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 22206

***Тропина Никиты Васильевича***

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

доцент, к.т.н.

А.Ю.Власенко

Новосибирск 2023

# СОДЕРЖАНИЕ

[ЦЕЛИ 3](#_Toc148802017)

[ЗАДАНИЕ 3](#_Toc148802018)

[ОПИСАНИЕ РАБОТЫ 4](#_Toc148802019)

[ЗАКЛЮЧЕНИЕ 5](#_Toc148802020)

[Приложение 1. Реализованная программа на C++. 6](#_Toc148802021)

[Приложение 2. Сгенерированный листинг на ассемблере под архитектуру x86-64 с уровнем оптимизации -O0. 7](#_Toc148802022)

[Приложение 3. Сгенерированный листинг на ассемблере под архитектуру x86-64 с уровнем оптимизации -Ofast. 10](#_Toc148802023)

# ЦЕЛИ

1. Изучить программную архитектуру x86/x86-64.
2. Проанализировать ассемблерные листинги программы для архитектуры x86/x86-64.

# 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.

1. Для программы на языки 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 один из уровней O3/Ofast.
2. Проанализировать полученные листинги и сделать следующее:

* сопоставить команды языки Си с машинными командами;
* определить размещение переменных языка Си в программах на ассемблере (в каких регистрах, в каких ячейках памяти);
* выписать о оптимизационные преобразования, выполненные компилятором;
* (опционально) сравнить различия в программах для архитектуры x86 и архитектуры x86-64.

1. Составить отчет по лабораторной работе.

# ОПИСАНИЕ РАБОТЫ

В ходе практической работы были сгенерированы ассемблерные листинги при помощи godbolt.org для архитектуры x86-64, используя уровни оптимизации O0 и Ofast, на основе ранее реализованной программы на языки C++ из практической работы «Определение времени работы прикладных программ». Программа представлена в Приложении 1.

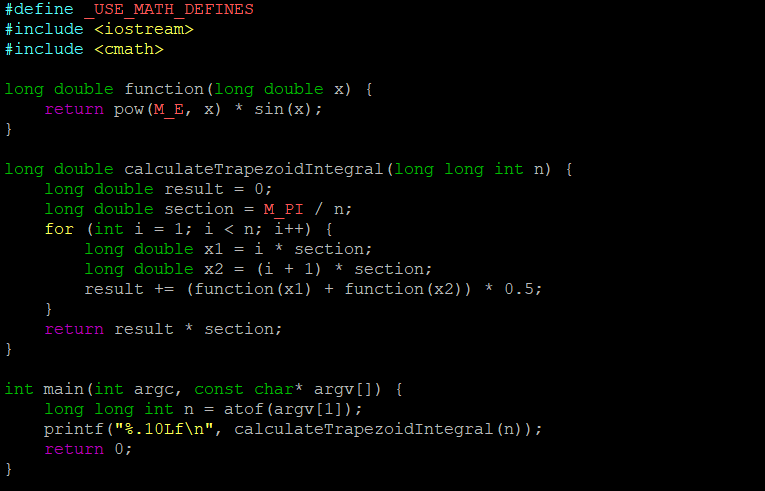
Полученные листинги были проанализированы. Команды языка C++ были сопоставлены с командами ассемблера.

Сравнив листинги, сгенерированные одним компилятором с разными ключами оптимизации, были выявлены следующие оптимизационные преобразования: уменьшенье обращения к памяти, инлайнинг функций, использование других функций, уменьшение размера используемой памяти, применение быстрого обнуления регистров, уменьшение кол-ва вычислений для ускорения работы.

# ЗАКЛЮЧЕНИЕ

При рассмотрении сгенерированных ассемблерных листингов под архитектуру x86-64 с уровнями оптимизации O0 и Ofast выявлены значительные отличия. На уровне O0 ассемблерный листинг исполняется в соответствии с исходным кодом, а на уровне Ofast видны оптимизационные преобразования, описанные в ходе работы.

# Приложение 1. Реализованная программа на C++.



# Приложение 2. Сгенерированный листинг на ассемблере под архитектуру x86-64 с уровнем оптимизации -O0.

function(long double):

        pushq   %rbp // занести rbp в стек

        movq    %rsp, %rbp // скопировать значение с вершины стека в rbp

        subq    $16, %rsp // выделить в стеке 16 байт

        fldt    16(%rbp) // запушить x из адреса

        fstpl   -8(%rbp) // записать st0 из стека и выпихнуть его

        movq    .LC0(%rip), %rax // записать е в rax

        movsd   -8(%rbp), %xmm1 // записать x в xmm1

        movq    %rax, %xmm0 // записать е в xmm0

        call    pow

        movsd   %xmm0, -8(%rbp) // результат возведения в степень в xmm0 копируем по адресу

        fldt    16(%rbp) // запушить x в st0

        fstpl   -16(%rbp) // записать st0 по адресу и выпихнуть

        movsd   -16(%rbp), %xmm0 // записать x в xmm0

        call    sin

        movapd  %xmm0, %xmm2 // копируем результат sin в xmm2

        mulsd   -8(%rbp), %xmm2 // произведение sin и степени е

        movsd   %xmm2, -8(%rbp) // результат копируем из адреса

        fldl    -8(%rbp) // результат пушим в st0

        leave // return, выход из функции

        ret

calculateTrapezoidIntegral(long long):

