###### МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

###### Факультет информационных технологий

**Кафедра параллельных вычислений**

ОТЧЕТ

О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

«Введение в архитектуру x86/x86-64»

студента 2 курса, группы 20204

**Дронова Дениса Юрьевича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Кандидат технических наук

А. Ю. Власенко

Новосибирск 2021

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](#_Toc85105911)

[ЗАДАНИЕ 3](#_Toc85105912)

[ОПИСАНИЕ РАБОТЫ 4](#_Toc85105913)

[ЗАКЛЮЧЕНИЕ 6](#_Toc85105914)

[Приложение 1. *Ассемблерный листинг x86-64\_O0* 7](#_Toc85105915)

[Приложение 2. *Ассемблерный листинг x86-64\_O3* 10](#_Toc85105916)

[Приложение 3. *Листинг программы sin(x)* 12](#_Toc85105917)

# ЦЕЛЬ

1. Знакомство с программной архитектурой x86/x86-64.

2. Анализ ассемблерного листинга программы для архитектуры x86/x86-64 при разных уровнях компиляции.

# ЗАДАНИЕ

1. Изучить программную архитектуру x86/x86-64:

· набор регистров,

· основные арифметико-логические команды,

· способы адресации памяти,

· способы передачи управления,

· работу со стеком,

· вызов подпрограмм,

· передачу параметров в подпрограммы и возврат результатов,

· работу с арифметическим сопроцессором,

· работу с векторными расширениями.

2. Для программы на языке Си (из практической работы 1) сгенерировать ассемблерные листинги (синтаксис AT&T, принятый в UNIX) для архитектуры x86 или архитектуры x86-64, используя уровни оптимизации O0 и O3.

3. Проанализировать полученные листинги и сделать следующее:

· сопоставить команды языка Си с машинными командами;

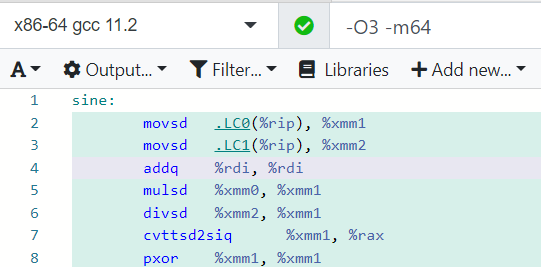
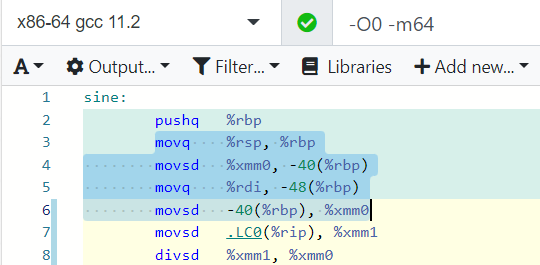
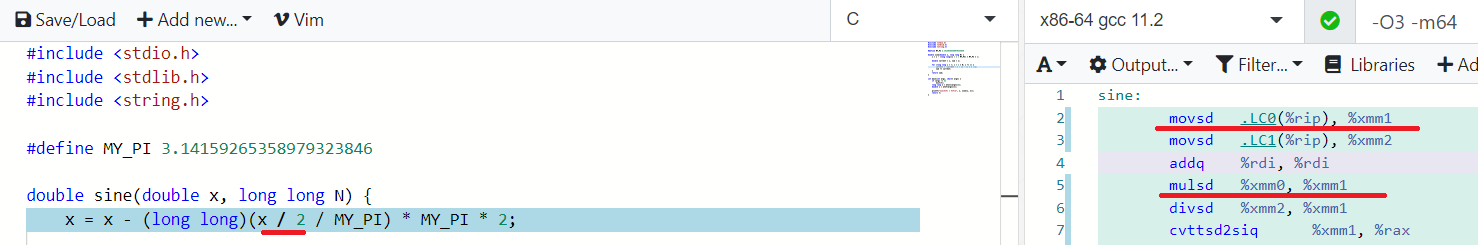
· определить размещение переменных языка Си в программах на ассемблере (в каких регистрах, в каких ячейках памяти);

