ШтФедеральное государственное бюджетное образовательное учреждение высшего образования «Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича»

Кафедра «Программная инженерия и вычислительная техника»

«Машино-зависимые языки программирования»

Отчет

по лабораторной работе №5

«Сопроцессор»

|  |  |
| --- | --- |
|  | Выполнил |
|  | студент группы ИКПИ-33 |
|  | А. И. Коньков |
|  |  |
|  | Проверил |
|  | Ст. преподаватель |
|  | И.Л. Савельев |
|  |  |

2024 г.

**1. Задание**

1) Вычислить заданное вещественное выражение для исходных данных в форматах Float(SINGLE для переменных a и b) и Int (INTEGER - все остальные переменные), используя арифметические операции сопроцессора

2) Научиться использовать подпрограммы

Программа должна быть написана целиком на языке Ассемблера.

Исходные значения переменных вводятся пользователем с клавиатуры. Они должны быть максимально приближены к максимально-возможным для тех типов данных, с которыми решается задача. При вводе данных рекомендуется вывести диапазон возможных значений. Размер и тип числителя, знаменателя и результата зависит от заданного выражения.

*Вариант №11*

**(2\*d-96/a)/(34/b-a+1)**

**2. Листинг программы**

**2.1. Модуль program.asm**

section .data

; Сообщения для ввода переменных с указанием диапазона

a\_msg db 'Enter a (float, range: 1.175494351e-38 to 3.402823466e+38): ', 0

b\_msg db 'Enter b (float, range: 1.175494351e-38 to 3.402823466e+38): ', 0

c\_msg db 'Enter d (int, range: -2147483648 to 2147483647): ', 0

; Сообщения для вывода результатов

num\_msg db 'Numerator: ', 0

denom\_msg db 'Denominator: ', 0

res\_msg db 'Result: ', 0

newline db 0Ah, 0 ; Символ новой строки

; Сообщения об ошибках

invalid\_input\_msg db 'Invalid floating-point input.', 0Ah, 0

div\_by\_zero\_msg db 'Division by zero error.', 0Ah, 0

dot db '.', 0 ; Десятичная точка

; Переменные для вычислений

a dq 0.0

b dq 0.0

c dq 0.0

numerator dq 0.0

denominator dq 0.0

result dq 0.0

; Константы

two dq 2.0

ninety\_six dq 96.0

thirty\_four dq 34.0

one dq 1.0

ten dq 10.0

section .bss

input resb 32 ; Буфер для ввода строки (до 31 символа + 0)

integer\_buffer resb 21 ; Буфер для преобразования целого в строку (max 20 digits + null)

section .text

global \_start

\_start:

; Ввод переменной a

mov rdi, a\_msg

call print\_string

call read\_float

fstp qword [a]

; Ввод переменной b

mov rdi, b\_msg

call print\_string

call read\_float

fstp qword [b]

; Ввод переменной c

mov rdi, c\_msg

call print\_string

call read\_int

fstp qword [c]

; Вычисление числителя (2 \* d - 96 / a)

fld qword [c] ; ST(0) = d

fmul qword [two] ; ST(0) = 2 \* d

fld qword [ninety\_six] ; ST(0) = 96, ST(1) = 2 \* d

fld qword [a] ; ST(0) = a, ST(1) = 96, ST(2) = 2 \* d

fdiv ; ST(0) = 96 / a, ST(1) = 2 \* d

fsub ; ST(0) = 2 \* d - (96 / a)

fstp qword [numerator] ; Сохранение результата в переменной numerator

; Вычисление знаменателя (34 / b - a + 1)

fld qword [thirty\_four] ; Загружаем 34

fld qword [b] ; Загружаем b

fdiv ; Делим 34 на b

fsub qword [a] ; Вычитаем a

fadd qword [one] ; Прибавляем 1

fstp qword [denominator]

; Проверка деления на ноль

fldz

fcomp qword [denominator]

fstsw ax

sahf

je division\_by\_zero ; Переход к обработчику деления на ноль

; Вычисление результата (numerator / denominator)

fld qword [numerator]

fld qword [denominator]

fdiv

fstp qword [result]

