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 ЛАБОРАТОРНОЙ РАБОТЫ

«Введение в архитектуру x86/x86-64»

Студента 2 курса, 21211 группы

Петрова Сергея Евгеньевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Антон Юрьевич Кудинов

Новосибирск 2022

СОДЕРЖАНИЕ

ЦЕЛЬ

- *Знакомство с программной архитектурой x86/x86-64;*
- *Анализ ассемблерного листинга 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;*
2. *Написать программу на языке C или C++, которая реализует алгоритм вычисления числа Пи методом Монте-Карло. Алгоритм состоит в следующем. Сначала в квадрат с центром в начале координат и со стороной два вписывается круг с единичным радиусом. Затем в этом квадрате случайным образом с равномерным распределением генерируются N точек. Точка может попасть в окружность или нет (условие попадания $x^2 + y^2 \leq 1$). Далее определяется число M точек, попавших в круг. При достаточно большом числе бросков N, по значениям M и N вычисляется число Пи:*

$$\pi \approx \frac{4M}{N};$$

3. *Для программы сгенерировать ассемблерные листинги для архитектуры x86 и архитектуры x86-64, используя различные уровни комплексной оптимизации;*
4. *Проанализировать полученные листинги и сделать следующее:*
    - *Сопоставьте команды языка Си с машинными командами;*
    - *Определить размещение переменных языка Си в программах на ассемблере (в каких регистрах, в каких ячейках памяти);*
    - *Описать и объяснить оптимизационные преобразования, выполненные компилятором;*
    - *Продемонстрировать использование ключевых особенностей архитектур x86 и x86-64 на конкретных участках ассемблерного кода;*
    - *Сравнить различия в программах для архитектуры x86 и архитектуры x86-64;*
5. *Составить отчет по лабораторной работе. Отчет должен содержать следующее:*
    - *Титульный лист;*
    - *Цель лабораторной работы;*
    - *Полный компилируемый листинг реализованной программы и команды для ее компиляции;*

- *Листинг на ассемблере с описаниями назначения команд с точки зрения реализации алгоритма выбранного варианта;*
- *Вывод по результатам лабораторной работы;*

ОПИСАНИЕ РАБОТЫ

Пошаговое описание выполненной работы

1. *Сгенерировал ассемблерные листинги при помощи godbolt.org, используя следующие наборы флагов `-m32 -O0`, `-m32 -O3`, `-m64 -O0`, `-m64 -O3`;*
2. *Сопоставил команды исходного кода на Си с машинными командами, оставляя комментарии в листинге (см. Приложение 2);*

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы:

- *Познакомился с программной архитектурой x86/x86-64;*
- *Проанализировал ассемблерный листинг программы для архитектуры x86/x86-64;*

По результатам проведённых исследований можно сделать следующие выводы:

- *Архитектура x86 и уровень оптимизации O0:*
  - *Данные хранились в основном в стеке вызовов;*
  - *До вызова функции аргументы помещались в вершину стека вызовов (8, 13, 93 строки);*
  - *Вещественные вычисления производились в сопроцессоре;*
  - *До выполнения арифметических операций все необходимые данные заносились в стек сопроцессора, например, константы из меток .LC1, .LC3 загружались в стек сопроцессора (26-30, 55-59 строки);*
  - *Значения типа double помещались в стек вызова двумя частями: старшей и младшей (102-103 строки);*
- *Архитектура x86 и уровень оптимизации O3:*
  - *Данные хранились в стеке вызовов и регистрах общего назначения (ebx, esi);*
  - *До вызова функции аргументы помещались в вершину стека вызовов (7, 9, 81 строки);*
  - *Вещественные вычисления производились в сопроцессоре;*
  - *До выполнения арифметических операций не все необходимые данные заносились в стек сопроцессора, например, константы из меток .LC1, .LC3 не загружались в стек сопроцессора (25-27, 44-46 строки);*
  - *Используются побитовые операции, например, вместо присвоения регистру 0 используется XOR (19, 92 строки);*
- *Архитектура x86-64 и уровень оптимизации O0:*
  - *Данные хранились в основном в стеке вызовов;*
  - *До вызова функции аргументы помещались в регистры rdi и xmm0 (7, 9, 78 строки);*
  - *Все вещественные вычисления выполнялись в блоке XMM;*
  - *До выполнения арифметических операций все необходимые данные заносились в стек сопроцессора, например, константы из меток .LC1, .LC3 загружались в регистры блока XMM (21-25, 47-50 строки);*
  - *Для приведения типов использовалась команда CVTSI2SDL*
  - *32 битные данные хранятся в младших частях регистров (7 строка - edi, 56 строка - eax);*
- *Архитектура x86-64 и уровень оптимизации O3:*
  - *Данные хранились в основном в стеке вызовов;*
  - *До вызова функции аргументы помещались в регистры rdi и xmm0 (6, 12, 75 строки);*
  - *Все вещественные вычисления выполнялись в блоке XMM;*