        pushq   %rbp // запушить rbp в стек

        movq    %rsp, %rbp // копировать значение rsp в rbp

        subq    $112, %rsp // выделить 112 байт на стеке

        movq    %rdi, -88(%rbp) // поместить n по адресу

        fldz // загрузить число 0.0

        fstpt   -16(%rbp) // записать по адресу 0 и вытолкнуть с регистров сопроцессора

        pxor    %xmm1, %xmm1 побитовое, исключающее ИЛИ с n

        cvtsi2sdq       -88(%rbp), %xmm1 // преобразовать n в long double

        movsd   .LC3(%rip), %xmm0 //перемещение скалярного double π в xmm0

        divsd   %xmm1, %xmm0 // скалярное деление π на n, результат в xmm0

        movsd   %xmm0, -112(%rbp) // поместить значение xmm0 по адресу

        fldl    -112(%rbp) // загрузить на вершину стека число

        fstpt   -48(%rbp) // записать и вытолкнуть число со стека (section)

        movl    $0, -20(%rbp) // i = 0

        jmp     .L4 // for

.L5:

        fildl   -20(%rbp) // запушить целое число i в st0

        fldt    -48(%rbp) // запушить section в st0

        fmulp   %st, %st(1) // перемножить, результат в st(1) вытолкнуть

        fstpt   -64(%rbp) // вытолкнуть x1 по адресу

        movl    -20(%rbp), %eax // i копировать в eax

        addl    $1, %eax // прибавить 1

        movl    %eax, -112(%rbp) // записать i

        fildl   -112(%rbp) // запушить i в st0

        fldt    -48(%rbp) // запушить section в st0

        fmulp   %st, %st(1) // перемножить, результат в st(1) вытолкнуть

        fstpt   -80(%rbp) // вытолкнуть x2 по адресу

        pushq   -56(%rbp) // запушить значение с этого адреса

        pushq   -64(%rbp) // запушить x1

        call    function(long double)

        addq    $16, %rsp // освободить 16 байт в стеке

        fstpt   -112(%rbp) // вытолкнуть из st0 результат функции в адрес

        pushq   -72(%rbp) // запушить значение по адресу

        pushq   -80(%rbp) // запушить x2

        call    function(long double)

        addq    $16, %rsp // освободить 16 байт в стеке

        fldt    -112(%rbp) // запушить в st0 результат функции для x1 в адрес

        faddp   %st, %st(1) // сложить результаты функций

        fldt    .LC4(%rip) // загрузить константу 0.5

        fmulp   %st, %st(1) // перемножить

        fldt    -16(%rbp) // запушить result в st0 из адреса

        faddp   %st, %st(1) // сложить result и полученное значение

        fstpt   -16(%rbp) // значение st0 записать по адресу и выпихнуть из стека

        addl    $1, -20(%rbp) // добавить к i единицу

.L4:

        movl    -20(%rbp), %eax // i = 0, поместить в аккумулятор eax

        cltq // конвертирует eax в четверное слово

        cmpq    %rax, -88(%rbp) // сравнение i с n

        jg      .L5 // тело for

        fldt    -16(%rbp) // запушить в st0

        fldt    -48(%rbp) // запушить в st0 section

        fmulp   %st, %st(1) // умножение этих чисел

        leave // return

        ret

.LC5:

        .string "%.10Lf\n"

main:

        pushq   %rbp // поместить rbp в стек

        movq    %rsp, %rbp // скопировать значение rsp в rbp

        subq    $32, %rsp // выделить 32 байта в стеке

        movl    %edi, -20(%rbp) // запись argc

        movq    %rsi, -32(%rbp) // запись argv

        movq    -32(%rbp), %rax // копирование argv в rax

        addq    $8, %rax // адрес argv[1]

        movq    (%rax), %rax // в rax argv[1]

        movq    %rax, %rdi // копирование argv[1] в rdi

        call    atof // передали rdi

        cvttsd2siq      %xmm0, %rax // результат в xmm0 преобразовали в double, результат поместили в rax

        movq    %rax, -8(%rbp) // скопировали n по адресу

        movq    -8(%rbp), %rax // передали n в rax

        movq    %rax, %rdi // копировать n в rdi

        call    calculateTrapezoidIntegral(long long)

        leaq    -16(%rsp), %rsp // считает адрес и пишет его в вершину стека

        fstpt   (%rsp) // записать result по адресу и вытолкнуть со стека

        movl    $.LC5, %edi // присвоить значение константы в edi

        movl    $0, %eax // инициализируем выход на 0

        call    printf

        addq    $16, %rsp // высвобождаем 16 байт в стеке

        movl    $0, %eax // return

        leave // выход из функции

        ret

.LC0:

        .long   -1961601175

        .long   1074118410

.LC3:

        .long   1413754136

        .long   1074340347

.LC4:

