cwd

add ax, 4

mov [res\_int+2], dx

mov [res\_int], ax

;imul eax

; mov bx, 8

; idiv ebx

; mov [res\_int], eax

itos dword [res\_int], buf

print outp\_res\_int

print buf

print endl

call \_exit

unint\_inv:

print outp\_res\_unint

mov rsi, err\_num

call \_print

call intt\_func

int\_inv:

print outp\_res\_int

mov rsi, err\_num

call \_print

call \_exit

\_print:

push rsi

xor rdx, rdx

print\_cikl:

inc rdx

inc rsi

mov cl, [rsi]

cmp cl, 0

jne print\_cikl

mov rdi, 1

mov rax, 1

pop rsi

syscall

ret

input:

mov rsi, buf

mov rdx, 1

mov rdi, 0

input\_cikl:

xor rax, rax

syscall

mov cl, [rsi]

inc rsi

cmp cl, 10

jne input\_cikl

dec rsi

mov [rsi], byte 0

ret

is\_digit:

mov cl, [rsi]

cmp cl, '-'

je is\_digit\_cikl

sub cl, '0'

cmp cl, 9

jbe is\_digit\_cikl

ja digit\_false

is\_digit\_cikl:

inc rsi

mov cl, [rsi]

cmp cl, 0

je digit\_true

sub cl, '0'

cmp cl, 9

jbe is\_digit\_cikl

ja digit\_false

digit\_false:

xor rsi, rsi

ret

digit\_true:

mov rsi, 1

ret

str\_to\_int:

xor rdx, rdx

xor rcx, rcx

mov [sign], byte 0

mov cl, [rsi]

cmp cl, '-'

je set\_sign

jne str\_to\_int\_cikl

str\_to\_int\_cikl:

mov cl, [rsi]

cmp cl, 0

je str\_to\_int\_end

sub cl, '0'

imul rdx, 10

add rdx, rcx

inc rsi

jmp str\_to\_int\_cikl

set\_sign:

mov [sign], byte 1

inc rsi

jmp str\_to\_int\_cikl

str\_to\_int\_end:

cmp byte [sign], 1

je negative

ret

negative:

neg rdx

ret

check\_for\_int:

mov rax, 65535

cmp rsi, rax

jg int\_false

cmp rsi, -32768

jl int\_false

mov rsi, 1

ret

int\_false:

xor rsi, rsi

ret

int\_to\_str:

xor rbx, rbx

mov rcx, 10

cmp rax, 0

je res\_nul

jl res\_neg

push rsi

add rsi, 14

jmp int\_to\_str\_cikl

res\_nul:

mov [buf], rax

mov [rsi], byte '0'

ret

res\_neg:

mov [rsi], byte '-'

inc rsi

push rsi

add rsi, 13

neg rax

jmp int\_to\_str\_cikl

int\_to\_str\_cikl:

xor rdx, rdx

test rax, rax

jz int\_to\_str\_end

div rcx

add dl, '0'

mov [rsi], dl

dec rsi

inc rbx

jmp int\_to\_str\_cikl

int\_to\_str\_end:

add dl, '0'

mov [rsi], dl

pop rax

inc rsi

inc rbx

shift:

test rbx, rbx

jz shift\_end

mov dl, [rsi]

mov [rsi], byte 0

mov [rax], dl

dec rbx

inc rax

inc rsi

jmp shift

shift\_end:

ret

inv\_arg\_int:

mov rsi, eia\_int

call \_print

jmp \_exit

inv\_num:

mov rsi, err\_num

call \_print

jmp \_exit

divz:

mov rsi, err\_d

call \_print

jmp \_exit

\_exit:

mov rax, 60

mov rdi, 0

syscall