- *До выполнения арифметических операций не все необходимые данные заносились в стек сопроцессора, например, константы из меток .LC1, .LC3 не загружались в блок XMM (24-26 строки);*
- *Для приведения типов использовалась команда CVTSI2SDL*
- *32 битные данные хранятся в младших частях регистров (6 строка - edi, 18 строка - eax, 19 строка - r14d);*

ПРИЛОЖЕНИЕ 1 (ЛИСТИНГ ПРОГРАММЫ НА СИ)

src / main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

double MonteCarloAlgorithm(int count)
{
    double insideCount = 0.0;
    srand(time(NULL));

    for (int i = 0; i < count; ++i)
    {
        double x = (double)rand() / RAND_MAX;
        double y = (double)rand() / RAND_MAX;

        if ((x * x) + (y * y) <= 1.0)
        {
            insideCount += 4.0;
        }
    }

    return insideCount / count;
}

int main()
{
    int count = 100000000;

    double pi = MonteCarloAlgorithm(count);
    printf("PI: %lf\n", pi);

    return EXIT_SUCCESS;
}
```

ПРИЛОЖЕНИЕ 2 (АССЕМБЛЕРНЫЕ ЛИСТИНГИ ПРОГРАММЫ)

Ассемблерный листинг для архитектуры x86 и уровнем оптимизации O0

```
1.  MonteCarloAlgorithm:
2.          pushl   %ebp                    // добавить адрес возврата в стек
3.          movl    %esp, %ebp              // запомнить адрес текущего кадра стека
4.          subl    $56, %esp               // зарезервировать 56 байт для локальных
5.                                          // переменных
6.          subl    $12, %esp               // зарезервировать ещё 12 байт для
7.                                          // выполнения srand(time(NULL));
8.          pushl   $0                      // добавить в стек NULL для time
9.          call    time                    // вызов time
10.         addl    $16, %esp               //
11.         subl    $12, %esp               // сместить esp, чтобы исключить NULL
12.                                         // из стека
13.         pushl   %eax                    // поместить значение eax в стек (в eax
14.                                         // хранится результат time)
15.         call    srand                   // вызов srand
16.         addl    $16, %esp               // сместить esp, чтобы исключить значение
17.                                         // eax и вернуть 12 байт
18.         fldz                            // загрузить 0 в стек сопроцессора
19.                                         // (инициализация double insideCount)
20.         fstpl   -16(%ebp)               // сохранить st(0) в стек
21.         movl    $0, -20(%ebp)           // создать i и присвоить 0
22.         jmp     .L2                     // безусловный переход в метку .L2
23. .L5:
24.         call    rand                    // вызов rand
25.         movl    %eax, -52(%ebp)         // копировать eax(результат rand()) в стек
26.         fildl   -52(%ebp)               // загрузить результат rand() в стек
27.                                         // сопроцессора и преобразовать в double
28.         fldl    .LC1                    // загрузить константу из метки .LC1 в
29.                                         // стек сопроцессора
30.         fdivrp  %st, %st(1)             // разделить результат st(1) на st(0),
31.                                         // присвоить в st(1) и вытолкнуть
32.                                         // st(0) (rand() / RAND_MAX)
33.         fstpl   -32(%ebp)               // извлечь st(0) в стек (double x)
34.         call    rand                    // вызов rand
35.         movl    %eax, -52(%ebp)         // копировать eax(результат rand) в стек
36.         fildl   -52(%ebp)               // загрузить результат rand в стек
37.                                         // сопроцессора и преобразовать в double
38.         fldl    .LC1                    // загрузить константу из метки .LC1 в
39.                                         // стек сопроцессора
40.         fdivrp  %st, %st(1)             // разделить результат st(1) на st(0),
41.                                         // присвоить в st(1) и вытолкнуть
42.                                         // st(0) (rand() / RAND_MAX)
43.         fstpl   -40(%ebp)               // извлечь st(0) в стек (double y)
44.         fldl    -32(%ebp)               // загрузить x в стек сопроцессора
45.         fld     %st(0)                  // дублировать вершину стека сопроцессора
46.         fmulp   %st, %st(1)             // умножить x на x и вытолкнуть st(0)
47.         fldl    -40(%ebp)               // загрузить y в стек сопроцессора
48.         fmul    %st(0), %st             // умножить y на y
49.         faddp   %st, %st(1)             // сложить x*x + y*y и вытолкнуть st(0)
50.         fld1                            // загрузить 1 в стек сопроцессора
51.         fcomip  %st(1), %st             // сравнить st(1) и st(0) (x*x+y*y и 1.0)
52.         fstp    %st(0)                  // сохранить вершину стека в st(0)
53.         jb      .L3                     // переход в метку .L3,
54.                                         // если по результатам fcomip %st(1) < %st
55.         fldl    -16(%ebp)               // загрузить insideCount в
56.                                         // стек сопроцессора
57.         fldl    .LC3                    // загрузить константу из метки
58.                                         // .LC3 (4.0) в стек сопроцессора
59.         faddp   %st, %st(1)             // прибавить к 4.0 insideCount
60.         fstpl   -16(%ebp)               // извлечь st(0) в стек (присвоить
61.                                         // результат сложения insideCount)
62. .L3:
```