· выписать оптимизационные преобразования, выполненные компилятором;

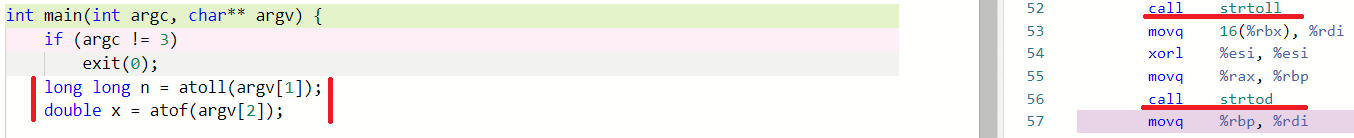
· сравнить различия в программах для архитектуры x86 и архитектуры x86-64.

# ОПИСАНИЕ РАБОТЫ

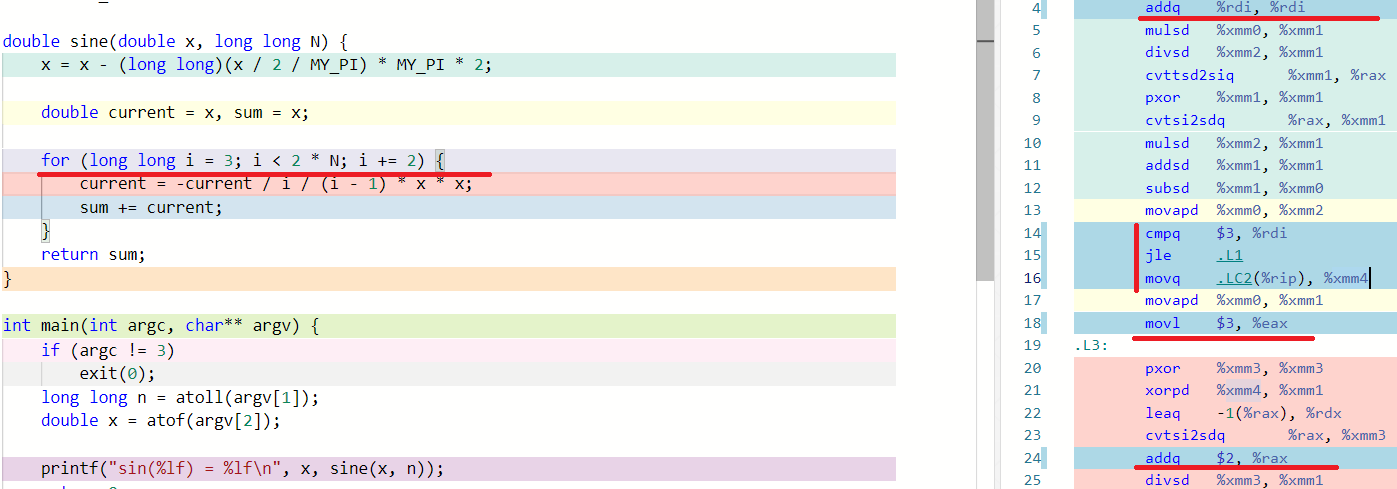
После компиляции программы, написанной на языке Си при помощи ключа -s и ключей оптимизации -O0, -O3, были получены ассемблерные листинги, приведенные в приложении, которые в последующем были проанализированы. Во время разбора были сопоставлены инструкции на языке Си с машинными программами. После анализа каждой ассемблерной программы были составлены комментарии и обнаружены следующие оптимизации компилятора:

1. **Исключение лишних копирований регистров**
2. **Замена операторов на их более низкоуровневые аналоги**