; Вывод числителя

mov rdi, num\_msg

call print\_string

fld qword [numerator]

call print\_float

; Вывод знаменателя

mov rdi, denom\_msg

call print\_string

fld qword [denominator]

call print\_float

; Вывод результата

mov rdi, res\_msg

call print\_string

fld qword [result]

call print\_float

jmp exit

division\_by\_zero:

mov rdi, div\_by\_zero\_msg

call print\_string

jmp exit

exit:

mov rax, 60

xor rdi, rdi ;

syscall

; Вспомогательные функции

print\_string:

; Выводит строку на консоль

; Вход: rdi указывает на строку с завершающим нулем

push rdi

push rax

push rdx

push rcx

call string\_length ; Получение длины строки

mov rdx, rax ; rdx <- длина строки для write

mov rax, 1 ; write

mov rsi, rdi ; буфер

mov rdi, 1 ; stdout

syscall

pop rcx

pop rdx

pop rax

pop rdi

ret

string\_length:

; Вычисляет длину строки

; Вход: rdi указывает на строку с завершающим нулем

; Выход: rax содержит длину строки

xor rax, rax

.loop: ; Цикл подсчета символов

cmp byte [rdi+rax], 0 ; Проверка на конец строки

je .end ; Переход к концу, если символ равен 0

inc rax ; Увеличиваем счетчик

jmp .loop ; Переходим к следующему символу

.end: ; Конец функции string\_length

ret

read\_float:

; Читает число с плавающей точкой из stdin

mov rax, 0 ; sys\_read

mov rdi, 0 ; stdin

mov rsi, input ; Буфер для ввода

mov rdx, 32 ; Размер буфера

syscall

; Преобразование строки в число с плавающей точкой

push qword 0 ; Заполнитель для значения float

; Проверка на ошибки ввода

lea rdi, [input]

call parse\_float

pop rax ; Удаление заполнителя

ret

parse\_float:

; Преобразует строку в число с плавающей точкой

; Вход: rdi - указатель на строку. Результат помещается на вершину стека FPU.

push rbp

mov rbp, rsp

sub rsp, 16 ; Место для локальных переменных

finit ; Инициализировать FPU

fldz ; ST(0) = 0.0 (инициализация результата)

xor rax, rax ; rax - целая часть

xor rcx, rcx ; rcx - флаг отрицательного числа

xor rdx, rdx ; rdx - флаг дробной части

xor r13, r13 ; r13 - счетчик

xor r14, r14 ; r14 - костыль

.check\_sign: ; Проверка знака числа

movzx r8, byte [rdi]

cmp r8b, '-' ; Проверка на минус

jne .parse\_integer ; Если не минус, переходим к разбору целой части

inc rcx ; Устанавливаем флаг отрицательного числа

inc rdi ; Пропускаем знак минус

.parse\_integer: ; Разбор целой части числа

movzx r8, byte [rdi]

cmp r8b, 0 ; Конец строки?

je .finalize\_float ; Переход к завершению, если конец строки

cmp r8b, 10 ; '\n'?

je .finalize\_float ; Переход к завершению, если символ новой строки

cmp r8b, '.' ; Десятичная точка?

je .parse\_fraction ; Переход к разбору дробной части

sub r8b, '0' ; Преобразование символа в цифру

cmp r8b, 0 ; Проверка на корректность цифры

jl .invalid\_input

cmp r8b, 9

jg .invalid\_input

inc r13 ; Увеличиваем счетчик

.cycle\_int: ; Формирование целой части

cmp r13,r14

je .end\_int

imul rax, 10 ; Умножаем целую часть на 10

inc r14

jmp .cycle\_int

.end\_int:

add rax, r8 ; Добавляем текущую цифру

inc rdi ; Переходим к следующему символу

jmp .parse\_integer ; Переходим к следующей цифре целой части

.parse\_fraction: ; Разбор дробной части числа

inc rdi

fldz ; ST(0) = 0.0 (дробная часть)

fld qword [ten] ; ST(0) = 10.0 (делитель)

xor r13, r13 ; r13 - счетчик

.parse\_fraction\_loop: ; Цикл разбора дробной части

movzx r8, byte [rdi]

cmp r8b, 0 ; Конец строки?