```
63.         addl    $1, -20(%ebp)           // прибавить 1 к i (++i)
64.  .L2:
65.         movl    -20(%ebp), %eax         // копировать i в eax
66.         cmpl    8(%ebp), %eax           // сравнить на равенство i и count
67.         jb      .L5                     // переход в метку .L5, если i < count
68.         movd       8(%ebp), %xmm0       // копировать count в регистр xmm0
69.         movq       %xmm0, -48(%ebp)     // занести xmm0 в стек
70.         fildq   -48(%ebp)               // загрузить count в стек сопроцессора
71.         fldl    -16(%ebp)               // загрузить insideCount в
72.                                         // стек сопроцессора
73.         fdivp   %st, %st(1)             // разделить st(1) на st(0)
74.                                         // (insideCount / count)
75.         leave                           // сбросить кадр стека
76.         ret                             // выход из подпрограммы
77.  .LC5:
78.         .string "PI: %lf\n"
79.  main:
80.         leal    4(%esp), %ecx           // вычислить эффективный адрес 4(%esp)
81.                                         // и поместить в ecx
82.         andl    $-16, %esp              // логическое и -16 & %esp
83.         pushl   -4(%ecx)                // добавить в стек содержимое -4(%ecx)
84.         pushl   %ebp                    // добавить адрес возврата в стек
85.         movl    %esp, %ebp              // запомнить адрес текущего кадра стека
86.         pushl   %ecx                    // добавить в стек содержимое %ecx
87.         subl    $20, %esp               // зарезервировать 20 байтов
88.                                         // для локальных переменных
89.         movl    $100000000, -12(%ebp)   // создать count и
90.                                         // инциализировать 100000000
91.         subl    $12, %esp               // отступить 12 байтов от
92.                                         // локальных переменных
93.         pushl   -12(%ebp)               // добавить в стек count, как аргумент
94.                                         // для вызова MonteCarloAlgorithm
95.         call    MonteCarloAlgorithm     // вызов MonteCarloAlgorithm
96.         addl    $16, %esp               // вернуть 16 байт, использованные
97.                                         // при вызове MonteCarloAlgorithm
98.         fstpl   -24(%ebp)               // извлечь вершину st(0) в стек
99.                                         // (результат MonteCarloAlgorithm)
100.        subl    $4, %esp                // отступить 4 байтов от
101.                                        // локальных переменных
102.        pushl   -20(%ebp)               //
103.        pushl   -24(%ebp)               // добавить в стек pi, как второй
104.                                        // аргумент для вызова printf
105.        pushl   $.LC5                   // добавить в стек строку из метки .LC5,
106.                                        // как первый аргумент для вызова printf
107.        call    printf                  // вызов printf
108.        addl    $16, %esp               // вернуть 16 байт, использованные
109.                                        // при вызове printf
110.        movl    $0, %eax                // копировать в eax 0 (код завершения
111.                                        // функции main)
112.        movl    -4(%ebp), %ecx          // копировать в ecx -4(%ebp)
113.        leave                           // сбросить кадр стека
114.        leal    -4(%ecx), %esp          // вычислить эффективный адрес -4(%ecx)
115.                                        // и поместить в esp
116.        ret                             // выход из подпрограммы main
117. .LC1:
118.        .long   -4194304                // RAND_MAX
119.        .long   1105199103
120. .LC3:
121.        .long   0                       // 4.0
122.        .long   1074790400
```

Ассемблерный листинг для архитектуры x86 и уровнем оптимизации O3