*Умножение на ½ вместо деления на 2*

1. **Замена некоторых функций на их более универсальный аналог**

*Использование strtoll вместо atoll, strtod вместо atof, т. к. функции вида ato при неудачной конвертации строки в число возвращают 0, что может привести к undefined behavior*

1. **Изменение порядка выполнения инструкций**

*Сначала производится умножение N на 2 путем сложения N c N, затем сравнение 2N c константой, равной 3, позже значение, равное 3 присваивается переменной i, и по окончании тела цикла к i прибавляется 2*

# ЗАКЛЮЧЕНИЕ

Во время выполнения практической работы была изучена архитектура x86/x84-64: набор регистров, основные арифметико-логические команды, работа со стеком, вызов подпрограмм. На основе анализа ассемблерных листингов можно прийти к выводу о том, что при компиляции на уровне O0 программный код практически дословно переводится в набор команд на ассемблерном коде. При компиляции на уровне O3 программный код подвергается оптимизациям, приведенным ранее.

# Приложение 1. *Ассемблерный листинг x86-64\_O0*

sine:

        pushq   %rbp ;сохранили указатель кадра программы, вызвавшей подпрограмму

                        ;(поместили в стек значение в регистре rbp)

        movq    %rsp, %rbp   ;формируем указатель новго кадра

                     ;в регистр ebp пишем значение регистра esp

        movsd   %xmm0, -40(%rbp)    ;поместить содержимое регистра

;xmm0 по адресу rbp-40(x)

        movq    %rdi, -48(%rbp)

;поместить содержимое регистра rdi по адресу rbp-48(N)

        movsd   -40(%rbp), %xmm0    ;записать в регистр xmm0 содержимое rbp-40(x)

        movsd   .LC0(%rip), %xmm1   ;записать в регистр xmm1

;разыменнованное значение

                    ;метки LC0 (константу 2)

        divsd   %xmm1, %xmm0        ;xmm0 = xmm0 / xmm1 (x = x / 2)

        movsd   .LC1(%rip), %xmm1   ;xmm1 = разыменнованное значение метки LC1

                                    ;(константа MY\_PI)

        divsd   %xmm1, %xmm0        ;xmm0 = xmm0 / xmm1 (x = x / MY\_PI)

        cvttsd2siq      %xmm0, %rax ;преобразование x, лежащего в xmm0 к longlong

                     ;и запись в регистр rax

        pxor    %xmm1, %xmm1        ;взаимоисключающее или MY\_PI с MY\_PI???

        cvtsi2sdq       %rax, %xmm1 ;преобразование к типу

        movsd   .LC1(%rip), %xmm0   ;поместить константу LC1 в регистр xmm0

        mulsd   %xmm1, %xmm0        ;умножить значение, вычеcленное ранее,

;и лежащее в xmm0

                         ;на константу MY\_PI, лежащую в xmm1, запись значения

                        ;в регистр xmm0

        movapd  %xmm0, %xmm1        ;xmm1 = xmm0

        addsd   %xmm0, %xmm1        ;xmm1 = xmm1 + xmm0 (xmm1 \* 2)

        movsd   -40(%rbp), %xmm0    ;xmm0 = значение лежащее по адресу rbp-40

                    ;xmm0 = x

        subsd   %xmm1, %xmm0     ;xmm0(x) = xmm0(x) - xmm1(вычесленное выражение)

        movsd   %xmm0, -40(%rbp) ;записать на адрес

;rbp-40 значение, лежащее в xmm0

        movsd   -40(%rbp), %xmm0    ;xmm0 = значение по адресу rbp-40(x)

        movsd   %xmm0, -8(%rbp)     ;записать по адресу rbp-8 значение ,лежащее

                    ;в xmm0 (current = x)

        movsd   -40(%rbp), %xmm0    ;xmm0 = значение по адресу rbp-40(x)