je .finalize\_float ; Переход к завершению, если конец строки

cmp r8b, 10 ; '\n'?

je .finalize\_float ; Переход к завершению, если символ новой строки

cmp r8b, '0' ; Проверка на корректность цифры

jl .invalid\_input

cmp r8b, '9'

jg .invalid\_input

sub r8b, '0' ; Преобразование символа в цифру

mov [rbp-8], r8 ; Сохраняем цифру

fild qword [rbp-8] ; Загружаем цифру в FPU

fdiv st0, st1 ; Делим на делитель

fstp qword [rbp-16] ; Сохраняем результат деления

fstp st0 ; Удаляем делитель из стека

fld qword [rbp-16] ; Загружаем результат деления обратно

fadd st0, st1 ; Добавляем к дробной части

fstp st1 ; Сохраняем новую дробную часть

fld qword [ten]

inc r13

xor r14, r14 ; r14 - костыль

.cycle\_fraction:

cmp r13, r14

je .end\_fraction

fmul qword [ten] ; Умножаем 10

inc r14

jmp .cycle\_fraction

.end\_fraction:

inc rdi

jmp .parse\_fraction\_loop ; Переходим к следующей цифре

.finalize\_float:

fstp st0 ; очищаем стек FPU

mov [rbp-8], rax ; сохраняем rax во временную переменную

fild qword [rbp-8] ; загружаем целую часть в стек FPU

fadd st0, st1 ; добавляем дробную часть к целой

cmp rcx, 0 ; проверяем на отрицательность

je .end\_parse

fchs

.end\_parse:

add rsp, 16 ; Восстанавливаем стек

pop rbp

ret

.invalid\_input: ; Обработка некорректного ввода

mov rdi, invalid\_input\_msg

call print\_string

mov rax, 60

mov rdi, 1

syscall

read\_int:

; Читает целое число из stdin

mov rax, 0 ; sys\_read

mov rdi, 0 ; stdin

mov rsi, input ; буфер для ввода

mov rdx, 32 ; размер буфера

syscall

finit ; инициализация FPU

fldz ; ST(0) = 0.0

; Конвертация строки в целое

xor rax, rax ; результат

xor rbx, rbx ; знак (1 для '+', -1 для '-')

mov rbx, 1

lea rsi, [input] ; указатель на буфер ввода

.parse\_loop: ; Цикл разбора числа

movzx rcx, byte [rsi]

cmp rcx, 0 ; конец строки?

je .end\_parse ; переходим к концу, если конец строки

cmp rcx, 10 ; '\n'?

je .end\_parse ; переходим к концу, если символ новой строки

cmp rcx, '-' ; проверка на знак '-'

jne .digit ; если не '-', переходим к обработке цифры

mov rbx, -1 ; устанавливаем знак '-1'

inc rsi ; пропускаем символ '-'

jmp .parse\_loop ; переходим к следующей итерации цикла

.digit: ; Обработка цифры

cmp rcx, '0' ; проверка на корректность ввода

jl .end\_parse ; если меньше '0', завершаем разбор

cmp rcx, '9' ; проверка на корректность ввода

jg .end\_parse ; если больше '9', завершаем разбор

sub rcx, '0' ; преобразуем символ в число

imul rax, 10 ; умножаем текущий результат на 10

add rax, rcx ; добавляем текущую цифру к результату

inc rsi ; переходим к следующему символу

jmp .parse\_loop ; переходим к следующей итерации цикла

.end\_parse: ; Конец разбора

imul rax, rbx ; применяем знак

sub rsp, 8 ; выделяем место на стеке

mov [rsp-8], rax ; сохраняем результат на стеке

fild qword [rsp-8] ; загружаем результат в FPU

add rsp, 8 ; очищаем стек

ret

; Функция вывода числа с плавающей точкой

print\_float:

push rbp

mov rbp, rsp

; Очистка буфера integer\_buffer

lea rdi, [integer\_buffer]