```
1.  MonteCarloAlgorithm:
2.          pushl   %esi                    // добавить в стек содержимое esi
3.          pushl   %ebx                    // добавить в стек содержимое ebx
```

```
4.          subl    $48, %esp               // зарезервировать 48 байт для
5.                                          // локальных переменных
6.          movl    60(%esp), %esi          // копировать в esi 100000000
7.          pushl   $0                      // добавить в стек 0 (NULL)
8.          call    time                    // вызов time
9.          movl    %eax, (%esp)            // копировать eax (результат time(NULL))
10.                                         // в вершину стека
11.         call    srand                   // вызов srand
12.         addl    $16, %esp               // сместить esp, чтобы исключить
13.                                         // значение eax и NULL
14.         testl   %esi, %esi              // логическое сравнение esi и esi
15.                                         // (устанавлвает флаг ZF, если esi == 0)
16.         jle     .L6                     // переход в метку .L6, если esi == 0
17.         fldz                            // загрузить 0 в стек сопроцессора
18.                                         // (создание insideCount)
19.         xorl    %ebx, %ebx              // обнулить ebx (создание i)
20.         fstpl   24(%esp)                // извлечь st(0) в стек (обнулить
21.                                         // insideCount)
22. .L5:
23.         call    rand                    // вызов rand
24.         movl    %eax, 12(%ebp)          // копировать eax(результат rand()) в стек
25.         fildl   12(%esp)                // загрузить результат rand() в стек
26.                                         // сопроцессора
27.         fdivl   .LC1                    // разделить st(0) на константу из метки
28.                                         // .LC1 (rand() / RAND_MAX)
29.         fstpl   16(%esp)                // извлечь st(0) в стек (double x)
30.         call    rand                    // вызов rand
31.         movl    %eax, 12(%ebp)          // копировать eax(результат rand()) в стек
32.         fildl   12(%esp)                // загрузить результат rand() в стек
33.                                         // сопроцессора
34.         fdivl   .LC1                    // разделить st(0) на константу из метки
35.                                         // .LC1 (rand() / RAND_MAX)
36.         fldl    16(%esp)                // загрузить x в стек сопроцессора
37.         fmul    %st(0), %st             // умножить x на x
38.         fxch    %st(1)                  // поменять местами st(0) и st(1)
39.         fmul    %st(0), %st             // умножить y на y
40.         faddp   %st, %st(1)             // сложить x*x и y*y и вытолкнуть st(0)
41.         fld1                            // загрузить 1 в стек сопроцессора
42.         fcomip  %st(1), %st             // сравнить st(1) и st(0) (x*x+y*y и 1.0)
43.         jb      .L3                     // переход в метку .L3, если (x*x+y*y)<1.0
44.         fldl    24(%esp)                // загрузить insideCount в стек
45.                                         // сопроцессора
46.         fadds   .LC3                    // прибавить к st(0) (insideCount)
47.                                         // константу из метки .LC3 (4.0)
48.         fstpl   24(%esp)                // извлечь st(0) в стек (присвоить
49.                                         // результат сложения insideCount)
50. .L3:
51.         addl    $1, %ebx                // прибавить 1 к i
52.         cmpl    %ebx, %esi              // сравнить i и count
53.         jne     .L5                     // переход в метку .L5, если i != count
54. .L2:
55.         movl    %esi, 12(%esp)          // копировать count в стек
56.         fildl   12(%esp)                // загрузить count в стек сопроцессора
57.         fdivrl  24(%esp)                // разделить insideCount на count
58.         addl    $36, %esp               // сместить esp, чтобы исключить
59.                                         // локальные переменные
60.         popl    %ebx
61.         popl    %esi
62.         ret                             // выход из подпрограммы
63.                                         // MonteCarloAlgorithm
64. .L6:
65.         fldz                            // загрузить 0 в стек сопроцессора
66.         fstpl   24(%esp)                // извлечь st(0) в стек (обнулить
67.                                         // insideCount)
68.         jmp     .L2                     // безусловный переход в метку .L2
69. .LC5:
70.         .string "PI: %lf\n"
```