        movsd   %xmm0, -16(%rbp)    ;записать по адресу rbp-16 значение ,лежащее

                    ;в xmm0 (sum = x)

        movq    $3, -24(%rbp)      ;записать по адресу rbp-24 константу 3 (i = 3)

        jmp     .L2 ;переход к L2

.L3:

        movsd   -8(%rbp), %xmm0     ;xmm0 = current

        movq    .LC2(%rip), %xmm1   ;xmm1 = -1

        xorpd   %xmm1, %xmm0        ;xmm0 = (current) xor (-1)

        pxor    %xmm1, %xmm1 ;xmm1= xmm1(-1) pxor xmm1(-1) => 0 (чистка регистра)

        cvtsi2sdq   24(%rbp), %xmm1 ;преобразование значения,

; лежащего в rbp-24(i)

                     ;к типу? и запись в xmm1

        divsd   %xmm1, %xmm0        ;xmm0 = xmm0 / xmm1(xmm0 = -current / i)

        movq    -24(%rbp), %rax     ;rax = значение, лежащее по адресу rbp-24

                    ;(rax = i)

        subq    $1, %rax        ;rax = rax - 1

        pxor    %xmm1, %xmm1        ;(i pxor i)чистка регистра xmm1?

        cvtsi2sdq       %rax, %xmm1 ;преобразование значения rax к типу

                    ;и запись значения в регистр xmm1

        divsd   %xmm1, %xmm0        ;xmm0 = xmm0 / xmm1(i-1)

        mulsd   -40(%rbp), %xmm0    ;xmm0 = xmm0 \* (rbp-40)(x)

        movsd   -40(%rbp), %xmm1    ;xmm1 = rbp-40(x)

        mulsd   %xmm1, %xmm0        ;xmm0 = xmm0 \* xmm1(x)

        movsd   %xmm0, -8(%rbp)     ;current(rbp-8) = xmm0

        movsd   -16(%rbp), %xmm0    ;xmm0 = sum(rbp-16)

        addsd   -8(%rbp), %xmm0     ;xmm0 = xmm0+current(rbp-8)

        movsd   %xmm0, -16(%rbp)    ;sum(rbp-16) = xmm0;

        addq    $2, -24(%rbp)       ;i+=2

.L2: ;осуществление цикла

        movq    -48(%rbp), %rax     ;rax = значение по адресу rbp - 48(N)

        addq    %rax, %rax          ;умножение N на два(запись в тот же регистр)

        cmpq    %rax, -24(%rbp)     ;сравнение 2N с i

        jl      .L3 ;переход к L3 если i меньше 2N

        movsd   -16(%rbp), %xmm0    ;xmm0 = sum(rbp-16)

        movq    %xmm0, %rax     ;rax = xmm0

        ;выход из функции

        movq    %rax, %xmm0

        popq    %rbp    ;извлечение из стека места продолжения программы

                        ;после выхода из функции

        ret

.LC3:

        .string "sin(%lf) = %lf\n"

main:

        pushq   %rbp           ;сохраняем указатель кадра вызвавшей программы

        movq    %rsp, %rbp     ;формируем указатель кадра

        subq    $32, %rsp      ;выделяем место на стеке для нашего кадра

        movl    %edi, -20(%rbp)    ;rbp-20 = argc

        movq    %rsi, -32(%rbp)    ;rbp-32 = argv

        cmpl    $3, -20(%rbp)      ;проверка argc != 3

        je      .L6        ;если argc == 3 --> переход к L6

        movl    $0, %edi       ;exit(0)

        call    exit           ;вызов подпрограммы exit

.L6:

        movq    -32(%rbp), %rax    ;rax = значение по адресу (rbp-32)

        addq    $8, %rax       ;rax + 8 (argv[1])