mov rcx, 21

xor rax, rax

.clear\_loop: ; Цикл очистки буфера

mov byte [rdi], al

inc rdi

loop .clear\_loop

; Сохранение числа из FPU в память и преобразование в строку

sub rsp, 32 ; выделение места на стеке для числа

fstp qword [rsp] ; сохраняем число из FPU в память

lea rdi, [rsp] ; адрес числа

lea rsi, [integer\_buffer] ; адрес буфера для строки

call double\_to\_string ; преобразование числа в строку

; Вывод строки на консоль

mov rax, 1 ; sys\_write

mov rdi, 1 ; stdout

lea rsi, [integer\_buffer] ; буфер с результатом

mov rdx, 20 ; максимальная длина вывода

syscall

mov rdi, newline ; вывод новой строки

call print\_string

finit ; очищаем стек FPU

add rsp, 32 ; очистка стека

pop rbp

ret

; Преобразует число с плавающей точкой в строку

; Вход: rdi - адрес числа (qword), rsi - адрес буфера

double\_to\_string:

push rbp

mov rbp, rsp

sub rsp, 32 ; временные переменные

xor r10, r10 ; r10 = 0 (флаг для знака)

fld qword [rdi] ; загружаем число

fst qword [rbp-16] ; сохраняем копию

; Проверка знака

mov rax, [rbp-16]

test rax, rax

jns .positive ; если положительное, пропускаем

mov r10, 1 ; устанавливаем флаг отрицательного числа

fchs ; меняем знак

.positive: ; Обработка положительного числа

; Преобразование целой части

fld st0 ; дублируем число

fisttp qword [rbp-8] ; сохраняем целую часть с усечением

mov rsi, [rbp-8]

lea rdi, [integer\_buffer]

call int\_to\_string ; преобразуем в строку

lea rsi, [integer\_buffer]

call string\_copy ; копируем в буфер

; Добавление точки

mov al, '.'

stosb

push rdi ; сохраняем rdi

; Преобразование дробной части

fld qword [rbp-16] ; загружаем положительное число

fisttp qword [rbp-32] ; целая часть с усечением

fild qword [rbp-32] ; загружаем целую часть в FPU

cmp r10, 1 ; Проверка знака

jne .subtract\_fraction ; Если положительное, вычитаем

jmp .add\_fraction ; Если отрицательное, складываем

.subtract\_fraction: ; Вычитание для положительных чисел

fsubrp st1, st0 ; st0 = дробная часть

jmp .continue\_fraction ; Переход к общей части

.add\_fraction: ; Сложение для отрицательных чисел

faddp st1, st0 ; st0 = дробная часть

.continue\_fraction: ; Общая часть обработки дробной части

fabs ; Берем модуль дробной части

mov rcx, 6 ; Количество знаков после запятой

.fraction\_loop: ; Цикл умножения на 10

fld qword [ten] ; 10.0

fmulp ; Умножаем на 10

loop .fraction\_loop

fisttp qword [rbp-24] ; Сохраняем дробную часть как целое

lea r9, [rdi] ; Сохраняем адрес

mov rsi, [rbp-24]

mov rdi, r9

call int\_to\_string ; Преобразуем в строку

; Добавление знака '-'

cmp r10, 1

jne .end ; Если не отрицательное, пропускаем

lea rsi, [integer\_buffer]

call string\_shift\_right ; Сдвигаем строку вправо

mov byte [integer\_buffer], '-' ; Добавляем знак '-'

.end: ; Конец функции double\_to\_string

finit ; очистка стека FPU

pop rdi

mov rsp, rbp

pop rbp

ret

; Сдвигает строку вправо на одну позицию

; Вход: rsi - указатель на строку

string\_shift\_right:

push rbp

mov rbp, rsp

sub rsp, 64 ; место для временного буфера

mov rdi, rsp ; указатель на временный буфер

; Копирование строки во временный буфер

lea rsi, [integer\_buffer]

xor rcx, rcx

.copy\_loop: ; Цикл копирования

cmp byte [rsi + rcx], 0

je .find\_end ; если конец строки, переходим к .find\_end

mov al, [rsi + rcx]

mov [rdi + rcx], al

inc rcx

jmp .copy\_loop

.find\_end: ; Находим конец строки

dec rcx ; rcx теперь указывает на последний символ

lea rsi, [rsp]