        .long   0

        .long   -2147483648

        .long   16382

        .long   0

# Приложение 3. Сгенерированный листинг на ассемблере под архитектуру x86-64 с уровнем оптимизации -Ofast.

function(long double):

        subq    $24, %rsp // выделить 24 байта на стеке

        fldt    32(%rsp) // запушить из адреса значение х в st0

        fstpl   (%rsp) // из st0 копировать х по адреса

        movsd   (%rsp), %xmm2 // копировать скалярное значение из адреса в xmm2

        movapd  %xmm2, %xmm0 // копировать х из xmm2 в xmm0

        call    exp

        movsd   %xmm0, 8(%rsp) // значение экспоненты записать по адресу

        movsd   (%rsp), %xmm0 // копировать х в xmm0

        call    sin

        movsd   8(%rsp), %xmm1 // копировать экспоненту в xmm1

        mulsd   %xmm0, %xmm1 // произведение e \* sin

        movsd   %xmm1, (%rsp) // результат записать по адреса

        fldl    (%rsp) // запушить результат в стек

        addq    $24, %rsp // высвободить 24 байта со стекка

        ret

calculateTrapezoidIntegral(long long):

        pushq   %rbp // поместить rbp на вершину стека

        pxor    %xmm0, %xmm0 // result = 0

        pushq   %rbx // поместить rbx в стек

        cvtsi2sdq       %rdi, %xmm0 // поместить n в xmm0

        subq    $72, %rsp // выделить 72 байта на стеке

        movsd   .LC4(%rip), %xmm1 // загрузить π в xmm1

        divsd   %xmm0, %xmm1 // разделить π на n

        movsd   %xmm1, (%rsp) // переместить результат деления по адресу

        fldl    (%rsp) // запушить в st0 результат

        fstpt   48(%rsp) // достать из st0 значение section и выпихнуть его

        testq   %rdi, %rdi // сравнение с 0

        jle     .L7 // начало for

        fldz // загрузить 0, result = 0

        fldz // загрузить 0

        movq    %rdi, %rbp // копировать rdi в rbp

        xorl    %ebx, %ebx // обнуление ebx, i = 0

.L6:

        fstpt   32(%rsp) // записать из st0 в адрес и выпихнуть со стека

        addl    $1, %ebx // добавить 1 к i

        movl    %ebx, (%rsp) // копировать i в rsp

        fstpt   16(%rsp) // записать и вытолкнуть число в адрес

        fildl   (%rsp) // загрузить целое i на стек в виде вещественного числа

        fldt    48(%rsp) // загрузить значение section в st0

        fmulp   %st, %st(1) // перемножить i и section = x2

        fstpl   (%rsp) // достать x2 из st0 и положить в rsp

        movsd   (%rsp), %xmm3 // переместить x2 в xmm3

        movapd  %xmm3, %xmm0 // перемещение упакованных выровненных double

        call    exp

        movsd   %xmm0, 8(%rsp) // результат exp из xmm0 копировать по адресу

        movsd   (%rsp), %xmm0 // поместить x2 в xmm0

        call    sin

        movsd   8(%rsp), %xmm2 // результат exp копировать в xmm2

        mulsd   %xmm0, %xmm2 // умножение sin и exp

        movsd   %xmm2, (%rsp) // результат умножения копировать по адресу

        fldl    (%rsp) // результат запушить в st0

        fldt    16(%rsp) // запушить в st0 значение из адреса

        fadd    %st(1), %st // сложить значения функций для двух секций

        fmuls   .LC5(%rip) // умножить st0 на константу 0.5

        fldt    32(%rsp) // запушить result в st0

        faddp   %st, %st(1) // прибавить к result полученное значение

        cmpl    %ebx, %ebp // сравнение i и n

        jne     .L6 // повторение цикла

        fstp    %st(1) // значение st1 запушить в st0

        fldt    48(%rsp) // вытащить result из st0

        addq    $72, %rsp // освободить 72 байта со стека

        popq    %rbx // вытащить из стека rbx

        popq    %rbp

        fmulp   %st, %st(1)

        ret

.L7:

        addq    $72, %rsp // освободить 72 байта из стека

        fldz // пуш 0 в st0

        popq    %rbx // вытащить вершину стека в rbx

        popq    %rbp // вытащить вершину стека в rbp

        ret

.LC7:

        .string "%.10Lf\n"

main:

        subq    $8, %rsp // выделить 8 байт на стеке

        movq    8(%rsi), %rdi // записать argv[1] в rdi

        xorl    %esi, %esi // исключающее или (обнуление младшей части rsi)

        call    strtod // запись argv[1] в xmm0

        cvttsd2siq      %xmm0, %rdi // копирование n в rdi

        call    calculateTrapezoidIntegral(long long)

        subq    $16, %rsp // выделить 16 байт на стеке

        movl    $.LC7, %edi // записать строку в edi

        xorl    %eax, %eax // обнуление регистра

        fstpt   (%rsp) // записать в адрес st0 и вытолкнуть его

        call    printf

        xorl    %eax, %eax // вернуть в return 0

        addq    $24, %rsp // освободить 24 байта со стека

        ret // выход

.LC4:

        .long   1413754136

        .long   1074340347

.LC5:

        .long   1056964608