```
main:
        leal    4(%esp), %ecx           // вычислить эффективный адрес 4(%esp)
                                        // и поместить в ecx
        andl    $-16, %esp              // логическое и -16 & %esp
        pushl   -4(%ecx)                // добавить в стек содержимое -4(%ecx)
        pushl   %ebp                    // добавить адрес возврата в стек
        movl    %esp, %ebp              // запомнить адрес текущего кадра стека
        pushl   %ecx                    // добавить в стек содержимое %ecx
        subl    $16, %esp               // зарезервировать 16 байтов для
                                        // локальных переменных
        pushl   $100000000              // добавить в стек значение 100000000
        call    MonteCarloAlgorithm     // вызов MonteCarloAlgorithm
        movl    $.LC5, (%esp)           // копировать в стек строку из метки .LC5,
                                        // как первый аргумент для вызова printf
        fstpl   4(%esp)                 // извлечь вершину st(0) (результат
                                        // MonteCarloAlgorithm) в стек до вершины,
                                        // как второй аргумент для вызова printf
        call    printf                  // вызов printf
        movl    -4(%ebp), %ecx          // копировать в ecx -4(%ebp)
        addl    $16, %esp               // вернуть 16 байт, использованные при
                                        // вызове printf
        xorl    %eax, %eax              // обнулить eax (код завершения main)
        leave                           // сбросить кадр стека
        leal    -4(%ecx), %esp          // вычислить эффективный адрес -4(%ecx)
                                        // и поместить в esp
        ret                             // выход из подпрограммы main
.LC1:
        .long   -4194304                // RAND_MAX
        .long   1105199103
.LC3:
        .long   1082130432              // 4.0
```

Ассемблерный листинг для архитектуры x86-64 и уровнем оптимизации O0

```
MonteCarloAlgorithm:
        pushq   %rbp                    // добавить адрес возврата в стек
        movq    %rsp, %rbp              // запомнить адрес текущего кадра стека
        subq    $48, %rsp               // зарезервировать 48 байт для
                                        // локальных переменных
        movl    %rdi, -36(%rbp)         // добавить значение rdi (count) в стек
        movl    $0, %edi                // копировать 0 в edi (создание NULL)
        call    time                    // вызвать time
        movl    %eax, %edi              // копировать eax (результат
                                        // time(NULL)) в edi
        call    srand                   // вызвать srand
        pxor    %xmm0, %xmm0            // исключающее ИЛИ между xmm0 и xmm0
                                        // (обнуление xmm0)
        movsd   %xmm0, -8(%rbp)         // добавить значение xmm0 в стек
                                        // (создание double insideCount)
        movl    $0, -12(%rbp)           // создать i и присвоить 0
        jmp     .L2                     // безусловный переход в метку .L2
.L5:
        call    rand                    // вызов rand
        pxor    %xmm0, %xmm0            // обнулить xmm0
        cvtsi2sdl       %eax, %xmm0     // копировать eax (результат rand())
                                        // в xmm0
        movsd   .LC1(%rip), %xmm1       // копировать константу из метки .LC1
                                        // (RAND_MAX) в xmm1
        divsd   %xmm1, %xmm0            // разделить rand() на RAND_MAX
        movsd   %xmm0, -24(%rbp)        // копировать rand() / RAND_MAX в
                                        // стек (создание double x)
        call    rand                    // вызов rand
        pxor    %xmm0, %xmm0            // обнулить xmm0
        cvtsi2sdl       %eax, %xmm0     // копировать eax (результат rand())
                                        // в xmm0
        movsd   .LC1(%rip), %xmm1       // копировать константу из метки .LC1
```