        movq    (%rax), %rax   ; Записать в регистр rax операнд,

                   ;который содержится в оперативной памяти по адресу метки rax

        movq    %rax, %rdi     ;поместить содержиоме регистра rax в регистр rdi

        call    atoll          ;вызов подпрограммы atoll

        movq    %rax, -8(%rbp) ;поместить значение реистра rax

;по адресу rbp-8 (n = atoll)

        movq    -32(%rbp), %rax    ;rax = значение по адресу (rbp-32)

        addq    $16, %rax      ;rax + 16 (argv[2])

        movq    (%rax), %rax       ; Записать в регистр rax операнд,

                   ;который содержится в оперативной памяти по адресу метки rax

        movq    %rax, %rdi     ;поместить содержиоме регистра rax в регистр rdi

        call    atof           ;вызов подпрограммы atof

        movq    %xmm0, %rax    ;поместить содержимое xmm0 в регистр rax

        movq    %rax, -16(%rbp)    ;x = atof

        movq    -8(%rbp), %rdx     ;rdx = n

        movq    -16(%rbp), %rax    ;rax = x

        movq    %rdx, %rdi

        movq    %rax, %xmm0

        call    sine

        movq    %xmm0, %rdx    ;rdx = sine result

        movq    -16(%rbp), %rax    ;rax = x

        movq    %rdx, %xmm1

        movq    %rax, %xmm0

        movl    $.LC3, %edi    ;поместили в edi значение метки .LC3(строку вывод)

        movl    $2, %eax       ;eax содержит кол-во параметров printf

        call    printf

        movl    $0, %eax

        leave   ;Это эквивалентно movl %ebp, %esp; popl %ebp

                ;Так мы восстанавливаем состояние стека и кадра,

                ;которые были до вызова

        ret

.LC0: ;константа (2)

        .long   0

        .long   1073741824

.LC1: ;константа (MY\_PI)

        .long   1413754136

        .long   1074340347

.LC2: ;константа (-1)

        .long   0

        .long   -2147483648

        .long   0

        .long   0

# Приложение 2. *Ассемблерный листинг x86-64\_O3*

sine:

    ;xmm0 = x;

    ;rdi = N;

    movsd   .LC0(%rip), %xmm1 ;xmm1 = 1/2

    movsd   .LC1(%rip), %xmm2 ;xmm2 = MY\_PI

    addq    %rdi, %rdi    ;rdi = 2 \* rdi (rdi = 2N)

    mulsd   %xmm0, %xmm1      ;xmm1 = xmm1 \* xmm0 (xmm1 = 1/2 \* x)

    divsd   %xmm2, %xmm1      ;xmm1 = xmm1 / MY\_PI

    cvttsd2siq      %xmm1, %rax ;приведение к типу long long

                    ;и запись в rax

    pxor    %xmm1, %xmm1     ;очистка xmm1;

    cvtsi2sdq       %rax, %xmm1 ;приведение значения, лежащего по адресу rax

                    ;к типу double и запись в xmm1

    mulsd   %xmm2, %xmm1    ;xmm1 = xmm1 \* MY\_PI

    addsd   %xmm1, %xmm1    ;xmm1 \*= 2

    subsd   %xmm1, %xmm0    ;xmm0 = xmm0 - xmm1 (x = x - xmm1)

    movapd  %xmm0, %xmm2    ;sum = x

    cmpq    $3, %rdi    ;сравнение 3 и 2N

    jle     .L1     ;если 2N <= 3 -> выход из цикла

    movq    .LC2(%rip), %xmm4 ;xmm4 = -1

    movapd  %xmm0, %xmm1    ;current = x

    movl    $3, %eax    ;eax = 3 (i = 3)