.shift: ; Цикл сдвига

cmp rcx, 0

jl .shift\_done ; если достигли начала, завершаем сдвиг

mov al, [rsi + rcx]

mov [rsi + rcx + 1], al

dec rcx

jmp .shift

.shift\_done: ; Завершение сдвига

mov byte [rsi], ' ' ; заполняем первый символ пробелом

; Копируем обратно из временного буфера

lea rsi, [rsp]

lea rdi, [integer\_buffer]

xor rcx, rcx

.copy\_back: ; Цикл копирования обратно

cmp byte [rsi + rcx], 0

je .done

mov al, [rsi + rcx]

mov [rdi + rcx], al

inc rcx

jmp .copy\_back

.done:

add rsp, 64

pop rbp

ret

; Копирует строку из rsi в rdi

string\_copy:

push rbp

mov rbp, rsp

.loop: ; Цикл копирования строки

movsb ; копируем байт

cmp byte [rdi-1], 0 ; проверяем на конец строки

jne .loop ; если не конец строки, продолжаем цикл

pop rbp

ret

; Преобразует целое число в строку

; Вход: rsi - число, rdi - адрес буфера

int\_to\_string:

push r8

push r9 ; сохраняем r9 для выравнивания стека

push rdx

push rdi ; сохраняем rdi

mov r8, rdi ; сохраняем адрес буфера

mov rax, rsi ; число для преобразования

xor rcx, rcx ; счетчик цифр

.digit\_loop: ; Цикл преобразования числа в цифры

xor rdx, rdx ; очищаем rdx

mov rsi, 10 ; делитель

div rsi ; делим rax на 10, остаток в rdx

add rdx, '0' ; преобразуем остаток в ASCII

push rdx ; сохраняем цифру на стеке

inc rcx ; увеличиваем счетчик

or rax, rax ; проверяем, есть ли еще цифры

jnz .digit\_loop ; если есть, продолжаем цикл

mov r9, rcx ; сохраняем количество цифр

.write\_loop: ; Цикл записи цифр в буфер

pop rax ; извлекаем цифру из стека

stosb ; записываем цифру в буфер

loop .write\_loop ; повторяем, пока есть цифры

mov byte [rdi], 0 ; добавляем нуль-терминатор

pop rdi ; восстанавливаем rdi

pop rdx

pop r9 ; восстанавливаем r9

pop r8

ret

**2.2. Модуль makefile**

all:

nasm -f elf64 -g -o program.o program.asm -O0

ld -o program program.o

./program

**3. Сборка проекта**

make

**4. Выполнение программы**

**4.1. Запуск программы**

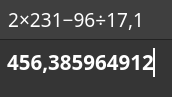
./program (хотя здесь программа запускается через make)

**4.2. Входные данные**

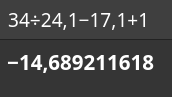
Enter a (float, range: 1.175494351e-38 to 3.402823466e+38): 17.1

Enter b (float, range: 1.175494351e-38 to 3.402823466e+38): 24.1

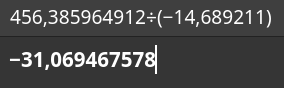
Enter d (int, range: -2147483648 to 2147483647): 231

**4.3. Ожидаемый результат выполнения**

Numerator: ~456.386

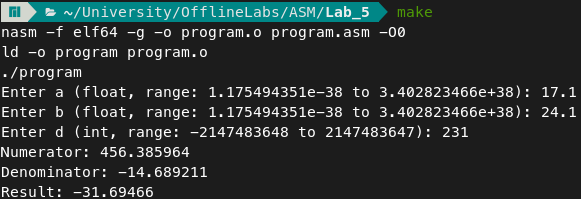


Denominator: ~-14.6892



Result: 3.17304

**4.4. Результат выполнения**



**5. Вывод**

Результат выполнения программы соответствует ожидаемому результату. Работа выполнена в полном объеме.