```
                                        // (RAND_MAX) в xmm1
        divsd   %xmm1, %xmm0            // разделить rand() на RAND_MAX
        movsd   %xmm0, -32(%rbp)        // копировать rand() / RAND_MAX в стек
                                        // (создание double y)
        movsd   -24(%rbp), %xmm0        // копировать x в xmm0
        movapd  %xmm0, %xmm1            // дублировать x
        mulsd   %xmm0, %xmm1            // умножить x на x
        movsd   -32(%rbp), %xmm0        // копировать y в xmm0
        mulsd   %xmm0, %xmm0            // умножить y на y
        addsd   %xmm0, %xmm1            // сложить x*x и y*y
        movsd   .LC2(%rip), %xmm0       // копировать 1.0 в xmm0
        comisd  %xmm1, %xmm0            // сравнить x*x+y*y и 1.0
        jb      .L3                     // переход в метку .L3,
                                        // если x*x+y*y < 1.0
        movsd   -8(%rbp), %xmm1         // копировать insideCount в xmm1
        movsd   .LC3(%rip), %xmm0       // копировать константу из метки .LC3
                                        // (4.0) в xmm0
        addsd   %xmm1, %xmm0            // прибавить к 4.0 insideCount
        movsd   %xmm0, -8(%rbp)         // добавить xmm0 в стек (присвоить
                                        // результат сложения insideCount)
.L3:
        addl    $1, -12(%rbp)           // прибавить 1 к i (++i)
.L2:
        movl    -12(%rbp), %eax         // копировать i в eax
        cmpl    -36(%rbp), %eax         // сравнить i и count
        jl      .L5                     // переход в метку .L5, если i < count
        pxor    %xmm1, %xmm1            // обнулить xmm1
        cvtsi2sdl     -36(%rbp), %xmm1  // копировать count в xmm1
        movsd   -8(%rbp), %xmm0         // копировать insideCount в xmm0
        divsd   %xmm1, %xmm0            // разделить insideCount на count
        movq    %xmm0, %rax             // копировать xmm0 в rax
        movq    %rax, %xmm0             // копировать rax в xmm0
        leave                           // очистить кадр стека
        ret                             // выход из подпрограммы
                                        // MonteCarloAlgorithm
.LC4:
        .string "PI: %lf\n"
main:
        pushq   %rbp                    // добавить адрес возврата в стек
        movq    %rsp, %rbp              // запомнить адрес текущего кадра стека
        subq    $16, %rsp               // зарезервировать 16 байтов для
                                        // локальных переменных
        movl    $100000000, -4(%rbp)    // создать count и
                                        // инциализировать 100000000
        movl    -4(%rbp), %eax          // копировать count в eax
        movl    %eax, %edi              // копировать eax в edi (добавление
                                        // аргумента MonteCarloAlgorithm)
        call    MonteCarloAlgorithm     // вызов MonteCarloAlgorithm
        movq    %xmm0, %rax             // копироваеть xmm0 (результат
                                        // MonteCarloAlgorithm) в rax
        movq    %rax, -16(%rbp)         // копировать rax в стек
        movq    -16(%rbp), %rax         // копировать значение из стека в rax
        movq    %rax, %xmm0             // копироваеть rax в xmm0 (второй
                                        // аргумент printf)
        movl    $.LC4, %edi             // копировать константу из метки .LC4
                                        // в edi (первый аргумент printf)
        call    printf                  // вызов printf
        movl    $0, %eax                // копировать в eax 0 (код завершения
                                        // функции main)
        leave                           // очистить кадр стека
        ret                             // выход из подпрограммы main
.LC1:
        .long   -4194304                // RAND_MAX
        .long   1105199103
.LC2:
        .long   0                       // 1.0
        .long   1072693248
```

```
100.  .LC3:
              .long   0                       // 4.0
```

Ассемблерный листинг для архитектуры x86-64 и уровнем оптимизации O3