.L3:

    pxor    %xmm3, %xmm3    ;очистка xmm3

    xorpd   %xmm4, %xmm1    ;current \*= -1

    leaq    -1(%rax), %rdx  ;rdx = адрес(rax - 1) (к значению по адресу

                ;rax - 1 обращения нет

    cvtsi2sdq       %rax, %xmm3 ;приведение к типу и запись в xmm3

    addq    $2, %rax    ;i += 2

    divsd   %xmm3, %xmm1    ;xmm1 = -current(xmm1) / i(xmm3)

    pxor    %xmm3, %xmm3    ;очистка xmm3

    cvtsi2sdq       %rdx, %xmm3 ;xmm3 = i - 1

    divsd   %xmm3, %xmm1    ;xmm1 = xmm1 / (i-1)

    mulsd   %xmm0, %xmm1    ;xmm1 = xmm1 \* xmm0 (xmm1 \*= x)

    mulsd   %xmm0, %xmm1    ;xmm1 = xmm1 \* xmm0 (xmm1 \*= x)(xmm1 - ccurrent)

    addsd   %xmm1, %xmm2    ;sum += current

    cmpq    %rdi, %rax  ;сравнение 2N и i

    jl      .L3     ;если i < 2N переход к очередной итерации цикла

.L1:

    movapd  %xmm2, %xmm0 ;retval(xmm0) = sum(xmm2)

    ret

.LC3:

    .string "sin(%lf) = %lf\n"

main:

    pushq   %rbp        ;формирование кадра стека

    pushq   %rbx        ;rbx - регистр общего назначения

    subq    $8, %rsp    ;выделяем место на стеке

    cmpl    $3, %edi    ;сравнение argc != 3 (в edi лежит argc)

    je      .L8     ;если argc == 3, то переход к L8

    xorl    %edi, %edi  ;очистка регистра edi

    call    exit        ;вызов exit

.L8: ;выполняется если argc == 3

    ;si - индекс источника

    ;di - индекс назначения

    movq    8(%rsi), %rdi   ;rdi = содержимое(rsi + 8) (rdi = argv[1])

    movl    $10, %edx   ;edx = 10;

    movq    %rsi, %rbx  ;rbx = содержимое(rsi)

    xorl    %esi, %esi  ;очистка esi

    ;strtoll выводит ошибку, если не удалось

;произвести конвертацию строки к числу

    ;atoll возвращает 0, если не удалось произвести конвертацию

    call    strtoll

    movq    16(%rbx), %rdi  ;rdi(n) = result(atoll(argv[1]))

    xorl    %esi, %esi  ;очистка esi

    movq    %rax, %rbp  ;rbp = rax

    call    strtod

    movq    %rbp, %rdi  ;rdi = n

    movapd  %xmm0, %xmm5    ;xmm5 = x

    call    sine    ;вызывается от xmm0 и rdi

    movl    $.LC3, %edi ;edi = -1

    movl    $2, %eax    ;eax = 2(кол-во параметров printf)

    ;xmm0 = result(sine)

    movapd  %xmm0, %xmm1    ;xmm1 = result(sine)

    movapd  %xmm5, %xmm0    ;xmm0 = x

    call    printf

    popq    %rdx        ;|

    xorl    %eax, %eax  ;|

    popq    %rbx        ;|

    popq    %rbp        ;|

    ret

.LC0: ;(2)

    .long   0

    .long   1071644672

.LC1: ;(MY\_PI)

    .long   1413754136

    .long   1074340347

.LC2: ;(-1)

    .long   0

    .long   -2147483648

    .long   0

    .long   0

# Приложение 3. *Листинг программы sin(x)*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MY\_PI 3.14159265358979323846

double sine(double x, long long N) {

    x = x - (long long)(x / 2 / MY\_PI) \* MY\_PI \* 2;

    double current = x, sum = x;

    for (long long i = 3; i < 2 \* N; i += 2) {

        current = -current / i / (i - 1) \* x \* x;

        sum += current;

    }

    return sum;

}

int main(int argc, char\*\* argv) {

    if (argc != 3)

        exit(0);

    long long n = atoll(argv[1]);

    double x = atof(argv[2]);

    printf("sin(%lf) = %lf\n", x, sine(x, n));

    return 0;

}