```
1.    MonteCarloAlgorithm:
2.            pushq   %r14                    // добавить содержимое регистра r14
3.                                            // в стек
4.            pushq   %rbp                    // добавить адрес возврата в стек
5.            movq    %rdi, %rbp              // копировать 100000000 в rbp
6.            xorl    %edi, %edi              // обнуление ebx (создание NULL)
7.            pushq   %rbx                    // добавить содержимое регистра rbx
8.                                            // в стек
9.            subq    $16, %rsp               // зарезервировать 16 байтов для
10.                                           // локальных переменных
11.           call    time                    // вызвать time
12.           movl    %eax, %edi              // копировать eax (результат time(NULL))
13.                                           // в edi
14.           call    srand                   // вызвать srand
15.           testq   %rbp, %rbp              // логическое сравнение rbp и rbp
16.                                           // (сравнение rbp с 0))
17.           jle     .L6                     // переход в метку .L6, если rbp <= rbp
18.           xorl    %ebx, %ebx              // обнуление ebx (создание i)
19.           xorl    %r14d, %r14d            // обнуление r14d - младшая часть r14
20.                                           // (создание insideCount)
21.   .L5:
22.           call    rand                    // вызов rand
23.           pxor    %xmm0, %xmm0            // обнуление xmm0
24.           cvtsi2sdl       %eax, %xmm0     // копировать eax (результат rand())
25.                                           // в xmm0
26.           divsd   .LC1(%rip), %xmm0       // разделить rand() на RAND_MAX
27.           movsd   %xmm0, 8(%rsp)          // копировать rand() / RAND_MAX в стек
28.                                           // (создание double x)
29.           call    rand                    // вызов rand
30.           movsd   8(%rsp), %xmm0          // копировать x в xmm0
31.           pxor    %xmm1, %xmm1            // обнуление xmm1
32.           movsd   .LC2(%rip), %xmm2       // копировать константу из метки .LC2
33.                                           // (1.0) в xmm2
34.           cvtsi2sdl       %eax, %xmm1     // копировать eax (результат rand())
35.                                           // в xmm1
36.           divsd   .LC1(%rip), %xmm1       // разделить rand() на RAND_MAX
37.                                           // (создание double y)
38.           mulsd   %xmm1, %xmm1            // умножить y на y
39.           mulsd   %xmm0, %xmm0            // умножить x на x
40.           addsd   %xmm1, %xmm0            // сложить x*x и y*y
41.           comisd  %xmm0, %xmm2            // сравнить x*x+y*y и 1.0
42.           jb      .L3                     // переход в метку .L3, если
43.                                           // x*x+y*y < 1.0
44.           movsd   .LC3(%rip), %xmm3       // копировать константу из метки .LC3
45.                                           // (4.0) в xmm3
46.           movq    %r14, %xmm4             // копировать insideCount в xmm4
47.           addsd   %xmm4, %xmm3            // прибавить insideCount к 4.0
48.           movq    %xmm3, %r14             // добавить xmm3 в стек (присвоить
49.                                           // результат сложения insideCount)
50.   .L3:
51.           addq    $1, %rbx                // прибавить 1 к i (++i)
52.           cmpq    %rbx, %rbp              // сравнить i и count
53.           jne     .L5                     // переход в метку .L5, если i != count
54.   .L2:
55.           pxor    %xmm1, %xmm1            // обнуление xmm1
56.           addq    $16, %rsp               // сместить esp, чтобы исключить
57.                                           // локальные переменные
58.           movq    %r14, %xmm0             // копировать insideCount в xmm0
59.           cvtsi2sdq       %rbp, %xmm1     // копировать count в xmm1
60.           popq    %rbx                    // извлечь вершину стека в rbx
61.           popq    %rbp                    // извлечь вершину стека в rbp
```

```
62.         popq    %r14                    // извлечь вершину стека в r14
63.         divsd   %xmm1, %xmm0            // разделить insideCount на count
64.         ret                             // выход из подпрограммы
65.                                         // MonteCarloAlgorithm
66. .L6:
67.         xorl    %r14d, %r14d            // обнуление r14d - старшая часть r14
68.                                         // (создание insideCount)
69.         jmp     .L2                     // переход к метке .L2
70. .LC4:
71.         .string "PI: %lf\n"
72. main:
73.         subq    $8, %rsp                // зарезервировать 8 байтов для
74.                                         // локальных переменных
75.         movl    $100000000, %edi        // поместить 100000000 в edi
76.                                         // (аргумент MonteCarloAlgorithm)
77.         call    MonteCarloAlgorithm     // вызов MonteCarloAlgorithm (результат
78.                                         // в xmm0)
79.         movl    $.LC4, %edi             // поместить строку из метки .LC4 в edi
80.                                         // (аргумент printf)
81.         movl    $1, %eax                // копировать 1 в eax
82.         call    printf                  // вызов printf
83.         xorl    %eax, %eax              // обнуление eax (будущий резуьтат main)
84.         addq    $8, %rsp                // сместить esp, чтобы исключить
85.                                         // локальные переменные
86.         ret                             // выход из подпрограммы main
87. .LC1:
88.         .long   -4194304                // RAND_MAX
89.         .long   1105199103
90. .LC2:
91.         .long   0                       // 1.0
92.         .long   1072693248
93. .LC3:
94.         .long   0                       // 4.0
95.         .long   1074790400
```